TECHNOLOGICAL EDUCATIONAL INSTITUTE OF CRETE

Paravirtualized applications on top of an L4/Fiasco Microkernel

by

Emmanouil Fragkiskos Ragkousis

A thesis submitted in partial fulfillment for the degree of Bachelor of Science

in the

Technological Educational Institute of Crete
Department of Informatics Engineering

March 31, 2019

TECHNOLOGICAL EDUCATIONAL INSTITUTE OF CRETE

Σύνοψη

Technological Educational Institute of Crete
Department of Informatics Engineering

Bachelor of Science

by Emmanouil Fragkiskos Ragkousis

In this thesis we added Zedboard support to L4/Fiasco, which allowed us to use it as a hypervisor. In this way we can achieve a better use of resources by sharing them between multiple operating systems and/or bare metal programs. It also allows us to achieve better security by controlling access to parts of the hardware.

Thesis Supervisor: Kornaros Georgios

Title: Assistant Professor at Department of Informatics Engineering, TEI of Crete

TECHNOLOGICAL EDUCATIONAL INSTITUTE OF CRETE

Σύνοψη

Technological Educational Institute of Crete
Department of Informatics Engineering

Bachelor of Science

by Emmanouil Fragkiskos Ragkousis

Σε αυτή την πτυχιακή, προσθέσαμε υποστήριξη για το Zedboard στο L4/Fiasco, με σκοπό να το χρησιμοποιήσουμε σαν hypervisor. Με αυτόν τον τρόπο μπορούμε να επιτύχουμε καλύτερη χρήση των πόρων του συστήματος, μοιράζοντας τον μεταξύ διαφορετικών λειτουργικών συστημάτων και εφαρμογών. Επίσης μας επιτρέπει να επιτύχουμε καλύτερη ασφάλεια ελέγχοντας το την πρόσβαση στο hardware.

Επιβλέπων Πτυχιακής: Γεώργιος Κορνάρος

Τίτλος: Επίκουρος Καθηγητής του τμήματος Μηχανικών Πληροφορικής, ΤΕΙ Κρήτης

Acknowledgements

This bachelor thesis is my first academic milestone and looking back I would like to thank all the people, that are not listed below, for their support and encouragement until today.

First of all I would like to thank my thesis supervisor George Kornaros, for his much needed advice, patience and help, during my thesis.

My friend and colleague Othon Tomoutzoglou for supporting and helping me in times of need.

My friends Dennis Bautembach and Giannis Halvatzakis for the endless talks for our aspirations and dreams for the future.

Finally I would like to thank my family for all their support all these years.

Contents

List of Figures

List of Code Snippets

Abbreviations

CPU -> Central Processing Unit

FPGA -> Field Programmable Gate Array

I/O -> Input/Output

IPC -> Inter-process Communication

IRQ -> Interrupt Request

SLCR -> System-Level Control registers

SoC -> System On Chip

This page is intentionally left blank

Chapter 1

Introduction

This chapter describes the knowledge background to understand this thesis. Furthermore it discusses the motivation behind it, as well similar work done previously. Lastly, a short description of the thesis structure can be found.

1.1 Background and Motivation

A microkernel system is a system where the functionality of the system is moved out of the kernel, into a set of userspace applications that have little effect on the stability of the system.

The Zedboard is a Zynq based board which contains an ARM CPU together with an FPGA.[1]

A L4/Fiasco microkernel[2][3] based system is able to be used as a hypervisor to run other system on top of it. The motivation of this thesis is to prove that:

1. L4/Fiasco can be used on the Zedboard.
2. A Linux system can be used on top of this as a userspace application.
3. We can control which application has access to which parts of the hardware.
4. Access and use FPGA based devices from Linux.

1.2 Similar Work

Perfomance measurements of a second generation L4 micro kernel have been done by modifying a Linux kernel to run on top of it and compare it to a normal Linux system.[4]

A number of previous studies have analysed ways to isolate application running on a cpu either by testing the security of standard lightweight OS level virtualization technologies [5] by comparing the provided isolation of various implementations, or implemented a secure kernel level execution environment for ARM processors and guarantee isolation [6].

Other approaches are being done though hardware by utilizin a system-level MMU to contol access to the system resources [7], [8], or by segmenting the memory and implementing access levels based on application priviliges[9].

Studies have been done on improving the perfomance of systems by learning how a resource is used and how to better allocate it in the future based on a specific goal [10], or by distributing the workload on energy efficient SoCs [11] .

Other similar engineering work has been done, from Taeung Song and Yeongchann Han where they ported L4 and L4Linux to Freescale i.MX6 Quad SABRE SD[12].

1.3 Thesis Structure

Chapter 1 informs the reader of the background of this thesis and contains a short description of the thesis structure.

Chapter 2 explains the theory needed in order to understand the combination of hardware and software used.

Chapter 3 describes implementation details on how to reproduce the work, how Zedboard support was added and how L4Linux can access the hardware.

Chapter 4 presents the results of running applications on L4/Fiasco.

Chapter 5 finishes this thesis with the conclusions of this bachelor thesis and thoughts on possible future improvements.

Chapter 2

Theory and State of the Art

The purpose of this chapter is to make a basic introduction of the software/hardware architecture used in this thesis. The reader needs to have an understanding of what he is going to read, regardless of his familiarity with the subject.

2.1 The L4/Fiasco System

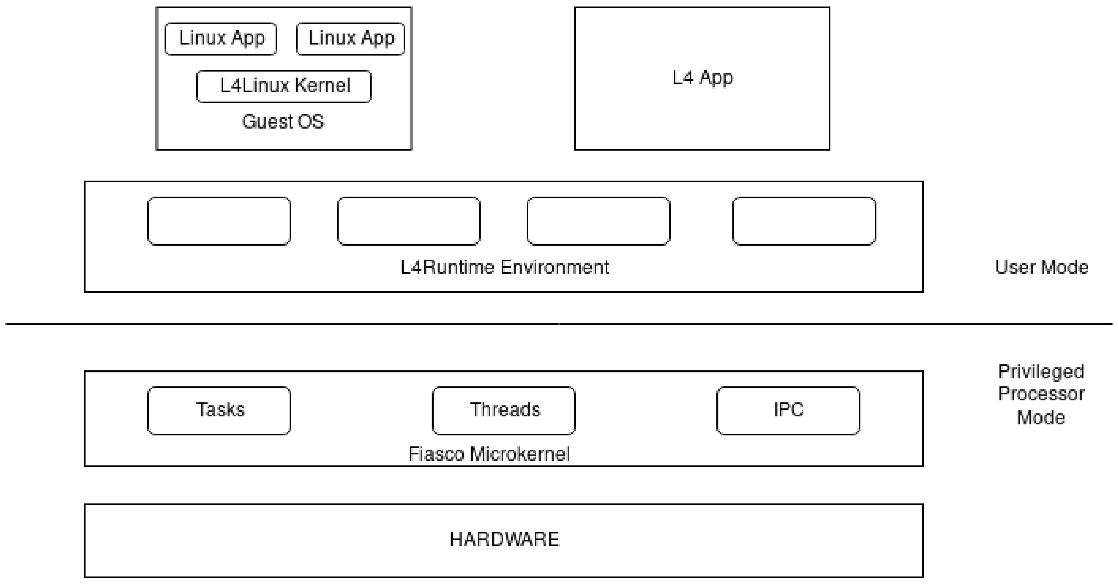


Figure 2.1: L4/Fiasco system architecture

2.1.1 Hardware

The hardware in this thesis is the ZedBoard using the Zynq 7000 SoC. This thesis intoduces ZedBoard peripheral support into the L4/Fiasco upstream code.

2.1.2 The Fiasco Microkernel

Fiasco is a 3rd-generation microkernel developed by the Operating Systems Group TUD:OS of the TU Dresden in 1998 and released under the GNU General Public License version 2 (GPLv2). It implements the L4 ABI, along with various extensions. Fiasco supports the x86, x86_64, MIPS and Arm architectures.

Fiasco supports running real-time, time-sharing and virtualization applications concurrently on the same machine. This make it suitable as a hypervisor to run different kernel on top of it.

Inside Fiasco we can find tasks, threads and the IPC. A task comprises of a memory address space (represented by the task's page table) and an object space (holding the kernel protected capabilities). Each task can be bound with multiple threads, which are used to execture code and are controlled by the Fiasco scheduler. The Inter Process Communication (IPC) is the syncronous communication mechanism used to transmit arbitrary data between threads and to resolve hardware exceptions.

2.1.3 L4 Runtime Environment

The L4 Runtime Environment (L4Re) was developed in order to provide the necessary infrastructure on top of Fiasco.OC for conveniently developing applications. The microkernel is the only program that runs in privileged processor mode and it does not include any complex services. L4Re runs on top of the kernel and provides a user-level infrastructure that includes basic services such as program loading, memory management and device drivers. L4Re also provides the environment for applications, including libraries and processing local functionality. The L4Re includes the IO server which controls access to the hardware.

2.1.4 L4Linux

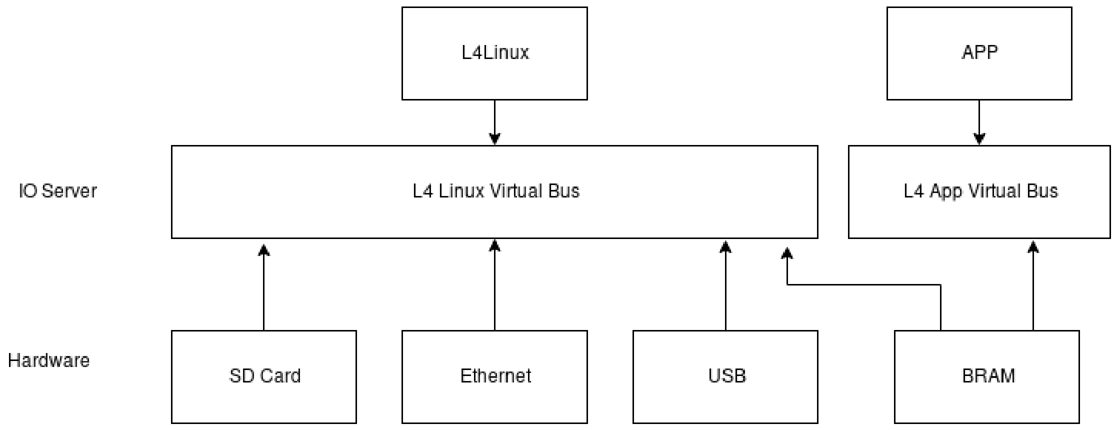


FIGURE 2.2: L4/Fiasco system architecture

L4Linux is a port of the Linux kernel, modified to be able to run para-virtualized on top of a hypervisor L4/Fiasco in our case, completely without privileges. L4Linux runs in user-mode on top of the micro-kernel, side-by-side with other micro-kernel applications such as real-time components.

Through the IO server we create two virtual buses, one called for the Linux guest and one for the L4 App. Each of these buses have a different peripherals assigned to them. A task (Linux guest/App) is assigned to each bus. The task cannot access hardware outside of the virtual bus.

2.2 State of the Art

This section presents current alternatives available on ARM platforms.

2.2.1 Docker, LXC and Nix/Guix

Docker, LXC and Nix/Guix all offer a way to have lightweight VMs (called containers) while sharing the same kernel. The different containers work in completely seperate environments and cannot access each other's working internals. This is achieved through the usage of kernel namespaces and/or virtual chrooted environemts. LXC is a userspace interface which offers an API that can be used on its own or indirectly from Docker, Nix and Guix. The implementation differences between Docker and Nix/Guix are out of the scope of this thesis.

All the solution described above require running a full Linux system.

2.2.2 Xen Hypervisor

Xen is almost identical to what L4/Fiasco can offer. It has a mikrokernel in its core which allows for paravirtualized operating systems to run on top of it. The difference is that while Xen is designed with only virtualization in mind, L4/Fiasco can also run simple L4 Apps which directly control only a specific part of the hardware. These L4 App run as servers which then other application can communicate with and exchange arbitary data.

Chapter 3

Implementation

This chapter contains details on everything related to reprodue a fully usable L4/Fiasco based system on the Zedboard. It includes a description of the tools needed to properly build everything, the details about porting ARM Linux drivers to L4Linux and lastly how to access FPGA hardware from within L4Linux. This chapter describes the work which I did.

3.1 Setting up the environment

This sections contains information on how to reproduce the environment needed to achieve what is described in this thesis.

3.1.1 Sources

The orignal sources used are from the svn and/or git repositories of the related projects. Listed bellow are the individual svn urls along with which revision/commit was used. Modifications to those are currently in private git repositories and will probably be upstreamed.

SVN: Fiasco Kernel Revision 72
SVN: L4 Runtime Environment Revision 72
SVN: L4Linux Revision 54

rootfs: Linaro precise dev image

3.1.2 Tools

To successfully build the specific source version used, the Linaro 4.9-2014.09 toolchain is needed. In order to automate the tools deployment Guix was used and a set of bash scripts. Below are some samples. The code is pretty self-explanatory.

The commands to setup the environment are:

1. guix environment l4fiasco-env –pure
2. source setup_env.sh

```
(define-public linaro-toolchain-4.9
  (package
    (name "linaro-toolchain")
    (version "4.9-2014.09_linux")
    (source
     (origin
       (method url-fetch)
       (uri (string-append "https://releases.linaro.org/archive/14.09/components/\
toolchain/binaries/gcc-linaro-arm-linux-gnueabihf-" version ".tar.xz"))
       (sha256
        (base32
         "148q9xnwn8ygqzpxpv8ayj06cdnf27h4p8fzvn5krsx0mq6arzqc"))))
    (build-system trivial-build-system)
    (arguments
     `(#:modules ((guix build utils))
       #:builder (begin
                   (use-modules (guix build utils)
                                (srfi srfi-26))

                   (let* ((source (assoc-ref %build-inputs "source"))
                          (tar    (assoc-ref %build-inputs "tar"))
                          (xz     (assoc-ref %build-inputs "xz"))
                          (output (assoc-ref %outputs "out")))
                     (setenv "PATH" (string-append xz "/bin"))
                     (system* (string-append tar "/bin/tar") "xvf"
                              source)
                     (copy-recursively "gcc-linaro-arm-linux-gnueabihf-4.9-2014.09_linux/"
                                       output)
                     #t))))
    (native-inputs `(("tar" ,tar)
                     ("xz" ,xz)))
    (supported-systems '("i686-linux" "x86_64-linux"))
    (home-page "https://releases.linaro.org")
    (synopsis "Linaro arm cross-toolchain")
    (description
     "Linaro arm cross-toolchain binaries that can be used to build Fiasco, L4
 and L4Linux")
    (license license:gpl2)))

(define-public l4fiasco-env
  (package
    (name "l4fiasco-env")
    (version "0.1")
    (source #f)
    (build-system trivial-build-system)
    (native-inputs
     `(("linaro-toolchain" ,linaro-toolchain-4.9)
       ("u-boot" ,u-boot-tools)
       ("dtc" ,dtc)
       ("doxygen" ,doxygen)))
    (synopsis "Octopress Ruby Environment")
    (description "This file automates the creation of a cross-arm environment so I can
build Fiasco,L4 and L4Linux.")
    (home-page #f)
    (license license:expat)))
```

Code Snippets 3.1: Guix environment example

```
#! /bin/sh

if [ -n "$GUIX_ENVIRONMENT" ]
then
    export GCC_PATH_PREFIX=$GUIX_ENVIRONMENT/bin/arm-linux-gnueabihf-
else
    export GCC_PATH_PREFIX=/opt/gcc-linaro-arm-linux-gnueabihf-4.9-2014.09_linux/bin/arm-linux-g
    export PATH=/home/$USER/git_repos/u-boot-xlnx/tools/:$PATH
fi
export PATH=$(pwd)/bin:$PATH
export L4ARCH=arm
export ARCH=arm
export CROSS_COMPILE=arm-linux-
```

Code Snippets 3.2: setup_env.sh Script

3.2 Adding Zedboard Support

This sections will go through the process of getting a live L4Linux system, on top L4/Fiasco. It will describe the problems encountered and how they were solved. It will also show how a bram was added and accessed from inside the L4Linux.

3.2.1 Building Fiasco.OC

True to its microkernel nature Fiasco is really small. As a result it's really easy to configure it to run on the Zynq processor.

The exact steps are:

1. make BUILDDIR=build
2. cd build
3. make config
4. make

In config we should enable:

- Architecture (ARM processor family)
- Platform (Xilinx Zynq)
- UART (Use UART 1)
- CPU (ARM Cortex-A9 CPU)

Everything else can be left at default. UART 1 is the UART port used on the Zedboard by default.

3.2.2 Buidling L4Re

The L4 runtime environemt is built in two steps. First we need to build the libraries and servers.

1. mkdir build
2. make O=build config
3. make O=build

The following config options should be enabled:

- Target Architecture (ARM architecture)
- CPU variant (ARMv7A type CPU)
- Platform Selection (Xilinx Zynq Zedboard)

3.2.3 Buidling a Hello App

Before actually building L4Linux it had to be proven that a simple Hello App can be written and run. The following describes how this simple setup boots.

1. Fiasco.OC — Microkernel
2. Sigma0 — Root Pager
3. Moe — Root Task
4. Ned — Init Process
5. hello — Hello World Application

In order to build the Hello App, one needs a modules.list file containing information on the modules to be used and a cfg script. Moe is the first task which is started in L4Re-based systems. What Moe does, is offer basic L4Re abstractions for Ned, as well as instructions on how to proceed. Moe is used as a loader from Ned, as a mean for starting applications.

Moe starts Ned as the default init process. Ned's job is to bootstrap the system running on L4Re. The main thing to do here is to coordinate the startup of services and applications as well as to provide the communication channels for them. The central facility in Ned is the Lua script interpreter with the L4Re and ELF-loader bindings.

The boot process is based on the execution of one or more Lua scripts that create communication channels (IPC gates), instantiate other L4Re objects, organize capabilities (access rights) to these objects in sets, and start application processes with access to those objects (or based on those objects).

```
modaddr  0x002000000
entry hello
  kernel   fiasco -serial_esc
  roottask moe rom/hello-test.cfg
  module   l4re
  module   ned
  module   hello-test.cfg
  module   test-bare-hello
```

Code Snippets 3.3: modules.list

```
local L4 = require("L4");
L4.default_loader:start({}, "rom/test-bare-hello");
```

Code Snippets 3.4: hello-test.cfg

Finally, from inside the L4 source dricectory, the uImage file can be produced with:

```
make uimage O=build E=hello-test MODULE_SEARCH_PATH=path/to/fiasco/sources
```

This image when run, will continuesly print 'Geia sou kosme' in the uart port.

3.2.4 Building L4Linux

Building L4Linux and gaining access to the peripherals was a major challenge. The purpose of the next sections will be to guide you through the details of adding support for those devices in L4Linux.

3.2.4.1 L4Linux status on Zynq

Before the work described in this thesis, L4Linux could boot on the Zynq, and had a basic system with the help of a generic arm ramdisk. But there was no driver support for any of the devices on the Zedboard.

When running L4Linux

3.2.5 Giving L4Linux access to devices

For L4Linux to find and use the devices we need to 1) describe where the devices are, and 2) enable the drivers.

3.2.5.1 The Io Server

All platform devices and resources such as I/O memory and interrupts are handled by the Io Server. Io grants access to the clients based on configuration written in Lua.

Io's configuration consists of two parts:

- The description of the real hardware
- The description of virtual buses

Both descriptions represent a hierarchical (tree) structure of device nodes. Where each device has a set of resources attached to it. And a device that has child devices can be considered a bus.

```
local Res = Io.Res
local Hw = Io.Hw

Io.hw_add_devices(function()
      SLCR = Hw.Device(function()
            compatible = {"xlnx,zynq-slcr", "syscon", "simple-mfd"};
            Property.hid  = "xlnx,zynq-slcr";
            Resource.mem = Res.mmio(0xf8000000, 0xf8000fff);
      end);
      NIC = Hw.Device(function()
            compatible = {"cdns,zynq-gem", "cdns,gem", "cdns,macb"};
            Property.hid  = "cdns,zynq-gem";
            Resource.irq  = Io.Res.irq(54);
            Resource.mem = Res.mmio(0xe000b000, 0xe000bfff);
      end);
      MMC = Hw.Device(function()
            compatible = {"arasan,sdhci-8.9a"};
            Property.hid  = "arasan,sdhci-8.9a";
            Property.flags = Io.Hw_device_DF_dma_supported;
            Resource.irq = Io.Res.irq(56);
            Resource.regs = Io.Res.mmio(0xe0100000, 0xe0100fff);
      end);
      DMAC = Hw.Device(function()
            compatible = {"arm,pl330", "arm,primecell"};
            Property.hid  = "arm,pl330";
            -- Property.flags = Io.Hw_device_DF_dma_supported;
            Resource.irq = Res.irq(45);
            Resource.irq = Res.irq(46);
            Resource.irq = Res.irq(47);
            Resource.irq = Res.irq(48);
            Resource.irq = Res.irq(49);
            Resource.irq = Res.irq(72);
            Resource.irq = Res.irq(73);
            Resource.irq = Res.irq(74);
            Resource.irq = Res.irq(75);
            Resource.regs = Io.Res.mmio(0xf8003000, 0xf8003fff);
      end);
end);
```

Code Snippets 3.5: zedboard.devs

```
local Hw = Io.system_bus()

Io.add_vbus("l4linux", Io.Vi.System_bus
{
   slctrl = wrap(Hw.SLCR);
   ethernet = wrap(Hw.NIC);
   sdhci = wrap(Hw.MMC);
   dmacontroller = wrap(Hw.DMAC);
   bram = wrap(Hw.FPGA_BRAM);
})
```

Code Snippets 3.6: hw_devices.io

3.2.5.2 Device trees

We also need to tell Linux where to find those devices. For this we use a modified device tree file.

```
/*
 * Basic DT for L4Linux on zynq.
 */

/dts-v1/;

/ {
        model = "L4Linux (DT)";
        compatible = "L4Linux";

        #address-cells = <1>;
```

```
#size-cells = <1>;
chosen { };
aliases { };


amba {
        compatible = "simple-bus";
        #address-cells = <0x1>;
        #size-cells = <0x1>;
interrupt-parent = <&intc>;
        ranges;

intc: l4icu {
    compatible = "l4,icu";
    interrupt-controller;
    l4icu-type = "gic";
    #interrupt-cells = <3>;
            #address-cells = <0>;
};


gem0: ethernet@e000b000 {
    compatible = "cdns,zynq-gem", "cdns,gem";
    reg = <0xe000b000 0x1000>;
    status = "okay";
    interrupts = <0 22 4>;
    clocks = <&clkc 30>, <&clkc 30>, <&clkc 13>;
    clock-names = "pclk", "hclk", "tx_clk";
    #address-cells = <1>;
    #size-cells = <0>;
    phy-mode = "rgmii-id";
    phy-handle = <&ethernet_phy>;

    ethernet_phy: ethernet-phy@0 {
      reg = <0>;
    };
        };


sdhci0: sdhci@e0100000 {
  compatible = "arasan,sdhci-8.9a";
  status = "okay";
  clock-names = "clk_xin", "clk_ahb";
  clocks = <&clkc 21>, <&clkc 32>;
  interrupt-parent = <&intc>;
  interrupts = <0 24 4>;
  reg = <0xe0100000 0x1000>;
        };


slcr: slcr@f8000000 {
                #address-cells = <1>;
                #size-cells = <1>;
                compatible = "xlnx,zynq-slcr", "syscon", "simple-mfd";
                reg = <0xF8000000 0x1000>;
```

```
                        ranges;
                        clkc: clkc@100 {
                                #clock-cells = <1>;
                                compatible = "xlnx,ps7-clkc";
                                fclk-enable = <0>;
                                clock-output-names = "armpll", "ddrpll", "iopll", "cpu_6or4x",
                                                "cpu_3or2x", "cpu_2x", "cpu_1x", "ddr2x", "ddr3x
                                                "dci", "lqspi", "smc", "pcap", "gem0", "gem1",
                                                "fclk0", "fclk1", "fclk2", "fclk3", "can0", "can
                                                "sdio0", "sdio1", "uart0", "uart1", "spi0", "spi
                                                "dma", "usb0_aper", "usb1_aper", "gem0_aper",
                                                "gem1_aper", "sdio0_aper", "sdio1_aper",
                                                "spi0_aper", "spi1_aper", "can0_aper", "can1_ape
                                                "i2c0_aper", "i2c1_aper", "uart0_aper", "uart1_a
                                                "gpio_aper", "lqspi_aper", "smc_aper", "swdt",
                                                "dbg_trc", "dbg_apb";
                                reg = <0x100 0x100>;
                ps-clk-frequency = <33333333>;
                        };

                        rstc: rstc@200 {
                                compatible = "xlnx,zynq-reset";
                                reg = <0x200 0x48>;
                                #reset-cells = <1>;
                                syscon = <&slcr>;
                        };

                        pinctrl0: pinctrl@700 {
                                compatible = "xlnx,pinctrl-zynq";
                                reg = <0x700 0x200>;
                                syscon = <&slcr>;
                        };
                };
            dmac_s: dmac@f8003000 {
                compatible = "arm,pl330", "arm,primecell";
                reg = <0xf8003000 0x1000>;
                interrupt-parent = <&intc>;
                interrupt-names = "abort", "dma0", "dma1", "dma2", "dma3",
                        "dma4", "dma5", "dma6", "dma7";
                interrupts = <0 13 4>,
                             <0 14 4>, <0 15 4>,
                             <0 16 4>, <0 17 4>,
                             <0 40 4>, <0 41 4>,
                             <0 42 4>, <0 43 4>;
                #dma-cells = <1>;
                #dma-channels = <8>;
                #dma-requests = <4>;
                clocks = <&clkc 27>;
                clock-names = "apb_pclk";
                };

        };
};
```

Code Snippets 3.7: zynq.dts

3.2.6 Adding clock driver support to L4Linux for Zynq

There are two way in which L4Linux can access the platform clocks. One is to give Linux direct access to the clocks and the other is to have a L4 server which will control the clocks and serve clock requests.

In this thesis the first way was used.

3.2.6.1 The SLCR Registers

The System-Level Control registers (SLCR) consist of various registers that are used to control the behavior of the system. In order to access the I/O peripherals, including the peripheral clocks, the SLCR needs to be unlocked.

The problem was that L4Linux L4 architecture did not have support for this. Fortunately the SLCR drivers could be ported from the Linux ARM architecture. After the drivers were added, L4Linux need to be instructed to initialize the SLCR early in the system boot, before initializing device interrupts.

```
@@ -30,6 +30,9 @@
 #include <l4/sys/icu.h>
 #include <l4/sys/cache.h>
 #include <l4/io/io.h>
+#ifdef CONFIG_L4_PLATFORM_ZYNQ
+#include <mach/common.h>
+#endif

 enum { L4X_PLATFORM_INIT_GENERIC = 1 };

@@ -257,6 +261,10 @@ static void __init l4x_mach_init_of_irq(void)

 static void __init l4x_mach_init_irq(void)
 {
+
+#ifdef CONFIG_L4_PLATFORM_ZYNQ
+       zynq_early_slcr_init();
+#endif
        /* Call our generic IRQ handling code */
        l4lx_irq_init();
```

Code Snippets 3.8: arch/l4/kernel/arch-arm/l4.c

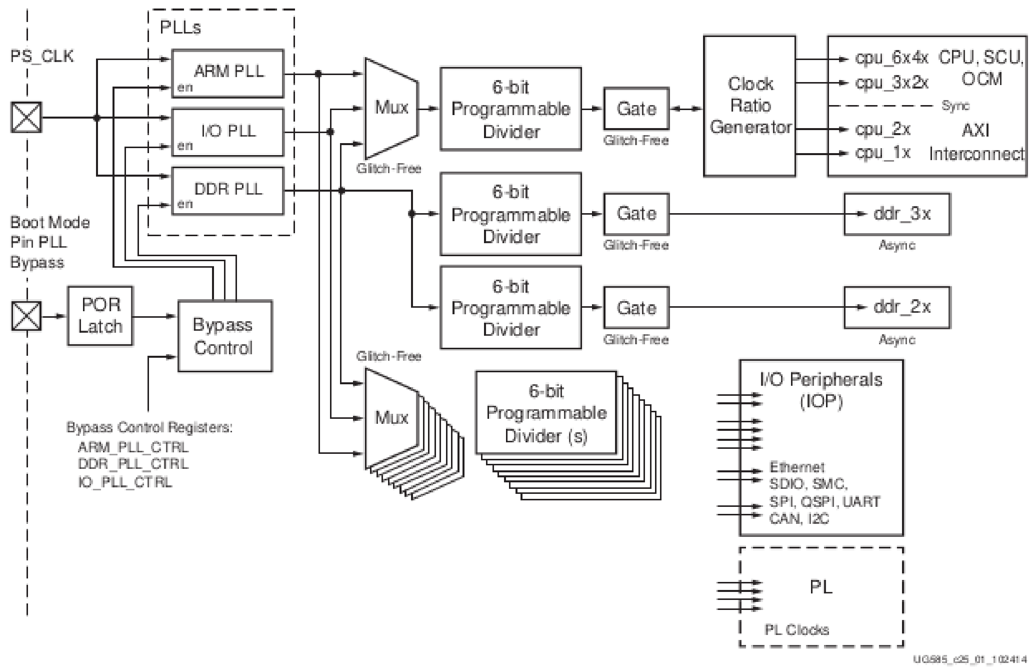


FIGURE 3.1: Zynq Clock Subsystem

3.2.6.2 The Zynq Clocks

The cpu and ddr clocks are controlled by L4/Fiasco and provide COMMON_CLK which can be used from L4Linux as the system main clock source. But for example in order to control the ethernet device, Linux needs access to SLCR_GEM0_CLK_CTRL

```
#define SLCR_GEM0_CLK_CTRL          (zynq_clkc_base + 0x40)
...
static DEFINE_SPINLOCK(gem0clk_lock);
...
static const char *gem0_mux_parents[] __initdata = {"gem0_div1", "dummy_name"};
static const char *const gem0_emio_input_names[] __initconst = {
  "gem0_emio_clk"};
...
for (i = 0; i < ARRAY_SIZE(gem0_emio_input_names); i++) {
  int idx = of_property_match_string(np, "clock-names",
                  gem0_emio_input_names[i]);
              if (idx >= 0)
                      gem0_mux_parents[i + 1] = of_clk_get_parent_name(np,
                              idx);
      }
      clk = clk_register_mux(NULL, "gem0_mux", periph_parents, 4,
                      CLK_SET_RATE_NO_REPARENT, SLCR_GEM0_CLK_CTRL, 4, 2, 0,
                      &gem0clk_lock);
      clk = clk_register_divider(NULL, "gem0_div0", "gem0_mux", 0,
                      SLCR_GEM0_CLK_CTRL, 8, 6, CLK_DIVIDER_ONE_BASED |
                      CLK_DIVIDER_ALLOW_ZERO, &gem0clk_lock);
```

```
        clk = clk_register_divider(NULL, "gem0_div1", "gem0_div0",
                        CLK_SET_RATE_PARENT, SLCR_GEM0_CLK_CTRL, 20, 6,
                        CLK_DIVIDER_ONE_BASED | CLK_DIVIDER_ALLOW_ZERO,
                        &gem0clk_lock);
        clk = clk_register_mux(NULL, "gem0_emio_mux", gem0_mux_parents, 2,
                        CLK_SET_RATE_PARENT | CLK_SET_RATE_NO_REPARENT,
                        SLCR_GEM0_CLK_CTRL, 6, 1, 0,
                        &gem0clk_lock);
        clks[gem0] = clk_register_gate(NULL, clk_output_name[gem0],
                        "gem0_emio_mux", CLK_SET_RATE_PARENT,
                        SLCR_GEM0_CLK_CTRL, 0, 0, &gem0clk_lock);
  ...
  clks[gem0_aper] = clk_register_gate(NULL, clk_output_name[gem0_aper],
  clk_output_name[cpu_1x], 0, SLCR_APER_CLK_CTRL, 6, 0,
     &aperclk_lock);
```

Code Snippets 3.9: drivers/clk/zynq-l4/clkc.c

The same has to be done for every peripheral device, such us SDIO and USB.

3.2.7 Adding interrupt support

Finally interrupts need to be enabled. The IO server knows which interrupts each device has, from the zedboard.devs file, and provides access to Linux.

```
#ifndef __ASM_ARM__MACH_ZYNQ__IRQS_H__
#define __ASM_ARM__MACH_ZYNQ__IRQS_H__

#define NR_IRQS_HW      210

#include_next <mach/irqs.h>

#endif /* __ASM_ARM__MACH_ZYNQ__IRQS_H__ */
```

Code Snippets 3.10: arch/l4/include/asm/mach-arm/zynq/mach/irqs.h

We also need to make sure Linux will know where to find the L4 interrupt controller.

```
        intc: l4icu {
            compatible = "l4,icu";
            interrupt-controller;
            l4icu-type = "gic";
            #interrupt-cells = <3>;
                    #address-cells = <0>;
        };
```

Code Snippets 3.11: zynq.dts

3.2.8 Adding peripheral driver support to L4Linux for Zynq

Now that the peripheral clocks and interrupts work for L4Linux, the peripheral drivers need to be enabled for usage on L4Linux. Because of the approach used, the drivers think they are actually on bare metal system.

3.2.8.1 Device Controller

The first step is to enable the drivers for usage with the L4 architecture.

For example for the ethernet driver:

```
 config NET_VENDOR_XILINX
        bool "Xilinx devices"
        default y
-       depends on PPC || PPC32 || MICROBLAZE || ARCH_ZYNQ
+       depends on PPC || PPC32 || MICROBLAZE || ARCH_ZYNQ || ARCH_L4
        ---help---
          If you have a network (Ethernet) card belonging to this class, say Y.

@@ -18,7 +18,7 @@ if NET_VENDOR_XILINX

 config XILINX_EMACLITE
        tristate "Xilinx 10/100 Ethernet Lite support"
-       depends on (PPC32 || MICROBLAZE || ARCH_ZYNQ)
+       depends on (PPC32 || MICROBLAZE || ARCH_ZYNQ || ARCH_L4)
        select PHYLIB
        ---help---
          This driver supports the 10/100 Ethernet Lite from Xilinx.
```

Code Snippets 3.12: drivers/net/ethernet/xilinx/Kconfig

In appendix A the whole L4Linux configuration used can be found.

3.2.9 Adding a bram

Adding a bram to the system is simple. First we need to tell IO to give Linux access to the device, add it to the virtual bus and then inform Linux where to find it. On the Linux side we will use UIO to test the bram.

```
FPGA_BRAM = Hw.Device(function()
      compatible = {"generic-uio"};
      Property.hid  = "generic-uio";
      Resource.mem = Res.mmio(0x40000000, 0x40000fff);
end);
```

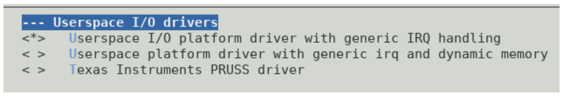
CODE SNIPPETS 3.13: zedboard.devs with Bram

```
bram = wrap(Hw.FPGA_BRAM);
```

CODE SNIPPETS 3.14: hw_devices.io with Bram

3.2.10 Access bram through uio

First of all we need to build the kernel with generic uio support.



```
--- Userspace I/O drivers
<*>   Userspace I/O platform driver with generic IRQ handling
< >   Userspace platform driver with generic irq and dynamic memory
< >   Texas Instruments PRUSS driver
```

FIGURE 3.2: Enabling Generic UIO driver in L4Linux

Finally we need to pass 'uio_pdrv_genirq.of_id=generic-uio' to the kernel bootargs in order for the kernel to match the bram with the uio generic driver.

```
rom/vmlinuz mem=312M console=ttyLv0 l4x_dtb=rom/zynq-zed.dtb
root=/dev/mmcblk0p2 rw rootwait uio_pdrv_genirq.of_id=generic-uio
```

CODE SNIPPETS 3.15: L4Linux bootargs for bram

And now the bram is ready to be used with a userspace driver.

Chapter 4

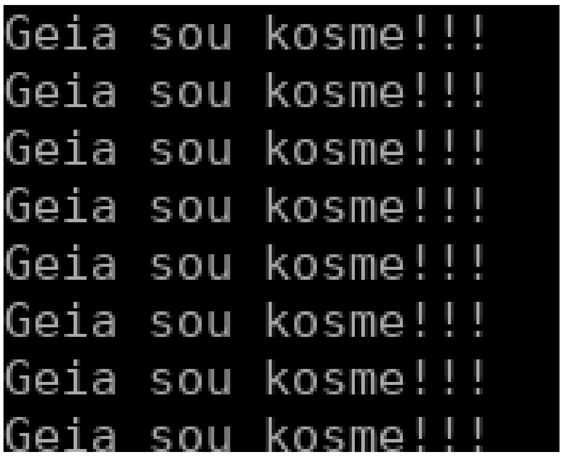
Results

This chapter contains proof that the system operates as expected. Proof for each of the run cases is given as well as code where necessary.

4.1 Testing

4.1.1 Simple Hello App

The first thing I did with L4/Fiasco was to have a simple server on top of Fiasco, continuesly printing 'Geia sou kosme!!!' on the serial port.



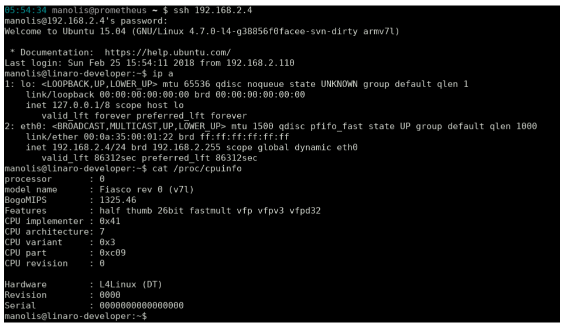
```
Geia sou kosme!!!
Geia sou kosme!!!
Geia sou kosme!!!
Geia sou kosme!!!
Geia sou kosme!!!
Geia sou kosme!!!
Geia sou kosme!!!
Geia sou kosme!!!
```

FIGURE 4.1: Fiasco Server Printing Geia sou kosme!!!

This was a proof that I could run a server on Fiasco. Next is L4Linux as a server.

4.1.2 L4Linux

L4Linux is binary compatible with ARM. As a result Linaro userspace built for armv7l can be used to have a full system. Bellow I connect to the on such system using SSH. Remember that L4Linux is running as a server on top of Fiasco.



```
05:54:34 manolis@prometheus ~ $ ssh 192.168.2.4
manolis@192.168.2.4's password:
Welcome to Ubuntu 15.04 (GNU/Linux 4.7.0-l4-g38856f0facee-svn-dirty armv7l)

 * Documentation:  https://help.ubuntu.com/
Last login: Sun Feb 25 15:54:11 2018 from 192.168.2.110
manolis@linaro-developer:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0a:35:00:01:22 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.4/24 brd 192.168.2.255 scope global dynamic eth0
       valid_lft 86312sec preferred_lft 86312sec
manolis@linaro-developer:~$ cat /proc/cpuinfo
processor       : 0
model name      : Fiasco rev 0 (v7l)
BogoMIPS        : 1325.46
Features        : half thumb 26bit fastmult vfp vfpv3 vfpd32
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x3
CPU part        : 0xc09
CPU revision    : 0

Hardware        : L4Linux (DT)
Revision        : 0000
Serial          : 0000000000000000
manolis@linaro-developer:~$
```

FIGURE 4.2: Connecting with SSH to L4Linux

As was stated before, L4/Fiasco abstracts the hardware and controls which parts Linux has access to.

4.1.3 Running Hello App and L4Linux concurrently

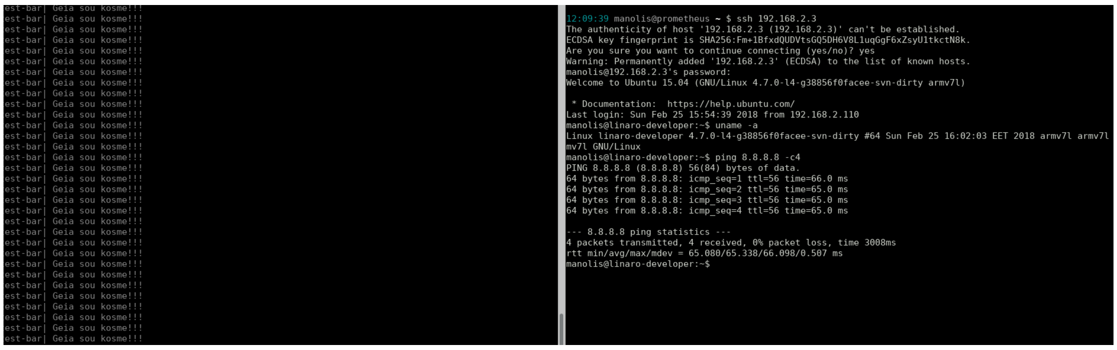
Running L4Linux with the Hello App concurrently is simple. We can tell Ned to start both servers, as shown bellow:

```
L4.default_loader:start({}, "rom/test-bare-hello");

L4.default_loader:start(
    { caps = {
           vbus = vbus_l4linux;
           },
           l4re_dbg = L4.Dbg.Warn,
           log = L4.Env.log:m("rws"),
    },
    "rom/vmlinuz mem=312M console=ttyLv0 " ..
    "l4x_dtb=rom/zynq-zed.dtb " ..
    "root=/dev/mmcblk0p2 rw rootwait " ..
    "uio_pdrv_genirq.of_id=generic-uio ");
```

CODE SNIPPETS 4.1: l4lx.cfg

The hello app will continuesly print on the uart a message, which uart is shared with L4Linux, and a user can use the system though ssh.



```
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
est-bar| Geia sou kosme!!!
```

```
12:09:39 manolis@prometheus ~ $ ssh 192.168.2.3
The authenticity of host '192.168.2.3 (192.168.2.3)' can't be established.
ECDSA key fingerprint is SHA256:Fm+1BfxdQUDVtsGQ5DH6V8L1uqGgF6xZsyU1tkctN8k.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.2.3' (ECDSA) to the list of known hosts.
manolis@192.168.2.3's password:
Welcome to Ubuntu 15.04 (GNU/Linux 4.7.0-l4-g38856f0facee-svn-dirty armv7l)

 * Documentation:  https://help.ubuntu.com/
Last login: Sun Feb 25 15:54:39 2018 from 192.168.2.110
manolis@linaro-developer:~$ uname -a
Linux linaro-developer 4.7.0-l4-g38856f0facee-svn-dirty #64 Sun Feb 25 16:02:03 EET 2018 armv7l armv7l
mv7l GNU/Linux
manolis@linaro-developer:~$ ping 8.8.8.8 -c4
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=56 time=66.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=56 time=65.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=56 time=65.0 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=56 time=65.0 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3008ms
rtt min/avg/max/mdev = 65.080/65.338/66.098/0.507 ms
manolis@linaro-developer:~$
```

Figure 4.3: L4Linux with Hello app

4.2 Comparisons with bare metal Linux

Because of the underlying L4/Fiasco core used, a Linux guest cannot have full access to the available memory. In our tests the guest L4Linux has access only to 312M of RAM. Moreover it has only access to parts of the system which the L4 configuration allows, as stated previously.

4.2.1 CPU Benchmarks

```
:~ sysbench --test=cpu --num-threads=1 run
Running the test with following options:
Number of threads: 1

Doing CPU performance benchmark

Threads started!
Done.

Maximum prime number checked in CPU test: 10000


Test execution summary:
    total time:                          341.1237s
    total number of events:              10000
    total time taken by event execution: 341.1105
    per-request statistics:
         min:                                 33.96ms
         avg:                                 34.11ms
         max:                                 39.32ms
         approx.  95 percentile:              34.14ms

Threads fairness:
    events (avg/stddev):           10000.0000/0.00
    execution time (avg/stddev):   341.1105/0.00
```

Code Snippets 4.2: Linux 3.19.0 CPU perfomance

```
:~ sysbench --test=cpu --num-threads=1 run
Running the test with following options:
Number of threads: 1

Doing CPU performance benchmark

Threads started!

Done.

Maximum prime number checked in CPU test: 10000


Test execution summary:
    total time:                          344.4229s
    total number of events:              10000
    total time taken by event execution: 344.3599
    per-request statistics:
         min:                                 33.99ms
         avg:                                 34.44ms
         max:                                131.99ms
         approx.  95 percentile:              34.98ms

Threads fairness:
    events (avg/stddev):           10000.0000/0.00
    execution time (avg/stddev):   344.3599/0.00
```

Code Snippets 4.3: L4Linux CPU perfomance

L4Linux does have a perfomance overhead on the cpu side, compared to bare metal Linux.

Table 4.1

GZIP compression	Average Time
L4Linux	713.82 s
Linux	653.24 s

Again the same can be seen with compression tests.

4.2.2 IO Benchmarks

This is a simple memory benchmark program, which tries to measure the peak bandwidth of sequential memory accesses and the latency of random memory accesses. Bandwidth is measured by running different assembly code for the aligned memory blocks and attempting different prefetch strategies.

Running memory bandwidth tests on bare metal Linux and L4Linux on L4.

```
C copy backwards                                    :    329.2 MB/s
C copy backwards (32 byte blocks)                   :    329.3 MB/s
C copy backwards (64 byte blocks)                   :    315.5 MB/s
C copy                                              :    327.1 MB/s
C copy prefetched (32 bytes step)                   :    508.0 MB/s
C copy prefetched (64 bytes step)                   :    330.1 MB/s
C 2-pass copy                                       :    259.7 MB/s
C 2-pass copy prefetched (32 bytes step)            :    436.7 MB/s
C 2-pass copy prefetched (64 bytes step)            :    388.9 MB/s
C fill                                              :   2124.3 MB/s
C fill (shuffle within 16 byte blocks)              :   2124.4 MB/s
C fill (shuffle within 32 byte blocks)              :   2124.2 MB/s
C fill (shuffle within 64 byte blocks)              :    546.7 MB/s
---
standard memcpy                                     :    355.2 MB/s
standard memset                                     :   2124.2 MB/s
---
ARM fill (STM with 8 registers)                     :   2124.3 MB/s
ARM fill (STM with 4 registers)                     :   2124.9 MB/s
```

Code Snippets 4.4: Linux 3.19.0 memory bandwidth more is better

```
C copy backwards                                    :    222.2 MB/s
C copy backwards (32 byte blocks)                   :    240.2 MB/s
C copy backwards (64 byte blocks)                   :    340.9 MB/s
C copy                                              :    332.2 MB/s
C copy prefetched (32 bytes step)                   :    506.4 MB/s
C copy prefetched (64 bytes step)                   :    339.9 MB/s
C 2-pass copy                                       :    300.7 MB/s
C 2-pass copy prefetched (32 bytes step)            :    436.6 MB/s
C 2-pass copy prefetched (64 bytes step)            :    456.1 MB/s
C fill                                              :   2120.3 MB/s
C fill (shuffle within 16 byte blocks)              :   2120.3 MB/s
C fill (shuffle within 32 byte blocks)              :   2120.3 MB/s
C fill (shuffle within 64 byte blocks)              :    578.5 MB/s
---
standard memcpy                                     :    356.4 MB/s
standard memset                                     :   2120.3 MB/s
---
ARM fill (STM with 8 registers)                     :   2122.4 MB/s
ARM fill (STM with 4 registers)                     :   2120.3 MB/s
```

Code Snippets 4.5: L4Linux memory bandwidth more is better

While in some cases there is a small drop in perfomance, in other cases we have almost identical perfomance.

Running memory latency tests on bare metal Linux and L4Linux on L4.

```
block size : single random read / dual random read
      1024 :    0.0 ns          /     0.0 ns
      2048 :    0.0 ns          /     0.0 ns
      4096 :    0.0 ns          /     0.0 ns
      8192 :    0.0 ns          /     0.0 ns
     16384 :    0.0 ns          /     0.0 ns
     32768 :    0.0 ns          /     0.2 ns
     65536 :   22.5 ns          /    33.8 ns
    131072 :   34.2 ns          /    42.2 ns
    262144 :   45.2 ns          /    50.2 ns
    524288 :   52.3 ns          /    55.4 ns
   1048576 :  110.3 ns          /   149.0 ns
   2097152 :  141.4 ns          /   180.1 ns
   4194304 :  158.5 ns          /   194.3 ns
   8388608 :  171.1 ns          /   208.5 ns
  16777216 :  184.7 ns          /   230.3 ns
  33554432 :  197.8 ns          /   254.1 ns
  67108864 :  211.5 ns          /   281.5 ns
```

Code Snippets 4.6: Linux 3.19.0 memory latency less is better

```
block size : single random read / dual random read
      1024 :    0.0 ns          /     0.0 ns
      2048 :    0.0 ns          /     0.0 ns
      4096 :    0.0 ns          /     0.0 ns
      8192 :    0.0 ns          /     0.0 ns
     16384 :    0.0 ns          /     0.0 ns
     32768 :    0.0 ns          /     0.3 ns
     65536 :   16.5 ns          /    25.6 ns
    131072 :   25.2 ns          /    32.8 ns
    262144 :   34.7 ns          /    43.8 ns
    524288 :   42.0 ns          /    52.1 ns
   1048576 :  100.9 ns          /   141.6 ns
   2097152 :  131.9 ns          /   171.9 ns
   4194304 :  149.2 ns          /   186.2 ns
   8388608 :  161.2 ns          /   198.8 ns
  16777216 :  172.6 ns          /   215.0 ns
  33554432 :  185.4 ns          /   237.5 ns
  67108864 :  198.7 ns          /   262.8 ns
```

Code Snippets 4.7: L4Linux memory latency less is better

Interestingly latency has actually decreased. The reason for this may be that the memory access is actually controlled by Fiasco an abstracted. Linux does not have to worry about the actual mechanism of reading from memory. As a result the reads end up being faster.

Now running block device related io benchmarks:

```
:~ sysbench --test=fileio --file-test-mode=seqwr --num-threads=1 run
Running the test with following options:
Number of threads: 1

Extra file open flags: 0
128 files, 16Mb each
2Gb total file size
Block size 16Kb
Periodic FSYNC enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing sequential write (creation) test
Threads started!
Done.

Operations performed:  0 Read, 131072 Write, 128 Other = 131200 Total
Read 0b  Written 2Gb  Total transferred 2Gb  (4.5296Mb/sec)
  289.90 Requests/sec executed

Test execution summary:
    total time:                          452.1343s
    total number of events:              131072
    total time taken by event execution: 434.4915
    per-request statistics:
         min:                                  0.11ms
         avg:                                  3.31ms
         max:                               5099.59ms
         approx.  95 percentile:              29.10ms

Threads fairness:
    events (avg/stddev):           131072.0000/0.00
    execution time (avg/stddev):   434.4915/0.00
```

Code Snippets 4.8: Linux 3.19.0 block device bandwidth and latency

```
:~ sysbench --test=fileio --file-test-mode=seqwr --num-threads=1 run
Running the test with following options:
Number of threads: 1

Extra file open flags: 0
128 files, 16Mb each
2Gb total file size
Block size 16Kb
Periodic FSYNC enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing sequential write (creation) test
Threads started!
WARNING: Operation time (0.000000) is less than minimal counted value, counting as 1.000000
```

```
WARNING: Percentile statistics will be inaccurate
Done.

Operations performed:  0 Read, 131072 Write, 128 Other = 131200 Total
Read 0b  Written 2Gb  Total transferred 2Gb  (6.8747Mb/sec)
  439.98 Requests/sec executed

Test execution summary:
    total time:                          297.9036s
    total number of events:              131072
    total time taken by event execution: 292.9806
    per-request statistics:
         min:                                  0.00ms
         avg:                                  2.24ms
         max:                               2476.99ms
         approx.  95 percentile:              19.99ms

Threads fairness:
    events (avg/stddev):           131072.0000/0.00
    execution time (avg/stddev):   292.9806/0.00
```

Code Snippets 4.9: L4 Linux block device bandwidth and latency

4.2.3 Math Operations Benchmarks

Using matmul, 4 different possible algorithms were tested. The naive, the transposed, sdot without hints and sdot with hints. We did not use any implementations not supported by our cpu (i.e SSE sdot). The argument '-n' is the size of the matrix.

Implementation	**Long description**
Naive	Most obvious implementation
Transposed	Transposing the second matrix for cache efficiency
sdot w/o hints	Replacing the inner loop with BLAS sdot()
sdot with hints	sdot() with a bit unrolled loop

Table 4.2: Matrix multiplication supported implementations

Algorithm	-a	Linux, -n2000	L4Linux, -n2000	Linux, -n4000	L4Linux, -n4000
Naive	0	345.037	328.272	493.473	391.752
Transposed	1	105.02	105.054	859.85	844.593
sdot w/o hints	4	109.642	105.045	870.752	844.618
sdot with hints	3	88.5898	88.673	714.472	714.278

Table 4.3: Matrix multiplication implementations less is better

As seen the perfomance was again the same or in some cases slighty better. Faster memory access, as show in previous tests above, does help increasing the perfomance.

4.2.4 Security and Resource Management

On a normal Linux we have access to all the devices available to the system:

```
linaro-developer
    description: Computer
    product: Zynq Zed Development Board
    width: 32 bits
  *-core
       description: Motherboard
       physical id: 0
       capabilities: xlnx_zynq-zed xlnx_zynq-7000
     *-cpu:0
          description: CPU
          product: cpu
          physical id: 0
          bus info: cpu@0
     *-cpu:1
          description: CPU
          product: cpu
          physical id: 1
          bus info: cpu@1
     *-memory
          description: System memory
          physical id: 2
          size: 499MiB
  *-network
       description: Ethernet interface
       physical id: 1
       logical name: eth0
       serial: 00:0a:35:00:01:22
       size: 1Gbit/s
       capacity: 1Gbit/s
       capabilities: ethernet physical tp mii 10bt 10bt-fd 100bt 100bt-fd 1000bt 1000bt-fd auton
       configuration: autonegotiation=on broadcast=yes driver=macb duplex=full ip=192.168.2.127
```

Code Snippets 4.10: lshw Linux

Naturally the L4Linux case we only have access to those parts we have given access to.

```
linaro-developer
    description: Computer
    product: L4Linux (DT)
    width: 32 bits
  *-core
       description: Motherboard
       physical id: 0
       capabilities: l4linux
     *-memory
          description: System memory
          physical id: 0
          size: 309MiB
     *-cpu
          physical id: 1
          bus info: cpu@0
  *-network
       description: Ethernet interface
       physical id: 1
       logical name: eth0
       serial: 00:0a:35:00:01:22
       size: 1Gbit/s
       capacity: 1Gbit/s
       capabilities: ethernet physical tp mii 10bt 10bt-fd 100bt 100bt-fd 1000bt-fd autonegotiat
       configuration: autonegotiation=on broadcast=yes driver=macb duplex=full ip=192.168.2.127
```

Code Snippets 4.11: lshw L4Linux

In L4Linux case, as we are still running the test applications on top of a Linux system, all the standard Linux application memory protections exist.

Chapter 5

Conclusions and Future Work

This chapter contains the conclusion of this thesis, possible improvements and future work.

5.1 Conclusions

This thesis proves that using Fiasco.oc as a hypervisor on an Arm system is indeed feasible and may even prove beneficial. Instead of running complex monolithic systems and use programs such as qemu, one can have a very simple microkernel with a small codebase, which can be verified much easier, and with a smaller point of failure.

5.2 Improvements and Future work

The are a lot of things that could be improved in this system. Ideally we want to have every resource managed from a different L4/Fiasco server and abstract them. As a result Linux will only have access to a virtual version of the devices, as if running on Qemu for example. Also it will allows us to share the same resources between different operating systems/apps running concurrently. The to-do list is as follows:

1. Reduce the L4/Fiasco inter-process communication overhead

2. Abstract Peripheral Devices (Clock Server, Network Server etc)
3. Modify L4Linux's drivers to use those server through inter-process communication
4. Being able to control system power settings from L4Linux
5. Reproduce this thesis on a stronger system than the Zedboard

5.2.1 Reduce the L4/Fiasco overhead

As we saw in chapter 4, there is a decrease in perfomance. A solution could be to measure which parts of the process introduces the bigest bottleneck and optimize it.

5.2.2 Abstract Peripheral Devices (Clock Server, Network Server etc)

Instead of actually having Linux drivers accessing the hardware, use L4 servers to control it, and have custom Linux drivers just communicate with them. This way we could also allow multiple guests to have access to the now shared resources.

5.2.3 Modify L4Linux's drivers to use those server through inter-process communication

This is actually a continuation of the previous point. In order to fully support this, the Linux driver subsystem will have to be rewritten to take advantage of the L4 server design.

5.2.4 Being able to control system power settings from L4Linux

Currently L4Linux cannot properly control the power of the board. As a result Linux cannot initiate a shutdown. First there is a need to create an authentication mechanism to determine which guests will have this capability, and then implement it. Second a server must be created which will accept requests from the guests to control the board power. Third, in case of Linux, a driver will need to be created to take care of communicating with the L4 server.

Another alternative is to use the Uvmm virtual machine monitor supported by L4Re, to control power events. Unfortunately with the approach the guest machine will have direct control over the hardware.

5.2.5 Reproduce this thesis on a stronger system than the Zedboard

Ideally this work could be reproduced simply in a stronger Arm system, as long as there is L4/Fiasco support for it. Depening on the availability of resources, L4/Fiasco should be able to scale and take advantage of them.

Appendix A

Appendix

```
CONFIG_L4=y
CONFIG_L4_ARCH_ARM=y
CONFIG_USE_OF=y
CONFIG_L4_PLATFORM_ZYNQ=y
CONFIG_L4_ARM_UPAGE_TLS=y
CONFIG_L4_SERVER=y
CONFIG_L4_VBUS=y
CONFIG_L4_BLK_DS_DRV=y
CONFIG_L4_CHR_DS_DRV=y
CONFIG_L4_EVENTS=y
CONFIG_L4_SERIAL=y
CONFIG_L4_SERIAL_CONSOLE=y
CONFIG_L4_SERIAL_SHM=y
CONFIG_L4_DEBUG=y
CONFIG_L4_DEBUG_REGISTER_NAMES=y
CONFIG_L4_DEBUG_SEGFAULTS=y
CONFIG_L4_VCPU=y
CONFIG_L4_CONFIG_CHECKS=y
CONFIG_L4_USE_L4SHMC=y
CONFIG_ARM=y
CONFIG_MIGHT_HAVE_PCI=y
CONFIG_SYS_SUPPORTS_APM_EMULATION=y
CONFIG_HAVE_PROC_CPU=y
CONFIG_STACKTRACE_SUPPORT=y
CONFIG_LOCKDEP_SUPPORT=y
CONFIG_TRACE_IRQFLAGS_SUPPORT=y
CONFIG_RWSEM_XCHGADD_ALGORITHM=y
CONFIG_FIX_EARLYCON_MEM=y
CONFIG_GENERIC_HWEIGHT=y
CONFIG_GENERIC_CALIBRATE_DELAY=y
CONFIG_NEED_DMA_MAP_STATE=y
CONFIG_ARCH_SUPPORTS_UPROBES=y
CONFIG_NEED_MACH_MEMORY_H=y
CONFIG_GENERIC_BUG=y
CONFIG_IRQ_WORK=y
CONFIG_BUILDTIME_EXTABLE_SORT=y
```

```
CONFIG_BROKEN_ON_SMP=y
CONFIG_LOCALVERSION_AUTO=y
CONFIG_HAVE_KERNEL_GZIP=y
CONFIG_HAVE_KERNEL_LZMA=y
CONFIG_HAVE_KERNEL_XZ=y
CONFIG_HAVE_KERNEL_LZO=y
CONFIG_HAVE_KERNEL_LZ4=y
CONFIG_KERNEL_GZIP=y
CONFIG_SWAP=y
CONFIG_SYSVIPC=y
CONFIG_SYSVIPC_SYSCTL=y
CONFIG_POSIX_MQUEUE=y
CONFIG_POSIX_MQUEUE_SYSCTL=y
CONFIG_CROSS_MEMORY_ATTACH=y
CONFIG_FHANDLE=y
CONFIG_GENERIC_IRQ_PROBE=y
CONFIG_GENERIC_IRQ_SHOW=y
CONFIG_GENERIC_IRQ_SHOW_LEVEL=y
CONFIG_HARDIRQS_SW_RESEND=y
CONFIG_IRQ_EDGE_EOI_HANDLER=y
CONFIG_IRQ_DOMAIN=y
CONFIG_HANDLE_DOMAIN_IRQ=y
CONFIG_IRQ_FORCED_THREADING=y
CONFIG_GENERIC_CLOCKEVENTS=y
CONFIG_HZ_PERIODIC=y
CONFIG_TICK_CPU_ACCOUNTING=y
CONFIG_TINY_RCU=y
CONFIG_SRCU=y
CONFIG_GENERIC_SCHED_CLOCK=y
CONFIG_CGROUPS=y
CONFIG_SYSFS_DEPRECATED=y
CONFIG_SYSFS_DEPRECATED_V2=y
CONFIG_BLK_DEV_INITRD=y
CONFIG_RD_GZIP=y
CONFIG_RD_BZIP2=y
CONFIG_RD_LZMA=y
CONFIG_RD_XZ=y
CONFIG_RD_LZO=y
CONFIG_RD_LZ4=y
CONFIG_CC_OPTIMIZE_FOR_PERFORMANCE=y
CONFIG_SYSCTL=y
CONFIG_ANON_INODES=y
CONFIG_HAVE_UID16=y
CONFIG_BPF=y
CONFIG_EXPERT=y
CONFIG_UID16=y
CONFIG_MULTIUSER=y
CONFIG_SYSFS_SYSCALL=y
CONFIG_SYSCTL_SYSCALL=y
CONFIG_KALLSYMS=y
CONFIG_KALLSYMS_BASE_RELATIVE=y
CONFIG_PRINTK=y
CONFIG_PRINTK_NMI=y
CONFIG_BUG=y
CONFIG_ELF_CORE=y
```

```
CONFIG_BASE_FULL=y
CONFIG_FUTEX=y
CONFIG_EPOLL=y
CONFIG_SIGNALFD=y
CONFIG_TIMERFD=y
CONFIG_EVENTFD=y
CONFIG_SHMEM=y
CONFIG_AIO=y
CONFIG_ADVISE_SYSCALLS=y
CONFIG_MEMBARRIER=y
CONFIG_EMBEDDED=y
CONFIG_HAVE_PERF_EVENTS=y
CONFIG_PERF_USE_VMALLOC=y
CONFIG_VM_EVENT_COUNTERS=y
CONFIG_COMPAT_BRK=y
CONFIG_SLAB=y
CONFIG_HAVE_OPROFILE=y
CONFIG_HAVE_EFFICIENT_UNALIGNED_ACCESS=y
CONFIG_ARCH_USE_BUILTIN_BSWAP=y
CONFIG_HAVE_KPROBES=y
CONFIG_HAVE_KRETPROBES=y
CONFIG_HAVE_OPTPROBES=y
CONFIG_HAVE_NMI=y
CONFIG_HAVE_ARCH_TRACEHOOK=y
CONFIG_HAVE_DMA_CONTIGUOUS=y
CONFIG_GENERIC_SMP_IDLE_THREAD=y
CONFIG_GENERIC_IDLE_POLL_SETUP=y
CONFIG_HAVE_REGS_AND_STACK_ACCESS_API=y
CONFIG_HAVE_CLK=y
CONFIG_HAVE_DMA_API_DEBUG=y
CONFIG_HAVE_PERF_REGS=y
CONFIG_HAVE_PERF_USER_STACK_DUMP=y
CONFIG_HAVE_ARCH_JUMP_LABEL=y
CONFIG_ARCH_WANT_IPC_PARSE_VERSION=y
CONFIG_HAVE_CC_STACKPROTECTOR=y
CONFIG_CC_STACKPROTECTOR_NONE=y
CONFIG_HAVE_CONTEXT_TRACKING=y
CONFIG_HAVE_VIRT_CPU_ACCOUNTING_GEN=y
CONFIG_HAVE_IRQ_TIME_ACCOUNTING=y
CONFIG_HAVE_MOD_ARCH_SPECIFIC=y
CONFIG_MODULES_USE_ELF_REL=y
CONFIG_ARCH_HAS_ELF_RANDOMIZE=y
CONFIG_HAVE_ARCH_MMAP_RND_BITS=y
CONFIG_HAVE_EXIT_THREAD=y
CONFIG_CLONE_BACKWARDS=y
CONFIG_OLD_SIGSUSPEND3=y
CONFIG_OLD_SIGACTION=y
CONFIG_ARCH_HAS_GCOV_PROFILE_ALL=y
CONFIG_HAVE_GENERIC_DMA_COHERENT=y
CONFIG_SLABINFO=y
CONFIG_RT_MUTEXES=y
CONFIG_MODULES=y
CONFIG_BLOCK=y
CONFIG_LBDAF=y
CONFIG_MSDOS_PARTITION=y
```

```
CONFIG_EFI_PARTITION=y
CONFIG_IOSCHED_NOOP=y
CONFIG_IOSCHED_DEADLINE=y
CONFIG_IOSCHED_CFQ=y
CONFIG_DEFAULT_CFQ=y
CONFIG_INLINE_SPIN_UNLOCK_IRQ=y
CONFIG_INLINE_READ_UNLOCK=y
CONFIG_INLINE_READ_UNLOCK_IRQ=y
CONFIG_INLINE_WRITE_UNLOCK=y
CONFIG_INLINE_WRITE_UNLOCK_IRQ=y
CONFIG_ARCH_SUPPORTS_ATOMIC_RMW=y
CONFIG_FREEZER=y
CONFIG_MMU=y
CONFIG_ARCH_L4=y
CONFIG_MACH_L4=y
CONFIG_L4_ARM_BUILD_FOR_V7=y
CONFIG_CPU_V7=y
CONFIG_CPU_32v6K=y
CONFIG_CPU_32v7=y
CONFIG_CPU_ABRT_EV7=y
CONFIG_CPU_PABRT_V7=y
CONFIG_CPU_CACHE_V7=y
CONFIG_CPU_CACHE_VIPT=y
CONFIG_CPU_COPY_V6=y
CONFIG_CPU_TLB_V7=y
CONFIG_CPU_HAS_ASID=y
CONFIG_CPU_CP15=y
CONFIG_CPU_CP15_MMU=y
CONFIG_ARM_THUMB=y
CONFIG_ARM_VIRT_EXT=y
CONFIG_KUSER_HELPERS=y
CONFIG_OUTER_CACHE=y
CONFIG_OUTER_CACHE_SYNC=y
CONFIG_MIGHT_HAVE_CACHE_L2X0=y
CONFIG_CACHE_L2X0=y
CONFIG_ARM_L1_CACHE_SHIFT_6=y
CONFIG_ARM_DMA_MEM_BUFFERABLE=y
CONFIG_ARM_HEAVY_MB=y
CONFIG_HAVE_SMP=y
CONFIG_VMSPLIT_3G=y
CONFIG_PREEMPT_NONE=y
CONFIG_HZ_100=y
CONFIG_ARM_PATCH_IDIV=y
CONFIG_AEABI=y
CONFIG_OABI_COMPAT=y
CONFIG_HAVE_ARCH_PFN_VALID=y
CONFIG_ARCH_WANT_GENERAL_HUGETLB=y
CONFIG_FLATMEM=y
CONFIG_FLAT_NODE_MEM_MAP=y
CONFIG_HAVE_MEMBLOCK=y
CONFIG_NO_BOOTMEM=y
CONFIG_NEED_PER_CPU_KM=y
CONFIG_GENERIC_EARLY_IOREMAP=y
CONFIG_ALIGNMENT_TRAP=y
CONFIG_SWIOTLB=y
```

```
CONFIG_IOMMU_HELPER=y
CONFIG_ATAGS=y
CONFIG_FPE_NWFPE=y
CONFIG_VFP=y
CONFIG_VFPv3=y
CONFIG_BINFMT_ELF=y
CONFIG_ELFCORE=y
CONFIG_BINFMT_SCRIPT=y
CONFIG_BINFMT_MISC=y
CONFIG_COREDUMP=y
CONFIG_SUSPEND=y
CONFIG_SUSPEND_FREEZER=y
CONFIG_PM_SLEEP=y
CONFIG_PM=y
CONFIG_PM_CLK=y
CONFIG_CPU_PM=y
CONFIG_ARCH_SUSPEND_POSSIBLE=y
CONFIG_ARM_CPU_SUSPEND=y
CONFIG_ARCH_HIBERNATION_POSSIBLE=y
CONFIG_NET=y
CONFIG_PACKET=y
CONFIG_UNIX=y
CONFIG_XFRM=y
CONFIG_INET=y
CONFIG_IP_PNP=y
CONFIG_IP_PNP_DHCP=y
CONFIG_INET_XFRM_MODE_TRANSPORT=y
CONFIG_INET_XFRM_MODE_TUNNEL=y
CONFIG_INET_XFRM_MODE_BEET=y
CONFIG_INET_DIAG=y
CONFIG_INET_TCP_DIAG=y
CONFIG_TCP_CONG_CUBIC=y
CONFIG_HAVE_NET_DSA=y
CONFIG_NET_RX_BUSY_POLL=y
CONFIG_BQL=y
CONFIG_WIRELESS=y
CONFIG_MAY_USE_DEVLINK=y
CONFIG_HAVE_CBPF_JIT=y
CONFIG_ARM_AMBA=y
CONFIG_UEVENT_HELPER=y
CONFIG_DEVTMPFS=y
CONFIG_DEVTMPFS_MOUNT=y
CONFIG_STANDALONE=y
CONFIG_PREVENT_FIRMWARE_BUILD=y
CONFIG_FW_LOADER=y
CONFIG_FIRMWARE_IN_KERNEL=y
CONFIG_ALLOW_DEV_COREDUMP=y
CONFIG_REGMAP=y
CONFIG_REGMAP_MMIO=y
CONFIG_DTC=y
CONFIG_OF=y
CONFIG_OF_FLATTREE=y
CONFIG_OF_EARLY_FLATTREE=y
CONFIG_OF_ADDRESS=y
CONFIG_OF_IRQ=y
```

```
CONFIG_OF_NET=y
CONFIG_OF_MDIO=y
CONFIG_OF_RESERVED_MEM=y
CONFIG_ARCH_MIGHT_HAVE_PC_PARPORT=y
CONFIG_BLK_DEV=y
CONFIG_BLK_DEV_RAM=y
CONFIG_SCSI_MOD=y
CONFIG_NETDEVICES=y
CONFIG_NET_CORE=y
CONFIG_ETHERNET=y
CONFIG_NET_VENDOR_ARC=y
CONFIG_NET_CADENCE=y
CONFIG_MACB=y
CONFIG_NET_VENDOR_BROADCOM=y
CONFIG_NET_VENDOR_CIRRUS=y
CONFIG_NET_VENDOR_EZCHIP=y
CONFIG_NET_VENDOR_FARADAY=y
CONFIG_NET_VENDOR_HISILICON=y
CONFIG_NET_VENDOR_INTEL=y
CONFIG_NET_VENDOR_I825XX=y
CONFIG_NET_VENDOR_MARVELL=y
CONFIG_NET_VENDOR_MICREL=y
CONFIG_NET_VENDOR_NATSEMI=y
CONFIG_NET_VENDOR_NETRONOME=y
CONFIG_NET_VENDOR_8390=y
CONFIG_NET_VENDOR_QUALCOMM=y
CONFIG_NET_VENDOR_RENESAS=y
CONFIG_NET_VENDOR_ROCKER=y
CONFIG_NET_VENDOR_SAMSUNG=y
CONFIG_NET_VENDOR_SEEQ=y
CONFIG_NET_VENDOR_SMSC=y
CONFIG_NET_VENDOR_STMICRO=y
CONFIG_NET_VENDOR_SYNOPSYS=y
CONFIG_NET_VENDOR_VIA=y
CONFIG_NET_VENDOR_WIZNET=y
CONFIG_NET_VENDOR_XILINX=y
CONFIG_PHYLIB=y
CONFIG_MARVELL_PHY=y
CONFIG_WLAN=y
CONFIG_WLAN_VENDOR_ADMTEK=y
CONFIG_WLAN_VENDOR_ATH=y
CONFIG_WLAN_VENDOR_ATMEL=y
CONFIG_WLAN_VENDOR_BROADCOM=y
CONFIG_WLAN_VENDOR_CISCO=y
CONFIG_WLAN_VENDOR_INTEL=y
CONFIG_WLAN_VENDOR_INTERSIL=y
CONFIG_WLAN_VENDOR_MARVELL=y
CONFIG_WLAN_VENDOR_MEDIATEK=y
CONFIG_WLAN_VENDOR_RALINK=y
CONFIG_WLAN_VENDOR_REALTEK=y
CONFIG_WLAN_VENDOR_RSI=y
CONFIG_WLAN_VENDOR_ST=y
CONFIG_WLAN_VENDOR_TI=y
CONFIG_WLAN_VENDOR_ZYDAS=y
CONFIG_INPUT=y
```

```
CONFIG_INPUT_MOUSEDEV=y
CONFIG_INPUT_MOUSEDEV_PSAUX=y
CONFIG_INPUT_EVDEV=y
CONFIG_TTY=y
CONFIG_VT=y
CONFIG_CONSOLE_TRANSLATIONS=y
CONFIG_VT_CONSOLE=y
CONFIG_VT_CONSOLE_SLEEP=y
CONFIG_HW_CONSOLE=y
CONFIG_VT_HW_CONSOLE_BINDING=y
CONFIG_UNIX98_PTYS=y
CONFIG_LEGACY_PTYS=y
CONFIG_DEVMEM=y
CONFIG_DEVKMEM=y
CONFIG_SERIAL_CORE=y
CONFIG_SERIAL_CORE_CONSOLE=y
CONFIG_ARCH_HAVE_CUSTOM_GPIO_H=y
CONFIG_SSB_POSSIBLE=y
CONFIG_BCMA_POSSIBLE=y
CONFIG_MFD_SYSCON=y
CONFIG_FB=y
CONFIG_FIRMWARE_EDID=y
CONFIG_FB_CMDLINE=y
CONFIG_FB_NOTIFY=y
CONFIG_DUMMY_CONSOLE=y
CONFIG_FRAMEBUFFER_CONSOLE=y
CONFIG_LOGO=y
CONFIG_LOGO_LINUX_MONO=y
CONFIG_LOGO_LINUX_VGA16=y
CONFIG_LOGO_LINUX_CLUT224=y
CONFIG_HID=y
CONFIG_HID_GENERIC=y
CONFIG_USB_OHCI_LITTLE_ENDIAN=y
CONFIG_MMC=y
CONFIG_PWRSEQ_EMMC=y
CONFIG_PWRSEQ_SIMPLE=y
CONFIG_MMC_BLOCK=y
CONFIG_MMC_BLOCK_BOUNCE=y
CONFIG_MMC_ARMMMCI=y
CONFIG_MMC_SDHCI=y
CONFIG_MMC_SDHCI_PLTFM=y
CONFIG_MMC_SDHCI_OF_ARASAN=y
CONFIG_MMC_MTK=y
CONFIG_EDAC_ATOMIC_SCRUB=y
CONFIG_EDAC_SUPPORT=y
CONFIG_EDAC=y
CONFIG_EDAC_LEGACY_SYSFS=y
CONFIG_RTC_LIB=y
CONFIG_DMADEVICES=y
CONFIG_DMADEVICES_DEBUG=y
CONFIG_DMADEVICES_VDEBUG=y
CONFIG_DMA_ENGINE=y
CONFIG_DMA_VIRTUAL_CHANNELS=y
CONFIG_DMA_OF=y
CONFIG_AXI_DMAC=y
```

```
CONFIG_UIO=y
CONFIG_UIO_PDRV_GENIRQ=y
CONFIG_CLKDEV_LOOKUP=y
CONFIG_HAVE_CLK_PREPARE=y
CONFIG_COMMON_CLK=y
CONFIG_IOMMU_SUPPORT=y
CONFIG_IRQCHIP=y
CONFIG_HAVE_ARM_SMCCC=y
CONFIG_DCACHE_WORD_ACCESS=y
CONFIG_EXT4_FS=y
CONFIG_EXT4_USE_FOR_EXT2=y
CONFIG_JBD2=y
CONFIG_FS_MBCACHE=y
CONFIG_EXPORTFS=y
CONFIG_FILE_LOCKING=y
CONFIG_MANDATORY_FILE_LOCKING=y
CONFIG_FSNOTIFY=y
CONFIG_DNOTIFY=y
CONFIG_INOTIFY_USER=y
CONFIG_AUTOFS4_FS=y
CONFIG_FAT_FS=y
CONFIG_VFAT_FS=y
CONFIG_PROC_FS=y
CONFIG_PROC_SYSCTL=y
CONFIG_PROC_PAGE_MONITOR=y
CONFIG_KERNFS=y
CONFIG_SYSFS=y
CONFIG_TMPFS=y
CONFIG_MISC_FILESYSTEMS=y
CONFIG_CRAMFS=y
CONFIG_NETWORK_FILESYSTEMS=y
CONFIG_NLS=y
CONFIG_DEBUG_INFO=y
CONFIG_ENABLE_WARN_DEPRECATED=y
CONFIG_ENABLE_MUST_CHECK=y
CONFIG_DEBUG_FS=y
CONFIG_SECTION_MISMATCH_WARN_ONLY=y
CONFIG_MAGIC_SYSRQ=y
CONFIG_DEBUG_KERNEL=y
CONFIG_HAVE_DEBUG_KMEMLEAK=y
CONFIG_DETECT_HUNG_TASK=y
CONFIG_SCHED_DEBUG=y
CONFIG_DEBUG_MUTEXES=y
CONFIG_DEBUG_BUGVERBOSE=y
CONFIG_HAVE_FUNCTION_TRACER=y
CONFIG_HAVE_FUNCTION_GRAPH_TRACER=y
CONFIG_HAVE_DYNAMIC_FTRACE=y
CONFIG_HAVE_FTRACE_MCOUNT_RECORD=y
CONFIG_HAVE_SYSCALL_TRACEPOINTS=y
CONFIG_HAVE_C_RECORDMCOUNT=y
CONFIG_TRACING_SUPPORT=y
CONFIG_FTRACE=y
CONFIG_BRANCH_PROFILE_NONE=y
CONFIG_HAVE_ARCH_KGDB=y
CONFIG_ARCH_HAS_DEVMEM_IS_ALLOWED=y
```

```
CONFIG_ARM_UNWIND=y
CONFIG_DEFAULT_SECURITY_DAC=y
CONFIG_CRYPTO=y
CONFIG_CRYPTO_ALGAPI=y
CONFIG_CRYPTO_ALGAPI2=y
CONFIG_CRYPTO_AEAD2=y
CONFIG_CRYPTO_BLKCIPHER2=y
CONFIG_CRYPTO_HASH=y
CONFIG_CRYPTO_HASH2=y
CONFIG_CRYPTO_RNG2=y
CONFIG_CRYPTO_AKCIPHER2=y
CONFIG_CRYPTO_MANAGER2=y
CONFIG_CRYPTO_MANAGER_DISABLE_TESTS=y
CONFIG_CRYPTO_NULL2=y
CONFIG_CRYPTO_WORKQUEUE=y
CONFIG_CRYPTO_CRC32C=y
CONFIG_CRYPTO_AES=y
CONFIG_CRYPTO_DRBG_HMAC=y
CONFIG_CRYPTO_HW=y
CONFIG_BITREVERSE=y
CONFIG_HAVE_ARCH_BITREVERSE=y
CONFIG_RATIONAL=y
CONFIG_GENERIC_NET_UTILS=y
CONFIG_GENERIC_PCI_IOMAP=y
CONFIG_GENERIC_IO=y
CONFIG_ARCH_USE_CMPXCHG_LOCKREF=y
CONFIG_CRC16=y
CONFIG_CRC32=y
CONFIG_CRC32_SLICEBY8=y
CONFIG_ZLIB_INFLATE=y
CONFIG_LZO_COMPRESS=y
CONFIG_LZO_DECOMPRESS=y
CONFIG_LZ4_DECOMPRESS=y
CONFIG_XZ_DEC=y
CONFIG_XZ_DEC_X86=y
CONFIG_XZ_DEC_POWERPC=y
CONFIG_XZ_DEC_IA64=y
CONFIG_XZ_DEC_ARM=y
CONFIG_XZ_DEC_ARMTHUMB=y
CONFIG_XZ_DEC_SPARC=y
CONFIG_XZ_DEC_BCJ=y
CONFIG_DECOMPRESS_GZIP=y
CONFIG_DECOMPRESS_BZIP2=y
CONFIG_DECOMPRESS_LZMA=y
CONFIG_DECOMPRESS_XZ=y
CONFIG_DECOMPRESS_LZO=y
CONFIG_DECOMPRESS_LZ4=y
CONFIG_GENERIC_ALLOCATOR=y
CONFIG_HAS_IOMEM=y
CONFIG_HAS_IOPORT_MAP=y
CONFIG_HAS_DMA=y
CONFIG_DQL=y
CONFIG_NLATTR=y
CONFIG_ARCH_HAS_ATOMIC64_DEC_IF_POSITIVE=y
CONFIG_LIBFDT=y
```

```
CONFIG_FONT_SUPPORT=y
CONFIG_FONT_8x8=y
CONFIG_FONT_8x16=y
```

Code Snippets A.1: L4Linux final build config

```
  // C program to multiply two square matrices.
#include <stdio.h>
#define N 4

// This function multiplies mat1[][] and mat2[][],
// and stores the result in res[][]
void multiply(int mat1[][N], int mat2[][N], int res[][N])
{
    int i, j, k;
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        {
            res[i][j] = 0;
            for (k = 0; k < N; k++)
                res[i][j] += mat1[i][k]*mat2[k][j];
        }
    }
}

int main()
{
    int mat1[N][N] = { {1, 1, 1, 1},
                    {2, 2, 2, 2},
                    {3, 3, 3, 3},
                    {4, 4, 4, 4}};

    int mat2[N][N] = { {1, 1, 1, 1},
                    {2, 2, 2, 2},
                    {3, 3, 3, 3},
                    {4, 4, 4, 4}};

    int res[N][N]; // To store result
    int i, j;
    multiply(mat1, mat2, res);

    printf("Result matrix is \n");
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
           printf("%d ", res[i][j]);
        printf("\n");
    }

    return 0;
}
```

Code Snippets A.2: Matrix multiplication in c

Bibliography

[1] Xilinx. Zynq-7000 all programmable soc. 1:1–4, 2016. URL https://www.xilinx.com/support/documentation/product-briefs/zynq-7000-product-brief.pdf.

[2] TU Dresden. The fiasco microkernel. *Project website*, page 1, June 2016. URL https://os.inf.tu-dresden.de/fiasco/.

[3] TU Dresden. The l4 runtime environment. *Project website*, page 1, June 2016. URL https://l4re.org/.

[4] Hermann Härtig, Michael Hohmuth, Jochen Liedtke, Jean Wolter, and Sebastian Schönberg. The performance of $µ$-kernel-based systems. *SIGOPS Oper. Syst. Rev.*, 31(5):66–77, October 1997. ISSN 0163-5980. doi: 10.1145/269005.266660. URL http://doi.acm.org/10.1145/269005.266660.

[5] Elena Reshetova, Janne Karhunen, Thomas Nyman, and N. Asokan. Security of os-level virtualization technologies. In Karin Bernsmed and Simone Fischer-Hübner, editors, *Secure IT Systems*, pages 77–93, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11599-3.

[6] Ahmed M. Azab, Kirk Swidowski, Rohan Bhutkar, Jia Ma, Wenbo Shen, Ruowen Wang, and Peng Ning. SKEE: A lightweight secure kernel-level execution environment for ARM. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016. URL http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/skee-lightweight-secure-kernel-level-execution-environment-for-arm.pdf.

[7] George Kornaros, Konstantinos Harteros, Ioannis Christoforakis, and Maria Astrinaki. I/O virtualization utilizing an efficient hardware system-level memory management unit. In Jari Nurmi, Peeter Ellervee, Dragomir Milojevic, Ondrej Daniel, and Tommi Paakki, editors, *2014 International Symposium on System-on-Chip, SoC 2014, Tampere, Finland, October 28-29, 2014*, pages 1–4. IEEE, 2014. ISBN 978-1-4799-6890-9. doi: 10.1109/ISSOC.2014.6972448. URL https://doi.org/10.1109/ISSOC.2014.6972448.

[8] George Kornaros, Miltos D. Grammatikakis, and Marcello Coppola. Towards full virtualization of heterogeneous noc-based multicore embedded architectures. In *15th IEEE International Conference on Computational Science and Engineering, CSE 2012, Paphos, Cyprus, December 5-7, 2012*, pages 345–352. IEEE Computer Society, 2012. ISBN 978-1-4673-5165-2. doi: 10.1109/ICCSE.2012.55. URL https://doi.org/10.1109/ICCSE.2012.55.

[9] George Kornaros, Ioannis Christoforakis, Othon Tomoutzoglou, Dimitrios Bakoyiannis, Kallia Vazakopoulou, Miltos D. Grammatikakis, and Antonis Papagrigoriou. Hardware support for cost-effective system-level protection in multi-core socs. In *2015 Euromicro Conference on Digital System Design, DSD 2015, Madeira, Portugal, August 26-28, 2015*, pages 41–48. IEEE Computer Society, 2015. ISBN 978-1-4673-8035-5. doi: 10.1109/DSD.2015.65. URL https://doi.org/10.1109/DSD.2015.65.

[10] Gianluca Durelli, Marcello Coppola, Karim Djafarian, George Kornaros, Antonio Miele, Michele Paolino, Oliver Pell, Christian Plessl, Marco D. Santambrogio, and Cristiana Bolchini. SAVE: towards efficient resource management in heterogeneous system architectures. In Diana Goehringer, Marco Domenico Santambrogio, João M. P. Cardoso, and Koen Bertels, editors, *Reconfigurable Computing: Architectures, Tools, and Applications - 10th International Symposium, ARC 2014, Vilamoura, Portugal, April 14-16, 2014. Proceedings*, volume 8405 of *Lecture Notes in Computer Science*, pages 337–344. Springer, 2014. ISBN 978-3-319-05959-4. doi: 10.1007/978-3-319-05960-0_38. URL https://doi.org/10.1007/978-3-319-05960-0_38.

[11] Marcello Coppola, Babak Falsafi, John Goodacre, and George Kornaros. From embedded multi-core socs to scale-out processors. In Enrico Macii, editor, *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, pages 947–951. EDA Consortium San Jose, CA, USA / ACM

DL, 2013. ISBN 978-1-4503-2153-2. doi: 10.7873/DATE.2013.199. URL https://doi.org/10.7873/DATE.2013.199.

[12] Yeongchann Han Taeung Song. L4/fiasco.oc & l4 linux – porting & device driver guide. *Personal Blog*, 1:4–11, August 2016. URL https://www.dropbox.com/s/os65yqiieay83hs/L4_Porting_and_Device_Driver_Guide.pdf.