High Speed Multi-Processors System-On-Chip Simulation Platforms for Hardware Dependent Software Development

Frédéric Pétrot

System-Level Synthesis Group, TIMA Laboratory
46, Av Félix Viallet, 38031 Grenoble, France

Eugenio Villar

TEISA Department, University of Cantabria
Av. Los Castros, Santander, Spain

October 14th 2009

Hardware Dependent Software and System-On-Chip Simulation Platform

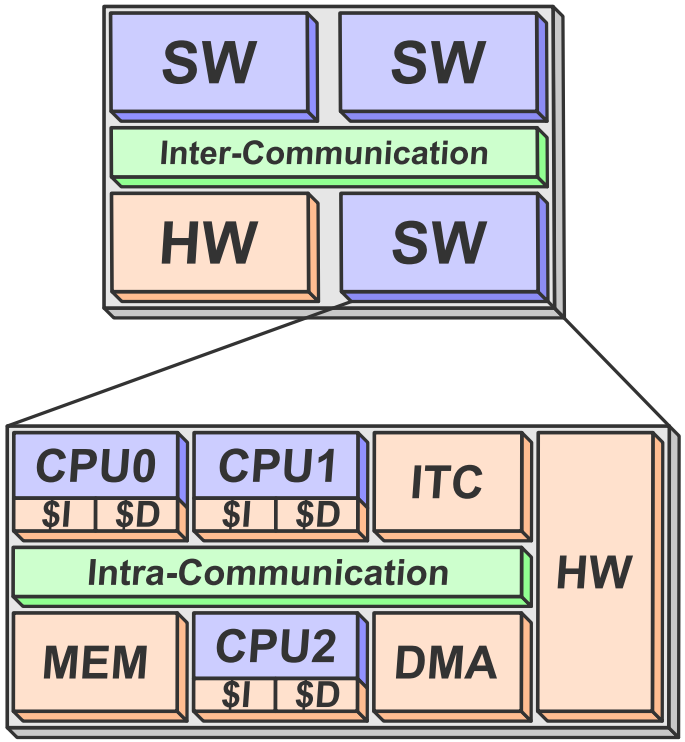
The Trends:

MPSoC architecture becomes complex

- Multiple processors for parallel applications
- Multiple accelerators and I/O devices

HDS development becomes complex also

- One platform can support several OS
- Device driver porting is tedious



Hardware Dependent Software and System-On-Chip Simulation Platform

The Trends:

MPSoC architecture becomes complex

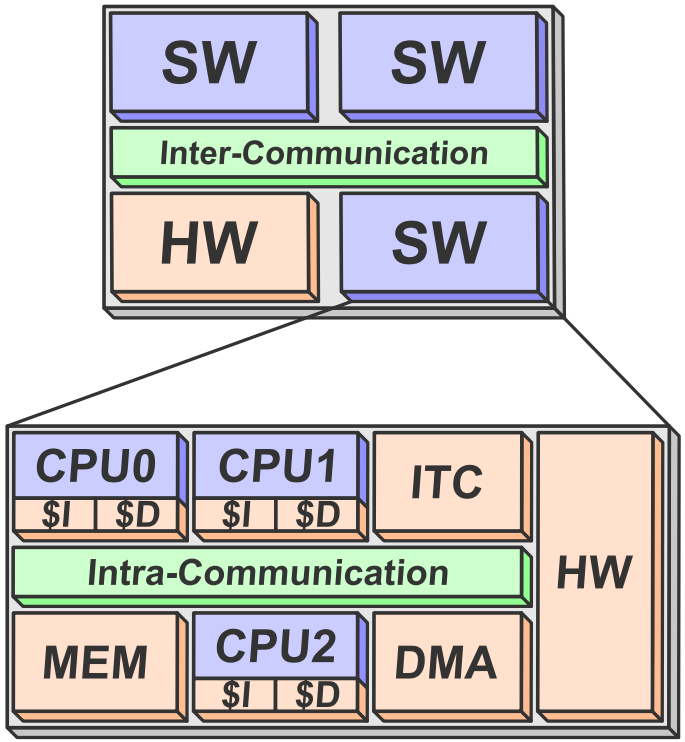
- Multiple processors for parallel applications
- Multiple accelerators and I/O devices

HDS development becomes complex also

- One platform can support several OS
- Device driver porting is tedious

The Challenges:

- HDS validation and debug



Hardware Dependent Software and System-On-Chip Simulation Platform

The Trends:

MPSoC architecture becomes complex

- Multiple processors for parallel applications
- Multiple accelerators and I/O devices

HDS development becomes complex also

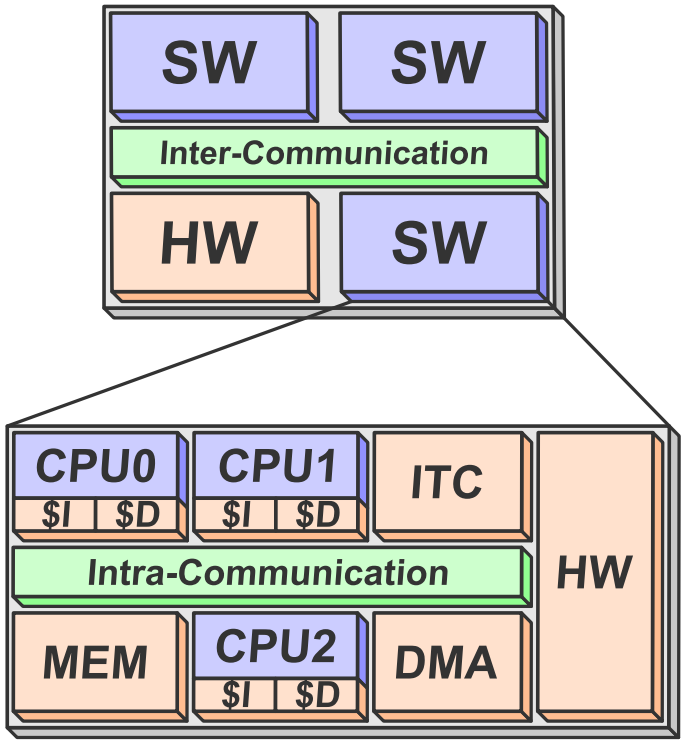
- One platform can support several OS
- Device driver porting is tedious

The Challenges:

- HDS validation and debug

Focus of these works:

- High speed HW/SW co-simulation
- With low level details of HDS
- At HAL level (TIMA)
- At RTOS level (U. Cantabria)



Hardware Dependent Software and System-On-Chip Simulation Platform

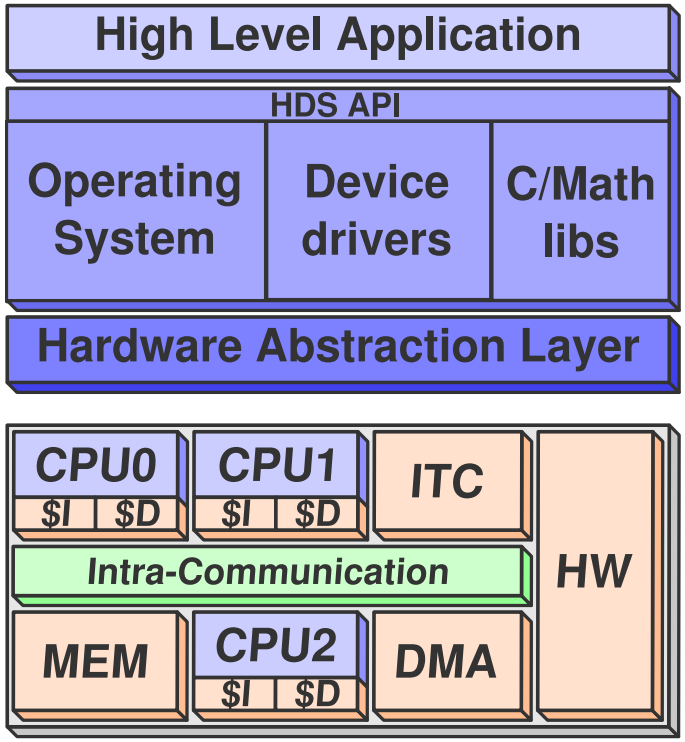
Classical approaches

Cycle accurate co-simulation environment

- Cross compiled embedded software
- Interpreted and executed by ISSs
- Accurate but slow
  $\Rightarrow$ ISSs is 3 orders of magnitude slower than the native simulation method
  $\Rightarrow$ QEMU is faster, 1 order of magnitude

TLM based co-simulation environment

- Abstraction of the hardware in TLM
- Software still interpreted by ISSs



Hardware Dependent Software and System-On-Chip Simulation Platform

Classical approaches

Cycle accurate co-simulation environment

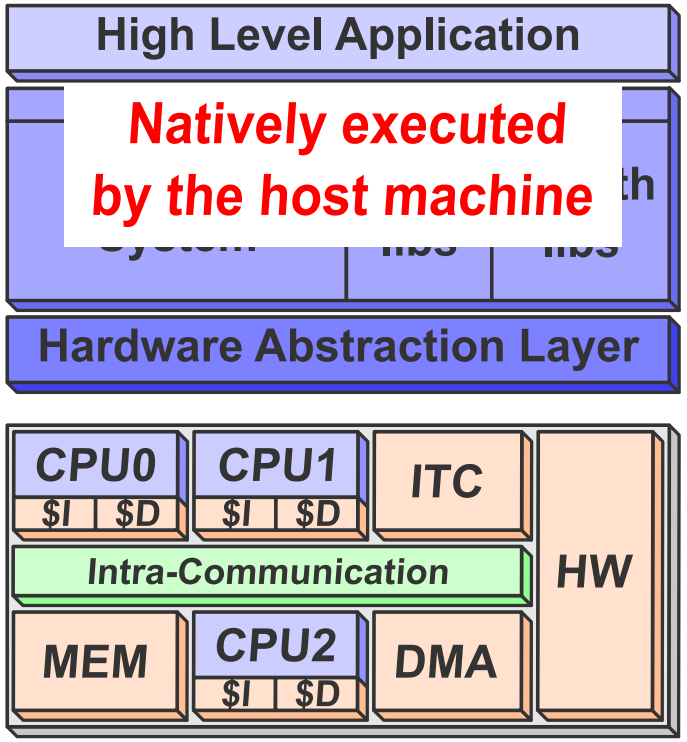
- Cross compiled embedded software
- Interpreted and executed by ISSs
- Accurate but slow
  $\Rightarrow$ ISSs is 3 orders of magnitude slower than the native simulation method
  $\Rightarrow$ QEMU is faster, 1 order of magnitude

TLM based co-simulation environment

- Abstraction of the hardware in TLM
- Software still interpreted by ISSs

Native HW/SW co-simulation approaches

- Software is executed by the host machine
- Considerable speedup
- No application modifications
- Functional validation of the whole system



Objectives & Contributions

Objectives: MPSoC simulation platform

- Taking into account hardware parallelism
- High simulation speed to allow
  - User application validation and debug
  - HDS validation and debug
  - Hardware architecture validation and debug
- No software source code modification
  - To debug the user application software
  - To debug the operating system (TIMA)
  - To debug the device drivers
  - To explore the design spaces (U. Cantabria)

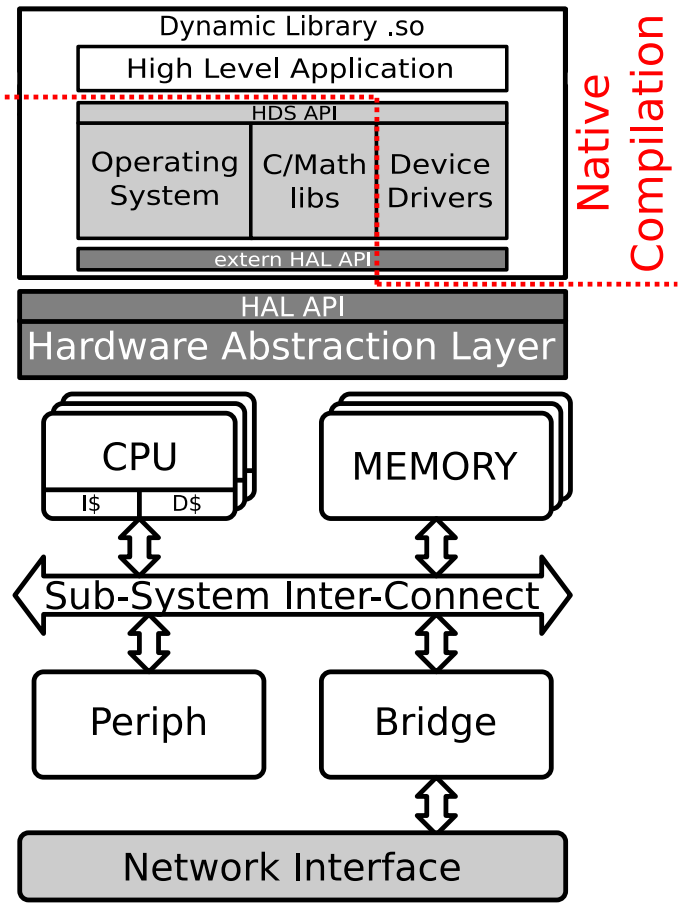
Contributions: A methodology to implement fast and precise MPSoC platforms

- With accurate memory modelization and hardware software interactions
- Based on native approaches to achieve simulation speed
- Allowing a complete software portability to the target platform
- Running on SystemC

RTOS Level Native Simulation Approach

Definition: Native simulation

SW application is compiled for the workstation processor instead of the chip one.
The binary is directly executed on the workstation processor.



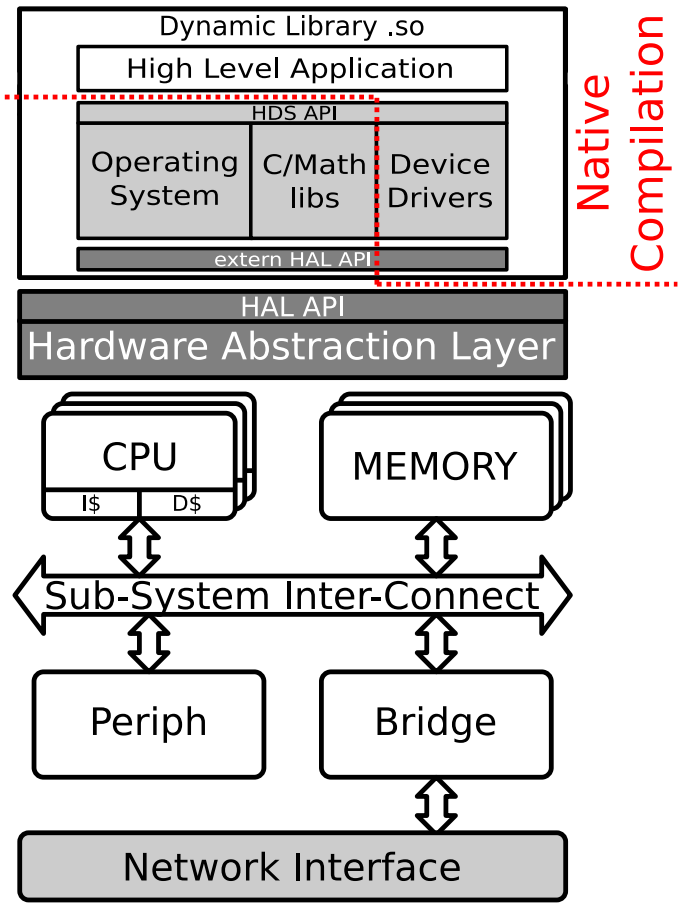
RTOS Level Native Simulation Approach

Definition: Native simulation

SW application is compiled for the workstation processor instead of the chip one.
The binary is directly executed on the workstation processor.

Simulated source-code

- Application
- Device drivers



RTOS Level Native Simulation Approach

Definition: Native simulation

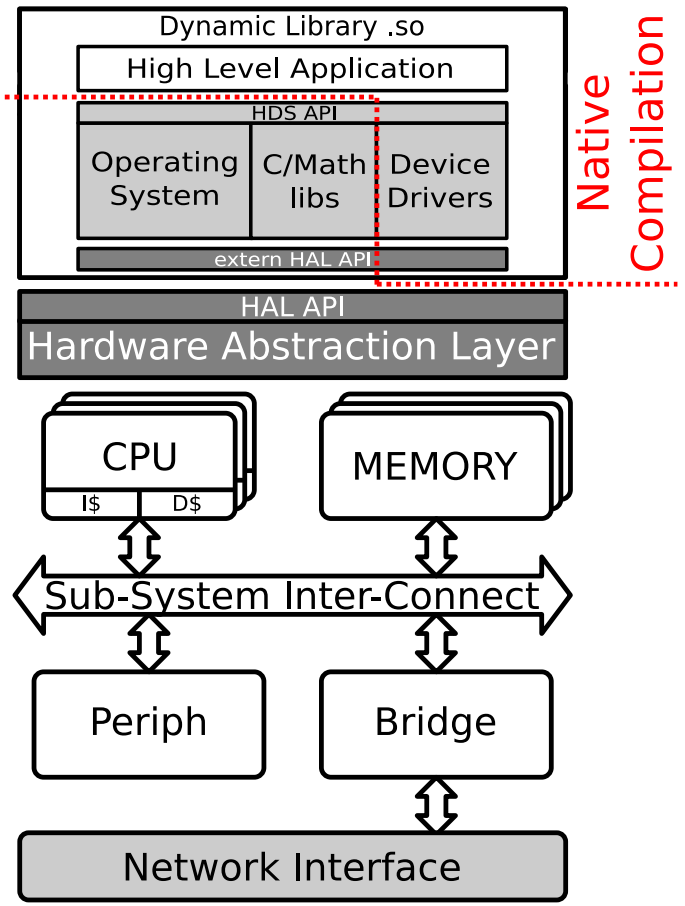
SW application is compiled for the workstation processor instead of the chip one.
The binary is directly executed on the workstation processor.

Simulated source-code

- Application
- Device drivers

Modeling elements

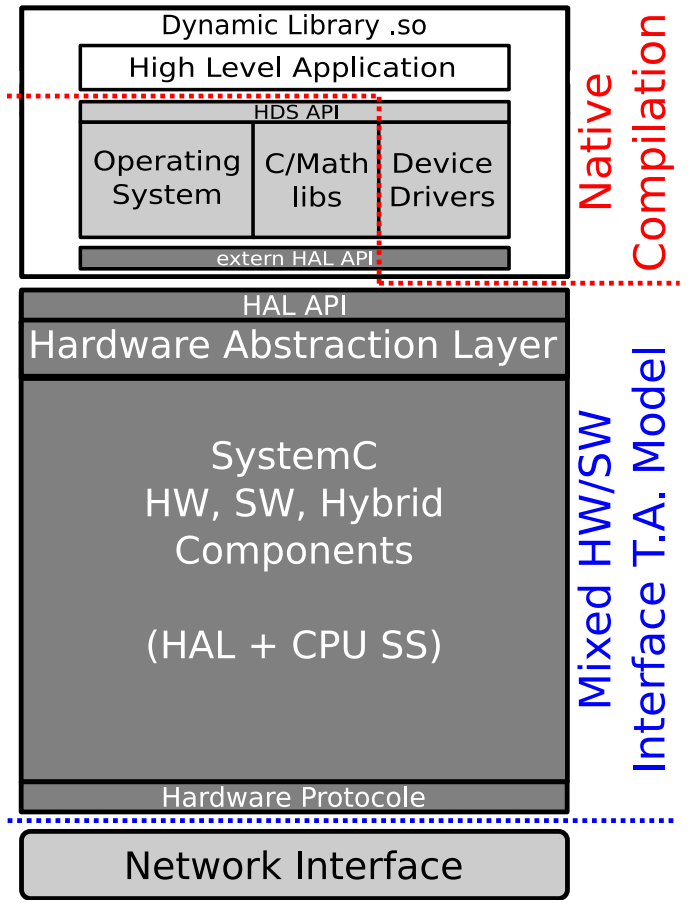
- Abstract model of the RTOS
- Host OS Libraries (C, Math, ...)
- Communication libraries (stack IP, ...)



Basic Concepts: Transaction Accurate Model (RTOS API Level)

Transaction Accurate Model

SystemC executable model of HW components processor modules handle the HAL APIs



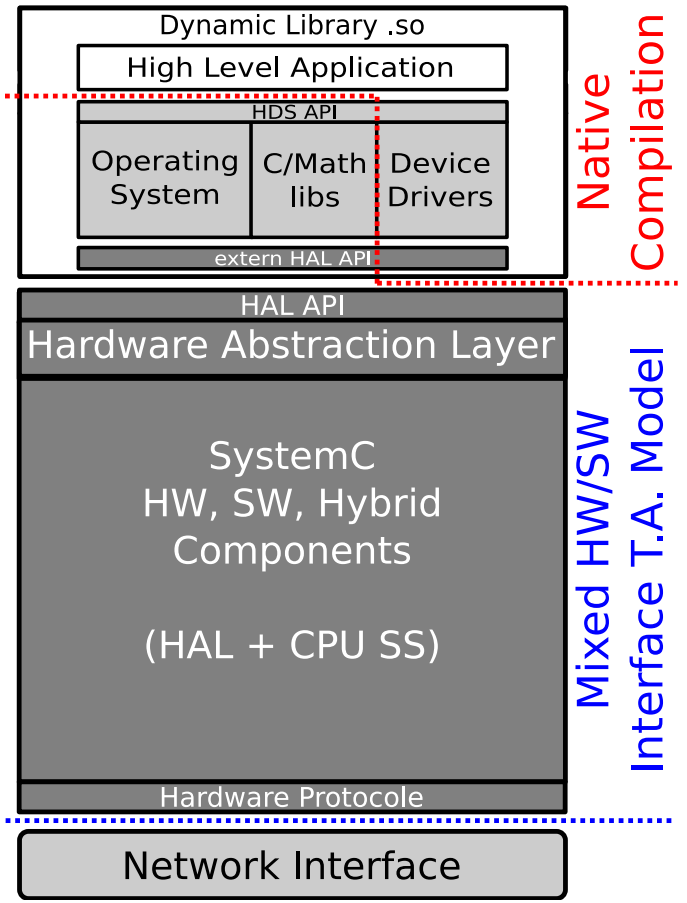
Basic Concepts: Transaction Accurate Model (RTOS API Level)

Transaction Accurate Model

SystemC executable model of HW components processor modules handle the HAL APIs

Application software interface: RTOS API

- POSIX, μCOS, WinCE



Basic Concepts: Transaction Accurate Model (RTOS API Level)

Transaction Accurate Model

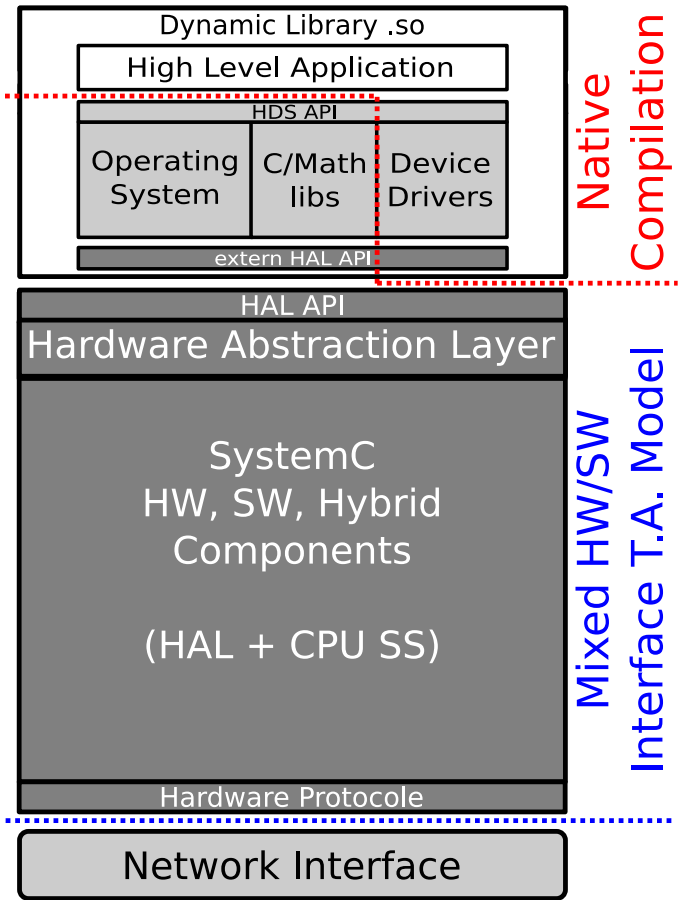
SystemC executable model of HW components processor modules handle the HAL APIs

Application software interface: RTOS API

- POSIX, μCOS, WinCE

Driver interface

- Linux 2.6



Basic Concepts: Transaction Accurate Model (RTOS API Level)

Transaction Accurate Model

SystemC executable model of HW components processor modules handle the HAL APIs

Application software interface: RTOS API

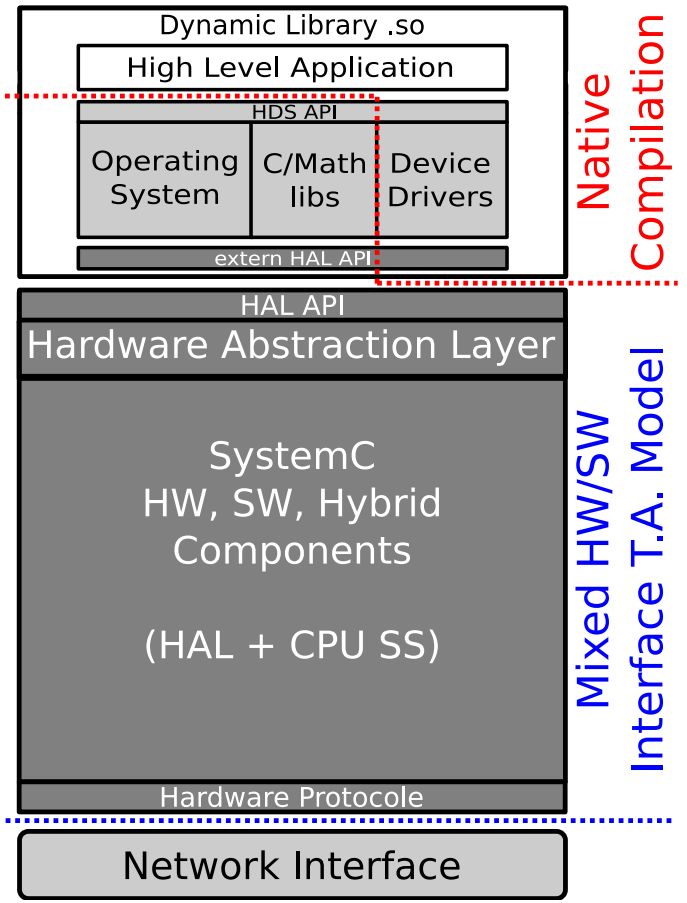
- POSIX, μCOS, WinCE

Driver interface

- Linux 2.6

Hardware interface: TLM2.0

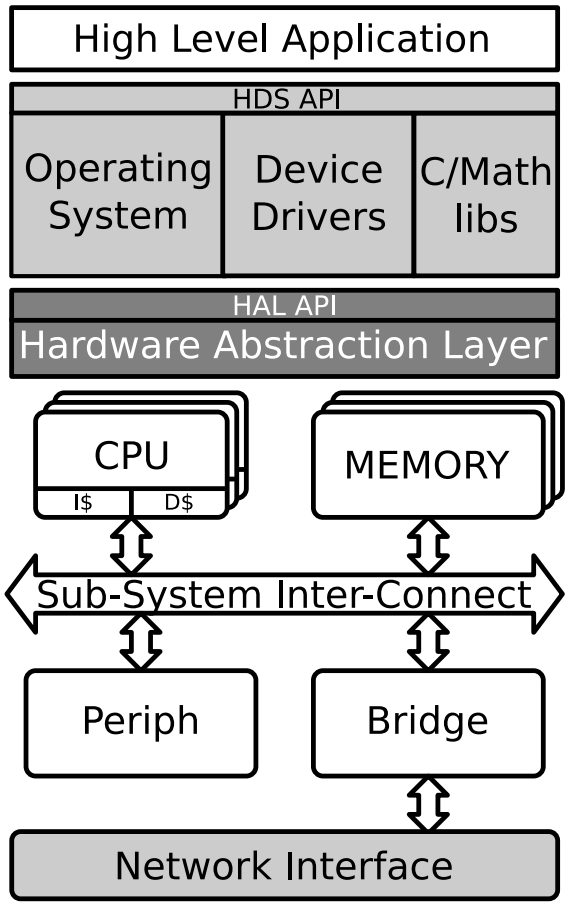
- Payload Bus model
- AMBA, ...



HAL Level Native Simulation Approach

Definition: Native simulation

SW application is compiled for the workstation processor instead of the chip one.
The binary is directly executed on the workstation processor.



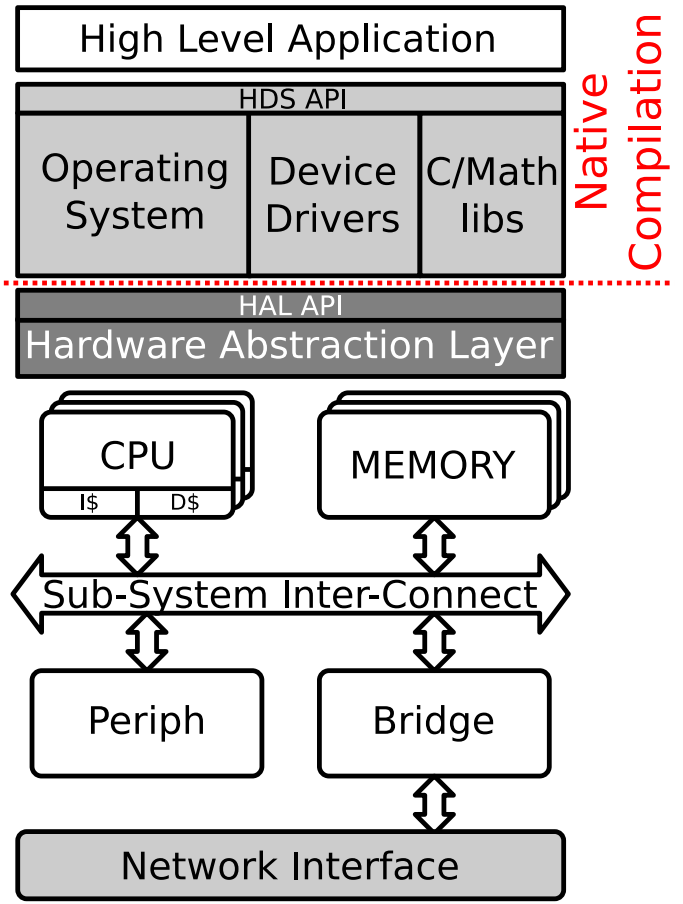
HAL Level Native Simulation Approach

Definition: Native simulation

SW application is compiled for the workstation processor instead of the chip one.
The binary is directly executed on the workstation processor.

All software layers above the HAL API

- Application
- Operating System
- Device drivers
- Libraries (C, Math, Com, ...)



HAL Level Native Simulation Approach

Definition: Native simulation

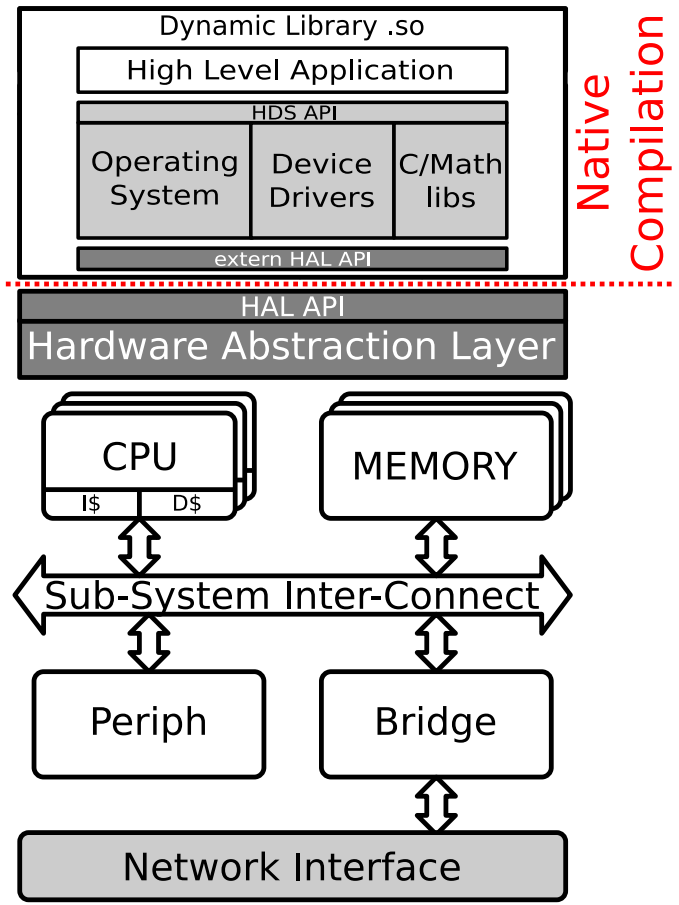
SW application is compiled for the workstation processor instead of the chip one.
The binary is directly executed on the workstation processor.

All software layers above the HAL API

- Application
- Operating System
- Device drivers
- Libraries (C, Math, Com, ...)

Software encapsulation

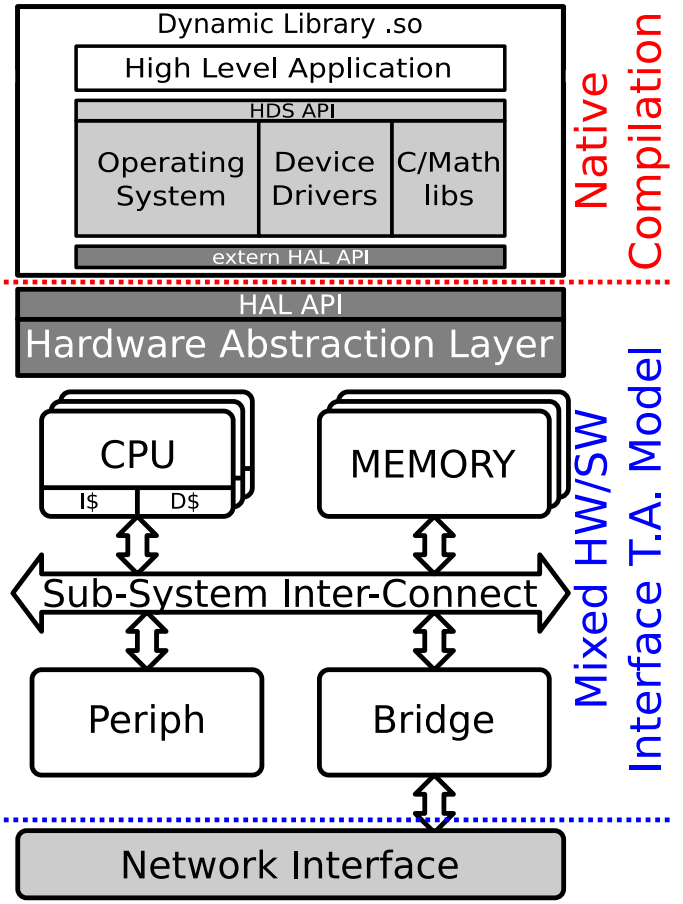
- In a classical dynamic library
- All HAL APIs are exposed to OS and drivers



Basic Concepts: Transaction Accurate Model (HAL API Level)

Transaction Accurate Model

SystemC executable model of HW components
processor modules handle the HAL APIs



Basic Concepts: Transaction Accurate Model (HAL API Level)

Transaction Accurate Model

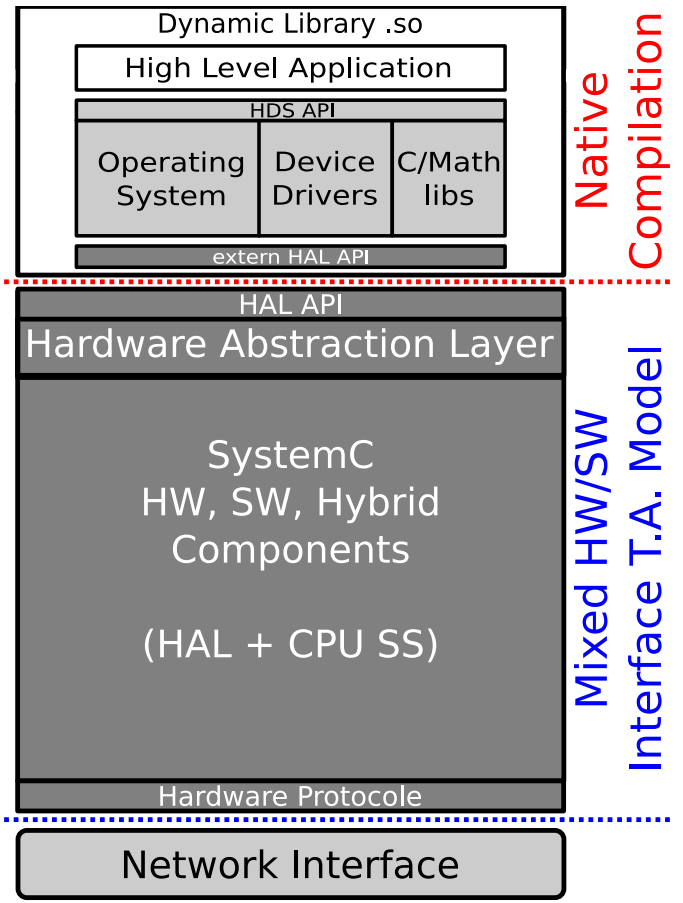
SystemC executable model of HW components
processor modules handle the HAL APIs

Software interface: HAL API

- Context switching
- Interrupt management
- Endianness conversion
- I/O accesses

Hardware interface: HW protocol

- VCI, AMBA, ...
- Specific HW interface (e.g. FIFO)



Efficient Native Simulation for MPSoC

Key Ideas:

- Keep the low level hardware details for software native simulation
- No application software and device driver code modification

Key Challenges:

- Memory representation
  - Considered private for each processor in classical native simulation
  - Shared memories have to be modeled to support parallel architectures
- Software application timing annotation

Problematic

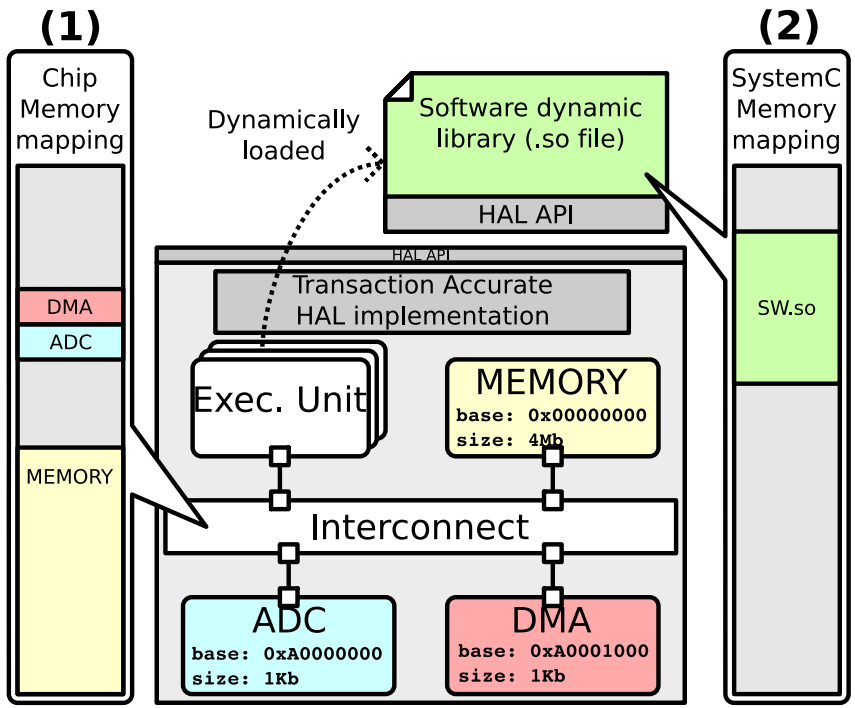
Two memory mappings to be considered

The chip memory mapping (1)

- Defined by the HW designers
- Used by the address decoder at simulation time

The SystemC memory mapping (2)

- Shared by the SW stack
- Host machine dependent
- Contains standard sections
  - Program in `.text`
  - Initialized data in `.data`
  - Uninitialized data in `.bss`



Problematic

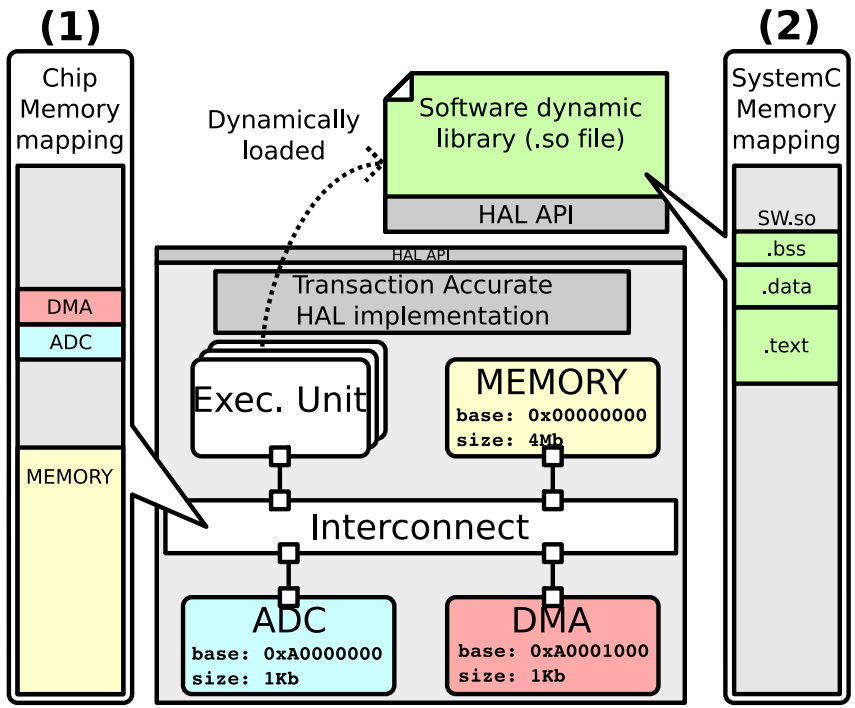
Two memory mappings to be considered

The chip memory mapping (1)

- Defined by the HW designers
- Used by the address decoder at simulation time

The SystemC memory mapping (2)

- Shared by the SW stack
- Host machine dependent
- Contains standard sections
  - Program in `.text`
  - Initialized data in `.data`
  - Uninitialized data in `.bss`



Problematic

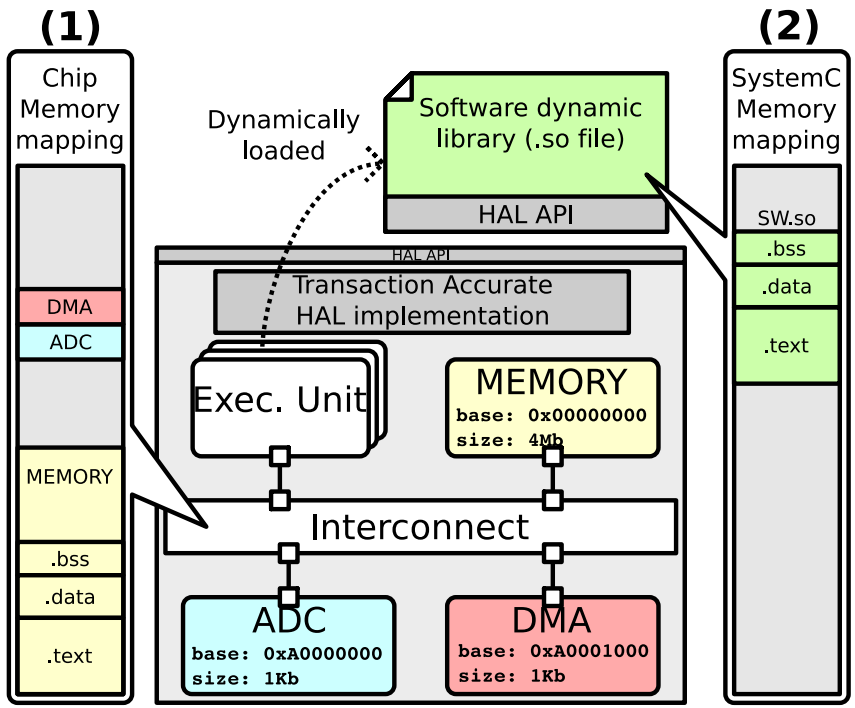
Two memory mappings to be considered

The chip memory mapping (1)

- Defined by the HW designers
- Used by the address decoder at simulation time

The SystemC memory mapping (2)

- Shared by the SW stack
- Host machine dependent
- Contains standard sections
  - Program in `.text`
  - Initialized data in `.data`
  - Uninitialized data in `.bss`



Unifying the memory mapping

A uniform memory mapping is required to

- Avoid software coding constraints
- Accurately model interactions between hardware components and software applications
- Model Symmetric Multi-Processor like architectures

Our choice:

- Use the SystemC memory mapping only

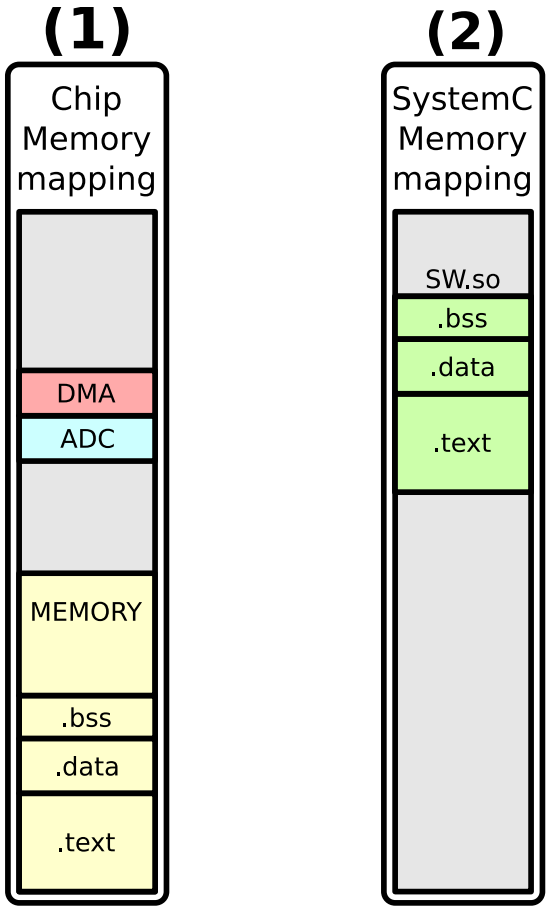
Proposed Solution

Proposed Solution

Use SystemC memory mapping only

Hardware components mapping

- HW components are modeled in SystemC
- For each modeled register, a corresponding address exists in the SystemC process mapping
- Use the register address in the SystemC mapping
- HW components must implement their real register mappings



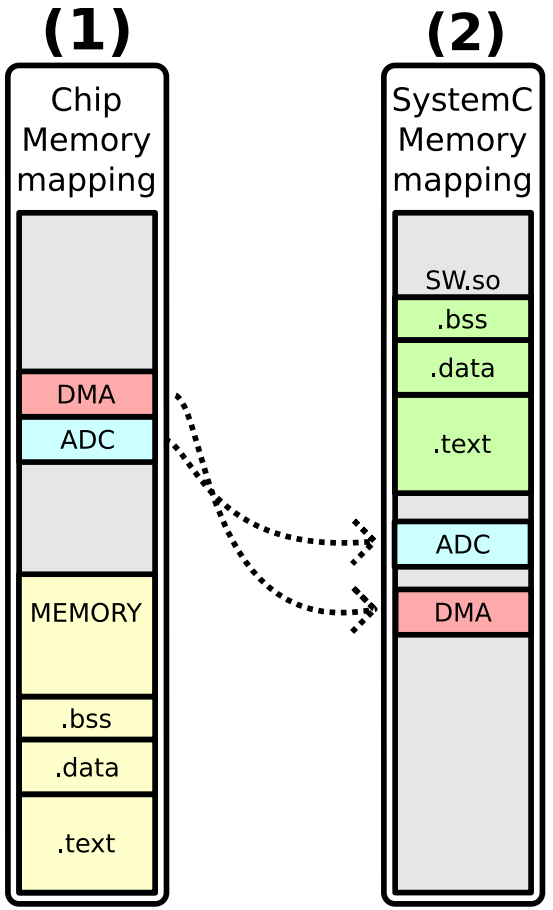
Proposed Solution

Proposed Solution

Use SystemC memory mapping only

Hardware components mapping

- HW components are modeled in SystemC
- For each modeled register, a corresponding address exists in the SystemC process mapping
- Use the register address in the SystemC mapping
- HW components must implement their real register mappings



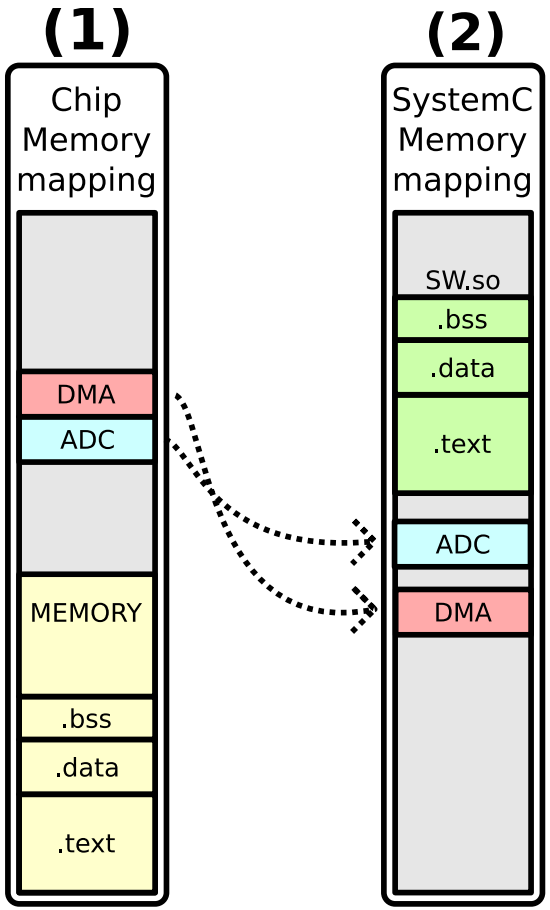
Proposed Solution

Proposed Solution

Use SystemC memory mapping only

Software section mapping

- Addresses of program and data sections obtained at compilation time are used directly
- Other memories are considered as HW components



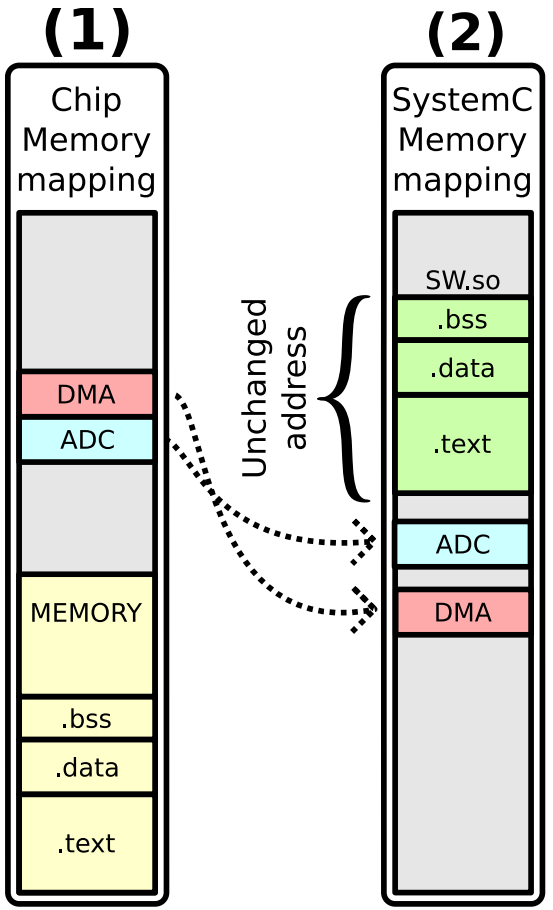
Proposed Solution

Proposed Solution

Use SystemC memory mapping only

Software section mapping

- Addresses of program and data sections obtained at compilation time are used directly
- Other memories are considered as HW components



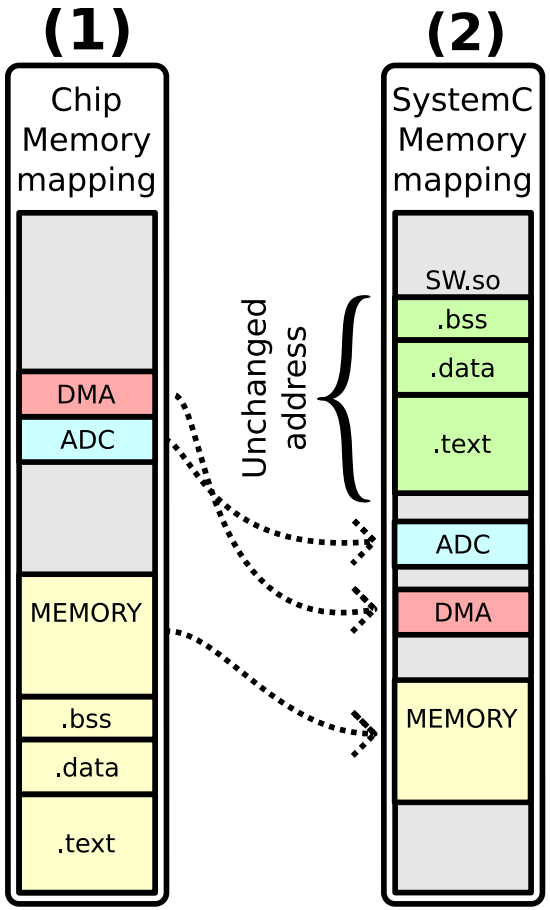
Proposed Solution

Proposed Solution

Use SystemC memory mapping only

Software section mapping

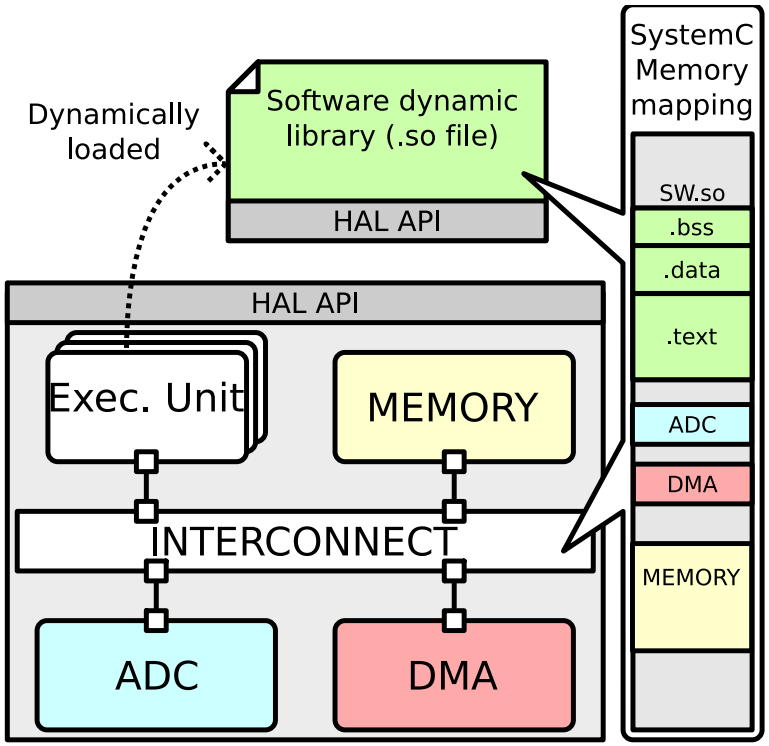
- Addresses of program and data sections obtained at compilation time are used directly
- Other memories are considered as HW components



Proposed Solution

The final memory mapping is obtained at simulation time

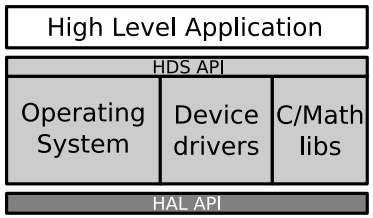
- Each mapped component provides its base address and size
- SW and HW rely on the same memory mapping
- The consistency of this memory mapping is ensured by the workstation



Approach Analysis

Code Reuse

- Native compilation of the SW stack for T.A. simulation
- Cross compilation of the same source code for low level simulations



Coding constraints (TIMA)

- HAL API must strictly be used
- Direct C pointer to access device registers are forbidden

```
int * adc_status = ADC_BASE;
*adc_status = 0x01F0; /*FORBIDEN*/
/* Correct use:*/
HAL_WRITE(UINT32,adc_status,0x01F0);
```

Approach Analysis

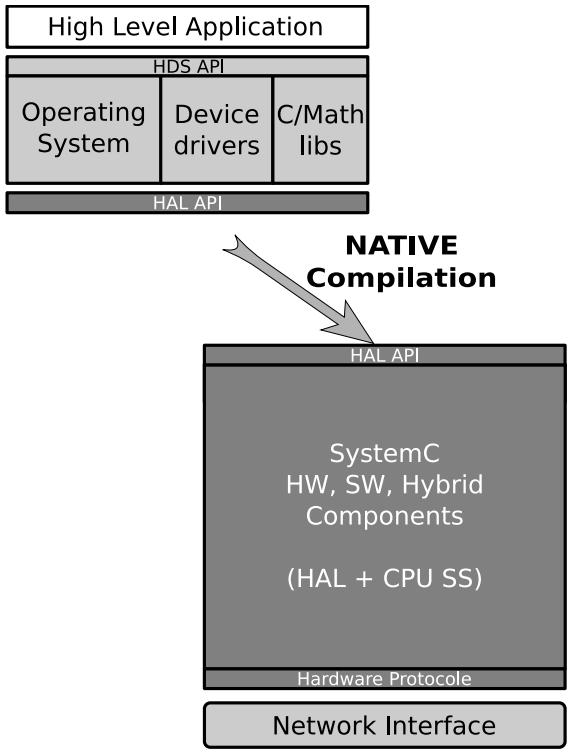
Code Reuse

- Native compilation of the SW stack for T.A. simulation
- Cross compilation of the same source code for low level simulations

Coding constraints (TIMA)

- HAL API must strictly be used
- Direct C pointer to access device registers are forbidden

```
int * adc_status = ADC_BASE;
*adc_status = 0x01F0; /*FORBIDEN*/
/* Correct use:*/
HAL_WRITE(UINT32,adc_status,0x01F0);
```



Approach Analysis

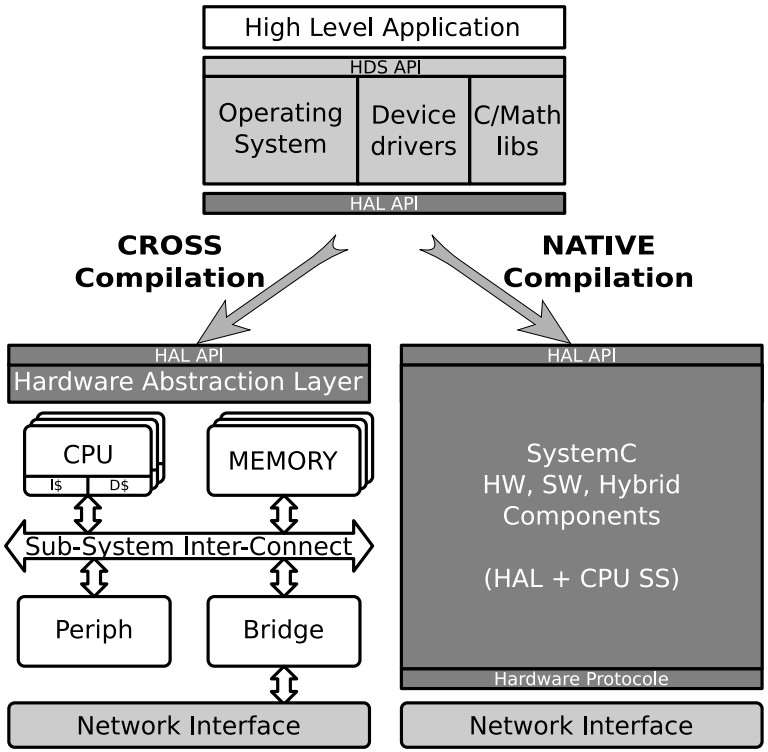
Code Reuse

- Native compilation of the SW stack for T.A. simulation
- Cross compilation of the same source code for low level simulations

Coding constraints (TIMA)

- HAL API must strictly be used
- Direct C pointer to access device registers are forbidden

```
int * adc_status = ADC_BASE;
*adc_status = 0x01F0; /*FORBIDEN*/
/* Correct use:*/
HAL_WRITE(UINT32,adc_status,0x01F0);
```



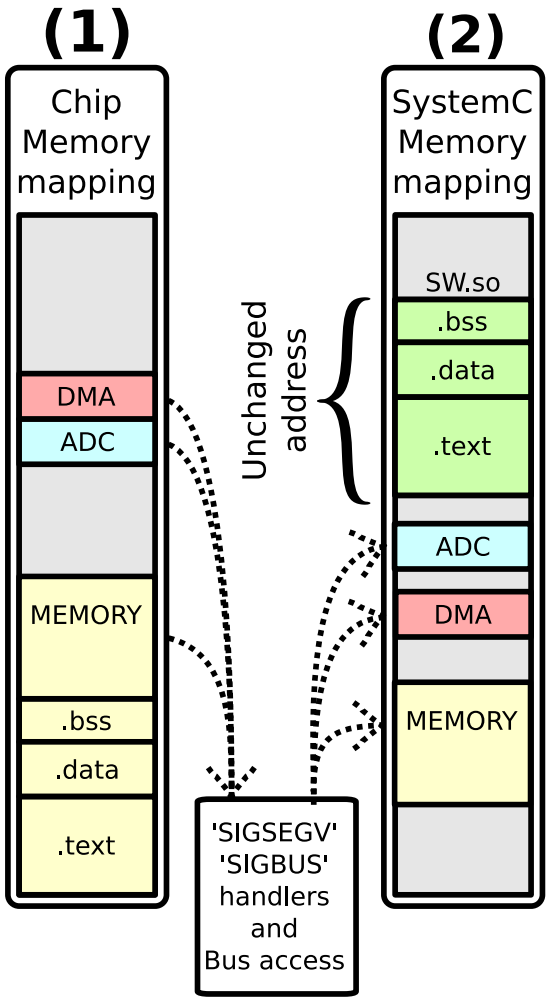
Proposed Solution

Proposed Solution

Use SystemC memory mapping only

Support of pointers to physical memory

- Handlers for exceptions 'SIGSEGV' and 'SIGBUS' are used to remap the address
- The corresponding model for the Bus access is executed



Approach Analysis

Memory representation

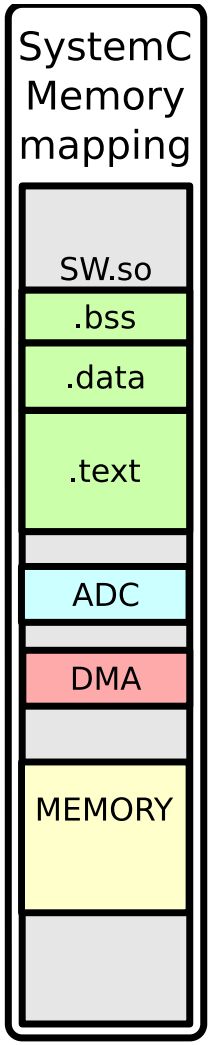
- Accuracy of the memory representation
- Shared and local memories can be modeled

Hardware components implementation

- Accuracy of the components registers mapping
- Meet the requirements of device driver development

Drawback

- Sections and devices registers base addresses are different
- Sizes of some sections are different (e.g. .text, stack)



Approach Analysis

Simulation and Validation

- Allow real processor parallelism (SMP like architectures)
- Simulation speed-up of the native based approaches
- Validation of all the software applications
- Validation of device driver and HDS above the HAL
- Functionnal validation of hardware

Conclusion

A SystemC-base MPSoC implementation methodology

- Flexibility on modeled architectures
- Realistic memory mapping representation
- High simulation speed to allow architecture exploration

Software source code reuse

- Directly cross-compiled and used on target processors
- No modifications needed
- No strict C coding rules, the code just has to respect the HAL API

Future Work

Future Work: Efficient software annotation techniques

- Considering the processor internal architecture and parallelism
- Complex processor type (DSPs)

Future Work: Efficient HdS model generation

- Encapsulating the HdS in IP-XACT
- Extracting the HdS model from the IP-XACT platform model