Imperial Journal of Interdisciplinary Research (IJIR)
Vol-3, Issue-4, 2017
ISSN: 2454-1362, http://www.onlinejournal.in

Efficient Multi-Core Virtual Platform for High Speed Full-System Simulation

Ahmed Abdel-Haleem[1], Magdy A. El-Moursy[2] &
Mohamed Dessouky[3]
[1]Mentor Graphics, Egypt
[2]Mentor Graphics, Egypt
Electronics Research Institute, Egypt
[3]Mentor Graphics, Egypt
Ain Shams University, Egypt

Abstract*: Multilateral Virtual Platforms (VPs) are increasingly adopted in industry to simulate today's complex MPSoC designs. In this paper, a fast, yet accurate, TLM VP is developed and validated for full-system simulation and continuous SW integration. The TLM VP contains an Instruction Set Simulator (ISS) that efficiently utilizes underlying multi-core host processor to run multi-thread applications at high speed. The ISS is based on Dynamic binary Translation (DBT) technology. A novel SystemC synchronization approach among emulated target cores is adopted. Using SPLASH-2 and EEMBC CoreMark benchmarks, the presented TLM VP, emulating the NXP Sabre-Lite IMX6Q Quad-Core Cortex-A9 board, achieves on average 3.9X speedup over baseline QEMU that is equivalent to 15% of the corresponding real HW board speed.*

Keywords: *Virtual Platform, Multi-core, TLM, ISS*

1. Introduction

Virtual Platforms (VPs) are leading the Electronic System Level (ESL) design methodology mutation for early SW development, SW/HW co-verification, architectural exploration and performance analysis. With the growing complexity of embedded systems, higher abstraction level VPs of MPSoC designs became a necessity in order to address various design challenges. Early bug detection during product life cycle, which is provided by VPs, can help reducing overall costs and time-to-market accordingly [1].

VPs modeled at Transaction Level (TLM VPs) provide an optimized solution in terms of simulation speed and timing accuracy. Unlike traditional cycle-accurate and RTL models, TLM VPs can achieve much higher simulation speeds due to event-driven nature of the simulation kernel and modeling granularity at transaction level [2]. Moreover, Multi-core TLM VPs offer a performance boost through efficient utilization of today's multi-core host processors to run multi-threaded applications. On the other hand, Loosely-Timed (LT) and Approximately-Timed (AT) modes which are introduced by TLM-2.0 standard [3] can sustain diverse levels of timing accuracy.

In this paper, a multi-core TLM VP of the NXP Sabre-Lite IMX6Q platform is developed (as shown in Fig. 1) introducing the novel capabilities of high-speed and full-system simulation using Vista tool by Mentor Graphics [4]. The performance of the TLM VP running in LT simulation mode is compared against corresponding QEMU emulated Sabre-Lite machine and HW Sabre-Lite board.

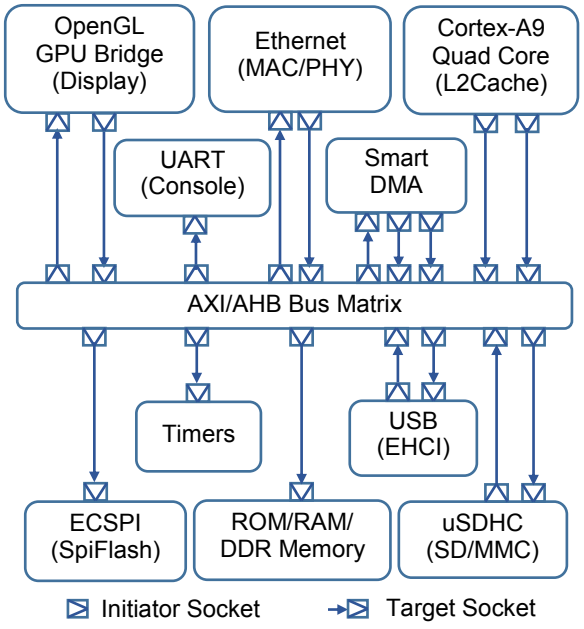

Figure 1. Block diagram of the developed Sabre-Lite IMX6Q TLM VP (not all blocks are shown)

Therefore, the main contributions of this paper are two-fold as follows.

1) A fast, yet accurate, multi-core Instruction Set Simulator (ISS) modeling methodology that exploits underlying host cores for high speed TLM VP simulation is introduced.
2) A Full-system TLM VP instantiating the multi-core ISS along with associated modeling techniques are presented.

The remainder of this paper is organized as follows. Background and related work are briefly discussed in section 2. Design methodology is presented in section 3. Experimental results are summarized in section 4. Finally, conclusions are presented in section 5.

2. Background and Related Work

In this section, QEMU is used as an example to briefly describe key techniques used in full-system emulation based on Dynamic Binary Translation (DBT). An overview of SystemC and TLM modeling methodology is presented. Also, relationship between this work and previous work is summarized.

2.1. Dynamic Binary Translation

QEMU main loop translates blocks of code at run-time as needed (Just-In-Time compilation) into intermediate operations through its Tiny Code Generator (TCG) front-end. These operations are then transformed into host instructions for later execution through its TCG back-end. The first time a Translation Block (TB) is generated, it is stored in a translation cache for better reuse. These TBs can be chained together dynamically in a continuous TB execution scheme for an enhanced performance.

QEMU emulates multi-core processors in a sequential round-robin fashion. Each emulated core can execute in a certain time slot, after which it yields the physical CPU to the next emulated core. Hence, atomic instructions are not necessarily emulated; as scheduling at basic blocks guarantees atomic execution per instruction. Furthermore, the time-slicing of cores results in bad performance scalability due to the serialization of parallel code.

2.2. SystemC and TLM

Unlike C/C++, SystemC provides enhanced modeling capabilities to emulate hardware features such as concurrency, events and signals. The standard TLM interface of the Open SystemC Initiative (OSCI) drives the interest of using SystemC in VP modeling. SystemC library includes an event-driven simulation kernel suitable for high abstraction level modeling. TLM separates the communication details from the functional implementation among various VP models [5]. Device interfaces are modeled using *target* and *initiator* sockets which are introduced by the TLM-2.0 standard. Also, device registers and their associated functionality are modeled within callback functions in response to *read* and *write* transactions that are received through corresponding TLM *target* socket. A TLM model utilizes an *initiator* socket for issuing *read* and *write* transactions to various addressable devices in the VP.

2.3. Related Work

The importance of high-speed full-system emulation has been confirmed by the considerable amount of effort committed by both industry and research communities in developing relevant designing tools. QEMU [6] is an example of sequential system emulators that lack parallel execution capability which can exploit the underlying host multi-core processor. Several attempts have been introduced to parallelize QEMU through assigning multiple QEMU instances to multiple threads such as COREMU [7]. Also, PQEMU [8] introduces a locking mechanism to serialize accesses to the shared components. In addition, some arrangement of critical sections and usage of LLVM optimizations are proposed in HQEMU [9].

Nevertheless, all of the above mentioned DBT-based emulators do not incorporate neither a standard SystemC TLM communication interface nor any level of timing accuracy needed for design space exploration. On the other hand, other approaches have been demonstrated that take the advantages of both TLM standard communication mechanism and fast DBT-based ISSs. For instance, QBox and QEMU-SC projects combine the usage of QEMU and SystemC in a single simulation environment [10]. In QBox, QEMU is treated as a SystemC module within a larger SystemC simulation context. SystemC simulation kernel remains the *master* of the simulation. Whereas in QEMU-SC, QEMU is the *master* during simulation execution. Also, it contains the emulated platform, instantiates and controls the SystemC part. However, such approaches are not successful till the moment to exploit parallelism which is offered by the underlying multi-core host machines in order to execute multi-threaded guest applications efficiently.

A similar approach is adopted in this paper to that mentioned in COREMU in the concept of multiple QEMU instantiation. However, in the presented approach, QEMU is instantiated within advanced SystemC TLM wrappers that provide a comprehensive multi-thread synchronization layer

through which concurrent host multi-core execution capabilities are leveraged. Actually, the proposed approach achieves a very close speedup factor, over baseline QEMU [11], to that announced in PQEMU at four parallel running threads. On the contrary, a positive speedup factor is introduced for single-thread execution instead of the introduced overhead of PQEMU using SPLASH-2 [12] benchmarks.

3. Design Methodology

In this section, modeling methodology of the ARM Cortex-A9MP multi-core ISS which is adopted by Vista Architect [4] is elaborated. Also, IMX6Q TLM VP modeling aspects are described.

3.1. ISS Topology

As depicted in Fig. 2, the ISS consists of two main components; functional core and custom peripherals. These components are instantiated inside a TLM-2.0-compliant wrapper, thus ensuring full interoperability with the TLM VP. The functional core contains the DBT engine (an old QEMU version with some modifications) that supports a memory management unit (MMU), a floating point unit (FPU) in addition to the instruction translation infrastructure. For each emulated core, a SystemC wrapper is developed to load a single instance of the DBT engine dynamically and control all I/O operations to/from it through a predefined set of API functions. These I/O operations include *read* and *write* transactions to different addressable devices, cache control operations, peripheral interrupts and reset requests. A novel asynchronous thread control module is designed within the functional core to synchronize multiple SystemC wrapper instances for an efficient multi-core processor emulation.

Custom peripherals are pure SystemC TLM models developed in accordance with ARM Cortex-A9MP specification [13]. They include a Snoop Control Unit (SCU), Generic Interrupt Controller (GIC) and timers/watchdog models for a complete functional behavior. Moreover, Level1 *instruction* and *data* caches and Level2 cache controller complying with ARM PL310 specification [14] are modeled. Furthermore, timing/power information which are specified through external files can be parsed by the SystemC wrapper for architectural exploration purpose that is mostly beneficial in AT simulation mode of the TLM VP.

3.2. Multi-core ISS on Multi-core Host

Unlike some approaches that aims at parallelizing the standard OSCI SystemC simulation kernel [15] for a faster simulation, different approach is adopted in this paper. The presented approach is based mainly on the use of POSIX (Portable Operating Systems Interface) threads or shortly *Pthreads* and TLM-2.0 Temporal Decoupling (TD) concept. These techniques are used to model target cores utilizing underlying host machine cores along with an intensive multi-thread synchronization layer. Hence, modifying the widely-used OSCI kernel is avoided for today's safety-critical applications.

POSIX is an IEEE standard that defines the interfaces or APIs between the operating system (OS) and the application [16]. The developed TLM VP utilizes the parallel execution capability of *Pthreads.* Therefore, the ISS can control multi-threaded applications through calling the POSIX APIs which are implemented by the underlying host OS. Consequently, a *Pthread* is dynamically created inside the SystemC wrapper for each emulated core within a *SC_THREAD* to execute the corresponding DBT engine main loop. Each *SC_THREAD* is responsible for modeling the timing of the corresponding core. Hence, parallel processes can be scheduled separately for each target core at run-time by the underlying host OS.

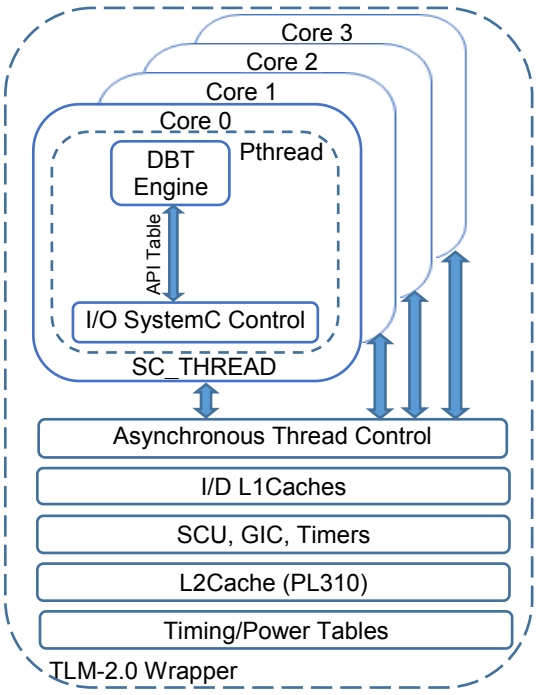

Figure 2. Basic components of the modeled quad-core Cortex-A9MP ISS

TD is a TLM-2.0 feature that allows processes to run ahead of time for a certain quantum before having to synchronize again with simulation time by calling *wait* [3]. The multi-core ISS exploits this feature for a performance boost in two ways. First, calling *wait* less often reduces simulation overhead and thereby speeds up simulation. Second, the DBT

engine performs better when executing a chunk of multiple target instructions at once in a quantum.

Combined use of *Pthreads* and TD in the TLM VP, allows multiple parallel threads to run freely in future within a reasonably high quantum value before synchronizing with the SystemC time again. Hence, efficiently utilizing the underlying host multi-core processor for a high-speed ISS.

3.3. ISS Multi-thread Synchronization

A novel approach is adopted in this paper for multi-thread synchronization. Since a separate DBT engine (QEMU) instance is instantiated inside the SystemC wrapper of each emulated core, multi-thread synchronization is handled in the SystemC side (no shared translation cache). An Asynchronous Thread Control (ATC) SystemC module is used to synchronize multiple parallel running threads through carrying out the following tasks:

Timing synchronization: ATC module keeps track of the local time of each running *Pthread* and performs necessary synchronization with the SystemC simulation time. Thus, a smart technique is adopted using atomic operations and instruction count instrumentation within the DBT engine to calculate next synchronization timing point. This technique does not use a clock for synchronization, and hence the name *asynchronous*, for an enhanced simulation performance.

Requests scheduling: I/O transaction requests to/from the DBT engine that include shared resources accesses are handled sequentially by the ATC module. These requests are scheduled to be executed only at *safe* SystemC time stamps at which no other I/O operations are in progress. Moreover, the proposed implementation supports thread joining, *semaphores* and *mutexes* to ensure thread-safe operation among various *Pthreads*.

Atomic instructions: The ISS provides two options to model atomic instructions like exclusive load/store or atomics swaps. First option is *async_fast* mode in which atomic instructions are implemented using x86 atomic instructions. Although this option is non-deterministic, it is faster when many atomic instructions are executed. Second option is *async_stable* mode in which each atomic instruction causes synchronization. Also, *spinlocks* are used to avoid race conditions during execution.

3.4. IMX6Q TLM VP Modeling Aspects

A TLM VP, that supports most of the blocks according to IMX6DQRM [17], is developed. Each model in the VP is fully TLM-2.0-compliant SystemC model. It contains Programmable registers definition, interfaces declaration, read/write callback Programmer's View (PV) functionality and run-time configurable parameters. Some VP models instantiate virtual I/O devices which are provided by the tool for user interactions. Furthermore, timing policies for architectural exploration are optionally supported in AT simulation mode.

VP models are optimized for high performance using *SC_METHODs* instead of *SC_THREADs* and other SystemC efficient coding guidelines. VP models are assembled together through many TLM connections via a graphical editor of Vista tool that auto-generates the top module. Individual SystemC test-benches are built to exercise various features in most of the developed models. Moreover, VP operation is verified through running open-source full-fledged software like u-boot v2014.07 and Linux kernel v3.17, commercial software like Mentor Graphics MEL 2014.05, Nucleus Ready-Start v2013.08.1 demos and a variety of networking and graphical Linux applications. The developed Sabre-Lite IMX6Q TLM VP contains the following models beside the Cortex-A9MP ISS, as shown in Fig. 1:

- Graphical display unit: Contains a partial implementation for the Image Processing Unit (IPU) that instantiates Simple Direct-Media Layer (SDL) client virtual I/O for graphical display. Also, it uses OpenGL Graphics Processing Unit (GPU) bridge that exploits host GPU for an accelerated video display.
- Ethernet (ENET): The MAC layer provides compatibility with half- or full-duplex 10/100 megabit Ethernet LANs and full-duplex gigabit Ethernet LANs. Also, a Virtual LAN (VLAN) client I/O is instantiated for connectivity to the host network.
- UART controller: Universal Asynchronous Receiver/Transmitter that provides serial communication capability with a virtual I/O character display console.
- ECSPI/uSDHC: Enhanced Configurable Serial Peripheral Interface and Ultra Secured Digital Host Controller models that instantiate and control virtual I/O SPI flash and SD/MMC memory block image devices respectively.
- USB: Universal Serial Bus controller provides high performance functionality that conforms to the USB 2.0 specification. It instantiates Enhanced Host Controller Interface (EHCI) virtual I/O that can connect to the host mouse, keyboard or to a virtual mass storage device.
- Other peripherals: That include Smart DMA supporting a micro RISC engine for enhanced performance, timers, buses supporting TLM AXI/AHB protocols and memory models supporting *blocking transport, transport debug and DMI* TLM interfaces.

3.5. Limitations

Multi-thread software running on top of the TLM VP that performs large number of I/O operations, for example during Linux boot, may suffer a bit slowness due to expensive run-time synchronization. Also, timing accuracy may be reduced in LT simulation mode when using very high quantum values, though it might be acceptable when only PV functional behavior is observed.

4. Experimental Results

Experimental setup and configurations are listed in Table 1. SPLASH-2 and EEMBC CoreMark [18] benchmarks are used to compare the performance of the TLM VP with respect to both baseline QEMU and HW Sabre-Lite board with different workloads.

4.1. QEMU Comparison

Wall clock execution times of SPLASH-2 benchmarks are measured at one and four running threads on top of both QEMU and TLM VP. As depicted in Fig. 3, the average performance speedup of the TLM VP is 3.689X (~3.7X) in case of four parallel threads and 1.2X in case of a single thread. For more reliable results, EEMBC CoreMark benchmark is used as well. The execution times of multiple cross-compiled CoreMark applications, each with a different number of parallel threads varying from one to eight, are measured as shown in Fig. 4. As the number of parallel threads increases, speedup increases as well which is expected due to the sequential execution nature of QEMU. After that, speedup tends to saturate due to existing number of TLM VP cores (four) and host cores (eight). Therefore, the speedup factor at four running workloads using CoreMark is 4.125X and at a single running workload is 1.096X. Combining both results of SPLASH-2 and CoreMark benchmarks by taking the geometric mean produces 3.9X speedup factor at four parallel threads and 1.15X at a single thread.

4.2. HW Board Comparison

As discussed earlier, VPs do not only provide an early execution platform for the SW, but also a versatile cost-saving alternative even after HW availability especially for large scale deployment. The TLM VP performance is compared to the HW Sabre-Lite physical board as well for a real-time performance measurement. Similarly, the wall clock execution times of SPLASH-2 benchmarks at one and four parallel threads and CoreMark benchmark from one to eight parallel workloads are measured on top of the Sabre-Lite HW board. As depicted in Fig. 5, the average TLM VP performance, as measured by SPLASH-2 benchmarks, at four parallel threads is 12.14% and at a single thread is 14.71% of the HW board speed. Also, the TLM VP performance, as measured by CoreMark benchmark, is 17.74% of the HW board speed at four parallel workloads and is 18.1% at a single workload as shown in Fig. 6. Taking geometric mean of both benchmarks produces 14.67% (~15%) of the HW board speed at four parallel threads and 16.32% at a single thread. Moreover, relative score ratio as evaluated by CoreMark benchmark for QEMU: TLM VP: HW board is 1: 6.6: 20.7 respectively as listed in Table 1.

Table 1. Experimental environment (Upper), EEMBC CoreMark benchmark reporting (Middle) and SPLASH-2 benchmark settings (Lower)

Experimental Environment	
Guest platform	Sabre-Lite IMX6Q board, 256KB RAM, 3GB DDR memory and Cortex-A9 x4 ARMv7a processor
Guest OS	Linux 3.2.84
Emulator	QEMU v2.8.0 release
TLM VP configuration	LT, async_stable mode time quantum = $2x10^7$ clock cycles
Host machine	Intel Xeon E7-2850 eight-core processor @ 2 GHz and 8 GB RAM
Host OS	x86_64 RHEL 6.2 (Linux 2.6.32-220)
EEMBC CoreMark (v1.0) Benchmark Reporting	
Reporting options	GCC4.8.3/ -O3 -Ofast -mfpu=neon -mcpu=cortex-a9 -lrt -lpthread/ Heap / N:PThreads (N: # parallel workloads)
# Iterations	10^5 (command line options)
scores	QEMU=575.9, TLM VP=3815.08, HW board=11933.53 @ N=4
SPLASH-2 Benchmark Settings	
cholesky	-C32768 with input tk29_cholesky.O
fft	-m22 -n1024 -l5
lu/-non	-n1024 -b16
radix	-n16777216 -r1024 -m33554432
Other	Default

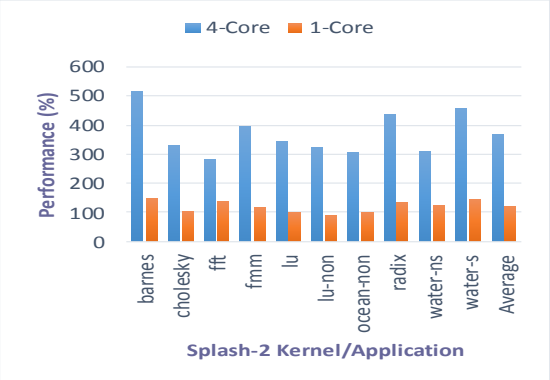

Figure 3. TLM VP performance by execution time of SPLASH-2 benchmarks. Baseline QEMU is treated as 100% basis.

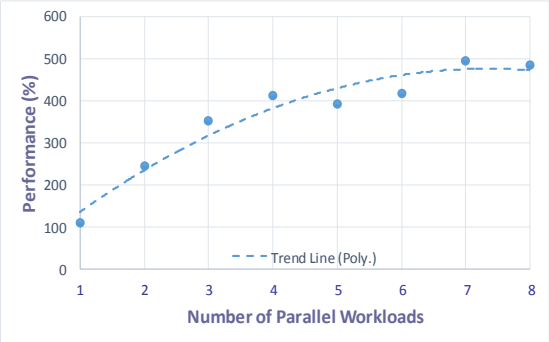

Figure 4. TLM VP performance by execution time of EEMBC CoreMark benchmark. Baseline QEMU is treated as 100% basis.

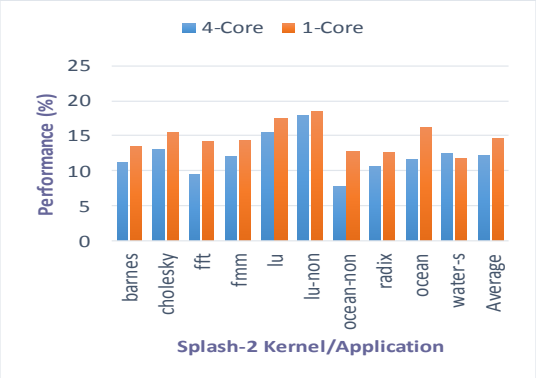

Figure 5. TLM VP performance by execution time of SPLASH-2 benchmarks. Sabre-Lite HW board is treated as 100% basis.

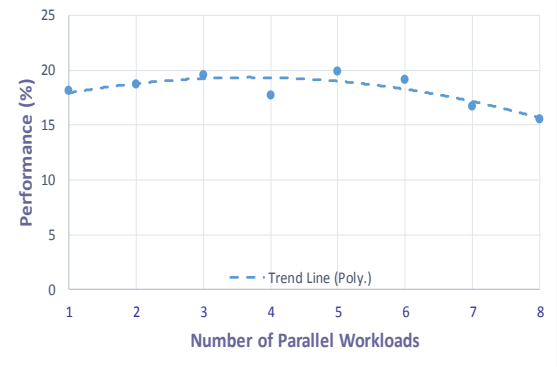

Figure 6. TLM VP performance by execution time of EEMBC CoreMark benchmark. Sabre-Lite HW board is treated as 100% basis.

5. Conclusions

High speed and full-system modeling capabilities of the developed TLM VP can provide versatile simulation environment that can be utilized not only in design space exploration, but also in SW continuous integration. As proposed in this paper, an efficient multi-core ISS modeling methodology using combined TLM and DBT features can achieve fast, yet accurate, simulation speed that can reach up to 3.9X speedup over baseline QEMU which corresponds to 15% of real HW board speed. This ISS is instantiated along with fully-compliant group of SystemC TLM-2.0 models through which full-system modeling can be achieved.

6. References

[1] Leupers, R.; Schirrmeister, F.; Martin, G.; Kogel, T.; Plyaskin, R.; Herkersdorf, A.; Vaupel, M., "Virtual platforms: Breaking new grounds," *The Proceedings of The Design, Automation & Test in Europe Conference*, pp. 685-690, March 2012.

[2] M. Safar, M. A. El-Moursy and A. Salem, "Ultra-Fast DMAC TLM Model for High Speed Virtual Platform Simulation," *The proceedings of International Workshop on Microprocessor Test and Verification*, pp. 39-44, December 2013.

[3] OSCI, TLM-2.0 Language Reference Manual. http://www.systemc.org.

[4] Mentor Graphics® Vista Architect™ Tool. https://www.mentor.com/esl/vista/flow

[5] F. Ghenassia, "Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems, " *Springer*, 2005.

[6] F. Bellard. "QEMU, a fast and portable dynamic translator," *USENIX Annual Technical Conference*, pp. 41–46, June 2005.

[7] Z. Wang, R. Liu, Y. Chen, X. Wu, H. Chen, W. Zhang, and B. Zang. "COREMU: a scalable and portable parallel full-system emulator," *The Proceedings of Principles and Practice of Parallel Programming*, pp. 213-222, February 2011.

[8] J.-H. Ding, Y.-C. Chung, P.-C. Chang, and W.-C. Hsu. "PQEMU: A parallel system emulator based on QEMU," *International QEMU Users Forum*, pp. 35-38, March 2011.

[9] Hong, Ding-Yong, et al. "HQEMU: a multi-threaded and retargetable dynamic binary translator on multicores." *The Proceedings of the International Symposium on Code Generation and Optimization*, pp. 104-113, April 2012.

[10] GreenSocs, U.K., "Greensocket Website," 2009. http://www.greensocs.com/projects/QEMUSystemC

[11] QEMU Open Source Processor Emulator. http://www.qemu-project.org/

[12] S. C. Woo, M. Ohara, E.Torrie, J.P. Singh and A. Gupta, "The SPLASH-2 Characterization and Methodological Considerations," *The Proceedings of International Symposium on Computer Architecture*, pp. 24-36, June 1995.

[13] ARM, Cortex™-A9 MPCore® Technical Reference Manual, Revision: r4p1.

[14] ARM, AMBA® Level 2 Cache Controller Technical Reference Manual, L2C-310 Revision: r3p1.

[15] J. H. Weinstock, C. Schumacher, R. Leupers, G. Ascheid, and L. Tosoratto. "Time-decoupled parallel SystemC simulation," *In the Proceedings of Design, Automation and Test in Europe Conf.*, March 2014.

[16] IEEE Standard for Information Technology, "Portable Operating System Interface (POSIX®) Base Specifications, Issue 7," *IEEE Std 1003.1-2008 (Revision of IEEE Std 1003.1-2004)*, pp. 1-3826, December 2008.

[17] NXP, i.MX 6Dual/6Quad Applications Processor Reference Manual, IMX6DQRM, Rev. 0, 11/2012.

[18] EEMBC. CoreMark®: An EEMBC Benchmark. http://www.eembc.org/coremark/index.php