Toolset for SystemC code generation

Intel Labs, St. Petersburg

Pavel Ivanov
Evgeny Gavrin

Agenda

1. Introduction
2. Technical description
3. Solution
4. Conclusion

Introduction

Introduction

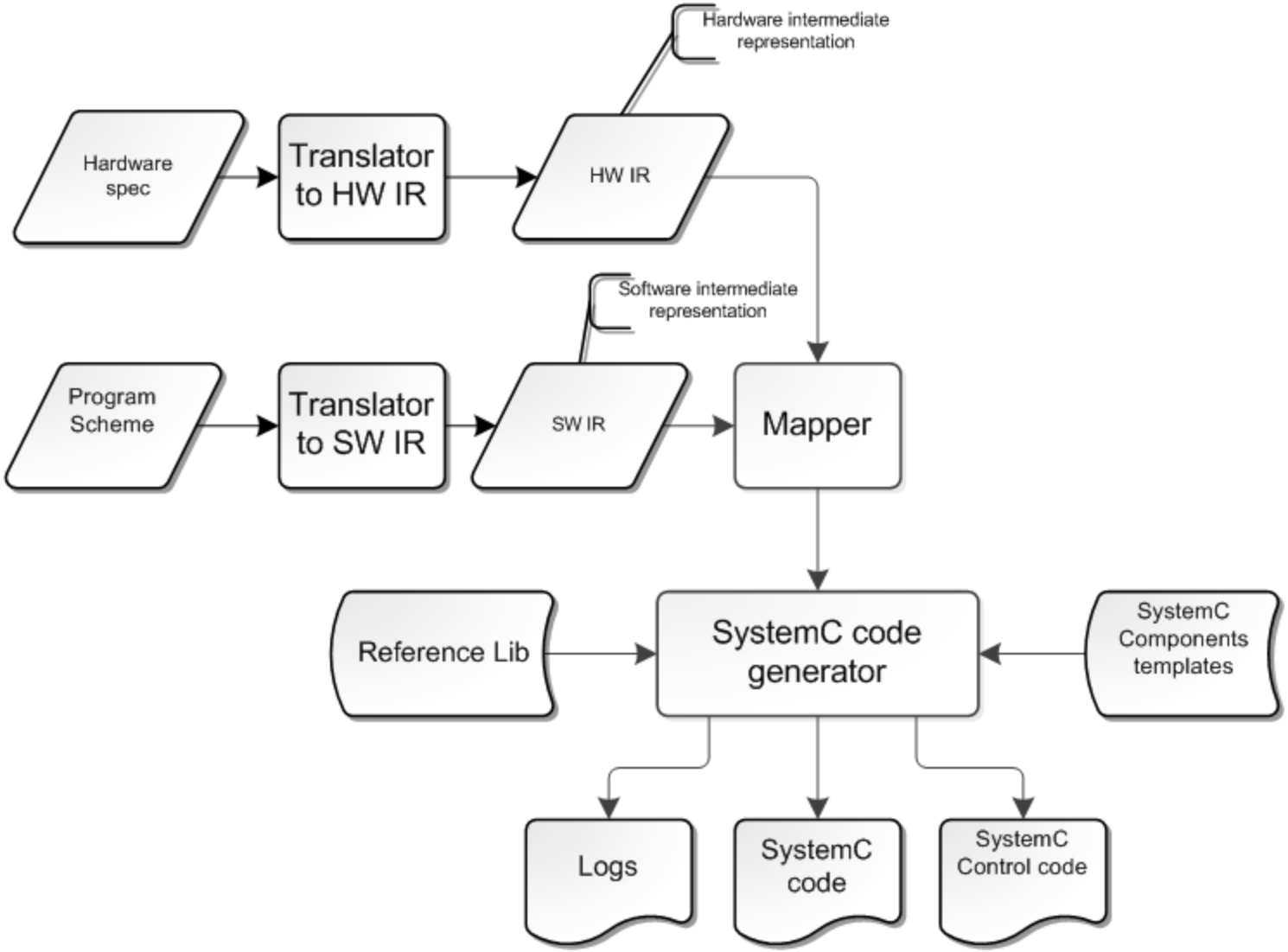
Systems-on-Chip development includes:

1. The conceptual design of the system;
2. **Development of a system level model (SystemC / System Verilog);**
3. Development of RTL (Verilog / VHDL);
4. Hardware verification;
5. Physical prototyping;
6. Embedded software development;
7. Software testing;
8. Testing of entire system.

Solution: Automatically code generation of a system level model

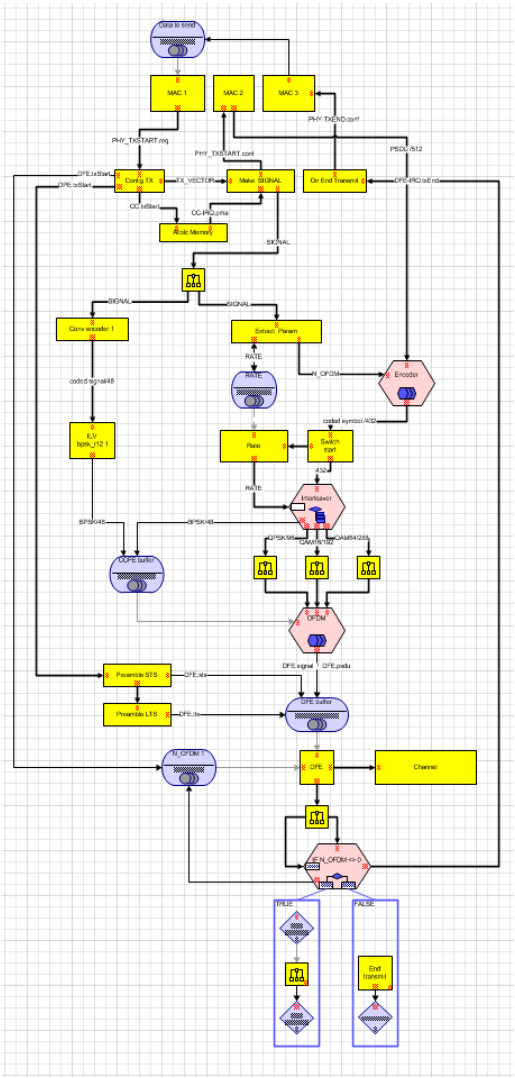
Technical description

Design tool flow



Solution

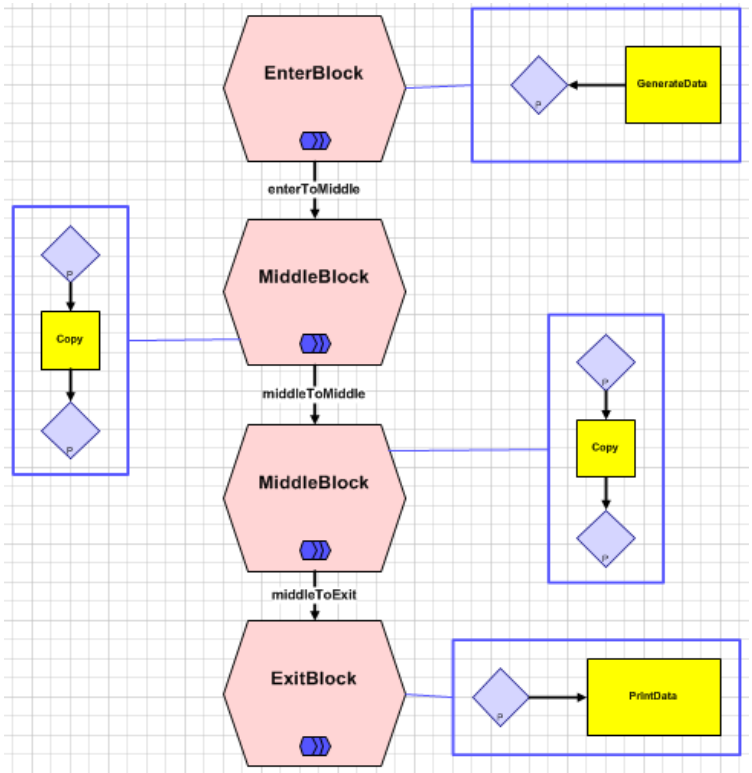
Program scheme: visual notation



- Microsoft Visio© add-in
- Customizable basic blocks library
- C-code behind the basic blocks
- Verification and auto-correction on design stage

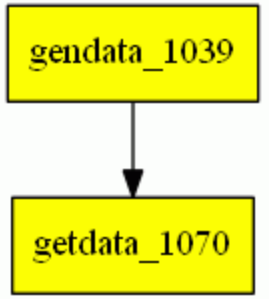
Program scheme. Concept of textual language

- Pattern-based declarative language
- Every language construct has MoC representation
- Recursion is prohibited




```
1  import "general/atomic"
2
3  let main ()
4  {
5      var enterToMiddle;
6      var middleToMiddle;
7      var middleToExit;
8
9      EnterBlock(enterToMiddle);
10     MiddleBlock(enterToMiddle, middleToMiddle);
11     MiddleBlock(middleToMiddle, middleToExit);
12     ExitBlock(middleToExit);
13
14 }
15
16 let EnterBlock (out int outData)
17 {
18     gendata(outData);
19 }
```

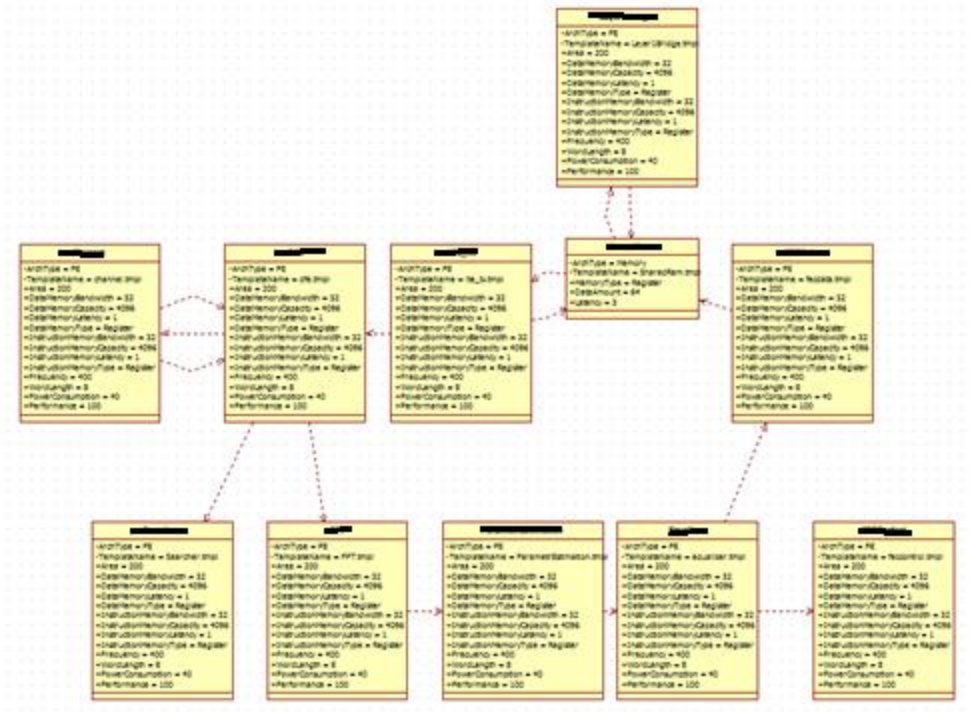
Software intermediate representation



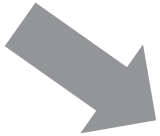

```
import "general/atomic"

let main ()
{
    var enterToMiddle;
    var middleToMiddle;
    var middleToExit;

    EnterBlock(enterToMiddle);
    MiddleBlock(enterToMiddle, middleToMiddle);
    MiddleBlock(middleToMiddle, middleToExit);
    ExitBlock(middleToExit);

}

let EnterBlock (out int outData)
{
    gendata(outData);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<program id="i1078" comment="" mainSection="i1079">
  <group subtype="section" comment="main" id="i1079">
    <nodes>
      <terminal subtype="functional" comment="gendata"
      id="i1080" tag="" execCost="100" isPermanent=
      "false">
        <port portNum="0" subtype="simple" id="i1081"
        comment="outData" tag="" link="i1084" />
      </terminal>
      <terminal subtype="functional" comment="getdata"
      id="i1082" tag="" execCost="100" isPermanent=
      "false">
        <port portNum="0" subtype="simple" id="i1083"
        comment="inputData" tag="" link="i1084" />
      </terminal>
    </nodes>
    <links>
      <link source="i1080" sourcePort="0" target="i1082"
       targetPort="0" comment="" id="i1084" tag=""
      isBackLink="False" subtype="read-erase"
      defDataAmount="1" />
    </links>
  </group>
</program>
```

Hardware intermediate representation



- UML Advantages:
  - Huge set of tools
  - Easy to describe



```
<UML:Attribute xmi.id="UMLAttribute.5" name="TemplateName" visibi
  <UML:Attribute.initialValue>
    <UML:Expression xmi.id="X.221" body="pe.tmpl"/>
  </UML:Attribute.initialValue>
</UML:Attribute>
<UML:Attribute xmi.id="UMLAttribute.6" name="Area" visibility="pu
  <UML:Attribute.initialValue>
    <UML:Expression xmi.id="X.223" body="200"/>
  </UML:Attribute.initialValue>
</UML:Attribute>
<UML:Attribute xmi.id="UMLAttribute.7" name="DataMemoryBandwidth"
  <UML:Attribute.initialValue>
    <UML:Expression xmi.id="X.225" body="32"/>
  </UML:Attribute.initialValue>
</UML:Attribute>
<UML:Attribute xmi.id="UMLAttribute.8" name="DataMemoryCapacity"
  <UML:Attribute.initialValue>
    <UML:Expression xmi.id="X.227" body="4096"/>
  </UML:Attribute.initialValue>
</UML:Attribute>
```

- Component properties includes:
  - Work frequency
  - Local memory capacity
  - Buffers types

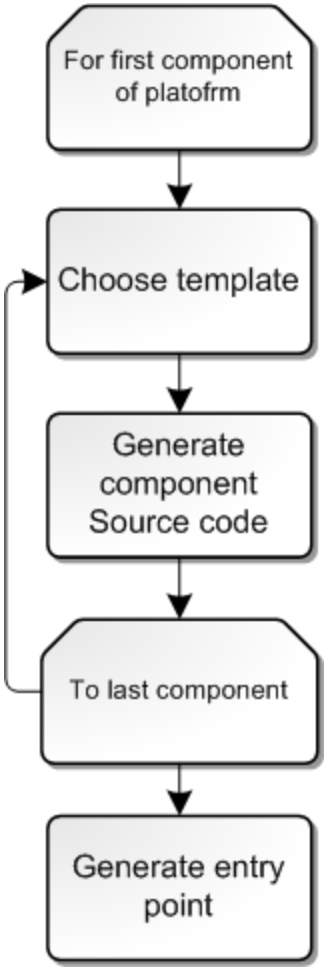
Mapping

- Different scheduling strategies
  - ASAP
  - ALAP
  - Pipeline
- Allocation
  - Simple load balancing algorithm
- Binding
- Control code generation

SystemC platform code generator

- Template-based code generator
- Basic (reconfigurable by tags) components :
  - Control Unit
  - Processing element
  - Interconnect (Bus, Switch)
  - Memory
- Log service:
  - Console log
  - Text log
  - Waveforms

Code generation flow. Template example




```
SC_MODULE(MODULE_%mod_name%)
{
        %ports_declaration%

        %behavioral%

        SC_CTOR(MODULE_%mod_name%)
        {
                %threads_declaration%
        };

};
```

Example of processing element file template

- The code generator, generate code for each component on element-by-element basys, during execution
- Tagging make code generation more simple

SystemC test bench code generator

- SystemC test bench generator has two scenarios:
  - Testing each component separately
  - Testing of the entire system (WIP)
- Generate set of test vectors
- Provide log service
- Test vectors have the following scenarios:
  - Testing of processing element or memory
  - Testing of interconnects
- Checking of transmitted data correctness and data delivery time
- Checking combinations of control signals
- Checking overflows

Conclusion

Conclusion

- Proposed approach makes development of system level model a more easily
- Saves time
- Does not require knowledge of SystemC

Thank You