From Simulink to NoC-based MPSoC on FPGA

Francesco Robino, Johnny Öberg
Department of Electronic Systems
Royal Institute of Technology (KTH), Sweden
Email: {frobino,johnnyob}@kth.se

***Abstract*—Network-on-chip (NoC) based multi-processor systems are promising candidates for future embedded system platforms. However, because of their complexity, new high level modeling techniques are needed to design, simulate and synthesize embedded systems targeting NoC-based MPSoC.**

Simulink is a popular modeling environment suitable to model at system level. However, there is no clear standard to synthesize Simulink models into SW and HW towards a NoC-based MPSoC implementation. In addition, many of the proposed solutions require large overhead in terms of SW components and memory requirements, resulting in complex and customized multi-processor platforms.

In this paper we present a novel design flow to synthesize Simulink models onto a NoC-based MPSoC running on low-cost FPGAs. Our design flow constrains the MPSoC and the Simulink model to share a common semantics domain. This permits to reduce the need of resource consuming SW components, reducing the memory requirements on the platform. At the same time, performances (throughput) of dataflow applications can increase when the number of processors of the target platform is increased. This is shown through a case study on FPGA.

I. INTRODUCTION

We are approaching the sea-of-cores/processors era [2]. Multiprocessor systems-on-chip (MPSoCs) will soon be composed of hundreds of heterogeneous processing elements (PEs) [5]. Networks-on-chip (NoCs) [8] have been proposed as an efficient communication structure between them. To simplify the specification, verification and implementation of NoC-based MPSoCs, embedded system designers need new abstraction layers above register-transfer level (RTL).

System-level [5] has been proposed as the new abstraction layer, and system-level design (SLD) is considered the next frontier in electronic design automation (EDA). At the system-level, resources are defined in terms of abstract functions (system behavior) and blocks (system architecture). Design targets include both software (SW) and hardware (HW) [1], which are generated automatically to guarantee correct functionality, and optimal performance and resource utilization.

However, while tools and libraries such as Simulink [9] and SystemC has been shown to be suitable to model and simulate at system-level, automated mapping and refining of system-level models onto MPSoCs has been shown to be very difficult to achieve [15]. In addition, the lack of common semantics between the abstraction layers involved in the refinement process leads to design errors that are not caught until well into the implementation phase.

In the published literature, only a few number of end-to-end approaches from Simulink to MPSoC have been proposed. Some of them use ad-hoc expensive-complex platforms (e.g. [13]) and require resource consuming operating systems (OSs) to support the flow (e.g. [1]).

In this paper, we describe a SLD methodology which starts from a Simulink model and generates a NoC-based MPSoC implementation of the model. HW and SW description files are automatically generated for fast prototyping on low-cost FPGAs, characterized by low memory and logic elements availability. The generated MPSoC prototype behaves according to the semantics of the Simulink model, providing exactly same results as the simulation. The platform generated with our flow does not need OSs to support the end-to-end flow.

The contributions of this work are:

- to introduce a SLD flow enabling refinement of a Simulink model on to a NoC-based MPSoC.
- to connect the execution semantics of the Simulink model and the NoC-based MPSoC platform.
- a working prototype on low-cost FPGA, running without the need of an OS, reducing memory requirements.

II. RELATED WORK

K.Popovici et al. [13] describe a platform-based design flow starting from a Simulink model, and allowing easy experimentation of several mappings of the application onto an heterogeneous NoC-based MPSoC [12]. Depending on the Simulink blocks specification the application is then extracted into an intermediate model of functionality (C tasks) and platform (SystemC), enabling virtual prototyping of the entire system at different abstraction level. Virtual prototyping is carefully addressed, but less details are reported regarding a real prototype. The virtual prototype requires an OS (e.g. eCos) and uses 1 RISC processor and 1 DSP, while we present an OS free design with 4 RISC processors.

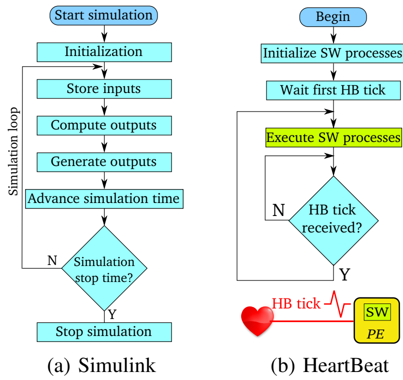
Kai Huang et al. [4] describe a system-level design flow using Simulink combined algorithm and architecture model (CAAM), a unified model that combines aspects related to the architecture (e.g. PEs available in the platform) into the algorithm model. From CAAM, it is possible to virtually prototype the entire system at different abstraction levels. In contrast to our approach, their PEs do not communicate through a NoC, they require an OS to support the flow, and the target platform is not a low-cost FPGA.

Caspi et al. [1] identify the periodic execution semantic of a subset of Simulink blocks. They are then able to transform the Simulink model into an intermediate layer described through the synchronous language Lustre [3], which executes functions periodically, triggered by a synchronous signal. Similarly, a MPSoC implementing the Time Triggered Architecture (TTA) model [7], enables significant events to happen periodically during specific time slots. Sharing such execution concept, a end-to-end design flow is proposed.

However, the TTA model can only be supported using an

[1]HW corresponds to implemented circuit elements (e.g. CPUs/PEs), SW corresponds to the instructions of functions performed by the HW.

978-3-9815370-2-4/DATE14/ © 2014 EDAA



(a) Simulink (b) HeartBeat

Fig. 1. Simulink simulation semantics and HeartBeat semantics

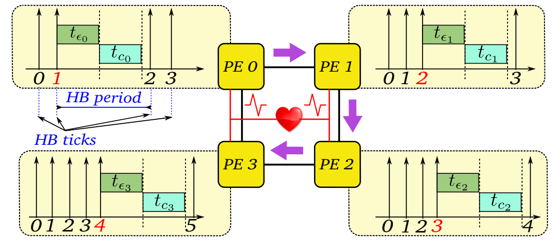


Fig. 2. MPSoC compliant with the HB model

OS on the PEs of the target MPSoC, and it requires complex HW to support synchronization between PEs without using a globally distributed clock.

III. Simulink Design Flow for Embedded Systems

A. An environment for system-level designs

Simulink [9] is a module of Matlab used to model, test and verify embedded systems. A Simulink model is graphically described through the use of *blocks* (e.g. an adder, a transfer function, etc.) and *subsystems* (a set of blocks), linked by *signals*. Using different blocks and subsystems, architecture and application specification can be combined in a mixed HW/SW model, as proposed by SLD methods. In this work, our proposal is to describe SW tasks through standard Simulink blocks, while the underlying HW is described using the concept of subsystem and signals.

B. Simulink execution model

The execution and validation of a Simulink model is performed through simulation. The semantics of the simulation process, shown in Fig. 1a, are documented in [13]. The first step of a simulation is the initialization. During this step the model is compiled to an executable file on the host machine. The compiler determines the invocation order of the blocks, depending on their dependencies.

After the initialization, the *simulation loop* is started. During the simulation loop the simulation time is frozen. Inputs are stored, subsequently the outputs and the state of the system are computed. Then the successive simulation time point where to evaluate the model is computed. Finally the simulation time is increased, and the loop starts again.

The successive time points at which the states and outputs are computed are called *time steps*. The length of time between steps is called *step size* (t_{step}). The simulation loop continues until the specified simulation time ends. Simulink provides a set of programs, known as *solvers*, to calculate outputs and step size. In this work we consider the *fixed-step solver*, solving the model without varying the step size from the beginning to the end of the simulation.

C. Simulink Embedded Coder

A Simulink model can be translated to C and C++ code for use on embedded processors through the Simulink Embedded Coder [9]. Following the Simulink execution model in Fig.1a, the generated program executes a background task, and it expects to be periodically interrupted by a timer. During the interrupt service routine (ISR), a generated function evaluating output and state of the system, `rt_OneStep`, must be executed. Executing `rt_OneStep` periodically in an ISR routine implements the simulation loop shown in Fig. 1a. Every tick of the timer the ISR is executed, inputs are stored, the system state is computed, and outputs are written out.

IV. From Simulink to NoC-based MPSoC

Compiling and mapping the code generated through the Embedded Coder onto NoC-based MPSoCs is still an open issue in the Simulink community. To enable an end-to-end SLD flow we follow the principles of the platform-based design methodology [15], constraining platform (MPSoC) and functionality (Simulink model) to share a common semantic domain. In this work, we aim to constrain the target MPSoC to reproduce the Simulink execution semantics shown in Fig.1a. The functions modeled in Simulink should be executed on the platform *once* each time step. In addition, the length of time between time steps (t_{step}) must be long enough to allow the platform to execute the functions. These conditions can be achieved if the platform behaves following the HeartBeat model.

A. The HeartBeat model

The HeartBeat (HB) model defines a set of rules constraining the execution semantics on a generic NoC-based MPSoC platform. A HB is a global periodic event which is made visible simultaneously to all PEs of the NoC-based MPSoC. Similarly to a clock in synchronous hardware, a HB can be represented through *HB ticks* repeated periodically with period t_{HB} (HB period). Every single received HB tick triggers a compute cycle on the PE. SW processes mapped on the PEs are executed once every HB tick, during an execution time t_{ϵ}. Afterwards, they can communicate with SW processes on other PEs through the NoC, taking a communication time t_c. The communicated data will be visible for the target PE on the following HB tick, so, in the HB model, functionalities running on different PEs become pipelined.

The principle is outlined in Fig.2. It shows a system following the HB semantics, composed by 4 PEs connected through a 2×2 mesh NoC. Each PE is a processor running a single SW process, communicating its result to a neighbor PE. The output from the SW process in PE0 will be available for the process running in PE1 one HB tick later, the output from PE1 will be available for PE2 one HB tick later, etc.

The HB execution semantics are enforced on each PE by the *HB wrapper*. It is an initial program containing the main function of the local PE. It embeds the SW processes mapped on the PE, and triggers their execution and communication, based on the HB tick, as shown in Fig.1b.

TABLE I. COMMON SEMANTICS PARAMETERS AND DESIGN RULES

Simulink	HB compliant MPSoC
time steps	HB ticks
step size (t_{step})	HB period (t_{HB})
simulation loop	SW processes triggered by HB wrapper
rt_onestep	SW running on one PE
blocks	instuctions of SW process
subsystem	SW processes on a single PE (rt_onestep)
signal	NoC communication path

B. Connecting Simulink and HB NoC-based MPSoC semantics

From Fig.1a and Fig.1b, we see that SW processes triggered by the HB wrapper execute once each HB period. Therefore, they are equivalent to the `rt_onestep` function that is executed in the simulation loop each Simulink simulation step. Consequently, if we map a single `rt_onestep` function on a single PE on the HB compliant platform, and $t_{step} = t_{HB}$, we will get exactly the same behavior.

The previous observation is clarified in Table I where it is shown that each parameter characterizing the Simulink semantics has a counterpart in a HB compliant MPSoC. This means that we can use Simulink models to program HB compliant MPSoCs. Simulink blocks can be used to describe instructions of functions performed by a PE (SW processes). Subsystems, which are composed by sets of blocks, are used to create SW processes that should run on different PEs. Each subsystem generates its own `rt_onestep` function, which is then mapped on a PE. Simulink signals connects subsystems (PEs), and they are mapped to the NoC intercommunication paths. These design rules are summarized in Table I.

C. The design flow

To enable an end-to-end design flow from Simulink to NoC-based MPSoC on FPGA, we use the NoC System Generator (NSG) tool [11] as back-end. The tool requires a XML input file, describing the platform architecture, and it generates synthesizable VHDL describing a multi-processor platform. PEs can be chosen between Nios2, uBlaze or Leon3 cores, and they are connected through the Nostrum NoC architecture [8].

SW processes running on the PEs are provided by the user as C code, and if the user specifies on which PE each C file should be mapped (through the XML input file), the tool automatically download the compiled C code on the specified PEs. NSG can generate MPSoC platforms compliant with the HeartBeat model [14], including a clock divider shared between the PEs, generating HB ticks.

In our design flow, shown in Fig.3, we extract the `rt_onestep` functions from the C files generated by the Embedded Coder for each subsystem. We then specify in the XML file to map a single `rt_onestep` function for each PE. NSG embeds each `rt_onestep` function in an HB wrapper, triggering their execution once each t_{HB}.

The t_{HB} for the generated platform must be specified in the XML file. It must comply to the following time constraints:

$$t_{step} = t_{HB} \geq \max_{i=0}^{M}(t_{\epsilon_i} + t_{c_i}) \quad (1)$$

where M is the number of PEs, while t_{ϵ_i} and t_{c_i} are the *worst case execution time* (WCET) and *worst case communication time* (WCCT) of the SW process(es) running on node PE_i, as introduced in Sec. IV-A.

TABLE II. WCET, MINIMUM t_{HB}, MEMORY REQUIREMENTS

	1 PE	4 PEs			
		Source	Noise	Filter	Sink
WCET - Min. t_{HB} [ms]	28	7,90	11,68	8,00	0,01
Mem. req. w/o OS [KB]	53	33	27	21	16
Mem. req. eCos [KB]	+20	+20 for each PE			
Mem. req. uCLinux [MB]	+2	+2 for each PE			

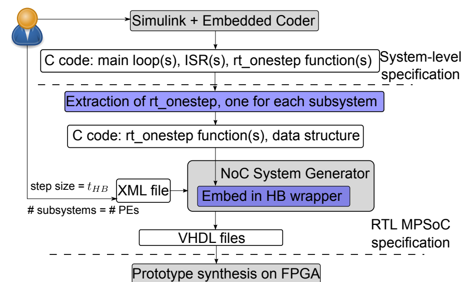


Fig. 3. Simulink to MPSoC design flow

V. CASE STUDY

To prove the feasibility of our approach, we have implemented the whole SLD flow starting from a Simulink model of a digital signal processing (DSP) application. The DSP application used in this case study is a Simulink tutorial [10], shown in Fig.4a. A sinusoidal source block generates a sinusoidal signal. Then, some random noise, generated by a random source block and a digital high pass FIR filter, is added to the sinusoidal signal. The noisy signal is then filtered by a low pass FIR filter, which removes the noise component. Fig.4c shows the most significant signals of the modeled system. The upper plot shows the signal produced by the sinusoidal source block and the noisy signal. In the lower plot, the red signal is the result of the noisy signal filtered through the low pass FIR filter. The design runs on a Cyclone IVE FPGA.

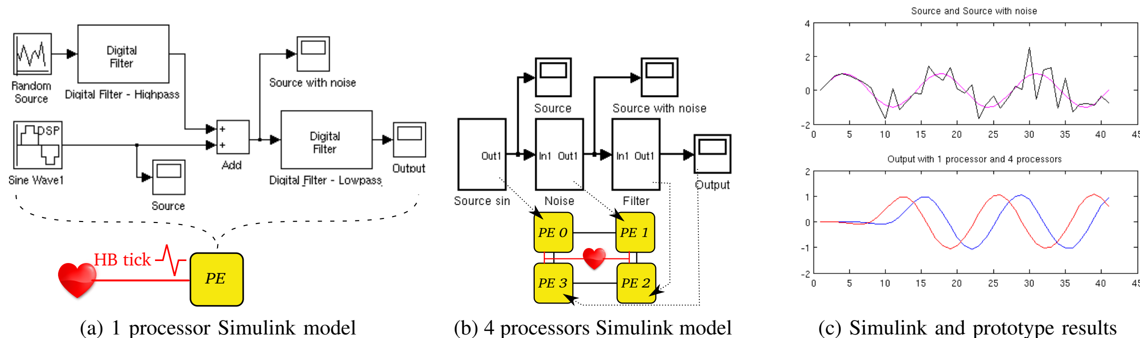
A. From Simulink to single processor

If the designer does not specify any subsystem, he/she is targeting the entire DSP system on a single processor, following the methodology proposed by the Embedded Coder [9] and shown in Fig.4a. The designer configures NSG to map the only `rt_onestep` function on a single PE. The function is triggered every HB tick, and it exhibits the exact same results between Simulink simulation and FPGA prototype, as shown in the red signal in Fig.4c (simulation and FPGA output is identical).

To *maximize the throughput we want to run our platform with the minimum HB period* (t_{HB}). The minimum t_{HB} can be calculated using Eq. 1, where $M = 1$ and $t_c = 0$ (no communication through the NoC). The value of t_ϵ, representing the WCET of `rt_onestep` on the selected PE (Nios2/e), has been found through extensive emulations, measuring start and end time of the executed function through a 50 MHz timer, and it is reported in Table II. For a single processor system, the maximum throughput (minimum t_{HB}) we could achieve on the FPGA prototype was one output sample every 28 ms.

B. From Simulink to 4 processor system

If the designer divides the DSP system in 4 subsystems, he/she can configure NSG to map the DSP system on a



(a) 1 processor Simulink model (b) 4 processors Simulink model (c) Simulink and prototype results

Fig. 4. Experiment setup and results

HB compliant platform, composed by 4 Nios2/e soft-cores connected through a 2×2 NoC, as shown in Fig.4b. Each subsystem will be mapped on a different PE, in accordance to the Simulink/HB translation rules shown in Table I. The first subsystem, *Source*, contains the sinusoidal source block. The second, *Noise*, contains the noise generator and the high pass FIR filter. The third, *Filter*, contains the low pass FIR filter, while the fourth, *Scope*, is just printing out the results.

The Embedded Coder generates one `rt_onestep` function for each subsystem. Each function is then automatically embed in a HB wrapper and mapped to one of the 4 PEs of the NoC-based MPSoC.

In this situation, to find the minimum t_{HB} (and so the maximum throughput) we use Eq. 1, with $M = 4$. t_ϵ for each subsystem has been evaluated with the same methodology presented in the previous subsection, and reported in Table II. From the table, we can see that t_ϵ of the *Noise* subsystem, 11.68 ms, is the largest. This means that it is the bottleneck of our system, defining the minimum t_{HB}. A upper bound of t_c for the Nostrum based platform created by the NSG, can be found through the following equation, discussed in [6], [14]:

$$t_c \leq n_{packets} \cdot (RNI_{send} + 5DN + RNI_{rcv}) \quad (2)$$

where $n_{packets}$ is the number of packets to be sent, RNI_{send} and RNI_{rcv} are the time it takes to inject/fetch a packet in/from the network, D is the diameter of the NoC, N is the maximum number of packets with highest priority in the network. For our 2×2 NoC, $t_c \leq 328$ ticks< 0.01 ms. This is several orders of magnitude less than the minimum t_{HB}, 11.68 ms, so t_c can safely be ignored in this example.

The lower plot in Fig.4c shows a comparison between the output signal of the prototype running on a single PE (red line) and the output signal of the prototype running on 4 PEs. The results are the same, but shifted in time since the 4 processor system functionality is pipelined across 3 HB periods.

Splitting the system in 4 subsystems using our methodology, increase the throughput of the system of $\sim 2.4\times$. If we would have created 4 subsystems having equal WCET (i.e. 7 ms), we could have reached a theoretical $4\times$ throughput increase. The increase in throughput comes at the expense of memory. The total on-chip memory needed by 4 PEs is larger than the one needed from the single PE case, because each PE does not need only the set of instruction represented by the `rt_onestep` function, but also boot code for each processor, in addition to the code for implementing the HB semantics.

Table II shows the on-chip memory needed for each PE (Nios2/e) when we map the whole system on 1 PE, and when we spread it on 4 PEs. However, our design flow still saves memory resources if compared with other flows using OSs (e.g. [1], [4], [13]). As shown in Table II, adding an OS on each PE requires an additional overhead of at least 20 KB for each PE, in addition to the memory needed to store the code.

VI. Conclusions

We have described a SLD flow that allows the synthesis of Simulink models to NoC-based MPSoCs, generating a working prototype on low-cost FPGAs.

The generated MPSoC is constrained to share a common semantics domain with the Simulink model, so that the results between simulation and implementation of the prototype are the same, without the need of resource consuming SW components (such as operating systems).

The proposed flow has been shown through a case study on FPGA, which is used to discuss benefits in terms of memory consumption and performances, together with the limitations.

References

[1] P. Caspi et al. From Simulink to scade/Lustre to TTA: a layered approach for distributed embedded applications. In *Proc. of conf. on Language, compiler, and tool for embedded systems*, LCTES, 2003.

[2] S. Davidson. Sailing on a sea of processors. *Design Test of Computers, IEEE*, 16(4):112, oct-dec 1999.

[3] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language Lustre. *Proc. of the IEEE*, 1991.

[4] K. Huang et al. Simulink-based MPSoC design flow: Case study of motion-jpeg and H.264. In *Design Automation Conference, DAC*, 2007.

[5] *The International Technology Roadmap for Semiconductors (ITRS), Design*, 2011. http://www.itrs.net/.

[6] A. Jantsch. Models of computation for networks on chip. In *Application of Concurrency to System Design. ACSD 2006.*, pages 165–178, 2006.

[7] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112 – 126, jan 2003.

[8] S. Kumar et al. A network on chip architecture and design methodology. In *VLSI. Proc. IEEE Computer Society Annual Symposium on*, 2002.

[9] Mathworks. Simulink documentation center. Website. http://www.mathworks.se/help/simulink/.

[10] Mathworks. Digital filter block. Website, August 2013. http://www.mathworks.se/help/dsp/ug/digital-filter-block.html.

[11] J. Öberg and F. Robino. A NoC system generator for the sea-of-cores era. In *Proc. of the 8th FPGAWorld Conference*, FPGAWorld '11, 2011.

[12] K. Popovici and A. Jerraya. Simulink based hardware-software codesign flow for heterogeneous MPSoC. In *Proc. of the Summer Computer Simulation Conference*, SCSC, 2007.

[13] K. Popovici et al. Embedded systems design: Hardware and software interaction. In *Embedded Software Design and Programming of Multiprocessor System-on-Chip*, Embedded Systems. Springer, 2010.

[14] F. Robino and J. Öberg. The HeartBeat model: a platform abstraction enabling fast prototyping of real-time applications on NoC-based MPSoC on FPGA. In *ReCoSoC*, 2013.

[15] A. Sangiovanni-Vincentelli. Quo vadis SLD: Reasoning about trends and challenges of system-level design. *IEEE*, 95(3):467–506, 2007.