Simulink2BIP Tool

G. Tsiligianis, V. Sfyrla, M. Bozga

VERIMAG, 26/5/2010
Outline

• Introduction

• Description of the Simulink2BIP tool

• Simulink library in Synchronous BIP

• An example

• Experimental Results

• Ongoing and Future work
MATLAB/Simulink is a commercial tool for modeling and simulating continuous/discrete systems.

In this work we study only the discrete-time fragment of Simulink.
A discrete Simulink model consists of:

- **Atomic blocks**: e.g. combinatorial, sources, sinks, delays, ...
- **Subsystems**, e.g. triggered, enabled, enabled periodic, ...
  - They contain atomic blocks or other subsystems
  - They are executed under conditions

In Simulink models, the execution period (sample time) of each block can be explicitly provided by the modeler or left unspecified.
The integrator example in Simulink

Graphical representation

Textual representation (.mdl)
What is BIP?

BIP is a *layered component based* framework

- **Behavior** - atomic components specified by automata or Petri Nets
- **Interactions** - set of ports of atomic components
- **Priorities** - selection amongst possible interactions
Synchronous BIP (SynBIP) is a subset of BIP for modeling synchronous systems

- The behavior of A, B and C is described by modal flow graphs
  - MFGs consist of ports and causal dependencies between ports
- In previous work we provided translation of Lustre into SynBIP
- We actually have code generator for Synchronous BIP in C (single loop code, no engine needed)
A Multi-period Clock Generator in Synchronous BIP

The possible executions are:

- tick
- tick act2
- tick act2 act4
- tick act3
- tick act2 act4 act3
- tick act2 act4 act3 reset
- tick act3 act2 act4 reset

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>Condition</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>q</td>
<td>strong</td>
<td>q must follow p</td>
</tr>
<tr>
<td>p</td>
<td>q</td>
<td>weak</td>
<td>q may follow p</td>
</tr>
<tr>
<td>p</td>
<td>q</td>
<td>conditional</td>
<td>q never precedes p</td>
</tr>
</tbody>
</table>
Simulink2BIP Tool

• Introduction

• Description of the Simulink2BIP tool

• Simulink library in Synchronous BIP

• An example

• Experimental Results

• Ongoing and Future work
Translation from Simulink to Synchronous BIP

Each Simulink block is associated to a unique synchronous component.

- Basic Simulink blocks are translated to elementary components
- Simulink subsystems are translated recursively as compositions of the components associated to their contained blocks

The translation is structural:

- Dataflow and activation links used within the subsystem are translated to connectors

Synchronous components obtained by translation enjoy several structural properties

- well-triggered, confluent and deadlock-free, …
General tool architecture

Simulink Model (.mdl) → Simulink2Bip → Synchronous BIP program

Synchronous BIP program includes simulinkLib.bip

BIP2C → C code → g++ → Executable
Simulink2bip translation
Tool schema

Simulink Model (.mdl)

Parser

Model (AST)

Model Transformation

Translation

clock generation

Synchronous BIP program

3503 lines of Java code
498 lines of grammar
76 grammar rules
37 Java classes
Model Structure

Parser

Block Array
- block1
- block2
- block3
- subsystem1
- block4
- ...

Line Array
- src: block1, dst: block2
- src: block4, dst: subsystem1
- src: subsystem1, dst: block5, dst: block1
- ...

Parser
Model Transformations

Model transformation is an intermediate operation between the Parser and the Translator.

- It transforms some of the structures of the model produced by the parser into structures than can be handled by the translator.

- Transformations are performed according to the translation theory

These transformations concern:

- several atomic blocks, e.g. unit delay, zero order hold, etc, and the specifying of their functionality in synchronous BIP.

- subsystems, i.e. apply rules of the translation

- activation events of the blocks in order to be synchronized.
Model Transformations

This is an example of the transformations, showing the subsystems initialization.
The translation of the transformed model is done as following:

- Blocks are translated into synchronous atomic components
- Lines are translated into data flow connections, e.g. $y_{io} = \{C.out, G.in\}$
- Blocks are synchronized with control event connections, e.g. $y_{act} = \{C.act, G.act\}$
The top level of the Simulink model is represented by the following compound component in Synchronous BIP.

The compound component has:

- **control events** \{act_1, act_2, ..., act_n\} and
- **data events** \{in_1, ..., in_i, out_1, ..., out_j\}.

The produced control events, which correspond to running time periods, need to be triggered by a clock generator.
Clock Generation

The clock generator produces control events corresponding to different sample times.
• *tic* is the **global clock** of the model. This port is triggered every synchronous step, and increases the value of *c* by one time unit.
• Each **clock event** *act*$_{ti}$ is produced when the counter value *c* is equal to the clock period *ti*.
• When the value of the global clock is equal to the least common multiple of the periods *ti* of all the clocks, it **resets** *c*.

```
tic

unit= gcd(t1,t2,…,tn)
c=c+unit

tic

[t1| c] act$_{t1}$

[t2| c] act$_{t2}$

... act$_{tk}$

[tk| c]

reset

[c = lcm(t1,t2,…,tn)]
c = 0

act$_{t1}$     act$_{t2}$     ...     act$_{tn}$

[c := 0]
```
Simulink2BIP Tool

- Introduction
- Description of the Simulink2BIP tool
- Simulink library in Synchronous BIP
- An example
- Experimental Results
- Ongoing and Future work
Simulink Library in Synchronous BIP

The library contains
- 54 atomic components and
- 5 connectors
- 1739 lines of code
Each Simulink block corresponds to an atomic component

Example of the gain atomic component

```plaintext
modal type Gain(float g)
data float x1
data float y
data float g1=g

export port ActPort act=act
export port FloatPort in1(x1)=in1
export port FloatPort out(y)=out

port ActPort gain
on act
on in1<=act
on gain<=in1
do { y = g1*x1; }
on out<=gain
end
```
Simulink Library in Synchronous BIP

Connectors are of two types:
- data connectors, $\gamma = \{ \text{in}, \text{out} \}$
- control connectors, $\gamma = \{ \text{acti} \}$, $i=1,..,n$

Example of a data connector
Simulink2BIP Tool

- Introduction
- Description of the Simulink2BIP tool
- Simulink library in Synchronous BIP
- An example
- Experimental Results
- Ongoing and Future work
Integrator example in Synchronous BIP

compound type Integrator

component simulinkLib.FloatSumSum Add
component simulinkLib.FloatConstant Constant1(1)
component simulinkLib.UnitDelaySimpleRes UnitDelay1(0.0)
component simulinkLib.OutPort eOut(6.0)

connector simulinkLib.conn Constant1_Add(Constant1.out, Add.in1)
connector simulinkLib.conn Add_UnitDelay1(Add.out, UnitDelay1.in1)
connector simulinkLib.conn Add_eOut(Add_UnitDelay1.cout, eOut.in1)
connector simulinkLib.conn UnitDelay1_Add(UnitDelay1.out, Add.in2)

connector simulinkLib.act4 actConn1_1(Constant1.act, Add.act, UnitDelay1.act, eOut.act)

export port simulinkLib.FloatPort out1 is eOut.out
export port simulinkLib.ActPort act0 is actConn1_1.act
export port simulinkLib.ActPort act1 is UnitDelay1.res
end
The enabled subsystem example by Mathworks

sample time of the model: 0.01
The enabled subsystem example by Mathworks

Pulse Generator

Triggered subsystem output

Enabled subsystem output
In the top level system there are:

- Subsystems
  - one enabled integrator
  - one triggered integrator
- Atomic blocks
  - a pulse generator and a scope

The two subsystems contain a discrete integrator which is executed according to some condition.
Translating the Top level system

```plaintext
compound type enable_subsystem

component simulinkLib.PulseGenerator DiscretePulseGenerator(1.0,20.0,40.0)

component Enabled_Integrator_sub Enabled_Integrator
component simulinkLib.FloatSink3 Scope1

component Triggered_Integrator_sub Triggered_Integrator
component simulinkLib.PeriodTransitionRes Enabled_Integrator_STA_out1_Scope1(6.0)
component simulinkLib.PeriodTransitionRes Triggered_Integrator_STA_out1_Scope1(0.0)
component simulinkLib.TriggerRise Trigger
component simulinkLib.EnablePortRes Enable

connector simulinkLib.conn DiscretePulseGenerator_Enable(DiscretePulseGenerator.out,Enable.in1)
connector simulinkLib.conn DiscretePulseGenerator_Scope1(DiscretePulseGenerator_Enable.cout,Scope1.in3)
connector simulinkLib.conn DiscretePulseGenerator_Trigger(DiscretePulseGenerator_Scope1.cout,Trigger.in1)
connector simulinkLib.conn Enabled_Integrator_STA_out1_Scope1_Scope1(Enabled_Integrator_STA_out1_Scope1.out,Scope1.in2)
connector simulinkLib.conn Enabled_Integrator_out1_Enabled_Integrator_STA_out1_Scope1(Enabled_Integrator.out1,Enabled_Integrator_STA_out1_Scope1.in1)
connector simulinkLib.conn Triggered_Integrator_STA_out1_Scope1_Scope1(Triggered_Integrator_STA_out1_Scope1.out,Scope1.in1)
connector simulinkLib.conn Triggered_Integrator_out1_Triggered_Integrator_STA_out1_Scope1(Triggered_Integrator.out1,Triggered_Integrator_STA_out1_Scope1.in1)

connector simulinkLib.act2 actConn1_1(Enable.enab0, Enabled_Integrator.act0)
connector simulinkLib.act2 actConn1_2(actConn1_1, Enabled_Integrator_STA_out1_Scope1.acti)
connector simulinkLib.act2 actConn3_1(Enable.res, Enabled_Integrator.act1)
connector simulinkLib.act2 actConn3_2(actConn3_1, Enabled_Integrator_STA_out1_Scope1.res)
connector simulinkLib.act2 actConn4_1(Trigger.trig, Triggered_Integrator.act0)
connector simulinkLib.act2 actConn4_2(actConn4_1, Triggered_Integrator_STA_out1_Scope1.acti)
connector simulinkLib.act4 actConn5_1(DiscretePulseGenerator.act, Enable.act, Scope1.act, Trigger.act)
connector simulinkLib.act2 actConn5_2(actConn5_1, Enabled_Integrator_STA_out1_Scope1.acto)
connector simulinkLib.act2 actConn5_3(actConn5_2, Triggered_Integrator_STA_out1_Scope1.acto)
connector simulinkLib.act2 actConn10_1(actConn4_2, Triggered_Integrator_STA_out1_Scope1.res)

export port simulinkLib.ActPort act0 is Enable.act0
export port simulinkLib.ActPort act1 is actConn5_3.act

end
```
Simulink2BIP Tool

• Introduction

• Description of the Simulink2BIP tool

• Simulink library in Synchronous BIP

• An example

• Experimental Results

• Ongoing and Future work
Comparing Performances

We compare the running time of the C code generated from Synchronous BIP (using BIP2C) and Simulink (using Real Time Workshop) for N iterations.
## Experimental Results

<table>
<thead>
<tr>
<th>FILE NAME</th>
<th>ITERATIONS</th>
<th>SIMULINK</th>
<th>MFG</th>
<th>ATOMIC</th>
<th>TRIGGERED</th>
<th>PERIODIC</th>
<th>ENABLED</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit counter</td>
<td>1,000,000</td>
<td>1.223s</td>
<td>0.565s</td>
<td>97</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-/-</td>
<td>10,000,000</td>
<td>11.744s</td>
<td>5.644s</td>
<td>97</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>64-bit counter</td>
<td>1,000,000</td>
<td>5.347s</td>
<td>3.115s</td>
<td>365</td>
<td>60</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-/-</td>
<td>10,000,000</td>
<td>53.652s</td>
<td>31.112s</td>
<td>365</td>
<td>60</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Big ABC</td>
<td>1,000,000</td>
<td>0.359s</td>
<td>0.239s</td>
<td>23</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>-/-</td>
<td>10,000,000</td>
<td>3.105s</td>
<td>2.024s</td>
<td>23</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Anti-lock breaking</td>
<td>1,000,000</td>
<td>0.345s</td>
<td>1.394s</td>
<td>39</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>-/-</td>
<td>10,000,000</td>
<td>3.200s</td>
<td>13.515s</td>
<td>39</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Steering Wheel</td>
<td>1,000,000</td>
<td>0.406s</td>
<td>1.676s</td>
<td>120</td>
<td>1</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>-/-</td>
<td>10,000,000</td>
<td>3.417s</td>
<td>16.755s</td>
<td>120</td>
<td>1</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>Multi period</td>
<td>1,000,000</td>
<td>0.465s</td>
<td>0.411s</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-/-</td>
<td>10,000,000</td>
<td>4.012s</td>
<td>3.685s</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Enabled Subsystem`</td>
<td>1,000,000</td>
<td>0.382s</td>
<td>0.380s</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>-/-</td>
<td>10,000,000</td>
<td>3.201s</td>
<td>3.458s</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Thermal model house</td>
<td>1,000,000</td>
<td>0.559s</td>
<td>0.853s</td>
<td>45</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>-/-</td>
<td>10,000,000</td>
<td>5.196s</td>
<td>9.624s</td>
<td>45</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
Simulink2BIP Tool

• Introduction

• Description of the Simulink2BIP tool

• Simulink library in Synchronous BIP

• An example

• Experimental Results

• Ongoing and Future work
Ongoing and Future work

Limitations for the tool

• naming of the blocks must follow certain rules
• sample time of the blocks must be defined in the model
• user defined functions or variables are not supported

Current work involves:

• Translation of the Clutch Model of MATLAB
• Model of a discrete integrator in Synchronous BIP containing the state port
• Complete the documentation of the tool
• Validate correctness of the tool using Dfinder.

Future work will concern:

• Automatic generation of several blocks e.g. sum block, etc.
• Finalizing of clock generator.
• Translate user defined functions from Simulink models
• Translation of Nuclear Reactor model