Linux Bootup Time Reduction for Digital Still Camera

Chanju Park
Samsung Electronics, Co.
bestworld@samsung.com

Kyuhyung Kim
Samsung Electronics, Co.
kyuhyung.kim@samsung.com

Youngjun Jang
Samsung Electronics, Co.
yj03.jang@samsung.com

Kyungju Hyun
Samsung Electronics, Co.
kyungju.hyun@samsung.com

Abstract

Bootup time is a very important issue in the DSC System because customers want to capture immediately specific image. In this paper, we present the experience of the implementation methods and the performance evaluation results for reduction of bootup time which was used in SAMSUNG DSC platform. At first we introduce the DSC platform and development environments and next we explain the optimization method for the bootloader and the kernel bootup time. We also describe root file system, device driver module configuration and DSC applications initialization for bootup time. There are various techniques for linux bootup time reduction and we explain the most important differences.

1 Introduction

Recent digital convergence trends drive CE products to have many functions in single device. The DSC also follows it that provides many functions such as MP3, PMP, and network function. Various real time operating system (RTOS), such as ITRON [1], ZORAN OS [2], VxWorks [3], etc., have been widely used for the operating system of DSC. However, some of RTOSes does not support the extensibility, adaptability, and flexibility [4]. In order to support these properties, we adopt the embedded Linux as an OS in our DSC which kernel features can be modified freely because its source codes are opened.

Linux uses standard device driver interfaces and it supports the POSIX APIs, so it provides diverse extensibility. Using the Linux can considerably reduce the long-term development cost so that the DSC can strengthen the one of the most important competitive powers- "Time to Market". There are many CE products adopting embedded Linux [5]. However, in DSC, only a few companies use it, such as Ricoh Company that made prototype Linux DSC [6].

There would be various important requirements to load embedded Linux in the DSC. Among them, the major requirement would be the real-time performance, and the bootup time. Recently, embedded Linux provides many features for real time and the DSC hardware also does many works for image processing to reduce S/W computation overhead. By H/W and S/W supporting, we can achieve real-time re-

quirement of DSC. However, we still have a important requirement, the bootup time.

In this paper, we will describe "bootup time reduction" which is one of the major requirements in the Linux DSC. This paper organized as follows. In the next session, we describe the boot procedure of the DSC, and then present the technologies relevant to improving bootup time for Linux. The last section presents the results of implementation and future research direction.

2 Bootup time reduction methods

2.1 Environments

We used the Samsung DSC reference platform for implementing Linux DSC. It has a 16/32-bit RISC microprocessor, designed to provide DSC features—6 mega pixel CCD, powerful JPEG encoder/decoder, Divx decoder, audio DSP, 64MB DDR memory controller, Camera interface, SD Host & Multi-Media Card interface, etc. Especially, it includes an OneNAND flash memory and no NOR flash memory.

The initial Samsung FPGA DSC platform is shown in Figure 1.

Figure 1: Initial Samsung FPGA DSC Platform using S5C7380x

S5C7380x is the high spec digital camera platform of Samsung, which integrates many peripherals for fast Image processing.

The major features of the platform are:

- **Core**: Arm926EJS (16K I/D cache)
- **Image processor**: Samsung S5C7380x
- **System clock**: 216Mhz Fclock, 108Mhz Hclock
- **Memory**: 64MB SDRAM , 64MB OneNAND flash
- **DSC Module**: CCD censor, Image Processing Unit, AF/Zoom/Shutter/Iris motor, Digital LCD, JPEG encoder/decoder, OSD, 3A(AE, AWB, AF) module, etc.
- **OS**: Linux kernel 2.4.20
- **Kernel Size**: about 1MB (uncompressed Image)
- **File System**: Cramfs for root file system, Robust File system for OneNAND

2.2 DSC Bootup Procedure

The booting of the DSC system is a process from power-up to ready-to-shoot. (After Ready- to-shoot state, we can capture any images). It consists of three main operations, "Boot loader", "Kernel initialization," and "Application Initialization". After power up, the boot loader initializes the system and starts the system process. Then it copies the kernel image into memory. Once the kernel is loaded, the kernel initializes many resources and loads H/W module into memory, then it mounts several file systems to the several mount points.

We summarize three steps of the bootup procedure as follows:

- Boot Loader
  1. System initialization
  2. Kernel image copy to RAM
- Kernel Initialization
  1. Init kernel
  2. Init device drivers
- Application Initialization
  1. Run RC script
  2. Run Applications
  3. Preview Mode (ready-to-shoot)

2.3 Boot loader

Bootloader is a program that runs just before an OS really starts its work. It initializes a system and loads a kernel image into RAM. If we use NOR flash as a boot device, we can shorten the bootup time using kernel XIP [7].

However, in our work, we should use the OneNAND flash memory instead of using NOR flash for two reasons. Samsung's OneNAND is a single chip flash that offers the ultra-high density of NAND and has the interface to NOR at very attractive price. The OneNAND is based on NAND architecture integrating buffer memory and logic interface. It takes both advantages from high-speed data read function of NOR flash and the advanced data storage function of NAND flash. It is mandatory to make additional small boot loader for copying the kernel image to memory.

After loading the kernel image, bootloader continues loading a RFS(Root File System) image into RAM. Typically the RFS image is stored in compressed form (gz), therefore, it must be loaded from storage and decompressed. But by making RFS on a cramfs File System, it allows fast boot time since only used files are loaded and uncompressed. In addition, we can consider the initializing device drivers. In order to shorten bootup time, the bootloader loads driver concurrently as many as it can.

2.4 File system

Root file system (RFS) is essential element for running kernel on embedded system. There are many file systems, and these can be used as Linux root file system. Each file system has its own functionality, various bootup methods with different bootup time. There are many sub works to do for mounting file system. For example, decompressing the compressed file system, copying itself from storage device to memory, searching the file system contents, searching inodes, journaling, and so on. Therefore, the reduction of RFS mounting time is very important.

To minimize mounting time, we adopt the CRAMFS as root file system. The CRAMFS is designed for simple and small file system, so it has smaller bootup time comparing to other

file systems. The CRAMFS reads only super block among entire file system element while it mounting root file system. We can have relatively short boot time using the CRAMFS. While the CRAMFS has a shorter boot time, it has some demerits. It can be used only with read-only attribute, so it's recommended that use the CRAMFS only on boot area, and use another file system on other area that needs to read and write operation. But if we use special options for cramfs, specific directories would not be compressed, so we can save the mount time.

2.5 Application Optimization Issues

Loading DSC application module is final sequence of bootup procedure In DSC system. After loading applications, bootup sequence is finished and the system becomes ready-to-shoot mode. This section describes about time consuming part while loading and running application on bootup sequence, and time reduction techniques of application.

1. **Init script** – After kernel bootup, kernel executes init program which is located at `/sbin/init`. This program does some tasks according to `/etc/inittab` script. For optimizing bootup time, it's necessary to remove unused service on init script and to run only necessary applications. As we mentioned on before, this init script and applications are included in root file system, CRAMFS. CRAMFS has an option which does not compress some area. By using this option, we can reduce bootup time.

2. **Resource loading time** – After initializing kernel and device drivers, system enters into preview mode, and waits for user input. So user can capture image whenever user wants after DSC init. On preview mode, system displays some information about system information, storage information, image quality, date, etc.

   This information is represented as icon, font, menu images on the LCD display unit. Because the OSD hardware unit in DSC use these resources, we call it as OSD data. All OSD data must be loaded from permanent storage media to memory. But it is time wasting job to load all OSD data during bootup time. We can reduce the loading time by selective loading only necessary resources for booting. If we need more OSD data, we can load them later dynamically.

3. **Lazy process creation** – During DSC system operates, many subtasks—like event processing, resource management, image processing, power management—are executed. Many processes are invoked for executing these tasks. When creating a process, system runs system call named fork(). But invoking fork() wastes time about tens of ms. It is an overhead when used on bootup time. So it is recommended to create processes when they are needed, not to create them when booting the system.

2.6 Other Optimization Methods

Until now, many DSC specific methods for reducing bootup time are introduced. In this section, we introduce some methods, what we adopted to our DSC system, outperforming in terms of boot time reduction.

1. **Preset loops_per_jiffy** – One of well known method of kernel bootup time reduction is 'preset_LPJ'. At each boot time,

the Linux kernel calibrates a delay loop for estimating system performance. This measures a `loops_per_jiffy` (LPJ) value in `calibrate_delay()`. By using a pre-calculated LPJ value, we can reduce loop overhead, and save bootup time.

- **Improvement**: about 250ms

2. **Disable Console Output** – The output of kernel bootup messages to the console takes time, but console output is not needed on a production system. So we can remove bootup messages by using 'quiet' argument to the kernel command line. For example:

   - **Improvement**: about 230ms

3. **Device Driver Initialization** – To control HW units on DSC system, the kernel device driver are needed. All device drivers have initialize routines, and these are called during kernel bootup time. So optimizing the device driver init routine will reduce kernel bootup time. The device driver can be loaded into kernel as two ways, the static method and the dynamic method. During kernel bootup, only static drivers are loaded, and dynamic drivers are loaded as modules after the bootup sequence. So, if a device driver is not necessary on system init, it will be better using dynamic loading rather than using static loading. By making device driver as module, we can reduce device driver init time while booting. On our DSC system, we made device drivers that are not used on bootup sequence—USB, MPEG, STROBE, WDT, TV, etc.—as modules, and loaded them at runtime.

   - **Improvement**: tens of ms

4. **Concurrent driver init** – DSC system is composed of various HW unit like motor (zoom, focus, iris), image processing unit, JPEG en/decoder, MPEG en/decoder, strobo, LCD, CCD, etc. Some of these units are initialized at bootup time because they are used right after bootup. Normally, these static device drivers are initialized in `do_initcalls()` function while bootup time. But some kinds of devices need long initializiation time. For example, zoom motor has to moved to some fixed location while system initializing, so it may take 1–2 seconds. This is a long time on system's view. If the zoom motor driver is initialized on do_initcalls(), it may be the main factor of boot time delay. So we initialized these device drivers like zoom motor at boot loader, the beginning part of whole boot sequence. By doing so, zoom motor is initialized in parallel with other bootup code. This is a device dependent method.

5. **Memory allocation** – The memory allocation function like `kmalloc()` at kernel or `malloc()` at application is time consuming function. If these functions are used during bootup sequence, it may not be helpful to reducing bootup time. We improved bootup time by removing memory allocation function on bootup sequence By allocating memory after bootup, or by using memory preallocation, we can remove memory allocation function.

   - **Improvement**: tens of ms

2.7 System suspend/resume

Bootup time is very important because DSC user want to capture the image as quick as possible, The methods as we shows before are for system initialization processes. But if we use the system suspend/resume method, the bootup time reduction will be implemented very effectively. The system suspend/resume is also

one of the power management method. System suspend means that all power of the system break down except SDRAM, and in SDRAM on which we store current system information like as cpu register, I/O map information. System suspend means that there is no power in the system except SDRAM and the system remembers its state in SDRAM like as cpu registers, I/O device status, runtime global/local variables, etc.

When system receives specific events like as power button, the resuming procedure will be started. The first thing for resuming is supplying power and initializing the CPU, memory, etc. And next check if current state was in resume mode and restore all data which was in SDRAM. If we use the suspend/resume method, reducing bootup time has good efficiency, because only restoring system information from the memory is needed. Additional works to do is initialize some devices like as LCD, Motor, CCD, Image Processing Devices. We can consider next things for suspend/resume.

1. At begging of the bootloader, initialization is needed for the devices which has long initialization time like as zoom motor. Of cause these kinds of devices have the feature of the concurrent initialization.

2. When system resumed, some user would modify the DSC state. So it is need to check system state and change the DSC application for that state if changed.

3. During the system is in suspend state, the power consumption has to satisfy the requirement of marketing issues of DSC.

We can consider that if some level of time has passed, the system would be power off automatically for low power consumption. Of course, next time the DSC will be booted using normal booting.

With Samsung DSC platform, when we using the suspend/resume method for bootup time reduction, total bootup time until review is 500ms which is faster than motor initialization time. So, it is need to use faster motor devices for fast bootup time.

3 Results

3.1 Bootup time results

Using DSC specific and general bootup time reduction methods which were described before, we can get following results from Samsung DSC platform.

We show the bootup time results on Samsung DSC in Table 1. Note: Times are approximate values and in milliseconds

From this table, we can get the result that the most time consumption areas are about 4 parts: Image copy from flash memory to SDRAM at boot loader, device driver initialization area, file system related area, and DSC application initialization area. So if we achieve more bootup time reduction, we have to concentrate at above areas.

3.2 System clock speed influence

System clock speed influences not only overall performance of system but also bootup time. Following graph shows that the variation of system clock speed influences bootup time at the same DSC H/W platform.

As the results, bootup time is proportionate to the system clock speed. So it is important to using maximum clock speed the system supports.

Booting Operation			time
Bootloader	Initialize CPU & RAM & Uboot		50
	Copy kernel image (from flash to RAM)		450
Kernel Init	setup_arch ()		50
	setup_arch ()		50
	trap_init ()		10
	kmem_cache_init()		10
	mem_init ()		20
	vfs_caches_init ()		20
	page_caches_init ()		10
	rest_init()	do_basic_setup()	190
		prepare_namespace()	20
		console_open()	20
Application	ready to use file system		480
	DSC process (preview mode)		650
total			1980

Table 1: Booting time results

3.3 Flash memory

We have seen in previous section that OneNAND has the feature of NOR and NAND flash memory. It supports two kinds of read/write operation modes. The one is the synchronous burst read mode and the other is asynchronous random read mode. If we use synchronous read mode, the read time will be very fast. If system support full feature for OneNAND, like as synchronous and cached mode, its performance is almost same as the case using NOR flash.

Figure 3 shows the results of comparison of OneNAND and NOR flash. The shadowing means that kernel image will be copied into memory. This had been tested in another system by Samsung and presented at CELF for Linux NAND file system solutions [11].

S5C7380x does not support synchronous mode, but as Figure 3 shows, we have to check whether the system can support OneNAND synchronous mode when using other systems.

4 Further work

So far, we has introduced various methods for DSC bootup time reduction using Linux. But there are many other methods that were not adopted but already well known [9].

Another user application issue is that we have to check the remaining space of the card in the storage device. If there is no space to store any image, application has to display the information on LCD and has to processing relevant works. In addition, most DSC applications using the specific file system format like as DCF, which defines a common format for digital cameras for compatibility [12].

The DCF defines also the directory and file name structure at application booting time. But If we can store the first Image to internal memories like as flash or SDRAM, there is no need to initialize the card device at booting time, so we can save the time.

At the same time, if we use kernel XIP, there is no need to copy the kernel image from storage

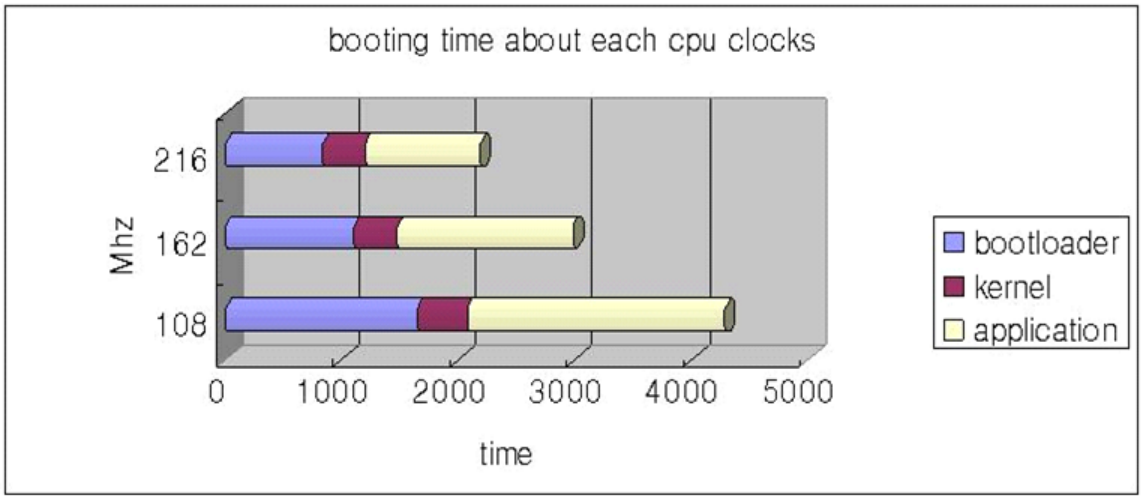

Figure 2: Bootup time about each cpu clocks. All times are in milliseconds

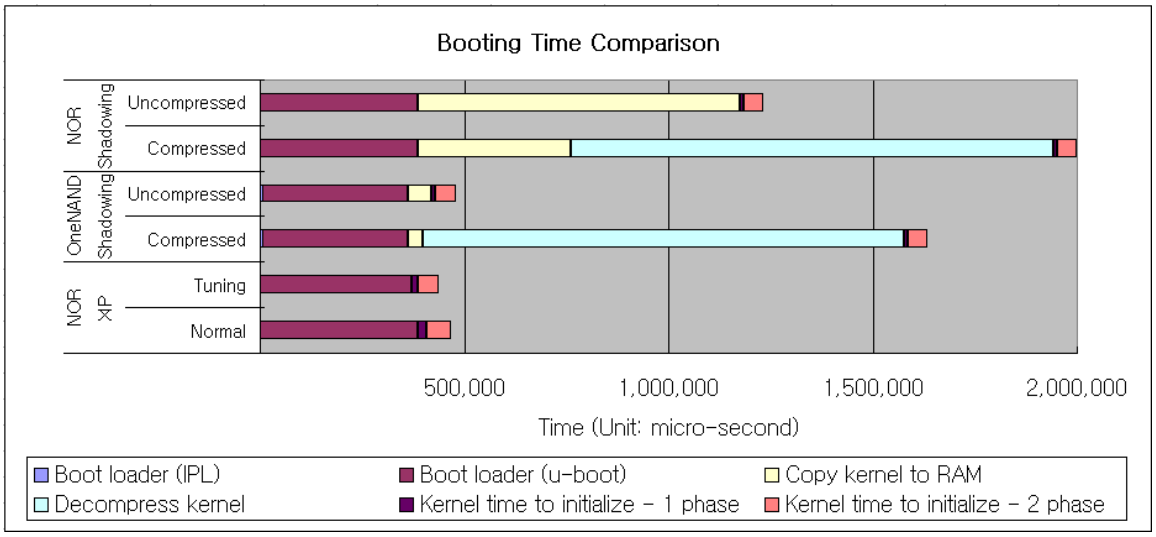

Figure 3: OneNAND booting time comparisons

device to memory, so the bootup time will be reduced dramatically. But this kind of methods gives a little bit runtime overhead and increasing the costs. There are many other methods for bootup time reduction: pre-linking, lazy linking, RTC read synch, and so forth. But in this paper these methods are not introduced [10].

5 Conclusion

The use of embedded Linux is a little bit risky on DSC for bootup time. When we implemented Linux on the DSC at first, the bootup time was more than 10 seconds. However, we get the reasonable bootup time by adopting suggested methods. Recently, the DSCs which have other RTOS show a very fast bootup time. We overcome the slow bootup time of conventional embedded Linux for DSC by using our methods.

Comparing to the performance of conventional DSC using RTOS, the DSC with embedded Linux shows a similar bootup time. As a result, we can solve the problem of embedded Linux bootup time for CE devices like the DSC. Of course, if we apply the additional method for the bootup time reduction, we can get better results.

We have to understand the feature of software and hardware of the DSC, For all of these, we have to evaluate the performance and the stability of the system although we can choose more method. After first version implementation of the DSC, the bootup time is more than 10 seconds. But when we implement the suggested methods for reduction, the bootup speed has good results.

In recently, the DSC which had other RTOS shows a very fast bootup time. And also, Samsung Linux DSC has reasonable results. Of course if we implement other methods for reduction, the bootup time will be faster To summarize, we need to understand both software and hardware of DSC and have to use the DSC specific feature. But because we can not adopt all methods for boot time reduction, we have to check which should be implemented and evaluate the over all performance results from adaptation.

References

[1] The Most Popular Operating System in the World, `http://technews.acm.org/articles/2003-5/1017f.html`, Linux Insider (10/15/03); Krikke, Jan

[2] Samsung Digimax V700 incorporates Zoran Coach 7, `http://www.letsgodigital.org/en/news/articles/story\2866.html`

[3] PLATFORM FOR CONSUMER DEVICES, VxWorks embedded real-Time Operating System (RTOS), Intel, Wind River Systems, Inc., `www.windriver.com`

[4] Embedded Linux startup reports success, growth, `http://www.linuxdevices.com/news/NS7176308845.html`

[5] Adaptability, Extensibility, and Flexibility in Real-Time Operating Systems, *Euromicro Symposium on Digital Systems Design* ,DSD'01, 2001

[6] Linux on a Digital Camera, Porting 2.4 Linux kernel to an existing digital camera *Alain Volmat Ricoh Company Ltd. Proceedings of the Linux Symposium*,July 21st-24th, 2004 Ottawa, Ontario Canada

[7] Bill Weinberg, Building Intelligent Devices with MontaVista Linux Consumer Electronics Edition, MontaVista Software, http://www.linuxpundit.com/cv/docs/wp_cee.pdf

[8] onenand_ebrochure_200503, http://www.samsung.com/Products/Semiconductor/OneNAND

[9] Tim R. Bird, Methods to Improve Bootup Time in Linux, *Sony Electronics tim.bird@am.sony.com*,Proceedings of the Linux Symposium July 21th-24th, 2004 Ottawa, Ontario Canada

[10] BootupTimeReductionHowto, http://tree.celinuxforum.org/CelfPubWiki/BootupTimeReductionHowto

[11] Case 3—comparing NOR XIP with OneNAND quick-copy to RAM, http://tree.celinuxforum.org/CelfPubWiki/KernelXIP

[12] Design rule for Camera File system, http://www.exif.org/dcf.PDF

Proceedings of the Linux Symposium

Volume Two

July 19th–22nd, 2006
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*

Review Committee

Jeff Garzik, *Red Hat Software*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat Software*
Ben LaHaise, *Intel Corporation*
Matt Mackall, *Selenic Consulting*
Patrick Mochel, *Intel Corporation*
C. Craig Ross, *Linux Symposium*
Andrew Hutton, *Steamballoon, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
David M. Fellows, *Fellows and Carr, Inc.*
Kyle McMartin