Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Programa de Maestría en Computación

Improving Operating Systems for Efficient Multimedia Handling in Embedded Systems

Thesis Proposal
Submitted in partial fulfillment of the requirements for the degree of

Magister Scientiæ en Computación

Author:
Diego Dompe

Advisor:
Ph.D. Francisco J. Torres-Rojas

May 2011

Abstract

Embedded Systems are one of the fastest growing markets in the computer industry, but the discipline of embedded software development is still young; best practices common on other software development areas aren't encouraged by the market dynamics, product development cycles and the requirements of low-level integration with the hardware. Furthermore embedded developers tend to shy away from existing frameworks that may improve re-usability when they can't meet their performance goals. This is particularly true for multimedia systems, where the amounts of data processing may easily impose severe overhead in an embedded architecture. All these factors lead to low re-usability of the software written for embedded systems. This work addresses the problem of improving multimedia software stacks for embedded systems, focusing on the GStreamer multimedia framework with the Linux operating system.

Resumen

Resumen en español

CONTENTS

CHAPTER

ONE

INTRODUCTION AND BACKGROUND

> People in the embedded space don't do prototypes. They hack something until it works, then it's done.
>
> xav user on lwn.net

1.1. Introduction

Embedded Systems are one of the fastest growing markets in the computer industry, with more than 10 billion embedded processors shipped in 2008 alone[7]. We can define embedded systems as microprocessor-based systems built to control a function or range of function, not designed to be programmed by the end user in the same way that a Personal Computer (PC) is[13].

Embedded Software development as discipline is younger than software development as intended for other environments like PCs or Mainframes. There are some particular issues that transform embedded software development into a complex eco-system:

- It usually requires a significant level of integration with the hardware that is uncommon in other software development areas.

- Most of the hardware integration that is done is very particular and unique, which usually makes the code hard to re-use if not designed properly (or if the technology is new and it isn't clear how to implement the software solution in a generic way).
- The market for embedded system is so fast-paced that developers are constantly pressured to deliver working devices as soon as possible. Few companies care about the quality or re-usability of their embedded code, except for hardware vendors or efforts from single open-source minded developers who want to contribute back. For example recent articles regarding the status of the contributions to the Linux™ kernel for the Advanced RISC Machine (ARM) architecture qualify these contributions' status as a "mess" due to the many different interests and agendas behind it [23].
- Many embedded developers may lack formal training on software development, but instead have an Electronic Engineering (EE) background.

These factors usually mean that embedded developers will not likely find or apply the same best-practices that the rest of the software industry uses. It is not rare to find embedded projects where each revision of the same product will have a completely new software stack or re-implementation from the previous iteration.

The fact that the code is tailored for specific hardware to the point where it can't be easily re-used goes against Operating Systems (OSs) theory which proposes that the OS should abstract all the software from the hardware implementation [27, p. 29]. However many embedded devices (especially ones with low memory footprint) may use a custom-made OS or use no OS at all (also called OS-less solutions).

1.1.1. Linux on embedded systems: relevance and challenges

On embedded systems capable enough to support it, Linux has gradually gained acceptance as an OS due to its open-source nature and support from embedded processors manufacturers, especially on the System on a Chip (SoC) market. SoCs are the main

trend for new product development in recent years due to the advantages provided by new chip development [28].

The advantages that Linux provides for the development of new embedded products are clear:

- It's a widely deployed OS which has been successfully used in different embedded markets.
- No royalty fees or licenses costs are associated directly with it (although some embedded vendors may add royalty fees if you use their custom distributions).
- Wide knowledge base available and many developers are familiarized with it.
- Usually hardware companies provide drivers and support for Linux in their latest devices (for example ethernet or wireless circuits, storage devices, etc.), simplifying the integration of hardware components for an embedded system. This is a critical element for minimizing the time-to-market of a product.

However Linux also presents some challenges for adaption within the embedded space:

- It's not designed to be a Real Time Operating System (RTOS), which make it not suitable for all type of embedded applications.
- There is a learning curve for RTOS developers, since Linux enforces some concepts that may be foreign to them:
    - Memory space protection: most RTOS do not provide memory space protection.
    - Separation between the kernel and user space applications: This is a side-effect of the memory space protection.
- There are several algorithms used in non-RTOSs like Linux that may affect the performance compared to RTOSs or OS-less solutions:

- Context Switches: there is a penalty associated on several hardware architectures with the memory protection.
- I/O Scheduling: the kernel page cache may defer I/O work. Linux offers full control to customize this behavior for embedded systems, but many developers may be unaware of it.
- Interrupts prioritization: mainline Linux kernel doesn't offer a way to control priorities for interrupt handlers. There are patches to transform interrupt handlers into kernel threads, but they aren't available for all kernel versions and may require specific analysis and tuning to achieve the desired results [29].

Despite the short-comings that Linux presents versus a RTOS, just the fact of having widely available drivers and software stacks for peripherals on the embedded space usually outweighs these problems. Many SoCs may require complex software stacks to interact with integrated peripherals; such is the case of heterogenous SoCs using a General Purpose Processor (GPP) and a Digital Signal Processor (DSP) or other sort of hardware unit for accelerating specific operations (such as video coding, cryptographic algorithms, etc). Implementing or porting the functionality of these software stacks required to interact with the desired peripheral to an OS-less or other non-supported RTOS is usually non-effective in terms of cost or time.

However many developers that are un-familiar with Linux and/or general purpose OSs design, approach the task of implementing the functionality for their embedded products bypassing Linux functionality. For example the author of this work has found several times kernel drivers for Linux designed just to specifically allow user-space access to the peripherals by any user space application in commercial Original Equipment Manufacturing (OEM)/Original Design Manufacturing (ODM) offerings for embedded development. There are several reasons for developers using this kind of approach:

- Some times the software stack was designed in an OS-less environment and ported

over to Linux. Developers just try to find a way to get the Linux hardware management out of the way of the software stack.

- Developers are simply un-familiar with Linux design.
- Linux may lack APIs or functionality to handle new scenarios posted by the hardware design. For example the ability to execute zero-memory-copy operations.
- Developers try to use Linux's APIs but find that performance is un-acceptable for unknown reasons. In the embedded community it is widely believed that general purpose OSs like Linux may be unsuitable for certain embedded applications since it is considered "bloated" for desktop systems.

Another common belief in the embedded community is that any highly specialized software design will provide better performance than implementing it over a generic solution. One example of this is multimedia software architectures on top of Linux, where the amount of data being processed requires data paths with zero memory copy operations across the whole software stack.

1.2. Problem statement

Many embedded developers tend to create software designs and architectures that aren't re-usable or maintainable for different reasons; one of them is the empiric premise that generic software architectures penalize performance. Furthermore there are little incentives from the product development cycle for embedded products to invest into optimizing the OSs. All these factors lead to the creation of software platforms that are hard to reuse or leverage into future project versions or other projects, potentially increasing the costs of development.

The described scenario in particular true for multimedia software stacks for embedded systems. Multimedia software stacks move big amounts of data at very high speeds between several hardware devices and across different subsystems in the OS. This work

focuses on improving the performance of GStreamer, a multimedia software stack for Linux, on the ARM architecture. GStreamer is a popular solution for embedded systems, and there are known issues with his performance but not complete clarity about their source[8].

1.3. Hypothesis

It's possible to improve multimedia software stacks in general purpose Operating Systems like Linux to minimize processing overhead to be negligible in terms of CPU, memory and I/O requirements.

1.4. Previous work

Plagemann et al.[20] provided a very comprehensive review of the existing work in the field regarding customization of OS algorithms for multimedia scenarios in respect to CPU and disk scheduling, resource allocation, filesystem structures, memory management and I/O subsystems. Their work summarizes different approaches and developments from the 90's decade, but the publication is now 12 years old and there doesn't seem to be more up-to-date surveys as comprehensive as this work.

Linaro's Multimedia Working Group (LMWG) has begun working recently on similar problems[25] to the one stated in this work. One of the short term objectives of LMWG is to work on multimedia validation tools to improve the understanding of system behavior. The scope of the present work will be narrower than the one aimed to by LMWG, and eventually this work could be a contribution to it.

1.4.1. Literature review

Performance issues can be analyzed from the perspective of system design. There are classic papers that address the different design approaches for general purpose

Operating Systems and their respective tradeoffs [12][2][17]. Embedded RTOSs are usually designed without memory or process protection; an approach that is well-suited for direct-access to the hardware from the applications, eliminates any type of overhead induced by the OS and allows deterministic latency in the scheduler.

The debate regarding general purpose Operating Systems design is a long standing and recurring topic in Computer Science (the Tanenbaum-Torvalds debate [10] has came up recently again[30] and even in other forms by different proponents[14]).

Of particular interest is Liedtke[17] work, since he showed that by using the right algorithms and analysis it's possible to implement software stacks that manage hardware access with minimal performance penalties. He faced the common belief that μ-kernels suffer from inherent design performance penalties and demonstrated that rather than a design issue it was an implementation issue the caused the performance problems. The problem of multimedia stack performance on embedded systems could be viewed as a generalization of the same problems faced by Liedtke:

- Common belief that the existing implementation is inherently inefficient by design.
- The problem can be approached by improving the algorithms and data structures used in the existing implementation.

However a fundamental part to approaching the issue of performance improvements is to find tools to validate the different dynamic scenarios. There is previous published work by this author regarding instrumenting GStreamer for specific multimedia scenarios[11] that includes some tools and methodologies to analyze its performance and, as mentioned previously, developing these tools is also an objective of the LMWG. Some simple tests has been written by GStreamer developers when analyzing specific performance issues[8] as well, however these tools are usually limited to instrumenting GStreamer and not the complete system behavior or characterization of its interactions with the different Linux subsystems.

To properly address the different factors involved in the performance of multimedia software stacks for embedded systems the work has to be split up into different areas:

- CPU scheduling and Memory Protection: scheduling and real-time behavior taking in consideration the tradeoffs imposed by the memory protection. On modern SoCs the scheduling should consider offloading work to co-processors in boss-worker modes.
- I/O utilization: data structures and algorithms for properly handling of I/O from and among different subsystems on modern complex platforms like heterogenous multi-core SoCs.

The following section addresses existing literature regarding multimedia software stacks and the different areas previously mentioned.

1.4.2. CPU Scheduling and Memory Protection

Poudel[21] reported notable performance gains in data throughput, during the development of the OpenCV DSP Acceleration project with the use of asynchronous processing APIs to the coprocessor (DSP). Even if this is not an scheduling algorithm per-se it shows the relevance of accounting for the nature of the processing architecture in the final throughput of the system. Regehr et al.[24] explored the relevance of the scheduling algorithms and the programming model to properly take advantage in the design of multimedia applications.

One of the main performance penalties and predictability for general-purpose OSs are due the memory protection semantics[3]. Liedtke's[17] work showed that it is important to provide architecture specific optimizations, so we need to explore work regarding optimizations for the ARM architecture. David et al.[9] found significant context switch overhead for Linux in the ARM platforms. Chanteperdrix and Cochran [4]) among others have implemented solutions to address this problems with different limitations, but

have never made it to the mainline Linux kernel.

1.4.3. I/O utilization

There are many previous works regarding optimizing OSs for efficient I/O handling of multimedia cases, with a special focus on optimized network streaming from disk (YIMA[26],MARS[15]), or optimizing the networking stack for zero-memory-copy (LyraNet[16], EIORTP[18], Solaris[5]). However these works don't address modern hardware requirements of embedded SoCs like other I/o subsystems beyond networking.

Work addressing improved I/O throughput on modern SoC architectures is found in some conference presentations for the SH processor[19], and the OMAP4[6]. These work present useful information regarding community work to address some of the problems that will be researched by this thesis.

1.5. Long-Range Consequences

This work will contribute to the analysis of the factors involved on measuring and improving performance to the Linux kernel for its use on embedded systems dealing with multimedia software stacks. Such work will contribute to industry-wide efforts like the LMWG.

CHAPTER

TWO

OBJECTIVES AND CONTRIBUTIONS

> Don't find fault, find a remedy.
>
> Henry Ford

2.1. General Objective

- Implement and analyze changes in the algorithms and data structures of the multimedia software stack GStreamer and Linux Operating System in order to improve performance and reduce penalties in embedded systems scenarios using the ARM architecture.

2.2. Specific Objectives

- Instrument and measure interactions and performance of the Linux kernel and GStreamer for a specific set of embedded systems, specialized for multimedia data processing, using a well-defined set of multimedia processing scenarios.
- Propose and implement algorithms or changes required to improve performance, aiming to generate reusable solutions that may be integrated on mainline projects.

2.3. Contributions to the subject

LMWG's Christian Reis, stated that "[multimedia is] one of the most complex and poorly-understood areas on Linux, stemming from the inherent challenges in providing high performance multimedia, IP restrictions on technology and content and the impressive rate at which new formats and capabilities have been developed to match increasing network and processing power" [25]

This work will provide information, tools and methodologies regarding the details of the specific tradeoffs for the implementation approaches of multimedia software on the target platforms, and could serve as basics for future work.

2.4. Scope and Limitations

Work will be done on the selected platforms using the latest version of the software (Linux and GStreamer) available for them.

This work will not cover the following subjects (these are subjects that can be explored in future work):

- Power consumption issues for the embedded platform.
- Real-time behavior for the multimedia stack.

2.4.1. Hardware

Work will be done specifically using the multimedia framework GStreamer on Linux embedded systems running on SoCs from Texas Instruments on two different architectures:

- DM36x architecture: this SoC has non-programable hardware accelerators to offload codec processing.
- DM37x architecture: this SoC has a DSP co-processor to offload codec processing.

These architectures have been selected due to the availability of cheap hardware (Beableboard-XM and Leopardboard), and the available support for GStreamer on both. Both platforms provide access to their underlaying hardware acceleration capabilities using the CodecEngine software stack[22].

The reason for using two different platforms is that they have different levels of processing power on their main processor, with the DM36x family sporting an ARMv5 core and the DM37x an ARMv7. We want to identify whether there are notable differences caused by the specific ARM architecture being used.

CHAPTER

THREE

METHODOLOGY AND SCHEDULE

> There is no greater harm than that of time wasted.
>
> Michelangelo

3.1. Methodology

Tools for characterizing and analyzing performance on each specific scenario will be identified and integrated on the testing platforms. There are existing open source tools that provide the required information, but little information regarding specific use to instrument the desired scenarios.

The following are the specific scenarios that will be evaluated on each platform:

- Video capture, and encoding.
- Audio capture, and encoding.
- Synchronized video and audio capture, encoding and storage in block-based storage devices using an standard multimedia file container.
- Synchronized video and audio capture, encoding and transmission over network using RTP/UDP/IP protocol.

- Video decoding and preview from block-based storage devices using a standard multimedia file container.

For block-based storage device we consider any storage device with semantics similar to that of standard hard disks, and specifically excluding devices based on flash memory unless they use any sort of Flash Transition Layer (FTL).

The following aspects will be considered for each specific scenario:

- CPU usage and scheduling overhead.
- Memory usage and overhead by the software stack.
- I/O usage: efficiency regarding use of the specific I/O architecture and overhead by the software stack.

3.2. List of Deliveries

The deliveries are:

1. Patches for the Linux kernel available for the specific platforms to improve their performance.
2. Patches for GStreamer to improve performance on the target scenarios and platforms.

As stated previously, all the patches will be designed with the intention of being accepted into their respective projects, but no guarantee is made about their merging on mainstream. Still, the work will be done while interacting with the respective project maintainers.

3.3. Schedule and Work Breakdown

Table 3.1: Work Breakdown

Task	Duration
1) Review Literature and write down existing work	**4w**
1.1) Review existing research and document the previous work	3w
1.2) Incorporate feedback from reviewers	1w
2) Collect initial performance measurements	**8w**
2.1) Review existing tools for performance measurement	1w
2.2) Define testing methodology and tools	2w
2.3) Setup working environment and tools	2w
2.4) Collect measurement information about initial status of the platform for a set of scenearios	2w
2.5) Incorporate feedback from reviewers	1w
3) Review performance optimizations at multimedia platform level	**5w**
3.1) Review current performance at no-op case for test scenarios	1w
3.2) Review possible performance optimizations for no-op cases	4w
4) Review performance optimizations for I/O operations	**22w**
4.1) Review performance for video interfaces	5w
4.2) Review performance for audio interfaces	3w
4.3) Review performance for co-processor interfaces	4w
4.4) Review performance for disk I/O	4w
4.5) Review performance for network I/O	6w
5) Write Thesis Document	39w

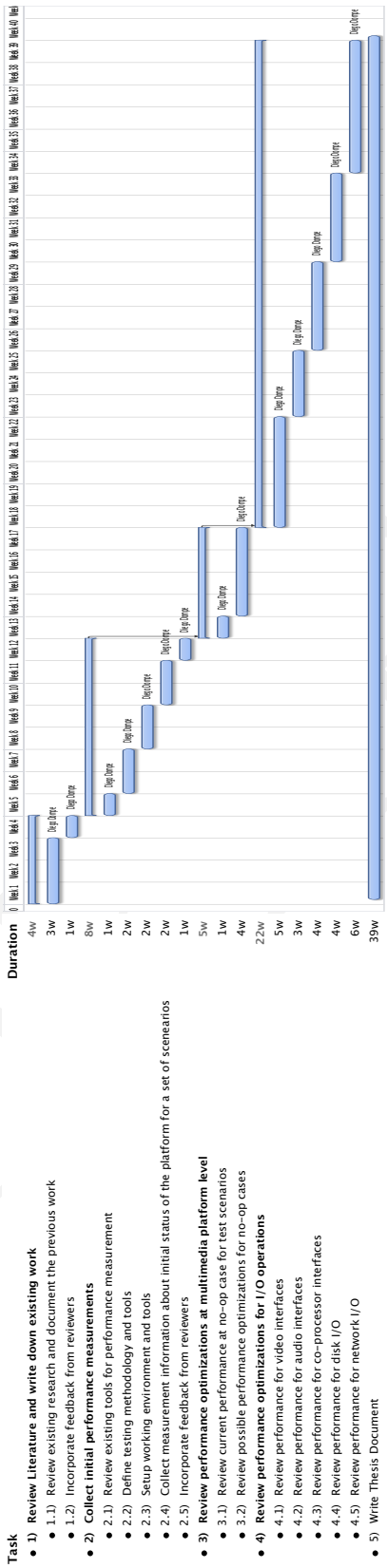

Figure 3.1: Gantt Chart

BIBLIOGRAPHY

[1] Difference Between ODM and OEM. URL http://www.differencebetween.net/miscellaneous/difference-between-odm-and-oem/.

[2] Brian N. Bershad, Stefan Savage, Przemys Pardyak, Emin Gun Sirer, Marc E. Fiuczynski, David Becker, Craig Chambers, and Susan Eggers. Extensibility, safety and performance in the SPIN operating system. pages 267–284, 1995.

[3] Bernard Blackham, Yao Shi, and Gernot Heiser. Protected Hard Real-time: The Next Frontier. In *Proceedings of the 2nd Asia-Pacific Workshop on Systems*, Shanghai, China, Jul 2011.

[4] Gilles Chanteperdrix and Richard Cochran. The ARM Fast Context Switch Extension. In *Proceedings from the Real Time Linux Workshop*, 2009.

[5] Jerry Chu and Sunsoft Inc. Zero-Copy TCP in Solaris. In *In Proceedings of the USENIX 1996 Annual Technical Conference*, pages 253–264, 1996.

[6] Rob Clark. GStreamer and OMAP4, April 2010. URL http://people.freedesktop.org/~robclark/gstcon2010.pdf.

[7] Peter Clarke. Embedded processor market to resist financial crisis, say VDC. *EE Times*, 2009. URL http://www.eetimes.com/electronics-news/4194731/Embedded-processor-market-to-resist-financial-crisis-say-VDC.

[8] Felipe Contreras. GStreamer, embedded, and low latency are a bad combination. URL http://felipec.wordpress.com/2010/10/07/gstreamer-embedded-and-low-latency-are-a-bad-combination/.

[9] Francis M. David, Jeffrey C. Carlyle, and Roy H. Campbell. Context switch overheads for linux on arm platforms. In *Proceedings of the 2007 workshop on Experimental computer science*, ExpCS '07, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-751-3. doi: http://doi.acm.org/10.1145/1281700.1281703. URL http://doi.acm.org/10.1145/1281700.1281703.

[10] Chris DiBona, Sam Ockman, and Mark Stone, editors. *Open Sources: Voices from the Open Source Revolution*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1999. ISBN 1565925823.

[11] Diego Dompe and Francisco J. Torres-Rojas. Tuning GStreamer on Embedded Systems for Soft Real-Time. In *Proceedings of the Embedded Technology Conference 2011*, volume 1, pages 44–50, 2011.

[12] Dawson R. Engler, M. Frans Kaashoek, and James O'toole. Exokernel: An operating system architecture for application-level resource management. pages 251–266, 1995.

[13] S. Heath. *Embedded systems design*. EDN series for design engineers. Newnes, 2003. ISBN 9780750655460. URL `http://books.google.com/books?id=BjNZXwH7HlkC`.

[14] Gernot Heiser. Do Microkernels Suck? URL `http://www.ok-labs.com/blog/entry/do-microkernels-suck/`.

[15] Xin Jane, Chen Milind, M. Buddhikot, Dakang Wu, and Guru M. Parulkar. Enhancements to 4.4 BSD UNIX for Efficient Networked Multimedia in Project MARS. In *In Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS'98*, pages 326–337, 1998.

[16] Yun-Chen Li and Mei-Ling Chiang. LyraNET: A Zero-Copy TCP/IP Protocol Stack for Embedded Operating Systems. *Real-Time Computing Systems and Applications, International Workshop on*, 0:123–128, 2005. ISSN 1533-2306. doi: http://doi.ieeecomputersociety.org/10.1109/RTCSA.2005.57.

[17] Jochen Liedtke. On μ-kernel construction. In *15th ACM symposium on Operating Systems Principles*, 1995.

[18] Nam-Sup Park and Chong-Sun Hwang. Effective I/O Scheme Based on RTP for Multimedia Communication Systems. *Journal of Computer Science and Technology*, 21:989–996, 2006. ISSN 1000-9000. URL `http://dx.doi.org/10.1007/s11390-006-0989-5`. 10.1007/s11390-006-0989-5.

[19] Conrad Parker. A Linux multimedia platform for SH-Mobil processors, April 2009. URL `http://www.lca2010.org.nz/slides/50315.pdf`.

[20] T. Plagemann, V. Goebel, P. Halvorsen, and O. Anshus. Operating system support for multimedia systems. *Computer Communications*, 23(3):267 – 289, 2000. ISSN 0140-3664. doi: DOI:10.1016/S0140-3664(99)00180-2. URL `http://www.sciencedirect.com/science/article/pii/S0140366499001802`.

[21] Pramod Poudel. OpenCV DSP Acceleration. URL `http://pramodpoudel.blogspot.com/`.

[22] Steve Preissig. Programming detals of the codec engine for davinci™ technology. Technical report, Technical Training Organization, Texas Instruments, 2006.

[23] Brian Proffitt. Linux ARM support: A hot mess, an ugly clean-up. *ITworld*, 2011. URL `http://www.itworld.com/mobile-wireless/175829/arm-and-linux-major-construction-ahead`.

[24] John Regehr, Michael B. Jones, and John A. Stankovic. Operating System Support for Multimedia: The Programming Model Matters. Technical report, 2000.

[25] Christian Robottom Reis. Multimedia on Linux. URL `http://www.linaro.org/linaro-blog/2011/07/15/multimedia-on-linux/`.

[26] Cyrus Shahabi, Roger Zimmermann, Kun Fu, and Shu-Yuen Didi Yao. Yima: A Second- Generation Continuous Media Server. *IEEE Computer*, pages 56–64, 2002.

[27] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts Essentials*. Wiley Publishing, 2010. ISBN 04701287209.

[28] Deepak Somaya and Greg Linden. System-on-a-Chip Integration in the Semiconductor Industry: Industry Structure and Firm Strategies. *SSRN eLibrary*, 2000. doi: 10.2139/ssrn.259878.

[29] Yang Song and Peter Chubb. Interrupts Considered Harmful. In *11th Linux.conf.au*, Wellington, New Zealand, Jan 2010. annotated slides from presentation.

[30] Andrew Tanenbaum. Tanenbaum-Torvalds Debate: Part II. URL `http://www.cs.vu.nl/~ast/reliable-os/`.

ACRONYMS

ARM Advanced RISC Machine is a hardware architecture designed by ARM Holdings and licensed to many companies worldwide.

DSP Digital Signal Processor is a special type of micro processor.

EE Electronic Engineering is the engineering area dedicated to electronic circuits technologies.

ECC Error Correction Codes are data used to calculate and correct errors in blocks of data.

FTL Flash Transition Layer is a software stack that allows to access flash storage with the same semantics for block read/write as standard hard disk, taking care of wear-leveling for the flash device and Error Correction Codes (ECC) handling on flash that require it.

GPP General Purpose Processor is a processor suited for standard computational operations, in contrast of a DSP for example.

LMWG Linaro's Multimedia Working Group is working group inside Linaro project that focus on Multimedia improvements for Linux on the ARM platform.

OEM Original Equipment Manufacturing refers to a company or a firm that is responsible for designing and building a product according to its own specifications, and then selling the product to another company or firm, which is responsible for its distribution. The one company produces products on behalf of another company, after which the purchasing company markets the product under its own brand name[1].

ODM Original Design Manufacturing is a company or firm that is responsible for designing and building a product as per another company's specifications[1].

OS Operating System

PC Personal Computer

RTOS Real Time Operating System is an operating system that is designed to meet with Real Time deadlines.

SoC System on a Chip is a technology that consist on integrating several functional blocks of re-usable electronics logic (even from different vendors) into single die.