Pythonized SystemC
A non-intrusive scripting approach

Eyck Jentzsch, MINRES Technologies GmbH
Rocco Jonack, MINRES Technologies GmbH
Introduction

• MINRES focuses on VP and architectural modeling
  – Providing support in various forms
• Development often in parallel with HW development
  – what if scenarios are important for architecture decisions
  – Platform definition is not fixed
• VP based embedded software development for large systems requires the use of partial and subsystems to get reasonable simulation speed and runtime
• Flexibility in Reconfiguration is key for efficient model development
Addressing Flexibility

• Complex configurations system
  – Reading and interpreting a configuration file
  – Done in several tools by parsing XML or JSON files

• Code generation
  – Based on some configuration input generated glue logic

• Scripting languages as frontend
  – There are tools which provide such solutions
  – Allows integration of different functionality
  – Limited by scripting API
Scripting Solutions for SystemC

• There are several existing integration into scripting languages
• As part of commercial tools based on TCL/TK, Python
• Open-source solutions
  – SoC ROCKETS Universal Scripting Interface (USI)
  – GreenSoCs GreenScript
  – SystemPy
  – Kosim
SystemC and Python

• We opted for an interpretation "frontend" based on Python
• Python is well-known and existing libraries can be reused
• Besides support for structural construction, simulation control and dynamic model parametrization can be supported
• Existing Python integrations require preparation work
  – Definitions of API into libraries which have been compiled
  – Quite often modification of the libraries to fulfill requirements implied by the interpreter
• Therefore there are no integrations for SCV or CCI available
PySysC

• CERN developed several tools for the analysis of LHC generated data
  – CINT: home-grown Python bindings piggy-backed on C++ reflection for serialization and interactivity
  – CLING: C++ interpreter (https://root.cern/cling)
  – PyPy/CPPYY: Cling-based Python-C++ bindings
• Cppyy can be leveraged for any library
• This is the basis for the PySysC module
PySysC Advantages

- No preparation of libraries to be integrated
- No need to have the sources of the code, even 3\textsuperscript{rd} party binary only libraries can be used
- Allows introspection of the interfaces and thus dynamic generation
- If Python is not sufficient JIT allows to compile on-the-fly generated C++ code
PySysC Example

1. Instantiation of a module
2. Instantiation of a templated module
3. Named signal connection
4. TLM2.0 socket connection
5. Simulation run

```python
from cppy import gbl as cpp
from cppy.gbl import sc_core
from pysysc.structural import Connection, Signal, Module, Simulation
# loading required libraries
...
# instantiating modules
clk_gen = Module(cpp.ClkGen).create("clk_gen") ## (1)
initiator = Module(cpp.Initiator).create("initiator")
memories = [Module(cpp.Memory).create(name) for name in ["mem0", "mem1", "mem2", "mem3"]]
router = Module(cpp.Router[4]).create("router") ## (2)
# creating connections
clk = Signal("clk")
    .src(clk_gen.clk_o)
    .sink(initiator.clk_i)
    .sink(router.clk_i) ## (3)
[clk.sink(m.clk_i) for m in memories]
Connection()
    .src(initiator.socket)
    .sink(router.target_socket) ## (4)
[Connection()]
    .src(router.initiator_socket.at(idx))
    .sink(m.socket)
    for idx,m in enumerate(memories)]
# run simulation
sc_core.sc_start() ## (5)
```
Advantages of Python Usage

• Due to broad availability of Python integrations plenty of libraries can be used and combined
  – Computational models using numpy/scipy etc.
  – UIs and cockpits using GTK, wxWidgets or Qt
Evaluation and Results

- Table 1 compares a simple design in SystemC and PySysC in terms of runtime and LoC.
- Table 2 uses a RISC-V based VP in different scenarios using plain SystemC and PySysC based structural description.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>SystemC</th>
<th>PySysC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>build</td>
<td>run</td>
</tr>
<tr>
<td>hello world</td>
<td>15s</td>
<td>8.7s</td>
</tr>
<tr>
<td>dhrystone</td>
<td>15s</td>
<td>120s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scenario</th>
<th>SystemC</th>
<th>PySysC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>hello world</td>
<td>15s</td>
<td>8.7s</td>
</tr>
<tr>
<td>dhrystone</td>
<td>15s</td>
<td>120s</td>
</tr>
</tbody>
</table>

Table 1

Table 2
Outlook

• PySysC is available as module via git
  – The Python module: https://git.minres.com/VP/PySysC
  – The examples: https://git.minres.com/VP/PySysC-SC

• Development is work in progress

• Will be used as a basic building block of the BMBF funded project “RAVEN: Acceleration of Virtual Hardware/Software Development Platforms by Reconfigurable Logic'
Questions

Finalize slide set with questions slide
Guidelines (1)

• Please keep the default font size for main lines at 28pt (or 26pt)
  – And use 24pt (or 22pt) font size for the sub bullets
• Use the default bullet style and color scheme supplied by this template
• Limited the number of bullets per page.
• Use keywords, not full sentences
• Please do not overlay Accellera or DVCon logo’s
• Check the page numbering
Guidelines (2)

• Your company name and/or logo are only allowed to appear on the title page.
• Minimize the use of product trademarks
• Page setup should follow on-screen-show (4:3)
• Do not use recurring text in headers and/or footers
• Do not use any sound effects
• Disable dynamic slide transitions
• Limit use of animations (not available in PDF export)
Guidelines (3)

• Use clip-art only if it helps to state the point more effectively (no generic clip-art)
• Use contrasting brightness levels, e.g., light-on-dark or dark-on-light. Keep the background color white
• Avoid red text or red lines
• Use the MS equation editor or MathType to embed formulas
• Embed pictures in vector format (e.g. Enhanced or Window Metafile format)