MPSoC'13

Using Hardware-Assisted Virtualization for Native Simulation of MPSoC

Frédéric Pétrot, Hao Shen, Mian M. Hamayun

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

July 18th, 2013

Native Software Execution In MPSoC Simulation

A Technology that can be used for Rapid Architecture Exploration and Design of MPSoC Systems

Key Ideas of this Talk

- Simulation of hardware/software systems
- Focusing on the fast simulation of software
- Within the model of a hardware environment (SystemC/TLM)
- Support for performance estimation/annotations

Clarification

Host:	machine on which the simulator is executed (e.g. x86)
Host code:	code directly executable on the host
Native code:	code executable on the host once linked with a simulator
Target:	machine which is simulated (e.g. ARM, MIPS)

Motivations/Issues

Why native simulation?

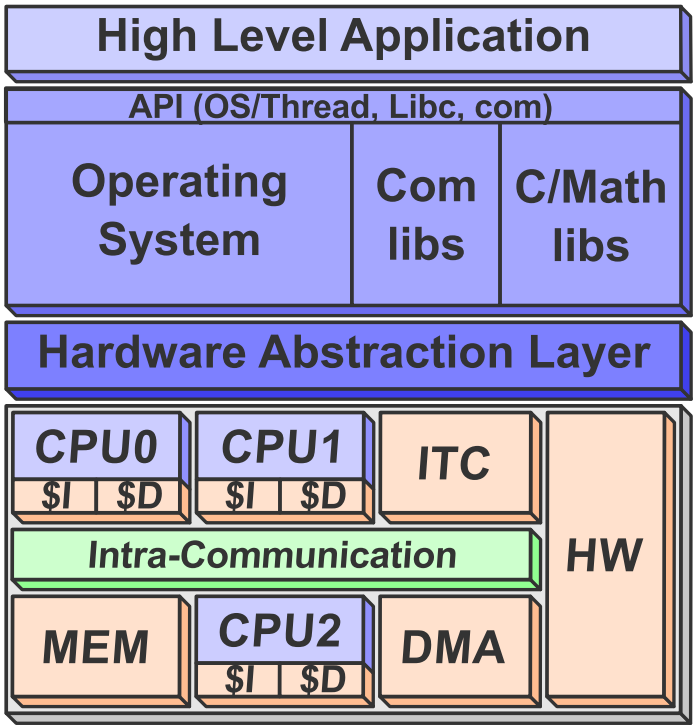
- Software is Compiled for Host ⇒ Fastest Functional Simulations
- Software is Executed Natively ⇒ No need for ISS Development
- Software Executes in *Zero Time* w.r.t H/W ⇒ Functional Verification

Key Issue

- H/W models use target addresses whereas S/W uses host addresses
  - Chip Memory Mapping
  - SystemC Memory Mapping

Requirement: MPSoC platform for realistic native simulation

- Maximize the source code reusability
- Keep low level hardware details



Motivations/Issues

Why native simulation?

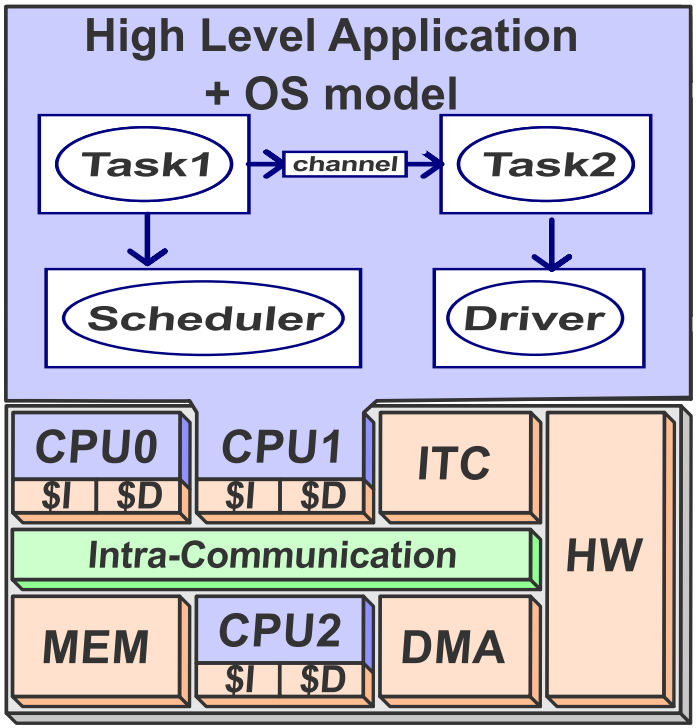
- Software is Compiled for Host ⇒ Fastest Functional Simulations
- Software is Executed Natively ⇒ No need for ISS Development
- Software Executes in *Zero Time* w.r.t H/W ⇒ Functional Verification

Key Issue

- H/W models use target addresses whereas S/W uses host addresses
  - Chip Memory Mapping
  - SystemC Memory Mapping

Requirement: MPSoC platform for realistic native simulation

- Maximize the source code reusability
- Keep low level hardware details



Motivations/Issues

Why native simulation?

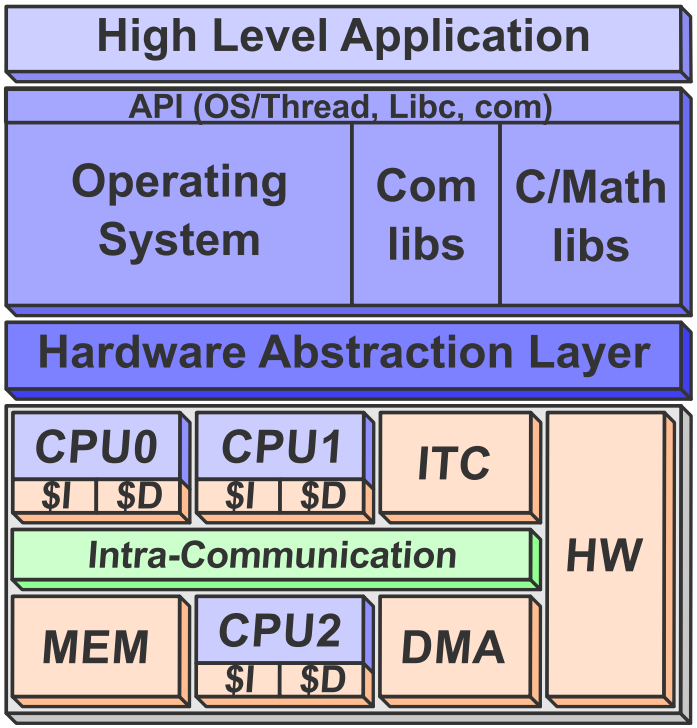
- Software is Compiled for Host ⇒ Fastest Functional Simulations
- Software is Executed Natively ⇒ No need for ISS Development
- Software Executes in *Zero Time* w.r.t H/W ⇒ Functional Verification

Key Issue

- H/W models use target addresses whereas S/W uses host addresses
  - Chip Memory Mapping
  - SystemC Memory Mapping

Requirement: MPSoC platform for realistic native simulation

- Maximize the source code reusability
- Keep low level hardware details



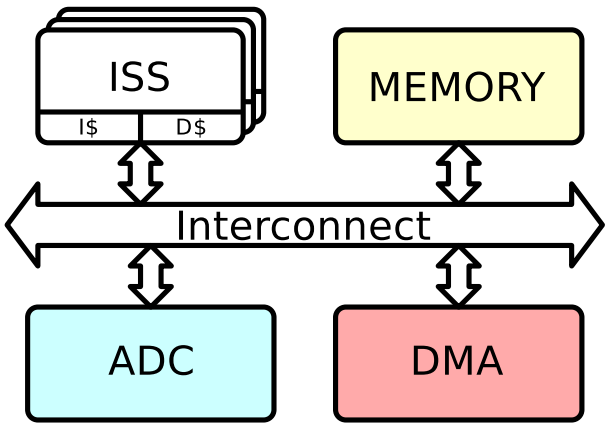
Memory Representations

Real System or CABA Platforms

- SW binary loaded in memory
- HW address decoder uses physical chip memory mapping
- SW uses real chip memory mapping

Transaction Accurate Simulation

- SW dynamically loaded and executed on the Host
- HW address decoder uses physical chip memory mapping
- SW uses SystemC process memory mapping



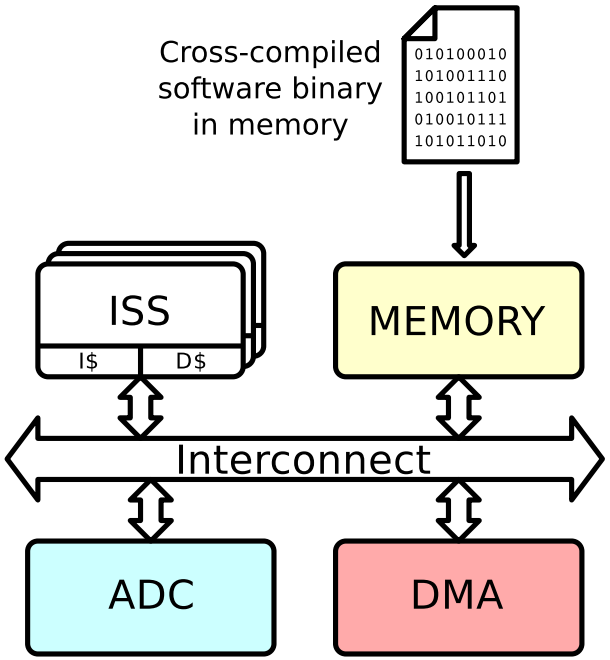
Memory Representations

Real System or CABA Platforms

- SW binary loaded in memory
- HW address decoder uses physical chip memory mapping
- SW uses real chip memory mapping

Transaction Accurate Simulation

- SW dynamically loaded and executed on the Host
- HW address decoder uses physical chip memory mapping
- SW uses SystemC process memory mapping



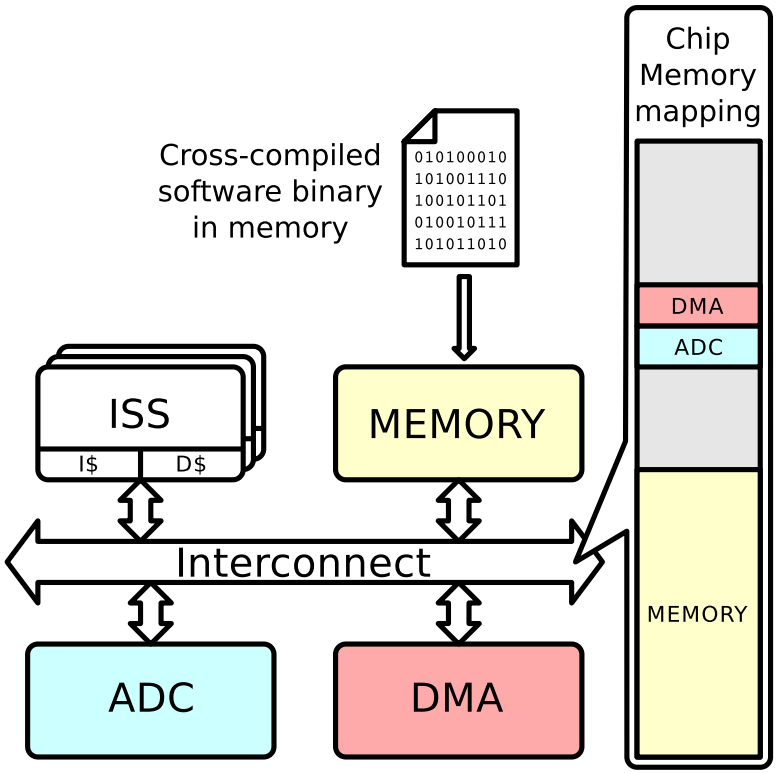
Memory Representations

Real System or CABA Platforms

- SW binary loaded in memory
- HW address decoder uses physical chip memory mapping
- SW uses real chip memory mapping

Transaction Accurate Simulation

- SW dynamically loaded and executed on the Host
- HW address decoder uses physical chip memory mapping
- SW uses SystemC process memory mapping



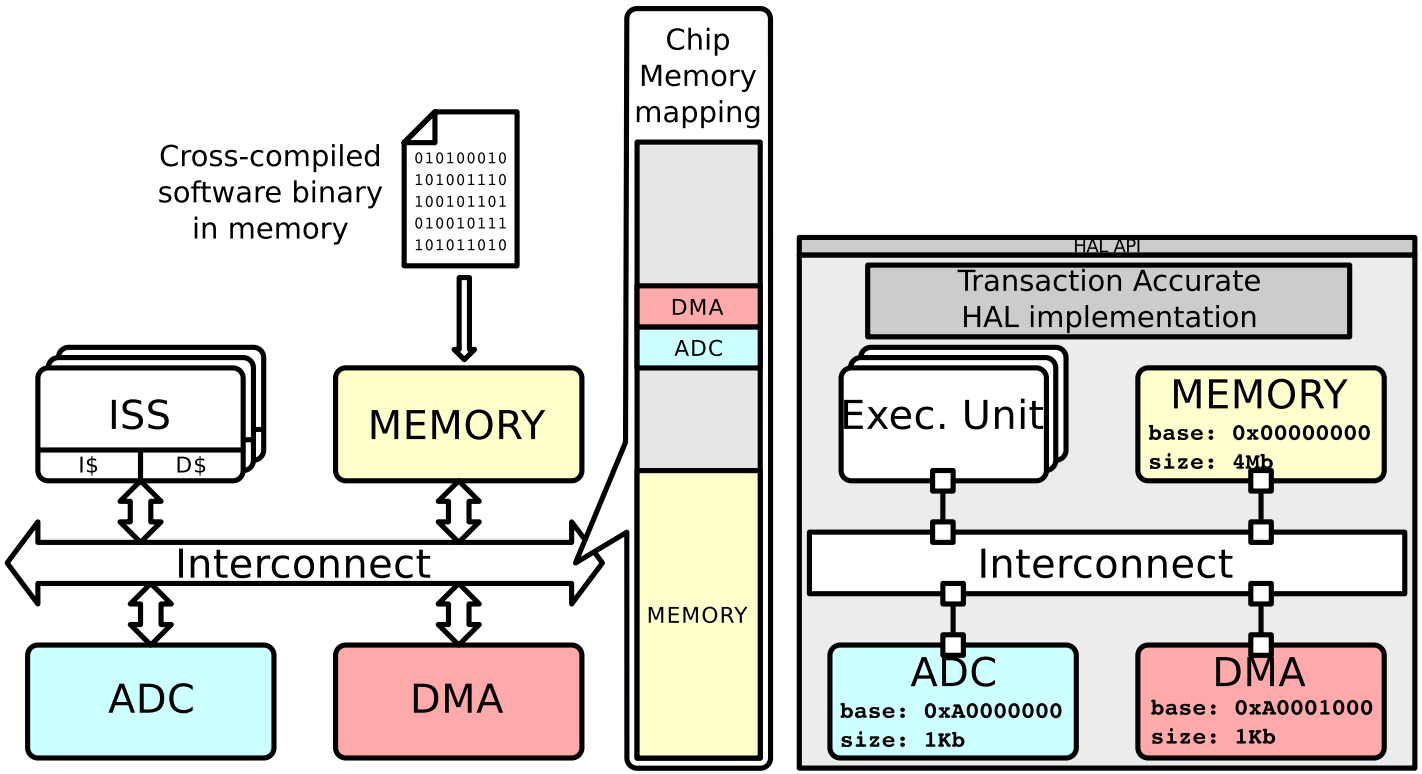
Memory Representations

Real System or CABA Platforms

- SW binary loaded in memory
- HW address decoder uses physical chip memory mapping
- SW uses real chip memory mapping

Transaction Accurate Simulation

- SW dynamically loaded and executed on the Host
- HW address decoder uses physical chip memory mapping
- SW uses SystemC process memory mapping



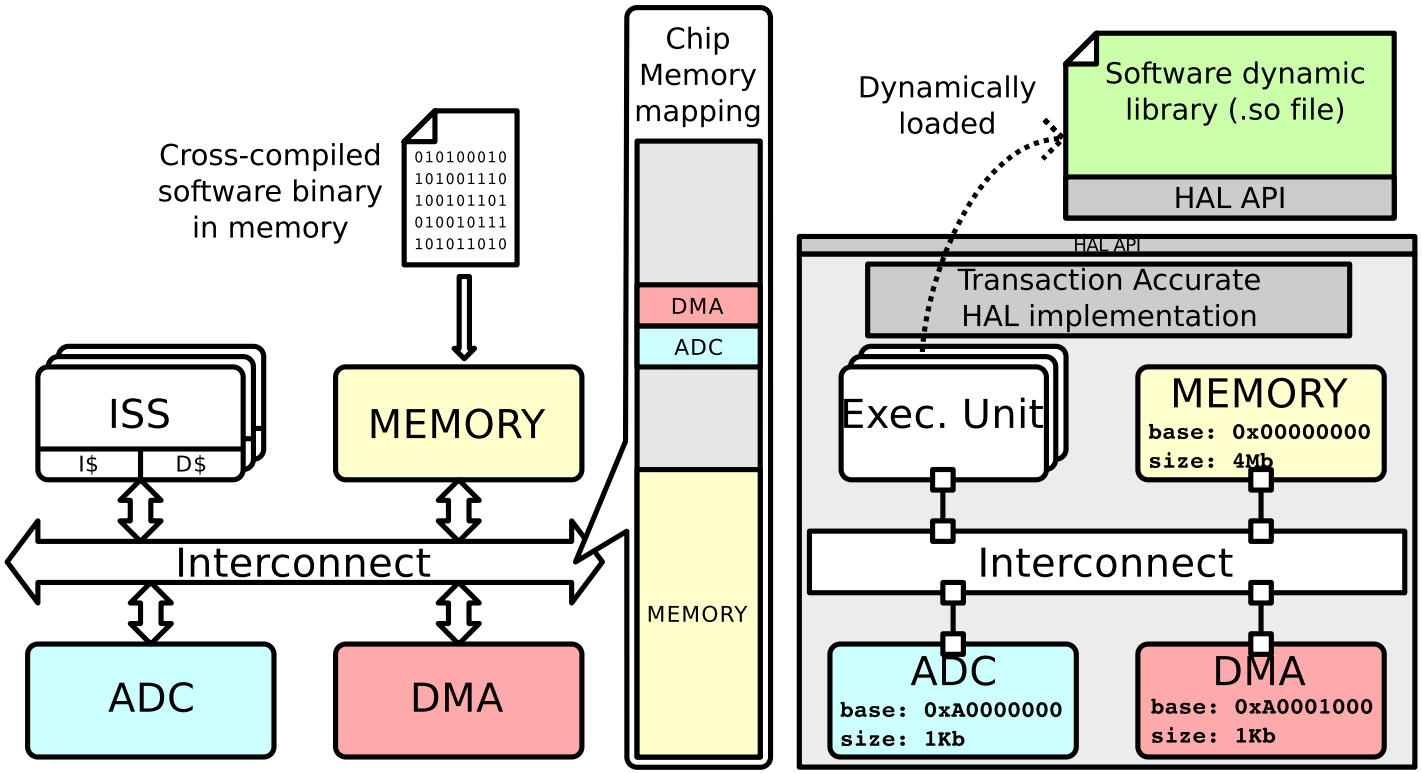
Memory Representations

Real System or CABA Platforms

- SW binary loaded in memory
- HW address decoder uses physical chip memory mapping
- SW uses real chip memory mapping

Transaction Accurate Simulation

- SW dynamically loaded and executed on the Host
- HW address decoder uses physical chip memory mapping
- SW uses SystemC process memory mapping



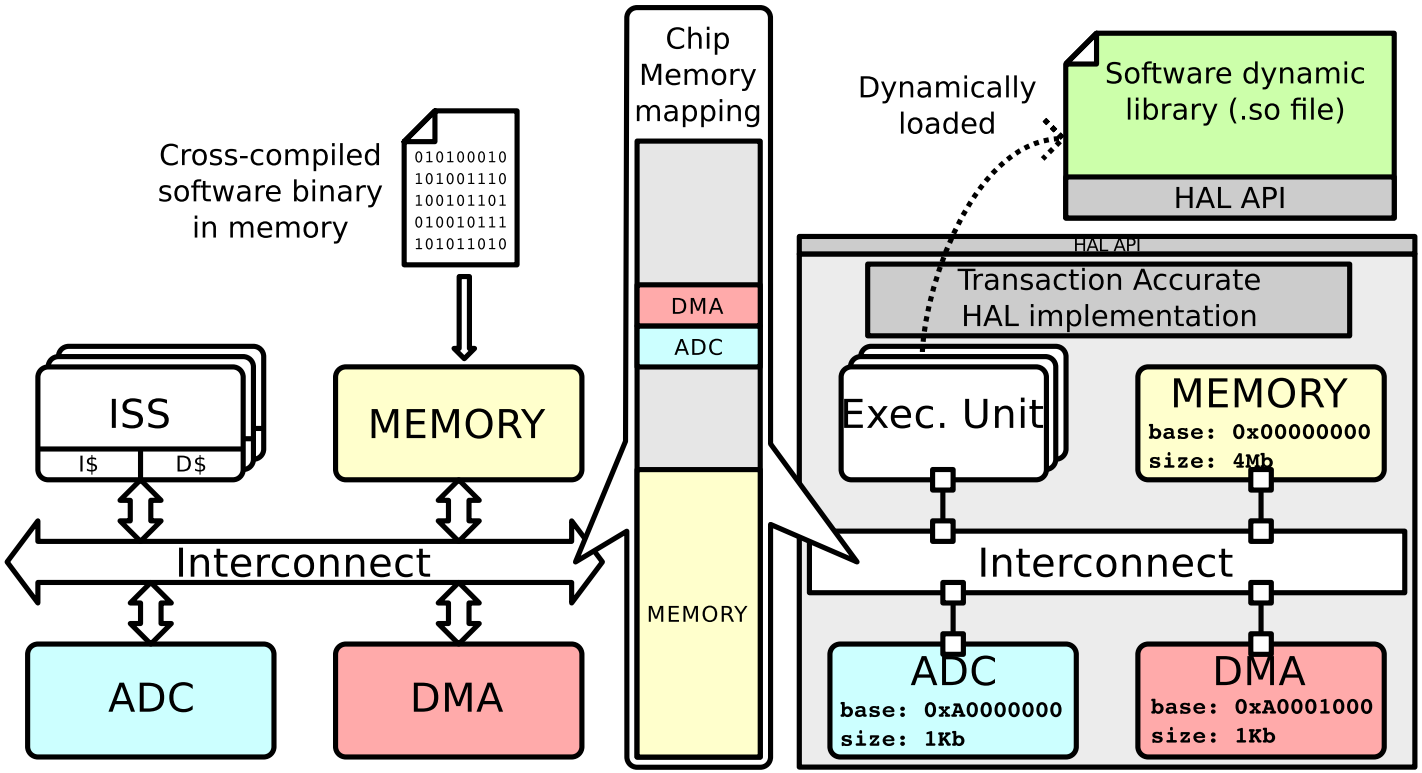
Memory Representations

Real System or CABA Platforms

- SW binary loaded in memory
- HW address decoder uses physical chip memory mapping
- SW uses real chip memory mapping

Transaction Accurate Simulation

- SW dynamically loaded and executed on the Host
- HW address decoder uses physical chip memory mapping
- SW uses SystemC process memory mapping



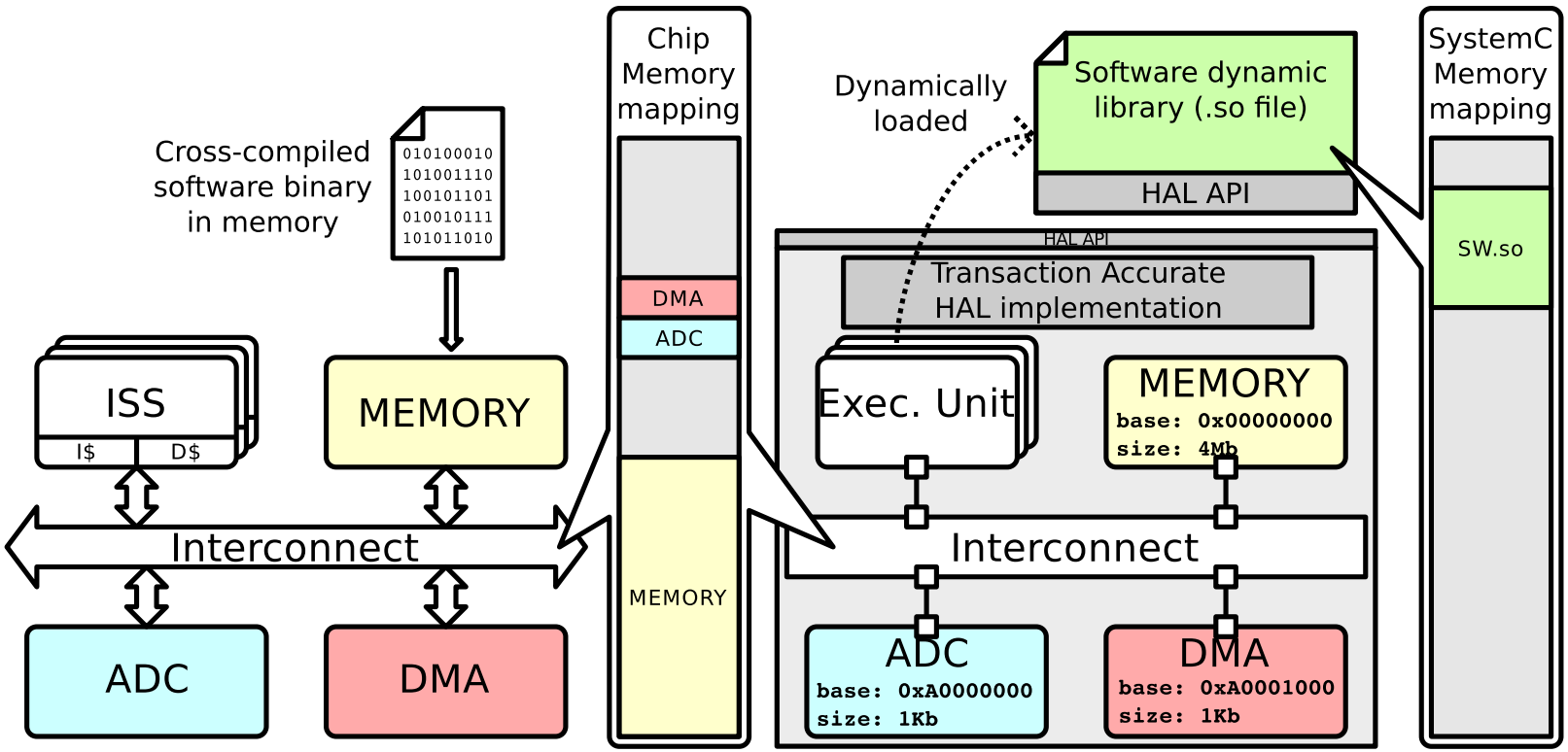
Memory Representations

Real System or CABA Platforms

- SW binary loaded in memory
- HW address decoder uses physical chip memory mapping
- SW uses real chip memory mapping

Transaction Accurate Simulation

- SW dynamically loaded and executed on the Host
- HW address decoder uses physical chip memory mapping
- SW uses SystemC process memory mapping



Problem

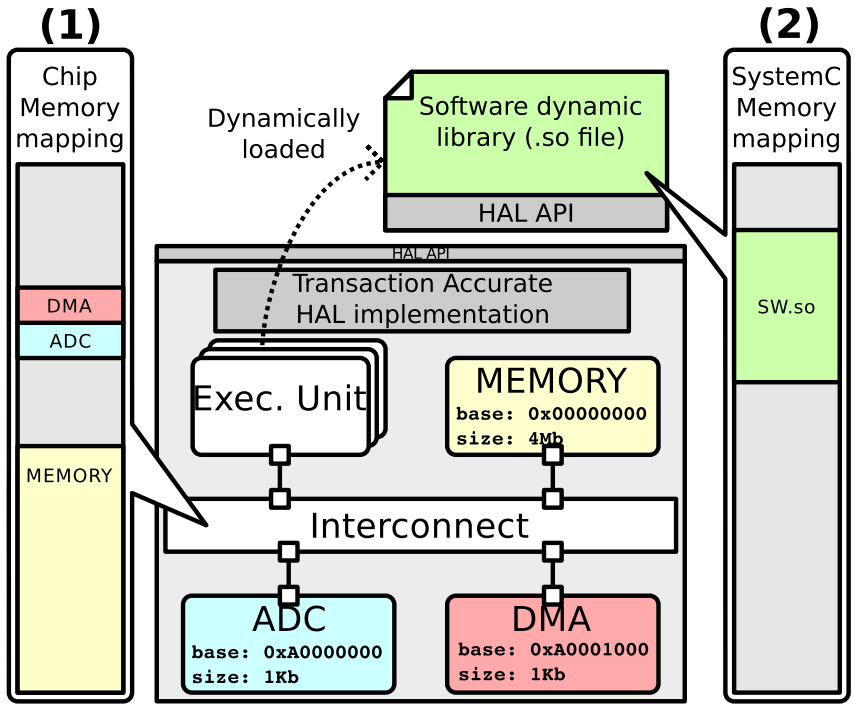
Two uncorrelated memory mappings have to be considered

Chip Memory Mapping (1)

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

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`



Problem

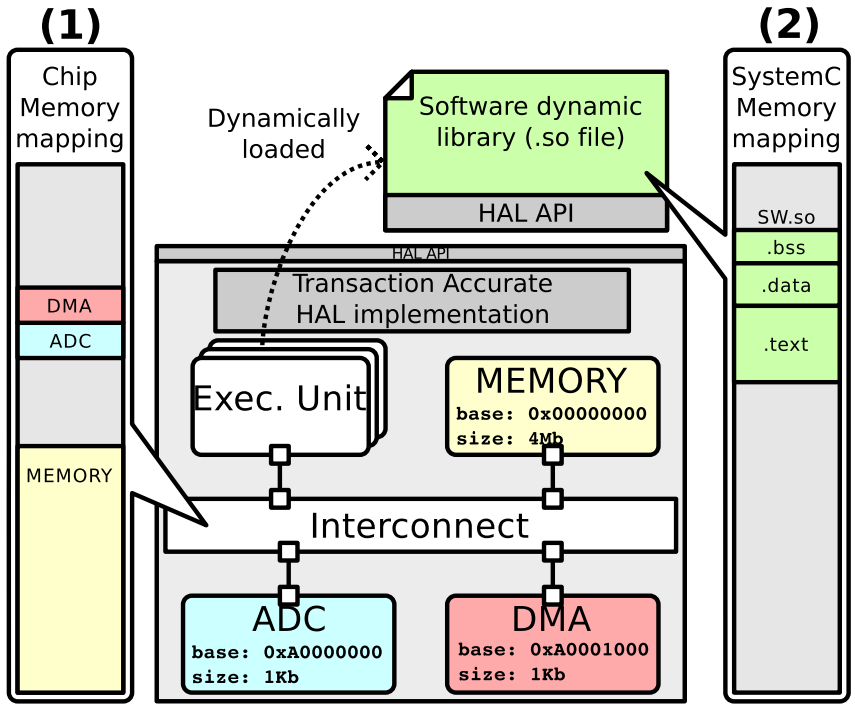
Two uncorrelated memory mappings have to be considered

Chip Memory Mapping (1)

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

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`



Problem

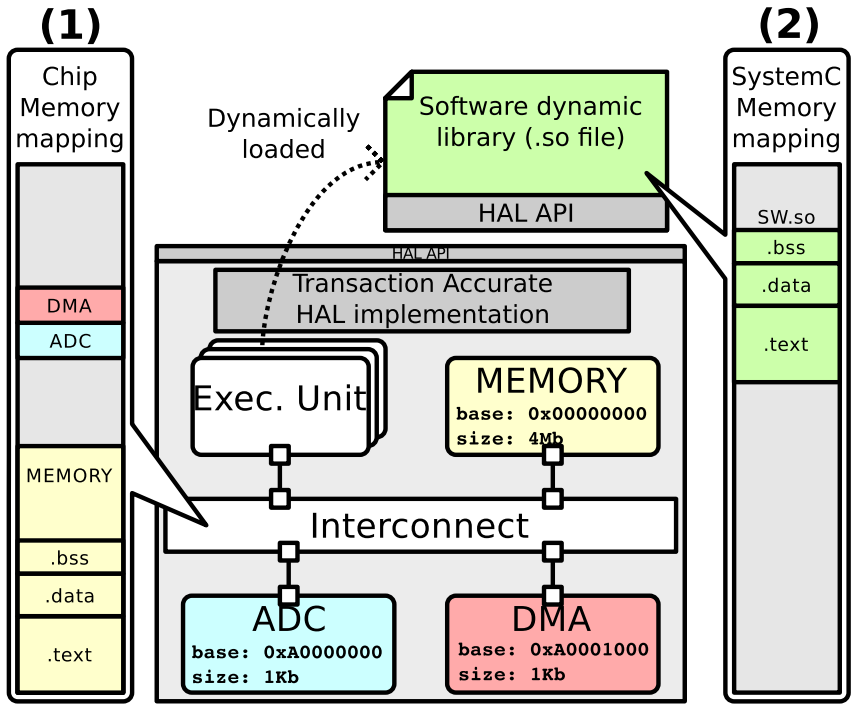
Two uncorrelated memory mappings have to be considered

Chip Memory Mapping (1)

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

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`



Mixing both memory mappings ?

DMA Transfer Example

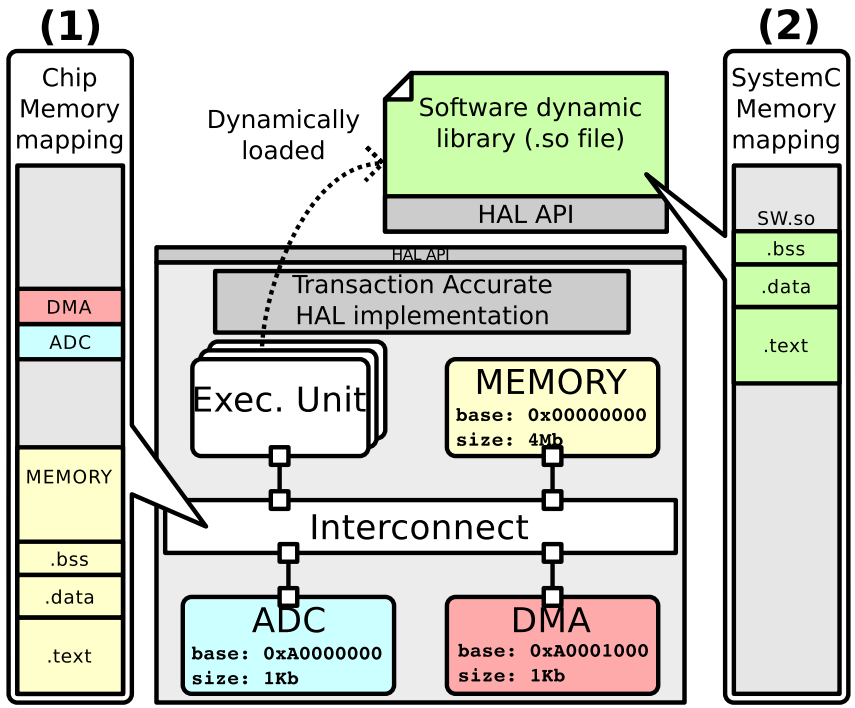
- Read data in the ADC
- Write data into memory

Source in Chip Mapping (1)

- Source address is valid in address decoder
- So DMA can access source address

Destination in SystemC Mapping (2)

- Destination address is valid in SW
- But *Invalid* in the address decoder
- So DMA cannot access the destination address



Mixing both memory mappings ?

DMA Transfer Example

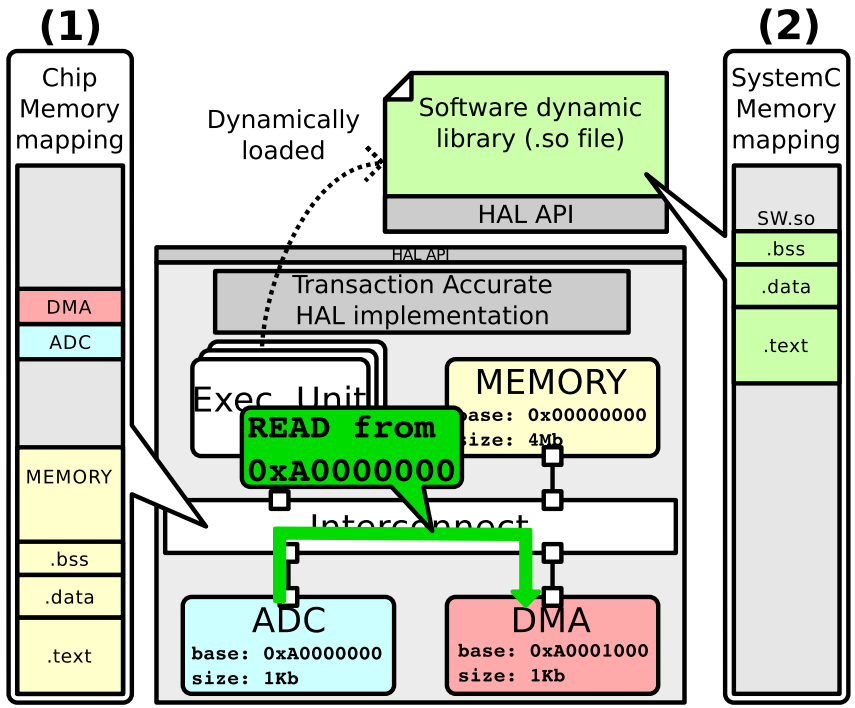
- Read data in the ADC
- Write data into memory

Source in Chip Mapping (1)

- Source address is valid in address decoder
- So DMA can access source address

Destination in SystemC Mapping (2)

- Destination address is valid in SW
- But *Invalid* in the address decoder
- So DMA cannot access the destination address



Mixing both memory mappings ?

DMA Transfer Example

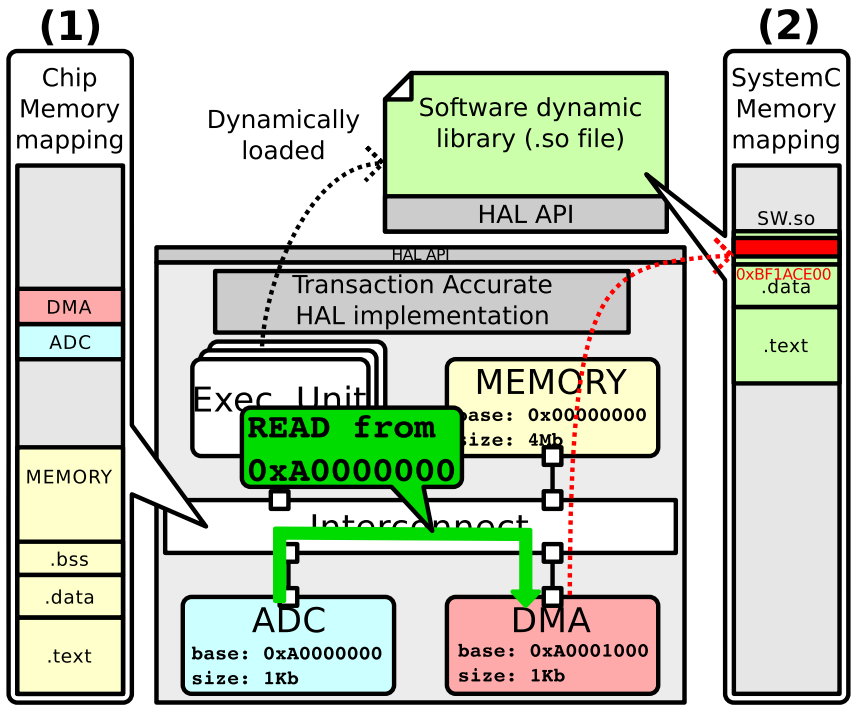
- Read data in the ADC
- Write data into memory

Source in Chip Mapping (1)

- Source address is valid in address decoder
- So DMA can access source address

Destination in SystemC Mapping (2)

- Destination address is valid in SW
- But *Invalid* in the address decoder
- So DMA cannot access the destination address



Mixing both memory mappings ?

DMA Transfer Example

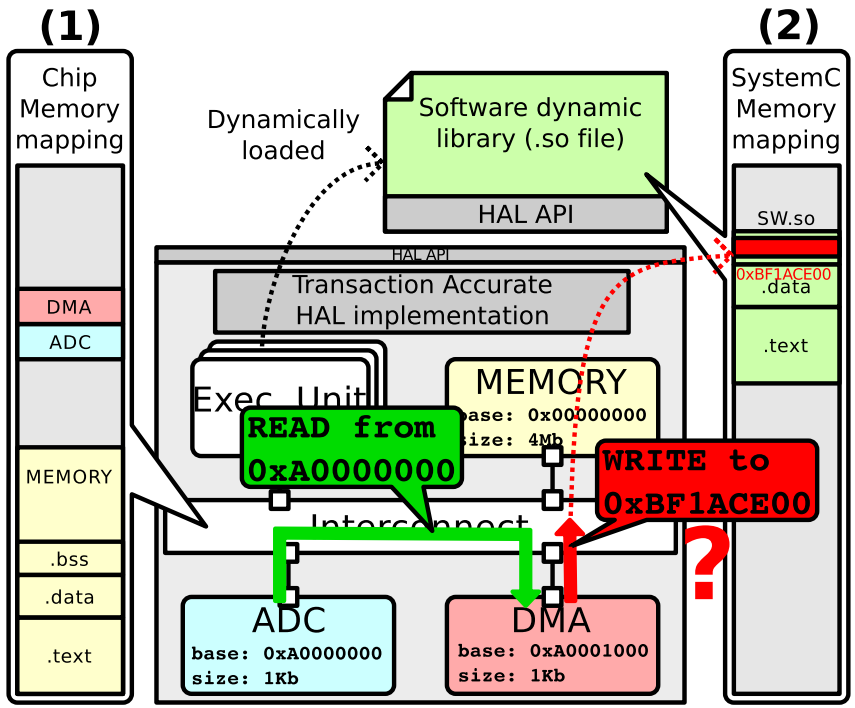
- Read data in the ADC
- Write data into memory

Source in Chip Mapping (1)

- Source address is valid in address decoder
- So DMA can access source address

Destination in SystemC Mapping (2)

- Destination address is valid in SW
- But *Invalid* in the address decoder
- So DMA cannot access the destination address



HW Support for CPU Virtualization

Saved by virtualization's needs in the software industry!

New Guest Operating Mode

- Hardware Support for *Root* and *Non-Root* operations
  - Root Operations for VMM and Non-Root for Guest System
- Hardware Based Guest <-> Host Mode Switching
- Guest Mode Exit Reason Reporting
  - PMIO, MMIO, Signal Pending, Shutdown ?

Available in most popular CPUs since the mid 2000's:
x86 (Intel, AMD), Power, Cortex A15 (ARM), Sparc, ...

New Transitions

- VM Entry and VM Exit
- Swapping of Registers and Address Space in one Atomic Operation

VM Control Structure (VMCS)

- Controlled by software
- Keeps track of Guest OS State
- Controls when VM Exits occur

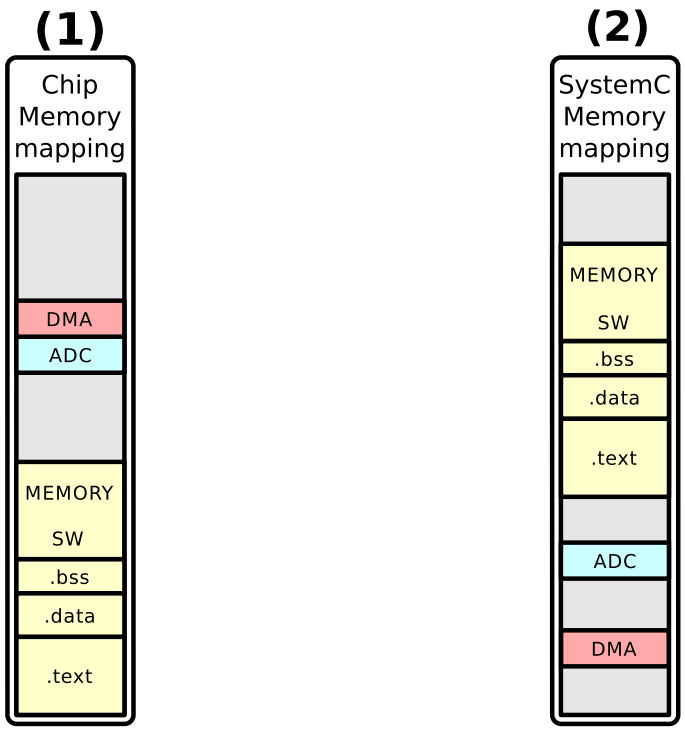
HW Support for Memory Virtualization

HAV based Memory Virtualization

- Software is Compiled as a *Static* binary and executes in Target Address-Space
- SystemC models simulate target addresses and remain un-modified
- Translation layer provides bidirectional accesses between SW and HW components.

Hard-coded addresses in SW

Even hard-coded addresses can be used in Software i.e.
((uint8_t *) 0x0A000010) = 0x33;



Memory Virtualization != Virtual Memory

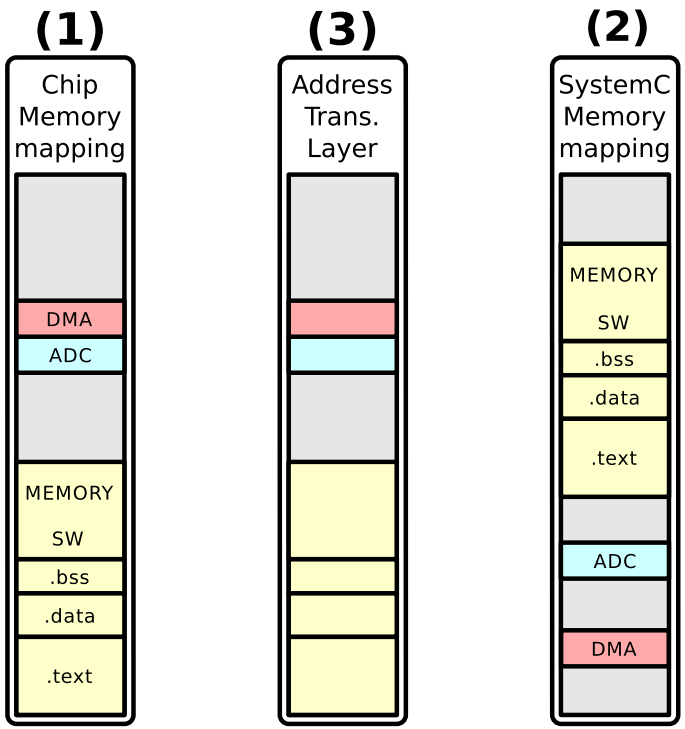
HW Support for Memory Virtualization

HAV based Memory Virtualization

- Software is Compiled as a *Static* binary and executes in Target Address-Space
- SystemC models simulate target addresses and remain un-modified
- Translation layer provides bidirectional accesses between SW and HW components.

Hard-coded addresses in SW

Even hard-coded addresses can be used in Software i.e.
((uint8_t *) 0x0A000010) = 0x33;



Memory Virtualization != Virtual Memory

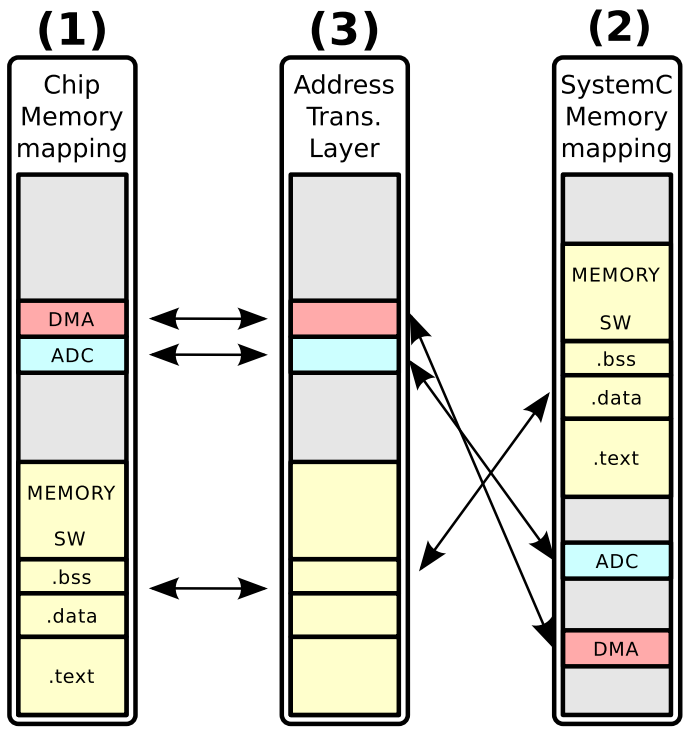
HW Support for Memory Virtualization

HAV based Memory Virtualization

- Software is Compiled as a *Static* binary and executes in Target Address-Space
- SystemC models simulate target addresses and remain un-modified
- Translation layer provides bidirectional accesses between SW and HW components.

Hard-coded addresses in SW

Even hard-coded addresses can be used in Software i.e.
((uint8_t *) 0x0A000010) = 0x33;



Memory Virtualization != Virtual Memory

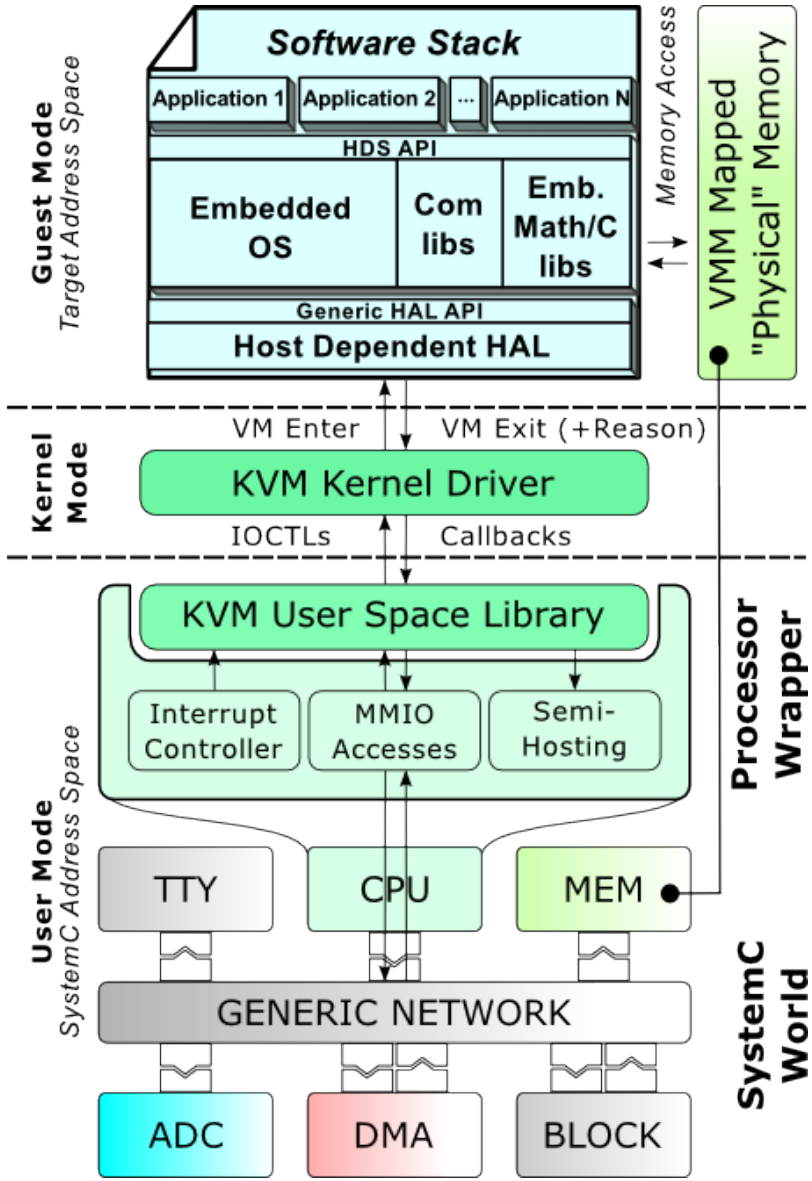
Using HAV in Event Driven Simulation

Transparent Memory Accesses

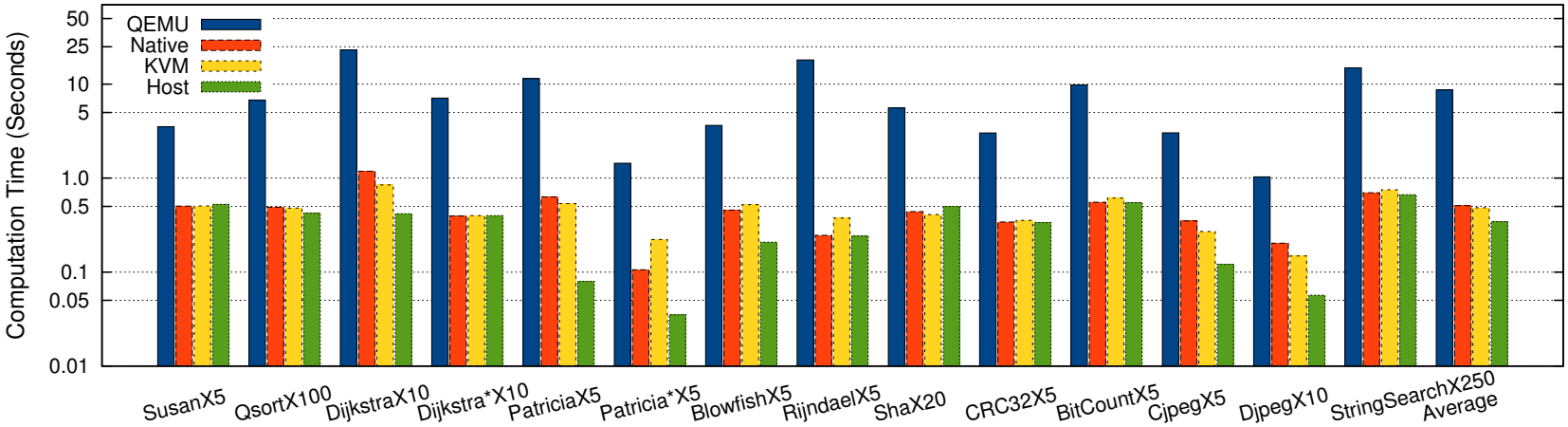
- Using the Virtual MMU implementation in KVM
- Full address space available in Guest mode
- Transparent access i.e. No mode switch on memory accesses

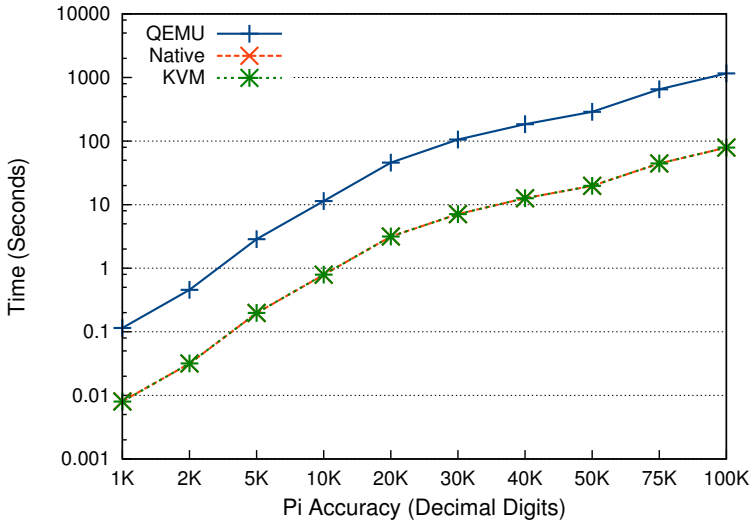
I/O Emulation

- I/O (MMIO & PMIO) accesses force VM Exits
- We exploit PMIO exits for providing Semi-hosting support. e.g. Annotations



Computation Speed Comparisons

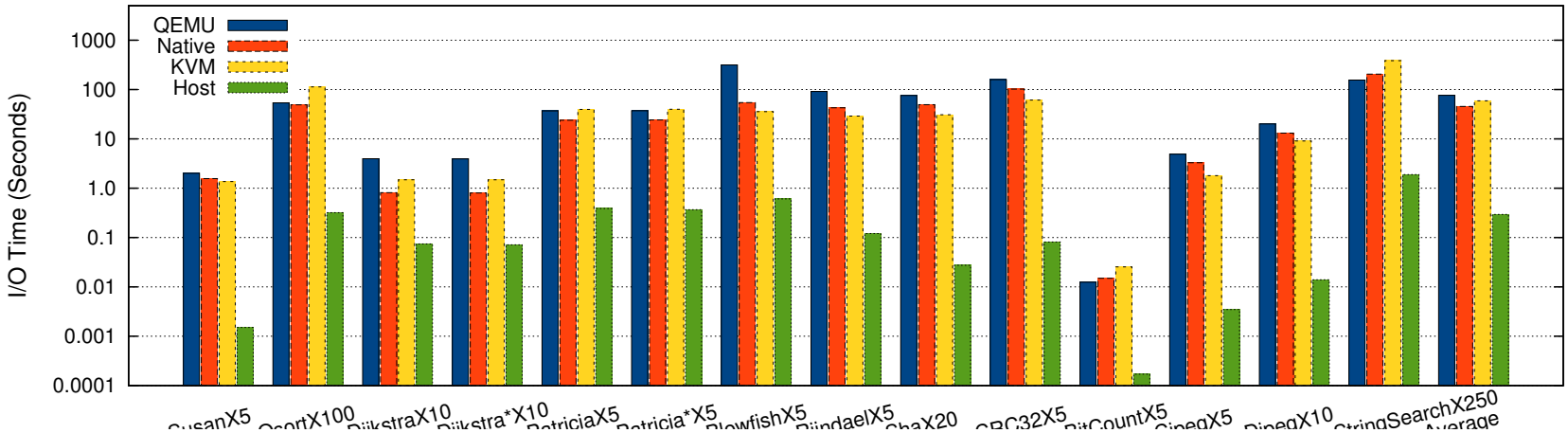


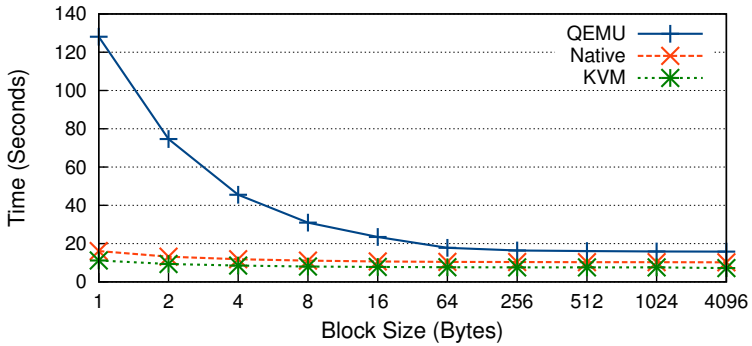


Simulation Platform	Best-Case	Worst-Case	Total Time
	Rijndael	Djpeg	All Applications
QEMU	18.081s	1.033s	104.099s
KVM	0.376s	0.148s	5.814s
Speedup/Slowdown	**48.10X**	**6.96X**	**17.91X**

	Dijkstra	Rijndael	All Applications
Native	1.185s	0.246s	6.104s
KVM	0.846s	0.376s	5.814s
Speedup/Slowdown	**1.40X**	**0.66X**	**1.05X**

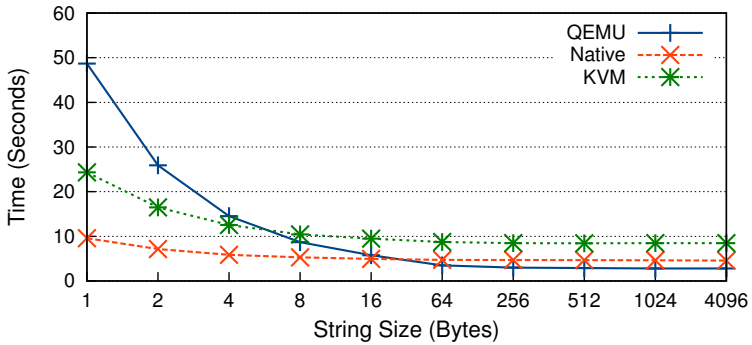
I/O Speed Comparisons & Slowdown w.r.t Direct Host





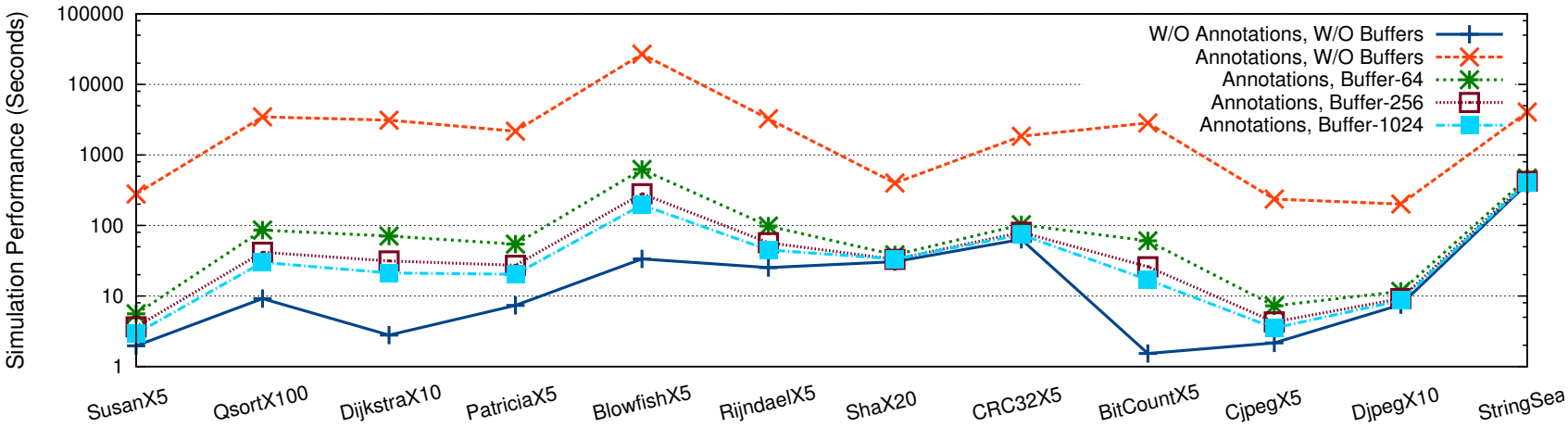
Simulation Platform	Best-Case	Worst-Case	Total Time
	Blowfish	StringSearch	All Applications
QEMU	314.190s	155.247s	916.112s
KVM	35.768s	385.659s	708.708s
Speedup/Slowdown	**8.78X**	**0.40X**	**1.29X**

	Cjpeg	Qsort	All Applications
Native	3.271s	48.946s	545.136s
KVM	1.801s	113.054s	708.708s
Speedup/Slowdown	**1.82X**	**0.43X**	**0.77X**



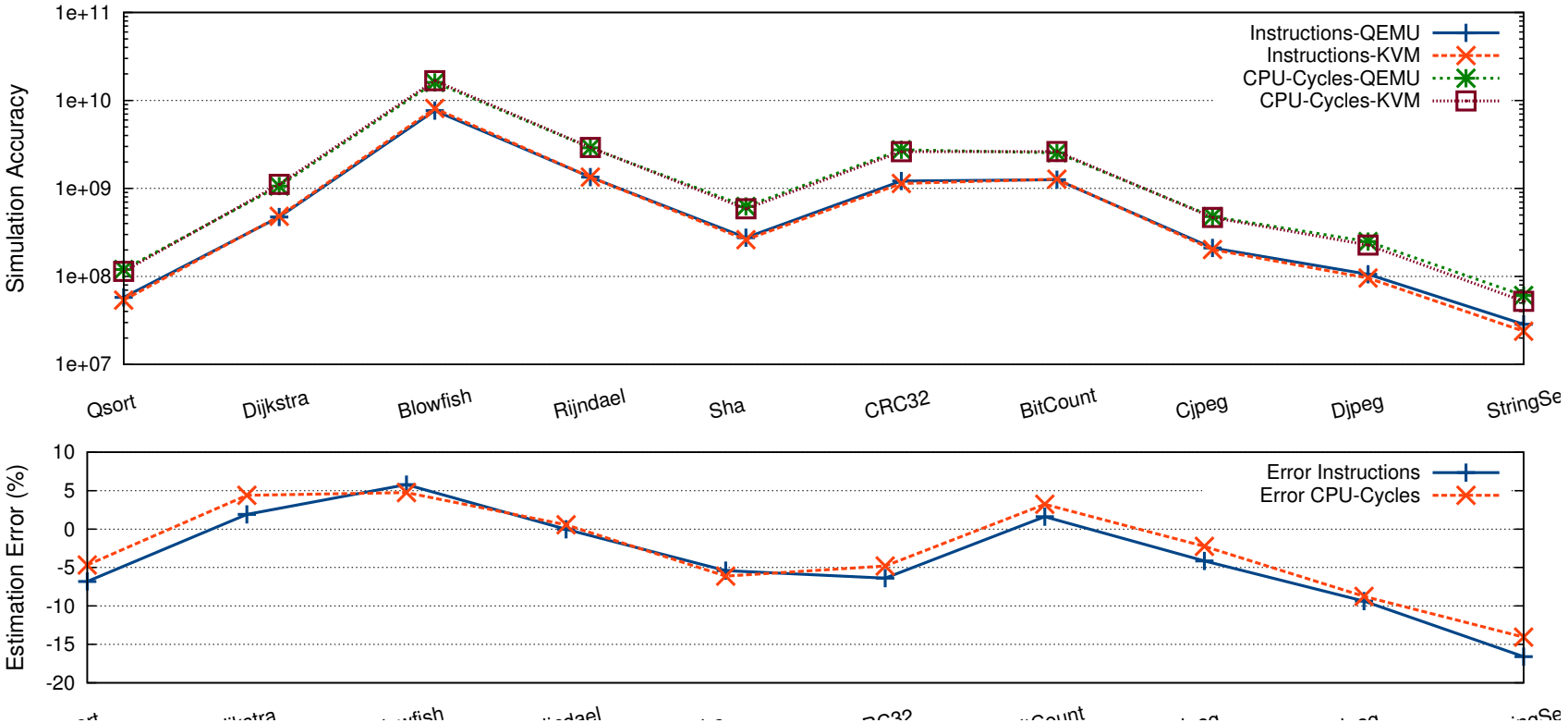
	Host	QEMU	Native	KVM
Comp. Time	4.127s	104.099s	6.104s	5.814s
Comp. Slowdown	**1X**	**25.23X**	**1.48X**	**1.41X**
I/O Time	3.528s	916.112s	545.136s	708.708s
I/O Slowdown	**1X**	**259.69X**	**154.53X**	**200.90X**
Total Time	7.654s	1020.211s	551.240s	714.522s
Total Slowdown	**1X**	**133.28X**	**72.02X**	**93.35X**

Simulation Performance with S/W Annotations



Buffer Size	Best-Case	Worst-Case	Avg. Slowdown
	StringSearch	BitCount	All Apps
0	10.38X	1862.38X	84.56X
64	1.20X	39.69X	2.83X
256	1.08X	17.00X	1.77X
1024	1.04X	11.12X	1.49X

Accuracy of S/W Annotations in Native Simulation



Error Type	Best-Case	Worst-Case	Average Error
	Rijndael	StringSearch	All Applications
Error Instructions	-0.02%	-16.60%	-3.95%
Error CPU-Cycles	+0.55%	-14.06%	-2.77%
Abs. Error Instructions	0.02%	16.60%	5.80%
Abs. Error CPU-Cycles	0.55%	14.06%	5.36%

Conclusions and Future Works

Conclusions

- HAV provides a novel way of implementing native simulation
- Memory virtualization solves the conflicting address-spaces issue
- Support for complex MPSoC architectures and legacy software is possible as shared memories can be modeled tranparently
- Simulation performance very close to previous native solutions

Limitations

- I/O performance is a bottle-neck, as all I/O requests must trap
- S/W debugging support is currently not available in KVM

Future Directions

- Finding a solution to minimize the Guest-to-Host transitions
- Devising a translation scheme for complex architecture simulation e.g. VLIW Machines
- Improving the annotation technique to increase estimation accuracy

Related Publications

1. Mian-Muhammad Hamayun, Frédéric Pétrot and Nicolas Fournel. Native simulation of complex VLIW instruction sets using static binary translation and Hardware-Assisted Virtualization. In *Proceedings of the 18th Asia and South Pacific Design Automation Conference*, pages 576-581, 2013
2. Hao Shen, Mian-Muhammad Hamayun, and Frédéric Pétrot. Native Simulation of MPSoC Using Hardware-Assisted Virtualization *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, Vol. 31, n° 7, pages 1074-1087, 2012
3. Patrice Gerin, Mian Muhammad Hamayun, and Frédéric Pétrot. Native mpsoc co-simulation environment for software performance estimation. In *Proceedings of the 7th International Conference on Hardware/Software Codesign and System Synthesis*, pages 403–412, 2009.
4. Patrice Gerin, Xavier Guerin, and Frédéric Pétrot. Efficient Implementation of Native Software Simulation for MPSoC. In *Proceedings of Design, Automation and Test in Europe*, pages 676-681, 2008.

Questions & Answers

Thanks for Your Attention

Questions ?