Project Acronym:	**TRESCCA**
Project Title:	Trustworthy Embedded systems for Secure Cloud Computing
Project number:	European Commission – 318036
Call identifier	FP7-ICT-2011-8
Start date of project:	01 Oct. 2012 **Duration:** 36 months

Document reference number:	**D4.1**
Document title:	System-on-Chip and FPGA-based prototype
Version:	1.0
Due date of document:	30th of March 2015
Actual submission date:	11th of November 2015
Lead beneficiary:	CS
Participants:	A. Herrholz (CS), S. Gentzen (CS), C. Stehno (CS), R. Pacalet (IMT), M. Paolino (VOSYS), M. Grammatikakis (TEI), M. Coppola (ST), M. Soulie (ST)

Project co-founded by the European Commission within the 7th Framework Programme		
DISSEMINATION LEVEL		
PU	Public	**X**
PCA	Public with confidential annex	
CO	Confidential, only for members of the consortium (including Commission Services)	

History of Changes

ED.	REV.	DATE	REASON FOR CHANGES
AHe	0.1	2014-10-14	Intermediate version with information on available target platform and current demonstrators.
AHe	1.0	2015-11-11	Final version for 3rd Technical Review

CONTENTS

FIGURES

1 Introduction

This is the accompanying report for the prototype deliverable D4.1. This deliverable documents the development process and the results of the client prototypes that are being developed within WP4 of the TRESCCA project. Following a recommendation of the 1st Technical Meeting, the partners of the consortium have initially developed several earlier individual prototypes that eventually would be integrated into larger prototypes demonstrating multiple or all of the TRESCCA technology components. The client prototypes will be used in Task 4.2 to develop scenario-oriented cloud-based applications for final evaluation of the TRESCCA results.

Chapter 2 presents the different target platforms considered for the integrated platforms and explains why towards the end the Xilinx Zynq device and the Digilent Zedboard were chosen as the primary platforms despite shortcomings in the virtualization support.

Chapter 3 describes the hardware and software architecture of the integrated prototype, which towards the end of the project integrated the HSM-Mem, two HSM-NoC firewall and a STNoC interconnect together with an UART-controlled attacker. This setup provides a good and easy starting point for testing and validation of the individual components as well as for setting up and deploying larger applications like the WeSave application of WT.

In Chapter 4 an overview on the individual prototypes developed during the project is given including a performance analysis of running Virtual Machines on the Zedboard without hardware virtualization support.

The report ends with a conclusion and outlook in Chapter 5.

2 TRESCCA client prototyping platforms

We define a TRESCCA client device as a general embedded computing platform incorporating one or more TRESCCA security components, which can be hardware and/or software. The initial goal of this task was to develop an integrated TRESCCA prototype platform that could be used for all application scenarios and for evaluating each of the TRESCCA security components. However during the first one and a half year of the project it became clear that there would not be one platform that would serve all prototyping requirements. Most importantly, the prototype platform needs to provide hardware prototyping capabilities, i.e. it needs to incorporate an FPGA, enabling us to implement and integrate hardware components in addition to an existing System-on-Chips. Also, the components developed in WP3 require hardware virtualization support which is not yet commonly available in embedded computing platforms. In the end, the partners decided to use and develop several platforms, with the Digilent Zedboard being the primary platform and other alternatives used for cases in which the Zedboard was not usable. The following sections give an overview on the different platforms considered and for which use cases they have been used.

2.1 Xilinx/Digilent Zedboard

In a TRESCCA client device the HSM-Mem sits between the on-chip system interconnect and the controllers of external memories. In most existing Systems on Chip (SoC) the layout of the memory architecture is frozen and cannot easily modified to add the HSM-Mem.

The Zynq SoC by Xilinx [1] is a notable exception and is thus a very interesting hardware target to prototype and evaluate the HSM-Mem technology. A Zynq SoC (see Figure 1) embeds a classical microprocessor and its peripherals (an ARM dual core Cortex A9) and a Field Programmable Gate Array (FPGA) matrix on the same silicon. As shown on Figure 1**,** the microprocessor subsystem is denoted *Processing System* (PS) while the FPGA part is denoted *Programmable Logic* (PL). Thanks to the PS it is possible to turn a Zynq-based system in a regular Personal Computer (PC). GNU/Linux complete distributions have been ported on Zynq-based systems.

A Zynq-based board like, for instance, the ZedBoard by Digilent [2] (see Figure 2) is thus a good representative of the typical TRESCCA client (PC, set-top boxes, game consoles, smart meters, etc.) But the PL is really what makes Zynq a very appealing target for TRESCCA: it can be used to implement any hardware peripheral at a very reasonable cost and with good performance. Moreover, the microprocessor can access the external memory either directly through its memory controller (red path on Figure 1) or indirectly through the PL (blue path). It is thus possible to configure the PL to implement the HSM-Mem and to route part or all of the PS memory accesses through it.

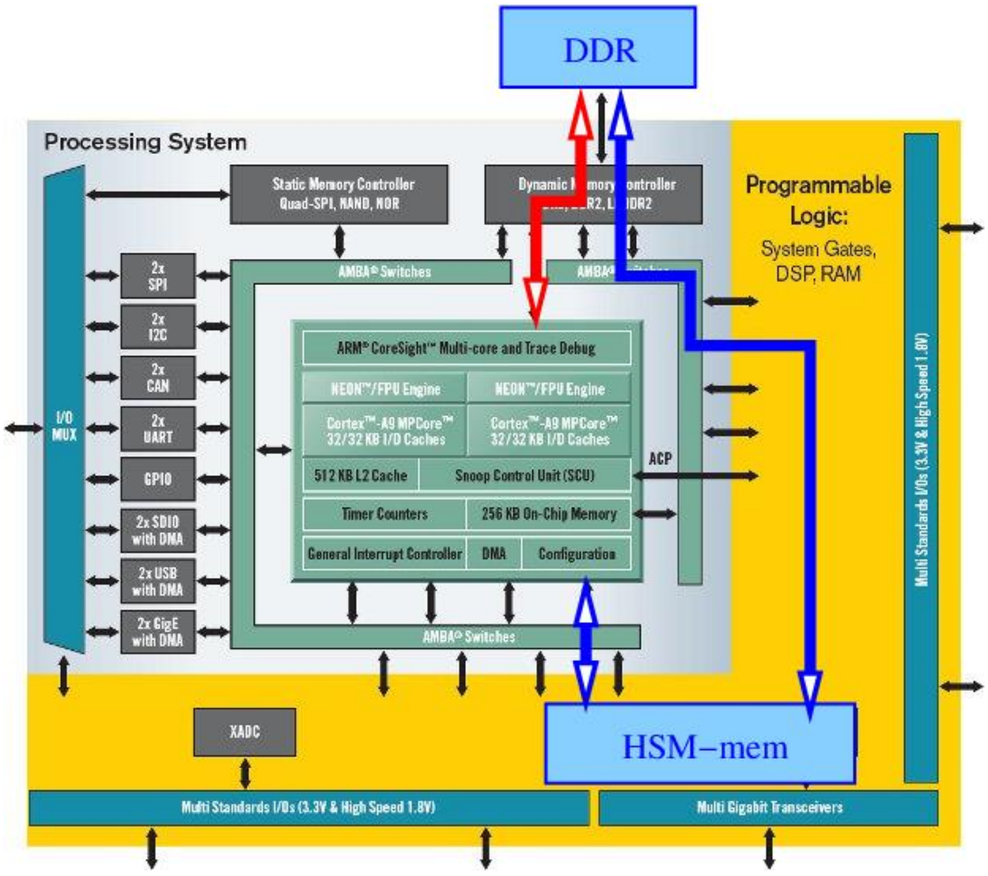


Figure 1: The Xilinx Zynq architecture

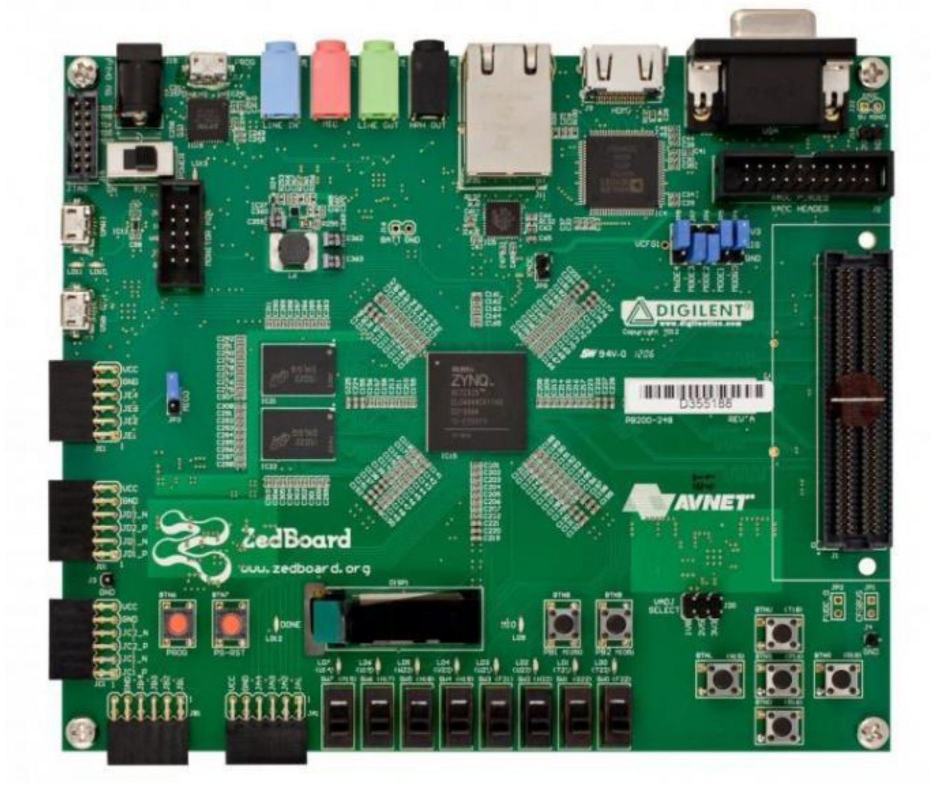


Figure 2: The Zynq-based ZedBoard by Digilent

Finally, the PL can also be configured to embed hardware emulators of attackers. These emulators can be used to tamper with the memory accesses and validate the countermeasures implemented by the HSM-Mem.

Figure 2 shows the ZedBoard by Digilent with its Zynq core (centre), the two external DDR memory chips (to the left of the Zynq core) and the various Inputs/Outputs (I/Os): Ethernet, audio, video, switches, etc. For all the above-mentioned reasons, the Xilinx-Zynq and the ZedBoard have been retained as the demonstration platform of choice for the TRESCCA HSM-Mem technology.

Despite its very good properties there are several limitations to this choice:

- The PL maximum clock frequency is about 4 to 5 times less than the clock frequency of the PS. When routing the PS memory accesses through the PL, a performance penalty is thus added[1] that blurs the overall performance degradation due to the HSM-Mem operations. As a consequence, accurate evaluations of the performance impact of the HSM-Mem are much more difficult. In order to overcome this, the clock frequency of the PS can be artificially reduced to match that of the PL. The performance evaluation can now be done very accurately at the expense of a global slowdown of the prototype of about one order of magnitude, compared with an actual client. This can be considered as acceptable for a prototype.
- The Cortex A9 microprocessor by ARM pertains to the ARMv7-A, 32-bits, architecture and lacks the hardware virtualization support that equips more advanced ARM processors. If it is an (almost) perfect target for the HSM-Mem prototyping, it is thus not the ideal one for experiments and demonstrations about virtualization. A Cortex A15 would be a much better target.

2.2 VersatileExpress

The secondary option considered for the client prototype is the ARM Versatile Express platform. This prototyping platform available from ARM, enables flexible configurations of ARM CPU cores and peripheral HW through the use of CPU and FPGA modules that can be plugged into a base board. However, due to the high costs and the limited availability it is currently only an option for partners who already own such platform. The advantage over the Zeboard is that it enables the use of other ARM CPU cores up to the upcoming 64-bit ARMv8 cores.

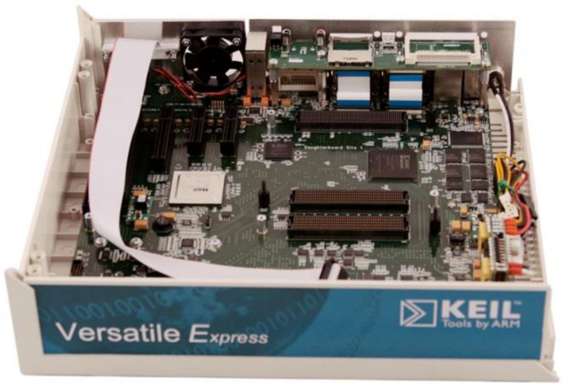


Figure 3 Versatile Express development system

[1] That adds to the one due to the HSM-Mem operations.

ST and TEI are investigating options to prototype and implement the HSM-NoC on the Versatile Express system. While more complex and expensive than the Zedboard, the Versatile Express board is more powerful due to its large flexibility and larger FPGA fabric. It would allow integrating multiple fully-featured NoC firewalls in combination with the STNoC on-chip network and the HSM-Mem. Also it would enable us to use other more advanced ARM-CPUs such as the Cortex-A15 or later, also providing hardware support for virtualization. Depending on the results of the investigations and the availability of the VersatileExpress system, ST, TEI and CS agreed to collaborate on the development and integration work for that system. If work on the Zedboard progresses as expected and use of the VersatileExpress seams feasible, work on that platform will be done in parallel to the Zedboard-based development.

The Versatile Express platform is used for one of the **HSM-NoC** demonstrators. CS started to work on a fully integrated prototype client using a Versatile Express provided by ST and TEI. However it turned out to be complex, so CS focused on the Zedboard for integration.

2.3 Samsung Chromebook

A Chromebook is a lightweight computer laptop which follows specifications defined by Google. It designed to be lightweight and compact. Typically Chromebooks run ChromeOS (a modified Linux version using Google Chrome and Chrome applications). While nowadays, there are Chromebooks using Intel x86 CPUs, typically Chromebooks use ARM-based SoCs that are comparable to the ones used in high-end smartphones.

Figure 4 Samsung Chromebook

The Samsung Chromebook used in TRESCCA includes a quad-core Cortex-A15 SoC, providing hardware virtualization support and TrustZone. This enables us to implemented two main features of a TRESCCA platform: a Trustworthy Execution Environment based on TrustZone and a framework for virtualization and VM migration. However the Chromebook SoC does not include any kind of hardware prototyping options, hence the TRESCCA hardware components cannot be included.

The Samsung Chromebook is used in TRESCCA for implementing and demonstrating **tresccad**, the VM migration service, and the **TEE**. For this purpose a linux-based operating system is required. For higher flexibility the ChromeOS has been replaced by Ubuntu 14.04.

2.4 ARM Juno development platform

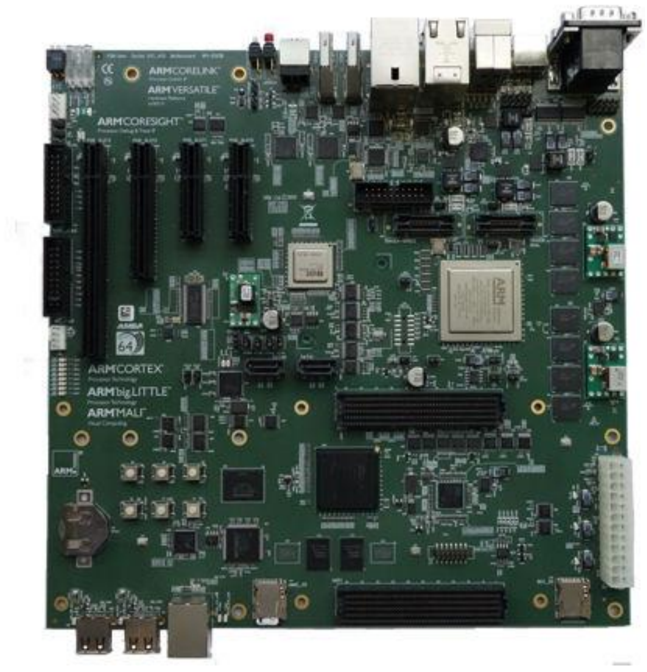


Figure 5 ARM Juno platform

The ARM Juno platform is a prototyping system provided by ARM for ARMv8 (64-bit ARM) development mainly targeted at silicon and low-level software developers. It is one of the earliest available platforms for 64-bit ARM development, though it is not be used for actual product development. Its price is in the same range as the predecessor, the Versatile Express system.

In TRESCCA, the Juno board is used by Virtual Open Systems for prototyping the **Trusted Execution Environment**.

2.5 Virtual prototypes

During the first year of the project no hardware prototypes were available. Instead virtual prototypes of the hardware components have been developed based on their functional specifications. Hence, those prototypes were fully functional, providing the same features and interfaces as the final hardware implementations. Though, as being virtual, they could only be used for simulation and integration into larger virtual prototypes of whole System-on-Chips. The three main virtual prototyping technologies used were Gem5, SystemC/SoCLib, ARM Fastmodels, QEMU. For reference, a short description of each technology is given here. For details on the virtual prototypes using Gem5 and SoCLib please refer to deliverable D2.2.

2.5.1 SystemC/SoCLib

SystemC is the well-known IEEE standard 1666 for system-level modelling. It is modelling language and simulator available as a C++ class library which is used on top of standard C++. It provides modelling elements and means to create executable models of parallel systems consisting of modules with parallel processes, ports, and communication channels. Originally developed as a C-based hardware description language, it has now grown into powerful language for virtual prototyping as well as high level synthesis of electronic circuits. TLM 2.0 is a SystemC-based modelling technique and library which enables the creation of so-called Transaction Level Models of hardware systems which are more abstract with respect to structure, function and timing compared to more traditional

RT-level hardware models. Many virtual prototype models and tools available today are either based of or at least compatible with TLM 2.0 enabling interfacing of models and simulators from different vendors.

SoCLib is an open platform for virtual prototyping of Multi-Processors System on Chip (MP-SoC) based on SystemC. The project started as a French publicly funded project. It is now maintained at Lip6. The core of the platform is a library of SystemC simulation models for virtual components (IP cores). The main concern is true interoperability between the SoCLib IP cores. All simulation models are written in SystemC, and can be simulated with the standard SystemC simulation environment. Two types of models are available for each IP-core: CABA (Cycle Accurate / Bit Accurate), TLM-DT (Transaction Level Modelling with Distributed Time). All simulation models and most associated tools are distributed as free software.

In TRESCCA SoCLib is being used for developing and simulating the HSM-Mem prototype integrated into ARM or MIPS-based system-on-chips.

2.5.2 Gem5

Gem5 is a computer architecture simulator initially developed at the University of Michigan and now supported and used by other universities and companies, such as ARM, AMD, Texas Instruments and HP. It is a merge of two previously separate simulators namely M5 and GEMS. In contrast to SystemC (which does not come with any actual models), Gem5 is not an HDL, but has been developed as a computer architecture modelling framework mainly used for concept validation and preliminary exploration studies. It is an open, high-level, configurable simulation framework, already including multiple ISAs, and diverse CPU models, as well as a detailed and flexible memory system. It includes multiple cache coherence protocols and interconnect models and currently supports most commercial ISAs (ARM, ALPHA, MIPS, Power, SPARC, and x86), including booting Linux on three of them (ARM, ALPHA, and x86).

In TRESCCA Gem5 is being used by TEI for developing prototypes and system simulations of HSM-NoC-based systems.

2.5.3 ARM Fast Models

ARM Fast Models are virtual prototypes of ARM processors and System-on-Chip commercially available from ARM itself. The models themselves use proprietary technology, however they can be extended with SystemC and other 3rd party models through TLM 2.0 interfaces.

The models are available for a wide range of ARM processor architectures and IP components and can be used with standard software compilers and debuggers.

In TRESCCA ARM Fast models are being used by VOSYS for developing the TEE and secure hypervisor.

2.5.4 QEMU

QEMU is an open-source emulator which was originally developed for emulating non-x86-based processor architectures on x86 processors. Since its beginning it has grown into a system which can emulate full computing systems including GPUs, network interfaces, etc. Nowadays it is mostly used to emulate peripheral components for Virtual Machines, where the CPU itself is not emulated but using the hardware virtualization support of modern desktop CPUs. However such approach does only work if the host CPU has the same Instruction Set Architecture (ISA) as the CPU of the Virtual Machine.

In TRESCCA, QEMU has been used to emulate Versatile Express based systems. In that use case, QEMU emulates a Versatile Express including its CPU module and all standard peripheral components found on a Versatile Express system. The CPU features emulated by QEMU include TrustZone, so it is possible to run the OP-TEE in a QEMU-emulated virtual machine.

One drawback of QEMU is, that it cannot be easily extended with custom hardware, such as the HSM-Mem and the HSM-NoC. There are approaches using the PCI-Express interface provided by QEMU to connect SystemC modules, though this would require that the SystemC modules simulate a PCI express interface, which is not a common case. Also QEMU has been designed with fast performance and not exact hardware simulation. Therefore QEMU is not suitable for doing performance analysis of hardware systems.

2.6 ARM-based development boards

Additional alternatives for prototyping to be considered are typical ARM development boards, similar to the RasberryPI, the ODROID board or the Cubieboard. Those development boards usually contain an ARM-SoC in combination with typical peripherals and interfaces, such as USB ports, HDMI connectors and SDCard slots. The clear disadvantage of these boards compared to the Zedboard and the VersatileExpress is the missing FPGA area. The hardware of the devices is fixed and therefore any integration of the HSM-NoC and HSM-Mem is infeasible. However the boards are quire affordable, starting at ca. 50 EUR, and are available with up-to-date ARM CPUs such as Cortex-A15 or Cortex-A7 providing features such as hardware supported virtualization (which is not available on the Zynq device).

While not usable for the final prototype such boards could be used for intermediate prototypes showcasing only software-based features, e.g. using the Secure Hypervisor and TEE in combination with hardware virtualization and TrustZone. Also, due to their good software support, they are also a good starting point for application development before moving to the more complex FPGA-based platforms.

An overview and comparison of potential candidates to be used in TRESCCA is shown in Table 1. The most promising candidate is the **Cubieboard2** or the **Cubietruck** which both include a dual-core Cortex-A7 SoC. This processor has been designed by ARM for low cost mobile applications or as the low-power helper processor for the so called *big.LITTLE* multi-core architecture (high performance processors (e.g. Cortex-A15) in combination with small low power processors). The big advantage of the Cortex-A7 is that despite being small and not very expensive it already includes most of the advanced ARM instruction set features including hardware virtualization support.

Board	Cubieboard1	Cubieboard2	Cubietruck (Cubieboard3)	PandaBoard	PandaBoard ES
SoC	Allwinner A10	Allwinner A20	Allwinner A20	OMAP4430	OMAP4460
CPU Core(s)	1 x Cortex-A8	2 x Cortex-A7	2 x Cortex-A7	2 x Cortex-A9	2 x Cortex-A9
RAM	1 GB DDR3@480M	1 GB DDR3@480M	1/2 GB DDR3@480M	1 GB DDR2	1 GB DDR2
Flash	4GB NAND, up to 64 GB micro SDCard	3.4GB NAND, up to 64 GB micro SDCard	8GB NAND, micro SDCard	SDCard	SDCard
USB	2xUSB2, 1xUSB2-OTG	2xUSB2, 1xUSB2-OTG	2xUSB2, 1xUSB2-OTG	2xUSB2, 1xUSB2-OTG	2xUSB2, 1xUSB2-OTG
Ethernet	10/100 MBit	10/100 MBit	10M/100M/1G MBit	10/100 MBit	10/100 MBit
Wifi	only via USB	only via USB	Wifi, BT	WiFi,BT	WiFi,BT

Video IO	1080p HDMI out	1080p HDMI out	1080p HDMI&VGA out	HDMI out	HDMI out, DVI-D out
IO	96 extended pin	96 extended pin	54 extended pins	camera, display/LVDS, expansion	camera, display/LVDS, expansion
Price	50-64 EUR	60-75 EUR	90-100 EUR	150 EUR	180 EUR
Availability	Good	Good	Good	Good	Bad
Remarks			"Made for products "		

Table 1 Overview and comparison of possible ARM development boards for TRESCCA

At the end of the project, none of the investigated boards were used for prototyping. Though WT did initial experiments using the Cubieboard before finally moving to the Zedboard for the final implementation.

3 Zedboard-based integrated prototype

The Zedboard-based integrated prototype is a full integration of all TRESCCA hardware components into a single prototype platform. The Zedboard was chosen as it is available to all partners due to its low costs. Also the FPGA resources provided are sufficient to implement also larger and more complex components like the HSM-Mem and the Network-on-Chip.

3.1 Hardware architecture

In addition to the processing system (dual-core ARM Cortex-A9, DDR RAM controller, UART, Ethernet, MMC, etc.) of the Zynq device the hardware architecture of the integrated prototype includes:

- An STNoC (Network-on-Chip) with two master input ports and one slave output port
- 2 HSM-NoC (Network-on-Chip firewalls)
  - 1 placed between CPU and STNoC
  - 1 placed between on-chip attacker and STNoC
- 1 HSM-Mem placed between STNoC and DDR-RAM controller
- 1 UART-controlled attacker module (acting as internal and external attacker)

A simplified view of the architecture is shown in Figure 6. The orange components are provided by TRESCCA partners (HSM-NoC – TEI, STNoC – ST, HSM-Mem, Attacker – IMT). For the purpose of illustrating the use cases and attack scenarios covered by the prototype, the UART controlled attacker is shown as two separate components. However it is in fact one component acting as attacker to the memory bus (between HSM-Mem and DDR-Ctrl) as well as an internal on-chip attacker with full access to the internal bus.

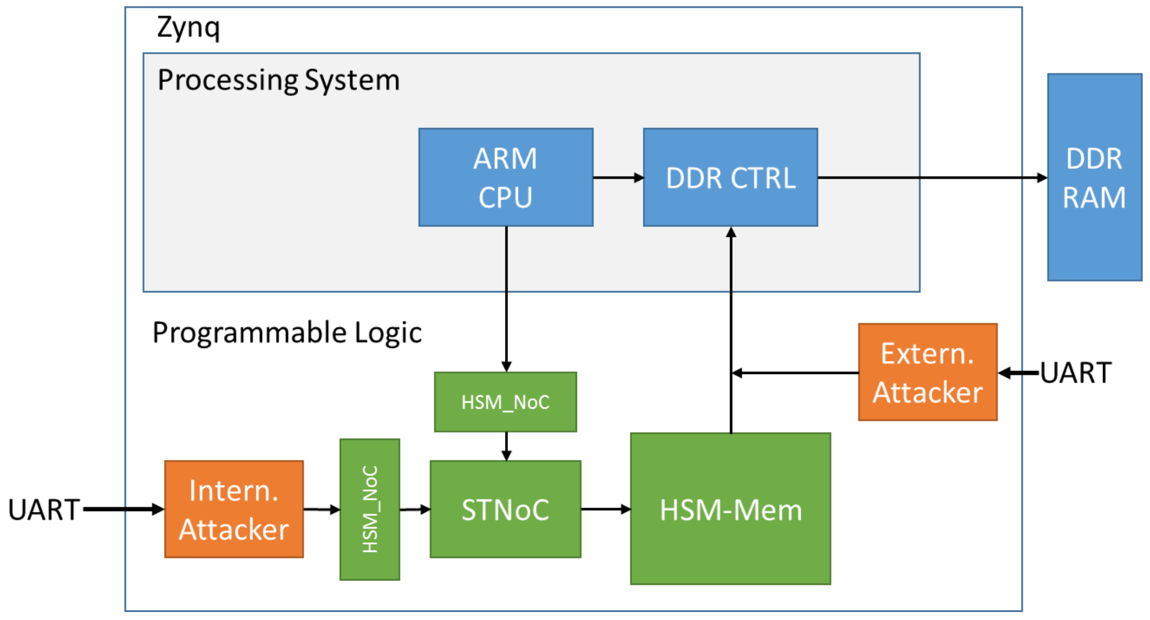
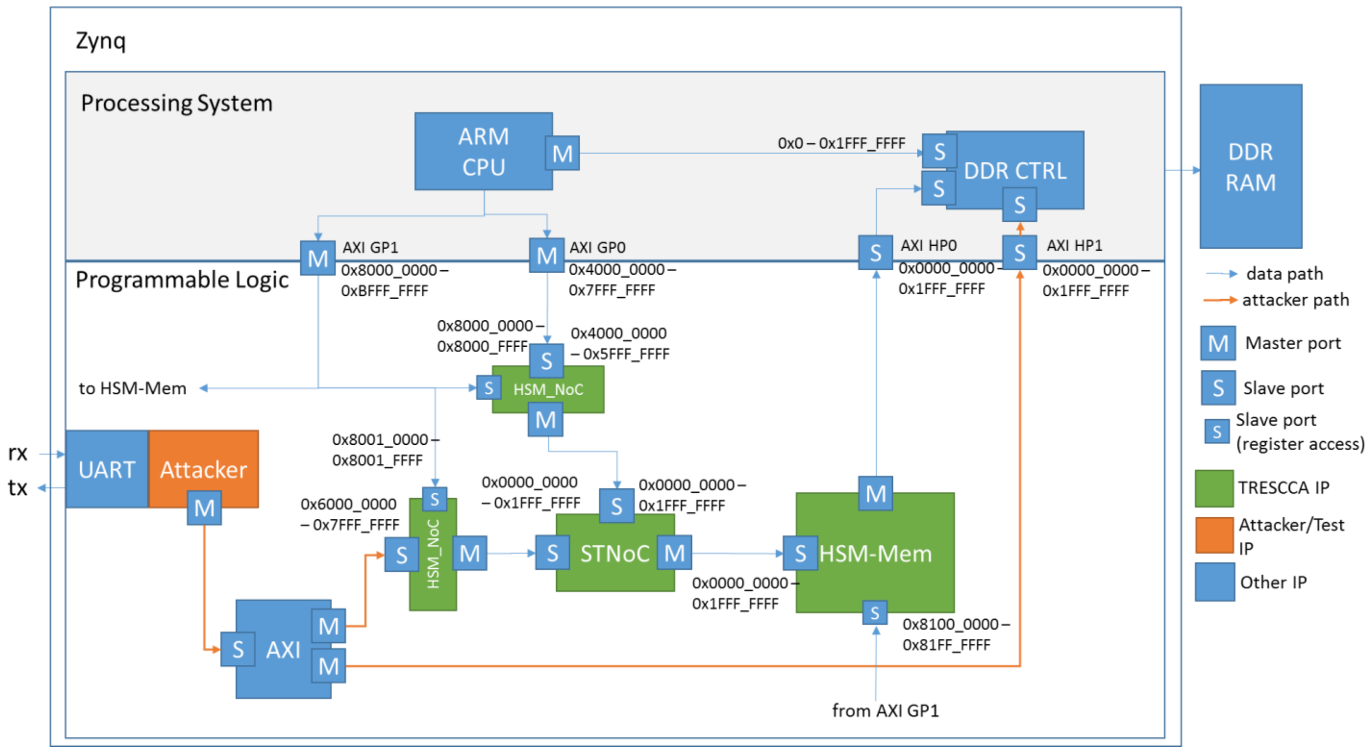


Figure 6 Simplified view of architecture of integrated prototype

In a real world scenario, the external attacker would actually attack the bus signals between the DDR-RAM controller and the external memory. The internal attacker could be another System-on-Chip component like a GPU or USB controller. Such internal attacker could intentionally be malicious or just by accident, e.g. if there is a malfunction.

The detailed architecture of the whole setup including the address ranges used by the different components is shown in Figure 7.



Zynq
Processing System
ARM CPU
M
0x0 – 0x1FFF_FFFF
S
DDR CTRL
S
S
DDR RAM
M
AXI GP1
0x8000_0000 – 0xBFFF_FFFF
M
AXI GP0
0x4000_0000 – 0x7FFF_FFFF
S
AXI HP0
0x0000_0000 – 0x1FFF_FFFF
S
AXI HP1
0x0000_0000 – 0x1FFF_FFFF
Programmable Logic
0x8000_0000 – 0x8000_FFFF
0x4000_0000 – 0x5FFF_FFFF
S
to HSM-Mem
S
HSM_NoC
M
rx
tx
UART
Attacker
M
0x8001_0000 – 0x8001_FFFF
0x6000_0000 – 0x7FFF_FFFF
S
S
HSM_NoC
M
0x0000_0000 – 0x1FFF_FFFF
S
0x0000_0000 – 0x1FFF_FFFF
S
STNoC
M
M
S
HSM-Mem
0x0000_0000 – 0x1FFF_FFFF
S
0x8100_0000 – 0x81FF_FFFF
S
AXI
M
M
from AXI GP1
data path
attacker path
M Master port
S Slave port
S Slave port (register access)
TRESCCA IP
Attacker/Test IP
Other IP

Figure 7 Detailed architecture of integrated prototype

3.1.1 ARM-CPU

The ARM CPU of the Zynq processing system is a dual-core Cortex A9. Though in the demonstrator only one core is used. The Cortex-A9 supports the ARMv7 instruction set including extensions such as TrustZone. The ARM CPU can either access the DDR-RAM directly or through the programmable logic through the NoC firewall, the STNoC and the HSM-Mem.

The address space to use either of the two paths are:

- **0x0000_0000 – 0x1FFF_FFFF** for direct access to the DDR-RAM
- **0x4000_0000 – 0x5FFF_FFFF** for access through the programmable logic

Both address spaces map to the same addresses in DDR-RAM, e.g. accessing address 0x4500_0000 or 0x0500_0000 would be the same location in memory.

It should be noted that the DDR-RAM is only accessible starting from an offset of 0x01000_0000 of the respective base address. Below that address only on-chip RAM can be accessed, however only via the direct access.

In addition to the memory address ranges, the CPU has access to the configuration registers of both HSM-NoCs and the HSM-Mem. Those are accessible via the following base addresses:

- **0x8000_0000** : HSM-NoC 1 (between CPU and STNoC)
- **0x8001_0000** : HSM-NoC 2 (between Attacker and STNoC)
- 0x8100_0000 : HSM-Mem

Via those configuration registers the rules of the HSM-NoC as well as the security policies of the HSM-Mem can be configured. No other component in the system has access to these registers, i.e. the CPU is in control of the security enforced by HSM-NoC and HSM-Mem.

3.1.2 HSM-NoC

The HSM-NoC firewall instances integrated into the prototype are identical to the stand-alone versions implemented on the Zedboard and the Versatile Express. However there are now two instead of one. As can be seen from Figure 6 one HSM-NoC is located between the CPU and the STNoC while the other one is located between the internal attacker and the STNoC. The former is supposed to protect the path from the CPU over the STNoC through the HSM-Mem to the DDR RAM. The latter is supposed to protect the internal bus (STNoC) from the attacker.

Both HSM-NoC allow to setup several memory ranges for which rules can be defined. If the address of any incoming bus transaction is within one of the defined ranges, the corresponding rule for that range will be checked. If the rule denies the requested access, the transaction will not be forwarded but instead completed by the firewall returning invalid data. Also the firewall will issue an interrupt to signal that a rule was activated.

The interrupt signal of the HSM-NoCs are both connected to the CPU, hence the CPU will be notified by any invalid access trying to pass through the firewalls.

The address ranges covered by the two HSM-NoCs are:

- **0x4000_0000 – 0x5FFF_FFFF** for the firewall between CPU and STNoC
- **0x6000_0000 – 0x7FFF_FFFF** for the firewall between Attacker and STNoC

Both firewalls are designed to tie the first 3 most significant bits of the address of any incoming transaction to 0 at the output, i.e. the address of the outgoing transaction will be between **0x0000_0000** and **0x1FFF_FFFF**. While this is not relevant for the CPU and the Attacker, it is relevant for any bus component located at the output of the firewall, i.e. the STNoC and the HSM-Mem will only see transactions within that range. This is also the address range that will be active at the input of the DDR-RAM controller from the programmable logic.

3.1.3 STNoC

The STNoC is network-on-chip specifically designed for this prototype by ST. It provides two input ports (1 for each firewall) and one output port to the HSM-Mem. All transactions incoming to the two input ports are forwarded to the output port. No modifications are applied to either addresses or data used by the transactions, hence all transactions appear at the HSM-Mem as they have left the firewalls.

3.1.4 HSM-Mem

The HSM-Mem is integrated between the output port of the STNoC and the input port of the processing system, which itself is connected to the DDR-RAM controller. Any bus traffic leaving the firewalls will be forwarded to the input of the HSM-Mem. Depending on its configuration, data will either just passed or protected for confidentiality and/or integrity.

The configuration registers of the HSM-Mem can only be accessed from the CPU. The base address of the registers is **0x8100_000**.

3.1.5 UART-controlled attacker

The UART controlled attacker has been design by IMT for integration into the HSM-Mem prototype system and the integrated prototype. It has been designed to enable easy generation of bus transactions inside a TRESCCA prototype design from a PC or laptop through a simple UART interface.

The basic structure of the attacker is shown in Figure 8. The module consists of a bus master module which is able to generate read and write transaction via its right master interface. The right master interface can be connected to any AXI-compatible bus interface. Via its left master interface the bus master module is directly connected to a UART IP core accessing the receive and transmit registers of that core. Internally the bus master has a set of registers which can be read and written by sending the right commands via the UART. The bus master will receive that commands by constantly polling the receive register of the UART and respond by sending data to the transmit register. By writing a certain set of commands to the bus master, any terminal client connected to the UART interface can generate read and write transactions on the bus.

For example, by first sending a "set-address" command to the internal register of the bus master, and then sending a "issue read request" command, the bus master will generate a read transaction with that address on the bus. After sending another "read-data" command to the bus master, it will respond with the value returned by the read transaction. More details on the structure and use of the attacker module can be found at https://secbus.telecom-paristech.fr/raw-attachment/wiki/Downloading/uart2maxilite.pdf.

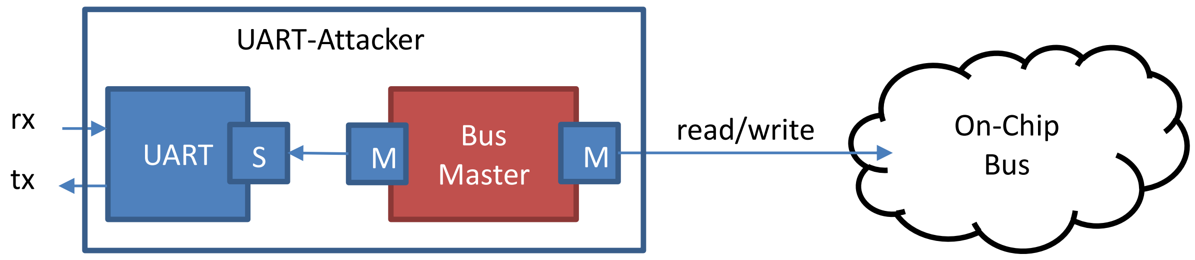


Figure 8 Structure of UART-controlled attacker

In the integrated prototype the UART-controlled attacker is used to play the role of two possible attackers to the system:

- An internal attacker, for example trying to access memory used by software running on the CPU or running a denial of service attack by constantly generating transaction on the on-chip bus.
- An external attacker trying to read and/or manipulate data running between the System-on-Chip and an external memory.

For this purpose the bus master interface of the attacker has been split to two outputs with different address ranges:

- **0x0000_0000 – 0x1FFF_FFFF**: This output is directly routed to the DDR-RAM controller, bypassing the HSM-Mem, and by this playing the role of the external attacker.
- **0x6000_0000 – 0x7FFF_FFFF**: This output is connected to the input of the second firewall, playing the role of the internal attacker.

3.2 Software architecture

The software architecture of the integrated prototype consists of base software supporting the test and evaluation of the integrated components as well as the setup and execution of applications. While the prototype can be used with bare metal applications (manual programming of hardware, no operating system) in general, the prototype comes with a set of startup scripts and loaders to automatically initialize the hardware design and start a Linux environment.

3.2.1 SW setup and startup process

The whole software environment is available as a set of files which are put on a FAT32-formatted SD card. The SD Card is put into the Zedboard and the Zedboard is powered on. The system then automatically runs through the startup process shown in Figure 9. Initially a first stage bootloader (FSBL) is loaded which initialized and calibrates the processing system of the Zynq device. Then the FSBL loads the hardware programming bitfile for the FPGA. After the programming, the programmable logic contains the architecture described in 3.1. Finally the FSBL loads U-Boot.

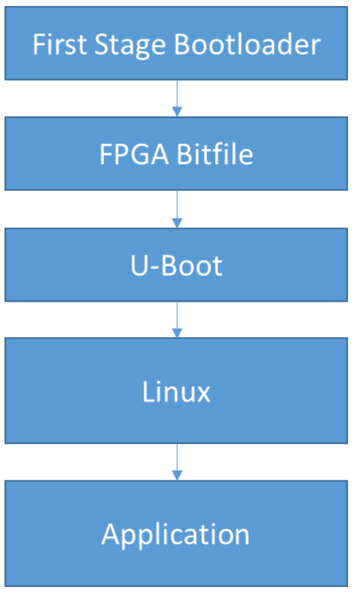


Figure 9 Software startup process of integrated prototype

U-Boot is common boot-loader used for loading Linux on embedded systems. U-Boot is open-source and can be customized to support different hardware architectures and boards. U-Boot loads the three main components of Linux:

- Device tree – contains all information on devices present in the hardware systems including system memory, base addresses of registers, IRQ numbers and configuration properties
- Kernel – the actual operating system kernel
- Ramdisk – a minimal rootfile system of the operating system which is loaded into RAM. Any changes made during runtime are lost after a reboot.

U-Boot and the Linux kernel have been configured to have Linux use the address range of 0x4800_0000 to 0x5FFF_FFFF as system memory, hence all memory accesses generated by Linux will fall into that range and therefore be routed through the STNoC and the HSM-Mem.

In principle the kernel would still be able to access physical addresses outside of that range, however they would not be recognized as RAM but only as addresses for memory-mapped I/O.

3.2.2 Linux operating system

The Linux provided with the integrated prototype is a minimal Linux enabling tests and execution of demo applications. It can either be accessed by serial console (USB-cable connected to Zedboard) or via network (if ethernet cable is plugged in). There is no support for graphical output via the HDMI port.

After successful boot, the user is greeted with a login prompt. Users can then login to the system, run tests and applications. To support easy use of user-defined files, the system automatically mounts the SD-Card file system, hence any file available on the SD card can be accessed from the system.

Though designed as a minimal system, the rootfile system includes Python 2.7 enabling easy setup and execution of the Python-based WeSave application.

Additionally the system includes the tool *devmem*, which can be used to easily access any physical address reachable from the CPU. This significantly eases the task of initially checking the existence of custom hardware components by testing access to their registers.

3.2.3 HSM-NoC Linux driver

The HSM-NoC Linux driver has initially been designed by TEI in WP3 and tested in the HSM-NoC Zedboard prototype.

For the integrated prototype, the drivers have been modified by CS. The main reasons and the resulting modifications are:

The initial driver was designed for one firewall with a fixed configuration address and IRQ number. However there are now two instances with different configuration addresses and IRQ numbers. Therefore the drivers have been modified to enable the setting of such parameters when loading the drivers into the kernel.

The initial scenario used a Linux running in the range of 0x0000_0000 to 0x1FFF_FFFF (low memory range) with direct access to the memory. The memory range of 0x4000_0000 to 0x5FFF_FFFF (high memory range) was routed through the HSM-NoC to the DDR-RAM but only available as IO memory. The protected process allocated memory in the lower address range while the attacker process tried to access that memory via the higher memory range (through the firewall). When the firewall was active, that access was successfully prevented.

In the current scenario of the integrated prototype, Linux is already using the high memory range routed through the firewall. For a similar approach the role of attacker and protected process needed to be modified. The attacker is now trying to access the memory through the high memory range, while the protected module allocated IO memory in the lower memory range.

Figure 10 and Figure 11 are illustrating the different approaches.

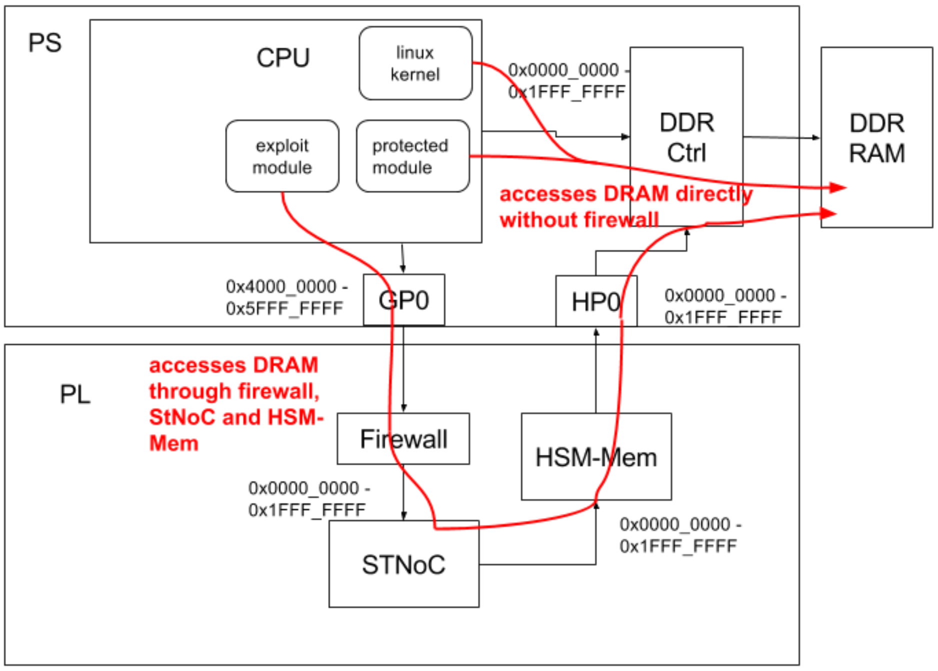

PS
CPU
linux kernel
exploit module
protected module
0x0000_0000 - 0x1FFF_FFFF
DDR Ctrl
DDR RAM
accesses DRAM directly without firewall
0x4000_0000 - 0x5FFF_FFFF
GP0
HP0
0x0000_0000 - 0x1FFF_FFFF
PL
accesses DRAM through firewall, StNoC and HSM-Mem
Firewall
HSM-Mem
0x0000_0000 - 0x1FFF_FFFF
STNoC
0x0000_0000 - 0x1FFF_FFFF

Figure 10 Initial TEI scenario with Linux accessing RAM via low memory range

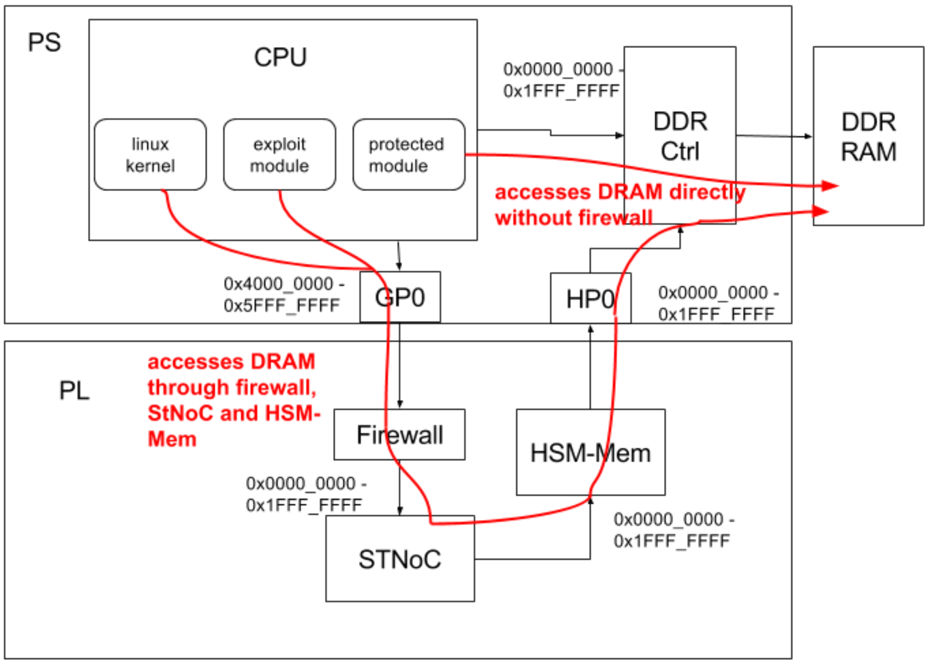

PS
CPU
linux kernel
exploit module
protected module
0x0000_0000 - 0x1FFF_FFFF
DDR Ctrl
DDR RAM
accesses DRAM directly without firewall
0x4000_0000 - 0x5FFF_FFFF
GP0
HP0
0x0000_0000 - 0x1FFF_FFFF
PL
accesses DRAM through firewall, StNoC and HSM-Mem
Firewall
HSM-Mem
0x0000_0000 - 0x1FFF_FFFF
STNoC
0x0000_0000 - 0x1FFF_FFFF

Figure 11 Scenario of integrated prototype with Linux accessing RAM via high memory range

Though for the final test scenarios the focus will be on the second firewall protecting the internal bus from the attacker. The attacker is independent of the CPU and provides the more realistic scenario, where some System-on-Chip component other than the CPU tries to access memory or generates denial-of-service attacks. Here the protected process can allocate memory provided by the kernel within the high memory range, while the UART-controlled attacker is trying to access that memory from his position. The firewall can be configured via the driver to protect that specific memory area, preventing the attacker from accessing and manipulating it. In case of an invalid access detected by the firewall, an IRQ will be issued, which will be handled by the IRQ handler of the HSM-NoC driver.

3.3 UART attacker sample programs

IMT has provided a set of elemental attacker programs that can be used on a Linux PC or Laptop connected via a serial cable (or USB-to-Serial adapter) to the UART-controlled attacker.

The programs included are

- *uart2mem_read*: Read and return the value at a given address.
- *uart2mem_write*: Write a given value to a given address.
- *uart2mem_f2m*: Write the content of a given text file to a given address.
- *uart2mem_m2f*: Read a given number of bytes from a given address and write the returned data to a given text file.

3.4 Initial test results

Initial tests have been run on the integrated prototype testing its general operation and availability and functionality of the integrated HW components. Details of the final tests and evaluation are reported in D4.3.

4 Individual client prototypes and tests

During the 1st Technical Review the reviewers recommended to use a more *agile* development process, i.e. to produce early and intermediate demonstrators and prototypes documenting and validating the progress in WP2 and WP3. Also it would help to identify early on incompatibilities or other issues for the integration.

During Period 2 and Period 3 the project partners worked on individual prototypes showcasing parts of the results of the individual HW/SW developments. That prototype finally led to the development of the integrated prototype presented in Chapter 3.

4.1 Virtual Prototypes

In WP2 two main virtual prototypes have been developed:

- A virtual prototype of the HSM-Mem developed by IMT and written in SystemC. It is integrated into the SoCLib environment and thus follows the SoCLib modelling style. It is distributed as free software and is publically available via the website *http://secbus.telecom-paristech.fr*. Future updates of the virtual prototype will be made available via this website. The model includes a functionally complete model of the HSM-Mem integrated into either an ARM- or MIPS-based system on chip. Additionally IMT provides a boot loader, operating system (MutekH) and software driver, allowing full software tests of the HSM-Mem using the virtual prototype.
- A virtual prototype of the HSM-NoC has been developed by IMT. This was initially developed as a bit-accurate SystemC model of the HSM-NoC in order to assist in early design space exploration. Later on a C++ model was developed from the SystemC model and ported to the network interface of a Gem5 network model (Garnet fixed pipeline) in order to fully characterize performance and power metrics as early as possible in system design.

More details on the virtual prototypes of HSM-Mem and HSM-NoC are reported in D2.2.

4.2 GlobalPlatform TEE / DRM scenario prototype

VOSYS has developed a first prototype of the DRM scenario using an initial implementation of the GlobalPlatform Trusted Execution Environment (TEE). The TEE is hosting a key management software that is being accessed by an audio player application to get a key to decrypt and play an encrypted music file. The TEE and the application are isolated through virtualization. While the application runs inside a virtualized Android environment, the TEE runs inside a virtual machine using the L4/Fiasco.OC kernel and run-time. While currently implemented on top of an x86 architecture using virtualization, it is planned to port the application first to an ARM-based development board and later to the Zedboard using TrustZone to isolate and protect the TEE from the Android system.

4.3 FPGA-based prototype of HSM-NoC

TEI has developed an initial FPGA prototype of the HSM-NoC implementing a simplified version of the firewall rule checking. The demo runs on the Zedboard and already enables to run performance measures and execution of bare metal software applications.

4.4 FPGA prototype of (simplified) HSM-Mem

IMT has developed a first FPGA-based prototype of a simplified HSM-Mem showcasing the principal architecture that will also be used for the final client prototype. The HSM-Mem is implemented inside the FPGA fabric and connected to the ARM CPU and the on-chip memory controller via the AXI on-chip bus. All memory access can be routed via the bus through the HSM-Mem first and to the memory controller afterwards. In addition, a DMA-capable attacker is also implemented inside the FPGA fabric and connected to the bus, bypassing the HSM-Mem. As can be shown by the demo, all features of the HSM-Mem can be used to protect confidentiality as well as integrity of all data going from the CPU to memory and vice versa against the DMA attacker.

4.5 Zedboard with Linux and workflow for HW integration and driver customization

CS has set up a work flow and framework based on the Zedboard enabling fast integration of different on chip HW peripherals and easy implementation and porting of Linux SW drivers for these peripherals.

This will be used for the final integration and set up of the HSM-Mem and HSM-NoC into one common client platform prototype.

4.6 Performance analysis and comparison of target platforms

During the beginning of the 3rd Period CS has setup and executed benchmarks on most target platforms considered for the integrated prototype. The main reason of it was to analyze the performance impact of running Virtual Machines on the Zedboard without hardware virtualization support. In that case, virtualization would be realized using QEMU emulation which by principle is significantly lower as hardware supported virtualization, as the instruction code is only executed by emulation and not directly on the host CPU. At that time the Versatile Express board was still considered to be an option for the integrated prototype, as it would provide hardware virtualization support. However, even though performance would have been better, the increased complexity for hardware prototyping and integration compared to the Zedboard required us to give up the plan after several failed attempts to make progress.

For the performance analysis and comparisons of the different platforms and setups we have used the benchmark tool Coremark (http://www.eembc.org/coremark/index.php). This benchmark has been designed as a more portable and less artificial alternative to the widely-used Dhrystone benchmark, and is in particular targeted at the needs of embedded systems. The benchmark can be run as a bare metal program or as a user-level program in Linux. As part of our analysis we compiled and ran the benchmark in several different environments and setups including:

- Bare metal on a Zedboard
- Linux on a Zedboard
- A Linux Virtual Machine using QEMU on a Zedboard
- Linux on a Chromebook
- A Linux Virtual Machine using QEMU on a Chromebook with and without KVM (hardware virtualization support)
- A i7-based PC running Linux Mint 16 as reference

The results of the performance tests are shown in Figure 12.

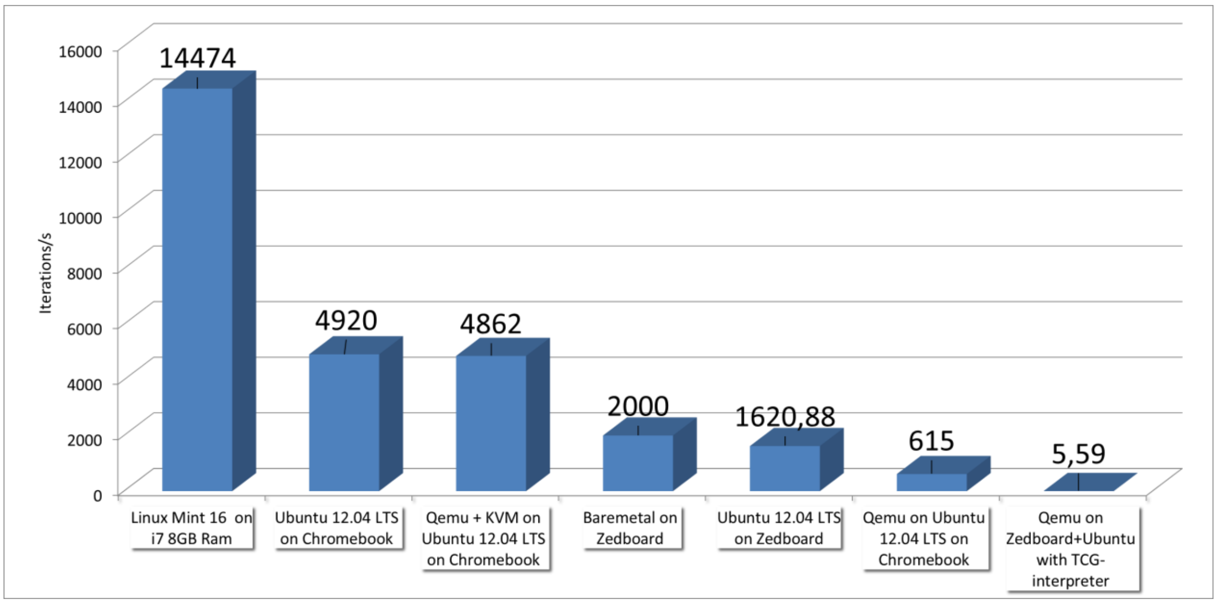


Figure 12 Coremark results on different target platforms

As can be seen the Virtual Machine running on the Zedboard using QEMU (5,59) is more than 350 times slower than the baremetal run on the Zedboard. At the same time, on the Chromebook with hardware virtualization support, the VM is only about 1 percent slower than its non-virtualized version.

Another important result is that we had issues to use QEMU with the native mode interpreter on the Zedboard. When using the native mode the instruction set of the guest system is directly translated to instructions of the host system. In principle this is support for ARM-to-ARM as well as for x86-ARM. However it was not stable on the Zedboard which crashed several times when running the benchmark. As a fallback QEMU was using with the so-called “TCG-interpreter” mode. In that mode, QEMU emulates the instruction set using a virtual CPU. This is obviously the least efficient way to execute ARM code on an ARM processor.

In comparison, the native mode of QEMU used on the Chromebook was only 8 times slower than without QEMU emulation. While still significantly slower than the non-virtualized version this could have been an acceptable performance for some of application scenarios evaluated by TRESCCA.

We can conclude from this, that the QEMU-based virtualization on the Zedboard could have been acceptable for a certain set of applications, if the native-mode of QEMU would have worked. However the TCG-interpreter performance was worse by several magnitudes. Therefore, it was decided with other partners, that QEMU-based virtualization on the Zedboard is not an option for the TRESCCA application scenarios. Hence, those scenarios depending on virtualization would require an alternative platform, like the Chromebook. Also in that cases, no TRESCCA hardware components could be integrated.

5 Conclusion and outlook

This report described the development and architecture of the integrated prototype of the TRESCCA client platform serving as the primary platform for developing and setting up the application scenarios of Task 4.2.

During the initial phase of the project the partners investigated options for a target platform for the integrated prototype. Initial plans of designing a custom board were discarded very early, as the expected additional flexibility would not justify the additional cost and effort, as there were already other low-cost solutions available.

The main requirement for the target platform was the possibility to prototype hardware components in addition to a System-on-Chip. The only two options at that time were the Versatile Express system from ARM and the Zynq device from Xilinx. Both platforms provided a System-on-Chip and a programmable area. Though as an integrated device the Zynq was less expensive than the low-volume and costly Versatile Express system. In fact the Zynq-based Zedboard proved to be an affordable solution for all partners. However one drawback of the Zynq was the lack of hardware virtualization support. Finally the partners agreed to use the Zedboard for all hardware prototyping and use alternatives like the Versatile Express or a Samsung Chromebook for prototypes and scenarios that required virtualization support.

During the second year the partners developed individual prototypes, each demonstrating and testing single components, like the HSM-Mem, the HSM-NoC or the OP-TEE. During the third periods activities to develop an integrated prototype have been intensified, finally leading to an integrated prototype with all security hardware components included. In addition an easy to use UART-controlled attacker allows easy setup and scripting of attacks to the internal bus as well as to the external memory bus. The prototype includes a minimal Linux systems also having support for Python 2.7 applications providing a starting point for implementing the WeSave application of WT on the platform.

In the future, parts of the integrated prototype will be made available open-source. Excluded from publication are the HSM-NoC and the STNoC. For the open-source release those components will be removed. Furthermore with the upcoming release of a new generation of Zynq devices incorporating ARM CPU cores with hardware virtualization support it would be worth investigating options to port the design to a new platform which could include all TRESCCA outcomes in once affordable device. Though this will not be possible during the remaining time of the project.