Towards Virtual Prototyping of Synchronous Real-time Systems on NoC-based MPSoCs

Razi Seyyedi[*], MT Mohammadat[†], Maher Fakih[*], Kim Gruettner[*], Johnny Oeberg[†], and Duncan Graham[‡]
[*]OFFIS - Institute for Information Technology, Oldenburg, Germany
Email: razi.seyyedi@offis.de
[†]KTH Royal Institute of Technology, Sweden
[‡]Imperas, United Kingdom

Abstract—**NoC-based designs provide a scalable and flexible communication solution for the rising number of processing cores on a single chip. To master the complexity of the software design in such a NoC-based multi-core architecture, advanced incremental integration testing solutions are required. In this presents a virtual platform based software testing and debugging approach for a synchronous application model on a 2x2 NoC-based MPSoC. We propose a development approach and a test environment that exploits the time approximation within Imperas OVP instruction accurate simulator and a functional model of the Nostrum NoC, for both software instructions and hardware clock cycles at larger time stamps called Quantums that does not sacrifice functional correctness. The functional testing environment runs the target software without running it on the real hardware platform. With the help of Nostrum NoC we can support a synchronous system execution that is reasonably fast and precise with respect to a global synchronization signal, called HeartBeat. As work in progress, this work also discusses several possible timing refinement and their possible implication on the simulation semantics and performance and how it is tackled in future work.**

I. Motivation and Introduction

As chips are becoming more and more densely populated with cores/processors comprising hundreds of heterogeneous processing elements, a scalable and flexible communication structure becomes more demanding. A traditional bus-based communication does not scale well with the rising number of processing elements in multi-processor systems-on-chip (MPSoCs). Networks-on-Chips (NoCs) have been proposed as an efficient and scalable solution [1] due to their efficient and flexible connectivity. Designing and programming a NoC-based MPSoCs is a complex activity. Even harder is to debug such systems due to their concurrent many-core nature which makes observability and functional testing very limited in physical hardware platforms. Therefore, to enable an early functional testing of parallel real-time applications running on MPSoCs, the need for correctly modeling and simulating the system functional and timing behavior before deploying the software on a hardware becomes vital.

A desirable simulator would simulate the system relevant functionalities including the inter-processor communications reasonably fast while being precise with respect to timing. In this work, we aim to setup a virtual-platform to test the functionality and timing of synchronous applications running on NoC-based MPSoCs. In Synchronous systems, a global periodic clock governs execution and computation time where communication time are provided in terms of ticks of the global clock and the outputs are not visible to other computations until the next tick. Synchronous Model of Computation (MoC) simplifies the description, analysis, and testability for real-time applications which makes it appropriate for embedded multi/many core design, coping with their rising complexity.

For the experimental setup of this work, we will use the instruction-accurate Imperas OVP (Open Virtual Platform) as the simulation platform [2] and Nostrum NoC [3] for the interconnection. We will show how to model the NoC in the virtual-platform and enable synchronousness in our model with the help of so called HeartBeat concept in Nostrum NoC.

The remainder of this paper is structured as follows. Section II provides the background and reviews related works. Section III discusses the case study. Sections IV presents our approach and its implementation. Section V concludes our work with the future plan.

II. Previous and Related Work

A. Nostrum Synchronous NoC

In this work, we use the Nostrum NoC architecture and design methodology [3], where each Processing Element (PE) is connected through a Resource Network Interface (RNI) to a switch of the network. Both 2D and 3D NoC architectures are supported. Nostrum provides both best effort (BE) and guaranteed bandwidth (GB) packet delivery, making the platform suitable for performance and real-time oriented embedded systems [4].

Describing a system using a synchronous model simplifies the scheduling problem and ensures determinism by making the system easier to understand and easier to verify. Synchronous MoC divides time into abstract discrete notion of time, such as integers. Computation and communication is constrained by the beginning and end of each time slot. In a synchronous Model of Computation (MoC) [5], a global pulse triggers computation for all cores. A system is modeled using the synchronous MoC if it uses only synchronous signals and processes. Consequently, they are oriented to model real-time applications.

The HeartBeat model in Nostrum NoC [6], is an intermediate platform model bridging the abstraction gap between the synchronous MoC and a NoC-based MPSoC platform. It provides the synchronous communication system and enables the MPSoC to expose the same semantics of the synchronous MoC. It has been conceived targeting a generic NoC-based MPSoC, so it can be applied on different NoC-based MPSoCs independent of the type of PE and routing technique (deflective, wormhole, etc.).

A HeartBeat (HB) is a global periodic event which is made visible simultaneously to all PEs of the NoC-based MPSoC. Similar to a clock in synchronous digital design, a HB can be represented through ticks (HB ticks) repeated periodically with period t_{HB} (HB period). Every single received HB tick triggers a compute cycle on the PE. Processes scheduled on the PEs start executing once every HB tick and they communicate with processes on other PEs through the NoC. When a PE sends data to another PE in a precise HB tick, the receiving PE sees the value in the next HB tick. The time required for the job execution (t_e) and the time it takes for communication (t_c) are two factors that in NoC should be taken into account. When the condition $t_e + t_c \leq t_{HB}$ is satisfied, the behavior of the application executed on a NoC-based MPSoC is equivalent to the semantics of a sequential system in the synchronous MoC. The HeartBeat model allows to synthesize high level models of simply periodic and pipelined signal processing applications based on synchronous semantics onto a NoC-based MPSoC platform. The HeartBeat is implemented as a programmable timer (or clock divider) and provided to the PEs through the RNIs in the generated NoC. The node generating the HB tick is called *Pacemaker*. The tick of the timer sets a register in the RNIs. This enables the connected PEs to interpret the timer tick as an HB tick. For each HB tick the PE computes the input values (through the synchronous processes mapped on that particular PE), and sends the results through the NoC to the destination. The received value is stored in the RNI of the destination PE, and it is made available on the following HB tick. The HB tick (periodic event) is visible and shared between all PEs of the MPSoC.

B. Open Virtual Platform Simulation

The OVP processor models are binary equivalent to the real processors [2]. They are able to execute the real binary compiled for the actual target CPU. In other words, the executable running is totally unaware that is running on a simulation platform, so the same executable can be launched, as it is, on the target platform without any changes.

A very basic timing model is already included in OVP. The processor model counts the number of executed instructions. Together with the annotated nominal processor speed in MIPS (Million Instructions per Second), the number of instructions executed in a certain period in the simulation can be calculated. The execution time is obtained by dividing the number of instructions executed in all simulation steps by the nominal processor speed. This level of accuracy is available with instruction accurate (IA) simulators. This is enough for many aspects in software development and test. However, there are other situations where this is not sufficient, for instance for the real-time analysis of time critical control systems.

The simulation step (Δ) in OVP is called Quantum. A Quantum, in a multiprocessor simulation, is the number of instructions each processor executes in each turn. The order in which the processors are executed is not deterministic, but it works as it is shown in Figure 1. The Quantum value is set by the user based on the trade off between accuracy and simulation time. For example, simulator executes 10,000 instructions in the first Δ of P_0 (blue arrows). Then it moves to P_1 and executes Quantum number of instructions, then P_2, and $P3_0$. After a Quantum for all processors is passed, the simulation time is advanced. Therefore, the Quantum value defines the timing granularity of the simulation. Increasing the number of

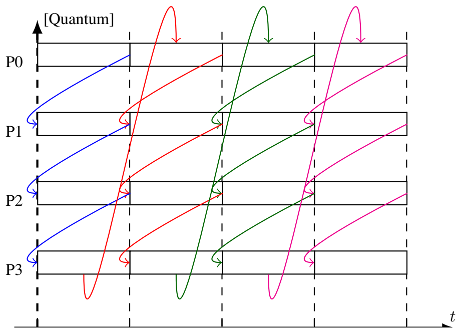

Fig. 1: Quantum (Δ) concept in OVP
Arrows show execution sequences

instructions running on one process in one turn (Quantum), will reduce the time spent by the simulator for context switching, so improves simulation speed. However, it will also increase the chance that interactions between processes will be inaccurate with respect to the timing, especially if they communicate through shared memory. As shown in Figure 1, after a Quantum is passed and the time is advanced, simulator starts the next round (red arrows). By executing the next Quantum for each process after another, it moves to the next turns (green arrows, magenta, etc.).

In this work, we decided to choose OVP simulator for the following reasons [7]:

1) OVP is commercially proven, provides a rich set of functional processor models and is one of the fastest simulators in its class;
2) As for all dynamic binary translation approaches, different target compilers can be used with all possible optimizations
3) The target processor debugging tools can be used directly.

C. Related Work

There are other works that try to improve NoC modeling in complex systems. In both [8] and [9], an OVP extension in SystemC is presented to simulate NoC-based Multiprocessor Systems-on-Chip (MPSoCs). In these two works, however, NoC is modeled outside of OVP using SystemC and the OVP simulator is just used to simulate the processor. One advantage of our approach is, when integrating the NoC inside the simulation platform, namely OVP, the simulation speed increases since no need anymore for synchronization between OVP and the SystemC module. Also debugging in an integrated environment is easier and less error-prone compared to the one connecting two different technologies.

There are other simulation platforms for NoC-based MPSoC, each targeting different aspect of the simulation. In all [10], [11], [12], [13], [14], [15], [16], and [17] a different technology than OVP is used. The main difference to our work is that, by using OVP as our simulation technology, we can test and debug the target code as it will be deployed i.e. the real binary that will be run on the system with the same memory layout as the real system.

[18] evaluates software energy cost at early stage. This work is also based on OVP with instruction-driven energy analysis approach. With the help of Cadence Incisive tool, it performs

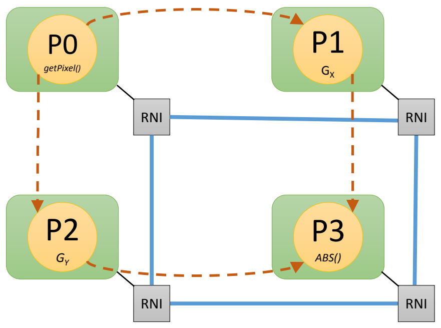

Fig. 2: Processing elements connected via NoC

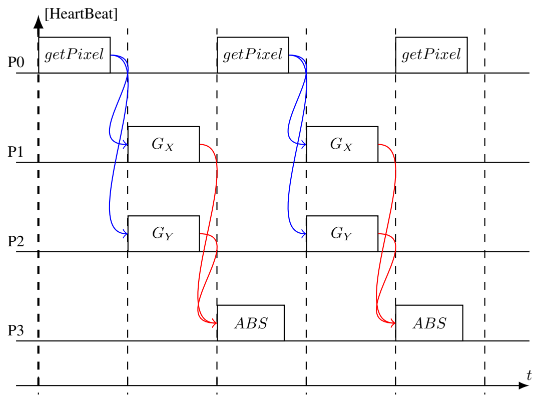

Fig. 3: Task scheduling per processing node (P_X)

the gate-level simulation in order to obtain the execution time. However, they do not consider NoC and Synchronous MoC in this work.

The main contribution in our work is to introduce synchronous MoC into the virtual platform and make use of extra-functional properties to boost simulation speed and improve accuracy. It is suitable for testing of real-time and time-triggered architecture with respect to time and debugging many cores.

III. Case Study

The following case-study was implemented and will be used as a motivational example for the proposed approach in this paper and for future experiments.

The Sobel Filter is used in image processing, particularly for edge detection. Signal processing applications are nowadays also used in safety-critical applications for e.g. to detect pedestrians crossing the street or to recognize traffic signs. We use a more complex Sobel filter with a 9×9 mask as it is used in [19]. In this case-study, Sobel filter consists of two 9×9 kernels which are convolved with the original image to calculate approximations of the derivatives. One kernel for the horizontal changes (G_X), and the other one for vertical (G_Y).

For the hardware implementation we use a Xilinx ZC702 evaluation board. Because of Nostrum NoC constraints, the board should operates at 50MHz. The platform consists of a NoC size of 2×2 with 4 bi-directional virtual channels at each node. The topology as it is shown in Figure 2, is a $2D$ mesh with 54-bit flit size (32bit data, 22 bit header) and packet size of 128 32-bit words. The routing algorithms are wormhole, deflective, distributed yx.

The Sobel filter is a good fit for a 2×2 NoC. Sobel filter has four tasks, each of which assigned to one of the 2×2 NoC node. Four tasks are as follows, as depicted in Figure 2: *getPixel* function that selects the part of image to apply the algorithm assigned to P_0. Two tasks are G_X and G_Y which move on the X and Y axis and are assigned to P_1 and P_2 respectively. The last actor, *ABS* function that calculates the gradient magnitude is assigned to P_3. The task scheduling is shown in Figure 3.

P_0 is implemented on ARM core and the rest is implemented on the FPGA fabric. Utilizing both processing and programmable logic part of Xilinx Zynq platform helps to better prove the functionality of the proposed approach by making a generic interface for processing elements. Results obtained from hardware implementation is shown in Table I. It shows the needed time for each processing element to perform its tasks.

The optimum HeartBeat is calculated based on both communication and computation times. The most time consuming task ($getPixel$) determines the HB period. The HB should be set equal to WCET[1] of $getPixel$ when executed on P_0 plus the worst-case communication time (WCCT) of $getPixel$ sending the needed packet over the NoC. In this case it would be:

$$8055 + 2 \times (40 + ceil(128/4) \times 20(cycles)) = 8735cycles$$

Here the WCET is 8055 cycles. For WCCT, since P_0 sends two messages that have to physically be serialized, the WCCT would be twice as much as sending of one message. One packet carries 128 integers (32-bit integer). If $HeartBeat < WCET + WCCT$, the messages will not arrive on-time and the synchronization will not be correct. Then the implementation does not match the model.

IV. Proposed Approach

The main challenge towards realizing a framework for virtual-prototyping of NoC-based MPSoCs is to have a model which achieves a good trade-off between accuracy and simulation speed. Both Instruction Accurate (IA) and Cycle Accurate (CA) simulators can be used for functional verification. In the following we will take a look into a first approach how to enable a synchronous execution within OVP and how to model the Nostrum NoCs.

A. Modeling Nostrum NoC and integration to an OVP MPSoC Model

In OVP, the Nostrum NoC is implemented using the peripheral modeling technology and *PacketNet* interface. It is implemented as a functional model of the network interface. It provides the interface as seen by the software and supports the communication mechanisms used by the software. *PacketNet* is used for the abstraction of a packet based Network-on-Chip. It is a packet based mechanism for broadcasting data to several possible targets. A packet transaction is modeled as an instantaneous event. Network speed and latency is modeled in the sending devices. A *PacketNet* communicates by callbacks and shared memory.

The transmitting model creates a packet in its local memory then calls the transmit function. This causes a notification function to be called in each receiving model in turn, passing a

[1]In this case it is the worst-case measured execution time.

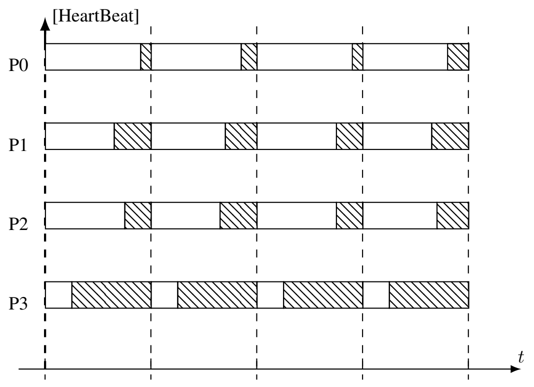

Fig. 4: Slack/Idle time in each HB period for different core

pointer and number of bytes in the packet. The peripheral model API can send and receive through the PacketNet interface. A PacketNet is bidirectional; a model can send and receive from the same PacketNet (though it does not have to). In our model, for the sake of simplicity we use one directional communication as shown in Figure 2. In our approach, the NoC is created as a virtual platform using a four processor sub-systems which use a peripheral model to create the network connection. Nevertheless to maintain performance in the virtual platform, we do not model the low level network protocols. We provide the HeartBeat call to the application software to operate as it would on the hardware while maintaining simulation performance.

B. Realizing a Heart-Beat Synchronous Execution in OVP Discrete Event Simulation

In a first approach, we propose to set the Quantum of OVP simulation to be equal to the HeartBeat in terms MIPS. This is reasonable, since in a synchronous application the synchronisation of the tasks is always done at the begining of the global clock tick (in this case at the begining of the HeartBeat). Consider for e.g. 4 cores connected via a NoC as depicted in Fig. 2 and the task mapping in Fig. 4. As seen these four cores are independent of each other and communicate (synchronize) only at specific time (e.g. Gx and getPixel at HB1), and that is why the Quantum can be set to the same number of instructions needed to simulate a HeartBeat.

Since the HeartBeat is computed according to the most time consuming task (including the communication) in the system (see case-study in Fig 3), it is gauranteed that the computation and communication are finished within the HB boundary. By considering the HB time as the time anchor, the functional result and the global time (w.r.p.t. number of cycles) is correct.

Yet with this assumption, the exact time indicating when the execution of a specific task on a specific core is finished within a HB cycle can not be traced within the simulation. In the future work, we will take a look how to improve that.

V. Future Work

For the timing model in OVP, we will use the quasi-cycle-accurate timing model introduce in [7]. This model improves the accuracy without involving the simulation slow-down of a really cycle accurate processor model. This is done by using information about the executed instructions on the processor model and the processor instruction set and its micro-architecture.

Now, in OVP we can detect the HeartBeat and when it is detected check if the processors/tasks have reached their idle state. With combination of quasi-cycle accurate timing model, we can semi-accurately calculate the execution and slack time in the virtual platform. Slack time are now visible to us by having a timing model during each HB slot as it is shown in Figure 4. Also, since we would like to target time-triggered architecture in safety-critical situations, we plan in the future work, to interface the Nostrum NoC with a time-triggered network interface.

Acknowledgment

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687902 (SAFE-POWER).

References

[1] Hemani, Ahmed, et al. *"Network on chip: An architecture for billion transistor era"* Proceeding of the IEEE NorChip Conference, 2000.
[2] Open Virtual Platforms (OVP) website http://www.ovpworld.org/
[3] Millberg, Mikael, et al. *"The Nostrum Backbone - a Communication Protocol Stack for Networks on Chip"* Proceedings of the 17th International Conference on VLSI Design (VLSID04), 2004.
[4] Millberg, Mikael, et al. *"Network on chip: An architecture for billion transistor era"* Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE04), 2004.
[5] Edward A. Lee, and Alberto Sangiovanni-Vincentelli *"A Framework for Comparing Models of Computation"* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 17, No. 12, 1998.
[6] Robino, Francesco, et al. *"The HeartBeat model: A platform abstraction enabling fast prototyping of real-time applications on NoC-based MPSoC on FPGA."* Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2013 8th International Workshop on. IEEE, 2013.
[7] Schreiner, Soeren, et al. *"A quasi-cycle accurate timing model for binary translation based instruction set simulators."* Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), 2016 International Conference on. IEEE, 2016.
[8] Wehner, Philipp, et al. *"MPSoCSim: An extended OVP simulator for modeling and evaluation of network-on-chip based heterogeneous MPSoCs."* Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015 International Conference on. IEEE, 2015.
[9] Real, Maria Méndez, et al. *"MPSoCSim extension: An OVP Simulator for the Evaluation of Cluster-based Multi and Many-core architectures."* Proc. of the 4th Workshop on Virtual Prototyping of Parallel and Embedded Systems (ViPES) as part of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XVI), Samos, Greece. 2016.
[10] Jain, Lavina, et al. *"NIRGAM: a simulator for NoC interconnect routing and application modeling."* Design, Automation and Test in Europe Conference. 2007.
[11] Fazzino, Fabrizio, et al. *"Noxim: Network-on-chip simulator."* URL: http://sourceforge. net/projects/noxim (2008).
[12] Jiang, Nan, et al. *"A detailed and flexible cycle-accurate network-on-chip simulator."* Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on. IEEE, 2013.
[13] Ben-Itzhak, Yaniv, et al. *"Hnocs: Modular open-source simulator for heterogeneous nocs."* Embedded Computer Systems (SAMOS), 2012 International Conference on. IEEE, 2012.

TABLE I: Analysis of computation time of a Sobel filter with a 9×9 mask on a 2×2 NoC

phase	**exec. time [cyc.]**		
	best	avg.	worst
`getP.` comp.	7875	7948,5	8055
`GX` comp.	4575	4575.0	4575
`GY` comp.	4575	4575,0	4575
`ABS` comp.	52	52.0	52

[14] Duenha, Liana, et al. *"Mpsocbench: A toolset for mpsoc system level evaluation."* Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014 International Conference on. IEEE, 2014.

[15] Cong, Jason, et al. *"MC-Sim: An efficient simulation tool for MPSoC designs."* Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design. IEEE Press, 2008.

[16] Renau, Jose, et al. *"SESC simulator."* (2005): 6.

[17] Benini, Luca, et al. *"Mparm: Exploring the multi-processor soc design space with systemc."* The Journal of VLSI Signal Processing 41.2 (2005): 169-182.

[18] Rosa, Felipe, et al. *"Fast energy evaluation of embedded applications for many-core systems."* Power and Timing Modeling, Optimization and Simulation (PATMOS), 2014 24th International Workshop on. IEEE, 2014.

[19] Schlaak, Christof, et al. *"Power and Execution Time Measurement Methodology for SDF Applications on FPGA-based MPSoCs."* arXiv preprint arXiv:1701.03709 (2017).