Timesys University

Track Two

Building an Internet Radio with the TI Sitara AM3517 using LinuxLink

Session 4

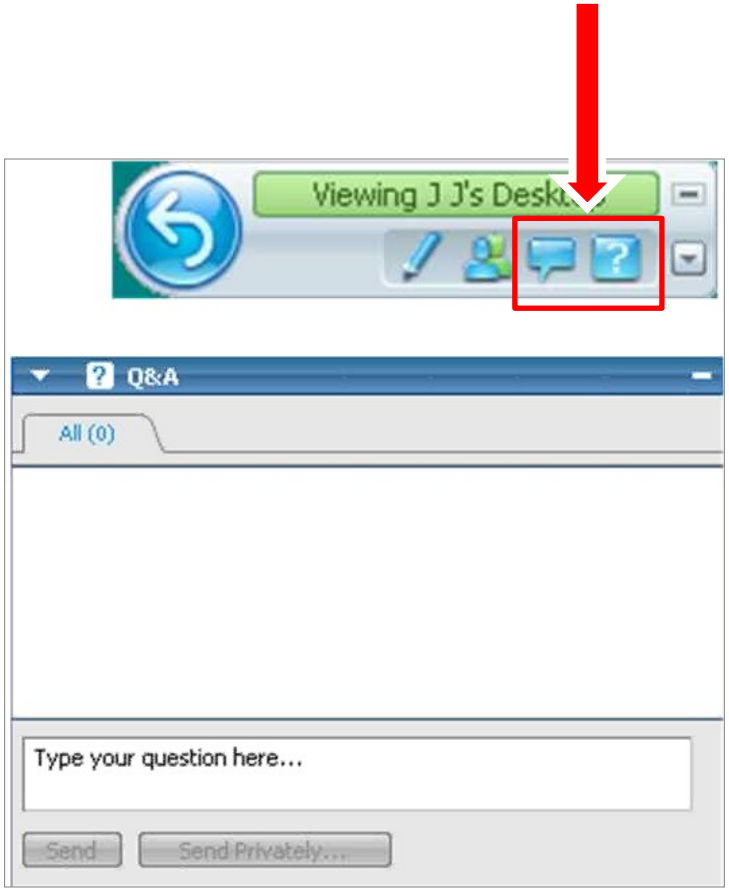
How to optimize, test and integrate the solution for fast booth and quick deployment

Audio streaming is available for this event.
Turn on your speakers to listen.

Tools You Can Use

Q&A and/or Chat

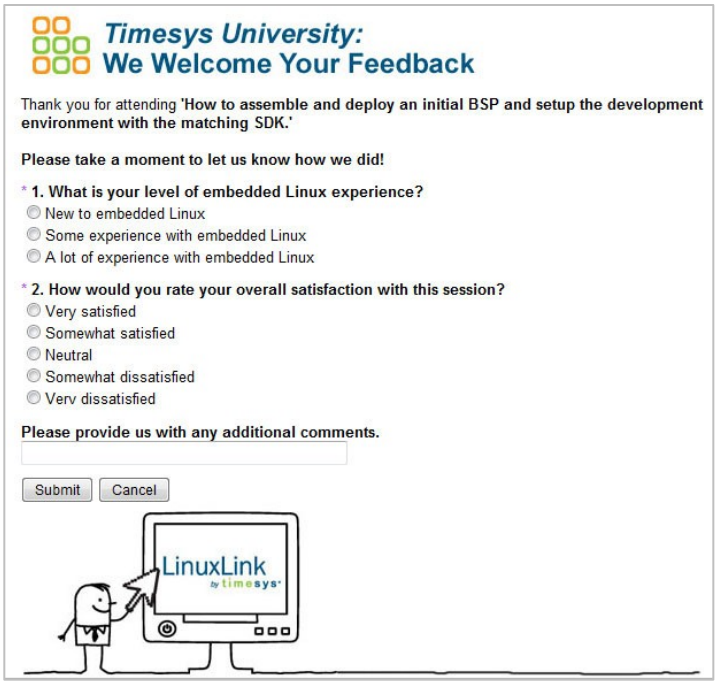
- Click on Q&A panel or chat panel icon in the bottom, right corner
- Type in your question in the space provided
- Click on “Submit”



Tools You Can Use

Polling

- The poll will appear on your screen
- Select your answer for each question
- Click on “Submit”



Session Information

- **You can download the slides for today's session at** http://www.timesys.com/embedded-linux/training/timesys-university/ti_am3517
- **You can view a recording of today's session at** http://www.timesys.com/embedded-linux/training/timesys-university/ti_am3517
- **Today's speaker:**

Maciej Halasz
Director, Product Management
Timesys

Building an Internet Radio with the TI Sitara AM3517

- **Session 1 – Recording Available**
  How to assemble and deploy an initial BSP and setup development environment with the matching SDK
  http://www.timesys.com/embedded-linux/training/timesys-university/ti_am3517
- **Session 2 – Recording Available**
  How to build a modern User Interface to launch Internet Radio playback using Qt Embedded for Linux
- **Session 3 – Recording Available**
  How to decode a media stream and integrate Bluetooth functionality for a remote speaker
- **Session 4 – Today**
  How to optimize, test and integrate the solution for fast boot and quick deployment

Today's Agenda

- **Recap of what we have done so far**
- **Measure the boot time in our dev environment**
- **Integrate all developed software pieces**
  - Overlay concept
- **Fast boot optimizations**
  - Boot process overview
  - Bootloader optimizations
  - Kernel level optimizations
  - Filesystem optimizations
  - Other options
- **Deployment**
  - SD card preparations

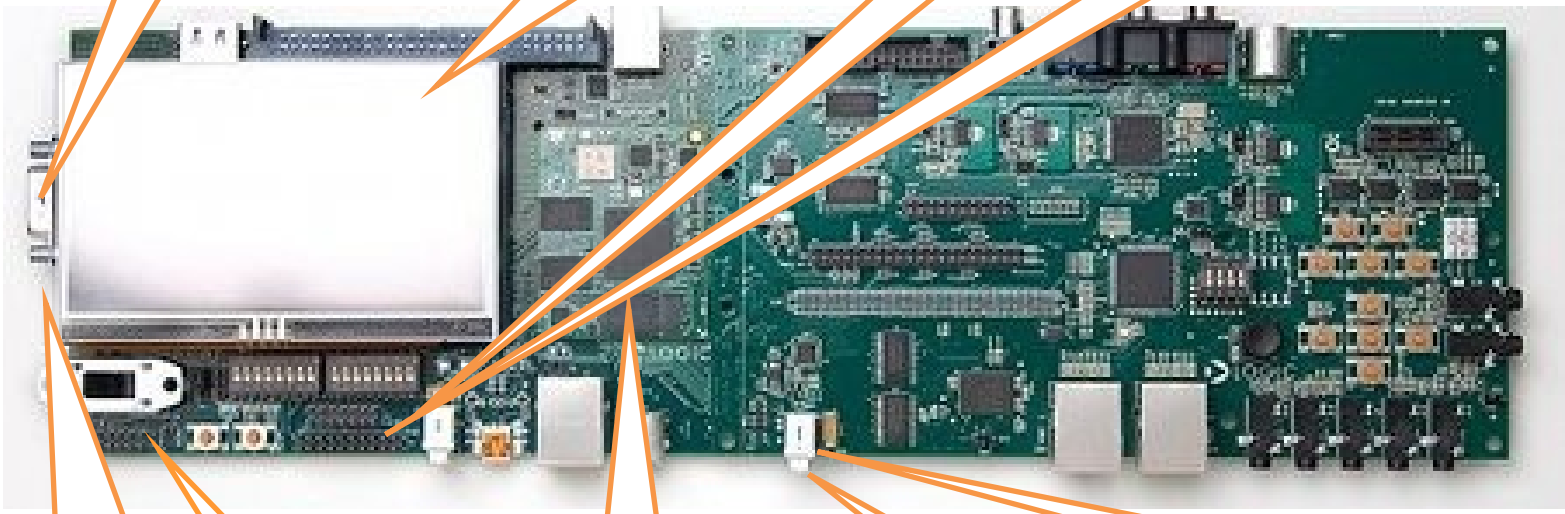
Past Sessions Recap

What We Have Accomplished So Far

- **Learned about TI AM3517 LinuxLink – needed for all exercises**
- **Built a custom BSP with LinuxLink Web Edition**
  - Reflected application requirements
- **Setup a Qt based IDE development environment**
  - Developed a UI with Qt widgets
  - Tested locally
- **Added and configured Bluetooth**
  - Established BT connections using A2DP profile
  - Added code to playback live streams
- **Deployed the system on the target via NFS for future development**

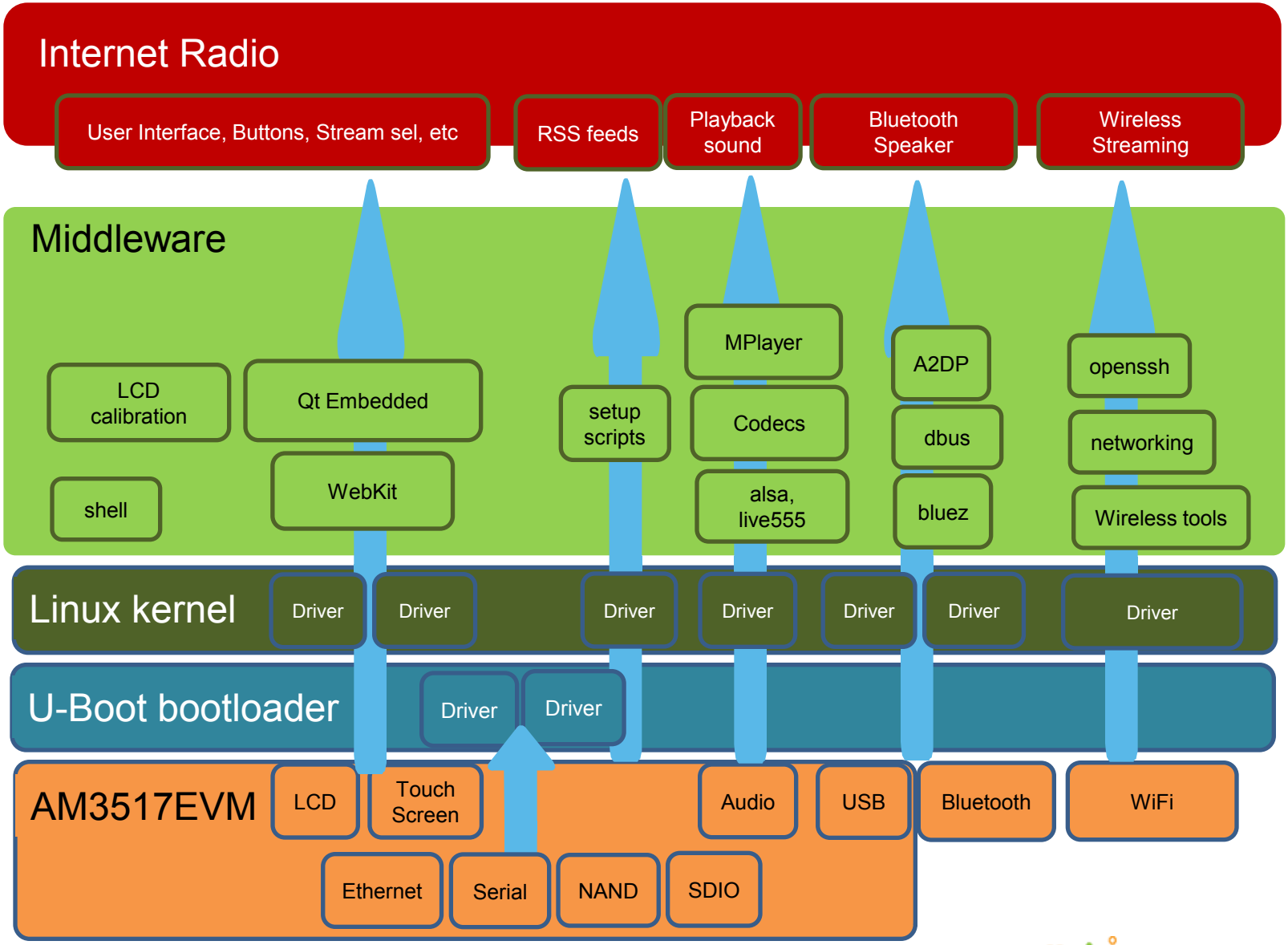
Project Requirements (TI AM3517EVM)

- **Serial port communication**
- **Graphics**
- **Touchscreen**
- **Applications**
  - Screen calibration
  - Playback control UI
- **USB**
  - Storage (USB stick)
  - Extensions
- **Ethernet**
  - Secure Connection
  - Transfer (FTP/SCP)
  - Console (Telnet/SSH)



- **Audio**
  - Alsa Mixer
  - Sound playback
- **SD/MMC Card**
  - Filesystem
  - Booting Linux
- **NAND Flash**
  - Boot from
  - Additional storage
- **WiFi (optional)**
  - Streaming audio
- **Bluetooth**
  - Sensor connections

Internet Radio (Blueprint) – COMPLETED!



Last Week's Homework – Share Your Experience

1. **Setup a generic Bluetooth network**
2. **Pair your system with a Bluetooth device of choice**
3. **Look at available Bluetooth profiles**
4. **Use your Bluetooth device from within your application**

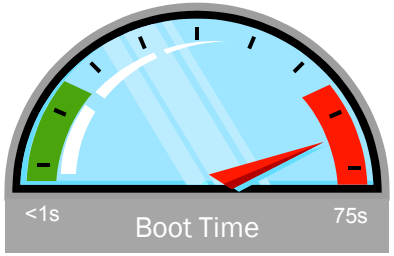
- Did you setup your Bluetooth network?
- What Bluetooth device(s) did you pair with?
- Did you run into any challenges?

Giveaway!!!

- If you attend at least ~~3~~ 2 out of 4 sessions in this Timesys University track, we will automatically enter you into a drawing for a chance to win a Logic PD Zoom AM3517 EVM Development Kit

Software Integration

Developed Components



- **U-Boot**
- **Linux Kernel**
  - Bluetooth, USB drivers,
- **Filesystem**
  - BlueZ, Mplayer, live555, Qt for Embedded, etc.
- **Internet Radio Application**
- **Setup scripts**
  - Bluetooth, DBus, Application startup

Overlay

- **A way to integrate custom Linux components with the underlying Linux system**
  - Structured development
  - Simplified maintenance
- **Create a custom SXX startup script in /etc/init.d/**
  - Export QWS and TSLIB variables
  - Start the internetradio application
- **Create an overlay**
  - Create a set of subfolders
  - Copy custom content to appropriate places
- **Define appropriate entries using Factory's Desktop interface**
  - Using Desktop Factory Interface navigate to:
  - Target Configuration->Build RFS->RFS Content Tarball

  file:///<fully qualified path to overlay's tar file>

Boot time Measurement

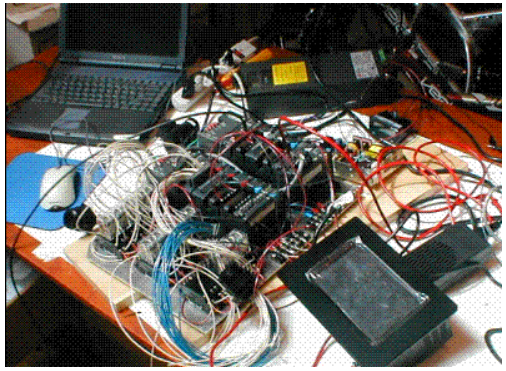
Measuring the boot time

- **Each embedded design is different**
- **Instrumentation is available**
  - Helps measure boot time at all stages
  - Most are open source based
  - Can be easily removed when you are done

The trick behind boot time optimization is to know where your system is spending time.
You may end up loosing a lot of time with little results!!!

Available Instrumentation

- **Bootloader**
  - Logic analyzer (using GPIO pins)
- **Kernel Measurement**
  - Printk
    - Compile kernel with: CONFIG_PRINTK_TIMES=y
    - Switch on dynamically: “echo Y >/sys/module/printk/parameters/time”
  - initcall_debug
    - On boot command line, use: initcall_debug=1
- **User space measurement**
  - Bootchart (www.bootchart.org)
  - Strace
    - Run strace –cT 2>/tmp/strace.log bluetoothd
  - Linux Trace Toolkit
- **System**
  - uptime
    - Add (echo -n "uptime:" ; cat /proc/uptime) to an RC script



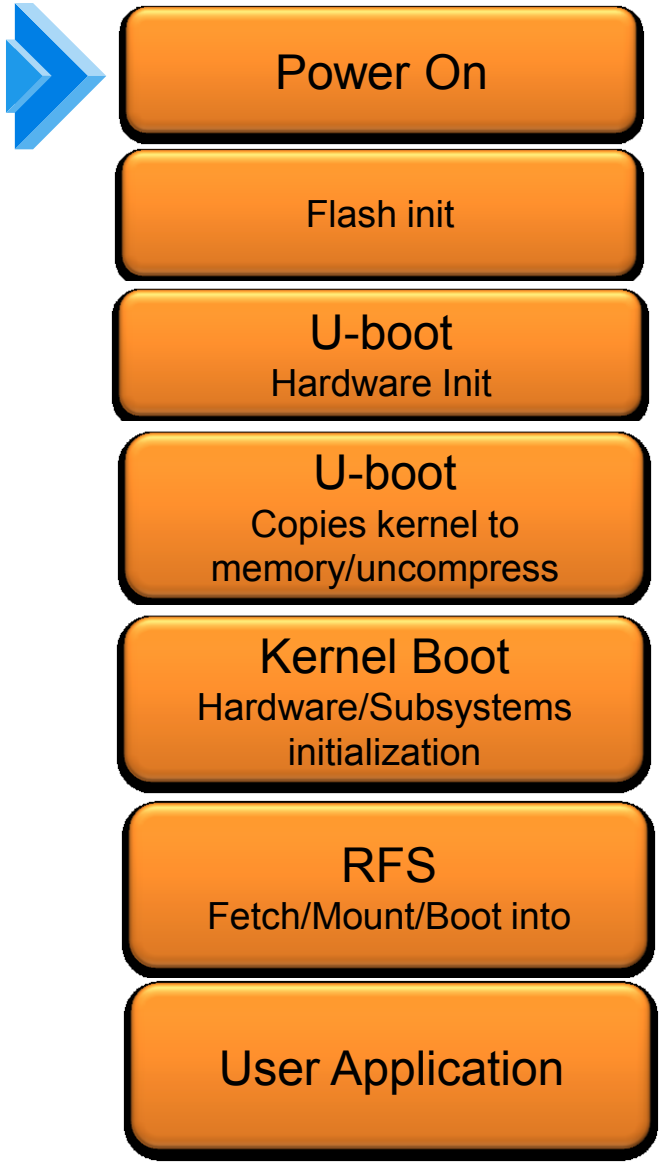
Fast boot Optimizations

Boot Process Overview

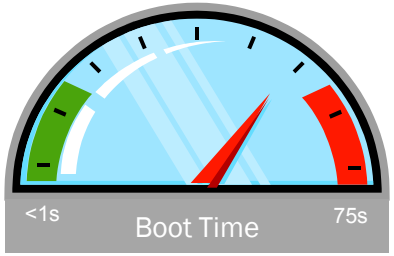
TIME

- **U-Boot**
  - Reset, copy U-Boot to SDRAM and jump to start address
  - Basic System Init. (IMPORTANT)
  - Copy the Linux kernel Image to SDRAM from SD
  - Decompress the kernel if needed
  - Jump to upload address and start the kernel
- **Kernel**
  - Run Kernel Init code
  - Init kernel subsystems (device drivers)
  - Init SD card
  - Mount SD card partition with RFS
  - Execute init script
- **Application**
  - Depends on your specific requirements

Boot Process Sequence

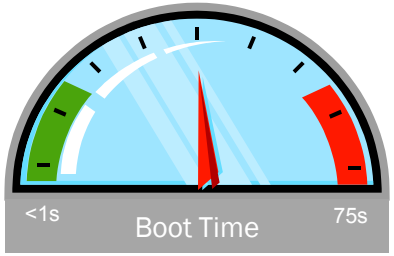


Bootloader Optimizations



- **Low hanging fruit**
  - Set the bootdelay variable to 0 (time savings 4s)
  - Preset the bootcmd; do not use setenv (time savings 0.5s)
  - Disable console (time savings 2s)
    - CFG_CONSOLE_INFO_QUIET
    - CONFIG_SILENT_CONSOLE
    - In our case- silent=yes
  - Disable other tests (time savings 2-6s)
- **Additional modification/enhancements**
  - If possible, use uncompressed Linux kernel
  - Optimize the NAND read operation, to shorten image copy time
  - Rewrite/disable CRC32 checksum code
  - Load the image directly to Entry point
    - Set CONFIG_LOADADDR
  - If NOR Flash is used, leverage XIP
  - For large kernel image, use different compression algorithms

Linux Kernel Optimizations



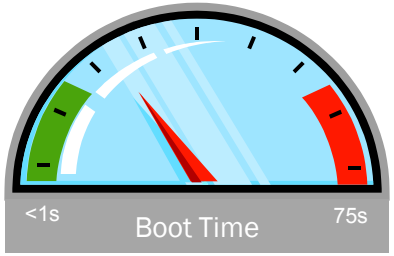
- **Low hanging fruit (time savings: 2-15+s)**
  - Use uncompressed kernel
    - Uncompressing takes time
  - Remove unused kernel options
    - Not used networking i.e. IPV6, multiple file systems
    - Debug features and symbols (for final deployment)
  - Build not initially used device drivers as Loadable Kernel Modules
    - Keep the features needed at boot time built into the kernel
    - Remaining drivers built as LKMs will make kernel smaller
  - Consider various approaches for your RFS deployment
    - JFFS2 with appended journal summary (skip flash scan)
    - CRAMFS, UBIFS
  - Suppress the console output
    - Use “quiet” with your kernel command line

Making Linux Kernel Small and Fast

Linux kernel options we will look at today

Kernel Option	Comment
CONFIG_EMBEDDED	Disables or tweaks a number of kernel options and settings. Think uClinux
CONFIG_IKCONFIG	Saves complete kernel configuration in the kernel
CONFIG_KALLSYMS	Prints our symbolic crash information and backtraces
CONFIG_BUG	Disables BUG and WARN functions
CONFIG_HOTPLUG	Can be disabled if no external devices will be attached and if you use static device files
CONFIG_DNOTIFY	File change notification to user space
CONFIG_EXT2	Disable if using jffs2 file system
CONFIG_PRINTK	Makes kernel silent when disabled
CONFIG_PRINTK_TIME	A way to track where time is spent at boot time
CONFIG_CC_OPTIMIZE_FOR_SIZE	Will select –Os instead of –O2 resulting in a smaller kernel

Userspace Optimizations



- **Filesystem modifications**
  - Optimize the RC Scripts (remove the ones you don't use)
  - If possible replace RC scripts with a single init
  - Pre-link
  - Link applications statically
  - Avoid udev – create needed device nodes ahead of time
  - Use static IP address
- **Additional modifications/enhancements**
  - Staged booting
  - Code level modifications
  - Add a splash screen

Fast Boot Offering from Timesys

- **If your requirements include fast booting, Timesys can help you save time**
- **Implement added levels of optimizations with open source techniques**
  - Some of the deep-level optimizations techniques, described earlier
- **Use non open source technologies available to Timesys to further speed up the boot-time**
  - Typically needed when many services have to be up and running asap
- **This engineering experience is available as a service**

Deployment: SD Card

SD Card Deployment

Card Preparations

- Use a card of appropriate size
- Format the card as follows:
  - Partition 1: VFAT – size: 6MB
  - Partition 2: ext2 – size: 48MB+
- Copy the images
  - Partition 1: uImage kernel
  - Partition 2: uncompress the file system created with the factory Desktop interface
- Insert the card in the target's SD Card slot
- Change the bootcmd variable to boot from SD Card

  setenv bootcmd "mmc init 0; fatload mmc 0 0x80000000 uImage-AM3517;bootm;"

  setenv bootargs "console=ttyS2,115200n8 root=/dev/mmcblk0p2 rw rootfstype=ext2 rootwait"
- Reset the system

What We Have Accomplished

- **Created an overlay – integrated custom code**
- **Learned about boot time optimizations**
  - Concepts
  - Linux component specific
- **Deployed Linux on SD Card**
- **Measured boot time**

This is it!!!

We hope you enjoyed this Timesys University Track

What We Have Accomplished

What We Have Accomplished

- **Cross-compiled/Deployed/Debugged our Internet Radio application on the Target**
- **Learned about how to add Bluetooth support**
  - User Space
  - Kernel Space
  - Autoboot
- **Added codecs/applications for media playback**
- **Modified Internet radio application**
  - Launch external application
  - Handle Play/Stop actions
  - Adding a new radio station from a browser

New Timesys Subscription Model

- **Best price/performance**
- **No complex contracts or usage terms**
- **Every seat includes live, unmetered support**

LinuxLink	LinuxLink + Pro Services NEW!	Pro Services Only
Starts @ $5495	Starts @ $7495; includes 16-hour block of pro services	Project based, discounts for larger projects
Includes live, expert support, multiple seats discount	Includes live, expert support, additional bundles available	
Perfect for engineering teams : • Looking to build embedded linux products and expertise in-house • Reduce the time & uncertainty associated with open source development • Want expert support when they get stuck	Perfect for engineering teams • Wanting to focus in-house teams on differentiated development • Want to develop in-house linux expertise fast • Want expert support when they get stuck	Perfect for engineering teams : • Having tight deadlines • Needing proven linux expertise – fast • Looking to eliminate development risk

Glossary

LinuxLink (Web Edition) – Web-based version of LinuxLink

LinuxLink (Desktop Edition) – Local version with full customization and third-party tools integration

Workorder – Stores definition of your software – filenames, versions

Bootloader – Runs first, initializes necessary hardware, loads Linux

Linux kernel – Operating system that manages hardware access and other features for higher level software

Device Driver – Code that's part of a Linux kernel, defines how software accesses specific hardware

File System – All files (libraries/utilities/scripts/etc.) combined on a single storage, e.g. NAND flash

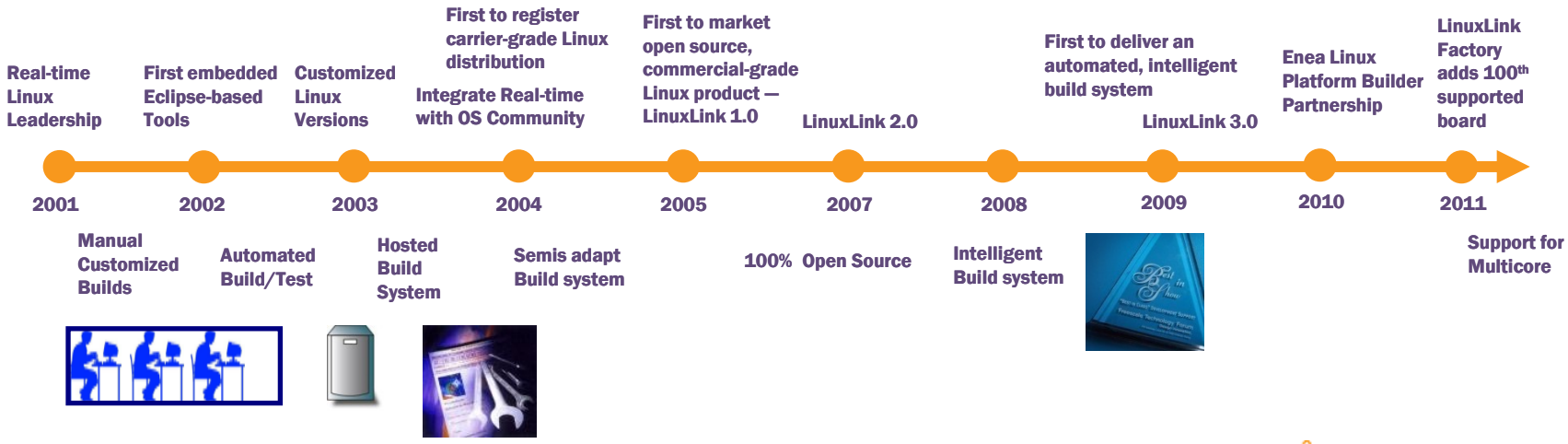
Middleware – Complete frameworks including APIs, utilities that provide specific functionality, e.g. Qt

API (library) – Used by applications, provide functionality, abstract hardware access

Toolchain (cross) – The most important part of the development environment. Used to compile source code into binaries.

About Timesys

- Carnegie Mellon University spin-off in 1996
- First real-time embedded Linux distribution
- First to register carrier grade Linux (CGL)
- First to market with an open source, commercial-grade embedded Linux development framework (LinuxLink)
- First to develop and deliver an award-winning, automated, intelligent, embedded Linux build system (LinuxLink 3.0)



More Info

- **You can download the slides for today's session at** http://www.timesys.com/embedded-linux/training/timesys-university/ti_am3517

- **You can view a recording of today's session at** http://www.timesys.com/embedded-linux/training/timesys-university/ti_am3517

Stay Online for Q&A!