Timesys University

Track One
Building a connected home automation device with the Digi ConnectCore™ Wi-i.MX51 using LinuxLink

Session 4
How to optimize, test and integrate the solution for quick deployment

Audio streaming is available for this event. Turn on your speakers to listen.
Tools You Can Use

- **Q&A**
  - Click on Q&A panel (?) or chat 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/digi_wi-imx51

- You can view a recording of today’s session at http://www.timesys.com/embedded-linux/training/timesys-university/digi_wi-imx51

- Today’s speakers:

  Maciej Halasz
  Director, Product Management
  Timesys
Building a Connected Home Automation Device

- **Session 1 – Recording available**
  How to assemble and deploy an initial BSP and setup development environment with the matching SDK
  

- **Session 2 – Recording available**
  How to build a control application using Qt Embedded for Linux to design a UI experience

- **Session 3 – Recording available**
  How to integrate a WIFI and Bluetooth functionality with the control application

- **Session 4 – today**
  How to optimize, test and integrate the solution for 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 ConnectCore Wi-i.MX51 LinuxLink – needed for all exercises
- Built a custom BSP with LinuxLink Web Edition based on product requirements
- Built a control application with TimeStorm and Qt
  - Added custom Widget
  - Compiled for local host and cross-compiled for the target
- Added GPIO control for two devices — LEDs
  - Used SYSFS
- Added WiFi and Bluetooth connectivity
- Develop code for
  - Remote accelerometers
  - Data upload to an ftp server
Project Requirements (Digi CC Wi-i.MX51)

- **Graphics**
- **Touchscreen**
- **Applications**
  - Screen calibration
  - Home automation application

- **Serial port communication**

- **Audio (optional)**
  - Alsa Mixer
  - Sound playback

- **Ethernet**
  - Secure Connection
  - Transfer (FTP/SCP)
  - Console (Telnet/SSH)

- **USB**
  - Storage (USB stick)
  - Extensions

- **Bluetooth**
  - Sensor connections

- **WiFi**
  - Data upload to a server

- **GPIO controlled LEDs**

- **SD/MMC Card**
  - Filesystem
  - Additional storage

- **NAND Flash**
  - Boot from
  - Additional storage
Home Automation Device (Blueprint)

Connected Home Automation Control Application
- User Interface, Buttons, Lights, etc
- Light Control
- Alarm sound
- Bluetooth Sensor
- Wireless Data Logging

Middleware
- LCD calibration
- Qt Embedded
- setup scripts
- alsa-utils
- alsa
- cwiid
- dbus
- bluez
- openssh
- networking
- Wireless tools

Linux kernel
- Driver
- Driver
- Driver
- Driver
- Driver
- Driver

U-Boot bootloader
- Driver
- Driver

Wi-i.MX51
- LCD
- Touch Screen
- GPIO
- Audio
- USB
- Bluetooth
- WiFi
- Ethernet
- Serial
- NAND
- SDIO
Digi ConnectCore™ Wi-i.MX51 Giveaway Reminder
Giveaway

- If you attend at least 3 out of 4 sessions in our Track 1, we will automatically enter you into a drawing for a prize.

- The results of the drawing of the complete ConnectCore Wi-i.MX51 Development Kit will be announced via e-mail, Twitter and on our site this Tuesday at 4pm EST

- Together with Digi International, we also offer the above mentioned kit at a discount price of $249+S&H

- If interested in buying a kit, send an email to sales@timesys.com with “CC-Wi-i.MX51 Kit SPECIAL OFFER” in subject line.
Boot-time Measurement
Software Integration
Developed Components

- **U-Boot**
- **Linux Kernel**
  - Modules for WiFi and Bluetooth
- **Filesystem**
  - BlueZ, WiFi utilities, Qt for Embedded, etc.
- **Home Automation Application**
- **Setup scripts**
  - Wifi, GPIO, Application startup
Overlay

- Way to integrate custom Linux components with the underlying Linux system
  - Structured development
  - Simplified maintenance

- Create an overlay
  - Create a set of subfolders
  - Copy custom content to appropriate places

- Define appropriate entries using Factory’s Desktop interface
  file:///<fully qualified path to overlay’s tar file>
Fast-boot Optimizations
Boot Process Overview

- **U-Boot**
  - Reset, copy U-Boot to SDRAM and jump to start address
  - Basic System Init. (IMPORTANT)
  - Copy 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
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 all possible 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

- **Additional modifications/enhancements**
  - Staged booting
  - Code level modifications
  - Add a splash screen
# Making Linux Kernel Small and Fast

## Linux kernel options we will look at today

<table>
<thead>
<tr>
<th>Kernel Option</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIG_EMBEDDED</td>
<td>Disables or tweaks a number of kernel options and settings. Think uClinux</td>
</tr>
<tr>
<td>CONFIG_IKCONFIG</td>
<td>Saves complete kernel configuration in the kernel</td>
</tr>
<tr>
<td>CONFIG_KALLSYMS</td>
<td>Prints our symbolic crash information and backtraces</td>
</tr>
<tr>
<td>CONFIG_BUG</td>
<td>Disables BUG and WARN functions</td>
</tr>
<tr>
<td>CONFIG_HOTPLUG</td>
<td>Can be disabled if no external devices will be attached and if you use static device files</td>
</tr>
<tr>
<td>CONFIG_DNOTIFY</td>
<td>File change notification to user space</td>
</tr>
<tr>
<td>CONFIG_EXT2</td>
<td>Disable if using jffs2 file system</td>
</tr>
<tr>
<td>CONFIG_PRINTK</td>
<td>Makes kernel silent when disabled</td>
</tr>
<tr>
<td>CONFIG_PRINTK_TIME</td>
<td>A way to track where time is spent at boot time</td>
</tr>
<tr>
<td>CONFIG_CC_OPTIMIZE_FOR_SIZE</td>
<td>Will select –Os instead of –O2 resulting in a smaller kernel</td>
</tr>
</tbody>
</table>
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 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: 16MB
  - 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-info; fatload mmc 1 uImage 90800000; bootm 90800000'
  ```
- 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!!!

Hope you enjoyed this Timesys University Track
Last Week’s Homework – Share Your Experience

1. **Setup a generic Bluetooth network**
2. **Pair your system with a BT device of choice**
3. **Look at available Bluetooth profiles**
4. **Setup a wifi network using the reference board you work with**

- Did you setup your Bluetooth network?
- What Bluetooth device(s) did you pair with?
- What WiFi chip/driver did you use in your wireless network setups?
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.
More Info

- You can download the slides for today’s session at http://www.timesys.com/embedded-linux/training/timesys-university/digi_wi-imx51

- You can view a recording of today’s session at http://www.timesys.com/embedded-linux/training/timesys-university/digi_wi-imx51

Stay Online for Q&A!

THANK YOU!