An Automated Design Flow to Prototype Simulink Models on MPSoC

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

***Abstract*—Simulink is a modeling environment suitable to model embedded systems at system-level. However, there is no standard to prototype Simulink models onto modern multi-processor system-on-chip (MPSoC). In this demonstration we show how our NoC System Generator tool can be used as part of an automated platform-based design flow to synthesize a Simulink model onto a network-on-chip based MPSoC implementation on FPGA. The performance of the generated prototype scales with the number of processors.**

I. Introduction

Simulink [1] is an industrial de-facto standard for building executable models of embedded systems and their environments, facilitating validation by simulation. In Simulink, systems are graphically described through the use of *blocks* (e.g. an adder, a transfer function, etc.) and *subsystems* (a set of blocks), linked by *signals*. Architecture and application specification can be combined in a mixed HW/SW model, enabling Simulink to be classified as *system-level design* (SLD) method [4].

However, automating the generation of a working prototype from a system-level model is not a trivial process. In addition, there is the need of techniques enabling fast-prototyping of system-level models onto state of the art multi-processor systems on chip (MPSoCs) [4].

In this demo, we show an automated design flow to synthesize a Simulink model onto a network-on-chip (NoC) based MPSoC implemented on FPGA. The flow is based on the NoC System Generator (NSG) tool [2], and it implements a platform-based design methodology, constraining the Simulink model and MPSoC to share a common semantics domain [3].

II. The Design Flow

Fig.1 shows the design flow. The user starts simulating a FIR low-pass filter using Simulink. The filter is composed by 4 subsystems, each one represented with a different color. The pink subsystem generates a sinusoidal signal, the blue one adds to it random noise, the yellow one filters the noise component, while the green one displays the filtered signal.

The Simulink Embedded Coder [1] automatically generates C source code for real-time implementation of each subsystem. Each 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 the subsystem functionality, `rt_OneStep`, must be executed. This represents the Simulink execution semantics.

The flow extracts the C code of each `rt_OneStep` function, so that it can be embedded as a pure function on the code running on each processor of the target MPSoC. In addition, communication routines to use the underlying NoC are added in the code. For example, the blue subsystem (noise generator) needs to receive data from the pink subsystem (sinusoidal generator) and send its result to the yellow one (filter). Its `rt_OneStep` function is consequently updated with a `receive` and `send` routine, enabling the communication through the NoC.

After this refinement, the C code is provided to the NSG tool. Together with an XML file, which describes the kind of MPSoC where we want to prototype the system (e.g. number and kind of processors), the NSG tool automatically generates

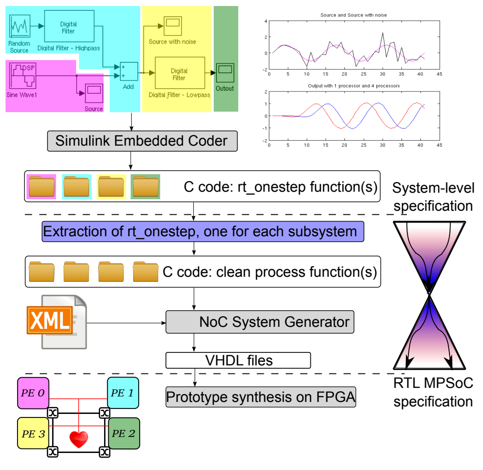
Random Source
Digital Filter
Digital Filter - Highpass
Source with noise
Sine Wave1
Source
Add
Digital Filter
Digital Filter - Lowpass
Outout
Source and Source with noise
Output with 1 processor and 4 processors
Simulink Embedded Coder
C code: rt_onestep function(s)
System-level specification
Extraction of rt_onestep, one for each subsystem
C code: clean process function(s)
XML
NoC System Generator
VHDL files
Prototype synthesis on FPGA
RTL MPSoC specification
PE 0
PE 1
PE 2
PE 3

Fig. 1. Simulink to MPSoC design flow

HDL files for the NoC and initializes the MPSoC components. In addition, it spreads and compiles the provided software on the processors, compiling the C code from each subsystem on a different processor. Finally the user configures the FPGA with the generated files to implement a working prototype.

In order to ensure the same results between the Simulink model and the MPSoC prototype, the NSG tool enforces the generated platform to exhibit the same execution semantics of the Simulink simulator. In fact, the software running on the processors is constrained to execute and communicate data through the NoC only on periodic events, provided by a hardware timer in the platform [3]. This reflects the execution semantics of the code generated by the Embedded Coder (executing during periodic ISRs) and ensures that the results provided by the prototype are the same as the simulation. This technique reflects the concepts of the platform-based design methodology, constraining MPSoC and high level model to share a common semantics domain [4].

III. Conclusions

In this demo, we show an automated design flow to prototype Simulink models on NoC-based MPSoC through the NSG tool. The main advantage of the flow is to enable fast prototyping. It also enables performance improvements. The presented example increase the throughput of the system of $2.4\times$, spreading the computation through 4 processors instead of running it on a single processor. With a N processors MPSoC, it is possible to reach a theoretical $N\times$ throughput increase by balancing the execution time of the subsystems in the Simulink model.

References

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

[2] 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.

[3] 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.

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