Automatic Conversion of Simulink Models to SysteMoC Actor Networks

Martin Letras, Joachim Falk, Stefan Wildermann, and Jürgen Teich
Hardware/Software Co-Design, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
SCOPES’17, St. Goar, Germany, June 12, 2017
Outline

• Introduction
• Data Flow Graphs and Simulink
• Automatic Conversion Method
• Case Study
• Conclusions
Motivation

● There exist automatic translation toolchains for generation of C or C++ code from Simulink models

What about heterogeneous SoC architectures?

● Intuitive block-based design
● Simulation
● Rapid prototyping

Source: Mathworks

Source: Cypress
Introduction
Introduction (1)

Simulink:
- Rapid prototyping and design tool
- Mainly focused on the signal processing domain
- Toolboxes for different applications

• Integrate Simulink in an ESL methodology
Integrate Simulink in an ESL methodology

However, automatic code generation for heterogeneous target architectures consisting of GPPs and hardware accelerators is currently not supported by Simulink.
Introduction (3)

- Convert Simulink specifications to the input language of a well established ESL flow

- Facilitate hardware/software co-optimization and Design Space Exploration (DSE)

- Automatic generation of hardware/software co-designs
Data Flow Graphs and Simulink
Data Flow Graphs in SysteMoC (1)

- Data flow allows modeling concurrent systems by concurrently executing actors.

- SysteMoC is the input language of SystemCoDesigner.

- A Data Flow Graph (DFG) is realized by a so-called actor network.
Data Flow Graphs in SysteMoC (2)

- Actors communication via channels only
Simulink

- **Simulink**
  - The basic elements are functional blocks

- The blocks can be composed in subsystems
Although there are many similarities between the Data Flow Graphs and Simulink, we must have in mind the next issues:

- **Data-triggered execution vs. Time-step execution**
- **Some Simulink structures are not allowed in Data Flow**
- **Multi-rate systems**
Automatic Conversion Method
Automatic Conversion Method

Extract Hierarchy

Extract Parameters

Apply Transformations

SysteMoc Code Generation

---

class: Add
definition: public

smoc_actor

{ public:
    /* Port Definition: */
    smoc_port_in<real_T> in_1;
    smoc_port_in<real_T> in_2;
    smoc_port_out<real_T> out_1;
    add(sc_module_name name):
        smoc_actor(name, start)
        {
            start = in_1(1) >> in_2(1) >>
                out_0(1) >>
                CALL(add::method_add) >> start;
        }

    protected:
        void method_add()
        {
            add_initialize();
            add_U.in1 = in_1[0];
            add_U.in2 = in_2[0];
            add_step();
            out_0[0] = add_Y.Out1;
        }
        smoc_firing_state start; 
    }
Extraction of the Hierarchy (1)

- Hierarchical structures are not defined in DFG
- Each atomic block is mapped to an Actor
- Create a flat graph for each subsystem in the Simulink
Create a flat graph but there are two additional signals:

- The *enable signal* determines when the subsystem is active
- The *trigger signal* activates the subsystem according to a trigger event
Extraction of the Hierarchy (3)

- Channels are exclusive for a source and sink actor
- Data replication on lines are converted by replicating output data
Extraction of Parameters

- Extract number of input/output ports
- Extract data types
- Resolve data types
  - Support for arrays and matrices
  - Encapsulate data in a token object
- Extraction of Sample time
- Extraction of specific parameters of each block

```cpp
template <typename T, int SIZE>
Token<T,SIZE>::Token()
{
}

template <typename T, int SIZE>
T &Token<T,SIZE>::operator[](int index)
{
  assert(0 <= index && index < SIZE);
  return Token<T,SIZE>::Data[index];
}
```
Apply Transformations (1)

- Time-step execution allows multiple execution rates
- An actor and two FIFOs are added to regulate the write/read operations
Apply Transformations (2)

- Without a correct initialization dead-locks may be reached
- An initial token is added in the FIFO
SysteMoC Code Generation (1)

Simulink Model

Data Flow Graph

Simulink Coder

Model.rtw

actor_generator.tlc

Target Files

Target Language Compiler

Generated makefile

Generated SysteMoC Code Files

vector_Sum()

start
SysteMoC Code Generation (1)

```cpp
class vector_sum: public smoc_actor
{
  public:
  /* Port Definition: */
  smoc_port_in<Token < real_T >> in_1;
  smoc_port_in<Token < real_T >> in_2;
  smoc_port_in<Token < real_T >> in_3;
  smoc_port_out<Token < real_T >> out_1;
  vector_sum(sc_module_name name): smoc_actor(name, start)
  {
    start = in_1(1)>> in_2(1)>> in_3(1)>> out_0(1) >>
    CALL(vector_sum::method_vector_sum) >> start;
  }
  protected:
  void method_vector_sum()
  {
    Token < real_T > token_out_0(128);
    vector_sum_initialize();
    for(int i = 0; i < 128; i++) {
      vector_sum.U.In1[i] = in_1[0].Data[i];
      vector_sum.U.In2[i] = in_2[0].Data[i];
      vector_sum.U.In3[i] = in_3[0].Data[i];
    }
    vector_sum_step();
    for(int i = 0; i < 128; i++)
    {
      token_out_1.Data[i] = vector_sum.Y.Out1[i];
    }
    out_0[0] = token_out_0;
  }
  smoc_firing_state start; }
```
class Graph_System: public smoc_graph
{
public:
    /* Actors Definition */
    sine_Wave sine_wave;
sine_Wave1 sine_wave1;
sine_Wave2 sine_wave2;
vector_Sum vec_sum;
FFT fft;
ComplexToMag complex_to_mag;
VectorScope vector_scope;

    Graph_System(sc_module_name name): smoc_graph(name),
    {
        connectNodePorts(sine_wave.out, vec_sum.in_1);
        connectNodePorts(sine_wave1.out, vec_sum.in_2);
        connectNodePorts(sine_wave2.out, vec_sum.in_3);
        connectNodePorts(vec_sum.out, fft.in);
        connectNodePorts(fft.out, complex_to_mag.in);
        connectNodePorts(complex_to_mag.out, vector_scope.in);
    }
};
Case Study
Case Study (1)

Application that performs a FFT, extracts the maximum frequency and magnitude from an input signal
The input signal is the function

\[ f(x) = 0.3 \sin(2\pi 40t) + \sin(2\pi 200t) + 0.6(2\pi 300t) \]

The DFG is composed by 21 actors and 16 FIFOs.
Case Study (3)

Output of Simulink

Output of the SysteMoC actor network

Extract the three frequencies in the signal (40, 200 and 300 Hz.)
### Comparison of related work

<table>
<thead>
<tr>
<th>Work</th>
<th>MoC Employed</th>
<th>Library Required</th>
<th>Target Language</th>
<th>Multirate Systems</th>
<th>Triggered Systems</th>
<th>DSE</th>
<th>Offset handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhou [3]</td>
<td>EFA</td>
<td>✓</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Warsitz [4]</td>
<td>KPN</td>
<td>✓</td>
<td>C and C++</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Boström[5]</td>
<td>SDF</td>
<td>✓</td>
<td>Boogie</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Chessa[6]</td>
<td>SDF</td>
<td>✓</td>
<td>C and C++</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Caspi[8, 9]</td>
<td>SDF</td>
<td>✓</td>
<td>Lustre</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Zhang[10]</td>
<td>SDF</td>
<td>✓</td>
<td>SysteMoC</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td><strong>Our work</strong></td>
<td><strong>SDF</strong></td>
<td><strong>✗</strong></td>
<td><strong>SysteMoC</strong></td>
<td><strong>✓</strong></td>
<td><strong>✓</strong></td>
<td></td>
<td><strong>✗</strong></td>
</tr>
</tbody>
</table>
Conclusions and Outlook

- We have presented a method for the automatic transformation of Simulink models to actor networks

- We are only constrained - and our translation coverage limited - by the class of Simulink blocks that can be converted by Simulink Coder™

- Independence from the input file (slx or mdl), and the ability of handling different data types (integers, floats, vectors as well as matrices)
Thank you
Questions?
References


Finite State Machine for Actors

- **Init**
  - Method: `method_init()`

- **Step**
  - Input: `#≥1 & checkStep()`
  - Output: `#≥1 / method_step()`

- **Terminate**
  - Input: `#≥1 & checkStep()`
  - Output: `#≥1 / method_terminate()`
Triggered Subsystems
Finite State Machines for Triggered Subsystems

- Init
  - Method: method_init()
  - Transition: trigger #≥1 & !checkTrigger() & output#≥1 / method_hold()

- Start
  - Transition: trigger #≥1 & checkTrigger() & output#≥1 / method_step()

- Hold
  - Transition: trigger #≥1 & !checkTrigger() & output#≥1 / method_hold()

- Step
  - Transition: trigger #≥1 & checkTrigger() & output#≥1 / method_step()
Enabled Subsystems
Finite State Machines for Enabled Subsystems

Finite State machine for hold/hold output