A cycle-count-accurate simulation platform with enhanced design exploration capability

Oana Boncalo[1], Alin Dobre[2], Alexandru Amaricai[1], Andrei Tanase[1],
[1] University Politehnica of Timisoara
[2] Movidius Ltd Timisoara, Romania
oana.boncalo@cs.upt.ro alin.dobre@movidius.com

ABSTRACT
This paper presents a simulation platform for architecture exploration of bus based heterogeneous multi-processor system-on chips (MPSoC) – *moviSim*. The tradeoff between accurate simulation results and simulation time has been obtained by the cycle-count-accurate approach. Its main attributes are: flexibility, integration with the targeted tool chain and increased tracing and analysis capability. The wide range of implemented metrics (program execution time, executed instructions, stalled cycles, bus logging, register and memory port detection, power consumption, function, data and code line profiling, cache metrics (miss/hit ratio, etc), number of memory/subsystem reads/writes performed by a master) allow enhanced architectural exploration capability for complex MPSoC on which large software applications are running. Due to easy integration with debugging tools, the source code targeting the hardware platform can be easily verified and analyzed with the proposed simulation platform.

Categories and Subject Descriptors
I.6.5 [**Simulation and Modeling**]: Model Development; I.6.7 [**Simulation and Modeling**]: Simulation Support Systems.

General Terms
Performance, Design, Simulation.

Keywords
Multi-core cycle-count-accurate simulator, Performance analysis, Architecture exploration.

1. INTRODUCTION
To remain competitive, SoC designers must keep pace with the increasing customers' demand, for a variety of functions and support for multiple standards, within an overwhelming choice of architectural solutions. All of these are complemented by increased pressure to lower time-to-market and reduce costs. An important support in this direction is given by tools, and in particular by system-on-a-chip (SoC) simulators. Four important features are required by such a tool: *flexibility* (i.e. ease of configuring the desired architecture from discrete elements and to facilitate component reuse by including off-the-shelf-components), *accuracy* (i.e. level of abstraction for component modeling), *tracing capability* (for software (SW) and hardware (HW) architecture analysis), and *simulation time*. Two in particular are critical for the produced results: simulation time and accuracy. However, maintaining a high level of accuracy by employing register transfer level (RTL) modeling would result in higher simulation time due to difficult design, slow verification, and poor scalability. However, lowering this constraint would result in loss in terms of accuracy. Cycle accurate models eliminate unnecessary HW component description while trying to preserve the timing accuracy.

A trade-off between simulation time and accuracy - cycle-count-accurate (CCA) simulation, has gained attention in recent work [1]-[3], [5]-[7]. It focuses on eliminating timing and internal component modeling details that would hinder the simulation, but at the same time it targets a correct component behavior with respect to its interface (i.e. the transaction timings are preserved). This approach is employed in our work as well. For the first feature – flexibility – it is worth emphasizing its importance in the context of hybrid multi-core HW platform solutions. At the heart of this requirement lies the current trend in SoC design - component/design re-use. Furthermore, as highlighted in [2], different usage scenarios for the simulation platform would require mixed abstraction-levels. Another aspect refers to the integration of existing IP core descriptions by embedding them within a wrapper. This would create premises to enrich the components pool with significantly reduced development and validation cost/effort. The validation of the correct behavior of the simulation environment requires significant effort, and is critical for industry products.

Work presented in this paper, addresses the aforementioned critical features in a multi-core SoC simulator designed for architecture exploration and software development support – *moviSim*. This simulator has been designed to target embedded architectures, which exhibit a larger diversity with respect to general-purpose computer systems. Its intended use is to support the process of architecture exploration for the development of the next generation Movidius MpSoC HW platform.

This paper is organized as follows: Section 2 includes a brief review of related work, Section 3 focuses on the proposed simulation platform, in Section 4 the trace capability to aid architecture exploration and software development is described, followed by a case study – the H.264 decoding application and conclusions.

Simutools 2012, March 19-23, Desenzano del Garda, Italy

DOI 10.4108/icst.simutools.2012.247809

Simutools 2012, March 19-23, Desenzano del Garda, Italy

DOI 10.4108/icst.simutools.2012.247809

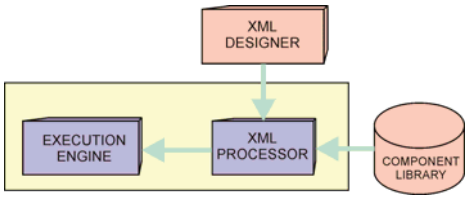

Figure1. moviSim architecture overview

2. RELATED WORK

The highest simulation result confidence is given by RTL modeling with rigorous component detailing, which requires significant simulation time. Thus, HW/SW co-simulation may become cumbersome [2]. An alternative approach that tries to address this issue is presented in very recent work from [1][3][6], mainly a new concept – *cycle-count-accurate (CCA)* model. It promises results that are close to traditional RTL SystemC, with an improvement of up to 100 times in simulation time [2] that is suitable for architecture exploration. The proposed simulator can support different abstraction levels, but it has been primarily developed to target cycle-count-accurate models for the correctness of the output results while lowering as much as possible the simulation overhead.

Regarding bus and memory modeling several approaches have been developed, such as the CCA-TL2 modeling for buses used in [8] and the CCA memory models proposed in [5]. Our approach is similar to the ones described above. There are two main differences. First, the proposed simulator does not require existing RTL descriptions, as it uses configurable components, with the possibility of integrating/developing new ones. Second, it relies on XML based component configuration (instead of automatic model generation) for tuning different system parameters.

Regarding multi-processor systems simulation, several approaches have been proposed, such as the ones presented in [12]-[15]. Although they have good simulation speeds, their modeling capability is rather limited, due to the following reasons:

1. Simulation is restricted to one processor core; furthermore, they do not provide means for including other types;
2. Limited bus modeling capability; hence no accurate means to analyze bus traffic is provided;
3. Lack of intra-processor synchronization and communication mechanisms implementations;
4. Except for [12], no methods for integration of hardware interfaces and accelerators simulation models is offered;

The proposed simulator incorporates these features.

3. CYCLE-COUNT-ACCURATE MODELING

3.1 Simulation Platform

The simulator core is a dynamic simulation environment, designed to facilitate the integration of different hardware modules, in order to aggregate discrete system components into a complete platform. They can be used in conjunction with user applications or benchmarks to evaluate platform performance. The simulation engine relies on the following components:

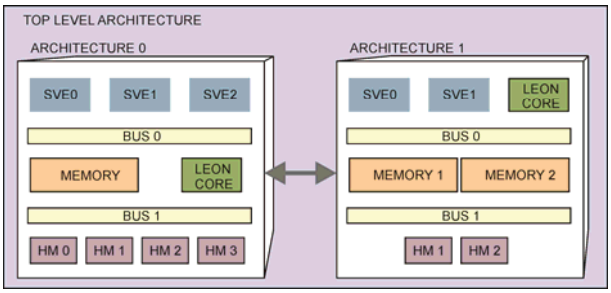

Figure 2. Nested architecture design

XML processor (XMP), Debugger Interface module, Simulation Runtime Engine Module (SREM) (see Fig.1). The hardware architecture is configured by means of an XML file. Several XML elements have been defined:

- ARCHITECTURE: is the top level, mandatory element and all the other elements are embedded hierarchically within one <architecture> tag.
- BUS: used to define a generic bus component.
- BRIDGE: interconnect element for communication between buses.
- MODULE: simulator contains several sub-classes of elements: PROCESSOR, which is used for user defined processor cores, specified via DLL Libraries; MODULE, which can be used to select several sub-types such as user defined peripheral modules, specified via DLL Libraries and specific elements like: SHAVE, CACHE, ICB, JTAG, MEM, CMX_MEM, T.
- SIGNAL: is used to simulate connections between modules that do not go through the bus (e.g. an interrupt line, the reset input etc.). A signal can be connected to one or more destination modules, but it is mandatory that it has a unique source module.

XMP is responsible for parsing the XML file and extracting the specific tags for each type of system component, followed by initialization of system components and then the building of the MPSoC system. SREM was designed to trigger the execution of each system element during a simulation cycle. An important feature favoring design reuse is the possibility of having imbricate architectures as depicted in Fig.2. These architectures can communicate through EBI interfaces models. Furthermore, the simulator supports multi-clock domain. The only restriction is that all clock signals must be derived from the system clock.

3.2 Component Library

moviSim offers an extensive range of configurable modules which can be used to construct target systems for performance-portability evaluation. These are configurable hardware modules (HM) that are common in embedded architectures and will be reviewed briefly in the following subsections. For further details please refer to [16] where the modeling issues of HW components (e.g. simulator interface, XML configuration parameters) are presented.

3.2.1 Interconnect elements

Bus component. The model is inspired by AMBA/AXI bus protocol [10]. For flexibility we have designed a generic bus that

models communication using *sendRequest* and *sendResponse* function calls. The targeted features that ought to be configured in the XML bus description are: *bus width, arbitration models* (RANDOM, ROUND_ROBIN, PRIORITY*), burst transfer mode*, *pipelined transfer*, *independent RD/WR channels*, *profiling information* and *monitors* (i.e. bus traffic, bus load, power consumption). If the selected arbitration policy is PRIORITY, each master on the bus must have the "priority" tag configured for its master interface.

Bridge component. We have implemented a generic bridge that is capable of transferring transactions from one bus to another. The modeled bridge is unidirectional, so two such bridges are required to achieve bidirectional communication between two buses. The unidirectional link between the two buses is configured in the XML file by configuring the master interface (contains the description of a master connection to a specific bus) and the slave interface (contains the description of a slave connection to the specified bus). This component is also responsible for transaction resize (i.e. to split transactions or assemble when possible transactions) in order to adjust the connection of two busses of different sizes.

3.2.2 Processor modeling

Two processing elements are offered by moviSim: Sparc V8, 32 bit processor with floating point unit and the Movidius Core processor. Special attention has been given to modeling the embedded DSP core. Movidius DSP core (SHAVE - Streaming Hybrid Architecture Vector Engine) – a 128 bit VLIW low power DSP with 8 parallel execution units (including an integer, scalar and vector SIMD floating point units – for both 16 and 32 bits floating point arithmetic). The entire register file with the corresponding RD/WR ports has been modeled. Support for port clashes detection exists in order to aid the software development process and to give further insight into core behavior. The pipeline stages have been implemented providing accurate delay accounting for data and instruction fetch (e.g. branch instruction) penalties. For the DSP cores power estimation has calculated according to equation (1), (2).

$$P_{total} = P_{leakeage} + P_{clk\_tree} + P_{dynamic} \quad (1)$$

$$P_{leakeage} = Gate\_count * P_{leakeage\_per\_gate} \quad (2)$$

$P_{dynamic}$ is calculated every cycle when the processor in not stalled based on: gate count, switching activity, and clock frequency. Hence, a runtime estimate of the power consumption for the application being run can be provided. Furthermore, much effort has been invested in testing the results correctness and accuracy. Validation has been performed against the RTL code, targeting 100% match for register contents, signals, memory contents, register ports.

3.2.3 Memory elements

Three types of HMs have been designed:

- CMX: combinational memory made of several SRAM tiles, which can be reorganized according to users' needs. Several tiles make up a slice. The slice size can be configured through XML description. Support for port-clash detection has been implemented.
- Cache model: a generic cache with XML parameter configuration for: size, line size, set-associativity, hit latency, enable write through field, replace policy (LRU, MRU, LFU, SLRU, AR), enable cache bypass (cache acts like a bridge between the master and the slave interface). Selective bypass - only a portion of the address space is bypassed, is also supported.
- Memory: generic memory HM with XML configuration options for: size, read latency (in simulator cycles), write latency (simulator cycles), read ports, write ports, port clash policy (RD_BEFORE_WR, WR_BEFORE_RD), enable parity checking bit.

Our modeling approach is similar to the one described in [5], with one significant difference. A potential drawback to [5] is the fact that the development of RTL memory models requires significant effort and may be unavailable. Thus, in our approach we try to provide a generic timed model, which can be tuned from the XML description according to the usage scenario. The memory timing behavior needs to be correct with respect to the bus interface. A typical memory read (RD) request would pass through the following phases:

- RD request is received by the HM on the slave interface and added in a request queue for servicing.
- Every cycle the request queue is inspected and if RD ports are sufficient (i.e. memory is not busy servicing another request) the memory access takes place.
- A counter is added to the data (i.e. read latency in simulation cycles).
- Each simulation cycle the counter is decremented (memory access delay is mimicked).
- When counter is 0, the data response is pushed back to the read responses queue; each cycle the slave interface sends the corresponding memory response (i.e. queue front).

In order to aid software developers, the HM allows the detection of access to uninitialized memory locations.

3.2.4 Miscellaneous

Several hardware accelerators and communication interfaces have been modeled and can be included in different simulation architectures, such as: UART, JTAG, I2C, EBI, H264 encoder and decoder accelerators, LCD interface, camera interface, DMA controller, etc. The behavior of these peripherals can be easily divided into computation and timing (with respect to the behavior observed from the bus interface). In the case of the LCD and camera interfaces, the outputs or inputs for these modules are represented by data files which contain the stream of frames which emulate the activity of the display or digital camera. Regarding the H264 encoder and decoder modules, these implement only the entropy CABAC/CAVLC encoding/decoding.

A bus traffic generator has also been implemented [16]. This module has no hardware counterpart. It can substitute the behavior (from the bus perspective) of hardware modules on a specific bus, replacing either a master (initiating transaction), either a slave (receiving transactions) or a master-slave HM.

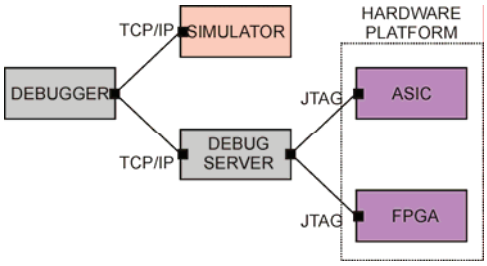

Figure 3. Simulator & debugger tool interaction

It relies on the fact that peripherals such as LCDs, Camera can easily be divided into computation and timing (with respect to the behavior observed from the bus interfaces). Hence, a traffic generator component using the HM bus logs and the HM *masterId* (needed for filtering) can be used as a substitute. It extracts the bus transactions for the corresponding HM and delivers them on the bus during the corresponding simulation cycle.

Synchronization points may be provided (i.e. wait for RD response from memory before issuing the next WR request and adjust the times stamps for the following transactions). It comes in two configurations: random and pattern mode. In random mode, it generates random transactions on the specific bus, and thus bus capabilities are analyzed. In pattern mode, it reduces simulation overhead by mimicking HM activity with respect to their bus behavior.

3.3 Interfacing External Processing Elements

Different HW/SW co-simulation campaigns have different goals and thus, different modeling accuracy requirements. Furthermore, reuse of already implemented simulation models is a plus in trying to reduce the time needed to prepare a simulation platform. Hence, special attention has been given to providing support for integrating existing models into our simulator: CA SystemC descriptions such as [10] or C-based CA descriptions such as [9]. In order to be used, the external model needs to conform to specific requirements. They are encapsulated in an external dynamic library that must implement a standard interface for the simulator to be able to initialize the library and dynamically create model objects. This interface consists of the following functions: *initLibrary* and *createObject*. Furthermore, the HMs are required to have the appropriate master and/or slave interfaces in order to be able to properly connect to a bus, and to extend an abstract interface that dictates the runtime methods.

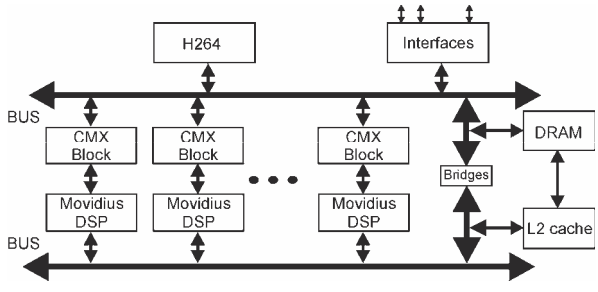

Figure 4. Simulation platform overview (the number of DSP units is variable)

For proof-of-concept we have designed appropriate wrappers for the ARM core described in [9], and the SystemC description of the PowerPC750 core from [10]. The interfacing effort has been increased due to the need of isolating the processor core. Our approach is similar to that of [13].

4. DEBUG & ARCHITECTURE EXPLORATION SUPPORT

4.1 Tool chain Integration & Validation

moviSim is a configurable simulator, designed for heterogeneous multiprocessor systems. Its purpose is to simulate architectures, allowing them to execute real software applications rather than execution traces. The features available in this simulator are: *Loading and execution of code* for the processors in the system, *Debugging facilities* through a client-server interface, which connects the simulator to a debugger, *Profiling facilities* (bus traffic analysis, memory access analysis, execution time/ /waiting time analysis, cache activity analysis), *data-logging* for later inspection.

It can run in two modes: as an independent application, or as a separate application communicating with the moviDebug debugger in client-server mode using a TCP/IP socket for commands and messages (see Fig.3). Debugging facilities are the same (e.g. debugger scripts) for both the simulator and the actual hardware. Basically the same code targets the simulator and the hardware and produces the same outputs.

Validation requires significant effort and plays a key role in the acceptance of a tool. Many teams in real world design report an effort of up to 90% for validation. Simulator validation has been obtained against an existing hardware platform (Movidius SABRE HW platform).

4.2 Trace Information for Design Exploration

Two types of profiling supported by the Movidius toolchain: simulator based (i.e. different counters are logged in internal structures and these may be enabled, reset, displayed or logged into files) and hardware based (i.e. based on HW performance counters which can be controlled using moviDebug commands). The first method has the following advantages: good resolution, a rich range of profiling information can be retrieved in a single run of the target software. The drawbacks for this approach are: simulation overhead and reduced accuracy with respect to the HW. HW support for profiling is non-intrusive, the code being executed without being interrupted by any breakpoint or instrumentation of the code. The disadvantages for the later are: multiple runs for multiple parameters profiling, lack of HW support for some of the targeted parameters, and may require code instrumentation by moviDebug when smaller block codes need to be profiled. This may lead to inappropriate profiling of the code that is time sensitive or contains some synchronizations.

The profiling information include: execution cycles, stall cycles, average ILP, power consumption, number of memory RDs/WRs, number of RDs/Wrs done by the Movidius DSP core DMA, number of RDs/WRs to each SHAVE register. For the power estimates, the hardware design team has offered measurements obtained from the physical device and from synthesis tools.

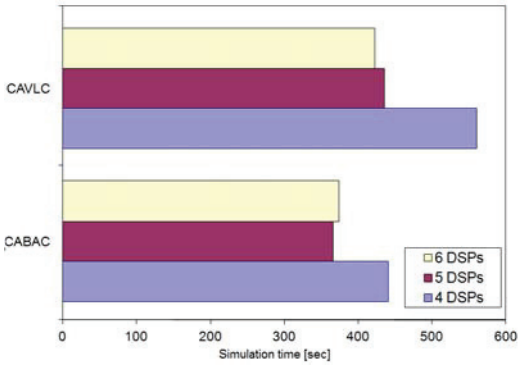

Figure 5. Simulation time for variable no. of cores [sec]

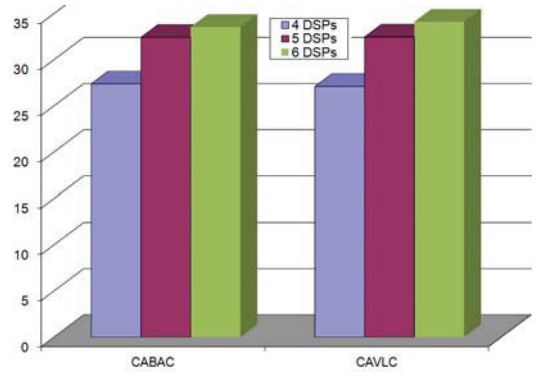

Figure 6. Overall power consumtion for DSP0 (in mW);

Furthermore, cache metrics (miss/hit ratio, etc), number of memory/subsystem reads/writes performed by a master is logged. Furthermore, it supports bus logging and Movidius disassembly core logging. Further information that can be extracted using the Movidius debugger for the SHAVE core with .mof (movidius object file) includes information for: functions, data and code labels (number of times each label has been reached/accessed), assembly lines (number of times one line has been executed).

5. A CASE STUDY

The architecture that we have used for testing is described in Fig.4. For the architecture exploration we vary the number of DSPs used, in order to evaluate the performance impact of the simulated HW platform when H264 decoding of 720p frames is performed. H264 decoding [20] requires 3 major steps: entropy decode, inverse transform and image reconstruct. The first step is the only one that cannot be parallelized. Hence, this step is suitable for HW acceleration. Two frames have been used for decoding: one using CABAC entropy encoding, while the other uses CAVLC. Thus, the decoding employs a HW/SW co-design solution mainly: a HW accelerator that performs entropy decoding, with the two following phases implemented in software. In order to further accelerate the decoding, Movidius DSPs are employed. We investigate the performance gain obtained by employing from 4 SHAVES to 6 SHAVES. Each Movidius core has a dedicated CMX slice for fast memory accesses needed during rendering. For both scenarios one Movidius DSP is used for programming the H264 entropy decoder accelerator and for adjusting the address for the CMX memory buffers. The rest of the DSP processing units are dedicated to performing rendering from the aforementioned memory units. The MPSoC simulated platform comprises of the following elements: Movidius DSPs, 8x128 kB CMX RAM memory directly linked to each Movidius DSP, hardware MUTEX for the CMX RAM memory, 128 kB L2 Cache with LRU replacement policy, 1x128-bit bus, 2x64-bit bus and 1x32-bit bus, bridges, 16 MB DRAM memory, interfaces (UART interface, JTAG port), H.264 entropy (encoder and decoder) acceleration unit. The simulations have been performed on an Intel I5 processor, 4GB DDRAM memory desktop computer with Windows 7 operating system. The results are reported in simulation cycles and are plotted in Figs.5-8.

6. CONCLUSIONS

This paper presents *moviSim* - a configurable simulator, designed for heterogeneous multiprocessor systems simulation. Its purpose is to simulate HW architectures, allowing them to execute real software applications rather than execution traces. The tradeoff between accurate simulation results and simulation time has been obtained by the cycle-count-accurate modeling. It supports architecture exploration and software development by rich tracing and analysis capability, logging of processing cores, and bus HM. Component reuse by wrapper design and encapsulation of an existing HM description and architecture reuse through nested architectures are supported. In addition to this, a wide variety of ready-to-use generic components that can be customized to the targeted platform are provided. We have designed and simulated a HW platform with variable number of Movidius DSP cores running H.264 decoding application on 720p frames. Future improvements consist of adding support for distributed computation on behalf of the simulation engine.

7. ACKNOWLEDGMENTS

This work has been partially supported by the EU Falx Daciae - SUIM 499/11844, POS CCE O2.1.1 research program.

8. REFERENCES

[1] Chen-Kang Lo, Li-Chun Chen, Meng-Huan Wu, and Ren-Song Tsay. Cycle-Count-Accurate Processor Modeling for Fast and Accurate System-Level Simulation. *in Proc. of Design, Automation & Test in Europe Conference*, 2011

[2] Zhe-Mao Hsu, Jen-Chieh Yeh, and I-Yao Chuang. An Accurate System Architecture Refinement Methodology with Mixed Abstraction-Level Virtual Platform. *in Proc. of Design, Automation & Test in Europe Conference,* 2010

[3] Gabor Madl, Sudeep Pasricha, Qiang Zhu, Luis Angel D. Bathen, Nikil Dutt, Formal Performance Evaluation of AMBA-based System-on-Chip Designs. *in Proc. of EMSOFT* , 2006

[4] OSCI. SystemC ver 2.1 (IEEE 1666 Standard), 2005.

[5] Y. Lo, M. Li, and R. Tsay, Cycle count accurate memory modeling in system level design, in *Proc. of the conf. on Hardware/Software Codesign and System Synthesis*, 2009

[6] S. Pasricha, N. Dutt, and M. Ben-Romdhane, Extending the Transaction Level Modeling Approach for Fast Communica-

tion Architectures Exploration, *in Proc. of the Design Automation Conf.*, pp. 113-118, 2004

[7] C. K. Lo and R. S. Tsay. Automatic Generation of Cycle Accurate and Cycle Count Accurate Transaction Level Bus Models from a Formal Model, in *Proc. of ASPDAC*, 2009

[8] AMBA AXI Specification www.arm.com/armtech/AXI

[9] M. Dales – SWARM Simulator – http://www.cl.cam.ac.uk/~mwd24/phd/swarm.html, 2003

[10] G. Mouchard – MicroLib PowerPC750 Simulator – http://www.lri.fr/~mouchard/PowerPC750/, 2002

[11] Hiren D. Patel and Sandeep K. Shukla. Model-Driven Validation of SystemC Designs, Hindawi Publishing CorporationEURASIP Journal on Embedded Systems, 2008, doi:10.1155/2008/519474

[12] Cong J., Gururaj K., Han G., Kaplan A., Naik M., Reinman G. MC-Sim: An Efficient Simulation Tool for MPSoC Design. *In Proceeding of the IEEE/SCM Conference on Computer-Aided Design (ICCAD)*, 364-371, 2008

[13] Makela J.M., Paakkuleinen J., Leppanen V. MVTSim – Software Simulator for Multicore on Chip Parallel Architectures. *In Proceedings International Conference on Computer Systems and Technologies – CompSysTech*, Article No. 15, 2009

[14] Renau J., Fraguela B., Tuck J., Liu W., Prvulovic M., Ceze L., Sarangi S., Sack P., Strauss K., Montesinos P. SESC Simulator , Available at: http://sesc.sourceforge.net, 2005

[15] International Telecommunication Union . Advanced Video Coding for Generic Audiovisual Services. - H264 Standard , 2005

[16] A. Amaricai, A. Dobre, O. Boncalo, A. Tanase, C. Valuch. Models and Implementations of Hardware Interface Modules in a Multi-Processor System-on-Chip Simulator. *in Proc. of the conf. 6th IEEE SACI,* 2011

Stall category	Cycles	
LSU1 Fifo Access Stall		1464544 (80.82%)
LSU0 CMX Access Stall		153681 (8.48%)
LSU1 CMX Access Stall		91394 (5.04%)
LSU1 Alignment Stall		53146 (2.93%)
LSU0 Alignment Stall		48926 (2.70%)
LSU 0 - Stall due to late read return		403 (0.02%)
LSU 1 - Stall due to late read return		73 (0.00%)
SHAVE0 - IDC: IP=0x1D000000 - Stall due to empty instruction fifo		5 (0.00%)

Figure 7. Stall sources and analysis for DSP0

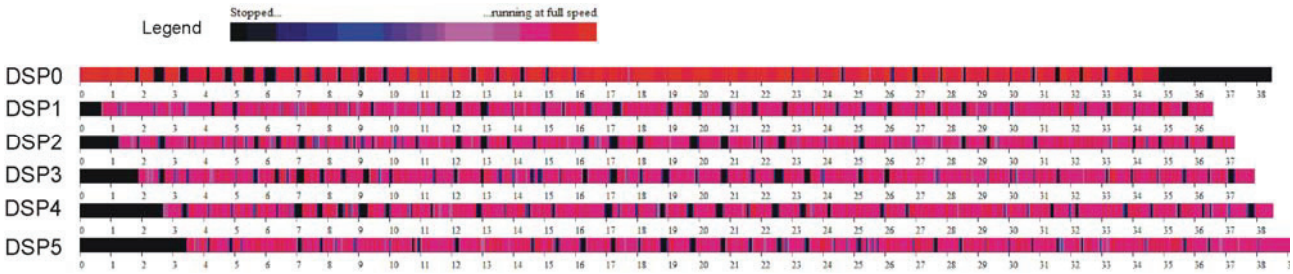

Figure 8. Activity profile of the 5 Movidius DSPs