LUT University

Asymmetric multiprocessing of Linux and hard real-time systems

Jouko Sinkkonen

Bachelor's thesis

6 September 2021

LUT School of Energy Systems

Electrical Engineering

ABSTRACT

Lappeenranta–Lahti University of Technology LUT
School of Energy Systems
Electrical Engineering

Jouko Sinkkonen

Asymmetric multiprocessing of Linux and hard real-time systems.

2021

Bachelor's thesis
25 pages.
Examiner: Associate professor Tuomo Lindh

Industrial control applications and lifesaving devices require hard real-time for them to function properly. While real-time operating systems and bare metal programs can meet these needs, their nature complicates the implementation of higher-level features like human machine interfaces. For this reason, general-purpose operating systems can be combined with real-time ones, forming asymmetric multiprocessing (AMP).

This thesis covered various tools and ways of creating AMP-supported systems and also went through the process of creating two such systems. The used platform was Digilent's development board Zybo Z7-10, which has a system on-chip called Zynq-7000 with a dual-core ARM processor and an FPGA. In the first practical example, Linux ran on the first core, loaded the bare metal program on the second one and the two communicated via the remote processor messaging framework. However, the bare metal did not do anything outside of waiting for messages to arrive from Linux and interrupts were not gotten to work so these things would have to be addressed next. The second example featured two bare metal programs running simultaneously on the Zynq and communicating via block RAM, located in the FPGA. Interrupts would be the next thing to implement for the dual bare metal system, but its lack of GPOS is a significant obstacle if higher level features were needed. Both systems serve as good starting points for developing AMP systems with working communication and remote launching capabilities, but they would still require more work to implement additional features that would be useful in real-life situations.

TIIVISTELMÄ

Lappeenrannan-Lahden teknillinen yliopisto LUT
School of Energy Systems
Sähkötekniikka

Jouko Sinkkonen

Asymmetric multiprocessing of Linux and hard real-time systems.

2021

Kandidaatintyö
25 sivua.
Tarkastaja: Tutkijaopettaja Tuomo Lindh

Teolliset säätöjärjestelmät ja hengenpelastuslaitteet vaativat kovaa reaaliaikaisuutta toimiakseen oikein. Reaaliaikaiset käyttöjärjestelmät ja bare metal -ohjelmat voivat vastata näihin tarpeisiin, mutta niiden luonne tekee korkean tason ominaisuuksien toteuttamisen hankalaksi. Tästä syystä voidaan muodostaa asymmetristä moniprosessointia (AMP) yhdistämällä yleiskäyttöiset käyttöjärjestelmät reaaliaikaisten kanssa.

Tämä työ tutustui erinäisiin työkaluihin, sekä tapoihin implementoida AMP ja kävi myös läpi kahden sellaisen järjestelmän luomisprosessin. Käytettynä laitteistona oli Digilentin valmistama Zybo Z7-10 kehitysalusta, jonka Zynq-7000 järjestelmäpiiri sisältää tuplaydin ARM prosessorin ja FPGA:n. Ensimmäisessä käytännön esimerkissä Linux pyöri ensimmäisellä ytimellä, latasi bare metal -ohjelman toiselle ytimelle ja ne kaksi kommunikoivat sen jälkeen remote processor messaging kehyksellä. Bare metal ei kuitenkaan tehnyt mitään viestien odottamisen ulkopuolella eikä keskeytyksiä saatu toimimaan, joten nämä asiat pitäisi korjata seuraavaksi. Toisessa esimerkissä kaksi eri bare metal -ohjelmaa suoritettiin samanaikaisesti Zynqillä ja ne kommunikoivat FPGA:han sijoitetun BRAM-lohkon kautta. Keskeytykset olivat seuraava askel tällekin järjestelmälle, mutta yleiskäyttöisen käyttöjärjestelmän puute on huomattava este, mikäli korkeamman tason ominaisuuksia haluttaisiin implementoida. Molemmat järjestelmät toimivat hyvinä lähtökohtina AMP järjestelmien luomiselle, mutta ne vaatisivat vielä työtä, jotta niihin saataisiin lisää oikean elämän sovelluksissa hyödyllisiä ominaisuuksia.

CONTENTS

Symbols and abbreviations

LIST OF SYMBOLS AND ABBREVIATIONS

AMP	Asymmetric Multiprocessing
BM1	Bare metal program running on an ARM processor's first core
BM2	Bare metal program running on an ARM processor's second core
BRAM	Block RAM
FPGA	Field-programmable gate array
FSBL	First-stage bootloader
GPOS	General-purpose operating system
HDMI	High-Definition Multimedia Interface
HMI	Human-Machine Interface
IoT	Internet of Things
OCM	On-Chip Memory
Rootfs	Root filesystem
RPMsg	Remote Processor Messaging
RTOS	Real-time operating system
SMP	Symmetric Multiprocessing
SoC	System on a chip
XSDK	Xilinx SDK
Zybo	Zybo Z7-10

1. INTRODUCTION

Connecting devices to the Internet is becoming increasingly commonplace, and this has given rise to the term Internet of Things (IoT). Internet connectivity can be a benefit in multiple domains, among them remote monitoring and process automation (Čolaković & Hadžialić, 2018). Human machine interface (HMI) is another feature that is required in various industry applications today (Chen et al., 2016). However, combining such high-level features with time-critical things like control systems can be a difficult task, that often requires two different kinds of software environments. An example of such a setup could be to have one software side handling the high-level things in a general-purpose operating system (GPOS) such as Windows or Linux and have the other side handle the control in a real-time operating system (RTOS) or a bare metal program. Having a GPOS and RTOS run in parallel on a multi-core processor is known as asymmetric multiprocessing (AMP).

This bachelor's thesis will be reviewing some important concepts related to AMP, available technologies and tools for creating AMP configurations and also covering some other alternatives for hard real-time systems. There are also references to similar systems that have been documented by researchers.

There will also be practical implementations of two different AMP configurations running on a development board called Zybo Z7-10, which has a system on a chip (SoC) called Zynq-7000 at its heart. The setup and used procedure will be discussed, as well as evaluating the results in terms of limitations and possible improvements.

2. BACKGROUND INFORMATION

2.1 Hard real-time

Programs on a microprocessor can be executed with varying degrees of urgency. Something like a music player running on a GPOS has a wholly different time requirement than a program that is used to guarantee human survival for instance. The music player can miss some instructions or not have them done immediately where lifesaving equipment cannot. Even outside of life and death situations, some control systems require immediate response

from the processor or else it could damage the system in question. The broad categories for separating these two scenarios are hard and soft real-time environments (Brown & Martin, 2010). GPOSs like Windows and Linux are meant for meeting soft real-time requirements, where it is not crucial to execute every instruction immediately. RTOSes like FreeRTOS or bare metal programs are meant for meeting hard real-time requirements. Bare metal refers to programs that are meant to be run on the processor without an operating system.

As computers get faster and faster, this might evoke the question of whether RTOSes are becoming obsolete. After all, if the CPU is fast enough, a GPOS could handle all its tasks quickly enough so that there would not be any discernible delay in real-world applications. A paper published in Indian Journal of Science and Technology in 2015, compared the performance of GPOS and RTOS with a mobile robot. The robot utilised an ARM11 processor and in this case the robot was found to have a response time of 20 μs with RTOS and 102 μs with GPOS (Murikipudi et al., 2015). So there does appear to still be need for RTOSes.

Besides raw latency there is also the question of cost. When mass producing products, the cost of components can quickly add up and having an excessively fast processor would be a waste of resources. An RTOS can also be relied upon to always process events in order of priority where a GPOS cannot.

Problem with using a strictly hard real-time environment, is that it requires a lot of work to implement higher level things – like an Internet connection or a graphical user interface – which is where the need arises to combine it with a soft real-time environment. In Linux and Windows for instance, Internet connectivity is often handled automatically with pre-installed software, thus saving the developer from having to create their own implementation. So, if a GPOS is available for the desired platform, the two could be combined to handle different things.

2.2 AMP and SMP

The key difference between AMP and symmetric multiprocessing (SMP) is that in AMP, different CPU cores can be running entirely different tasks under two different operating systems (or no operating system at all), whereas in SMP, all cores are running software under

the same operating system. AMP configurations thus allow for the building of a more flexible system, where some processor cores can be dedicated to running hard real-time tasks, while the rest are running instances of a GPOS.

2.3 Possible AMP configurations for hard real-time

2.3.1 Windows and TwinCAT

Windows has been used as a GPOS by Beckhoff in their TwinCAT product line, but since Windows does not support real-time execution, Beckhoff has had to develop their own kernel extension for it. This environment is usable on any Windows machine after purchasing a license and they have provided software that allows end-users to customize the real-time execution (Beckhoff Information System, n.d.a). TwinCAT works by getting a higher priority access to the processor than Windows, which allows it to execute certain software with a smaller latency. (Email, 2019) The minimum task cycle time you can use in TwinCAT is 50 µs, which means faster systems require some other implementation (Beckhoff Information System, n.d.b). If 50 µs is enough and Windows is available for your platform of choice, then TwinCAT could be an option since it is a ready-made system and provides you access to Beckhoff's customer support.

2.3.2 Linux and bare metal/RTOS

Since Linux is an open-source project, it has been compiled for more platforms than Windows. This also allows for more options in terms of hard real-time, since the user can write a bare metal program or utilize an RTOS, such as FreeRTOS and run these in an AMP configuration on two or more cores. An AMP system with Linux and bare metal was created in section 3.1 of this thesis. The same SoC and AMP setup have also been used for motion control purposes, with Linux handling the HMI and bare metal handling the control portion (Chen et al., 2016).

2.3.3 Dual bare metal/RTOS

In this scenario we have two different bare metal programs running on different cores. This setup naturally lacks the benefits a GPOS would bring, and so higher-level features will likely require more work to implement. A possibility in this case could be to have one of the bare metal programs communicate with a GPOS running on another computer, which then

handles the desired higher-level features. Dual bare metal might also be the only option if your desired platform does not have a GPOS available for it. An AMP system with dual bare metal was created in section 3.2 of this thesis.

2.4 PetaLinux

PetaLinux is a set of tools provided by Xilinx, which can be used to create embedded Linux solutions for their devices and was used for this purpose in this thesis' practical portion. It is available for various Linux distributions and can create the first-stage bootloader (FSBL) and u-boot for the Linux system. It can also handle the configuration and generation of the Linux kernel and a root filesystem (rootfs). (Xilinx Support Documentation, 2021a)

2.5 OpenAMP

OpenAMP is an open-source project that develops software for the creation of AMP-systems. It is contributed to by many different people and companies, among them Xilinx. In addition to other things, it provides an application programming interface (API) to a framework called remoteproc, which can be used to launch software on remote processors. It also provides an API to a messaging bus called RPMsg (Remote Processor Messaging) that can be used by Linux and bare metal to communicate with software running on remote cores. (Xilinx Wiki, 2019a)

OpenAMP has been utilized in a Zynq-based robot control system with Linux and FreeRTOS; the researchers used remoteproc to launch the FreeRTOS and RPMsg for communication between the two cores (Sun et al., 2019). OpenAMP was used for the same purpose in this thesis' practical work.

2.6 Hard real-time Linux

There are options for running hard real-time programs alongside Linux without the use of bare metal or an RTOS, thus bypassing the need for AMP entirely. Xilinx has mentions for running at least Xenomai on Zynq on their website but testing these out would have gone outside the scope of this thesis (Xilinx Wiki, 2020b).

Recent kernel patches have introduced real-time functionality to Linux with CONFIG_PREEMPT, which allows for most kernel operations to be pre-empted. So, if a program requires Linux to respond quickly, the currently executing task will be put on hold until the real-time operation finishes. Some kernel operations like interrupt service routines or spinlocked threads (thread that waits for a predetermined time for a resource to free up) still cannot be pre-empted, which is where PREEMPT_RT comes in. It allows for even harder real-time constraints to be met by allowing interrupts and spinlocks to be pre-empted. In the interrupts' case, this is achieved by handling the interrupt service routines as regular threads, which means that they are scheduled. (Brown & Martin, 2010)

Xenomai is another option and has two different working modes. The dual-kernel configuration works by running a real-time kernel called "Cobalt" alongside Linux. It takes priority over the regular kernel and handles things like interrupts and scheduling of real-time threads. Any real-time applications then interface with Cobalt via Xenomai APIs. The single kernel setup relies on the PREEMPT_RT patches of the Linux kernel to deliver real-time and the Xenomai APIs are then emulated to Linux's native threading library. (Xenomai Wiki, 2019)

3. PRACTICAL WORK

The platform used for these demonstrations is Zynq-7000 on a Zybo Z7-10 board, which can be seen in figure 3.1. Zynq-7000 is an SoC that has a dual-core ARM Cortex-A9 processor and a 28 nm Artix-7 field-programmable gate array (FPGA). Zybo Z7-10 will be referenced as Zybo, from here on in. For clarification, ZYBO is also the name of an older Digilent board, but Zybo Z7-10 is a different one and simply referenced as Zybo for short in this thesis.

Zybo was chosen because it was readily available at the university. One of the university courses also uses it as a platform for teaching embedded system programming, and so these projects could be used as a basis for an assignment or something similar. Zynq-7000 contains both a microprocessor and an FPGA, so it is a versatile platform for developing embedded systems.

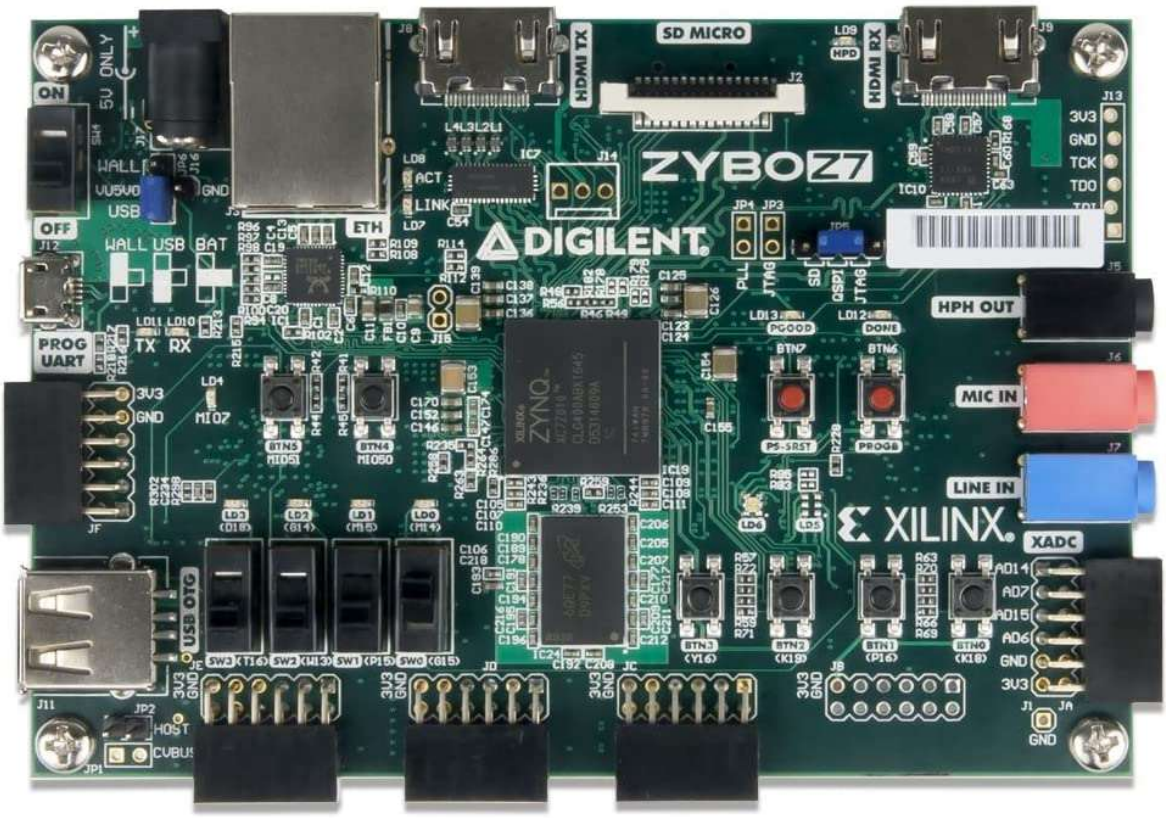

Figure 3.1 Zybo Z7-10, a development board made by Digilent. It contains an SoC called Zynq-7000 (in the middle), manufactured by Xilinx, which contains a dual-core ARM Cortex-A9 processor and a 28 nm Artix-7 FPGA. This board was used in this thesis' technical implementations of AMP systems.

The first practical work created an AMP system with Linux and bare metal running on two different cores and communicating with each other using RPMsg. With Xilinx providing tools for building Linux for their platforms, this kind of an AMP system was a natural choice. The second practical work created an AMP system with two different bare metal programs running on two different processor cores with communication handled through the FPGA. This kind of a system is useful because the two can be doing entirely different things while still retaining communication.

3.1 Linux and bare metal

This AMP system has the following specifications:

- One ARM core will be running Linux, another one a bare metal program.
- Linux launches the bare metal program using remoteproc.
- The Linux program invokes the bare metal program with RPMsg, and the bare metal sends back the state of one of the switches on the Zybo board.

- The Linux program receives the data and writes it to a file that the user can then read to verify that the connection between the two cores is working.

An overview of the system can be seen in figure 3.2.

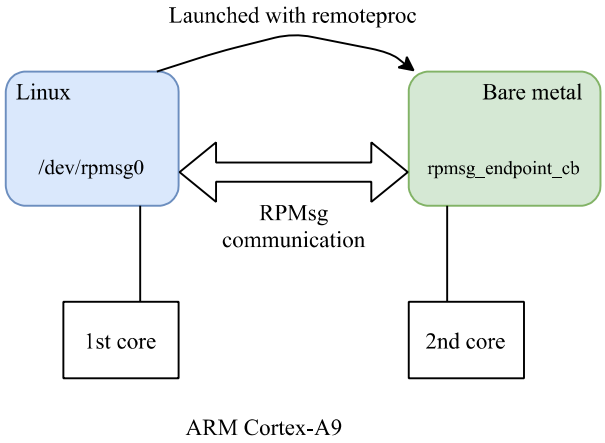

Figure 3.2 An overview of the finished AMP system with Linux running on the ARM processor's first core and bare metal on the second one. Linux boots up first and launches the bare metal program on the second core with remoteproc. The two then communicate with RPMsg; Linux writes to a file called /dev/rpmsg0, and this causes the bare metal to execute a function called rpmsg_endpoint_cb. Bare metal then sends back information through RPMsg, and Linux receives it in the same file.

This system only ensures that basic functionality related to launching of the bare metal program and communication between the two cores is properly handled. Nothing more complex is implemented.

3.1.1 The setup

The FPGA portion of Zynq-7000 needs to be configured every time the Zybo is turned on. This is done with a bitstream file that is created with Vivado; a program developed by Xilinx for designing FPGAs. Digilent, the company that manufactures Zybo, provides their own Vivado project on GitHub and this was used as a base with some modifications. However, these modifications included things like audio processing and thus did not impact this setup.

The bare metal-program was modified from an OpenAMP sample program that was included as a template in Xilinx SDK (XSDK). The program sets up an RPMsg endpoint, which can then be connected to from within Linux. The RPMsg endpoint is a function, that gets invoked every time it is connected to, and it could contain things like reporting the status of a certain

register. In this case the bare metal reports the status of one of the switches on the Zybo board. The Linux program was also modified from a sample OpenAMP demo program provided by Xilinx. The original one was used for testing the integrity of data sent to and from the remote processor, and it was reduced to the bare minimum required for sending and receiving data via RPMsg. The Linux program will receive the switch's status and write it to a file.

To boot a Linux from an SD card on the Zybo, the SD card needs to contain at least two files on the first partition:

1. BOOT.BIN, which contains the FSBL, bitstream file and u-boot. (Xilinx Wiki, 2020c)
2. Image.ub, u-boot supported image that contains a minimal rootfs, Linux kernel image and a device tree. (Xilinx Wiki, 2019c)

Optionally the SD card can also contain a second partition that has a premade Linux filesystem on it.

Digilent hosts a ready-made PetaLinux project for their Vivado project on GitHub. Said Vivado project was modified earlier, and these modifications were carried over by reconfiguring the PetaLinux project with the modified project's hardware description file, generated by Vivado and containing a blueprint of the FPGA design. Digilent's PetaLinux project was created for PetaLinux version 2017.4, so this version was also used in this project.

PetaLinux has commands for building the two previously mentioned files and handles the creation of the FSBL, u-boot, rootfs and kernel as well. The kernel was configured with PetaLinux to support the loading of userspace drivers, which was done for RPMsg. Zynq's remoteproc implementation was also added. The kernel's configuration window is shown in figure 3.3.

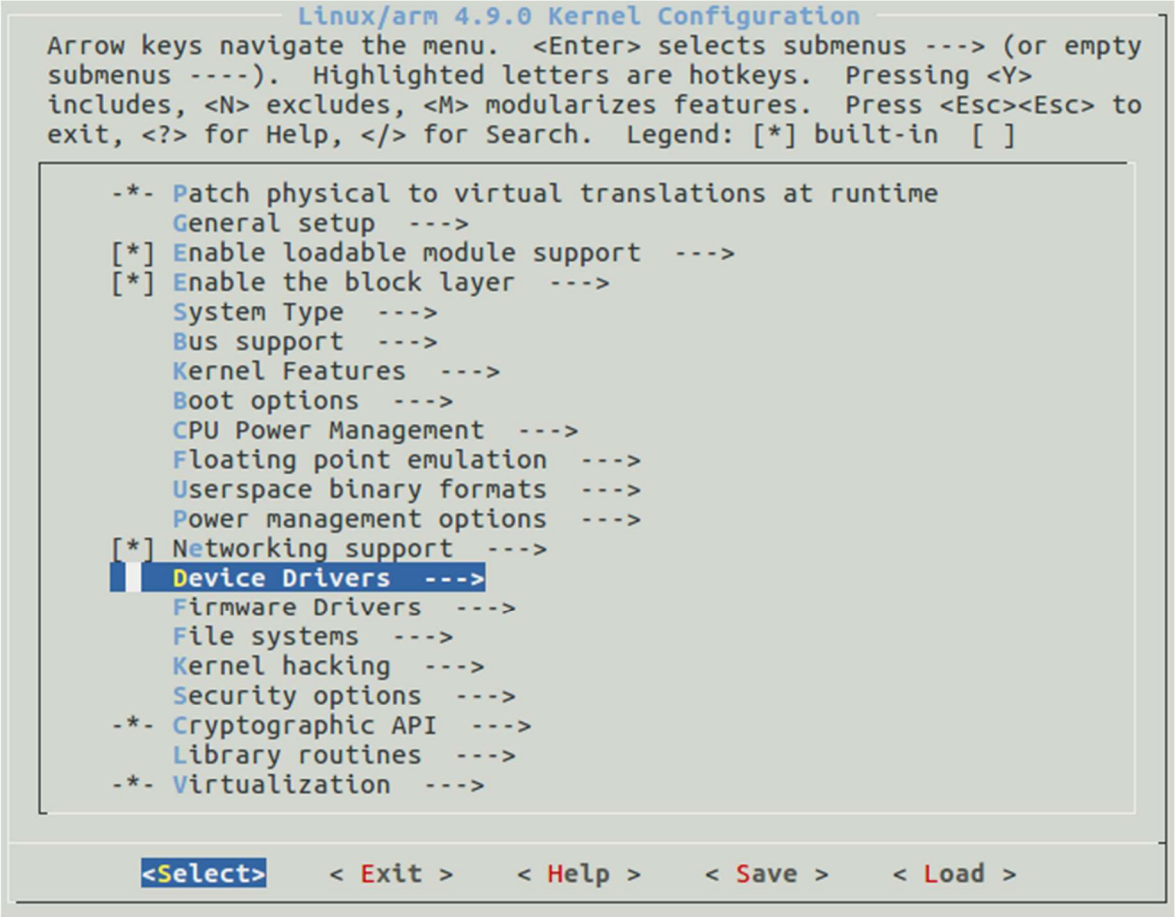

Figure 3.3 The kernel configuration window that was opened via PetaLinux. The kernel was configured to support the loading of userspace firmware drivers and Zynq's remoteproc implementation.

The default option for the rootfs was to create it in RAM with INITRAMFS but this would have meant that all changes to the rootfs would have been reset on power off. INITRAMFS would use the minimal rootfs from the image.ub-file. Instead, the system's boot arguments were configured so that it would load the rootfs from the SD card's second partition. The used rootfs was configured with PetaLinux to contain the necessary OpenAMP filesystem packages and generated along with BOOT.BIN and image.ub.

Device tree gives a description of the hardware to the Linux, and it needs to be customized by the user (Xilinx Wiki, 2019b). Digilent's PetaLinux project's device tree was pre-configured to support things like High-Definition Multimedia Interface (HDMI) and Ethernet, so it only required some additions for remoteproc-related memory management. Note that the device tree only needs to contain components you want to access from Linux; the bare metal can access everything right away through the registers, if they are mapped within the hardware description file. This was tested and found to be the case.

Finally, the BOOT.BIN and image.ub were built with PetaLinux and placed on the SD card's first partition, which was formatted in FAT32. PetaLinux created an image of the rootfs in ext4, and this was burnt on the SD card's second partition, so that it will be loaded when booting the system. Linux can be accessed via a serial terminal on another computer or directly with a keyboard and HDMI display connected to Zybo.

3.1.2 The procedure

Linux is started on the first CPU core and OpenAMP's RPMsg driver is loaded into the kernel with the "modprobe" command. The bare metal program for the second core is within the rootfs, and its path is written to a file called "firmware" in the remoteproc's directory. The same directory also has a file called "state", which is used to start remoteproc, at which point the program specified in "firmware" gets loaded on the second core.

The bare metal program launches and sets up an RPMsg endpoint. Now the RPMsg bus is accessible on the Linux with a device file called rpmsg0. This file is used for both sending and receiving information. A program is started in Linux, that both writes to and reads from that file, and then outputs the read information to a separate file, which can then be used to verify that the connection is working as expected.

3.1.3 Results

The Linux program successfully created a file that reflects changes in one of the switches on the Zybo board as can be seen from figure 3.4. Since the switch's state was only read from the bare metal program running on the second core, the communication between the two must be working.

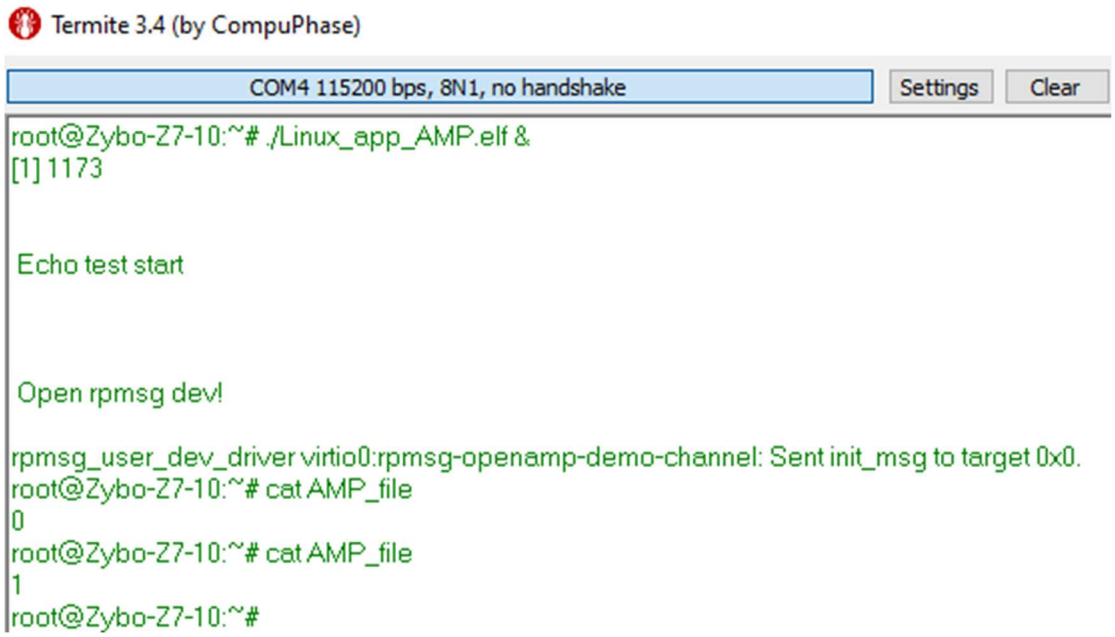

Figure 3.4 Screenshot of a serial terminal while logged into the Linux running on the Zybo board. The RPMsg driver has already been loaded into the kernel and the bare metal program launched on the second core with remoteproc. Linux_app_AMP.elf is the Linux program that sends RPMsg messages to the second core and then writes the received message to the file AMP_file. AMP_file reflects changes in one of the switches on the Zybo board and its state is only read by the bare metal program so communication between the two must be working.

3.1.4 Evaluation

The bare metal program needs to be launched from within Linux and it currently requires multiple commands to do, so the system cannot just be switched on. During the booting process there appears to be an attempt to load the bare metal program directly on the second core with remoteproc, which might be a possible fix. Another fix would be creating a shell script with the necessary commands and having that run after Linux starts.

Currently only the RPMsg works for communication between the two cores but another option would be to use the FPGA for communication by placing a block RAM (BRAM) within it. Xilinx has documentation for BRAM access from within Linux on their website, so this must be possible (Xilinx Wiki, 2020a). Initially this was tested by adding a BRAM block to the FPGA design and doing the necessary device tree modifications to make it usable from Linux. However, when attempting to write to the BRAM's memory address with "devmem", Linux encounters an error. After this a new FPGA design was made in Vivado to more closely follow Xilinx's instructions but this version would not boot on Zybo. Since Xilinx's instructions were made for a different Zynq, it is likely that some of the steps

would need to be done differently. Besides a BRAM block in the FPGA, Xilinx has documentation for using Zynq's on-chip memory (OCM) for communication between the two cores, which is another option worth looking into (Xilinx Support Documentation, 2013).

Currently the bare metal program only executes a function when it receives an RPMsg message and rest of the time runs in a while-loop doing nothing. This strips a lot of usability from the bare metal and would need to be addressed. Timer interrupts would be useful for scheduling tasks or running something like a control algorithm at a desired frequency. Timer interrupts were tested but not gotten to work.

3.2 Dual bare metal

This AMP system has the following specifications:

- The Zynq's ARM cores will be running two different bare metal programs. The first core's program will be referred to as BM1 and the second core's program as BM2 from now on.
- The programs communicate using BRAM that has been placed in FPGA.
- BM1 writes a value to BRAM, which is then read by BM2. BM2 prints the read value to a serial terminal and then writes the same value multiplied by 10 back to the BRAM. BM1 then reads this value and prints it to the same serial terminal. The communication can be verified to be working by checking the printed values.

An overview of the system can be seen in figure 3.5.

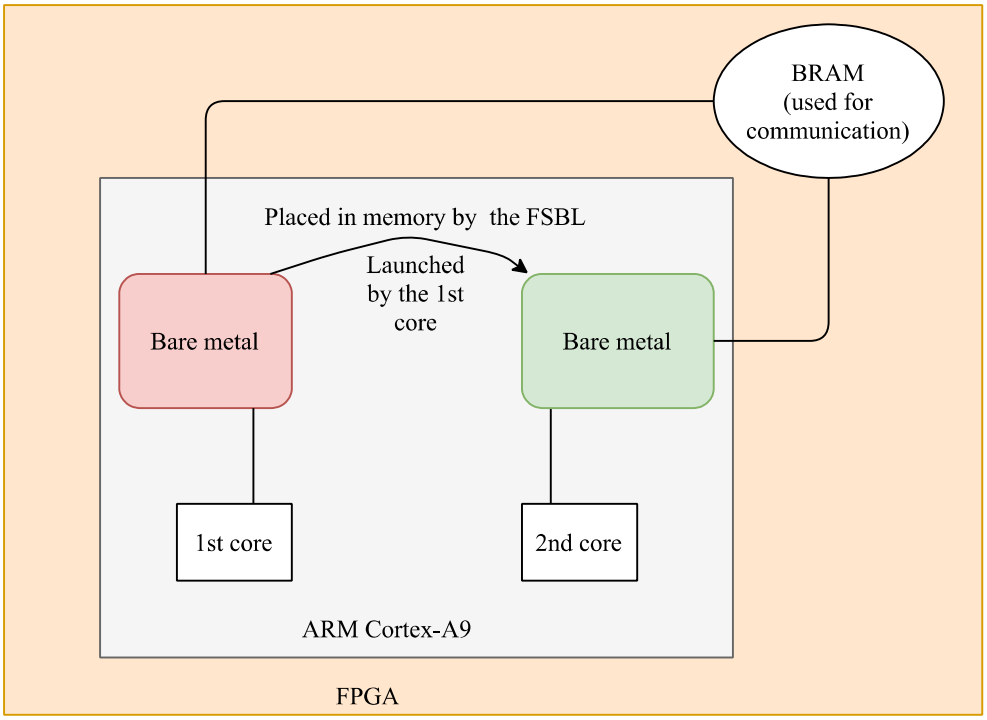

Figure 3.5 An overview of the created dual bare metal system. The FSBL launches the first core's bare metal program and loads the second one into memory. The second one is launched by the first one and the two then communicate via BRAM that has been placed in the FPGA.

The two bare metal programs only do what is necessary to verify that they are communicating successfully; nothing more complex has been implemented.

3.2.1 The setup

Since Linux is not involved this time, only the BOOT.BIN needs to be created. It again consists of the FSBL and bitstream at the start. Instead of u-boot however, it next contains BM1 and BM2, respectively. FSBL was created using a template located in the XSDK and was used without modifications. The FPGA design remains the same as in the previous setup and so the bitstream file is also the same.

When compiling an executable file, a text-file called linker script determines how the program's various parts will be situated in memory. BM1's linker script was modified so that all of the executable's sections were mapped to the memory region 0x100000 and BM2's script so that the sections were mapped to the region 0x200000 as can be seen from figure 3.6. This way it can be ensured that they do not overlap, and their memory locations are known.

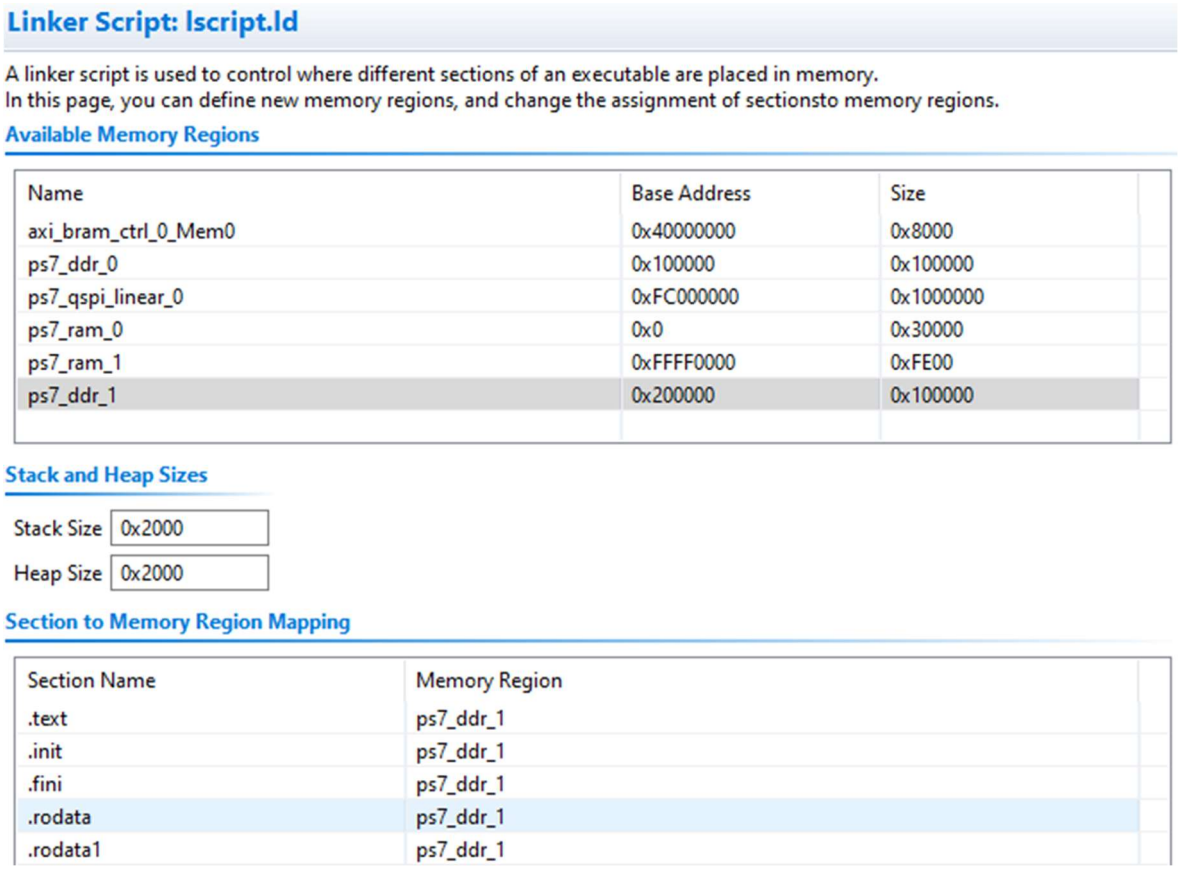

Figure 3.6 The second bare metal program's linker script open in XSDK's linker script editor. Linker script specifies where an executable's different sections are to be loaded in memory. A new available memory region was added, and all the executable's various sections were mapped to that same address, in this case 0x200000. Not all of the executable's sections are shown here.

The two programs communicate using a BRAM block that has been placed in the FPGA. Unlike the issues encountered in 3.1.4 with Linux, bare metal programs can utilize the BRAM block without any difficulty, so this was achieved by simply having the two programs write to the BRAM block's memory address (0x40000000). BM1 writes an 8-bit unsigned integer value to the BRAM and BM2 reads the value and prints it to the user via UART. BM2 then multiplies the received value by 10 and sends it back to BM1, which then also prints the value via UART.

3.2.2 The procedure

BM1 is launched by the FSBL, which also loads BM2 into memory. The second ARM core loads a small program from 0xFFFFFE00 in OCM. This program is a loop that waits for an event and then checks if the address 0xFFFFFFF0 contains a non-zero value. If it finds

something else, the second core jumps to that memory address. If it finds a zero, then the loop starts over. (Xilinx Support Documentation, 2014)

Normally when running a program only on the first core, the second one never does anything because even if an event happens, it only finds a zero in 0xFFFFFFF0. BM2 is stored in 0x200000 in memory, so this address is written to 0xFFFFFFF0 by BM1. Now when BM1 calls the ARM assembly command “Set Event” (sev), the second ARM core’s loop-program detects an event, sees a non-zero value in 0xFFFFFFF0, loads BM2’s address 0x200000 and starts executing the program.

Since BM2 outputs the value it reads from BRAM via UART and BM1 outputs the same value multiplied by 10 via UART, the connection can be verified to be working if they both print the desired information.

3.2.3 Results

As can be seen from figure 3.7, the serial terminal shows BM2 printing the value it reads and BM1 then printing the same value multiplied by 10, which means that the two are successfully communicating via BRAM.

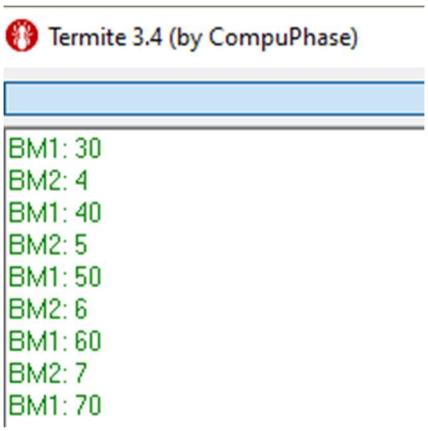

Figure 3.7 Screenshot of a serial terminal with the output of the two bare metal programs shown. BM2 outputs the number that was written to the BRAM by BM1 and BM1 outputs the same number multiplied by 10, which was written to the BRAM by BM2. Therefore, the two must be communicating via BRAM.

3.2.4 Evaluation

As discussed previously, the lack of a GPOS in this setup complicates the implementation of higher-level features. Unless there are pre-made bare metal programs for the used platform

to handle something like networking, it would have to developed from the ground up. Zybo could also be connected to another

Instead of a strict bare metal program, this same setup can also be used to run two FreeRTOS instances simultaneously. FreeRTOS being essentially a software library with readymade implementations for Zynq, one only needs to include it and have the first core's program do the same initialization as BM1. This provides a good scheduler for tasks but does not help with the lack of higher-level features.

Interrupts would be a good next thing to implement for this system. Timer interrupts were tested for both cores and worked for BM1 without any issues, but they did not work for BM2. The interrupts were generated with triple timer counters, and these are shared peripheral interrupts so they can be routed to generate interrupts for either one or both ARM cores (Xilinx Support Documentation, 2021b). This suggests that it is certainly possible to have timer interrupts for both cores at the same or different frequency and would just require more configuration. Also, since FreeRTOS worked fine on both cores, timer interrupts must be working in its case as well.

4. CONCLUSION

This thesis covered some technical information about AMP systems, various ways of implementing them and presented two practical examples of such systems. While the two systems did only display rudimentary functionality with reporting the state of switches or multiplying numbers, some important underlying achievements were made. Two cores of an ARM processor were made to run two different pieces of software and communication between the two was ensured. These serve as good starting points for future advancements in creating AMP systems on the Zybo Z7-10.

The Linux and bare metal system in particular would require work since the bare metal currently does nothing outside of receiving RPMsg messages. Functionality could be added with interrupts, which are currently unsupported. This would require investigating whether or not they work along with RPMsg, or if another communication method would be required, such as through the BRAM in the FPGA or via Zynq's OCM.

The dual bare metal system's lack of a GPOS hurts its usability in terms of developing higher level features, so it would likely be useful in a situation where there is only need for two different kinds of hard real-time applications. One option could also be to connect the Zybo to another computer that does have a GPOS installed and then using that for HMI or Internet connectivity. Regardless, the system would next need to configure interrupts for both cores as so far, they were only gotten to work on the first one. Interrupt support for both should be possible with a little work.

A hard real-time system is still often necessary in time critical applications to ensure proper functionality, but it can be hard to implement higher level features in such an environment. Combining hard real-time with a GPOS can also require work. Depending on your platform and needs, it can be as simple as installing TwinCAT on a Windows computer or as tricky and multifaceted as the Linux and bare metal system covered in this thesis. But in the long run such systems also present significant payoffs when integrating the Internet and HMI with industrial applications.

REFERENCES

Beckhoff Information System. n.d.a. *Real time without additional hardware and as a system basis*, viewed on 3 October 2019. Available at: <https://infosys.beckhoff.com/english.php?content=../content/1033/tcsystemover/html/tcsystemover_pcctrlrt.htm>

Beckhoff Information System. n.d.b. *Real Time Settings*, viewed on 4 December 2019. Available at: <https://infosys.beckhoff.com/english.php?content=../content/1033/tcsystemmanager/basics/TcSysMgr_ConfigRT_Intro2.htm>

Brown, J. & Martin, B. 2010. *How fast is enough? Choosing between Xenomai and Linux for real-time applications*, viewed on 1 September 2021. Available at: <https://pdfs.semanticscholar.org/9eb5/1dbe38fb23034e80b8664d8281996d2a5ef6.pdf>

Čolaković, A & Hadžialić, M. 2018. *Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues*. Computer Networks (The International Journal of Computer and Telecommunications Networking), volume 144. Oct 24.

Email received from Beckhoff's Control Automation Specialist Teppo Lepistö on 10 October 2019.

Chen, X., Gu, Y., Wang, C. & Guan, X. 2016. *Asymmetric Multiprocessing for Motion Control Based on Zynq SoC*. IEEE, International Conference on Field-Programmable Technology. Xi'an, China. Dec 7-9.

Murikipudi, A., Prakash, V., Vigneswaran. 2015. *Performance Analysis of Real Time Operating System with General Purpose Operating System for Mobile Robotic System*. Indian Journal of Science and Technology, volume 8.

Sun, Y., Li, E., Yang, G., Liang, Z. & Guo, R. 2019. *Design of a Dual-core Processor Based Controller with RTOS-GPOS Dual Operating System*. IEEE, International Conference on Mechatronics and Automation. Tianjin, China. Aug 4-7.

Xenomai Wiki. 2019. *How does Xenomai deliver real-time*, viewed 1 September 2021. Available at: <https://source.denx.de/Xenomai/xenomai/-/wikis/Start_Here#user-content-how-does-xenomai-deliver-real-time>

Xilinx Wiki. 2019a. *OpenAMP*, viewed 3 October 2019. Available at: <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841718/OpenAMP>

Xilinx Wiki. 2019b. *Build Device Tree Blob*, viewed 9 October 2019. Available at: <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842279/Build+Device+Tree+Blob>

Xilinx Wiki. 2019c. *U-Boot Images*, viewed 3 September 2021. Available at: <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842374/U-Boot+Images>

Xilinx Wiki. 2020a. *Accessing BRAM In Linux*, viewed 6 September 2020. Available at: <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842412/Accessing+BRAM+In+Linux >

Xilinx Wiki. 2020b. *Multi-OS Support*, viewed 31 August 2021. Available at: <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841668/Multi-OS+Support+AMP+Hypervisor>

Xilinx Wiki. 2020c. *Prepare boot image*, viewed 3 September 2021. Available at: <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841976/Prepare+boot+image>

Xilinx Support Documentation. 2013. *Simple AMP Running Linux and Bare-Metal System on Both Zynq SoC Processors*, viewed 31 August 2021. Available at: <https://www.xilinx.com/support/documentation/application_notes/xapp1078-amp-linux-bare-metal.pdf>

Xilinx Support Documentation. 2014. *Simple AMP: Bare-Metal System Running on Both Cortex-A9 Processors*, viewed on 18 April 2021. Available at: <https://www.xilinx.com/support/documentation/application_notes/xapp1079-amp-bare-metal-cortex-a9.pdf>

Xilinx Support Documentation. 2021a. *PetaLinux Tools Documentation*, viewed 24 August 2021. Available at: <https://www.xilinx.com/support/documentation/sw_manuals/xilinx2021_1/ug1144-petalinux-tools-reference-guide.pdf>

Xilinx Support Documentation. 2021b. *Zynq-7000 SoC Technical Reference Manual*, viewed 1 September 2021. Available at: <https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf>