A Framework for Optimization of the Boot Time on Embedded Linux Environment with Raspberry Pi Platform

Md. Farukh Hashmi
Anurag Group of Institutions
Ghatkesar, R. R. District
Hyderabad, (India)

M. Pramod Kumar
Anurag Group of Institutions
Ghatkesar, R. R. District
Hyderabad, (India)

K. S. Rao
Anurag Group of Institutions
Ghatkesar, R. R. District
Hyderabad, (India)

ABSTRACT
Embedded system performance and utilization has increased over years, these can be observed most obviously in the electronic consumer market once a mobile phone are now replaced by smart phones and internet tablets, once a car radios are now replaced by In-Vehicle Infotainment Systems. More and more functionality is introduced into the once single-purpose system to utilize the increasing computational power, driven by the system's main target of providing improved services to the user. That implies an even faster growing complexity to be handled by the embedded systems and availability on demand. Operating system based on the Linux kernel are used in most of these consumer electronic devices, the user of these devices except these devices to be available for use very soon after being turned on. This leads to optimization of startup time for Linux. In this paper, the boot process has been described under the Linux. Initial boot time and by using some of available technique how to reduce boot time for particular application. By using the boot chart he have measured the initial boot time and optimization time. After measuring the initial time he removed the unwanted applications and services which you might not needed but they are installed by default during OS installation and unknowingly start eating your system resources. Unwanted process need to kill. In order to kill a running process in Linux, use the 'Kill PID'command. But before running Kill command, he must know the PID of the process. Here I want to find a PID of 'cupsd' process. [nano@pramod]# ps ax | grep cupsd To kill that PID, run the following command. [nano@pramod]# kill -9 1511. He describe available techniques how to reduce boot time for particular application eg .reducing kernel boot time, System startup time and application speed, Application size and Ram usage

Keywords
Embedded, Linux, Boot, Optimization, Kernel, System, Raspberry Pi.

1. INTRODUCTION
Linux developed by Linux Torvalds. Today Linux has ported on different microprocessors and runs on all sorts of platform. Embedded Linux is a type of Linux operating system/kernel that is designed to be installed and used within embedded devices and appliances. It is a compact version of Linux that offers features and services in line with the operating and application requirement of the embedded system [1].

Embedded Linux offers relief from an unstable and unfocused business model, and there are other strong reasons to consider using Linux: Complete source code availability makes it easier to fix items yourself, rather than being dependent on black boxes that are under someone else's control. Development tools to support different processors are available as free downloads, or with a support offering that comes with a cost. There are many Linux distributions available with companies and organizations that support them, creating a lack of dependence on the whims of a single company. Many universities offer Linux courses, thus future engineers will be available with basic knowledge. Availability of different programming environments such as Java, Qt, and .NET also means a company can port existing Windows applications over the embedded Linux. BSP support is available for a variety of processors: ARM, x86, PowerPC, MIPS, etc. as well as driver support for many peripherals. Linux drivers are as readily available as Windows drivers [2].

Boot time i.e. the time taken by the system to show its "availability" since the power button was pushed on, is a becoming a key differentiator in the usability factor.

The important point to understand is that optimization of boot time should not compromise the system's existing functionality and stability by any degree but in turn help the system to enhance its booting process for faster system upgrade. Before optimizing, first we have to understand the boot process, measure the initial time and optimize it by using different reduction techniques [2].

2. LITRETURE SURVEY
Optimization techniques on embedded Linux and methods to improve boot up time in Linux so far developed by authors

1. Tim R. Bird," Methods to Improve Boot up Time in Linux": Users of consumer electronics products expect their devices to be available for use very soon after being turned on. Configurations of Linux for desktop and server markets exhibit boot times in the range of 20 seconds to a few minutes, which is unacceptable for many consumer products. No single item is responsible for overall poor boot time performance [1].

2. D. P. Bovet, M. Cesati, Understanding the Linux Kernel: In the spring semester of 1997, we taught a course on operating systems based on Linux 2.0. The idea was to encourage students to read the source code [2].

3. Christopher Hallinan, Reducing Boot Time: Techniques for Fast Booting: Fast Boot is Important to Many Products Consumer, Automotive, Medical Devices, etc. Significant gains w. minimal investment first, define "boot". Does it mean Splash screen? 1st user process started? Device fully up and running, connected? Boot time is affected by many factors Hardware Design Boot loader Implementation Kernel Configuration Application Profile [3].

4. Doug Abbott: "Linux for Embedded and Real time Applications", by Doug Abbott has been of great help in providing an introduction to the process of building embedded systems in Linux. It has helped us understand the process of configuring and building the Linux kernel and installing tool chains [4].

5. Google's Larry: Linux development tools, being available through the GPL license, are available online from sources such as RDP protocol. These online helpdesks have been the source for all the tool chains that we have downloaded and the subsequent development [5].

3. PROPOSED METODOLOGY

Initially when the board starts hardware initialization take place in the boot loader .next kernel starts to load and init scripts and applications can be run the board is connected to monitor using HDMI cable power supply given to board externally. Mouse and keyboard are connected to the board directly. The Proposed system Diagram shown in figure 1 below

Block Diagram

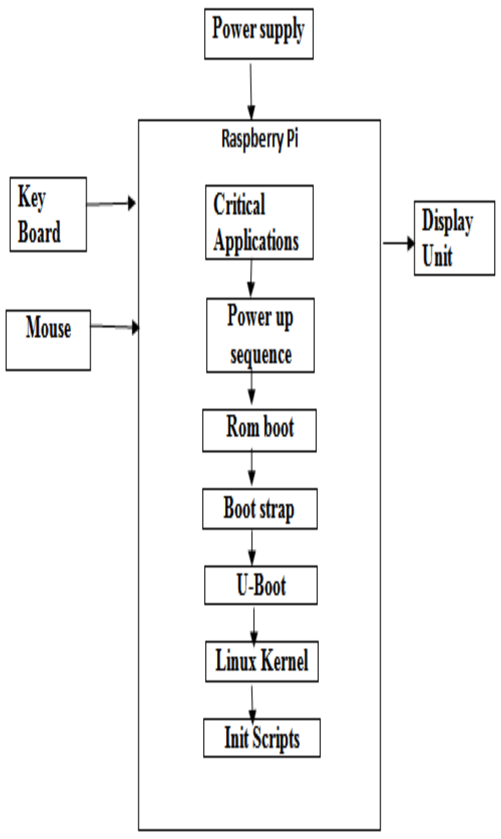


Fig 1: Detailed System Diagram

When the power is on ROM boot starts and next bootstrap starts .the boot loader is used to load the kernel and start it after loading the kernel .the kernel starts and init scripts run and after that critical applications run [3].

3.1 Various Optimization Techniques

To achieve the optimal or fast startup time the Linux embedded system need to be optimized based on the optimization process mentioned in the above section, once the need and process is identified, the optimization can be categorized and done by below activities reducing kernel boot time

Kernel boot time can be reduced by performing some or all of these activities

1. Disable IP auto config
2. Reducing the number of PTYs
3. Disable console output
4. Preset loops_per_jiffy
5. Kernel decompression
6. Reduce the kernel size
7. Faster rebooting
8. Copy kernel and initramfs from flash to RAM using DMA
9. A sync initcall
10. Deferred initcalls

3.2 System startup Time and Application Speed

System startup speed is dependent on multiple factors apart from the kernel, the file system, processor, IO and services, the optimization for startup time can be achieved by utilizing some or all the following activities.

1. Starting system services
2. Pre-fetching Reading ahead
3. Execute In Place (XIP)
4. Processor acceleration instructions
5. Use faster file systems
6. Speed up applications with tmpfs
7. Boot from a hibernate image
8. Reducing disk footprint and RAM size of the Linux kernel
9. Replacing initrd by initramfs

3.3 Application Size and RAM Usage

The application size can have a detrimental impact on the whole embedded system embedded system is generally starved for RAM and other hardware because of power consumption, size and other environmental constraints. The optimization of application size and RAM usage can be performed based on some or all of the following activities [3]

1. Static or dynamic linking
2. Library Optimizer
3. Using a lighter C library
4. Compressing file systems
5. Restartable applications
6. Merging duplicate files
7. Compiler space optimizations

4. LINUX BOOT PROCESS

4.1 Boot Process Flow

The boot time for an embedded system is of paramount importance. To optimize the boot time of the Linux, let's understand the process of booting a system running Linux operating system, the Linux booting process consists of multiple stages as shown in the figure 2 below

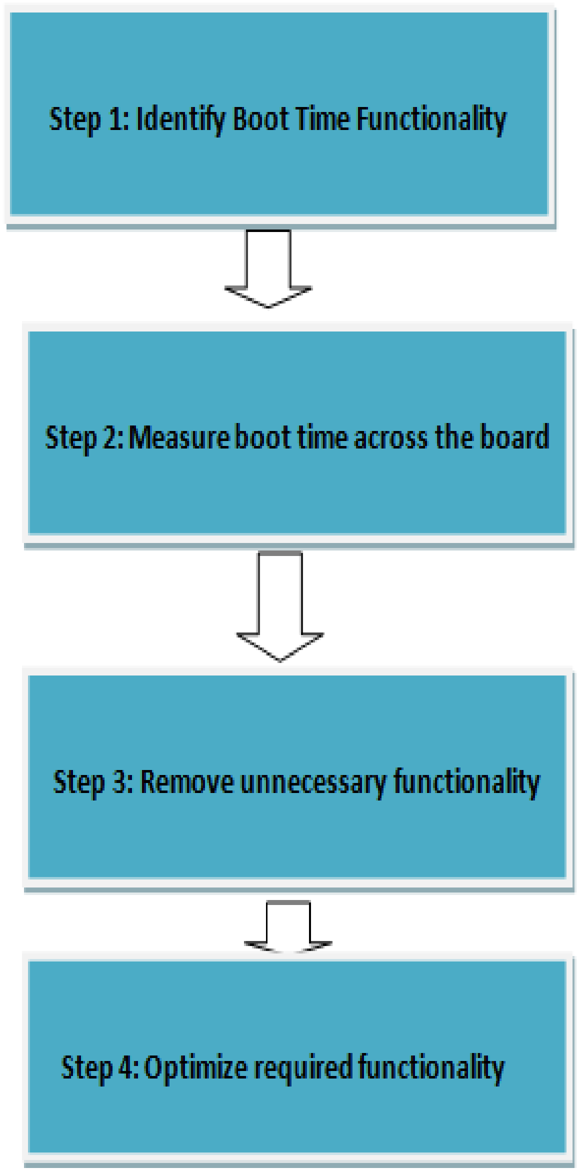

Step 1: Identify Boot Time Functionality
Step 2: Measure boot time across the board
Step 3: Remove unnecessary functionality
Step 4: Optimize required functionality

Fig 2: Process Flow for Optimization Process

This process includes early hardware initiation and interaction and loads the kernel from flash to RAM [9]-[10]. The time take during this process can be described as

1. Power/ Clock Stabilization ------ usually negligible but should be considered

2. Low Level CPU Initialization - ~ 100 ms ------ Boot loader (often multi-stage, i.e. secure boot)

Kernel Startup

This process does the following activities

1. Loading images (kernel, u -boot, root file system, dtb)

   a. Usually from NOR or NAND Flash

   b. Compressed kernel

2. Subsystem (Driver) initialization

3. Mounting a root file system

4.2 User space

The user space process covers

1. Init scripts

2. System processes

3. Applications

The user space process and configuration are very user and Applications dependent; the user space can have a display Terminal or may not have a display terminal. The best example of the user space is the user interface of the Android operating system which is working over the Linux kernel [15].

Booting components:

Boot-loader: A boot loader, also called a boot manager, is a small program that places the operating system (OS) of a computer into memory. When a computer is powered-up or restarted, the basic input/output system (BIOS) performs some initial tests, and then transfers control to the master boot record (MBR) where the boot loader resides. Here in embedded platform we use Bootstrap program (for example u-boot, red-boot)

X- Loader: The x-loader is a small first stage boot-loader derived from the u-boot base code. It is loaded into the internal static RAM by the OMAP ROM code. Due to the small size of the internal static RAM, the x-loader is stripped down to the essentials. The x-loader configures the pin muxing, clocks, DDR, and serial console, so that it can access and load the second stage boot-loader (u-boot) into the DDR.

U-boot : the u-boot is a second stage boot loader that is loaded by x-loader .the u-boot can perform CPU dependent and board dependent initialization and u-boot performs the operations not performed by x-loader[10].

Kernel: kernel is used to communicate user applications and hardware .it is interface between user and hardware. The main tasks of the kernel are: Process management, Device management, and Memory management, interrupt handling, I/O communication, File system, etc. During boot process, The kernel initializes devices, mounts the root file system specified by the boot loader as read only, and runs Init (/sbin/init) which is designated as the first process run by the system (PID = 1)

Root file system: The root file system is the file system that is contained on the same partition on which the root directory is located, and it is the file system on which all the other file systems are mounted (i.e., logically attached to the system) as the system is booted up (i.e., started up)[17].

5. HARDWARE AND SOFTWARE PLATFORM

Embedded Linux required hardware components .the hardware board used in this project is Raspberry pi board [6]. Explanation of Raspberry pi and software used in this project are explained in this paper.

5.1 Hardware Platform

The Raspberry Pi has a Broadcom BCM2835 system on a chip (SOC), which includes an ARM1176JZF-S 700 MHz processor, Video Core IV GPU, and was originally shipped with 256 megabytes of RAM, later upgraded (Model B & Model B+) to 512 MB. It does not include a built-in hard disk or solid-state drive, but it uses an SD card for booting and persistent storage, with the Model B+ using a Micro SD. Model B is the higher-spec variant of the Raspberry Pi, with 512 MB of RAM, two USB ports and a 100mb Ethernet port. It's our most popular model: you can use it to learn about computing; to power real-world projects (like home breweries, arcade machines, musical root vegetables, robot tanks and much more); as a web server; a bit coin miner; or you can just use it to play Mine craft[12]-[16].The Raspberry pi board as shown in figure 3 Below

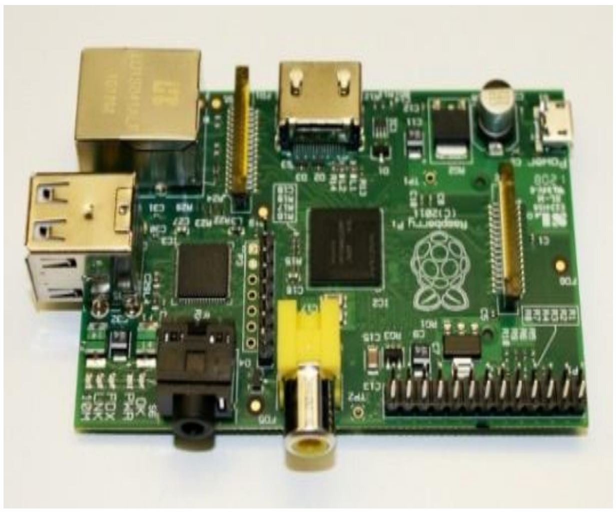


Fig 3: Raspberry pi Board

5.2 Software Platform

Linux or GNU/Linux is a free and open source software operating system for computers. The operating system is a collection of the basic instructions that tell the electronic parts of the computer what to do and how to work. Free and open source software (FOSS) means that everyone has the freedom to use it, see how it works, and changes it [7].

GCC compiler: The original GNU C Compiler (GCC) is developed by Richard Stallman, the founder of the GNU Project. Richard Stallman founded the GNU project in 1984 to create a complete Unix-like operating system as free software, to promote freedom and cooperation among computer users and programmers

1. Installation

Some simple guides to setting up the software on your Raspberry Pi. Which gives the user an operating system selection from the standard distributions? The recommended distribution for normal use is Raspbian. Alternatives are available, such as Open ELEC (XBMC media centre) or Arch Linux.

2. Installing operating system images:

How to install a Raspberry Pi Operating System image on an SD card. You will need another computer with an SD card reader to install the image.

3. Download the Image

Official images for recommended Operating Systems are available to download from the Raspberry Pi website: raspberrypi.org/downloads

Alternative distributions are available from third party vendors.

After downloading the .zip file, unzip it to get the image file (.img) for writing to your SD card.

4. Writing an Image to The SD Card

With the image file of the distribution of your choice, you need to use an image writing tool to install it on your SD card.

Installing operating system images on Linux:

Please note that the use of the dd tool can overwrite any partition of your machine. If you specify the wrong device in the instructions below you could delete your primary Linux partition. Please be careful. Run df -h to see what devices are currently mounted.

If your computer has a slot for SD cards, insert the card. If not, insert the card into an SD card reader, and then connect the reader to your computer.

Run df -h again. The new device that has appeared is your SD card. The left column gives the device name of your SD card; it will be listed as something like /dev/mmcblk0p1 or /dev/sdd1. The last part (p1 or 1respectively) is the partition number but you want to write to the whole SD card, not just one partition. Therefore you need to remove that part from the name (getting, for example, /dev/mmcblk0 or /dev/sdd) as the device for the whole SD card. Note that the SD card can show up more than once in the output of df; it will do this if you have previously written a Raspberry Pi image to this SD card, because the Raspberry Pi SD images have more than one partition [8].

Now that you've noted what the device name is, you need to unmount it so that files can't be read or written to the SD card while you are copying over the SD image.

Run un-mount /dev/sdd1, replacing sdd1 with whatever your SD card's device name is (including the partition number).

If your SD card shows up more than once in the output of df due to having multiple partitions on the SD card, you should un-mount all of these partitions.

In the terminal, write the image to the card with the command below, making sure you replace the input file if= argument with the path to your .imgfile, and the /dev/sdd in the output file of= argument with the right device name. This is very important, as you will lose all data on the hard drive if you provide the wrong device name. Make sure the device name is the name of the whole SD card as described above, not just a partition of it; for example sdd, not sdds1 or sddp1; or mmcblk0, not mmcblk0p1 [13].

```
dd bs=4M if=2015-05-05-raspbian-wheezy.img of=/dev/sdd
```

Please note that block size set to 4M will work most of the time; if not, please try 1M, although this will take considerably longer.

Also note that if you are not logged in as root you will need to prefix this with sudo.

The dd command does not give any information of its progress and so may appear to have frozen; it could take more than five minutes to finish writing to the card. If your card reader has an LED it may blink during the write process. To see the progress of the copy operation you can run pkill -USR1 -n -x dd in another terminal, prefixed with sudo if you are not logged in as root. The progress will be displayed in the original window and not the window with the pkill command; it may not display immediately, due to buffering.

Instead of dd you can use dcfldd; it will give a progress report about how much has been written [13]-[14].

You can check what's written to the SD card by dd-ing from the card back to another image on your hard disk, truncating the new image to the same size as the original, and then running diff (or md5sum) on those two images.

The SD card might be bigger than the original image, and dd will make a copy of the whole card. We must therefore truncate the new image to the size of the original image. Make sure you replace the input file if= argument with the right device name. diff should report that the files are identical.

```
dd bs=4M if=/dev/sdd of=from-sd-card.img
```

```
Truncate --reference 2015-05-05-raspbian-wheezy.img from-sd-card.img
```

```
Diff -s from-sd-card.img 2015-05-05-raspbian-wheezy.img
```

Run sync; this will ensure the write cache is flushed and that it is safe to unmount your SD card.

Remove the SD card from the card reader.

6. MEASUREMENT AND OPTIMIZATION

6.1 Initial Measurement

Optimization begins from knowing the current boot-time The overall boot process involves boot-loader(s), Linux kernel and the file system. We must identify the markers in the boot log that can be used as delimiters for each stage of the boot process. This helps in determining the time spent in each stage. By using boot chart we can calculate the initial measurement by typing this command boot chart /var/log/bootchart.tgz in the terminal after measuring the initial time we have to do optimization [11].

Boot chart: Boot chart is a tool for performance analysis and visualization of the GNU/Linux boot process. Resource utilization and process information are collected during the boot process and are later rendered in a PNG, SVG or EPS encoded chart.

Boot chart provides a shell script to be run by the kernel in the init phase. The script will run in background and collect process information, CPU statistics and disk usage statistics from the/proc file system. The performance data are stored in memory and are written to disk once the boot process completes.

6.2 Optimization Process

Linux boot optimizations methods are very platform and application dependent; the optimization need to consider the whole system architecture for selecting the boot optimization strategies

1. Size

The size dictates what would be kernel image size based on the available hardware or application, the size optimization process including

2. Speed

The speed optimization process includes

1. Optimize for target processor

2. Use faster medium for loading primary, secondary boot loaders and kernel.

3. Reduce number of tasks leading to the boot.

6.3 Remove Unwanted Services from Linux

Lets first know what kind of services are running on the system using the following commands.

```
[nano@pramod]# ps ax
```

Now, let's have a quick look at the processes accepting connection (ports) using the netstat command as shown below.

```
[nano@pramod]# netstat –lp
```

6.4 How to Kill a Process in Linux

In order to kill a running process in Linux, use the 'Kill PID' command. But, before running Kill command, we must know the PID of the process. For example, here he want to find a PID of 'cupsd' process.

```
[nano@pramod]# ps ax | grep cupsd
```

So, the PID of 'cupsd' process is '1511'. To kill that PID, run the following command.

```
[nano@pramod]# kill -9 1511
```

6.5 How to Disable a Services in Linux

In Debian based distributions such as Ubuntu, Linux Mint and other Debian based distributions use a script called updaterc.d.

For example, to disable the Apache service at the system startup execute the following command. Here '-f' option stands for force is mandatory.

```
[nano@pramod]# update-rc.d -f apache2 remove
```

After making these changes, The system next time will boot without these UN-necessary process which in-fact will be saving our system resource and the server would be more practical, fast, safe and secure.

Disable Wait for Network

Step1: sudo raspi-config

Step2: select option 4 to disable network whole booting

Step3: select Finish

Auto Login

Step 1: Open a terminal session and edit inittab file.

```
sudo nano /etc/inittab
```

Step 2: Disable the getty program.

Navigate to the following line in inittab

```
1:2345:respawn:/sbin/getty 115200 tty1
```

And add a # at the beginning of the line to comment it out

```
#1:2345: respawn:/sbin/getty 115200 tty1
```

Step 3: Add login program to inittab.

Add the following line just below the commented line

1:2345: respawn: /bin/login -f pi tty1 </dev/tty1 >/dev/tty1 2>&1

This will run the login program with pi user and without any authentication

Step 4: Save and Exit.

Press Ctrl+X to exit nano editor followed by Y to save the file and then press Enter to confirm the filename.

6.6 U-boot Optimization:

Recompile reboot to remove features not needed in production remove unnecessary functionality

1. disable features in include/configs/<soc>-<board>.h .this is board specific file

2. remove boot delay setenv bootdealy 0.it saves several seconds

3. CONFIG_ZERO_BOOTDELAY_CHECK : it allows to stop the auto boot process by hitting a key even if boot delay is set to 0.

4.simplify scripts: Some boards have over-complicated scripts:

bootcmd=run bootf0

bootf0=run ${args0}; setenv bootargs ${bootargs} \

maximasp.kernel=maximasp_nand.0:kernel0; nboot 0x70007fc0 kernel0

Let's replace this by our own scripts

5. Boot loader: copy the exact kernel size

When copying the kernel from flash to RAM, we still see many systems that copy too many bytes, not taking the exact kernel size into account. In U-Boot, use the nboot command:

nboot ramaddr 0 nandoffset

6. U-Boot - Remove kernel CRC check : Disable CRC checking with a U-boot environment variable: setenv verify no

7. Optimizing init scripts : Start all your services directly from a single startup script (e.g.

/etc/init.d/rcS). This eliminates multiple calls to /bin/sh. Replace udev with mdev. mdev is part of BusyBox. It is not running as a daemon and you can either run it only once or

have it handling hotplug events. Remove udev (or mdev) if you just need it to create device

files. Use devtmpfs (CONFIG_DEVTMPFS) instead, automatically managed by the kernel, and cheaper.

8. Reduce forking: fork/exec system calls are very expensive. Because of this, calls to executables from shells are slow. Even an echo in a Busy Box shell results in a fork syscall!

▸ Select Shells -> Standalone shell in Busy Box configuration to make the shell call applets whenever possible. Pipes and back-quotes are also implemented by fork/exec. You can reduce their usage in scripts. Example: cat /proc/cpuinfo | grep model

Replace it with: grep model /proc/cpuinfo

6.7 Reducing Kernel Boot Time

Reduce kernel size: Remove features not needed in your system: features, drivers, and also debugging functionality. By typing the command sudo rasp-config in this we can remove the features that are not requirdafter completing the optimization again we have to reboot by typing the command sudo reboot after whole optimization. Comparisons the Time without optimization and with optimization are shown Table 1.

Table 1. Comparisons the Time without optimization and with optimization

Boot Process	**WITHOUT OPTIMIZATION**	**WITH OPTIMIZATION**
u-boot	6sec	4 sec
Linux kernel	9 sec	7 sec
Init	11sec	8 sec
Total	25sec	19sec

Recompile u-boot to remove features not needed in production. Disable features in command line edition a smaller and simpler u-boot is faster to load and faster to initialize. We had removed boot delay it saves several seconds before you do boot delay recompile u-boot with config -zero-boot delay-check it allows to stop the boot process by hitting a key even if boot delay is set to zero. by simplifying scripts we saved many number of seconds .the scripts changed as our requirement it saves many number of seconds. In boot loader copy the exact kernel size from flash to memory. Instead of loading the boot loader and then the kernel load the kernel directly. Boot skips mem move operation by directly by loading the u-image at the right address Optimizing kernel code for size we reduced G-zip to lz4 it is a compressed file. To reduce the kernel size main mechanism is to use kernel modules. We compiled everything that is not needed at boot time as a module by this kernel will be smaller and load faster. In production console is not needed so disabled it by passing quiet argument in command line you still be able to get d messages to get kernel messages time between starting kernel and init without quiet and with quiet it vary. Less time will be saved in a reduced kernel. At each boot the Linux kernel calibrates a delay loop preset the loops per jiffy. Module loading and loading features are removed and power management features are removed and we have optimized the necessary functionality start all your services directly from a single startup. These eliminate multiple calls to bin shell script .replace u-dev to m-dev it is a part of busy box. Boot chart for raspberry pi with and without optimization as shown in figure 4 and figure 5

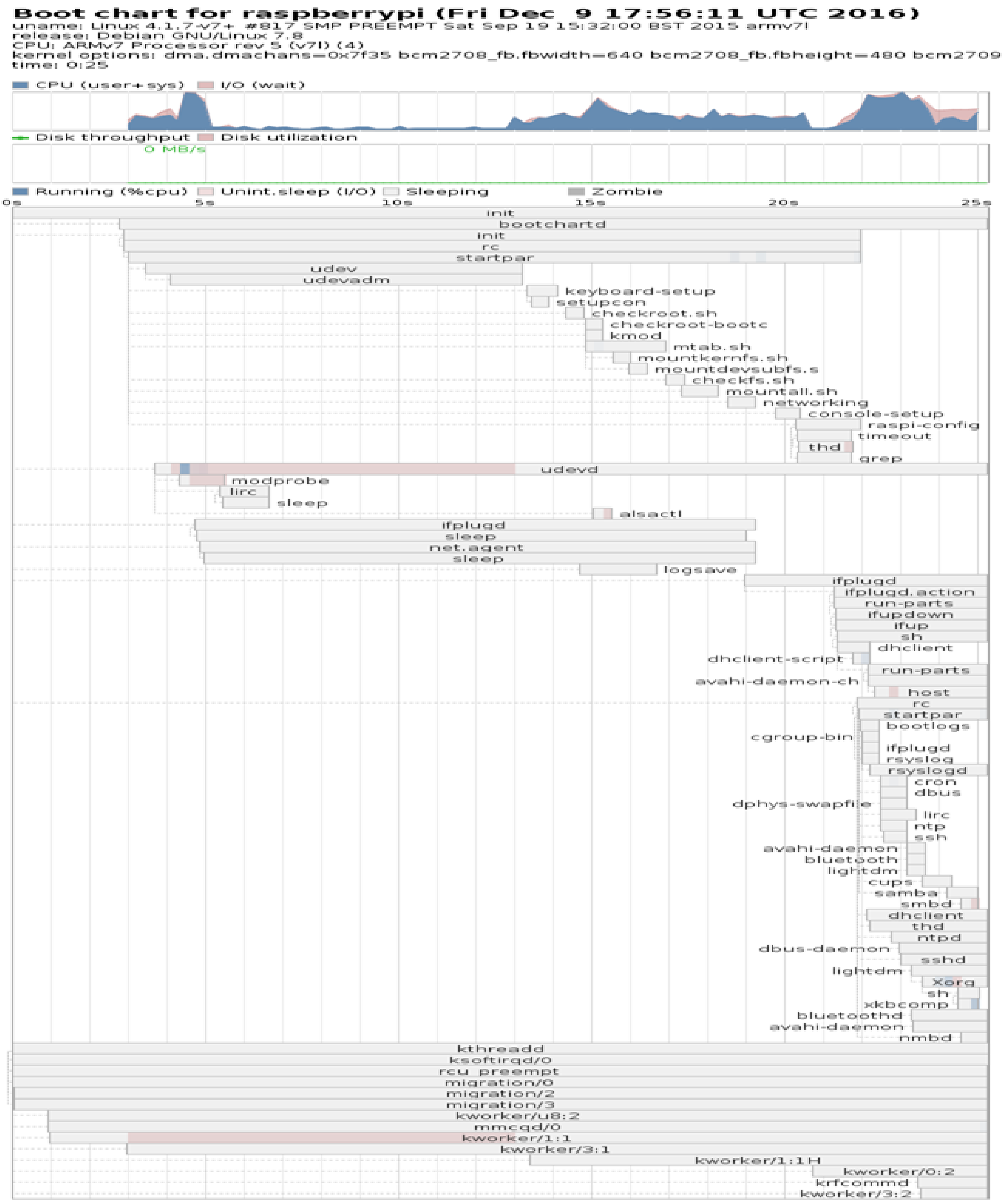
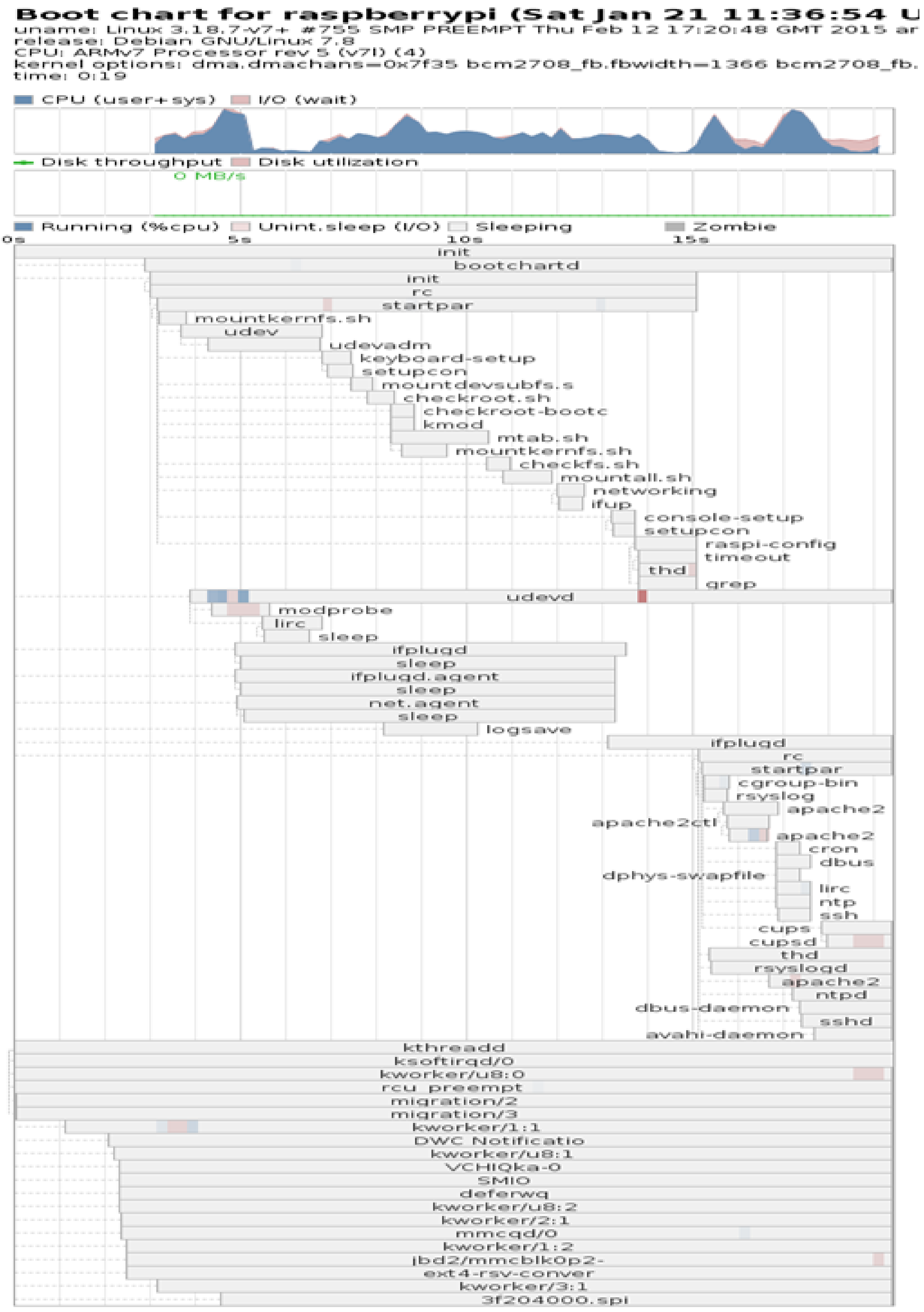


Fig 4: Boot Chart without Optimization

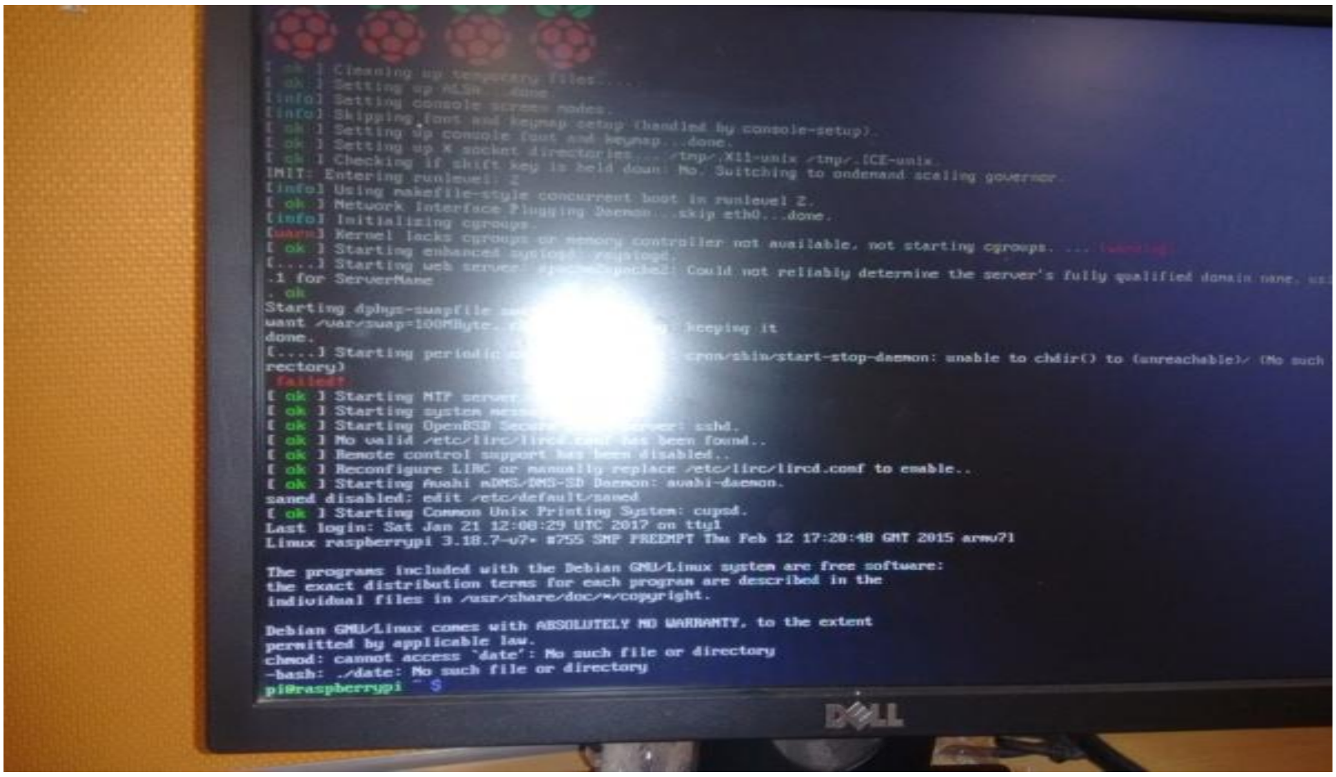


Boot chart for raspberrypi (Sat Jan 21 11:36:54 U

```
uname: Linux 3.18.7-v7+ #755 SMP PREEMPT Thu Feb 12 17:20:48 GMT 2015 ar
release: Debian GNU/Linux 7.8
CPU: ARMv7 Processor rev 5 (v7l) (4)
kernel options: dma.dmachans=0x7f35 bcm2708_fb.fbwidth=1366 bcm2708_fb.
time: 0:19
```

Fig 5: Boot Chart with Optimization

Os installation on Raspberry pi, Snapshot of Raspberry pi
Board with Accessories as shown in figure 6,7



```
[ ok ] Cleaning up temporary files....
[ ok ] Setting up ALSA...done.
[info] Setting console screen modes.
[info] Skipping font and keymap setup (handled by console-setup).
[ ok ] Setting up console font and keymap...done.
[ ok ] Setting up X socket directories... /tmp/.X11-unix /tmp/.ICE-unix.
[ ok ] Checking if shift key is held down: No. Switching to ondemand scaling governor.
INIT: Entering runlevel: 2
[info] Using makefile-style concurrent boot in runlevel 2.
[ ok ] Network Interface Plugging Daemon...skip eth0...done.
[info] Initializing cgroups.
[warn] Kernel lacks cgroups or memory controller not available, not starting cgroups. ... [illegible]
[ ok ] Starting enhanced syslogd: rsyslogd.
[....] Starting web server: [illegible]apache2: Could not reliably determine the server's fully qualified domain name, usi
.1 for ServerName
. ok
Starting dphys-swapfile [illegible]
want /var/swap=100MByte, [illegible] keeping it
done.
[....] Starting periodic [illegible] cron/sbin/start-stop-daemon: unable to chdir() to (unreachable)/ (No such
rectory)
 failed!
[ ok ] Starting NTP server[illegible]
[ ok ] Starting system mess[illegible]
[ ok ] Starting OpenBSD Secure [illegible] server: sshd.
[ ok ] No valid /etc/lirc/lircd.conf has been found..
[ ok ] Remote control support has been disabled..
[ ok ] Reconfigure LIRC or manually replace /etc/lirc/lircd.conf to enable..
[ ok ] Starting Avahi mDNS/DNS-SD Daemon: avahi-daemon.
saned disabled; edit /etc/default/saned
[ ok ] Starting Common Unix Printing System: cupsd.
Last login: Sat Jan 21 12:08:29 UTC 2017 on tty1
Linux raspberrypi 3.18.7-v7+ #755 SMP PREEMPT Thu Feb 12 17:20:48 GMT 2015 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
chmod: cannot access `date': No such file or directory
-bash: ./date: No such file or directory
pi@raspberrypi ~ $
```

Fig 6: Os installation on Raspberry pi

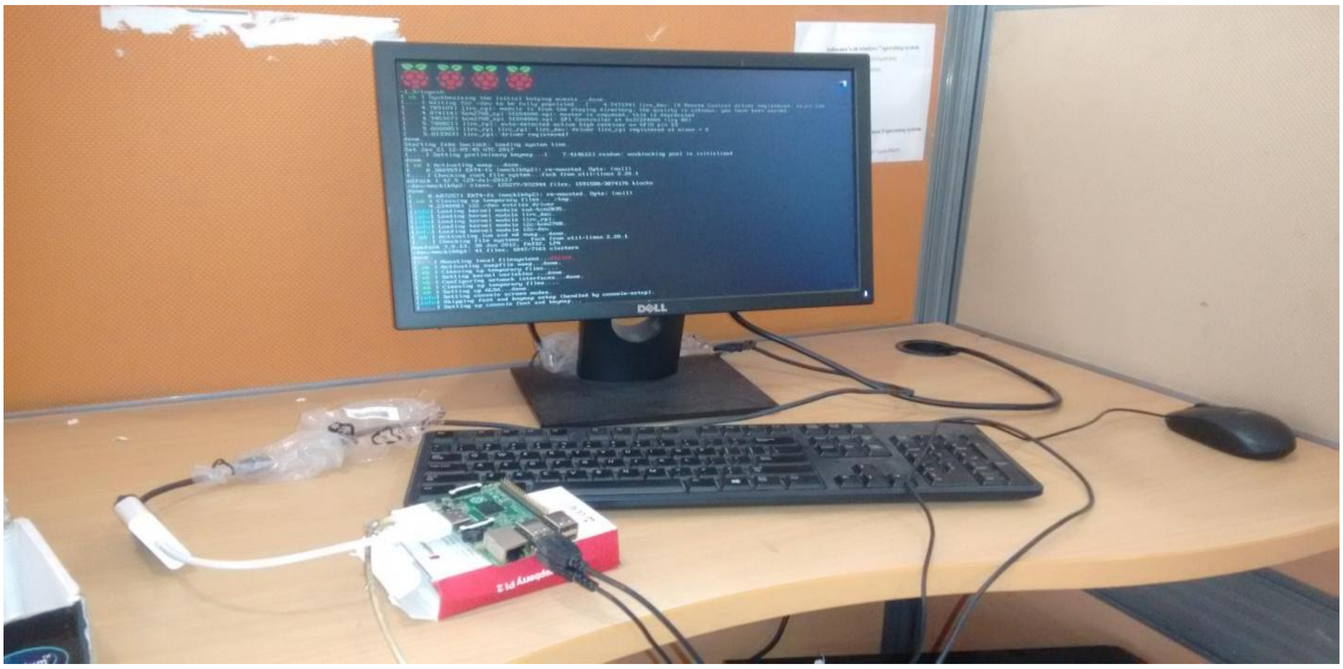


Fig 7: Snapshot of Raspberry pi Board with Accessories

7. CONCLUSION

The tool Boot chart is useful to gain insight in the boot process and to examine the different factors affecting boot time such as resource utilization. It is used in experiments performed to determine the time involved in kernel decompression. Determination of time consumed during these kinds of middle level steps can be a tedious task without help of tools like print K and Grab serial. Some of the reduction techniques which were initially meant for small embedded systems can also be used in desktops or server systems. only by few seconds but without causing any loss in functionality. Apart from these, further experiments and research need to be carried out in order to combine and implement various techniques on different systems successfully. From above results and experiments he conclude that booting process of the Linux system is analyzed under different conditions.

8. FUTURE WORK

Apart from reduction techniques discussed in many sections above, there are many other areas where research work is carried out in order to minimize total system downtime without any loss in functionality. Dynamic updates and K exec are among these areas.

9. ACKNOWLEDGMENTS

I express my sincere gratitude to my supervisor Dr. Md. Farukh Hashmi, for his precious suggestions, motivation and co-operation for the successful completion of this project.

10. REFERENCES

[1] Tim R. Bird," Methods to Improve Boot up Time in Linux, "Sony Electronics. Proceedings of the Linux Symposium Volume One July 21st–24th, 2004 Ottawa, OntarioCanada.Available:http://kernel.org/doc/ols/2004/ols2004v1-pages-79-88.pdf.[Jan.30, 2009].

[2] D. P. Bovet, M. Cesati, Understanding the Linux Kernel, O'Reilly press, 2002.

[3] Christopher Hallinan," Reducing Boot Time: Techniques for Fast Booting", MontaVista Software," http://www.mvista.com/download//power/Reducing-boot-time-techniquesfor- fast-booting.pdf [Accesses Jan 31 2009].

[4] Doug Abbott: "Linux for Embedded and Real time Applications",

[5] Google's Larry: Linux development tools Software,"http://www.mvista.com/download//power/Reducing-boot-time-techniquesfor-fast-booting.pdf [Accesses Jan 31 2009].

[6] Michael W. Godfrey, and QiangTu "Evolution in Open Source Software: A Case Study," Software Maintenance, 2000.Proceedings. International Conference on Publication Date: 2000, IEEE.

[7] Red Hat Linux Documentation, "Installing Red Hat Linux Boot Loader Configuration ," http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/installguide/ s1-x86-bootloader.html [Accesses Feb 23 2009]

[8] Linux on the IBM ESA/390 mainframe architecture, 2 Feb,2009,http://linas.org/linux/i370/i370.html. WhoUsesLinux?,Feb2009, http://www.lugod.org/presentations/ca4h/who_uses.html.

[9] Ibrahim F. Haddad," Open-Source Web Servers: Performance on a Carrier-Class Linux Platform,", Feb 2009. http://www.linuxjournal.com/article/4752.

[10] LotteMygind, Rune Hylsberg Jacobsen and Oskar Swirtun,"Introducing Linux and open source, http://www.ericsson.com/ericsson/corpinfo/publications/review/2006_01/files/.

[11] ChanjuPark,KyuhyungKim,Youngjun Jang and Kyungju Hyun ,"Linux Boot up Time Reduction for Digital Still Camera", Samsung Electronics, " ,Co. Proceedings of the Linux Symposium Volume Two ,July 19th–22nd, 2006 Ottawa, Ontario Canada . Available http://www.linuxsymposium.org/2006/linuxsymposium_procv2.pdf#page=303.

[12] Kernel XIP, http://elinux.org/Kernel_XIP Last Accesses Feb 25 2009].

[13] M. M. Lehman, D. E. Perry, and J. E Ramil. Implications of evolution metrics on software maintenance. In Proc. of the 1998 Inil.ConJ on Software Maintenance (ICSM'98), Bethesda, Maryland, Nov 1998.

[14] M. M. Lehman, J. E Ramil, P. D. Wemick, D. E. Perry, and W. M. Turski ," Metrics and laws of software evolution – the nineties view ," In Proc. of the Fourth Intl. Software Metrics Symposium (Metrics'97), Albuquerque, NM, 1997.

[15] Inwhee Joe, Sang Cheol Lee. "Bootup Time Improvement for Embedded Linux using Sanpshot Images Created on Boot Time," Proc. Of The 2nd Int. Conf. on Next Generation Information Technology-ICNIT, Gyeongju, South Korea, 2011

[16] https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/README.md

[17] https://en.wikipedia.org/wiki/Linux_startup_process