

Unified Modeling Abstraction for Fast Simulation and Emulation

Gummidipudi Krishnaiah*, Preeti Ranjan Panda*, Ashok Jagannathan†, Sreenivas Subramoney† and Anshul Kumar*

*Department of Computer Science and Engineering, Indian Institute of Technology Delhi

{krishna,panda,anshul}@cse.iitd.ac.in

†Intel Technology, India

{ashok.jagannathan,sreenivas.subramoney}@intel.com

With the increasing design size of SoC's and chip multi-processors, simulation models are not sufficiently fast to accommodate performance evaluation with realistic benchmarks. Emulating the performance models in programmable hardware (FPGA) is an attractive solution for speeding up performance analysis [1], [2], [9], [10], [11]. This poses the following requirements on the design of the performance model: (1) the simulation speed of the original HDL model needs to be high enough to allow fast functional verification and (2) the emulation speed of the design synthesized from this model should be acceptably high to support the simulation of large benchmarks.

We briefly review some characteristics of good simulation and emulation models using the SystemC [7] modeling infrastructure and a commercial behavioral synthesis system such as AutoPilot [8]. The requirements of fast simulation models are often different from those from which efficient hardware can be synthesized.

Some typical mechanisms to describe fast SystemC simulation models are: (1) minimise architectural detail – behavioral models (Figure 3) with algorithmic descriptions are 10x faster than those with architectural details (Figure 1); (2) minimise the number of processes to reduce context switch overhead [3]; and (3) maximise the use of native C++ data types [4].

When the objective is efficient synthesis into hardware, HDL models are optimised along different criteria: (1) A SystemC process is essentially sequential code, and typical behavioral synthesis algorithms in academia and industry have difficulties in automatic parallelisation of behavioral code into parallel hardware, which results in performance-inefficient synthesized hardware. Thus having more number of processes per design, i.e., explicit specification of parallelism by the programmer, is beneficial for generating parallel hardware and hence higher emulation speeds. (2) Since the latency of generated hardware from a SystemC process is unpredictable before synthesis as it depends on factors such as target platform, technology, tool dependent optimizations, etc., the synchronization between interacting processes needs to be handled explicitly by handshaking signals. Keeping synchronization to a minimum produces faster hardware. (3) Using appropriate data widths in the model (i.e., NOT using native C++ data types) results in more efficient hardware both in terms of area

and speed.

From the above we observe that the requirements for a good simulation model and that of an efficient synthesis/emulation model impose contradicting modeling constraints. Hence we need to identify an appropriate abstraction level at which the simulation model should be described, so that it complies with most of the above guidelines. Such a modeling abstraction where simulation and synthesis models have a common origin, if found efficient, can lead to significant productivity improvement in processor and SoC design flows, as it would eliminate the need to manually maintain consistency across the different models, and the associated verification challenges. Broadly there are two different levels of modeling abstraction that could be suitable for both simulation and synthesis specification, and we limit our present discussion to these: Architectural Level and Behavioral Level. Levels lower than the above, such as RTL, are unsuitable because the detailed clock cycle level accuracy results in very slow simulation. Levels higher than these, such as transaction level, are not sufficiently accurate.

The **Architectural model** is described by maintaining high-level structural details of the hardware through explicit parallel constructs. Consider an example of modeling an N -stage pipeline in SystemC as shown in Figure 1.

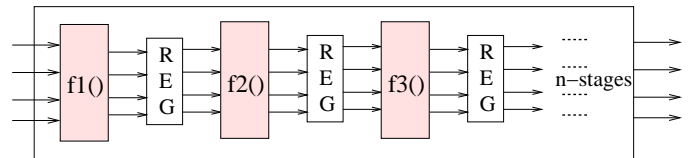


Fig. 1. Architecture Model of N -stage pipeline used for simulation. Simulation model does not require synchronization as there is no unpredictability in latency of each stage.

The design is explicitly partitioned by modeling each pipeline stage as a separate SystemC process. Each process is a functional description of the pipeline stage and the communication between them is only through ports. Synthesis from such a description generates a separate hardware module corresponding to each SystemC process, thus preserves pipelining in the design. Since the latencies of the generated hardware modules are dependent on factors such as target

platform, library, technology and also synthesis tool decisions, we cannot make assumptions regarding the delays of the pipeline stages. Hence explicit handshaking is required to maintain synchronization between any two communicating units. Since different pipeline stages have different latencies, each process waits after its completion until all other processes have also completed before it proceeds to next simulation cycle (each simulation cycle being realised as possibly different and variable number of cycles in the synthesized design). This is achieved by using a global synchronization (SYNC) module (similar to [2]), as shown in Figure 2.

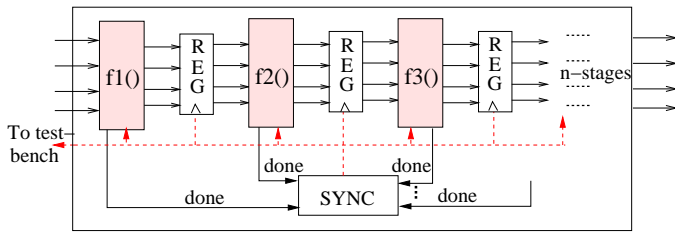


Fig. 2. Architecture Model of N-stage pipeline with global synchronization module. This model is used for emulation.

Each process informs the the SYNC module about its completion. When the SYNC module receives this notification from all units in the pipeline, it sends a signal to all processes to start the next simulation cycle. With the increasing number of communicating modules the overhead of synchronization becomes prominent in terms of simulation and emulation speeds. Results of our experiments plotted in Figure 5 show that the simulation time incurs a linear increase in the synchronization overhead. We also observed a 12% drop in emulation speed due to the synchronization overhead.

In a **Behavioral model**, the overall functionality of the hardware is described without any structural or architectural details. The timing characteristics of the hardware are annotated separately. For the above pipeline example, the behavioral model, as shown in Figure 3, is described by modeling its whole functionality using a single SystemC process and the timing is modeled separately using a delay unit (shift register) in the SystemC process. Thus the behavioral model exhibits the same timing and functional characteristics as the architectural model. Since a single hardware module is generated after synthesis, synchronization is only necessary between this component and the external interface to it as shown in Figure 4.

Thus behavioral models do not have synchronization overhead unlike architectural models, as shown from experimental results plotted in Figure 5.

We evaluated the architectural and behavioral models in terms of their simulation and emulation speeds for two designs: (1) the above pipeline design and (2) a more detailed and complex NoC router design.

The experimental results on the pipeline design are summarised below.

- We observed that the simulation speed of the architectural model is about 9 times slower when compared to behav-

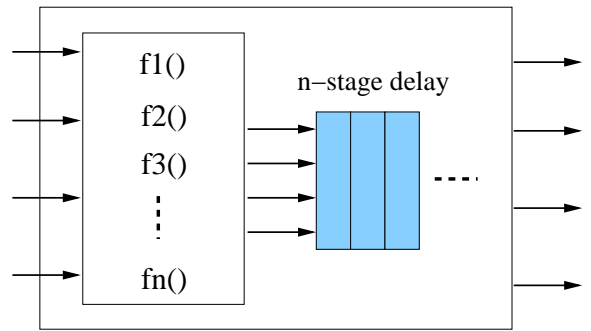


Fig. 3. Behavioral Model of N-stage pipeline used for simulation.

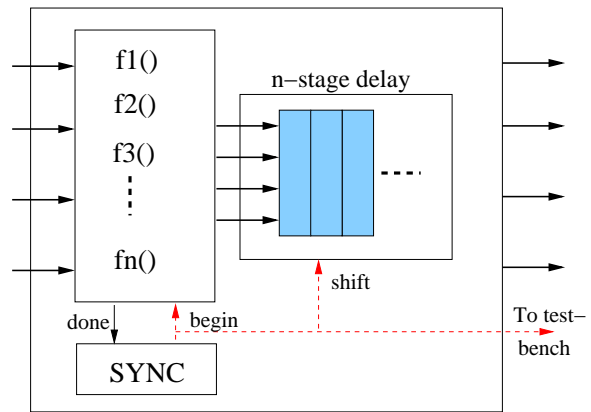


Fig. 4. Behavioral Model of N-stage pipeline with synchronization module. The synchronization module here is used for handshaking with testbench.

ioral model of the same hardware and this gap increases with increase in number of pipeline stages. This is due to the increasing overhead of process management by the simulation kernel.

- Synthesis of the architectural model resulted in parallel hardware which is faster than the one generated from the behavioral model. From our experiments, we observed that the emulation speed of architectural model is 5-23 times faster than that of the behavioral model. With the increasing number of stages the emulation speed of architectural model is not affected whereas for the behavioral model the speed decreases linearly with increasing number of stages primarily due to the synthesis tools inferring large and complex datapaths and FSMs in the generated circuits, which leads to sequentialisation of the processing. Thus the architectural model scales well with the pipeline stages until the SYNC modules becomes the bottleneck.
- A behavioral model is associated with lower development costs than an architectural model and is also more flexible and extendable. Consider a case where we wish to study the effect of varying the pipeline depth of a component on the rest of the system. An architectural model would require partitioning of the component into various pipeline depths and building models for the same, whereas in a behavioral model, we can mimic the effect of varying the

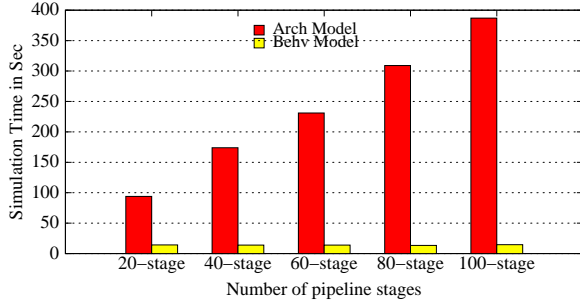


Fig. 5. Synchronization overhead in simulation time. The synchronization overhead increases at the rate of 3.66 for arch model and is negligible for behv model.

pipeline depth by simply varying the number of stages in the shift register (delay unit), and the synthesis tool can complete the details.

From the above observations, we see that the choice for an abstraction level that is best suited for both simulation and emulation is not obvious. An architectural model results in more efficient emulation hardware but is slow to simulate, difficult to develop, and also incurs higher synchronization overhead. A behavioral model has lower synchronization overhead, is faster to simulate, flexible, and easy to develop. However, the hardware synthesized from it could be sequential and thus result in lower emulation speeds. In the current work, we have focussed on techniques to improve the emulation speed of a synthesized behavioral model.

We discuss two effective techniques for enhancing the synthesized hardware from behavioral models and illustrate the effects of the same with a case study of a Network on Chip (NoC) [6], [5] whose router architecture is shown in Figure 6. The aim of this modeling effort is to measure the throughput and latency of the packets at flit (flow-control-unit) level accuracy. Since the simulation speed decreases with increasing size and complexity of the network, it is necessary to have an emulation model for fast design space exploration (of various router architectural parameters, routing algorithms, allocation policies etc).

We modeled a ring network (Figure 7) using the above router (Figure 6) module. We have used Xilinx Virtex II Pro XC2VP30 device for emulation. Since this device could accommodate upto four instances of the router module along with a transceiver for each to generate the traffic, we emulated a 4-stage ring network.

Since each router has three IO ports, each port having two virtual channels, we used a two dimensional loop for describing the behavior of each (independent) virtual channel. The variables (state variables, flags, counters, etc.) required for each virtual channel were declared as arrays. Such abstract behavioral models require minimum development effort and have very high simulation speeds. As shown in Figure 8, the behavioral model simulates 10 times faster than its corresponding architectural model. However, due to the sequential nature of the hardware description, the synthesis procedure

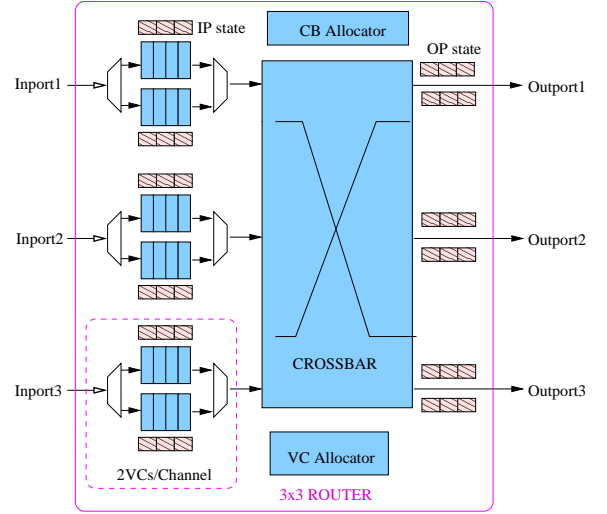


Fig. 6. 3-Port Router Architecture with two virtual channels per physical channel and credit based flow control policy. The router has a four-stage pipeline: (1) Routing (2) Virtual channel allocation (3) Cross-bar allocation and (4) Transmission. Worm-hole routing with best effort services is implemented.

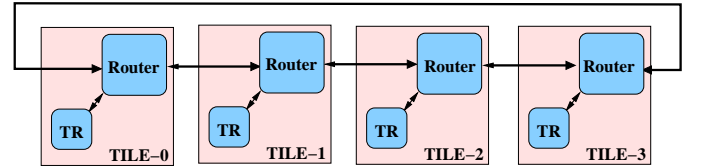


Fig. 7. 4-stage Ring Network

may generate inferior circuits. Further, due to extensive use of arrays (which are synthesized to one or more memory modules) the parallelism within the loops cannot be extracted due to resource constraints (limited ports in memories). Thus the emulation speed of such a model decreases with increasing complexity (number of I/O ports) of the design. From the experimental results in Figure 9, we observe that its emulation speed is 8 times slower compared to the architecture model.

- Automatic pipelining of the behavioral code during synthesis further improves emulation speeds of generated circuits. The advantage with automatic pipelining com-

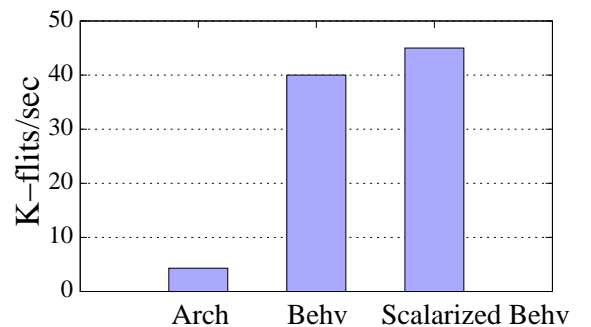


Fig. 8. Simulation speeds of the ring network with router modeled at different abstraction levels.

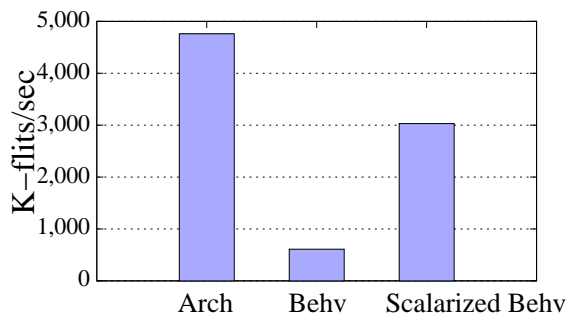


Fig. 9. Emulation speeds of the ring network with router modeled at different abstraction levels.

pared to manual/customized one is that, in the former case the stages are more balanced in terms of latency and thus the generated hardware can be clocked at higher frequencies. The custom pipelines on the other hand are generally based on functional partitioning (routing-vca-cba-transmission, in case of NoC router) rather than the latency of each stage, thus the emulation speed is equal to the slowest of all stages. There is a significant productivity increase when automating the pipelining because the process saves significant development effort.

- We observed that a significant amount of available parallelism has been restricted due to the use of a few arrays in the behavioral modeling and the consequent memory port constraints. All such arrays were identified and converted to scalar variables which resulted in their being synthesized as separate registers instead of memory modules. The scalar registers could now be accessed simultaneously, leading to faster hardware. The emulation speed, shown in Figure 9, of this scalarized behavioral model showed a significant improvement of almost 4-5 times. This is a very important observation because behavioral models use loops and arrays extensively. If arrays get synthesized blindly into memory units, as is the norm in current synthesis research and commercial solutions, port constraints in memories lead to longer schedule lengths and higher latencies. Thus by converting appropriate arrays to scalar variables and removing simultaneous access constraints, we could achieve substantial improvements in emulation speeds. The scalarized model also shows a marginal improvement in simulation speed over the original model because of reduced array index computations. The problem of automating the process of identification of arrays to be mapped to scalar registers involves compiler analysis and a suitable synthesis model that predicts the consequences of the scalarisation transformation. A related problem is the partitioning of larger arrays to multiple memory modules for improving parallel access.

In summary, significant productivity benefits could result from unifying the performance and synthesis/emulation models in the processor/SoC design flow. However, the modeling methodologies for the two flows are often conflicting. In this

work we have developed an NoC emulation prototype, and through this modeling experience, identified some techniques for optimising behavioral models so that we obtain both high simulation speeds and an acceptable emulation speed to a level that is comparable to the harder-to-develop architectural models.

REFERENCES

- [1] John Wawrzyniek, David Patterson, Mark Oskin, Shih-Lien Lu, Christoforos E. Kozyrakis, James C. Hoe, Derek Chiou, and Krste Asanovic. RAMP: Research Accelerator for Multiple Processors. *IEEE Micro*, 27(2), pp. 46-57, 2007
- [2] Michael Pellauer, Muralidaran Vijayaraghavan, Michael Adler, Arvind and Joel Emer. A-Ports: an efficient abstraction for cycle-accurate performance models on FPGAs. In *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, pages 87-96, 2008.
- [3] Rissa T, Donlin A, and Luk W. Evaluation of SystemC modelling of reconfigurable embedded systems In *Proceedings of Design, Automation and Test in Europe*, pages 253-258, 2005
- [4] Keding, H. Coors, M. Luthje, O. and Meyr, H. Fast bit-true simulation. In *Proceedings of Design Automation Conference*, pages 708-713, 2001.
- [5] William Dally and Brian Towles. Principles and Practices of Interconnection Networks. Morgan Kaufmann Publishers Inc. 2003
- [6] Luca Benini and Giovanni De Micheli. Networks on Chips: A New SoC Paradigm. In *Computer*, pages 70-78, 2002.
- [7] <http://www.systemc.org>
- [8] <http://www.autoesl.com>
- [9] Derek Chiou, Dam Sunwoo, Joonsoo Kim, Nikhil Patil, William H. Reinhart, D. Eric Johnson and Zheng Xu The FAST methodology for high-speed SoC/computer simulation. In *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*, pages 295-302, 2007.
- [10] Wolkotte P.T., Holzspies P.K.F. and Smit G.J.M. Fast, Accurate and Detailed NoC Simulations In *Proceedings of the First International Symposium on Networks-on-Chip*, pages 323-332, 2007.
- [11] T. Suh, H.-H. S. Lee, S.-L. Lu, and J. Shen. Initial Observations of Hardware/Software Co-Simulation using FPGA in Architectural Research. In *Proceedings of the Workshop on Architecture Research using FPGA Platforms*, held at HPCA-12, Feb. 2006.