Advanced simulation tools on top of SystemC

Finnish-Russian University Cooperation Program in Telecommunications (FRUCT) seminar
Turku, Finland, Nov-07

Michel Gillet and Sergey Balandin

michel.gillet@nokia.com, sergey.balandin@nokia.com

CTC Computing Structure, Architecture Solutions,

Nokia Research Center

Outline

- Embedded networks
- Protocol modeling
- Our experience with SystemC
- Protocol modeling with SystemC
- Beyond SystemC
- Conclusions

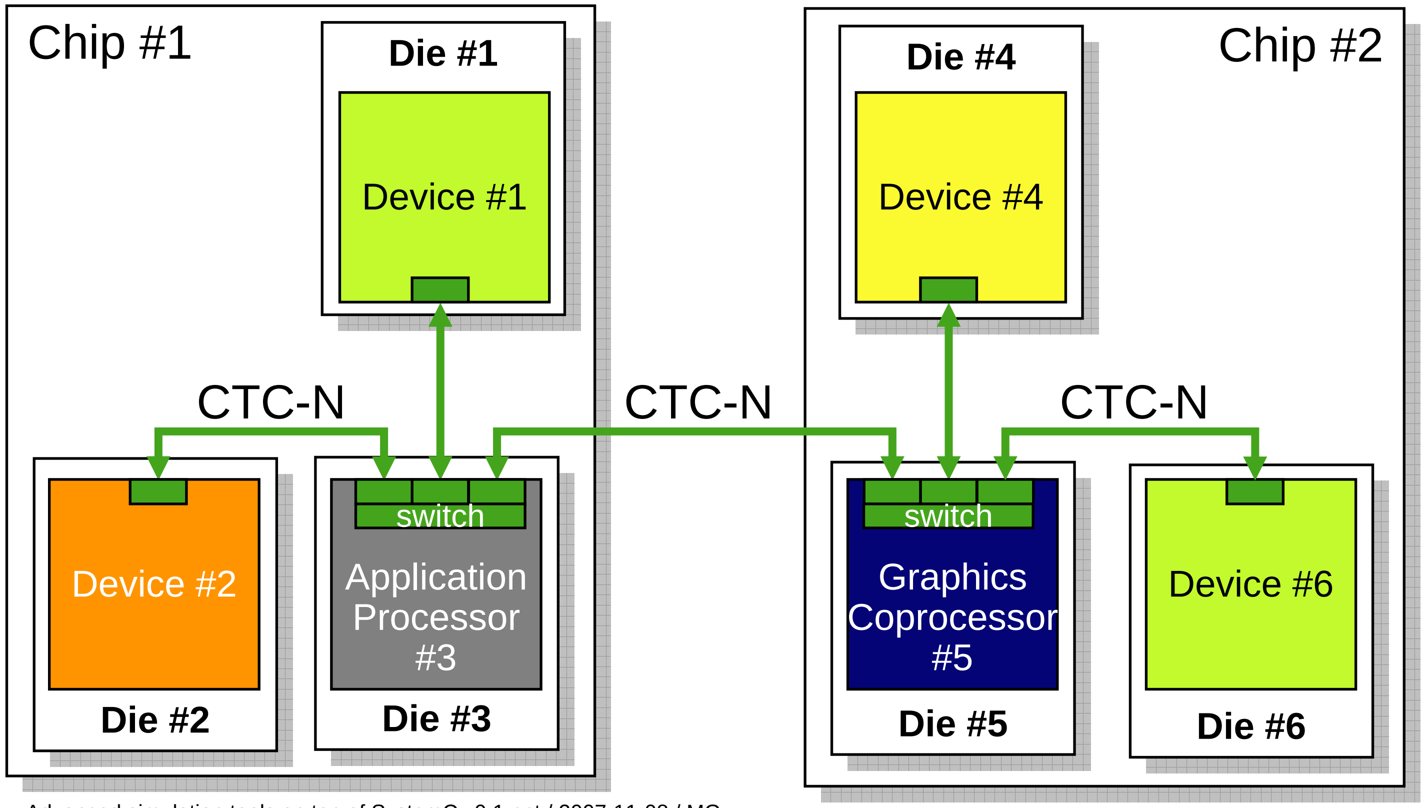
Embedded networks

Remark

- These slides assume prior knowledge of the presentation “Embedded networks” by the same authors shown earlier in this FRUCT seminar

Ultimate goal

- A embedded network technology to simplify integration by abstracting away the chip and die boundaries
  - 4 signal PINS, Gbit/s rates, low power, low EMC/EMI, can be pure hardware



Protocol modeling

Protocol modeling

- Protocols are classically described in layered model based on the OSI stack or any variant
- Since protocols can be very complex, modeling has always been at the very center of protocol design
- When speaking about protocol modeling, there are essentially 3 main approaches or goals :
  - Formal verification
  - Protocol design, exploration of protocol and their features
  - Performance evaluation
- And 2 classes of languages used:
  - Formal languages
  - Programming languages

Formal verification

- Formal verification of protocol is pretty much a science of its own, almost black magic for many
- As its name indicates, the goal of such modeling is to find a formal proof (a mathematical proof) of the correctness of a protocol
- There are many aspects of protocol correctness:
    - Deadlock free
    - Livelock free
    - Behavioral completeness (closure)
    - Etc.
- Of course, this is very important for standardization, but not optimum for protocol engineering and design

Formal verification languages

- This topic has been studied for so many years, that there is quite few different tools for formal verification
- Most of these tools are language based:
  - SDL
  - LOTOS
  - SPIN
  - UUPAAL
  - Etc.
- Their main drawback for protocol engineering in embedded network is their rigidity, their formal nature, which makes them inadequate for performance analysis or “protocol engineering”

Network simulator

- Another approach for protocol modeling is simply to take any programming language and build a so called “network simulator”
- These simulators are of course not suitable for formal verification,
- But tailored made for protocol engineering, performance analysis, comparison of different technical solutions, etc.
- They have extensive libraries of ready made protocols, tools for visualizing results and/or behavior, traffic generators, traffic sinks, often tools for statistical analysis, etc.
- The most famous is of course NS-2

NS-2 and embedded networks

- Embedded networks are by definition and by area of application pretty far from the TCP/IP world
- They are aggressively optimized for low power, low complexity, etc.
- NS-2, being strongly IP oriented, doesn’t fit naturally to embedded networks
- Furthermore, we were interested in technologies supporting QoS very low in the protocol stack, simply for efficiency
- We can’t reuse much of the existing models provided in NS-2 and we would need to create almost from scratch all layers

Programming languages

- The only alternative left for us was to choose a classical programming language, like C or C++, etc.
- The obvious issue is that they don't have any support for protocol modeling, particularly they don't have a notion of time
- Of course, some libraries for network simulation exists, but they are all either only supporting TCP/IP family of protocols, either are more generic but would require to model everything from scratch

SystemC – base platform of our simulators

- SystemC is “free” C++ based library (www.systemc.org)
- Comparing to pure C++, the SystemC library provides a set of C++ classes to represent time, includes a event-driven simulator, etc.
- SystemC has various levels of abstraction
- SystemC was already used internally for evaluation of ASIC architecture, chip design, co-design and co-simulation, etc.
- SystemC has become the de facto standard for system modeling
- Many tool vendors were supporting SystemC and it was really easy to include it in the VHDL design flow
- Conclusion: SystemC is well fitting our needs as the base platform for network simulator covering high abstraction level and at the same time we can be very close to hardware when needed

Our experience with SystemC

SpaceWire & our own solution

- From the requirements for an embedded network solution and after analysis of various existing protocols, SpaceWire was clearly the best match
- So naturally, we build a SpaceWire protocol simulator to evaluate its performance
- This work was essential for us to learn how SystemC could be best used for protocol modeling
- But more important, it showed us clearly the limitations of SystemC for such a purpose
- All what we learned was use for building the protocol of our own proprietary embedded network

Experience and learnings

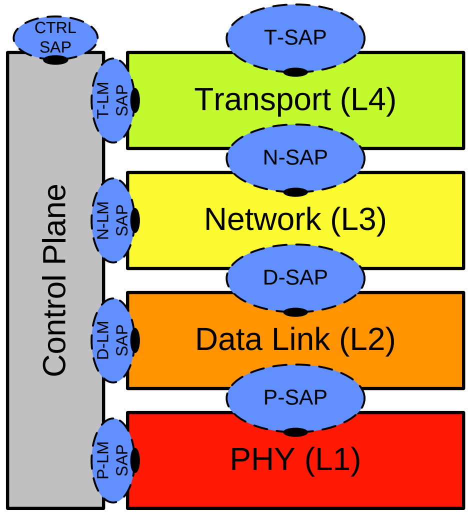
- SystemC modeling of the complete architecture gives unique understanding of the system
  - Especially important for system optimization when power, protocol efficiency and costs are hard constrains
- It is important to invest time in creating modeling test bench
  - The test benches made to test the protocols can be easily reused to test hardware implementation
- Modeling a complete system including software is straightforward and offers very efficient simulations
- This way of working has been adopted and used in multi-company and multi-site environment
  - First in Nokia and later introduced in MIPI UniPro standardization WG

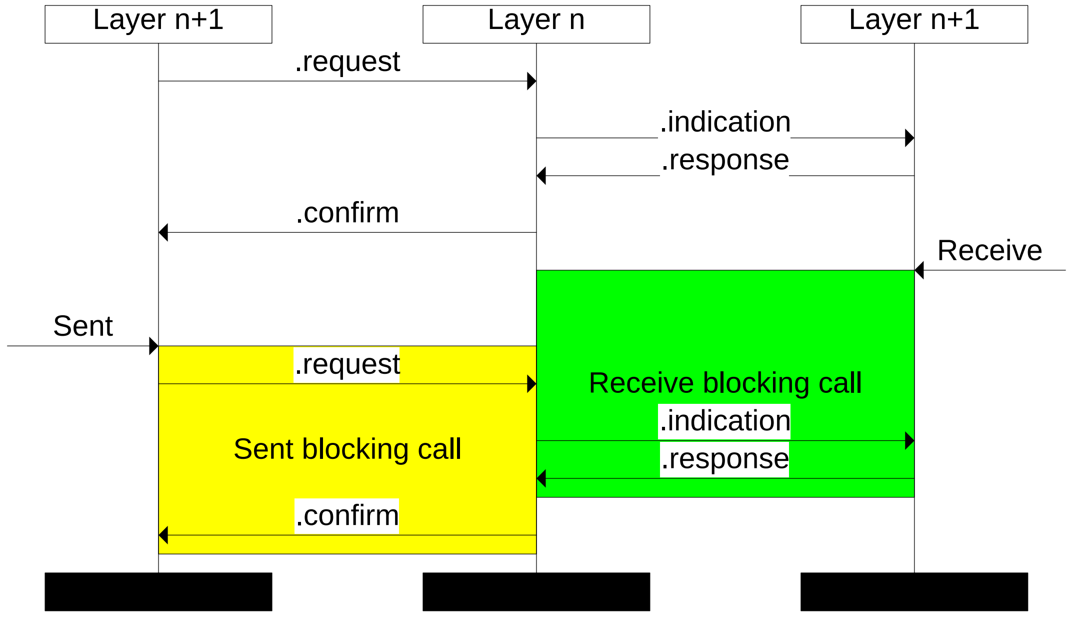
Protocol modeling with SystemC

NOKIA

Layered protocol design

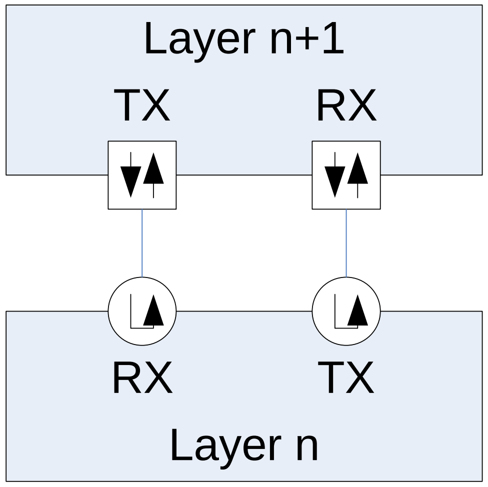
- All embedded networks we designed and are designing follows the same basic layering defined by OSI
- We also use the concept of SAP
- Basic primitives we use
  - Request and Confirm(_L)
  - Indication and Response



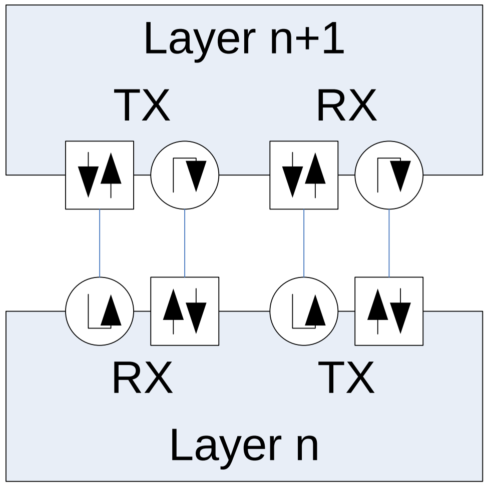


SystemC modeling of SAPs

- We have chosen the “blocking calls” since it reduce the number of interfaces and the number of processes needed by half.



Blocking calls



Split transactions

SystemC limitations (1/2)

- SystemC was designed in a way, which make it virtually easily usable only by experts
- It makes the embedded models created with SystemC usable only to very specialized individuals
- In SystemC, a class called sc_module is used to define the common behavior of every single logical block in a model
- To try to mitigate this issue, we have created a derivate class of sc_module called up_module which allows us
  - To create model factories
  - Change the implicit hierarchical architecture of the SystemC models based on the model names to an explicit hierarchical architecture

SystemC limitations (2/2)

- If all logical block in the models are themselves derivate class from up_module, a complete SystemC embedded network model can be instantiated and configured from a XML file
- It also allows the replacement of many manual tasks caused by SystemC semantic, by automated mechanisms completely transparent to the user of the models
  - Allows replacement of a logic block in a hierarchical-tree model by automatic binding of sc_ports without having to create new classes for all its parent leafs up to the root of the tree
- Enable simpler scripting
  - Python, Perl, etc.

Beyond SystemC

Our modification of SystemC – mySystemC (1/2)

- We started with SystemC 2.0.1 and identified the following issues:
  - No good support for latest GCC compiler
  - No support for shared library under Unix/Linux
  - No support for dynamic link libraries under windows
  - Not as platform independent as we needed
- A first set of modifications were attempted to solve those issues
  - but couldn’t be carried out because some other issues appeared
- The main setbacks were
  - The original SystemC library had some code that made impossible creation of DLLs
  - The source code was not always perfectly C++ OSI compliant

Our modification of SystemC – mySystemC (2/2)

- A new class was added in SystemC to replaces the sc_main
    - sc_main is a macro using the class SystemC
- Use libtool under Unix for building static and shared library
- Too strict usage of private members which hinders customization: changes made to allow different sc_sim_context implementation, interactive scheduler, etc.
- SystemC 2.0.1, 2.1 and 2.2 are supported

- Available soon for download from: http://www.hamletg.org/mambo/index.php?option=com_content&task=view&id=24&Itemid=32

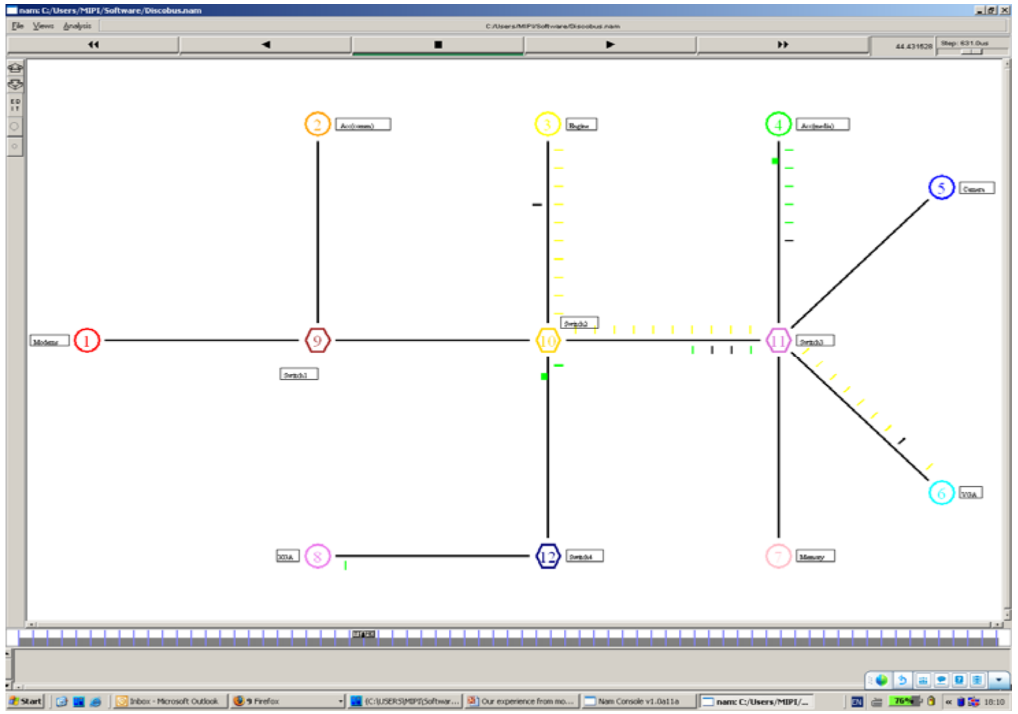
NOKIA

Speeding up development – Python & SystemC

- To simplify and speed up design of test benches for complex system, all the models are scripted in python
- pymySystemC is a python module, which exposes the C++ classes of SystemC through python
- For creating SystemC python module we used SWIG
    - SWIG (http://www.swig.org/) is an open source project
    - SWIG is a software development tool that connects programs written in C and C++ with a variety of high-level programming languages
- SystemC 2.0.1, 2.1 and 2.2 are supported
- Available soon for download from: http://www.hamletg.org/mambo/index.php?option=com_content&task=view&id=19&Itemid=32

Graphical extension – e.g. visualization of results

- Matplotlib
  - http://matplotlib.sourceforge.net/
- Network Animator (NAM)
  - Is distributed as a part of NS-2
  - http://www.isi.edu/nsnam/nam/



- Graphical interface
  - wxWidgets (http://www.wxwidgets.org/)
  - wxPython (http://www.wxpython.org/)

NOKIA

Conclusions

- Even if SystemC was not designed specifically for building network simulators, it provides very good base platform for building embedded network simulators
  - It has been proven by very good experience of using SystemC based simulator for internal and MIPI standardization activities
- Our study has shown that the following is useful for large SystemC models:
  - Add support for DLL and shared library in the official releases
  - Use of explicit hierarchy of sc_module, then creates the sc_module_name hierarchy
  - The current way is troublesome when using model factories, XML description of complete systems to be simulated, etc.
- Use of listed additional open source libraries and tools allow building complete toolset on top of SystemC for embedded network simulator, SoCs, NoCs, etc.
  - For making your platform successful it is extremely important to use tools for speeding up creation of the test scenarios and for graphical presentation of results
  - It is important to invest time in creating modeling test benches

NOKIA