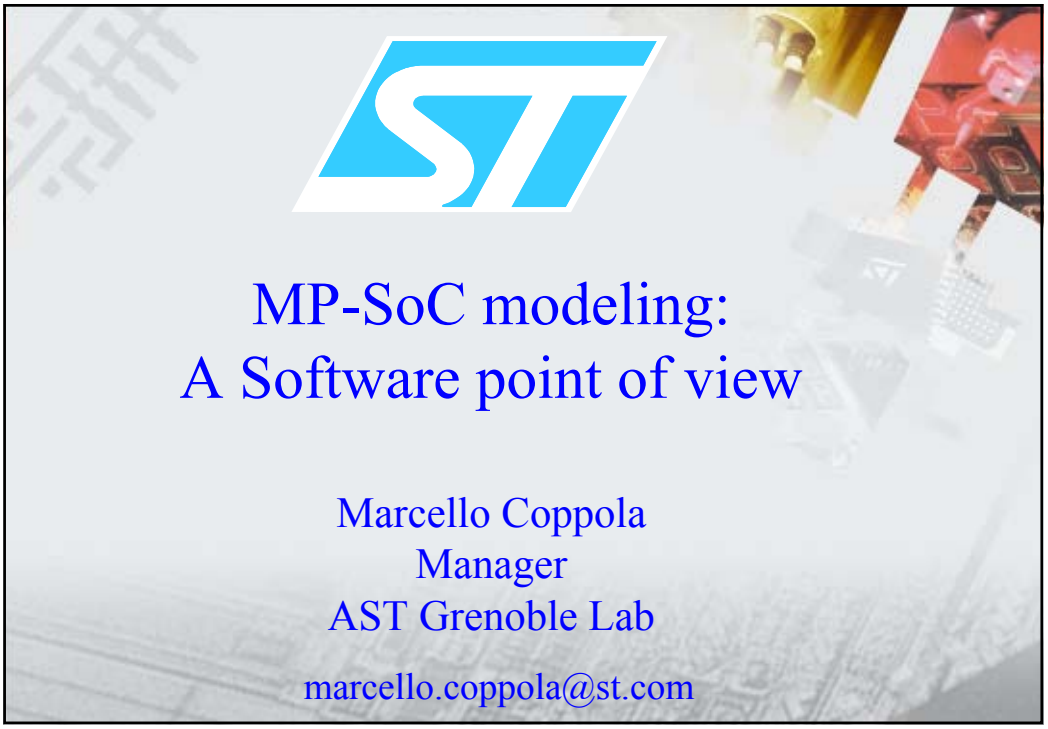


MP-SoC modeling: A Software point of view

Marcello Coppola
Manager
AST Grenoble Lab

marcello.coppola@st.com

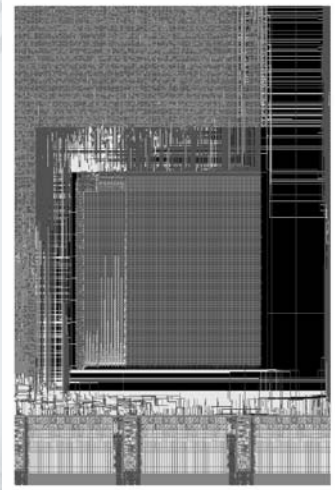
Embedded System

Devices designed for dedicated purposes where the correctness of the system depends not only on the logical result of the computation but also on the time at which the results are produced

"... the New York Times has estimated that the average American comes into contact with about 60 micro-processors every day...." [Camposano, 1996]

Latest top-level BMWs contain over 100 micro-processors [Personal communication]

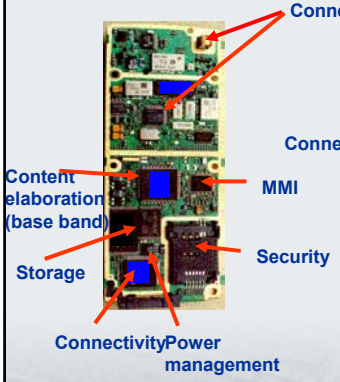
Embedded MP System on Chip

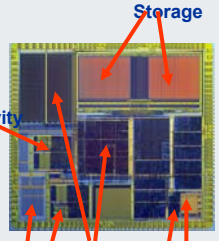


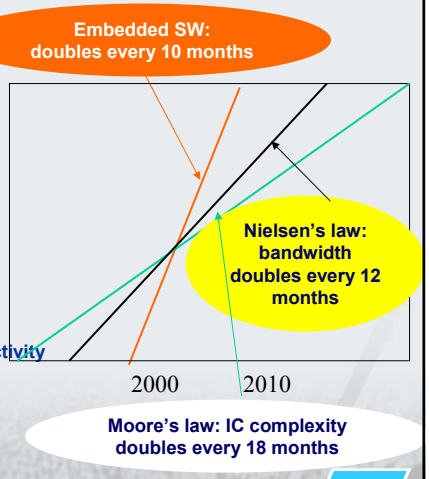
- Today SoC includes several modules such as
  - MCU,VLIW,DSP
  - Memories,
  - ASICs
  - Busing scheme
  - **Application Software**
  - **RTOS**
  - **BSP**

A better name for SoC is **MP-SoC**

MP-SoC developement: Technology pressure





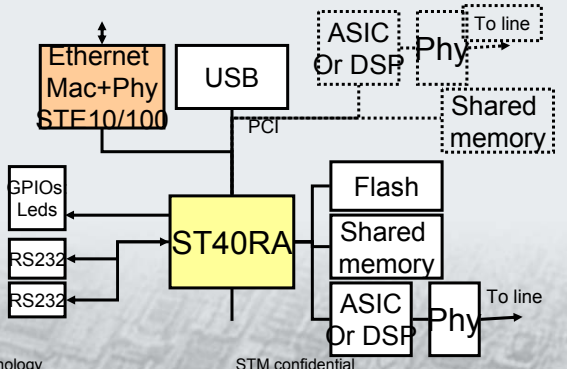


MP-SoC developement: Market pressure

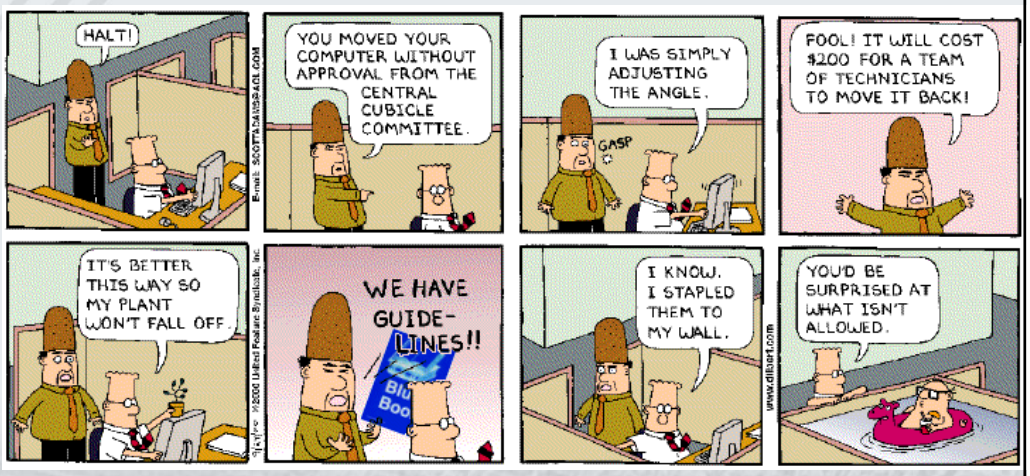
- Increasing features
- Decreasing time to market
- Decreasing prices
- Narrow windows

Solution: Reuse; Platform based?

To meet the demands of SoC deign flows we are moving the reuse from: IP -> Sub-system -> Platform



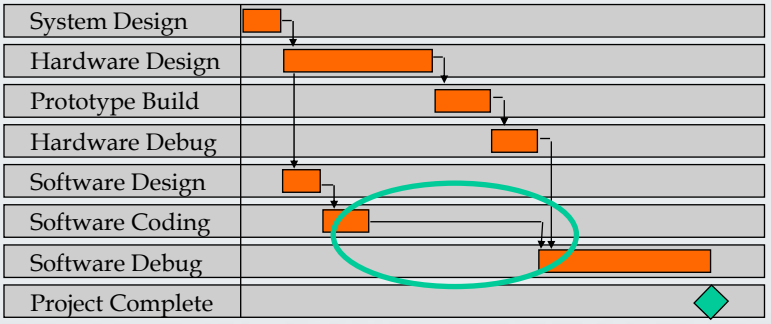
Reuse is not…



DILBERT by Scott Adams © 2000 United Feature Syndicate, Inc.

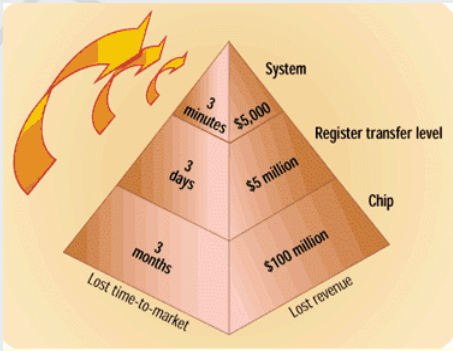
Embedded System Schedule

Typical embedded system project schedule



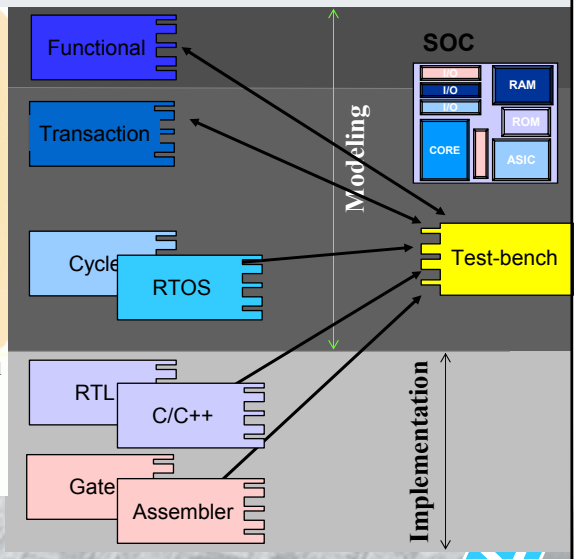
Source: Embedded Systems Programming, Jan 1996

Compromise: Multi-levels Validation

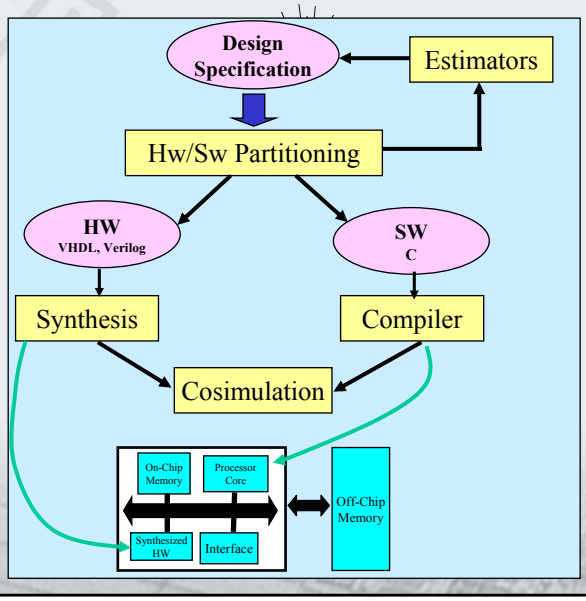


Source: Integrated Communications Design May, 2001

Higher Abstraction layer implies shorter Iteration Cycles and less Lost Revenue

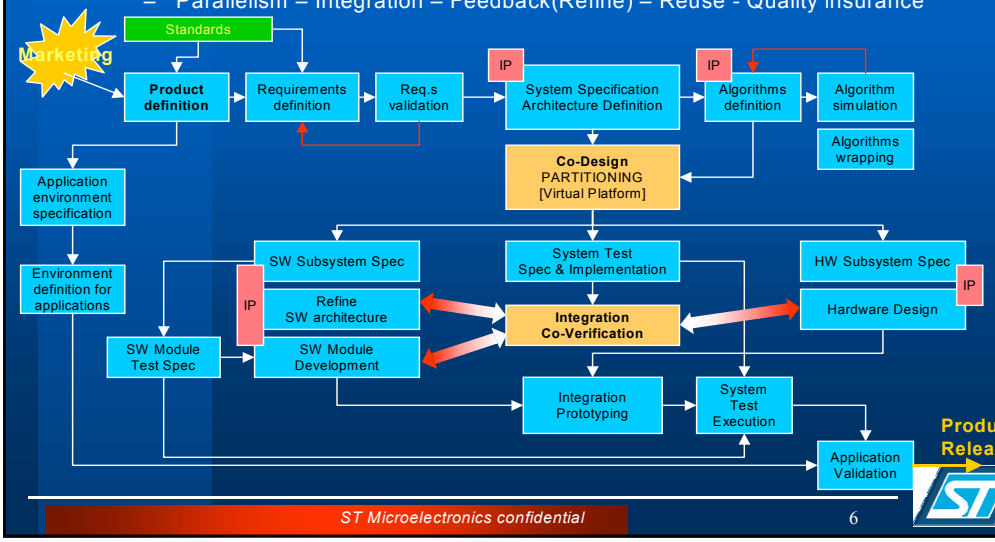


Design Flow



Project Flow

- The process has to include:
  - Parallelism – Integration – Feedback(Refine) – Reuse - Quality insurance

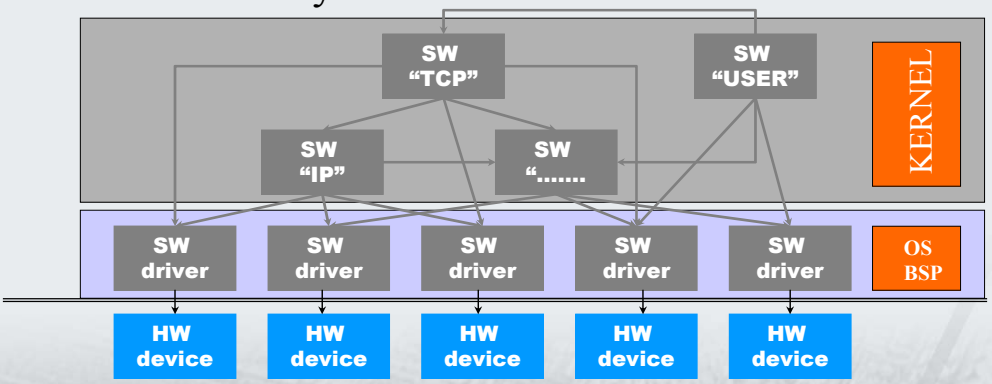


Specifying Hardware & Software

- Processor speed/ type
- Bus architecture
- RTOS
- Third-party sw libraries
- Third-party hw components

Complex Software Systems

- The growing complexity of real-time system functionality is reflected in the software



OS Basic

- Laplante (1997) defines operating system by describing a software hierarchy

User interface Shell	Operating system
File and disk support	Executive
Interprocessor communication and synchronization	Kernel
Task scheduling	Micro-kernel
Thread/task control Block management	Nano-kernel

What is a Real-time OS?

- A *RTOS* (Real-Time Operating System)
  - Is an Operating Systems with the necessary features to support a Embedded Real-Time System
  - What is a Embedded *Real-Time System?*
    - A system embedded where correctness depends not only on the correctness of the logical result of the computation, but also on the result delivery time

RTOS Structure

- **System Tables**
- **Dispatcher**
- **Interrupt Service Routines (ISR)**
- **System Services**

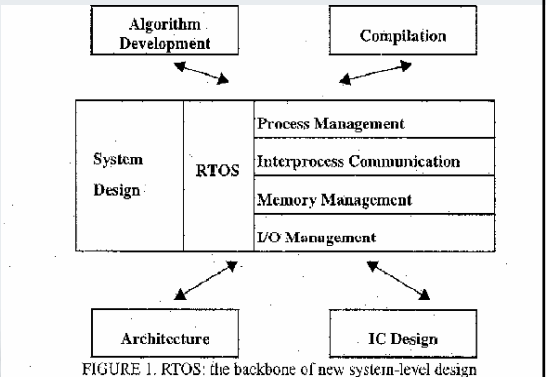


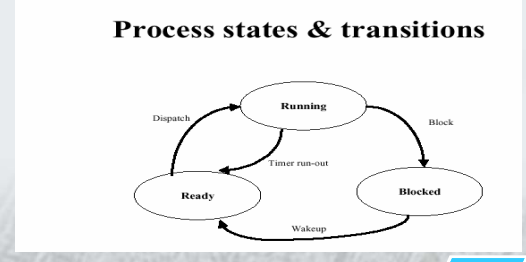
FIGURE 1. RTOS: the backbone of new system-level design process

Operating System Structure: System Tables

- **System Tables:**
  - **Task-Control-Blocks(TCB):** These hold information necessary to run the tasks.
  - **Device-Control-Blocks(DCB):** These hold the information necessary for the operating system to use the I/O devices. Also, each device may have an additional table or buffer to hold I/O information as the I/O device operates.
  - **Service-Control-Blocks(SCB):** These specify the parameters for a request for the operating system to perform some function.
- A complete system has one TCB for each task, one DCB for each I/O device, and many SCBs.

Operating System Structure: The Dispatcher

- The dispatcher is a program module that determines which task will run next after something triggers the system to switch control to a new task.
- A task is a separate execution path in a process



Kernel Scheduling

- **Polled-loop:** This scheme is used when there is no scheduling
- **State-driven:** This scheme is used when the process is break up in discrete segments of code. Each segment is one state of a FSM.
- **Cooperative:** This scheme is used within a multitasking environment depends on the running task relinquishing the control .
- **Preemptive:** This scheme is used as above, and it forces one task to give up the control in order to let another task to have it

Scheduling Schemes

- **Round-robin Scheduling:** Rotate control between all tasks.
- **Priority Scheduling:** A priority scheduling system assigns a priority to each task and continues down the list of tasks in priority order.
- **Background Task:** One task is always ready to run.
- **Combination Scheduling:** This scheduling algorithm uses a combination of the algorithms discussed above.

Context switching

- The action to saving and restoring sufficient info in order to switch between tasks

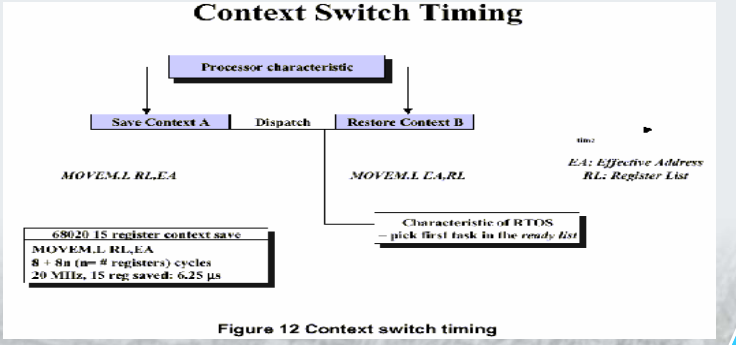


Figure 12 Context switch timing

Interrupt Service Routine(ISR) & Interrupts

- For each I/O device, there is an associated ISR that is run when that I/O device sends triggers an interrupt in the system. The ISR contains the details of the I/O devices operation.
- The ISR controls the device to provide data transfer between the outside world and the inside of the operating system.
- ISR's generally run with the interrupt system disabled, therefore the ISR should contain short program code in order to execute relatively quickly.

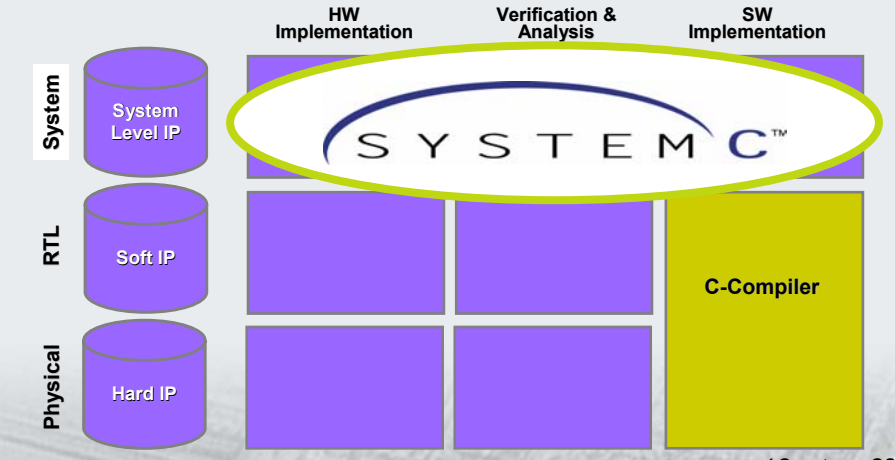
Interrupts

- Interrupt Dispatch Time
  - time the hardware needs to bring the interrupt to the processor
- Interrupt Routine
  - ISR execution time
- Other Interrupt
  - Time needed for managing each simultaneous pending interrupt
- Pre-emption
  - Time needed to execute critical code during which no pre-emption may happen

Synchronization & Communication

- **Synchronization and exclusion objects**
  - Semaphores – synchronization and exclusion
  - Mutexes - exclusion
  - Conditional variables – exclusion based on a condition
  - Event flags – synchronization of multiple events
  - Signals – asynchronous event processing and exception handling
- **Communications**
  - Queues – mulitple messages
  - Mailboxes – single messages

SystemC

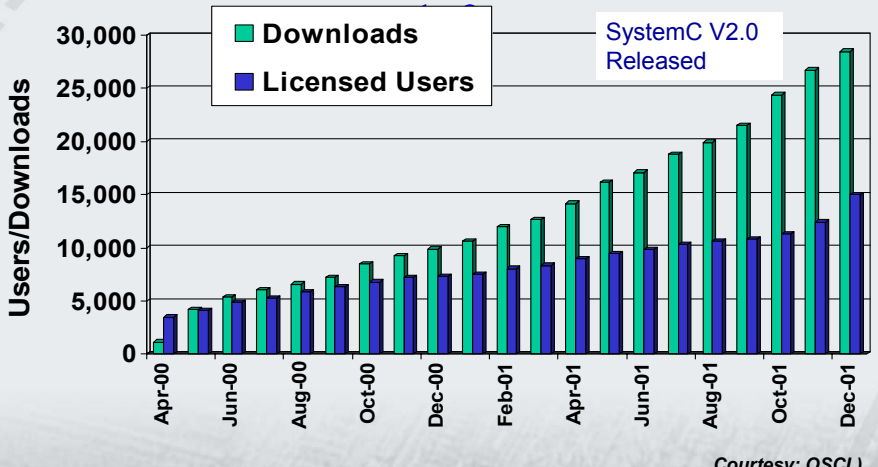


(Courtesy: OSCI)

Market Adoption

- End Users
  - Over 15,000 Licensees at over 500 companies/institutions
    - 3,000 regular downloaders with every release
  - > 30K successful downloads of SystemC source code
  - Unsolicited SystemC successes stories in conference proceedings
    - 10 Commercial ones - AMD, Infineon, Siemens, CSELT, FhG, STM
    - >10 more known ones slated for this year
- EDA, IP and Services
  - Over 25 Companies / over 40 Products in the works
    - EDA tools - 30
    - Training - 7
    - IP - 3
    - Open Source Resources - 10
  - See http://www.systemc.org Products and Solution section for more details

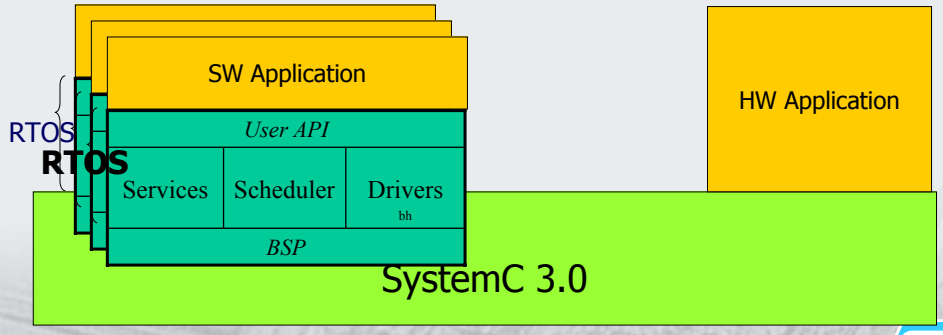
Strong SystemC Adoption since



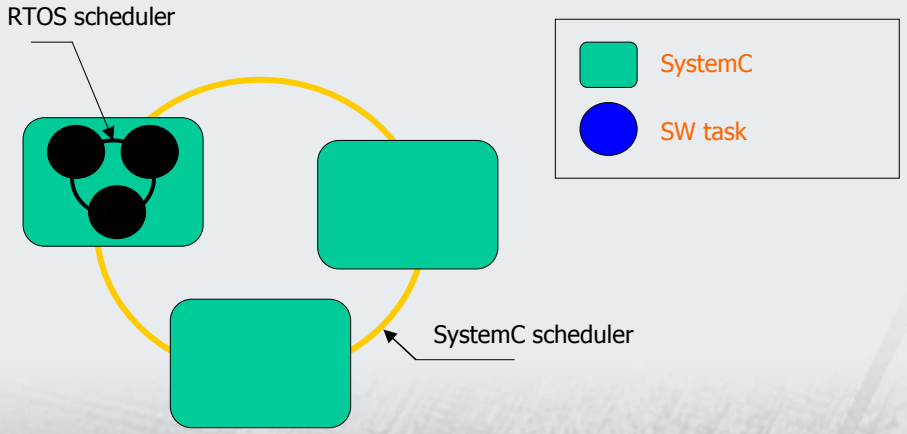
Courtesy: OSCI)

MP-SoC Modeling

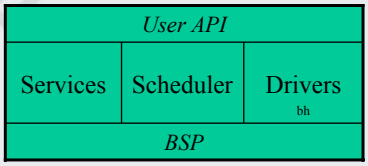
- SystemC can provide an environment, modular et extensible, that allow the overall SoC simulation/validation

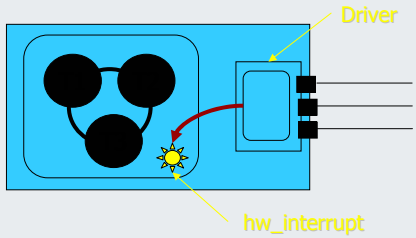


RTOS modeling in systemC2.0



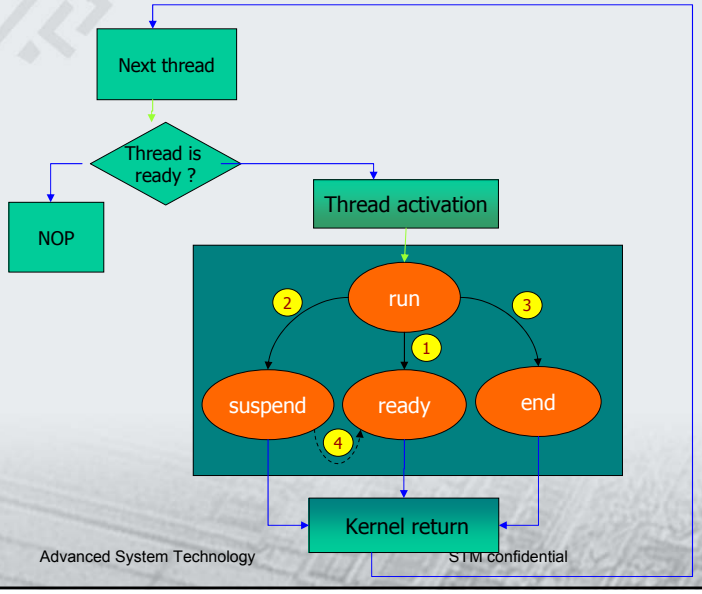
Modèle de base : interaction HW/SW





- Application Software and Hardware interaction are achieved by the systemC ports using drivers
- Interrupts are handled using a special channel
- Scheduler and services are modeled inside a sc_module.

RTOS scheduler

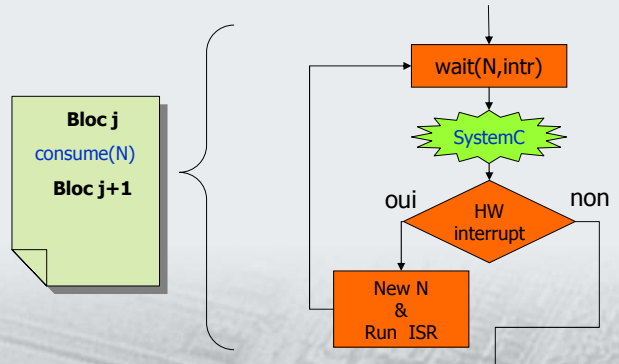


1. thread_yield()
2. thread_suspend(n) mutex_lock()…
3. thread_exit()
4. mutex_unlock()…

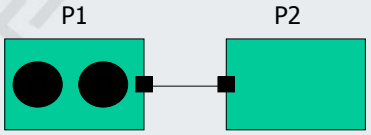
CPU time

A new function has been introduced consume(delay) allowing:

- model CPU processing time within an Application or RTOS
- Interrupt Management



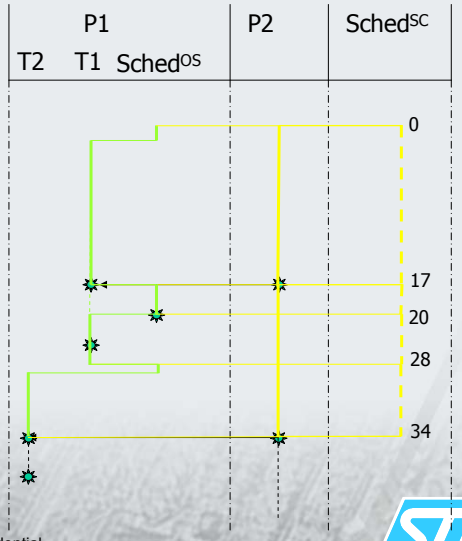
Example



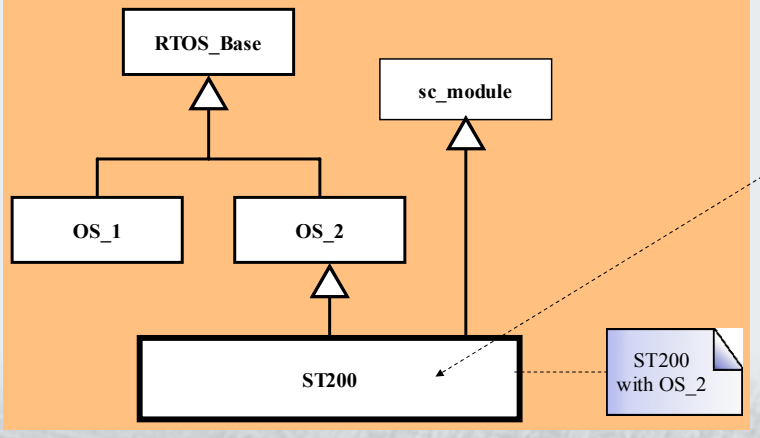

```
void T1(){
    while(1){
    consume(25,ns);
    yield();
    }
}
```

```
void T2(){
    while(1){
    consume(10,ns);
    yield();
    }
}
```

```
void entry(){
    while(1){
    wait(17,SC_NS);
    port = value;
    }
}
```



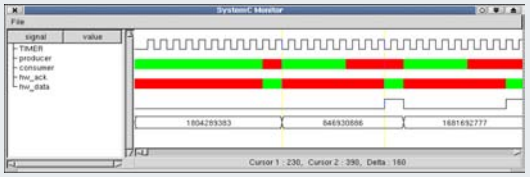
One possible implementation

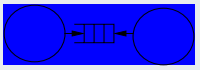



```
void T1(){
    while(1){
    consume(25,ns);
    yield();
    }
}
```

```
void T2(){
    while(1){
    consume(10,ns);
    yield();
    }
}
```

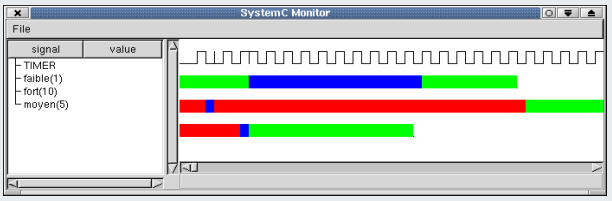
Example 1: Producer Consumer



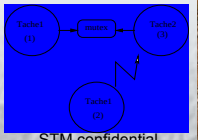


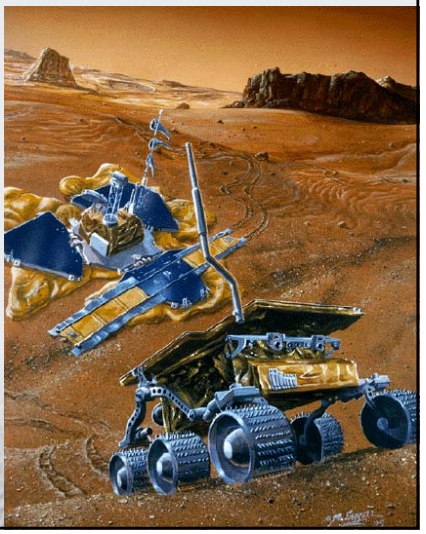
running
ready
suspended

Example 2: Priority inversion



running
ready
suspended





Conclusion

•SoC includes :Processors , memories, ASIC, DSP ...)

•Needed a design flow that starts from a specification functional and produce an architecture where a software application can be executed

•Simulation is used for system validation and system refinement in order to drive the differents choices that may be taken during the SoC

•New simulation semantics are needed

- SystemC 3.0 may provide the required semantic