Synthesis of Distributed Synchronous Specifications to SysteMoC

Mohamed Ammar Ben Khadra\textsuperscript{1}, Yu Bai\textsuperscript{2}, Klaus Schneider\textsuperscript{2}

\textsuperscript{1}EDA Group, \textsuperscript{2}Embedded Systems Group
TU Kaiserslautern

10.03.2014
MBMV’14, Böblingen, Deutschland
Outline

1 Motivation

2 Concepts introduced

3 Synthesis method

4 Results
Complexity of embedded systems

- Increased complexity of embedded systems is coupled with increased demand for more performance.

- Meeting correctness and non-functional requirements in multi-core software is challenging.

- Model-based design is a widely accepted methodology for addressing complexity and design-gap productivity issues.
The role of synchronous modeling

- Model-based design using synchronous languages has been commercially successful for synthesis of uni-processor software.

- Large theory has been developed for distributed software synthesis but there is a lack of simulation tools.

- We think that a model-based flow should mix multiple Models of Computation (MoC) to achieve better expressiveness.
A mixed MoC design flow

- Among many MoCs, we found that the synchronous and actor-oriented MoC complement each other nicely.

- Synchronous MoC is formally-verifiable, composable, and has explicit modeling of time.

- Actor-oriented MoC fits better for mapping and scheduling on multi-core systems.
Synthesis of Distributed Synchronous Specifications to SysteMoC

1. Motivation

2. Concepts introduced

3. Synthesis method

4. Results
Contribution

- A method for the synthesis (code generation) of elastic networks to SysteMoC.

- Elastic networks are a generic distributed form that consists of *Synchronous Modules* connected by *Synchronous Channels*.

- SysteMoC is an open-source actor-oriented modeling library based on SystemC.
Flow and operational semantics
1 Motivation

2 Concepts introduced

3 Synthesis method

4 Results
Synchronous module synthesis (1): model to GA

module MAC(int ?a, ?b, !s) {
    int i, t, o;
    loop {
        w1: pause;
        t = i;
        o = a * t;
    } ||
    loop {
        w2: pause;
        i = a + b;
        if (t < 0) {
            w3: pause;
            s = o;
        } else {
            w4: pause;
            if (b > 0)
                s = o + 1;
        }
    }
}

\[
\begin{align*}
\alpha_1 : & \quad \text{start} \lor w_1 \Rightarrow \text{next}(w_1) = \text{true} \\
\alpha_2 : & \quad \text{start} \lor w_3 \lor w_4 \Rightarrow \text{next}(w_2) = \text{true} \\
\alpha_3 : & \quad w_2 \land (t < 0) \Rightarrow \text{next}(w_3) = \text{true} \\
\alpha_4 : & \quad w_2 \land \neg(t < 0) \Rightarrow \text{next}(w_4) = \text{true} \\
\beta_1 : & \quad w_1 \Rightarrow t = i \\
\beta_2 : & \quad w_1 \Rightarrow o = a \times t \\
\beta_3 : & \quad w_2 \Rightarrow i = a + b \\
\beta_4 : & \quad w_3 \Rightarrow s = o \\
\beta_5 : & \quad w_4 \land (b > 0) \Rightarrow s = o + 1
\end{align*}
\]
Synchronous module synthesis (2): GA to EFSM

\[
\begin{align*}
\alpha_1 & : \text{start} \lor w_1 \Rightarrow \text{next}(w_1) = \text{true} \\
\alpha_2 & : \text{start} \lor w_3 \lor w_4 \Rightarrow \text{next}(w_2) = \text{true} \\
\alpha_3 & : w_2 \land (t < 0) \Rightarrow \text{next}(w_3) = \text{true} \\
\alpha_4 & : w_2 \land \neg(t < 0) \Rightarrow \text{next}(w_4) = \text{true} \\
\beta_1 & : w_1 \Rightarrow t = i \\
\beta_2 & : w_1 \Rightarrow o = a \times t \\
\beta_3 & : w_2 \Rightarrow i = a + b \\
\beta_4 & : w_3 \Rightarrow s = o \\
\beta_5 & : w_4 \land (b > 0) \Rightarrow s = o + 1
\end{align*}
\]
Synchronous module synthesis (3): EFSM to C++ code

- Actor model of module MAC (left) actor C++ code (right)

```cpp
class MAC : public smoc_actor {
    SC_HAS_PROCESS(MAC);
    smoc_firing_state s0, s1, s2, s3;
    // Definitions omitted.
    public:
        smoc_port_in<int> a, b; smoc_port_out<int> s;
        MAC(sc_module_name name) : // Constructor instantiations omitted.
        { // Only firing transitions of s1 are shown.
            s1=TILL(clk)<<(a(1)&b(1)&(guard1)>>s(1)
            >>CALL(MAC::action1)>>s2|
                TILL(clk)<<(a(1)&b(1)&(guard2)>>s(1)
                >>CALL(MAC::action1)>>s3;
        });
```

- The details of this synthesis procedure have been omitted from this work due to the lack of space. We refer the interested reader to [24] for a more detailed treatment.
Two synchronous channels with the same $\text{Delay} = 3$ and different $\text{Capacity}$, (a) set with $\text{Capacity} = 6$ and (b) set with $\text{Capacity} = 4$. 
Synchronous channel synthesis (2): channel buffer

- Each Channel Buffer models one clock cycle delay on a synchronous channel.
- FIFOs (smoc_fifo) model token storage on a synchronous channel.
1 Motivation

2 Concepts introduced

3 Synthesis method

4 Results
Experimental results

<table>
<thead>
<tr>
<th>Model</th>
<th>Time</th>
<th>LoC</th>
<th>Boolexp</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heron Sqr Root</td>
<td>0.1 s</td>
<td>462</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>Cruise Control</td>
<td>1 s</td>
<td>1266</td>
<td>335</td>
<td>11</td>
</tr>
<tr>
<td>SHA (Basic)</td>
<td>3 s</td>
<td>5076</td>
<td>553</td>
<td>60</td>
</tr>
<tr>
<td>SHA (Optimized)</td>
<td>12 s</td>
<td>38012</td>
<td>7301</td>
<td>163</td>
</tr>
</tbody>
</table>

Comparing code generation time, Lines of Code, canonicalized boolexp and FSM state number for different models.
Thanks for your attention.