North American SYSTEMC™ User's Group

September 29, 2004

SPACE: SystemC Partitioning of Architectures for Co-design of real-time Embedded systems

Jérome Chevalier[1], Maxime De Nanclas[1], **Guy Bois[1]** and Mostapha Aboulhamid[2]

1. École Polytechnique de Montréal 2. Université de Montréal

ÉCOLE POLYTECHNIQUE MONTRÉAL

CRSNG NSERC

MICRONET R&D

SPACE: SystemC Partitioning of Architectures for Co-design of real-time Embedded systems

Agenda

1. Motivations
2. Objectives
3. Overview of SPACE
4. Levels of abstraction
5. Facilities of exploration through a SystemC API
6. Details of each level
7. SOCP as channel interface model
8. Results
9. Conclusion and future works

1. Motivations

- The SystemC core language does not support thread *priority assignment*
- Its simulator does not offer all the necessary functionalities, such as *preemption* or scheduling by priority, generally present in any RTOS
- Then, *soft real-time* can be supported (i.e. missing a deadline does not cause serious damage)
- But *hard real-time* cannot be supported (i.e. missing a deadline may cause catastrophic consequences)

2. Objectives

- To find Hw/Sw partitions that satisfies the specified real-time constraints, while minimizing hardware resources
- To offer at high level the main services of a real-time kernel of an RTOS
- To propose a Hw/Sw refinement methodology with specific verification issues at each level
- To be able to move user modules from hardware to software part (and vice versa) without the necessity of recoding from a Hw language to C/C++ (and vice versa)

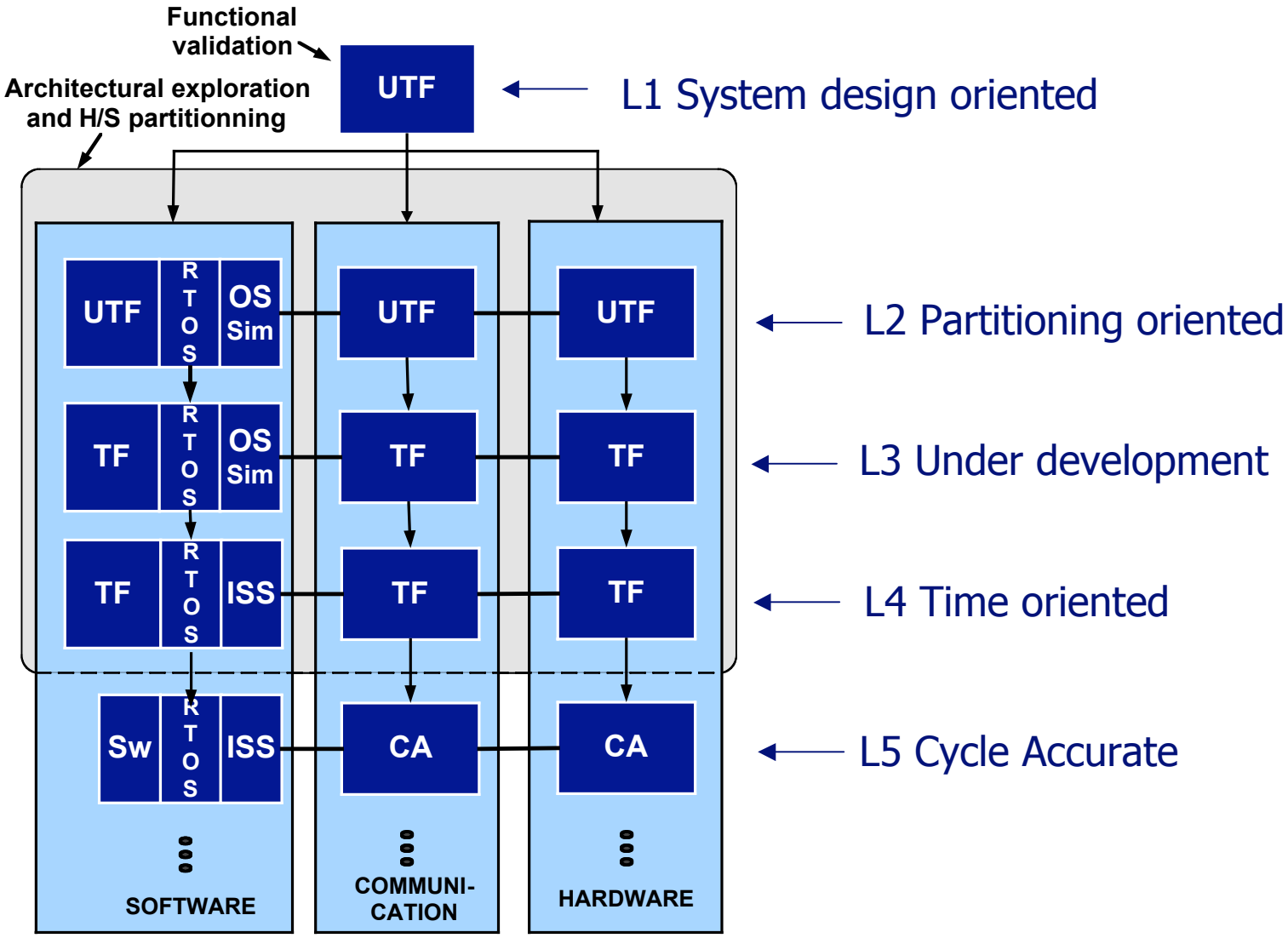
3. Overview of SPACE: 1st step: functional validation

- SPACE stands for *SystemC Partitioning of Architectures for Co-Design of Embedded systems*
- Simulate the system in a purely functional form
  - with a SystemC transactional model named *UTF channel*
- Check that the system's simulation respects the functional aspect of the specification

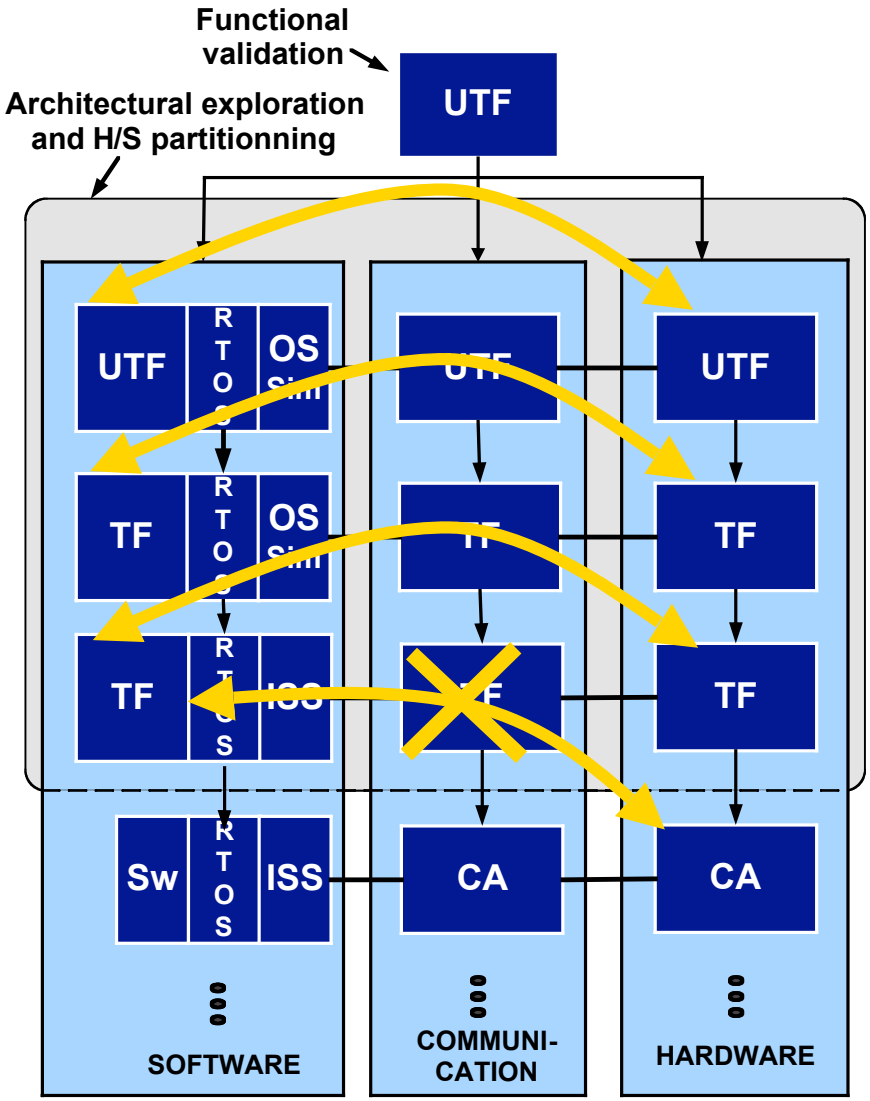
3. Overview of SPACE: 2nd step: partitioning stage

- Modules tested at functional level are taken without modifications
- They are placed in the software or hardware part of the architecture, based on the designer's best guess
  - Simulated at transactional level
  - Refinement through three levels of abstraction
- We iterate guided by the performance and cost metrics

4. Levels of abstraction

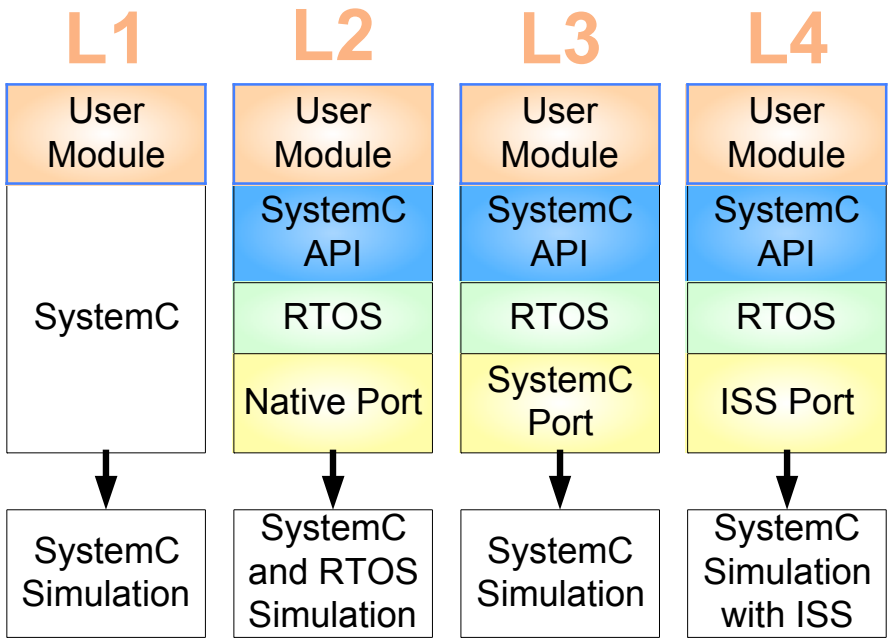


4. Levels of abstraction



At each level, we can easily move SystemC modules from Hw and Sw (and vice versa) and reevaluate our performance and cost metrics

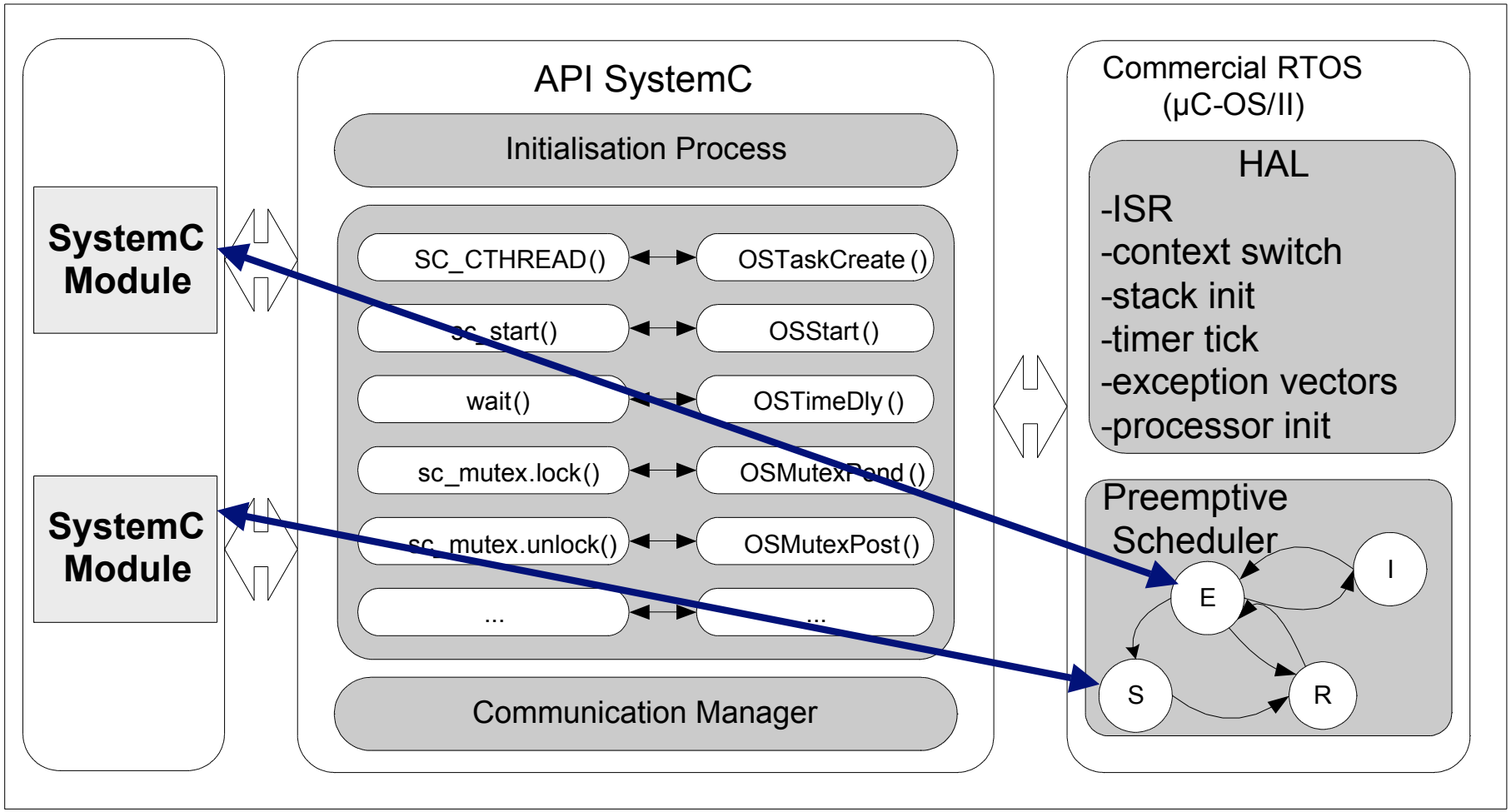
5. Facilities of exploration through a SystemC API



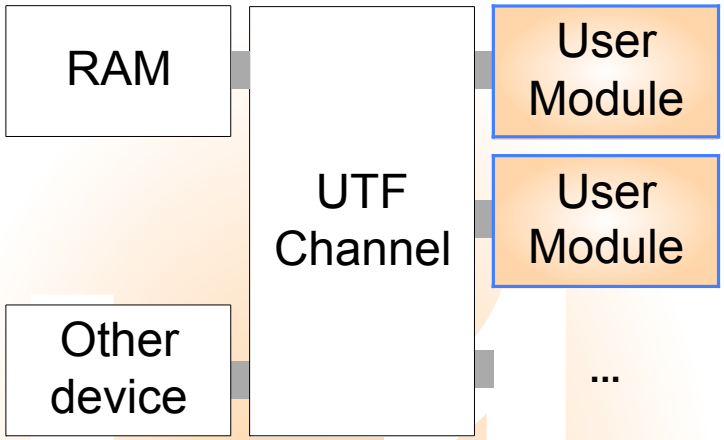
Sw Levels

- **API**
  - Mapping between SystemC 2.0 and RTOS functions
  - Message queues for communication
- **RTOS**
  - Schedules SystemC modules
  - Priorities and preemptive scheduling
- **Port - HAL**
  - H/S communication : handler to catch messages from hardware
  - Provides RTOS task context switch and timer (connection or emulation)

5. Facilities of exploration through a SystemC API

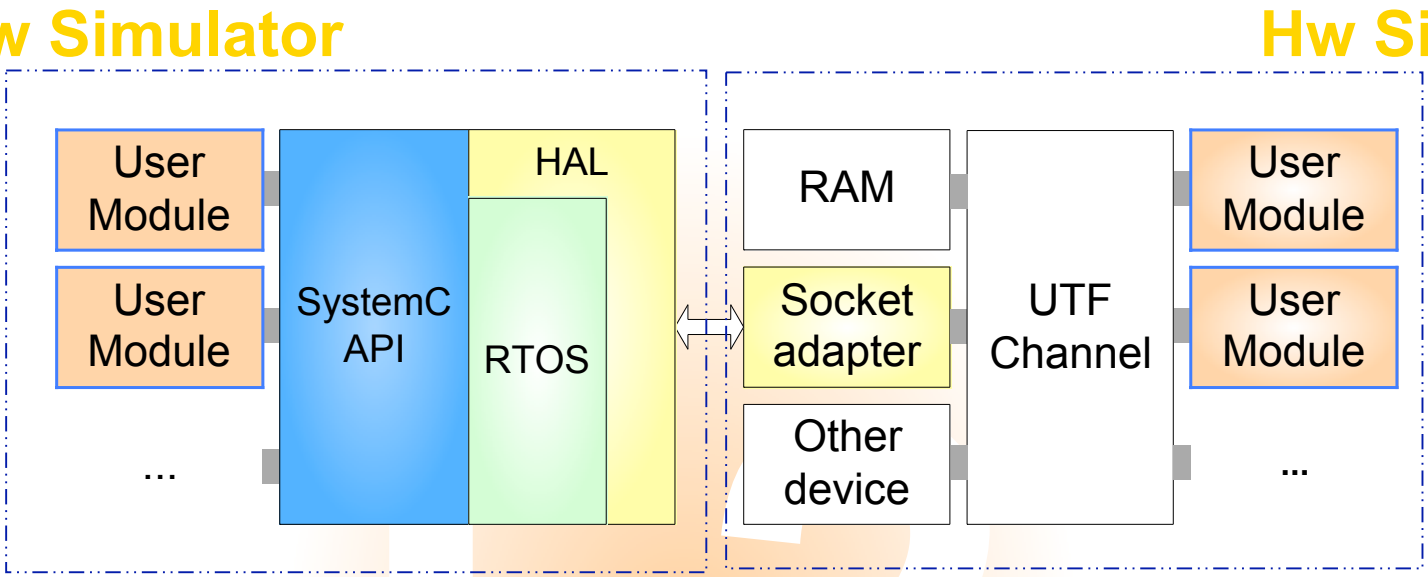


6. Details of each level of abstraction – L1



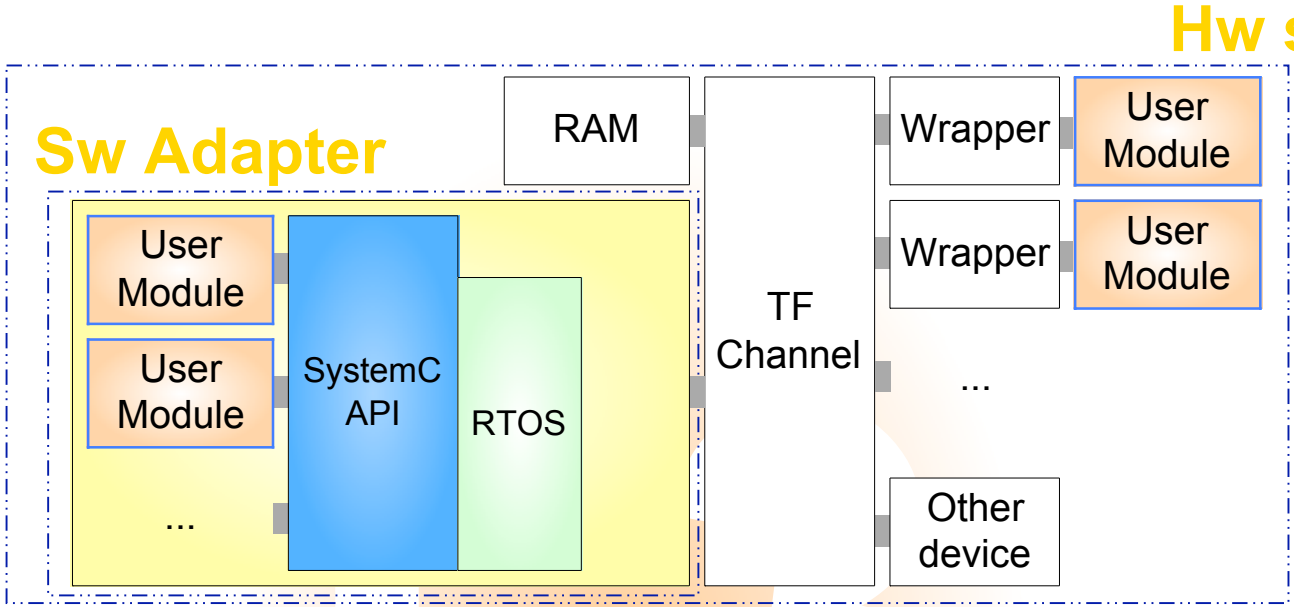
- Functionality capture in SystemC 2.0
- Modules represent user code
- Devices represent memory modules
- Specification and verification of the general idea
- Untimed and very fast simulation

6. Details of each level of abstraction – L2



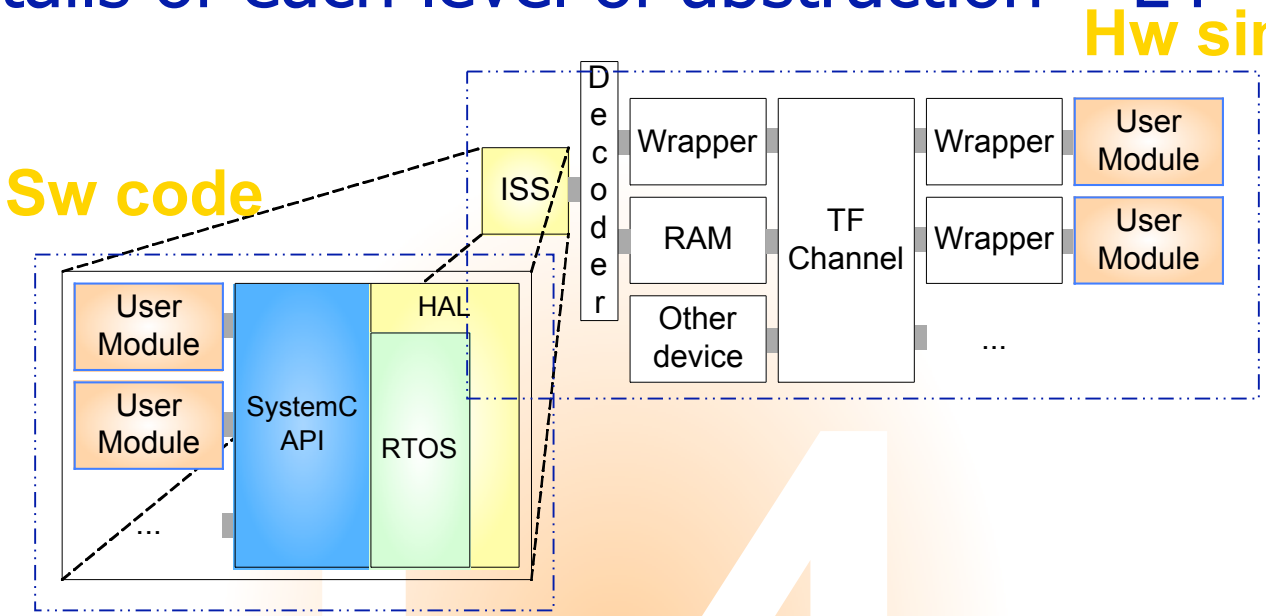
- RTOS selection
- Priority assignment with resource sharing
- Hw/Sw pre-partitioning
- Hardware and software simulation in two different processes
- Synchronization between hardware and software performed by message passing through a socket
- Untimed and fast simulation

6. Details of each level of abstraction – L3



- Timing constraints
- A SystemC module emulates the RTOS HAL
- Hardware and software simulated in the same process
- Both hardware and software advance in a locked step cycle
- Timed (and fast) simulation
- Under development

6. Details of each level of abstraction – L4

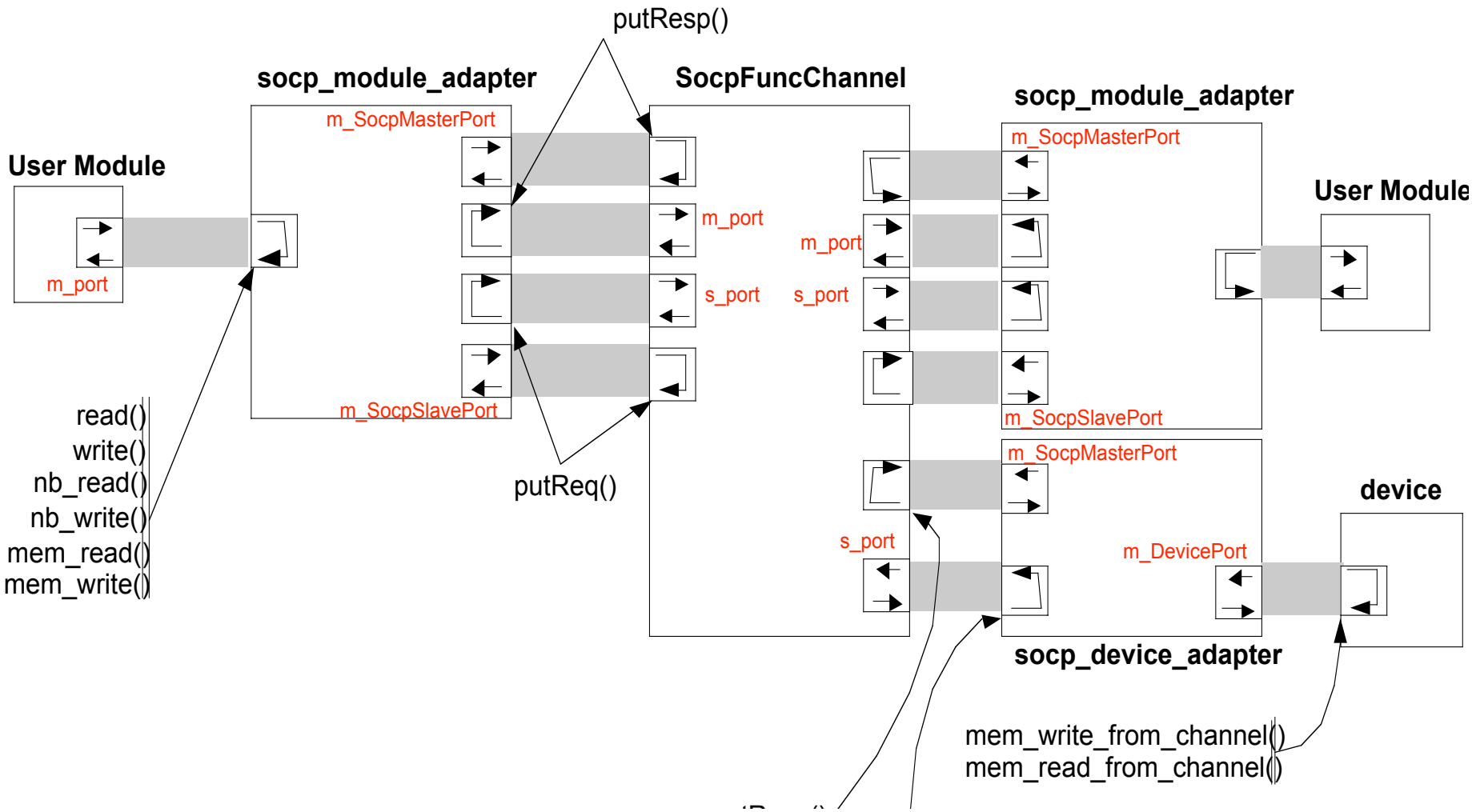


- CPU and bus selection.
- Use of CPU: interrupt latency, etc.
- Final HW/SW partitioning
- ISS with an RTOS for the software
- ISS and hardware modules simulated by SystemC
- Co-debugging with two debuggers
- Cycle true simulation

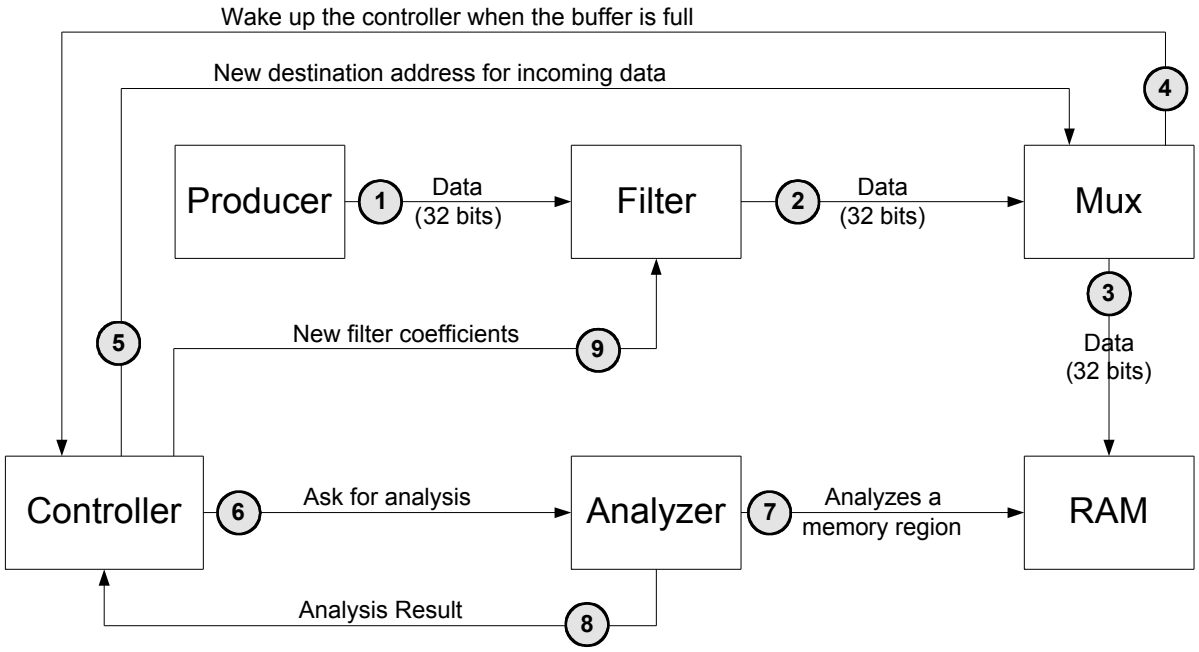
7. SOCP as channel interface model

SOCP
from StepNP
D&T Dec. 2002

7. SOCP as channel interface model

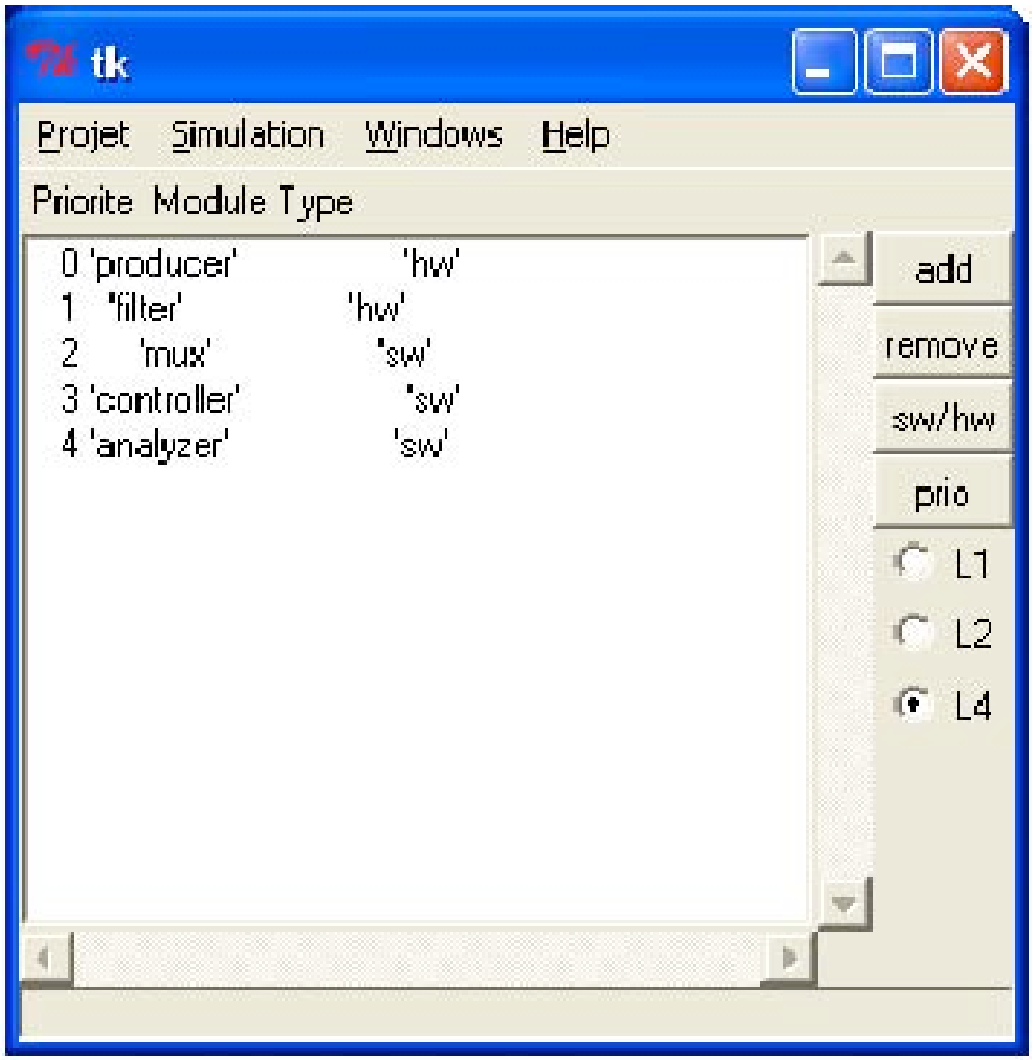


7. Results: Audio Filter



1. The Producer generates audio data with a rate of 44.1 kHz.
2. Filtered data is sent to the Mux.
3. The Mux puts the data in a buffer in memory.
4. When the buffer is full, the Mux wakes up the Controller.
5. The Controller gives a new buffer address to the Mux.
6. The Controller requests new filtering coefficients.
7. The Analyzer computes new coefficients with the buffered data.
8. The Analyzer sends new coefficients.
9. The Filter receives new filtering coefficients.

7. Results: Audio Filter



H/S choice under Invader

7. Results (on a Pentium IV - 1.8 GHz, 512 MB of RAMBus)

Level	Number of data generated (*producer* alway in Hw)	Number of iterations for the *controller*	Data analyzed per iteration	Efficiency (generated vs analyzed)	Simulation time	Cycles
L1	10000	5	2000	100%	3 s	4.6 M
L2 100% SW	10000	5	2000	100%	21 s	4.8 M
L2 PART.	10000	5	2000	100%	15 s	4.7 M
L4 100% HW	10000	5	2000	100%	20 s	4.7 M
L4 100% SW	100	2.5	20	50%	22 s	4.7 M
L4 PART.	10000	2.5	2000	50%	22 s	4.7 M

Here, two metrics are considered: bandwith and efficiency

7. Conclusion and future works

- We proposed a refinement methodology lets system designers validate their application functionality at each abstraction level and simulation provides more and more precise results through each level.
- Encapsulating RTOS functionalities into an API allows SystemC to be a common language
- Adapt a partitioning algorithm and add metrics
- Support SOCP protocol from OCP.
- Adapt a verification methodology
- Target an FPGA (Virtex-II Pro)

AP107-6PCI from Amirix