SITARA™ ARM® PROCESSORS

Boot camp

TEXAS INSTRUMENTS

Optimizing Linux Boot Time

This session gives an overview of methods for optimizing the boot time of a Linux system

LAB: http://processors.wiki.ti.com/index.php/Sitara_Linux_Training

July 2012

Creative Commons Attribution-ShareAlike 3.0 (CC BY-SA 3.0)

creative commons

You are free:

to Share – to copy, distribute and transmit the work

to Remix – to adapt the work

to make commercial use of the work

Under the following conditions:

Attribution – You must give the original author(s) credit

Share Alike - If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

With the understanding that:

Waiver — Any of the above conditions can be waived if you get permission from the copyright holder.

Public Domain — Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.

Other Rights — In no way are any of the following rights affected by the license:

Notice — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

CC BY-SA 3.0 License:
http://creativecommons.org/licenses/by-sa/3.0/us/legalcode

CC BY-SA

Texas Instruments

Pre-work Check List

- ❑ Installed and configured VMWare Player v4 or later
- ❑ Installed Ubuntu 10.04
- ❑ Installed the latest Sitara Linux SDK and CCSv5
- ❑ Within the Sitara Linux SDK, ran the setup.sh (to install required host packages)
- ❑ Using a Sitara EVM, followed the QSG to connect ethernet, serial cables, SD card and 5V power
- ❑ Booted the EVM and noticed the Matrix GUI application launcher on the LCD
- ❑ Pulled the ipaddr of your EVM and ran remote Matrix using a web browser
- ❑ Brought the USB to Serial cable you confirmed on your setup (preferable)

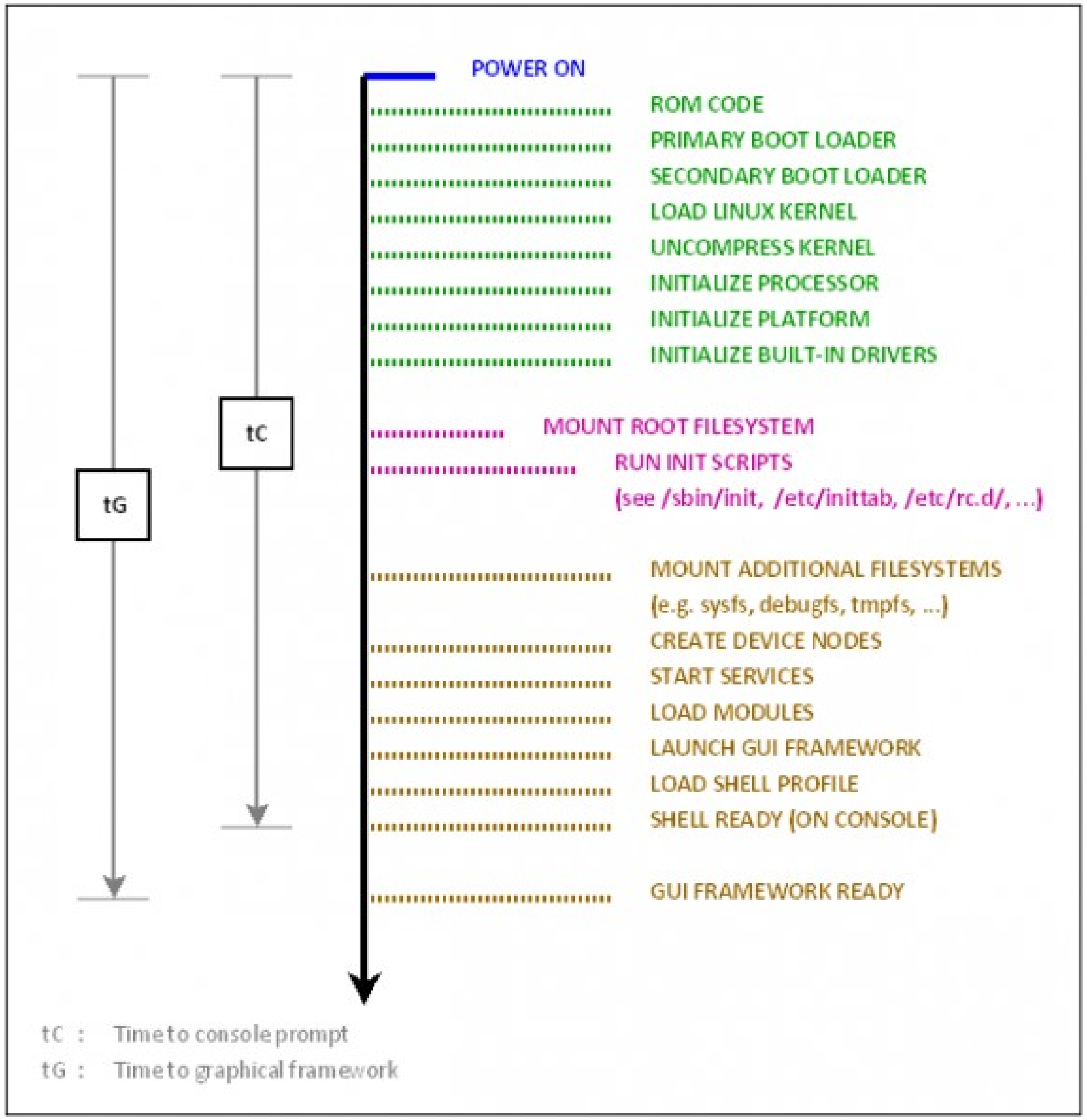
Agenda

- What is Fast Boot?
- Boot Process Overview
- Measuring Boot Time
- Identifying the Boot Steps
- Profiling
  - U-boot
  - Linux Kernel
- Optimization Techniques
  - U-boot
  - Linux Kernel
  - File System

What is Fast Boot?

- Fast boot refers to minimizing the boot time of a system. The boot time of the system is the time it takes from the application of power to the system becoming “available”
- “Available” has a lot of different meanings depending on the user expectations.
  - Appearance of the home screen for devices like cell phones
  - An audible tone or LED indicator
  - A Linux prompt
  - Becoming discoverable on the network
  - Having a key peripheral become available
- The above variances show why the Sitara Linux SDK is generally not fully optimized for boot time. Each user can have a different target and a generic SDK cannot satisfy all user targets
  - There are also many features that make for a good development environment, but which do not make for a fast booting environment

Boot Process Overview



Measuring Boot Time

- There are a couple of different options for measuring boot time. You can use a C program that time stamps each line on the serial port.
  - http://processors.wiki.ti.com/index.php/Measuring_Boot_Time
  - Compile with “gcc tstamp.c -o tstamp”
  - This can also be invoked as “cat /dev/<TTY DEVICE> | tstamp

- RealTerm for Windows supports time stamping the incoming serial data
  - http://realterm.sourceforge.net/
  - Support logging in Unix date format
  - Ability to log directly to a file
  - Ability to stop logging after a predefined time
  - No need to switch terminal for interactive session

- TeraTerm for Windows can now log time as well but does not give elapsed time measurements.

Measuring Boot Time - Cont

- Using the C program approach you will see output like:

```
18.066    0.093: Thu Jun 28 11:56:00 UTC 2012
INIT: Entering runlevel: 5
18.262    0.175: Starting system message bus: dbus.
18.280    0.018: Starting Hardware abstraction layer hald
19.474    1.194: Starting Dropbear SSH server: dropbear.
19.499    0.025: Starting telnet daemon.
19.538    0.039: Starting network benchmark server: netserver.
19.569    0.031: Starting syslogd/klogd: done
19.629    0.060: Starting thttpd.
19.731    0.102: Starting PVR
20.158    0.427: Starting Lighttpd Web Server: lighttpd.
20.176    0.018: 2012-06-28 11:56:02: (log.c.166) server started
```

- In the above output you can see that starting the Dropbear SSH server takes 1.194 seconds.
  - If you do not need SSH you can save 1.194 seconds by disabling Dropbear
  - Disabling Dropbear is as simple as removing the S10dropbear init script from the /etc/rc5.d directory

Identifying the Boot Steps

- One of the first things to help in optimizing boot time is to be able to recognize the markers indicating where a new part of the boot process starts
- SPL
  - The first newline character received on the serial console marks the start of SPL
- U-boot
  - The banner containing the U-boot version indicates the start of u-boot
    U-Boot 2011.09 (Jun 28 2012 - 11:20:36)
- Linux Kernel
  - First line after the below line indicates the start of the Linux kernel
    Uncompressing Linux… done, booting the kernel
- File System
  - The below line indicates the transition to the file system Init process
    INIT: version 2.86 booting

Profiling the Boot Loaders

- There is no direct profiling support in the boot loaders, but the serial print times can still be very useful
- For example, when booting the kernel image you will see output like:

```
5.335   0.010: ## Booting kernel from Legacy Image at 80007fc0 ...
5.339   0.004:    Image Name:   Arago/3.2.0-psp04.06.00.08.sdk/a
5.344   0.005:    Image Type:   ARM Linux Kernel Image (uncompressed)
5.348   0.004:    Data Size:    3164688 Bytes = 3 MiB
5.350   0.002:    Load Address: 80008000
5.352   0.002:    Entry Point:  80008000
6.273   0.921:    Verifying Checksum ... OK
6.276   0.003:    XIP Kernel Image ... OK
6.310   0.034: OK
```

- Notice that almost a second is spent verifying the kernel image checksum.
- If your system doesn't do anything about a bad image, why spend time verifying it?
  - This verification can be disabled by setting the "verify" u-boot parameter to n at the u-boot command prompt

    u-boot# setenv verify n

Profiling the Linux Kernel

- One of the simplest ways to profile the Linux kernel is to configure "CONFIG_PRINTK_TIME" for the Linux kernel. On most recent kernels this is enabled by default.
  - This adds the time since the kernel was booted in [ ]'s before each line

```
[    2.029785] mmc1: card claims to support voltages below the defined range. T.
[    2.048583] mmc1: queuing unknown CIS tuple 0x91 (3 bytes)
[    2.055053] mmc1: new SDIO card at address 0001
[    3.986724] PHY: 0:00 - Link is Up - 100/Full
[    4.015808] Sending DHCP requests ., OK
[    4.036254] IP-Config: Got DHCP answer from 0.0.0.0, my address is 128.247.10
[    4.044586] IP-Config: Complete:
[    4.047973]      device=eth0, addr=128.247.105.20, mask=255.255.254.0, gw=12,
[    4.056182]      host=128.247.105.20, domain=am.dhcp.ti.com, nis-domain=(non,
[    4.063781]      bootserver=0.0.0.0, rootserver=0.0.0.0, rootpath=
```

  - In The above output we can see that it takes almost 2 seconds to get the network Phy Link up and obtain a DHCP IP address.
    - Depending on network speed this could be longer. In the case where there is no DHCP server this can take minutes to timeout.
    - Setting "ip=off" in the bootargs will bypass kernel network configuration while still allowing you to configure the network in user space.

      setenv ip_method off

Profiling the Linux Kernel - Cont

- Instrument kernel initialization. This will help you find which static drivers in the kernel are taking the most time to initialize
  - Add “initcall_debug” to the bootargs in u-boot. With recent u-boots this can usually be done using:
    setenv optargs initcall_debug
    saveenv
  - When the Linux system is booted you can view these initcall lines using “dmesg | grep initcall”
    [xxxxx] initcall <init function> [<module>] returned 0 after <time> usecs
  - These lines can be sorted using the commands below to help find the modules with the largest init times
    - If CONFIG_PRINTK_TIME is enabled
      dmesg | grep initcall | sort -k8 -n
    - If CONFIG_PRINTK_TIME is not enabled
      dmesg | grep initcall | sort -k6 -n
  - It is possible to graphically view these initcall times using the bootgraph script in the Linux kernel sources. This requires CONFIG_PRINTK_TIME to be enabled
    cat <bootlog> | perl <kernel source dir>/scripts/bootgraph.pl > boot.svg

- Additionaly you can use other tools to help you analyze the Linux kernel such as:
  - Linux Trace Toolkit (LTTng)
    - Timing for certain kernel and process events
  - Oprofile
    - System wide profiler
  - Bootchart
    - Visualizes boot process

Areas of Optimization

- Optimizations generally fall into two areas
- Size
  - Reduce the size of binaries
  - Remove features not required to reduce component size
- Speed
  - Optimize for target processor
    - Neon optimizations
  - Use faster boot media
    - NOR/NAND vs. MMC/USB
  - Reduce the number tasks leading to boot
    - Do not check MMC if booting from NAND
    - Do not initialize network if booting from MMC
  - Reduce initialization operations
    - Do not bring up network during boot if not required
    - Do not start an SSH server on a device with no network

U-boot Optimization Techniques

- Reduce environment size so that less data is read into memory
  - CONFIG_ENV_SIZE
- Remove Unnecessary Console Print Statements
  - In board config file add
    #define CONFIG_SILENT_CONSOLE 1
  - In u-boot environment do
    setenv silent 1
- Set “bootdelay” to 0
- Disable un-used peripherals such as USB/MMC/Ethernet/UART
- Modify the config file for your device in <u-boot sources>/include/configs. i.e am335x_evm.h
  - Avoid long help text for the u-boot commands to save space
    - #undef CONFIG_SYS_LONGHELP
  - Use simple parser - instead of hush
    - #undef CONFIG_SYS_HUSH_PARSER
  - If no USB/NAND/MMC/SPI/NOR(FLASH)
    - #undef CONFIG_USB_*
    - #undef CONFIG_NAND
    - #undef CONFIG_MMC
    - #undef CONFIG_SPI
    - #undef CONFIG_FLASH_*

U-boot Optimization Techniques - Cont

- Remove -g option from the compiler
- Other Ideas
  - Disable UART boot
  - Remove Image Verification (covered before)
  - Perhaps try uncompressed image
  - Verify that kernel image is read to the proper memory location

Linux Optimization Techniques

- Remove un-necessary drivers/features from kernel configuration
  - Reduces driver initialization time
  - Reduces kernel size
- Build non-fast boot drivers as modules
  - Load them after the system is booted when there is more time
- Disable console output using “quiet”
  - Displaying messages on console takes time
  - Setting the “quiet” option in the bootargs disables display on console but messages are still logged

    u-boot# setenv optargs quiet
  - Remove un-used consoles. These take time to initialize.
    - These can be removed in the /etc/inittab file on the target file system
  - It is possible to completely disable printk but this will eliminate a lot of debug information

Linux Optimization Techniques - Cont

- Defer module init calls
  - It is possibly to defer module init calls without having to build the modules dynamically.
  - This requires modifying the kernel
  - For modules that are not needed at boot replace the “module_init()” function calls to “deferred_module_init()”
  - Once the system booted the deferred calls can be executed by doing:

    echo 1 > /proc/deferred_initcalls
  - Additional details at http://elinux.org/Deferred_Initcalls
- Remove the -g option from the compile
- Disable kernel debugging features
  - Kernel debugging
  - Debug Filesystem (NOTE: Some features may need this)
  - Tracers
- Remove any instrumentation you may have added such as initcall instrumentation.

Linux Optimization Techniques - Cont

- Pre-set loops per jiffy
  - You just need to measure this once
  - Find lpj value in kernel boot messages
    - Calibrating delay loop… 718.02 BogoMIPS (lpj=3590144)
  - Add the “lpj=3590144” to the bootargs
- Use Static IP addressing where possible
  - If you don’t need networking then disable it altogether
  - If you want networking capability but not NFS then be sure to set “ip=off” on the kernel command line.
- Set memory limit with “mem=“ option
  - Use only as much memory as needed to avoid DDR initialization time

File System Optimization Techniques

- Use minimal BusyBox file system
  - Reduces forking in shell
  - Build static of possible to reduce the need for un-used code in the file system
    - Be careful because static linking can also cause your file system size to increase dramatically.
- Avoid using ramdisk or initramfs
  - Must load entire ramdisk from flash into DDR
  - May only need a small part at boot time
  - i.e. may not need all of glibc but entire library will be loaded into DDR using ramdisk or initramfs
  - Buffer cache can keep frequently used files in memory

File System Optimization Techniques - Cont

- Pre-linking
  - Avoid run time linking penalty
  - Drawback: If library changes app must be rebuilt.
- Use tmpfs file system
  - No need to initialize file system for non-persistent data
- Use split file systems
  - Have multiple file system partitions
  - Put only the files needed for boot in the root file system partition
  - Put other files in second file system which can be mounted after boot

File System Optimization Techniques - Cont

- Strip executables
  - Removes un-needed symbols and reduces size
- Avoid udev for static systems
  - If the system doesn't change then make device nodes manually rather than udev creating them
  - Hotplug-daemon can still run later to add additional devices that are plugged in
- Disable init scripts that start unneeded services
- Use GNU_HASH to make dynamic linking faster (This is the default for the SDK)
- Use systemd for parallel initialization

Credits/Sources

- http://processors.wiki.ti.com/index.php/Optimize_Linux_Boot_Time
  - by Sanjeev Premi
- http://elinux.org/Boot_Time

SITARA™ ARM® PROCESSORS
Boot camp

For more Sitara Boot Camp sessions visit:
www.ti.com/sitarabootcamp

THANK YOU!