From Simulink to NoC-based MPSoC on FPGA
Simulink front-end for the NoC System Generator (NSG)

Francesco Robino
KTH Royal Institute of Technology

ICES seminar
Overview of the talk

- Motivation
  - What is the problem?
  - Our goal

- Reaching the goal
  - Simulink simulation semantics
  - The HeartBeat (HB) model in a MPSoC generated by NSG
  - Connecting Simulink and HB NoC-based MPSoC semantics
  - Experimental evidences and results

- Conclusion
Matlab/Simulink is today’s de-facto standard for model-based design in domains such as control engineering and signal processing.

NoC-based MPSoCs are promising candidates for future embedded system (potentially high performances, low power consumption, ...).

Synthesis of a Simulink model onto NoC-based MPSoCs is still an open issue.
Our goal

- To enable an end-to-end design flow, we follow the principles of the platform-based design methodology, constraining platform (MPSoC) and functionality (Simulink model) to share a common semantic domain.
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.
Simulink simulates a dynamic system by computing its states at successive time steps over a specified time span, using information provided by the model.

A solver determines the time of the next simulation step and applies a numerical method to solve the set of ordinary differential equations (ODEs) that represent the model.

Different solvers embody different approaches to solve a model.
Solvers and Simulink simulation semantics

Solvers:

- fixed-step VS variable-step
- discrete VS continuous
  - Continuous: compute model’s continuous states at the current time from the states at previous time steps and the state derivatives (requires ordinary differential equations).
- one-step VS multi-step
  - One-step solvers estimate $y(t_n)$ using only the solution at the preceding time point $y(t_{n-1})$
  - Multistep solvers use the results at several preceding time steps to compute the solution
Our approach today targets the following solver configuration:
- fixed-step (constant step size $t_{\text{step}}$)
- discrete
- one-step

However, it can be extended to other solvers too…
When we select a fixed-step solver, we can use the Simulink Embedded Coder to generate C code of the model for use on embedded processors. The code generated includes:

- Main scheduler sensitive on interrupt.
- `rt_onestep` function, implemented in the interrupt service routine (ISR), describing the functionality of the system.

The generated software is compliant with the execution model of the Simulink simulation!
The HeartBeat model in a MPSoC generated by NSG

Simulink model

NoC System Generator

HB process wrapper

rt_onestep

process network
and system specs

PE 0
PE 1
PE 2
PE 3

HB ticks

HB period

Begin

Initialize SW processes

Wait first HB tick

Execute SW processes

HB tick received?

Y
N

PE

HB tick SW

Initialize SW processes

Wait first HB tick

Execute SW processes

HB tick

SW

PE

HB ticks

HB period

PE 0
PE 1
PE 2
PE 3

NoC

HB ticks

HB period

PE 0
PE 1
PE 2
PE 3

NoC
Table: Common semantics parameters and design rules

<table>
<thead>
<tr>
<th>Simulink</th>
<th>HB compliant MPSoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>time steps</td>
<td>HB ticks</td>
</tr>
<tr>
<td>step size ((t_{step}))</td>
<td>HB period ((t_{HB}))</td>
</tr>
<tr>
<td>simulation loop</td>
<td>SW processes triggered by HB wrapper</td>
</tr>
<tr>
<td>rt_onestep</td>
<td>SW running on one PE</td>
</tr>
<tr>
<td>blocks</td>
<td>instructions of SW process</td>
</tr>
<tr>
<td>subsystem</td>
<td>SW processes on a single PE (rt_onestep)</td>
</tr>
<tr>
<td>signal</td>
<td>NoC communication path</td>
</tr>
</tbody>
</table>

F. Robino (KTH)

From Simulink to MPSoC on FPGA

20-05-2014 11 / 17
Case study: DSP system (Digital FIR filter)

Case study: Embedded coder vs SLD HB methodology

From Simulink to MPSoC on FPGA
Table: WCET, minimum $t_{HB}$, memory requirements

<table>
<thead>
<tr>
<th></th>
<th>Source</th>
<th>Noise</th>
<th>Filter</th>
<th>Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCET - Min. $t_{HB}$ [ms]</td>
<td>28</td>
<td>7,90</td>
<td>11,68</td>
<td>8,00</td>
</tr>
<tr>
<td>Mem. req. w/o OS [KB]</td>
<td>53</td>
<td>33</td>
<td>27</td>
<td>21</td>
</tr>
<tr>
<td>Mem. req. eCos [KB]</td>
<td>+20</td>
<td></td>
<td>+20 for each PE</td>
<td></td>
</tr>
<tr>
<td>Mem. req. uCLinux [MB]</td>
<td>+2</td>
<td></td>
<td>+2 for each PE</td>
<td></td>
</tr>
</tbody>
</table>

- Splitting the system in 4 subsystems using this methodology, increases the throughput of the system of ca $2.4 \times$. 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.
Case study: semantics preserving?

\[
\begin{align*}
[0.00003, 0.00006, -0.001011, -0.006998, ...] \\
[X, -0.014091, 0.043682, 0.440711, ...] \\
[0.000000, 0.453990, 0.809017, 0.987688, ...] \\
0.809017, 0.987688, ...]
\end{align*}
\]
Conclusions

- We have described a system-level design 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 OS).
- Design methodology based on process constructors — HB process wrappers to provide execution semantics to the MPSoC.
- Developed a synthesis methodology with similarities with synchronous HW design
  - Minimize HB period $t_{HB}$.
  - Exploits task, data and pipeline parallelism.
Questions?

Suggestions:

- Can this approach be extended to other Simulink solvers?
- Does this approach provide real-time guarantees?
  - No OS overhead
  - Quite precise measurement of WCET
  - When no shared connections, quite precise (and not pessimistic) WCCT
- Why not everything asynchronous\(^1\)? (see asynchronous CPUs)

---

\(^1\) Asynchronous circuits are not governed by a global clock, but they use signals to indicate completion of instructions and operations, specified by data transfer protocols.