MASTER'S THESIS | LUND UNIVERSITY 2016

Emulation-based Software Development for Embedded Systems

Simon Wallström, Adam Dalentoft

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2016-15

Emulation-based Software Development for Embedded Systems

Simon Wallström
`wallstrom.simon@gmail.com`

Adam Dalentoft
`dat11ada@student.lu.se`

June 3, 2016

Master's thesis work carried out at Axis Communications.

Supervisors: Flavius Gruian, `Flavius.Gruian@cs.lth.se`
Gustav Sällberg, `Gustav.Sallberg@axis.com`

Examiner: Krzysztof Kuchcinski, `Krzysztof.Kuchcinski@cs.lth.se`

Abstract

Software development for embedded systems has a lot of dependencies on the hardware of the system. To possibly reduce the lead time and ease the development process, an emulated model is investigated. This model is examined in terms of how it fits in the platform transition, daily development and testing processes at Axis. The results indicate that an emulated model is a powerful tool for a software developer. The emulator let developers start developing software before physical hardware exists and allows for peripheral exploration in the emulated environment. Results from substituting physical labs with emulators for testing yielded promising results in terms of execution speed but as of now it does not scale very well. The conclusion is that hardware emulation in embedded software development has a great potential to improve the overall process.

Keywords: MSc, Embedded Software, Emulation, QEMU, Software Development

Acknowledgements

We would like to thank our supervisors at Axis, Gustav Sällberg and Petter Johansson, for their support throughout the thesis work. We also thank Flavius Gruian, our supervisor at Lund University Faculty of Engineering, for great discussions and guidance throughout all steps of the thesis.

We would also like to thank Axis employees Rabin Vincent for his knowledge about QEMU and Andreas Olsson for his insights in Axis' software development process. Lastly we thank Jenny Kihlberg for her feedback and support during the writing of the report.

Contents

List of abbreviations

AMS	Analog/Mixed-Signal.
API	Application Programming Interface.
ASIC	Application-Specific Integrated Circuit.
ATF	Automated Testing Framework.
CPU	Central Processing Unit.
DHCP	Dynamic Host Configuration Protocol.
dtb	Device tree blob, compiled from a device tree source (.dts) file.
FPGA	Field Programmable Gate Array.
GDB	GNU Debugger.
IDE	Integrated Development Environment.
IEEE	Institute of Electrical and Electronics Engineers.
IP	Intellectual Property.
IP address	Internet Protocol Address.
ISS	Instruction Set Simulator.
KVM	Kernel Virtual Machine.
OS	Operating System.
OSCI	Open SystemC Initiative.
OVPsim	Open Virtual Platform simulator.
PTZ	Pan Tilt Zoom.
QEMU	Quick EMUlator.

RTL	Register-Transfer Level.
SICS	Swedish Institute of Computer Science.
TAP	Software defined network interface.
TLM	Transaction-Level Modeling.
UART	Universal Asynchronous Receiver/Transmitter.
VM	Virtual Machine.

Chapter 1

Introduction

This master thesis has been carried out at Axis communications, a company that develops both hardware and software for security cameras. At the time of writing this report they are in the beginning of a hardware platform change. The new platform exists but it is not yet used in any products. The hardware and core platform teams have an emulated model of the new platform. This emulator is not yet complete and it is only used to test basic functionality. However, lately other departments at Axis have caught interest of this new way of software development and want to know more.

Previous platform transitions have usually taken more time than expected, mostly due to delays in hardware design and production. When hardware is delayed, software is as well since most parts of software development and testing depends on hardware. This transition is one reason for the interest in hardware emulation, to see how it affects the process and if it can reduce the overall development time. If hardware emulation can ease the platform transition process it is also possible that it could be a tool in daily development as well.

This thesis discusses hardware emulation in software development for embedded systems. The goal behind hardware emulation is to construct a software model of the hardware, sometimes before it is completed in silicon, to test software functionality. This can be done in a numerous of ways with different methods and techniques. Some are whole systems designed for the purpose of emulating hardware, with FPGAs and complementary software. These kind of solutions are usually commercial products. Other methods are software based and open-source, but when the emulator is pure software, it has certain limitations. A few of these methods and their limitations will be discussed based on information later in the report and one will be chosen and evaluated in the context of Axis' platform transition and daily software development.

1.1 Problem definition

The challenges of embedded software development are many and this thesis will address some of them. The first challenge is the gap between hardware and software development. Today this is both inefficient and time consuming, since hardware usually needs to be completed before any software can be developed. In this case it is difficult to adjust the completed hardware, which makes it an issue to develop hardware and software that work perfectly together. For Axis, this is true in terms of both new hardware platforms and peripherals. This is where the work in this thesis and hardware emulation comes in; to investigate if it is possible to start the software development earlier and target an emulated platform. A way to start the software development and testing earlier would hopefully yield better systems and shorter overall development time.

This brings the topic on to the next challenge, which is daily software development and testing. Today developers at Axis work locally against multiple cameras and remote with larger tests in camera labs. This is neither cost, nor space efficient, especially when a lot of the cameras only differ in software configurations. This thesis will also investigate how hardware emulation can be used in the software development process and the differences between a virtual and a physical camera.

Hence, this thesis aims to explore how hardware platform emulation affects software development in terms of platform transition, daily development and testing.

1.2 Contributions

This thesis extends both the practical and theoretical knowledge about how hardware emulation can be used as a tool in software development for embedded systems. The work done also yields knowledge about the effect of substituting real cameras for emulations in daily development. As well as factors outside of software development, such as economical.

The work will also give knowledge about how open-source software can be used, in this case as a powerful emulator on which full systems can be tested. One system is partially developed to include an emulator, communication over serial port and external devices. This will give an indication of how emulators can be used together with other software and hardware when developing for new hardware platforms.

The work in this thesis was carried out by two students, thereby much work was done in parallel. Here are the individual contributions, both during the work and the report writing.

- Background research was a team effort, all articles were discussed before included in the thesis.
- Simon did most of the QEMU setup with network and serial ports and wrote associated parts in the report.
- Adam investigated Axis camera driver and communication protocol. He also wrote a demo for serial port communication and associated parts in the report.
- Simon extended the serial port demo and wrote the code for the camera block simulation and driver, as well as the respective parts in the report.

- Adam investigated Axis camera setup, configuration and behavior on the emulated platform.
- Simon extended the QEMU and network setup to support multiple instances of QEMU.
- Adam experimented with Axis nightly build test suite on both one and multiple guests and wrote the associated parts in the report.
- Most of the discussion section is based on discussion between the students and therefore a team effort, Simon had focus on the development process and Adam on the tests and their results.

1.3 Thesis outline

As an overview of the thesis and its core components this section gives an outline of the report. The chapters of this thesis are:

- **Background** gives a more in depth background of the subject and its context. This chapter define words and expressions which will be used later in the report. An introduction to the different emulation methods is given as well as a motivation for the most appropriate one. The chapter ends with related work and an interview with an Axis employee.
- **Approach** describes the work carried out. This includes set up of the emulation environment, implementation of a simulated camera block and small driver. As well as benchmarks to compare the emulated platform and a real camera.
- **Evaluation** describes the results from the experiments described in the approach chapter. The results are thereafter discussed together with the experience of doing the thesis work.
- **Conclusions** contains the final conclusions of the work done and proposes possible future work.

Chapter 2
Background

This chapter is based on a literature study and gives background and presents related work for the thesis. Some definitions of important terms are included to give the reader some initial understanding of the area. To enable the work in the thesis, several emulation methods and tools are investigated and one is chosen to be used. To get an overview of how the chosen method can be integrated in the development process, different development methods are presented. Then software development and testing for embedded systems are covered as well as related work with emulators. An interview with an Axis employee is included to enable discussion about how the emulated platform fits in to Axis current development process.

2.1 Emulation and Simulation

Emulation and simulation are two central, but easy to mix up, terms in this thesis. This section will try to clarify their meaning and context.

The dictionary definition of simulation is ”the technique of representing the real world by a computer program” [1]. Simulation will be used when talking about and implementing peripherals for the camera model. Simulated peripherals will not be able to replace the real world components, but a representation that functions as development model. Emulation, on the other hand, is a more complex version of simulation. In emulation one tries to mimic the real world and creating a model which could replace a real product. In terms of this thesis the mimicked representation of the real world will be Axis new camera platform. The emulation will run software as a real camera with the same platform would do. It is not an emulation of a full camera; the system used throughout this thesis is a mix of emulating the camera platform and simulating peripherals.

Virtualization is the software technology which allows emulated models to run [2]. Emulation is a direct consequence of virtualization and the two will be mentioned together in several of the proposed methods later in the report. Virtualization of a complete system with an Operating System (OS) is often called a Virtual Machine (VM). An example of this is when running a Linux OS with Linux applications on a computer that runs Windows. As mentioned earlier the goal with the camera platform emulation is to be able to run camera specific software in its real environment.

Virtualization of the emulated model is divided in two parts: host and guest. The host is the system on which the emulation is run. In this case the host will be a desktop computer running Debian Jessie as OS [3]. From the host one can change the settings and configurations of the emulation. A host, as depicted in Figure 2.1, can run several guests which gives the benefit to run several emulations on one physical machine. This will be further explored in terms of using a server to run tests on, instead of several cameras.

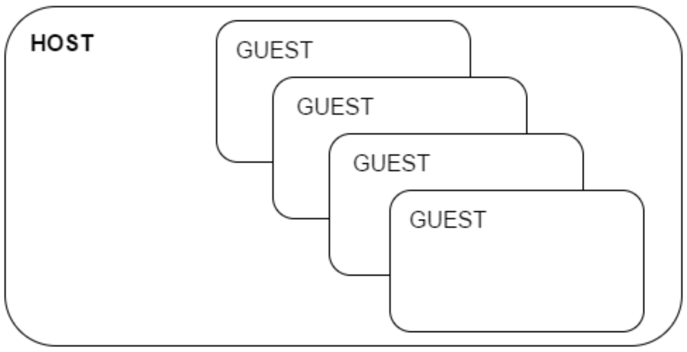

Figure 2.1: One host with multiple guests.

The guest is the system running on the host, i.e. the emulated platform or VM. As a guest the emulation is using the host hardware and has virtual network and serial port connections which makes it possible to communicate with the host, or reach the Internet if wanted. In this case the guest is running an embedded version of Linux, custom designed by Axis for their cameras. A system like this can be very useful, since it is easy to control, change and restrict the environment the VM runs in as well as change the VM itself.

2.2 Camera platform

Axis develops both hardware and software that together make a networked security camera. The emulated part in this thesis is the underlying chip architecture which will be referred to as the camera platform, the gray box in Figure 2.2. The camera platform is an essential part of the system, trying to mimic a complete camera. Simulated peripherals can be connected to the emulated platform and used as if they where connected to a real platform. As previously mentioned, a custom embedded Linux OS is used on the cameras. The OS is configurable to fit different camera models and peripheral setups.

One of the challenges in this thesis is that this is a new platform. Hence, there are no cameras with this setup to use for comparison or as reference, nor is there any software developed for this platform. This is a great opportunity to investigate how an emulated platform can affect the transition process between hardware platforms. Thereby the whole process of this thesis will be considered in the evaluation as it is a part of the result.

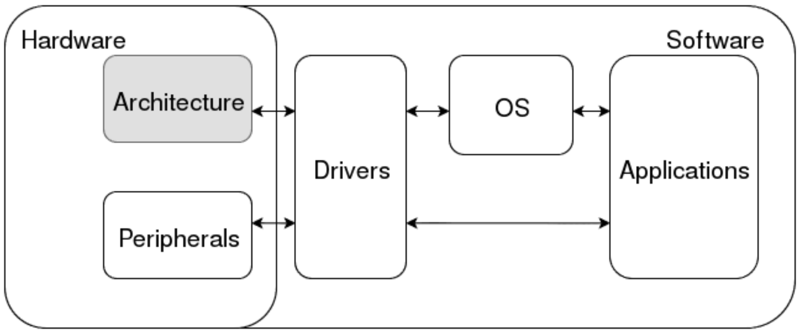

Figure 2.2: General system overview, gray box is the emulated platform.

2.3 Emulation methods and tools

The emulation environment for this thesis is chosen based on the problem area and Axis' preferences. The method of choice needs to support both the guest and host platform. An open-source method with open documentation and an active community would be preferable. A method that is already used inside the company would be a bonus and a good source of information. The method should at least be used in the industry. Lastly the method needs to be extendable in some way with custom made peripherals, to support hardware exploration.

SystemC

SystemC is an open system design library implemented in C++ [4]. Both hardware and software can be designed, tested and verified using SystemC. It was developed by Open SystemC Initiative (OSCI), which was a cooperation between industry leading companies. OSCI is now a part of and supported by Accellera, an IEEE partner and standardization organization. SystemC can, for example be used to implement an Instruction Set Simulator (ISS) which is a high level behavioral simulator. The SystemC library supports Transaction-Level Modeling (TLM) and Analog/Mixed-Signal (AMS) modeling of systems and is mostly used in the industry for architectural overview or functional testing of peripherals [4]. For example, Axis ASIC team uses SystemC in their development process before implementing IPs in Verilog. It is possible, but rather slow, to simulate a full architecture and be able to boot an OS. It is also possible to include SystemC peripherals in other popular emulation methods such as QEMU [5].

QEMU

Quick EMUlator (QEMU) is a hypervisor which performs hardware virtualization [6]. QEMU is an open-source project originally started by Fabrice Bellard. The most common architectures are available for emulation by QEMU including ARM, x86 and PowerPC. More specifically, a guest machine with an alternative architecture and OS can be run on the host machine. Several VMs and virtualizers are based on QEMU, for example VirtualBox, Xen and KVM.

QEMU supports two modes, one full system emulator and one user mode emulator. The full system emulator mode includes emulation of processor and peripherals. The other mode, user mode, enables execution of programs compiled for one CPU architecture on another CPU [7].

Other essential features of QEMU which makes it a portable and reasonable fast emulator are dynamic translation, condition code optimization and CPU state optimization. The dynamic translator used in QEMU is modified in a way which makes it easily portable and thereby makes the emulator fast for several different architectures. When executing a program in the emulator, the assembly code is translated to pre-compiled C functions, called *micro-operations*. These micro-operations are put together in to blocks, which are chained together to be executed on the host machine [8]. For condition code optimization QEMU uses lazy evaluation to only update condition code when it is necessary. However, this works different depending on architecture. X86 uses this kind of lazy condition code evaluation while ARM uses a different method [7].

Imperas - OVPsim and DEV

Open Virtual Platform simulator (OVPsim) was developed by Imperas between 2005 and 2008, with continued support. OVPsim is free for non-commercial projects and a lot of open-source models are available. OVPsim provides C, C++ and SystemC APIs for system and peripheral development. Imperas also offers a commercial platform development tool, DEV, that still supports open-source OVPsim models and offers software analysis and verification, better simulation performance and multi-processor systems with shared memory [9].

Imperas collaborate with universities and several companies in the hardware and software co-design business. Some examples are Cadence (which will be mentioned later in the report) and MIPS technologies (known for their MIPS CPU architecture). OVPsims open nature made it an interesting emulation tool for this thesis.

Wind river - Simics

Simics is a full system simulator originally developed by the Swedish Institute of Computer Science (SICS). For commercial use the project later created the company Virtutech, which today has been acquired by Intel and the sister company Wind River. Simics supports a wide range of guest architectures such as ARM, x86, Power and MIPS. It also supports simulation from small chips to full systems and has built in debugging tools [10].

Simics has features which enables the guest software to be run exactly as it should on physical hardware. One of the main benefits of Simics system simulator is the debugging part. It is possible to freeze, save and restore a complete system, which is naturally not possible on physical hardware. The limitations hardware and software development have are much less limited with a emulation system such as this [11].

It was proposed by Axis to have Simics in the list of possible emulation tools. However it turns out that it is not a perfect fit for this thesis. Mostly due to that it is a commercial product that is not already used in the company.

Cadence - System Development Suite

Cadence System Development Suite have two interesting parts, the Virtual System Platform and Palladium. The Virtual System Platform focuses on early software development and generates virtual hardware prototypes from functional specification. Palladium is a complete computing platform which supports both simulation and hardware acceleration of custom IPs. Both products can handle popular platforms such as x86 and ARM and support custom IPs. These are commercial products with individual pricing depending on project size [12].

The Virtual System Platform supports the previously mentioned SystemC and Imperas processor models among others. Another feature is the graphical user interface included in the platform which gives a overview of the currently running simulations [13]. This kind of program makes the debugging process easier. However, the main drawback of the Cadence products is that they are commercial, which makes it difficult to find information on the internals of the system. Palladium is sold as a complete server solution with FPGAs and accelerators, which could be useful when working with complex emulations, but is out of the scope of this thesis.

Motivation for QEMU

The chosen emulation environment in this project is QEMU full system mode. QEMU supports many different host and guest platforms that cover the needs for this project. There are standard ways to add peripherals and implement communication protocols in the emulator, which also satisfy the needs for this project. The active open-source community around QEMU, both the development on GitHub and the wiki on QEMU home page, is also an important part of the decision. It is widely used by several well known companies and organizations. For example University of California, Berkeley, uses QEMU to emulate their RISC-V architecture [14]. There is also knowledge about QEMU inside Axis which makes it a natural way to continue development. For example TMLu, a QEMU extension, is used by Axis to explore SystemC IPs in an emulated environment [15]. On the other hand, QEMU has rather poor official documentation.

On the commercial side, both *Wind River* and *Cadance* offer industry leading simulation systems. A negative point with these systems is the rather steep learning curve and it is recommended to take a course at the company to familiarize with the tools and documentation, which is out of the scope of this thesis. *Imperas* alternatives are not as interesting, both because the rather big performance difference between the free and *DEV* solutions and because it is not as widely used as QEMU.

SystemC is rather slow when simulating an entire system. It is widely used for smaller component simulation, but is losing ground to QEMU and OVPsim for more complex simulations. It is also possible to integrate SystemC modules in other simulations, such as RTL or TLM modeling with QEMU. Cucchetto, Lonardi and Pravadelli conclude that QEMU simulations can be up to four times faster than OVPsim [16].

2.4 Development methods

Development of hardware and software for embedded systems has some parts in common with traditional development in both areas, but it is not possible to look at it as an extension of either. Design and development of embedded systems adds both freedom and complexity. Freedom to decide what is going to be implemented in hardware respectively in software and complexity when designing with the constraints that hardware and software put on each other.

Simulation is often used in hardware development. Here SystemC is a popular choice for its implementation accuracy. As previously mentioned, SystemC is rather slow when simulating complete or larger systems. This is a problem when introducing hardware emulation in software development. This is further discussed along with proposal of a standard way to include SystemC models in QEMU and OVPsim, to be able handle legacy SystemC code [16].

In embedded system development there are two methods in different ends of the abstraction spectra, bottom-up and top-down [17]. Bottom-up method consists of developing components at each abstraction level, starting at the bottom. Components at the bottom layer are then used to create higher level components, yielding a complete system. Top-down on the other hand starts with a high level specification and from this specification smaller parts are specified and implemented. This iterates until a complete system is implemented. Both methods have its strengths and drawbacks. In bottom-up the developer have control over system metrics in each layer. Top-down might allow for an earlier start of software development, targeting the initial specification.

In practice neither of these extremes are used, instead a meet-in-the-middle method is used [17]. Here top-down is applied to high abstraction levels and bottom-up to lower, to get the most out of both methods. There are a couple of different ways of how to meet in the middle and how many levels each method should be applied to. Axis is using a version of meet-in-the-middle. Their hardware designs are based on a complete CPU architecture where different peripherals are implemented and added. A drawback with this approach is that it might be hard to optimize on a system level.

2.5 Software development

To be able to develop effective software for embedded systems it is important to know and understand the constraints hardware puts on the system. A big challenge here is the gap between hardware and software development, as the software development can usually not be started and especially not tested until hardware is available. More hardware constraints can lead to extra costs and development time which, together with some software development methods, are discussed further by Gracis, Pacheco and Herrera [18].

Another constraint is the system resources available for the software (memory, bandwidth, cores, I/O etc.). This kind of software development is different than general purpose systems (normal desktops and laptops) where there are standard ways for interactions, such as keyboard, mouse and screen. A general purpose system usually has a lot of computational power and memory, much more than one single program utilize.

According to Andreas Olsson, software developer at Axis, the development process depends on many different factors [19]. First, if the hardware is developed inside the company, the software development often only involves updating drivers or protocols from last iteration. If it is new hardware, there is usually a cooperation between hardware and software engineers to write efficient drivers and APIs. If the hardware comes from an external source, it differs, especially if it is bought from a competitor. All information are given through specifications on papers and in manuals. Thereafter the developer might get a development or demo version of the hardware to explore. These development boards are connected to a PC instead of a camera. In this case much communication between the developer and the manufacturer is required to interpret the information given and work around limitations. Some information is shown to only be theoretical and does not hold up in real world applications.

In preparations for new hardware, a few things can be done in terms of software development. To start with, prototypes and stubs using old hardware and similar platforms can be implemented. Even though this helps with the initial development on new hardware, several details can not be implemented until the hardware arrives. Sometimes specifications change, such as physical interfaces which can cause problems in the pre-hardware stage.

Waiting for hardware is the largest issue when it comes to embedded software development and it puts a lot of pressure on the software developers when it arrives. To change the process and let off some pressure from the developers, an emulated platform could help in several ways. According to Andreas Olsson, one of the most important benefits would be connection pass-through on the development PC. Thereby one could utilize the connection ports on the computer and connect the previously mentioned development boards to the emulated platform via the computer. Even though an emulator is not a full representation of a functional system it could still be beneficial. For example an emulator could be a way for initial testing and to find limitations before the physical hardware exists.

There is no general development method at Axis. The development is carried out in small teams which each have their own processes and methods. The method depends on the project and the developers.

2.6 Software testing

Axis is using an internal testing framework, Automated Testing Framework (ATF), developed in Python. This is used to test software and camera functionality, either in large camera labs or locally on a camera on the developers desk.

In software testing for embedded systems it is usually a difference between host and target testing. Logic of the software can be tested on any machine and are therefore usually tested on the host. Some features are environment dependent and behaves differently on the host and the target, then target testing is required. Testing is also split by unit and integration tests. Unit tests focus on stand alone parts of the code. Integration tests requires that the unit tests passes and then test the units together, integrated in the system [20]. To test embedded system, hardware for a testing environment needs to be acquired, set up and maintained. This makes it a lot more expensive to test software for embedded systems, than general purpose. This is an issue at Axis, where camera labs takes a lot of space and requires maintenance and infrastructure to work.

GNU Debugger (GDB) is a popular tool to inspect and debug software. It is used both internally at Axis and by the Unix community. GDB is, for example, used in Linux kernel development, where it can be connected to QEMU for debugging [21].

2.7 Emulation in previous work

Sampath and Rao conclude in a paper that it is possible to make an efficient development and testing platform for embedded systems using emulation [22]. The paper focuses much on the process of the development setup: which emulator/simulator to use, which development environment to be used and so on. QEMU is chosen as the most fitting emulation method for their system. Even though the system differs (both in terms of software and hardware) from the system in this thesis, several of the points mentioned about QEMU and other systems was considered in the choosing of QEMU for this thesis. Based on that it is possible to make a development platform, this thesis investigates how it affects the software development process for embedded systems at Axis.

A paper about testing embedded networked systems with an emulator, concludes that emulation has great potential in embedded system design and development [23]. In this paper an alarm system used to detect fire and notify its surroundings is discussed. To verify the robustness of the system an emulator is created. As the system is very large and wide spread it is more convenient to emulate the system to test it. Axis' system is also Linux based, embedded and networked, as in network surveillance cameras. Similar to the Axis system which is used in this thesis, the system in the paper is divided in two parts: one part that is emulated and one that is simulated. Between the two a serial connection is used for communication, though this thesis does not focus on the network and infrastructure parts as much as the paper does. However, the paper offers a good background for the possibilities to test software against virtual platforms.

A book by Boulé and Zilic brings up hardware emulation as a tool to dynamically test hardware simultaneously as developing it, which also enables software development [24]. The book also brings up the benefits of an emulator in comparison to simulation, an emulator is closer to a real implementation and can handle more complex systems efficiently. An emulator is more representative of a real system than a simulator as it tries to mimic the functionality of the real system instead of model it. The base in this thesis is built on hardware emulation and the book brings up how it can be used for system verification on larger systems. However, no hardware was developed during this thesis work.

The ability to combine QEMU together with other systems such as SystemC is discussed in several papers. SystemC will not be used in this thesis, however it may represent the use of peripherals in other ways. In another paper the combination between QEMU and SystemC is brought up to discuss the possibilities to combine the two to model a full system [25]. One of the main issues with SystemC is that it is unsuitable to run an OS. However, as a full system emulator QEMU is able to work around this limitation. Two different systems are tested in the paper, one running Intel x86 platform and one running ARM versatile platform. Together with different peripherals the system is evaluated, primarily in terms of speed. The emulation is fast enough to run a standard OS together with the peripherals, both simulated and real. This thesis will run simulations against an emulated platform in a similar fashion and this paper is a good indication that it is possible.

However, the emulation in this thesis will not run at native speed, that is the same speed as a real system, which will be taken in to consideration during the evaluation.

2.8 Chapter summary

The main focus of the chapter has been work that already been done within the area. Development processes both within Axis and elsewhere have been brought up, as well as how emulation could fit in to these processes. Different emulation methods have been mentioned, both open-source and commercial systems. After some research QEMU is the method further used and discussed in this report as it fits the purpose of the thesis best. According to Axis employee Andreas Olsson, the main issue is the wait for hardware to arrive, which emulation may be a solution to.

Chapter 3
Approach

As stated in the problem definition, this thesis aims to investigate how hardware platform emulation affects software development. This chapter describes the methods to evaluate how emulation fits in to Axis development process in terms of platform transition, daily development and testing.

A method is needed to evaluate the potential benefits of an emulator. In this case (picked out and motivated in the previous chapter) QEMU is chosen to be the emulation tool. QEMU is used to run the Axis made OS in a emulated environment and to run software developed for previous cameras. The main issue here is compatibility between the new hardware and the old software. To make these two work together with full functionality is beyond the time frame for this thesis. However some basic functionality will be implemented to run some tests on. The setup consists of a main driver, a communication protocol and external devices and will be described in more detail later in this chapter.

When the basics are in a functional state, they will be used to implement functionality similar to a real system. The network connection will be used to deploy tests on the emulator, which are also run on a real camera. The serial ports will communicate with a simulation of a camera block, which will be able to answer simple queries.

3.1 Emulation environment setup

Axis already has a model of their new chip implemented in QEMU. This implementation covers basic platform functionality and they are able to boot their Linux based camera OS, but lacks camera specific functionality. There are a few things needed to be set up before the QEMU model can be used for software development. To start with, some basic functionality needs to be working, such as network and serial port connection. These functions are essential for the emulator to function similar to a real camera and a typical challenge a developer is faced with when a new platform is available.

Network and serial communication between the host and guest is needed, which is set

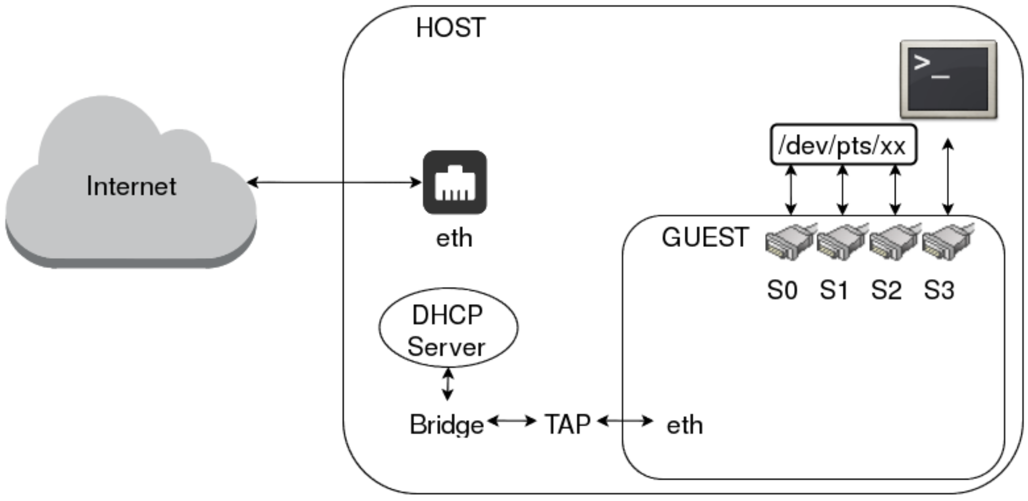

Figure 3.1: Guest and host setup with network and serial ports.

up as displayed in Figure 3.1. Technical details on the setup can be found in Appendix A. The network is needed to send programs and files to the guest and to receive results from tests. The camera has a built in web server for access to camera settings and live view. Access to this interface via the hosts web browser is verification that the network connection is up and running. A serial connection between the host and the guest is needed for connecting the camera console to the host and external peripherals to the guest. This part of the set up is verified by a simple demo implemented in C.

Once the emulation platform is up and running, the next challenge is to configure it with peripherals to resemble an existing camera. Since there is no camera using the new platform yet and the emulated platform lacks certain functionality, this configuration may not be an easy task. To port the OS with configuration possibilities and drivers to the new platform is a work in progress at other teams at Axis and out of scope for this thesis. Even if the configuration options work, the functionality of the emulated platform is still limited and might not work in this context. The configuration possibilities are investigated and as, a back up, a small stand alone driver for peripherals is also implemented.

3.2 Simulated camera block

This section covers technical background and implementation of a simulated camera block and complementary driver. This part of the work is done as background for discussion about development targeting the emulated platform.

3.2.1 Technical background

To understand the implementation details, some background information about Axis system is needed. The system to be set up is based on three different parts: the camera driver, external devices and the communication protocol which ties the other two together. This system is used in the real cameras and is made as a combination of emulation and simulation to emulate the camera and simulate the external devices.

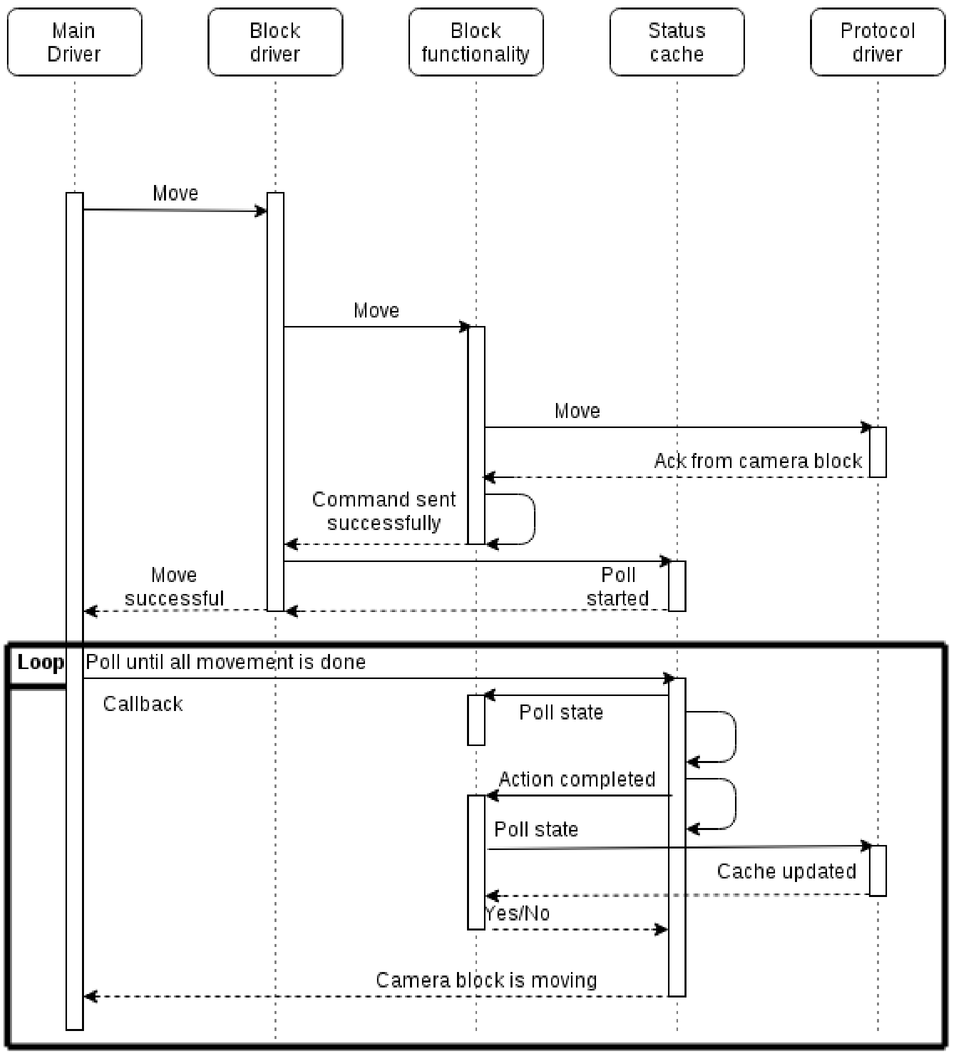

Figure 3.2: Sequence diagram of a move command, actions happen from top to bottom.

Camera driver

Axis Pan Tilt Zoom (PTZ) cameras use a driver which controls external PTZ devices, such as camera blocks. This driver is not yet fully supported on the emulated chip. As an experiment in this thesis a small part of the driver will be implemented. It will only support the zoom command (in Pan Tilt Zoom). The driver in a real camera supports several other external devices and commands.

The camera block driver controls the communication protocol which handles the commands sent over a serial port between the driver and external devices. Figure 3.2 shows a sequence diagram for the driver when it performs a move command. This is internally of the drivers running on the camera and in the camera block or for this thesis, QEMU and camera block simulation. More detailed information on how the communication protocol works will be described later.

The driver uses a state machine that utilizes polling to control the state of the command executed by the camera. This state machine checks all commands before they go through

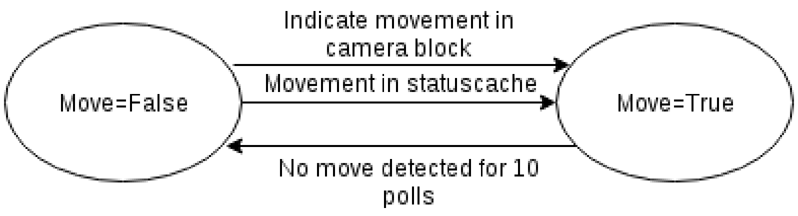
Indicate movement in camera block
Movement in statuscache
Move=False
Move=True
No move detected for 10 polls

Figure 3.3: Overview of the state machine when a move command is in action.

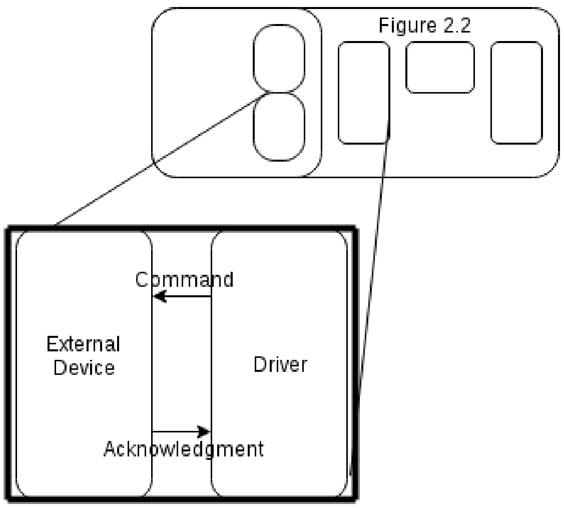

Figure 3.4: Zoom in on Figure 2.2 focusing on the driver, protocol and external device.

to the driver and holds a command until it is finished, terminated or a certain time has elapsed and nothing occurs. Figure 3.3 shows the movement state for an external device, such as the zoom command on a camera block. The two states for move, true and false, are changed depending on the situation. For example if the state machine notices movement in the camera block the state machine changes it state to true. On the other hand, if no movement is detected for 10 seconds the state is changed to false.

External camera devices

The driver is connected via a serial port to external devices which perform the commands the driver sends. External devices in this case describes peripherals that are inside of the camera housing but not part of the chip, for example a camera block and motors for a PTZ camera. These devices will not be emulated, however one of them will be simulated on the host for the guest to communicate with, to make the model more realistic. This will make it possible to test functionality without attaching hardware to the host computer. All communication between the driver and the external devices uses a communication protocol (described in the next section). As depicted in Figure 3.4, which display a zoom in on Figure 2.2, the protocol communicates by sending a command and receiving an acknowledgment between the driver and the external device.

Communication protocol

To communicate between the driver system and the peripherals, a protocol is needed. In this case the protocol used by Axis for one specific camera block is based on a third party protocol. The protocol is rather simple, with the common system of sending a command and getting an acknowledgment back. That is, each time a command is sent, the recipient sends back an acknowledgment to let the sender know it received the message, the basic procedure is seen in Figure 3.4. The protocol supports up to seven devices and can communicate both to specific devices and broadcast commands. There are a number of commands that can be sent to the devices depending on which functionality they support. Responses are fewer, the main ones are acknowledge and complete. Acknowledge is the most used response as it answers the most commands and complete verifies that the device completed the request. The last type of response is error, which lets the sender know that something went wrong. These responses are similar for all commands and do not change depending on the command purpose.

3.2.2 Implementation

This section covers the implementation of a simulated camera block that runs on the host and communicates with the guest over a serial port, as a real camera block does. The implemented camera block only supports zoom functionality. It simulates a real camera block in regards to listening to commands on a serial port and execute some of them. Commands for non implemented functionality get an acknowledge in return, in an attempt not too break to much of the PTZ driver. The camera block is simulated to make a more complete model of a camera and to investigate how software development against the emulated platform works in practice.

The implemented commands are power on and off, some zoom commands and zoom inquiries. The simulation keeps track of the zoom state to be able to respond to inquiries. This is built with a state machine to match the PTZ driver described in earlier sections. Implemented commands in the simulation get an acknowledge as a reply and then an additional reply when the task is completed.

Internally the UNIX standard library *termios* is used to communicate over the serial port together with UNIX system calls for serial ports [26]. This implementation is based on the previously mentioned serial port communication demo. Before the communication can start the ports need to be initiated. This is done by setting up a file descriptor for the port, save old settings and flush the port. Thereafter the host and the guest communicate over the ports with the UNIX calls `read` and `write` [27, 28]. At last, the host answers the guest in the same way and this procedure continues as long as it is needed. The ports are thereafter put to their original state and closed.

A smaller part of the camera block driver is implemented without integration with the rest of Axis PTZ driver. The small driver matches the simulated camera block in implemented functionality and it is an extension of the serial port communication proof-of-concept between the host and guest.

This camera block simulation is an example of how the emulated platform can be used in early software development. One aspect is to develop drivers without the need for physical peripherals. Another is to be able to explore different peripherals before buying or integrating them in to a camera.

3.3 Nightly build test suite

An interesting aspect is the execution time of software run inside the emulator. For a large system, which the Axis camera is, test suites can take several hours to run and verify. Therefore, it is interesting to examine the time these tests take in the emulator and compare to a real system. As previously mentioned in the background, QEMU does not run at the same execution speed as the real hardware. This is to be considered when running the tests, if the loss in performance is acceptable or not.

Axis has a test suite which they run on their products during nights and weekends. These tests check software functionality of the camera depending on product. The suites include tests for network, storage, video, audio and several other aspects of the camera. It is possible to create custom subsets of the tests to focus specific functionality or systems. Axis has tools for picking out subsets of the test suite that only covers supported functionality. This is done on the emulated platform. The current version of QEMU uses a few simulated parts for certain functionality which will not give the same results as a real camera. There might be similar problems when running these tests such as deploying the configuration and limitations in the current emulator.

To avoid these problems, the test selection needs to be manually checked to guarantee that the suite does not crash. Thereafter, this suite can be used to compare the speed and performance of the emulator to a real camera with similar specifications. The result of this comparison is later used to see if the emulator is good enough to run the tests on or if there are too many problems. Some factors to be considered here are speed, host resources required and the amount of errors and failures in the test results.

3.4 Multiple guests on one host

One of the benefits of a emulator is that several instances may be run in parallel on one machine. This can be useful both when developing and to be able to test software on a larger scale on servers instead of real hardware. To investigate the resources needed on the host, several instances of QEMU will be executed and the needed resources measured. In an attempt to isolate the QEMU instances further, they will also be executed in separate instances of VirtualBox. VirtualBox is a VM based on QEMU which is previously mentioned in the description of QEMU. By the introduction of VirtualBox there will be an additional layer of instruction translation before QEMU will share hardware. Thereby the performance and stability will be investigated to see how the additional layer of virtualization affect the QEMU instances.

Today Axis has big labs with cameras to run daily tests on different hardware and software configurations. Software developers in the PTZ team often have multiple cameras set up in their offices, also to be able to test different configurations. The retail price for the camera used as a reference in the tests is around 11,000 Swedish kronor, even though the cost may not be the same for Axis internally, it will accumulate. The cost of the labs themselves is also considered. These costs increase when deploying new hardware and at the same time keeping backwards compatibility. The emulated platform could help to reduce cost and ease the test process on different configurations.

The nightly build test suite used for comparison between QEMU and a real camera, will also be used to test several guests. Thereby it will be easier to notice any differences between the runs and how having multiple guests on one host affects performance.

3.5 Chapter summary

This chapter has given an overview of the system which has been investigated and implemented to create knowledge of how an emulator can affect software development. The system in question was described in more depth to provide relevant information to understand where the emulator can fit in and how it can be tested. The flexibility of an emulator was also described and how it can be used for further investigation. At last, a description of how an emulator can be used to test software on a larger scale on servers instead of their specific hardware, in this case a physical camera. In the next chapter this will be evaluated and discussed to see what is possible and potentially useful.

Chapter 4
Evaluation

This chapter presents and discusses the results and experiences from doing the work in the approach chapter. First the results from the test suits are presented as a base for discussion. Thereafter our experience from working with the emulated platform in different ways are discussed, in a context of Axis development process.

4.1 Testing on the emulated platform

The results presented come from benchmarks created by running Axis test suites on different setups of QEMU and a real camera. An older version of the operating system for the camera is used on the emulated platform, as it has been used throughout the thesis. This is done to get consistency between the test runs and to be able to compare errors. A more recent version of the operating system may give different results.

4.1.1 Nightly build test results

Raw data from running tests picked out to run on QEMU can be seen in Appendix B. The emulator is given one gigabyte worth of memory and utilizes one CPU core. The same tests have been run on a camera which uses the previous generation of both hardware and software. More specifically a P5515, which is a PTZ camera currently in production. Focus lies on speed and number of passing test cases. For these tests a failure involves tests which complete their task but in a wrong way. An error describes all tests which are unable to complete their task.

4.1.2 Nightly build tests

From the theory previously mentioned in the report, QEMU does not run at native speed and the performance loss differs depending on architecture and application. During the

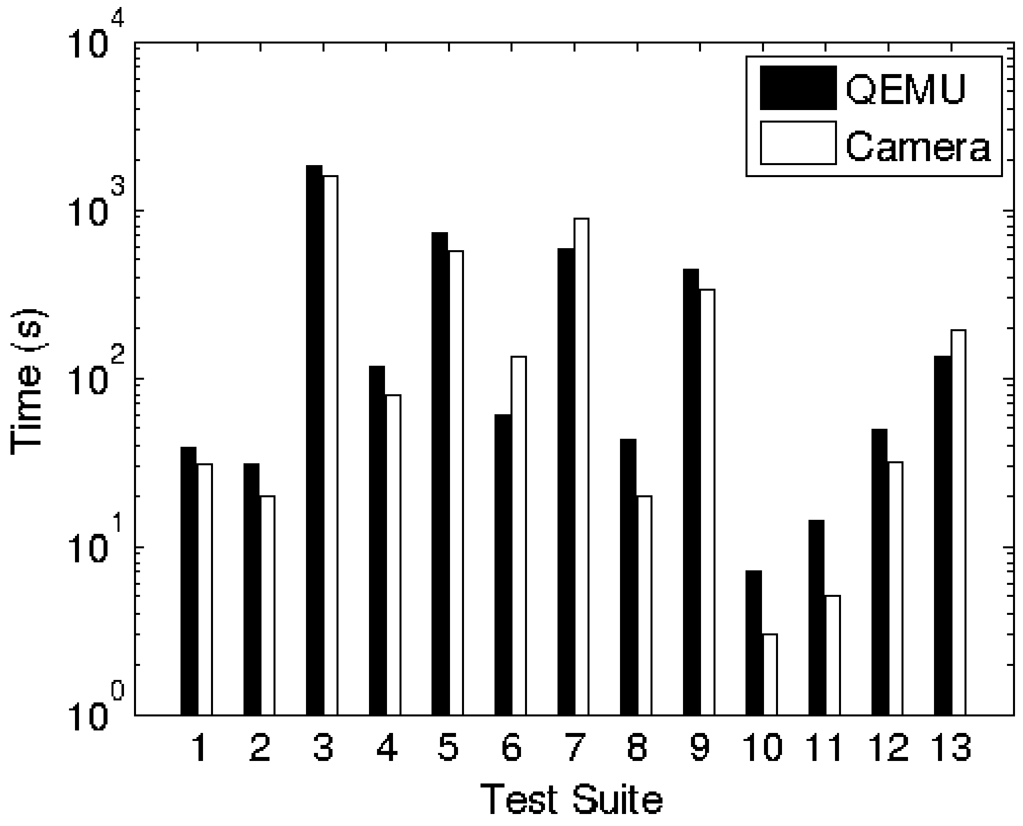

Figure 4.1: The time for the different test suites running in QEMU and on a real camera.

Axis nightly build tests this was confirmed to be true. As seen in Figure 4.1 the different test suites had rather similar running time on the two platforms. When summarizing the time from all the test suites we have that the P5515 camera ran the test suite of 1130 test cases in 3885 seconds compared with the emulated platform which did 1165 test cases in 4114 seconds. This is a very good result as the time comparison differs by just a few hundred seconds which indicates emulation might be a good way to test software. However, when trying to separate the QEMU instances with VirtualBox the time increases by a large margin, up to 5781 seconds. This is mainly due to the overhead required to run the tests in another layer of emulation because of additional instruction translation and network delay. Therefore it is a bit harder to justify the additional VM for one instance of QEMU.

Another important detail to be considered is the error and failure ratio of the two. When running the tests in QEMU the success rate lies around 88% while running the same test cases on the P5515 the success rate was 94%. In VirtualBox the amount of errors were even higher then just running on QEMU. In other words, the emulated platform received a higher amount of both failures and errors. When an error occurs, a common idea is to fail fast to lessen the impact of the error. There are, however, also tests which may take time to recover from errors and thereby take longer time than they should. Thereby the exact time of the runs on the camera and in QEMU may be different if the amount of failures and errors were more similar. Important to point out is that the real camera is modified to behave similar to the emulator. For example the camera does not have a SD-card inserted because this emulation setup does not support it. Hence a similar amount of errors during the storage tests as the emulated platform.

For larger test suites, the execution time might start to be more noticeable as the delays from the emulation start to accumulate. This subset is just a part of the total amount of tests in the Axis nightly build. For a more developed version of QEMU which supports the functionality of a full camera, the time differences may be much more crucial. However, as long as the run time does not increase tremendously for larger suites, emulators on a server may be a good way to test software in reduced physical space. When running in VirtualBox, the time is increased for a small subset and will probably increase further for a larger set. As long as there are no problems without VirtualBox it does not bring any benefits when running a single QEMU instance. Mechanical functionality is another detail which an emulator may have problem testing. The P5515 for example is a PTZ camera which has motors for pan and tilt movements. Test suites that tests this functionality may cause problems if a fully server based testing solution should be considered. However, the main reason to run these tests is to find software errors and not mechanical. Thereby it might be solvable to simulate this functionality similar to the work done in this thesis, with a simulated external device which will answer the queries from the main driver.

The reason behind the differences in amount of errors and failures is both due to the current OS implementation and the hardware differences. At the moment the new OS for this platform does not support all functionality that a real camera does. Even though the subset of tests was picked out to be able to run in both QEMU and on the P5515 there were some limitations. This concluded in the current amount of errors and failures seen in Figure 4.2. As the development of the QEMU platform and software porting projects continues the amount of passing test cases could be increased to a similar level to the P5515. When it comes to the difference in test amount, this was automatically changed by the camera during the tests, as the same suite was executed on all the platforms. Probably some of the tests compatible on the emulated version of the camera was not compatible on the P5515 as they run different hardware and software.

The hardware differences are also to be considered when comparing the two platforms. A newer platform is emulated in QEMU than the one P5515 uses. The tests included in the nightly build test suites are designed with the older hardware in mind. Therefore it is a good sign the tests run on the emulator, which indicates they can be used early on in the software development process. When starting software development for new hardware these tests can easily find functionality which do not work but should. From here it is possible for the developers to quickly get the new system up to par with the old system.

Running the tests in an emulated environment brings other factors to the table when it comes to error sources. Other processes currently running on the same machine have a negative impact on the execution time of the test suite. When running the same tests again the result does not change, but a difference in execution time was noticeable. The only difference was other processes running in the background on the same machine. To investigate this further several instances of QEMU will be executed simultaneously to see how many may be optimal to run on the same machine without too much of performance loss. This experiment will be further discussed in the next section.

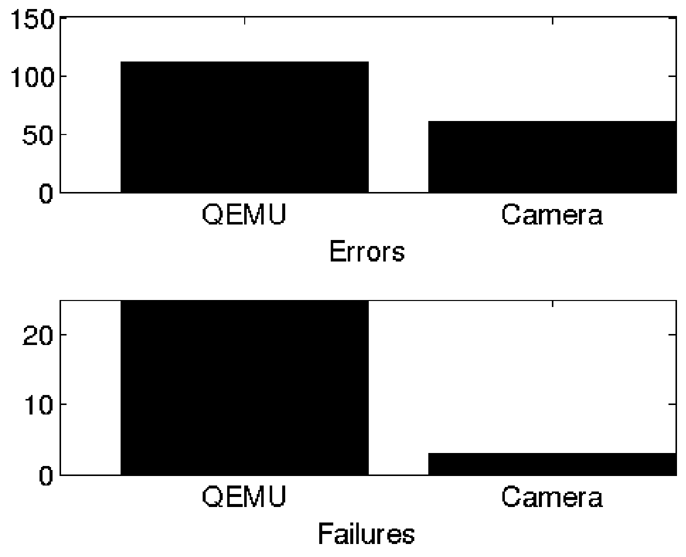

Figure 4.2: The number of errors and failures when running in QEMU and on a real camera.

4.1.3 Multiple guests on one host results

Raw data from running several instances of QEMU on one desktop machine, with an Intel i7 4770 @ 3.40GHz and 16 GB of RAM, are listed in Appendix B. The results also include the performance differences for the emulators and the speed of the test suites. Each instance is given one gigabyte worth of memory and one CPU core. The VM is running Ubuntu 16.04 LTS 64-bit with 3 gigabytes of memory and two CPU cores. The same test suite picked out for the single instance run is used for several instances. Focus here lies on the time differences between the platforms.

4.1.4 Multiple guests on one host

Each of the emulator instances occupies one core on the CPU, which would theoretically allow 8 instances to run as the i7 have 8 logical cores. For an even more powerful machine it may be possible to run additional instances. From here we can start to discuss emulators from a economical perspective. As previously mentioned the retail price of a P5515 is approximately 11,000 Swedish kronor. Of course there is a large range of models and cameras and this is just an example. A test server can be bought for approximately 2,480 USD which corresponds to 20,050 Swedish kronor in current stability. In terms of how many more instances of cameras can be run on one of these test servers compared to the current camera labs the economical savings can be noticeable. However for this to be reality, several improvements need to be implemented first.

Running several guests on one host entails new issues. One of the more crucial ones is stability, as one or several of the instances sometimes stops working and the test run terminates. For larger numbers of guests, the amount of crashes increases. For one and two instances the runs are stable and give the same results from all the runs. However, when running four instances the risk of a crash increases. The reason for this is not clear,

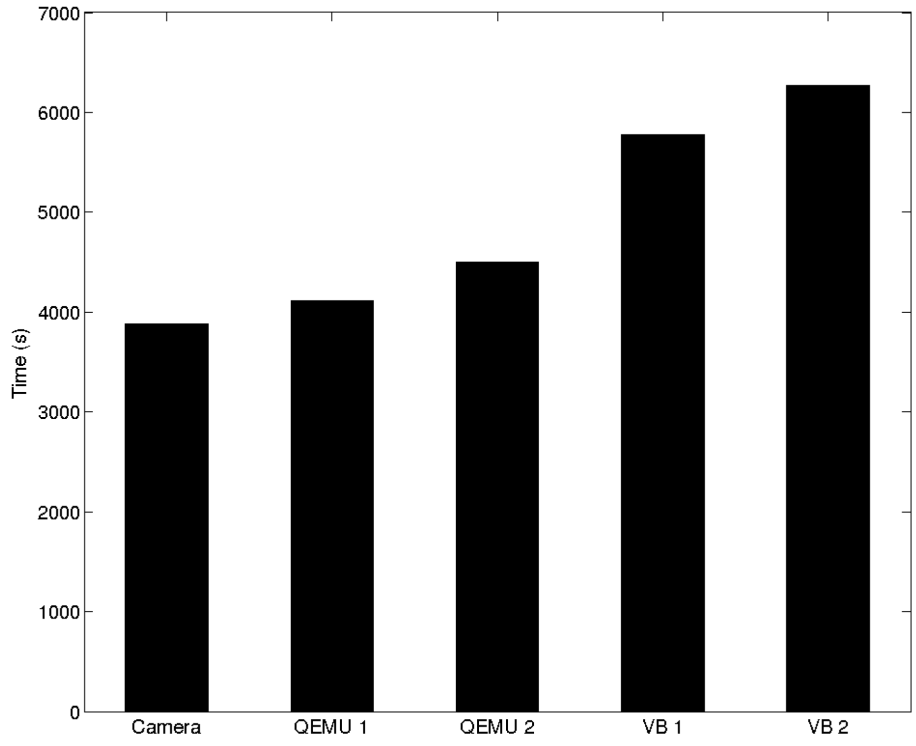

Figure 4.3: The time for each run for the different platforms with one and two instances of QEMU compared with the time for a real camera.

since the hardware on the host should be enough to handle higher amounts of guests, based on that each instance utilize 1 CPU core and 1 GB of RAM without to much overhead. From some quick investigation about the error we concluded that it may occur when the power source of the hardware is overloaded. However, this is for a regular machine, not for an emulated environment which made us discard this conclusion. When testing different OS versions the problem persists, which may indicate errors on a deeper level such as conflicts in shared resources. It may still be a consequence of emulator overhead, as the error message given indicates a processor lockup, even tough the host does not show any signs of being overloaded.

In an attempt to fix the potential issue with shared resources, we installed additional virtual machines, hence the introduction of VirtualBox. The reason behind this is to let each QEMU instance run in a separate VM. Thereby cutting all connections between the different instances to limit conflicts. To run the instances of VirtualBox, more hardware is required per instance to handle overhead, which result in the theoretical number of parallel instances are reduced. As seen in Table 4.3 this theory did not hold as the results do not reach over two instances. For more than two instances of QEMU and VirtualBox, the same error occurred as for running QEMU directly on the host machine, confirming that the error probably is in the QEMU implementation or the embedded OS. Which is entirely possible since the software is in an alpha stage and currently not developed to work in QEMU, but for a real hardware prototype.

Something else worth to note are the differences in amount of failures and errors between one and two instances. This is a bit hard to explain as all the runs in QEMU use the same software and test suites. One theory is the same reason that several instances of QEMU crash; the overhead causes errors for the tests and their timing or from limitations in the software.

The overall experience from working with this part of the thesis is that testing takes time. As seen in Table 4.3 each round of tests takes over an hour to complete. When the crashes do not occur until after nearly an hour of the suite, each change takes a lot of time. The introduction of VirtualBox took an additional few days to implement as all the tools and applications needed to work on a fresh installment.

4.2 Development process

This section discusses the experiences from doing the thesis work. The work has been a smaller version of a real development process. We started with the emulator running the new hardware and nearly no supported software, which we used as a base to develop basic functionality and configure for testing purposes. We discuss how QEMU works as a development tool, based on our experience we have from setting it up and the obstacles we encountered. Thereafter we cover how this tool can fit in the software development process at Axis.

4.2.1 QEMU as a development tool

In this section we discuss the set up and process for QEMU and implementation of peripherals.

QEMU setup

During the set up of the emulation environment we encountered numerous problems, this process gave us a deeper understanding of network in a Linux environment and QEMU. Both QEMU forums and colleagues at Axis where a great help during this process. As previously mentioned, QEMU is an open-source project and it is not uncommon for this kinds of projects to have poor documentation and different ways of doing the same thing. This was shown to be the case here.

Contradicting documentation and guides was the biggest challenge with the network setup. After sorting a few things out and choosing a DHCP server, the network worked as expected. As shown in A.4, it is possible to access the emulated camera's built in web server. This interface works to some extent as it shows camera status and debug log. Live view is only showing a red (black in the figure) hard coded picture, because nothing is implemented to produce a video stream.

It took more work to set up the serial port connection between the guest and the host, once again mostly due to the amount of different ways to set it up in different scenarios. We recommend that Axis standardize a way to set up QEMU, if they choose to continue using it as a development tool. The serial port set up is explained in Appendix A as well as helper functions for serial port communications that where used in the serial port demo.

It was not possible to configure the emulated camera platform the way we planned, because of compatibility issues. The OS version we used in the configuration of the camera did not support all the required functionality. To configure the camera (in this case in QEMU) we tried with an older version of the software and changed it when we encountered problems during the configuration process. However, the dependencies quickly became too complex for us to handle, which required us to put the custom configuration on hold to focus on other areas. The main reason behind this decision was that each configuration attempt nearly took an hour to do and each change made a very small difference. Thereby all tests and experiments are done on a generic configuration specially made to work on the new platform.

Camera block simulation

The implementation of the camera block simulation together with the earlier background is a proof that it is possible to use the emulated camera platform to develop drivers for peripherals. Since there are no products with the new camera platform, the platform is not yet supported by the PTZ driver package. Using this driver to control the simulated peripheral would have been a good test, but unfortunately not possible yet. Instead we implemented a small subset of the driver. The small driver implements parts of the camera block protocol and just like the PTZ driver it keeps the current status of the camera. The simulated camera block also keeps track of its own state and is able to respond to inquires and execute some commands, all according to the protocol. The driver extends the serial port communication demo earlier described in previous implementation chapter.

4.2.2 Emulation in Axis development process

This section discusses how the hardware emulation fits in to Axis current development process. This is discussed in terms of platform transition, new peripherals and testing in daily development work.

Platform transition

The upcoming change of hardware platform means a lot of work for software developers on all levels. In the work for this thesis we experienced some of the challenges a developer may face when transitioning to a new hardware platform. It started with network and serial port set up, which is a rather fundamental low level problem. A lot of smaller fixes and configurations can be made with the QEMU model of the camera platform, such as defining UART ports in the OS. Axis platform team has also used it to port parts of the Linux platform to the new hardware, with good results so far. The emulator is shown to be a great tool for fast feedback on configuration changes and could be useful in other areas as well. The work in this thesis shows that the emulator allows developers to start the transition process to the new platform before the physical hardware is available. This can lead to a shorter bring-up time when the new hardware arrives. Andreas Olsson said in the earlier interview that there is a lot of pressure on the software developers once the hardware arrives. Emulation of the platform can ease the preparations needed and thereby remove some of the pressure. This makes the gap between hardware and software, that were mentioned in the background, smaller and gives the software developers more time to explore the functionality and constraints of the new platform. More time to prepare for the platform transition and less stress when the hardware arrives could lead to both a better process and better products. On the other hand, the development time needed for the QEMU models is not accounted for here. As mentioned in the QEMU section, there is support for most architectures but there are also endless possibilities to extend the model with functionality closer to the real platform. Axis uses an extended QEMU model and the implementation time for this needs to be compared to the possible later time savings. However, the emulated platform only needs to be implemented once and can then be used for both transition to that platform and daily development. The current model is developed as a smaller project in the platform team at Axis.

Daily development

The integration process for peripherals in a camera differs in many ways from the transition to a new platform. But an emulated platform could help in a similar way, since one of the main problems Andreas Olsson talked about was waiting for hardware. The simulated camera block implemented in this thesis is a small example of how peripherals can be modeled to enable driver implementation. To make stubs and simulations of hardware and protocols is a well known technique in software development. In this thesis we use the simulated peripheral together with the emulated platform. This gives the software developers the possibility to run their application in its real environment as well as having more control and possibilities to explore different configurations and communication methods on the platform. In the camera block example it would have been rather easy to change the protocol or connection method. Andreas Olsson mentioned that this is a problem with changes in hardware and interfaces in their current approach. Working with early prototypes might increase these problems and therefore hurt productivity. But it could still be a way to explore different options and products. As mentioned earlier, the ASIC team at Axis is working with SystemC models before implementing peripherals in Verilog. These models could be used by developers as well, unfortunately this might be a problem with third party IPs.

Andreas Olsson also mentioned development boards with peripherals, available to connect to a PC. In Axis current setup it is not possible to connect these boards to a camera nor is there any other way to use the development boards within the camera platform. With the emulated camera platform these boards could be connected to the emulator via the PC serial port. Then the PC serial port could be bridged to a pseudo port connected to QEMU. This could ease the daily work for software developers because they will be able to run their applications as they would on the real platform. These boards are also a great way to explore hardware from different manufacturers before committing to one.

When implementing the simulated camera block we did not encounter any problems with documentation from Axis or the third party camera block provider. Andreas Olsson talked about communication problems with third party manufacturers and specifications that does not hold up in real applications. We implemented a well used and mature protocol, which is probably the reason that we did not notice these difficulties. To reduce future communication problems the emulated platform could be used together with simulated models. Hardware developers and manufacturers could provide simulated models, implementing the same API as their real product, to complement data sheets and specifications, to reduce misunderstandings and ease communication and cooperation. However, none of

Axis' providers provide such models, probably because it takes extra development time to produce up to date models for simulation and that service is not requested by Axis.

4.3 Chapter summary

This chapter discusses the result from the experiments, compares the emulated platform to a real camera and to multiple instances of the emulator. When running the test suites the emulator is very close in time to a real camera, which may open up possibilities to run tests on an emulator instead of physical hardware.

The work done in this thesis resembles a small low level development project. The experiences from this work have been discussed and the emulated platform is shown to be a useful tool, even though it still needs further development.

Chapter 5
Conclusions

This chapter summarizes the work done and results obtained throughout this thesis. Future work and improvements based on the work done is also included. The chapter, and the thesis, ends with a small section of closing thoughts.

5.1 Summary of the evaluation

We have seen throughout this thesis that it is technically possible to use QEMU as a development tool. We experienced problems with the emulated platform, some that needs to be addressed for it to be an efficient tool and some that is a part of the platform development process itself. Problems that are part of the platform development might be easier to address with the emulator, as it gives the developer more control.

Since the hardware platform does not have to exist physically for the emulation to be implemented, the gap between hardware and software becomes smaller. This lets the software developers start targeting the new platform earlier, which reduces the lead time since more work can be done in parallel.

We learned that exploration of external peripherals for Axis cameras can be a problem. The emulated platform can help in several ways. It gives, for example, the developers a way to connect peripherals to the camera platform, which is not possible to do with a real camera. We showed this with a simulation of a camera block, connected to the emulation.

Axis' nightly build test suite on the emulated platform yielded a promising result in terms of execution time. It was a bit slower then a real camera running on the previous platform, which was expected and not considered a problem. The time difference was a couple of minutes in a test that took about one and a half hour.

Running multiple instances of the emulator was a bit problematic, one and two instances worked as expected. However, when running three or more they started to behave differently. At least one instance crashed every time we tested, due to a soft lockup in the CPU. This is probably an error in the implementation of the emulator or OS.

5.2 Future work and recommendations

The work and investigations done in this thesis can be used as a base for different work and extensions in the future. We discovered limitations as well as possibilities for the emulated platform. The main areas are how an emulator can help during the development and the transition between platforms, as well as software testing. Even though the results from testing during the thesis work was mixed, future development can work with these issues and implement more suitable tools.

Emulated platform as a development tool

The possibilities for the development process is twofold and needs to be addressed in different ways. To use the emulated platform during Axis upcoming platform transition requires development of the emulated platforms. These processes could be merged, to port software to the new hardware via the emulated platform and fix problems on the platform as they emerge. However, this would increase the development time.

If the porting project integrate platform development as well, Axis will probably have an emulated platform fitting for most levels of daily development. There are still some issues that needs to be addressed if further emulator development is not included in the platform transition process, but still wanted as a tool for daily development on the new platform. Such as the soft lockup bug encountered during our testing.

In either case we recommend Axis to standardize a way to work with the emulator. This can be done in different ways, for this thesis we scripted the set up and start up process for a QEMU instance. There are other options for this that might be easier to use, for example using a VM manager [29]. This way there would be a standard set up and still possibilities to change the environment. Developers at Axis does not use a standard IDE. If they were they could look in to integrating QEMU in that environment, some of the background articles covered this kind of integration [22].

Andreas Olsson mentioned, in the earlier interview, that it would be advantageous to have the possibility to connect a developer peripheral board to the camera platform. This could, in theory, be done the same way the simulated camera block was connected over serial port. Driver developers could utilize a desktop PCs physical connections and link them to the pseudo ports connected to QEMU. The video output from the board could be connected to a computer screen and not to the emulation, this way the driver on the emulated platform could control the peripheral and the developer would have fast visual feedback. This approach does not allow any post processing of the video stream in the emulator, if that is wanted the PC connections might be utilized in another way.

GDB was mentioned in the background but not used in the thesis work. It is a popular tool that have good integration in QEMU. Another popular tool for debugging and profiling, that also works with QEMU is Valgrind [30]. Both of these tools could be a great help for low level developers, especially in cooperation with the emulated platform.

Testing

In terms of testing there are several possible improvements that can be made. To start of, several of the current issues in the case with several instances of QEMU (either directly on the host or in VirtualBox) needs to be resolved to ensure stability. If it is not possible to separate the instances with software another solution is required, most likely separate hardware for each instance of QEMU.

For testing on a server instead of real cameras to be successful the implementation of QEMU need to be on the same level as a real camera. For this to be possible real cameras with the new platform need to be available or a version of the old cameras need to be implemented in QEMU, preferably the first alternative. From here QEMU can be developed to be on par with the new camera and they can be compared. Even with the old cameras the differences between the success rate for the test is not remarkable high. Thereby it will probably be possible to bring the emulator to the same level as the cameras and from there start to use it in development and testing.

If the idea to separate the guest instances (in a similar way as VirtualBox) persists, there are alternatives. A rather modern approach is Docker [31], which instead of running a virtual machine uses containers. A container runs on the same kernel as the rest of the currently running applications but in a separate userspace. These containers run on any operating system and are thereby much more flexible than a virtual machine. The resources required for overhead is also much lower which result in less delay, waste of resources and a faster time when running the tests. Docker is open source which makes it easy to access and try out to see if it can be a possible future solution when it comes to running parallel instances of QEMU.

The number of running emulators can start being a problem to control when the number start to rise over three parallel instances at one moment. To make it easier a VM manager could be an option, there are several to choose from. With a VM manager it would be possible to control all the instances instead of manually start and stop each individual instance. However, the usefulness of a VM manager depends on the final method on how to run the emulators in parallel. Most VM managers support specific virtual machines and thereby the most successful solution might not have a supported manager. For direct QEMU management, virt-manager [32] is designed with virtual machines based on KVM (which is a part of QEMU) in mind. Virt-manager is open source and thereby easy to access. For VirtualBox, Oracle got their own VM manager which is built in and visible when launching VirtualBox.

5.3 Closing thoughts

This thesis set out to investigate how hardware platform emulation affects software development process. This has been done from three different angles: platform transition, daily development and testing. We have promising results in all three areas and we propose to Axis to keep using QEMU as a development tool and investigate it further. Our findings indicate that it can be a useful tool for the developers and therefore yield a better development process.

From our own perspective we have learned much about development and emulation. Some of the more important lessons are: The small tasks one think is simple may take much more time than expected. Sometimes it can be a good idea to stop and try a different approach. The result did not always turn out the way we expected, then we had to adapt and make the best of the given situation. Starting to write the report in an early stage saved us a lot of time and work in the end of the process and hopefully resulted in a better end product. The experiences we have received from the thesis work will be valuable in future projects and endeavors.

Bibliography

[1] Vocabulary.com. Simulation dictionary definition. URL https://www.vocabulary.com/dictionary/simulation.

[2] IBM. Virtualization in Education. 2007.

[3] debian.org. Debian jessie information page. URL https://wiki.debian.org/DebianJessie.

[4] Accellera. About SystemC. URL http://accellera.org/community/systemc/about-systemc.

[5] Shye Tzeng Shen, Shin Ying Lee, and Chung Ho Chen. Full system simulation with QEMU: An approach to multi-view 3D GPU design. In *ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems*. IEEE, 2010. doi:10.1109/ISCAS.2010.5537690.

[6] Fabrice Bellard. Qemu home page. URL http://wiki.qemu.org/Main_Page.

[7] QEMU Internals, 2015. URL http://qemu.weilnetz.de/qemu-tech.html#QEMU-Internals.

[8] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. In *'05 Proceedings of the annual conference on USENIX Annual Technical Conference*. USENIX Association Berkeley, 2005. URL http://static.usenix.org/event/usenix05/tech/freenix/full_papers/bellard/bellard.pdf.

[9] Imperas. Using Open Virtual Platforms to build, simulate and debug multiprocessor SoCs, 2008.

[10] Wind River Simics Product Page, 2015. URL http://www.windriver.com/products/product-overviews/Wind-River-Simics_Product-Overview.pdf.

[11] Jakob Engblom. Wind River Simics for Software Development, 2015. URL https://windriver.com/whitepapers/simics-for-software-development/WP_Simics_Software_Development.pdf.

[12] Ran Avinun. Concurrent Hardware/Software Development Platforms Speed System Integration and Bring-Up, 2011. URL http://www.cadence.com/rl/Resources/white_papers/system_dev_wp.pdf.

[13] Cadence Virtual System Platform, 2011. URL http://www.cadence.com/rl/Resources/datasheets/virtual_system_platform_ds.pdf.

[14] RISC-V foundation. Risc V QEMU, 2016. URL http://riscv.org/software-tools/riscv-qemu/.

[15] Edgar E. Iglesias. TLMu, 2011. URL https://edgarigl.github.io/tlmu/tlmu.pdf.

[16] Filippo Cucchetto, Alessandro Lonardi, and Graziano Pravadelli. A common architecture for co-simulation of SystemC models in QEMU and OVP virtual platforms. In *Very Large Scale Integration (VLSI-SoC), 2014 22nd International Conference*. IEEE, 2014. doi:10.1109/VLSI-SoC.2014.7004154.

[17] Daniel D. Gajski, Samar. Abdi, Andreas. Gerstlauer, and Gunar. Schirner. *Embedded System Design - Modeling, Synthesis and Verification*. Springer, 2009. doi:10.1007/978-1-4419-0504-8.

[18] Ivan Garcia, Leninca Pacheco, and Andreai Herrera. Defining a software process improvement-based methodology for embedded systems development. In *Proceedings - 2010 IEEE Electronics, Robotics and Automotive Mechanics Conference, CERMA 2010*, page 6. IEEE, 2010. doi:10.1109/CERMA.2010.95.

[19] Personal communication with software developer Andreas Olsson at Axis Communications. 2016-02-17 11:00.

[20] Hua-ming Qian and Chun Zheng. A Embedded Software Testing Process Model. In *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference*, Wuhan, 2009. IEEE. doi:10.1109/CISE.2009.5366362.

[21] OSDev.org. Kernel Debugging Wiki Page. URL http://wiki.osdev.org/Kernel_Debugging.

[22] Pradyumna Sampath and Rachana Rao. Efficient embedded software development using QEMU. In *13th Real Time Linux Workshop*. Linux Weekly News, 2011.

[23] Massimiliano D 'angelo, Alberto Ferrari, Ommund Ogaard, Claudio Pinello, and Alessandro Ulisse. A Simulator based on QEMU and SystemC for Robustness Testing of a Networked Linux-based Fire Detection and Alarm System. In *ERTS conference*, page 9. SPRINT Publications, 2012.

[24] Marc Boulé and Zeljko Zilic. *For Hardware Verification, Emulation, Post-Fabrication Debugging and On-Line Monitoring Generating Hardware Assertion Checkers*. Springer, Montreal, 2008. ISBN 978-1-4020-8585-7.

[25] Màrius Montón, Antoni Portero, Marc Moreno, Borja Martínez, and Jordi Carrabina. Mixed SWSystemC SoC Emulation Framework. *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*, pages 2338 – 2341, 2007. doi:10.1109/ISIE.2007.4374971.

[26] Linux manual *termios* page. URL `http://man7.org/linux/man-pages/man3/tcsetattr.3.html`.

[27] Linux manual *read* page. URL `http://linux.die.net/man/2/read`.

[28] Linux manual *write* page. URL `http://linux.die.net/man/2/write`.

[29] Tim Jones. Managing VMs with the Virtual Machine Manager, 2012.

[30] Christian Bornträger. Valgrind vs. KVM. In *KVM Forum 2014*, Düsseldorf, Germany, 2014. linux-kvm.org. URL `http://www.linux-kvm.org/images/d/d2/03x07-Valgrind.pdf`.

[31] Docker. Docker Home Page, 2016. URL `https://www.docker.com/`.

[32] Daniel P. Berrangé. Virt-manager, 2013. URL `https://virt-manager.org/`.

[33] Josh Boyer and Grant Likely. A Symphony of Flavours: Using the device tree to describe embedded hardware Grant Likely Secret Lab. In *Proceedings of the Linux Symposium*, volume 2, pages 27–37, Ottawa, Ontario, Canada, 2008.

Appendices

Appendix A
QEMU Network and Serial port setup

The network is set up with a DHCP server on the host that distributes IP addresses on a bridge interface. A TAP device is connected to the bridge interface and attached to the guest as a normal network connection. A TAP device is a software network device that handles Ethernet frames to user space application. In this case it take care of Ethernet frames between the bridge and the QEMU guest. The TAP device is attached to the QEMU instance with following command line arguments:

```
-net nic -net tap, ifname=tap0, script=no, downscript=no
```

Script and downscript are set to no, to disable QEMU from running `qemu-ifup` and `qemu-ifdown` when starting and stopping QEMU. These scripts are used to set up and take down a TAP device for QEMU, however they require a bridge interface to connect the TAP device to. This setup uses a custom script to set up both a network bridge and a TAP device and then start QEMU. The start script also close down all connections after the emulator is powered off. Without the ifup/down scripts QEMU can be started without `sudo`, which could be an advantage in some situations. As seen in Figure 3.1 the guests network is separated from the networks that the host is connected to. With this setup it is easy to add the hosts network connection to the bridge and let the guest access a larger network and the Internet, which is useful if the emulation is running on a remote location.

```
&uart0 {
        status = "ok";
        dmas = <>;
};
```

Figure A.1: Definition of an uart port in device tree source file.

This QEMU setup supports four serial ports between the host and guest. As seen in Figure 3.1, one is used to redirect the console to the host. The remaining three are connected to pseudo terminals on the host (`/dev/pts/xx`) and can be used to attach

```
  ...
char device redirected to /dev/pts/1 (label serial1)
char device redirected to /dev/pts/2 (label serial2)
char device redirected to /dev/pts/3 (label serial3)
  ...
[0.376395] Serial: UART driver
[0.380982] f8036000.uart: tty0 at MMIO 0xf8036000 (irq = 57)
[0.385707] f8037000.uart: tty1 at MMIO 0xf8037000 (irq = 58)
[0.386835] f8038000.uart: tty2 at MMIO 0xf8038000 (irq = 59)
[0.388013] f8039000.uart: tty3 at MMIO 0xf8039000 (irq = 60)
[0.388941] console [tty3] enabled
  ...
```

Figure A.2: Serial ports during boot, edited to only display relevant information.

peripherals to the guest. To be able to set up the serial ports they need to be defined in the device tree [33]. Figure A.1 shows the definition of an UART port in the device tree, this is done for ports 0-3. The device tree is compiled to a device tree blob (dtb) file and loaded by QEMU next to the kernel image. The kernel and dtb file are loaded and the serial ports are attached to pseudo terminals on the host with the following command line arguments:

```
       -nographic -kernel zImage.mon -dtb board.dtb
   -serial mon:stdio -serial pty -serial pty -serial pty
```

Where `zImage` is the kernel image and `board.dtb` is the device tree blob. The first serial flag directs the console to the terminal where QEMU is booted. The three following flags creates pseudo terminals on the host and attach them to serial ports on the guest. The created ports as well as the mapping between UARTs and serial ports can be seen during boot in Figure A.2.

When the serial ports works as expected a small demo is implemented with help from the functions in Figure A.3. This demo is later extended to handle the driver and camera block protocol.

A.1 Result from setup

The network connection works as expected and Axis live view web service can be reached from the host via a web browser, as seen in Figure A.4.

```
/* Open serial port for reading and writing,
save it's old state and set up for communication. */
int serial_init();

/* Restore old state and close. */
int serial_close();

/* Waits up to a specified time in ms for to
read form serial port to buffer. */
int serial_read(char*, size_t, size_t);

/* Writes a specified number of bytes from buffer
to serial port. */
int serial_write(const char *, size_t);
```

Figure A.3: Helper functions for serial port communication.

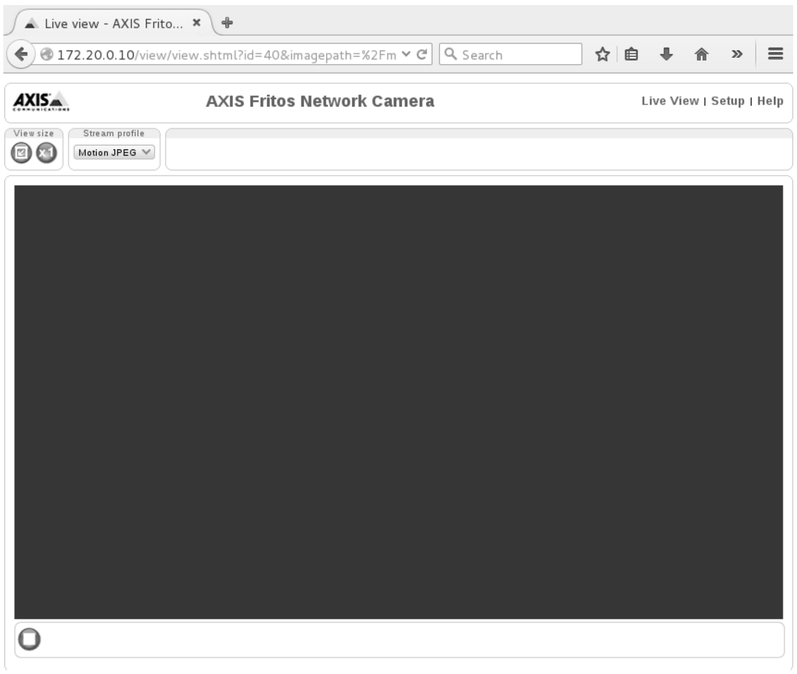
Live view - AXIS Frito...
172.20.0.10/view/view.shtml?id=40&imagepath=%2Fm
Search
AXIS
AXIS Fritos Network Camera
Live View | Setup | Help
View size
Stream profile
Motion JPEG

Figure A.4: Axis web live view as seen from the hosts browser.

Appendix B

Raw data from experiments

This Appendix contains the raw data which the graphs used in the evaluation are based on. The results are from experiments run on different platforms and environments described in the captions and labels.

Table B.1: Results from running the test suites on one instance of QEMU compared to a real camera.

QEMU				Real camera			
Test cases	Errors	Failures	Time (s)	Test cases	Errors	Failures	Time (s)
20	1	0	38	20	1	0	31
11	1	0	31	11	1	0	20
89	2	2	1814	89	0	2	1575
112	0	0	115	120	3	0	78
42	13	2	733	42	5	0	574
56	0	1	60	49	1	0	135
144	28	17	585	108	2	0	874
50	0	0	43	50	0	0	20
322	2	3	436	322	0	0	335
19	9	0	7	19	9	0	3
77	35	0	14	77	35	0	5
35	0	0	49	35	0	0	32
48	21	0	134	48	0	1	192
Summary							
1165	112	25	4114	1130	60	3	3885

The suite contains a subset of a larger test suite and each individual test suite have a set of test cases. These are displayed to give an indication of why the time differs between the different test suites.

Table B.2: Results from several instances of QEMU on one host and in VirtualBox compared to the results from a real camera.

Nbr of instances	Test cases	Errors	Failures	Success rate (%)	Time (s)
1	1165	112	25	88.24	4114
2	1165	122	17	88.07	4502
Real camera	1130	60	3	94.42	3885
VirtualBox 1	1165	123	20	87.73	5781
VirtualBox 2	1165	123	19	87.81	6268

Table B.2 summarize the results from the runs on one and two instances of QEMU, a real camera and when running QEMU in VirtualBox. Each row includes the summary of test cases, errors, failures, success rate of the test suite and the running time of the test suite.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTATIONSDAG 2016-06-03

EXAMENSARBETE Emulation-based software development for embedded systems
STUDENTER Simon Wallström, Adam Dalentoft
HANDLEDARE Flavius Gruian (LTH), Gustav Sällberg (Axis Communications)
EXAMINATOR Krzysztof Kuchcinski (LTH)

Hårdvara som mjukvara eller vad är en emulator och vad kan den användas till?

POPULÄRVETENSKAPLIG SAMMANFATTNING av **Simon Wallström, Adam Dalentoft**

Överallt omkring oss finns det små datorer i näst intill allt. Dessa innehåller både hårdvara och mjukvara vilket kan vara besvärligt under utvecklingen. Där mjukvara är program som körs på hårdvaran. För att undersöka möjligheter att ändra detta föreslås en emulator, ett sätt att köra hårdvara som mjukvara.

Små datorer överallt omkring oss

I dagens samhälle finns små datorer inbyggt i nästan allt. Till exempel telefoner, bilar, övervakningskameror, till och med i vitvaror så som kylskåp och tvättmaskiner. Det är tydligt att dessa inbyggda datorer har olika syften. Vissa har endast en uppgift medan andra kan nästan hantera lika mycket som en vanligt persondator. Dessa så kallade inbyggda system beror mycket på den använda hårdvaran, eftersom dess funktionalitet och prestanda begränsar vad det kan användas till. Det är även tidskrävande och kostsamt att bygga eller ändra hårdvara. Detta försvårar utvecklingen av mjukvaran till inbyggda system, eftersom hårdvaran praktiskt taget behöver vara klar innan mjukvaran kan börja utvecklas.

Vi har undersökt hur en emulator kan användas för att komma runt detta problem. Emulatorer beskrivs enklast som ett försök att härma funktionaliteten hos hårdvara i mjukvara och på så sätt byta ut hårdvaran mot mjukvara. Ett vanligt förekommande användningsområde av emulatorer hos personer utanför mjukvaruindustrin är till att spela spel till gamla spelkonsoler. Om vi vill göra detta installerar vi en emulator av den gamla konsolen på vår dator och kör spelet i den. På samma sätt kan man göra med andra system, till exempel köra olika operativsystem eller inbyggda system. Vi har därmed tagit hårdvaran till vårt inbyggda system och gjort om den till en emulator som kan köras på vilken persondator som helst.

Varför är detta användbart?

Med hjälp av emulatorn har vi plockat bort beroendet av färdig hårdvara och istället skapat en modell som går att använda i ett mycket tidigare skede i utvecklingsprocessen. Med denna emulator är det möjligt att köra program specifikt för just det inbyggda systemet på en vanlig dator. Som liknelse kan vi ta emulatorn till spelkonsolen igen, spelen kan liknas med våra program. Programmen behöver en specifik miljö att köra i för att de ska fungera. Därmed är det tekniskt möjligt att utveckla mjukvaran och hårdvaran till det inbyggda systemet parallellt istället för en sak i taget. Våra resultat pekar på att det kan vara smidigare för en mjukvaruutvecklare att använda en emulator, eftersom den är enklare att ändra på då den inte är huggen i sten.

Nu när vi har en emulator kan den användas på flera sätt, även efter själva utvecklingen av det nya systemet är klart. Att kunna byta ut fysisk hårdvara när det kommer till testning kan spara både plats och pengar. Istället för att testa mjukvaran direkt på ett chip är det möjligt att ha servrar som kör emulatorer. Därmed krävs endast underhåll för några servrar istället för ett rum fullt med olika varianter av hårdvara.

Som icke-utvecklare av inbyggda system har dock detta ingen direkt påverkan. Däremot indirekt kan en ändring i utvecklingsprocessen snabba upp lanseringar av nya produkter.