# Virtual Prototyping Platform for Multiprocessor System-on-Chip Hardware/Software Co-design and Co-verification


**Arya Wicaksana and Chong Ming Tang**

**Abstract** This paper describes the implementation of a virtual prototyping platform to address the ever-challenging multiprocessor system-on-chip (MPSoC) hardware/software co-design and co-verification requirements. The increasingly popular deployment of MPSoC brings complexity to system modeling, design, and verification. Fiercely competitive business environment makes it absolutely critical to rein in time-to-market and chip fabrication costs. The holy grail is to be able to verify the hardware design and synthesize to the gate level for physical layout, at the same time carry out software development for the hardware design using the same system models and verification platforms. One approach is to raise the abstraction level of system design and verification to ESL. In this paper, a virtual prototyping platform is built using SystemC with transaction-level modeling (TLM) and the open virtual platforms (OVP) processor model with instruction set simulator (ISS). As a demonstration of concept and feasibility, the virtual platform prototypes a 128-bit advanced encryption standard (AES) Cryptosystem MPSoC. The supporting subsystems and environment are also modeled, for example the system peripherals, the network-based interconnect scheme or Network-on-Chip (NoC), system firmware, the interrupt service handling, and driver. The virtual platform is scalable up to but not limited to twelve processing elements and configurable to the extent of the OVPs generic memory models (RAM and ROM) addresses and sizes, simulation parameters and debugging and tracing options.

The original version of this chapter was revised: Second author name has been changed. The correction to this chapter is available at https://doi.org/10.1007/978-3-319-60170-0_18

A. Wicaksana (✉)
Department of Computer Science, Universitas Multimedia Nusantara, Tangerang, Indonesia
e-mail: arya.wicaksana@umn.ac.id

C. M. Tang
Department of Electronic Engineering, Universiti Tunku Abdul Rahman, Kampar, Malaysia
e-mail: tangcm@utar.edu.my



R. Lee (ed.), *Computer and Information Science*, Studies in Computational Intelligence 719, DOI 10.1007/978-3-319-60170-0_7

**Keywords** Virtual prototyping platform • Transaction-level modeling
Multiprocessor system-on-chip • Hardware/Software co-design
Hardware/Software co-verification

## 1 Introduction

The over increasing popular deployment of multiprocessor system-on-chip (MPSoC) today brings two major challenges: to build hardware that the software designers could use effortlessly and to develop system software which could completely utilizes the hardware potentials. The presence of many processors with their attending subsystems sets new challenges in hardware and software design. In today's extremely costly advanced technology nodes and competitive market environment, this ability to co-design and co-verify hardware and software concurrently can very easily be a matter of survival for the product design and development entity of an MPSoC. Shorter product life cycles, sky-rocketing cost, heightened time-to-market, explosive complexity, and greater competitive environment make it no longer acceptable for the software development to proceed after the hardware prototypes are made available as it used to be. They have to start concurrently. To ensure no inconsistency, the software will have to be developed using the same models and testbenches used in the synthesis and verification of the hardware design. The MPSoC designers and engineers inevitably have to face up to the new challenges that has turned into a common reality today as described in [14].

In MPSoC hardware design, new interprocessor communication scheme based on terrestrial telecommunication structure had become common place. This shift to network-on-chip (NoC) implementation was necessary as bus-based interconnect schemes were no longer able to manage multiple processors communication efficiently on the silicon level [4]. The main challenge in hardware design today is to be able to develop a hardware that is easy to use by software that meets consumer specification while satisfying market constraints e.g., reducing the cost, footprint, or power utilization. The exponential growth of the hardware and the semiconductor industries consequently brought new challenges to the software design league that never existed before. Software design complexity for multiple processors system is many times that for a single processor system. In the early days, system-on-chip (SoC) system software could only be tested after the hardware prototype was available. This was due to the limitation of design methodology and tools for both hardware and software. Thus, new methodologies and tools must be developed to enable the hardware and software designers to perform co-design and co-verification of the same design specification documents, models, and verification platforms.

The electronic system-level (ESL) design methodology as described in [3] proposed a virtual prototyping platform for solving MPSoC hardware/software co-design and co-verification problems. The virtual prototyping platform is categorized into several types based on the use cases. Each of the use cases targets specific purpose: early software development, architectural exploration, and verification.

Based on the use cases, the abstraction level of the virtual prototyping platform is defined. There are three views: programmer's view, architectural's view, and verification view. Early software development will require faster simulation speed of the virtual prototyping platform to allow software engineers to boot firmware or even operating system (OS) within reasonable amount of time. This specific need obviously requires high-speed models which implies that the models do not carry much details implemented on them, as long as the models carry all of the functionalities that are required from the programmers view standpoint. On the other hand, for architecture exploration and verification purpose, high accuracy models are more favorable to provide estimates that can be used for exploring another architecture solutions and even more accuracy for doing verification in high-level.

In general, virtual platforms can be built using high-level languages e.g., SpeC, SystemC, and SystemVerilog along with the use of transaction-level modeling (TLM) for faster simulation speed. High-level reusable models of predefined standard components such as processors and memories are provided by open virtual platform (OVP) for ease of design. Other than that, high-level abstraction models that are suitable for fast high-level exploration at early design stages are developed using TLM with loosely timed (LT) coding style. However, the more accurate models which produce most precise results can be built using TLM with approximately timed (AT) coding style or without TLM in the cycle accurate (CA) abstraction level.

The objective of this paper is to discuss a general reference virtual prototyping platform for dealing with MPSoC hardware/software co-design and co-verification. We describe the specific characteristics of a virtual prototyping platform, and the features that may drive the adoption of virtual prototyping platform by both hardware/software co-design and hardware/software co-verification. Finally, we substantiate the discussion by reporting our experience in the AES-128 Cryptosystem MPSoC project, which is a proof-of-concept deployment of a virtual prototyping platform in dealing with MPSoC hardware/software co-design and co-verification. In this regard, we describe the technical solutions adopted for the realization of the virtual prototyping platform and report some of the measurements from it.

The rest of the paper is organized as follows. Section 2 overviews the technologies that are commonly associated with the hardware/software co-design and co-verification challenges of an MPSoC and that can be enabled by the deployment of a virtual prototyping platform. Section 3 provides a general overview of the system architecture for a virtual prototyping platform. More in detail, this section describes the TLM-LT approach for the realization of the virtual prototyping platform and also the hardware models in high-level of abstraction, with the related functionalities and communication protocols. Finally, Sect. 4 presents the AES-128 Cryptosystem MPSoC project, which exemplifies a possible implementation of a virtual prototyping platform and provides examples of the type of results that helps MPSoC engineers with the hardware/software co-design and co-verification.

## 2 Virtual Prototyping Platform

A virtual prototyping platform is basically a piece of software that mimics the true functionality of a complete system. In this case, it is an MPSoC. This way, the software engineers could perform early software development based on the virtual prototyping platform before the hardware is available. The usage is not limited only to that extent. Architectural exploration, functional verification, and estimated timing analysis could also be performed using the virtual prototyping platform. Each of the use cases mentioned in [5] is best achieved by building the virtual prototyping platform at specific abstraction level. The hardware engineers need implementation-accurate models to validate their designs. Meanwhile, the software engineers can get by with high-level behavioral models. The abstraction levels are described in Table 1.

The PV models [2, 10] provide virtual platform models with simulation speeds of ranges between 100 and 500 MIPS (million instructions per second). The characteristic of this platform is functionally accurate and executes really fast within the mentioned ranges. The reason why the simulation speed is the fastest amongst all is that there is no timing information at all (untimed) within the models. This allows the simulation speeds to be faster than more timing accurate models. Thus, these PV models are sufficient enough for software developer starting early software development [8]. More timing accurate models of these PV models could be built and generally called PV + T (programmers view + timing). Since time penalties increase with timing accuracy, designers usually start from a full LT model (untimed). As the design flow continues, modules with some time accuracy are added, [8] either via decomposition or through refinement. For the embedded software developer working with the processor architect, a modification requiring a change of the instruction set was almost immediate [6]: a new instruction set simulator (ISS) was generated and the embedded software could run very rapidly on the new ISS. The reason was that the processor was modeled in C as a functional model, and some wrapper code that represented the interface and communication to the processor peripherals.

The development of these abstract models is enabled by the design modeling language: SystemC [6]. SystemC, used as a vehicle to provide the TLM abstraction, has proven to be the key to the fairly fast deployment of this methodology [6]. There was no issue of proprietary language support by only one CAD (computer-aided design) vendor or university. There was also no issue of making a

**Table 1** Model abstraction levels

| Level | Abstraction | Common names |
|---|---|---|
| Highest | Behavioral | Programmer's view (PV), untimed (UT) |
| Higher | Timed | PVT (Programmer's view + timing), loosely timed (LT), approximately timed (AT), cycle approximate |
| Lower | Cycle accurate | CA, clock accurate |
| Lowest | Implementation accurate | RTL, design simulation model (DSM) |

purchase decision by the design manager for yet another costly design tool. Ultimately, with the collaboration of ARM and Cadence Design Systems, a full-blown proposal was made to the Open SystemC Initiative (OSCI), under the name PV and PVT. Indeed Programmer view certainly echoes the intent of this new abstraction level, which is to bridge the gap between the embedded software developer and the hardware architect (hardware/software co-development). Certainly, allowing the Algorithm, hardware, software, and functional verification teams to have confidence in the same functional model is saving valuable time by avoiding misunderstandings due to informal or even formal paper-based communication.

## *2.1 SystemC and Transaction-Level Modeling*

SystemC [8, 9] is a language built in standard C++ by extending the language with a set of class libraries created for design and verification. SystemC focuses on the urgency for a system design and verification language that covers hardware and software. SystemC are applied worldwide for doing system-level modeling, abstract analog/mixed-signal modeling, architectural exploration, performance modeling, software development, functional verification, and high-level synthesis [11]. The language is defined by OSCI and ratified as IEEE Std. 1666TM-2011 [8].

TLM standard interfaces for SystemC supplies an important framework required for model exchange within companies and across the IP supply chain [8]. This is intended for architecture analysis, software development, performance analysis, and hardware verification [8, 11]. It explicitly focuses on virtual prototyping in which SystemC models can be exchanged and organized with no difficulty within a system. This is achieved by providing a strong modeling foundation for virtual prototyping. The standard allows optimal reuse of models and modeling effort across different use cases [11].

Use cases have been categorized according to a range of criteria, leading to standard interfaces differentiated by loosely timed (LT) and approximately timed (AT) modeling styles. The extended APIs provide a fundamental, general purpose interoperability layer. A specific payload (generic payload or transaction), to be used in conjunction with these interfaces, helps achieve a higher degree of interoperability when generically modeling memory-mapped bus-based components. Several TLM features boost simulation performance enabling what is called speed interoperability in addition to model interoperability for SystemC virtual platforms. Temporal decoupling allows initiator models, such as instruction set simulators, to run ahead of the SystemC kernel and synchronize only periodically to significantly reduce the required number of costly context switches. The direct memory interface allows interconnect models to be bypassed, facilitating high-speed access to modeled memory. A dedicated transaction debug interface ensures that debugging is an integral part of a system model while enabling debug activity without interference with the system simulation [11]. Figure 1 shows the TLM uses cases with the coding styles, abstractions, and mechanisms.

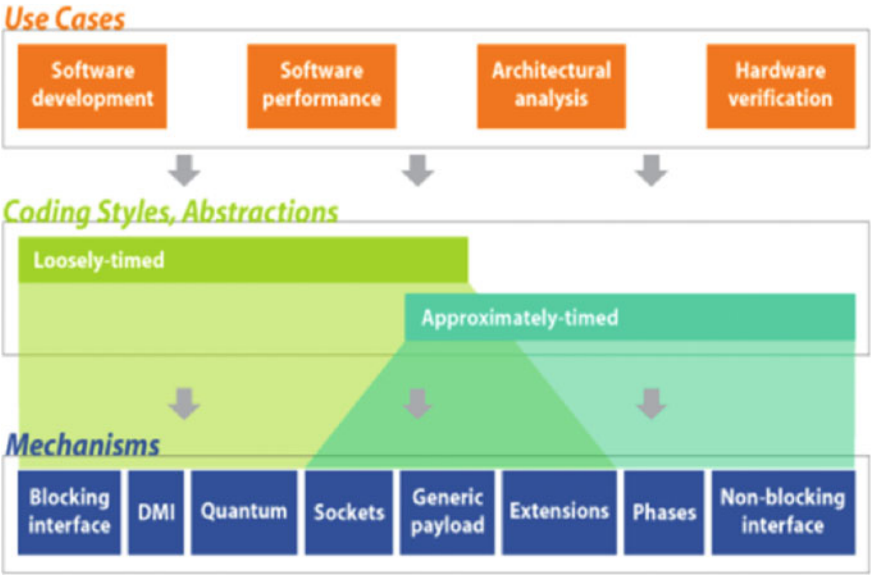


**Fig. 1** TLM use cases, coding styles, and mechanisms [11]

Different suppliers have adopted TLM broadly due to its interoperability of transaction-level models without compromise to simulation speed. It provides an essential ESL framework for architecture analysis, software development, software performance analysis, and hardware verification. Many companies across the ESL ecosystem already incorporate TLM in their products and solutions [11].

## 2.2 *Open Virtual Platform*

The open virtual platform [12] is the source of Fast Processor Models and Platforms. The focus of OVP is to accelerate the adoption of the new way to develop embedded software, especially for SoC and MPSoC. OVP has three main components: Open Source Models, OVPsim/Igen/ISS tools, and Modeling APIs. These components makes it very easy to put together advanced multicore heterogeneous or homogeneous platforms with complex memory hierarchies and layers of embedded software, that run at at 100 s of MIPS on standard desktop PCs [12].

The open source models are distributed in several different model categories: pre-compiled object code and source files. Currently, there are processor models of ARC, ARM, MIPS, PowerPC, Renesas, Altera, Xilinx, and OpenRisc families [12]. There are also models of many different types of system components including RAM (Random-access Memory), ROM (Read-only Memory), trap, cache, and bridge. There also peripheral models including DMA (direct memory access), UART (universal asynchronous receiver/transmitter), FIFO (first in, first out),

ethernet, and USB (Universal Serial Bus). There are also models of several different pre-built platforms which run operating systems including Linux, Android, MQX, Nucleus, Micrium, and FreeRTOS [12].

The OVPsim simulator is a very fast simulator and is currently released on 32-bit Windows and Linux. It provides the simulation capabilities to run platforms of OVP processor and peripheral models at very fast speeds. The simulator is a just-in-time (JIT) code morphing simulator engine that translates target instructions to x86 host instructions dynamically. It has been specifically designed for the fastest simulation throughput and includes many optimizations enabling simulation of platforms utilizing many homogeneous and heterogeneous processors with many complex memory hierarchies. OVPsim includes very efficient modeling of MMU/TLBs (memory management unit / translation lookaside buffer) and hardware virtualization [12].

OVPsim can be wrapped and called from within other simulation environments and comes as standard with wrappers for C, C++, and SystemC. Another key technology component of OVPsim is that it can encapsulate existing binary models of processors and behavioral models. Thus, the utilization of existing legacy processor models in an OVP simulation is not difficult. OVPsim comes with a GDB (GNU Project Debugger) RSP (Remote Serial Protocol) interface and is easy to use with standard debuggers that support this GDB RSP interface [12]. Moreover, the OVPsim is highly suitable for usage in many educational environment due to its tight research budget characteristic.

## 3 Virtual Platform Architecture

The virtual platform general specification is to prototype the AES-128 Cryptosystem MPSoC. This includes hardware and software models that are designed and engineered to work together. SystemC, TLM, and OVP technologies are used to build the virtual platform including the hardware system. The architecture of the virtual platform is defined in this section which contains the system partitioning task, the hardware specifications and functionalities, and the software specifications and functionalities. The architecture of the virtual platform (hardware and software) is defined according to the case study presented in Sect. 4.

The cryptosystem takes plaintext as the input data and produces ciphertext as the output data. The size of the data is 128 bit. In the real world, the data will be inserted into the system by another system. The encryption algorithm used within the system is AES. The architecture of the virtual platform is designed to be scalable and configurable to certain extent. The scalability factor is the number of processing elements (PEs). Thus, the user of this virtual platform is able to specify the number of processing elements available within the system during the virtual platform execution (runtime). The configurable factor is the start and end addresses of the memories (RAM and ROM) within the system. The size of the RAM could also be modified on the fly.

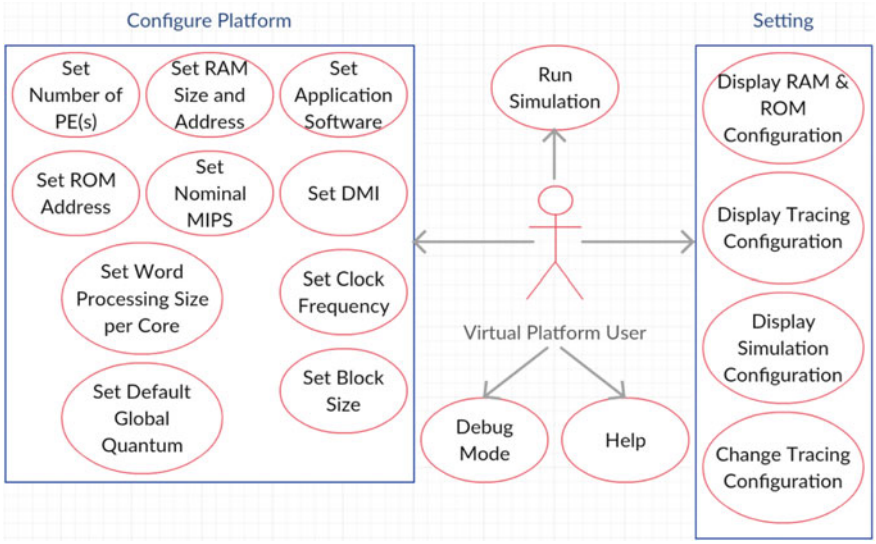


**Fig. 2** Virtual platform use case diagram

## *3.1 Virtual Platform Specification*

The virtual platform is built using TLM-2.0 LT coding style. The functional specifications of the virtual platform are presented in Fig. 2 using UML (Unified Modeling Language) 2.0 use case diagram for the sake of simplicity and presentation of this paper.

OVPs ARM Cortex-M0 processor models are used in the virtual platform, one in each processing element. Additional features of TLM i.e., temporal decoupling and DMI (direct memory interface) are used within the virtual platform to speed up simulation speed.

The prototyped system is partitioned into hardware and software. The implementation of the encryption is done in the hardware. The software sets the encryption key, and the plain text is from another hardware that is called host device in this paper. The host device is developed as a TLM module and is part of the system-level testbench. The functionalities of the virtual platform is built to provide scalability on the hardware architecture that is the number of the processing element and the configurability on the start and end addresses of the memories.

## *3.2 Hardware Architecture*

The default hardware architecture with four processing elements is displayed in Fig. 3. The processing elements are divided into two categories: central and

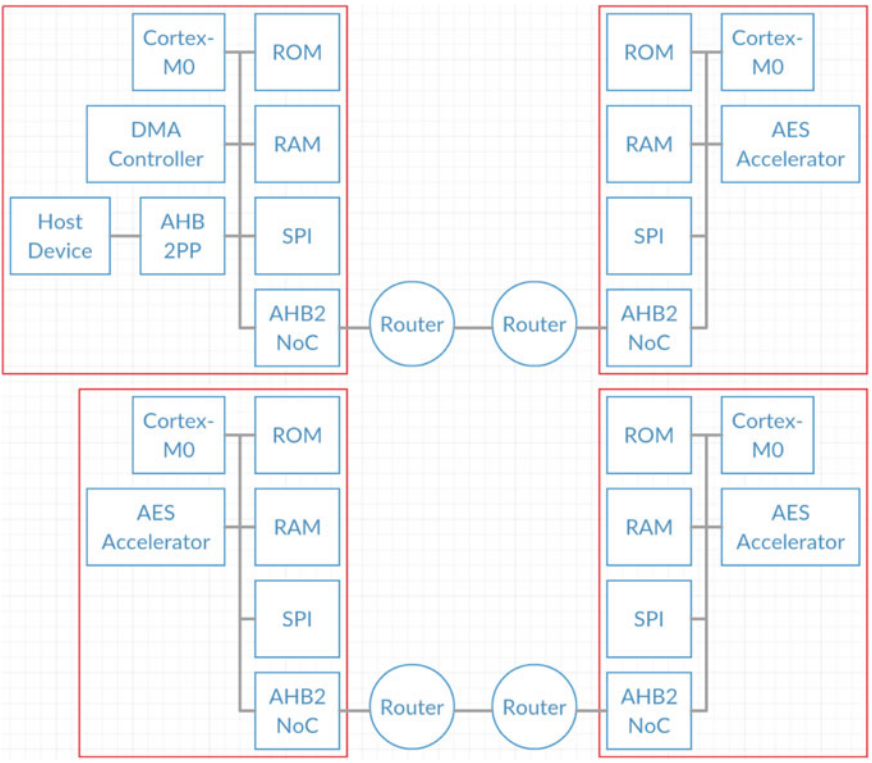


**Fig. 3** Virtual platform default hardware architecture

encryption. The central PE contains a DMA controller module, an AHB2PP (AHB to parallel port) module. The other encryption PEs pack an AES accelerator module each. The interconnect system between all of the PEs is NoC with 2D mesh architecture. The bridge between bus-based interconnect system and the NoC is the AHB2NoC (AHB to NoC) Module. The hardware system is attached to an external module host device as part of the system-level testbench. The Cortex-M0 Processor, ROM, RAM, and bus-based interconnect system are used from the OVP model repository.

The development of the hardware models other than stated previously is divided into three categories: TLM initiator module, TLM target module, and TLM interconnect module. The initiator module initiates transactions and sends it to other modules. The target module accepts transactions and reacts based on the transactions. In timed simulation, the additional delay occurred because of the operation done by the module is added into the transaction. The interconnect module simply resembles the function of an interconnect system, that is to pass the transaction without modifying it, except adding additional timing delay in the timed simulation.

The use of the TLM-2.0 mechanisms increases the modules interoperability and reusability. This has been proven with the integration of OVPs high-level models

with the rest of the models in this virtual prototyping platform. A wrapper modules are also developed i.e., the AES accelerator module that wraps the Texas instruments AES model implemented in C programming language [1, 7]. The DMA controller module has four independent channels to support data transfer between processing elements. The implementation of these channels uses `SC_THREAD` to provide the multithreading capability for each of the channels. This way the simulation has several active threads including the processor and the DMA controller modules active channel.

## 3.3 Software Architecture

The software architecture of the system is highly coupled with the hardware architecture. There is two type of firmware required by the system: central PE firmware (Fig. 4) and encryption PE firmware (Fig. 5). Other than the firmware, the software includes drivers, ISRs (interrupt service routines), and linker scripts.

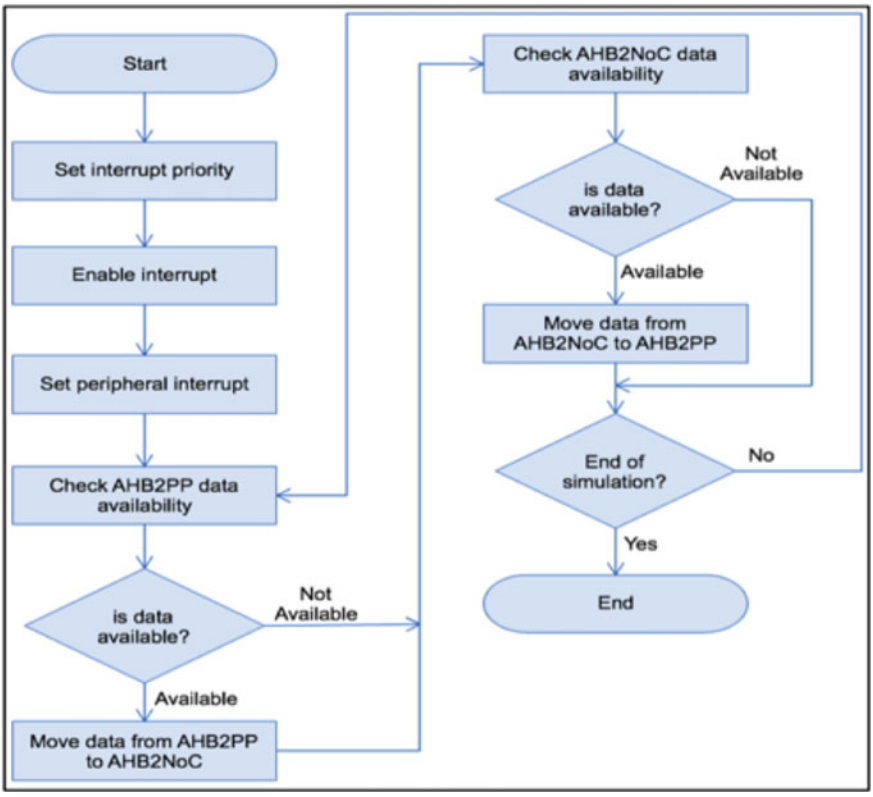


**Fig. 4** Central PE firmware flowchart diagram

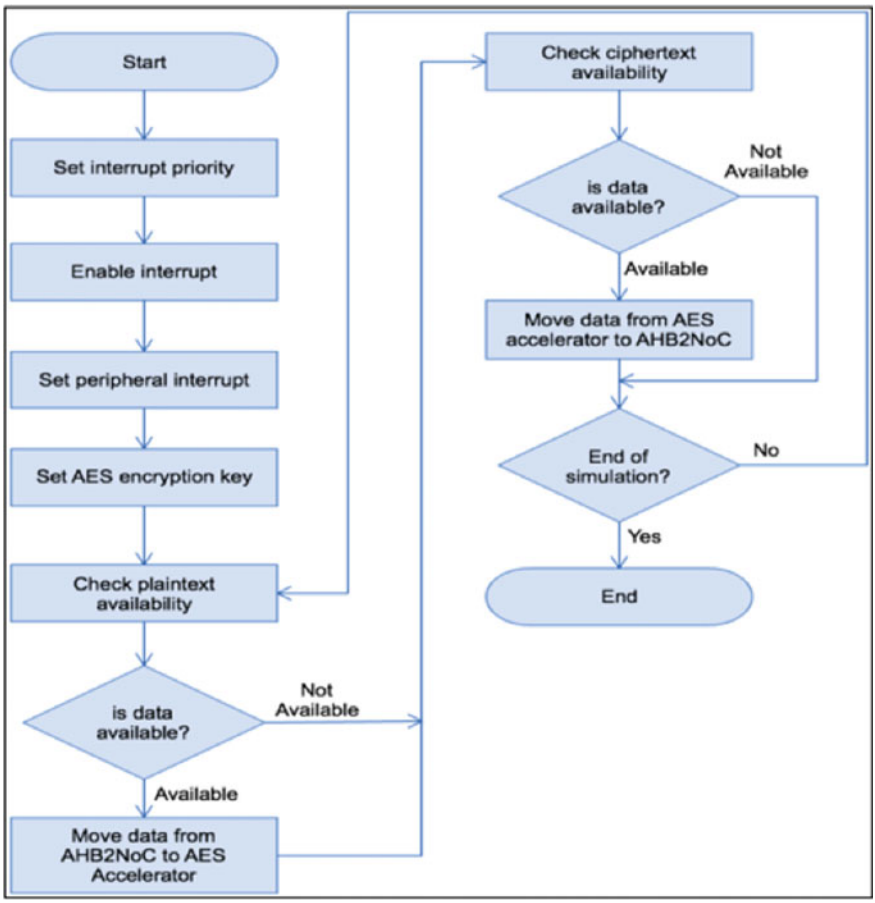


**Fig. 5** Encryption PE firmware flowchart diagram

There are five drivers for each of the hardware peripherals: AHB2PP, AHB2-NoC, SPI (Serial Peripheral Interface), DMA controller, and AES accelerator. These drivers provide the interfaces for both central and encryption PE firmware to access the peripherals internal registers and perform certain tasks. The drivers are cross compiled together with the firmware, ISR, and linker script into an executable binary file using Sourcery CodeBench Lite Edition. The whole encryption process is carried out by multiple processors and programmed by the two firmware. The pure encryption task is done by the encryption PE firmware. Among the encryption PEs, the division of the task is accomplished by coarse-grained parallelism implementation.

**Table 2** AES-128 encryption test-cases

| Test-case | Plaintext | Ciphertext |
|---|---|---|
| Gladman | 0x3243f6a8_885a308d_313198a2_e0370734 | 0x1995a9f6_764a79a4_acddc008_b805298a |
| All Zero | 0x00000000_00000000_00000000_00000000 | 0x8ffb667f_00df6bdd_5a224bdf_8f1d325b |
| AES Test Specification | 0x00112233_44556677_8899aabb_ccddeeff | 0x4c1cce19_def4305c_83bcf0d1_355074f8 |

### *3.4 System-Level Testbench*

The virtual platform is merely a platform that needs a testbench in order to verify the system-level functionalities of the virtual platform and also the MPSoC that it prototypes. A TLM initiator module is needed to drive the test-cases to the virtual platform. The test-cases is used from [13], and it includes Gladman Test-case, All Zero Test-case, and AES Specification Test-case. The TLM initiator module is named host device and the main task is to deliver the test-cases to the system and verifies the results once they are produced. It also specially carried out the test of the AES algorithm implementation in the AES Accelerator Module. The details of the test-cases used is shown in Table 2. The key used for the encryption is 0x12345678_12345678_12345678_12345678.

The testbench is composed of a set of AES-128 encryption test-cases and the host device module. The implementation of the host device module uses TLM-2.0 LT to match with the rest of the virtual platforms models. The testbench prints out an information to the terminal at the beginning of the testing process. The testbench then controls the repetition number and step of the host device module to completely and randomly transfers all of the test-cases according to the block size value that has been set earlier by the user in the configure platform menu. The simulation is kept alive untill all of the ciphertext are received back by the testbench (via the host device Module). At the end, the testbench displays the information about the test result and quits.

## 4 A Case Study: AES-128 Cryptosystem MPSoC

The simulation and testing of the virtual prototyping platform is carried on a virtual machine with Centos Red Hat (64-bit) operating system, with 8 GB of RAM DDR3 and an Intel Core i7 processor (2.2 GHz). In addition, five AES-128 benchmark software developed by the UTAR VLSI Research Centre for the RUMPS401 are used for the simulation and testing. They are: AES-128 Benchmark-16, AES-128 Benchmark-32, AES-128 Benchmark-64, AES-128 Benchmark-128, and AES-128 Benchmark-256. These benchmark software are used to measure the functional correctness of the virtual prototyping platform. The simulation is run using the virtual platforms configurations as shown in Table 3.

**Table 3** Lotus-G configuration for correctness testing

| Parameter | Value |
|---|---|
| PEs | 4 |
| MIPS | 100 |
| Quantum | 1 μs |
| DMI | Yes |
| Clock frequency | 0 |
| Temporal decoupling | Yes |
| Tracing and instrumentation | Off |

In this section, the result of practical implementation of a virtual prototyping platform is described. The virtual prototyping platform provides the abilities for the MPSoC engineers (both hardware and software engineers) to perform hardware/software co-design and co-verification specifically: early software development, architectural exploration, estimated timing analysis, and functional verification.

## *4.1 Early Software Development*

The virtual prototyping platform can be used to develop new software before the RTL platform or the hardware prototype is available. This is achieved by the development of the AES-128 Benchmark-DMA and AES-128 Benchmark-512. This is one of the key result that is important to note: the ability to design the software concurrently with the hardware. There are three architectures that are used to develop the software: 4 PEs, 8 PEs, and 12 PEs. For each of the architectures, different encryption block size is used as the parameter in developing the software, i.e., The 12 PEs and 256 encryption block size uses 9 Main0.c and 2 Main1.c, where each serves 24 blocks and 20 blocks, respectively.

## *4.2 Architectural Exploration*

The simulation and testing performed shows the use of the virtual prototyping platform for system designers to design, merge, and optimize complex systems meeting design specifications such as functionality and performance. Performance could be investigated together with functional verification and software execution.

The DMA controller module is attached and unattached in various configurations as displayed in Table 4. This is performed to demonstrate the ability of the virtual prototyping platform in enabling architecture exploration. The performance results of different hardware architectures are shown in Table 5 and the parameters of the simulations are displayed in Table 6.

The LT simulations give more timing accurate results compare to the untimed simulations. The timing delay produced by the models during the LT simulations is not 100% precise, and it is just an estimate used for the high-level modeling purpose.

**Table 4** Architectural exploration test results

| Simulation | DMA Controller Module | Untimed (s) | LT (s) |
|---|---|---|---|
| 1st | None | 0.13 | 23.41 |
| 2nd | One in Central PE | 3.40 | 11.34 |
| 3rd | One in each of the PEs | 3.52 | 13.22 |

**Table 5** Architectural exploration test parameters

| Parameter | Value |
|---|---|
| Number of PEs | 4 |
| Encryption block size | 256 |
| MIPS | 1 μs |
| Global quantum size | 100 μs |
| Clock frequency | 32 MHz (LT) |
| DMI feature | On |
| Tracing and instrumentation features | Both off |

**Table 6** Tested parameters range

| Parameter | Range value |
|---|---|
| Number of PEs | 1–12 |
| Encryption block size | 16–512 |
| MIPS | 1–100 |
| Global quantum size | 1 μs–1 s |
| Clock frequency | 16 MHz (LT) or 32 MHz (LT) |
| DMI feature | On or off |
| Tracing feature | On or off |
| Instrumentation feature | On or off |

Thus, more accurate delay time figures could be used to replace the existing value when available. In Table 4, the fastest LT simulations is achieved in the 2nd simulation, where the architecture has only one DMA Controller Module in the Central PE. Deeper and broader exploration could be performed further i.e., varying the number of the PEs, the interconnect systems, and the use of specific hardware accelerator modules in order to get the best architecture for the targeted application. Exploration on the software side could also be performed i.e., the firmware, driver and ISR, specifically the task division, the parallelism and the algorithm.

## *4.3 Estimated Timing Analysis*

The virtual prototyping platform could give estimated timing to the simulation based on the LT implementation of the TLM models. This enables estimated timing analysis to be done. Both hardware and software engineers could observe the consumed time based on the LT simulation. An example is shown in Fig. 6 where

```
Info ------------------------------------------------------
Info SIMULATION TIME STATISTICS
Info   Simulated time        : 0.01 seconds
Info   User time             : 3.52 seconds
Info   System time           : 0.00 seconds
Info   Elapsed time          : 3.52 seconds
Info ------------------------------------------------------

OVPsim finished: Wed Jun 15 21:09:54 2016
```

**Fig. 6** Simulation time statistics example

the simulation time statistics contains information i.e., simulated time, user time, system time, and elapsed time. Simulated time is the simulation duration in simulated time (LT). User time is the wall-clock time that is assigned to the simulator. System time is the time used for doing system chores on behalf of the simulator process. Elapsed time matches the wall clock from the start of the simulation until the simulation time statistics line is printed.

### *4.4 Functional Verification*

The virtual prototyping platform serves as a tool to verify the functionality of the hardware concurrently with the software. During the simulation, the hardware and the software are essentially verified together in order to achieve the expected outcomes and to behave correctly according to the system functional specifications. A semantic error caused by the software will cause the hardware to misbehave, hence stops the simulation. An error in the hardware will also cause the simulation to hang, and the error is observable by activating the tracing feature.

The DMA controller RTL simulation was performed using Synopsys VCS with a SystemVerilog HVL (hardware verification language) testbench created following the UVM (Universal Verification Methodology). The result compared well with that obtained using the high-level virtual platform.

## 5 Conclusion

The virtual prototyping platform was built using TLM in SystemC to enable the MPSoC hardware/software co-design and co-verification. It was demonstrated to work successfully in early software development, architectural exploration, estimated timing analysis, and functional verification, all using the same high-level models. It is also scalable and configurable, yet fast and accurate enough. In future, this technique will be extended to IoT (internet of things) systems with cryptographic security system development in mind.

**Acknowledgements** My deepest gratitude to my supervisor Mr. Tang Chong Ming, Prof. Lee Sze Wei, and Mr. Ng Mow Song for the support of this research, for the encouragement, enthusiasm, and immense knowledge. This would not have been possible without their guidance and support.

## References

1. AES-128 Advanced Encryption Standard | TI.com. http://www.ti.com/tool/aes-128. Accessed 17 Feb 2017
2. Bailey, B., Martin, G.: ESL Models and Their Application. Springer, Boston (2010)
3. Bailey, B., Piziali, A.: ESL Design and Verification. Morgan Kaufmann, Amsterdam (2010)
4. Cota, E., et al.: Reliability, Availability and Serviceability of Networks-on-Chip. Springer, Boston (2012)
5. Engblom, J., Aarno, D.: Software and System Development Using Virtual Platforms: Full-system Simulation With Wind River Simics. Morgan Kaufmann Publishers (2015)
6. Ghenassia, F.: Transaction-Level Modeling with SystemC. Springer, Dordrecht (2005)
7. Hall, J.: C Implementation of Cryptographic Algorithms. Texas Instruments (2013)
8. IEEE Standard for Standard SystemC Language Reference Manual. Institute of Electrical and Electronics Engineers, New York (2012)
9. Leupers, R., Temam, O.: Processor and System-on-Chip Simulation. Springer (2010)
10. SystemC Community. http://accellera.org/community/systemc. Accessed 17 Feb 2017
11. SystemC TLM. http://accellera.org/community/systemc/about-systemc-tlm. Accessed 17 Feb 2017
12. Technology. http://www.ovpworld.org/technology. Accessed 17 Feb 2017
13. Wagner, N.: The Laws of Cryptography: Test Runs of the AES Algorithm. http://www.cs.utsa.edu/~wagner/laws/AEStestRuns.html. Accessed 17 Feb 2017
14. Wolf, W., Jerraya, A.: Multiprocessor Systems on Chips. Elsevier, Amsterdam (2005)