carbon
design systems

Assembling and Debugging VPs of Complex Cycle Accurate Multicore Systems

July 2009

SOFTWARE BEFORE SILICON

Model Requirements in a Virtual Platform

- Control
  - initialization, breakpoints, etc
- Visibility
  - PV registers, memories, profiling
- Compatibility
  - support common methods with debuggers

All Models Must Support Same Flow as PV Models

Why Cycle Accurate RTL Based Models?

- RTL based models are the only way to insure accurate system level throughput and functionality
- Using RTL based models quickly increase the confidence of architectural decisions.
- RTL based models leverage virtual platform development done earlier during the program.

Issues with RTL models in VPs

- Debugger Integration
- Mapping PV Registers & Memories
- Port & Interface Abstraction

Debugger Integration Though HW

- Debugger commands via HW port
- Invasive & disrupts program flow
- Changes bus & system behavior
- Possible performance issues depending on number of registers & memory reads
- Simulator still “running”
- Limited breakpoint capability & control

Debugger Integration via VP

- Unaltered Program Flow
- Same system activity: Bus transactions, memory accesses, etc.
- Stall pipeline at specific Program Counter
- Upon completing outstanding operations, halt simulator and enable Debugger interaction
- Registers and memory subsystem in determinable state. No need to run simulator clock
- Required to integrate into PV virtual platform

Where's the challenge?

Typical pipeflow behavior for an ISS model versus concurrent behavior present in RTL

Problem: How to integrate a Debugger and be at a valid instruction boundary?

Legend

P1:	Instruction Prefetch 1
P2:	Instruction Prefetch 2
D:	Decode
A:	Address Generation/Instruction Issue
X:	Integer Execution
W:	WriteBack
L1:	L1 Access
L2:	L2 Access
M1:	Multiply Stage 1
M2:	Multiply Stage 2
Q:	Execution Unit Instruction Q
ACC:	Accumulate

ISS Behavior

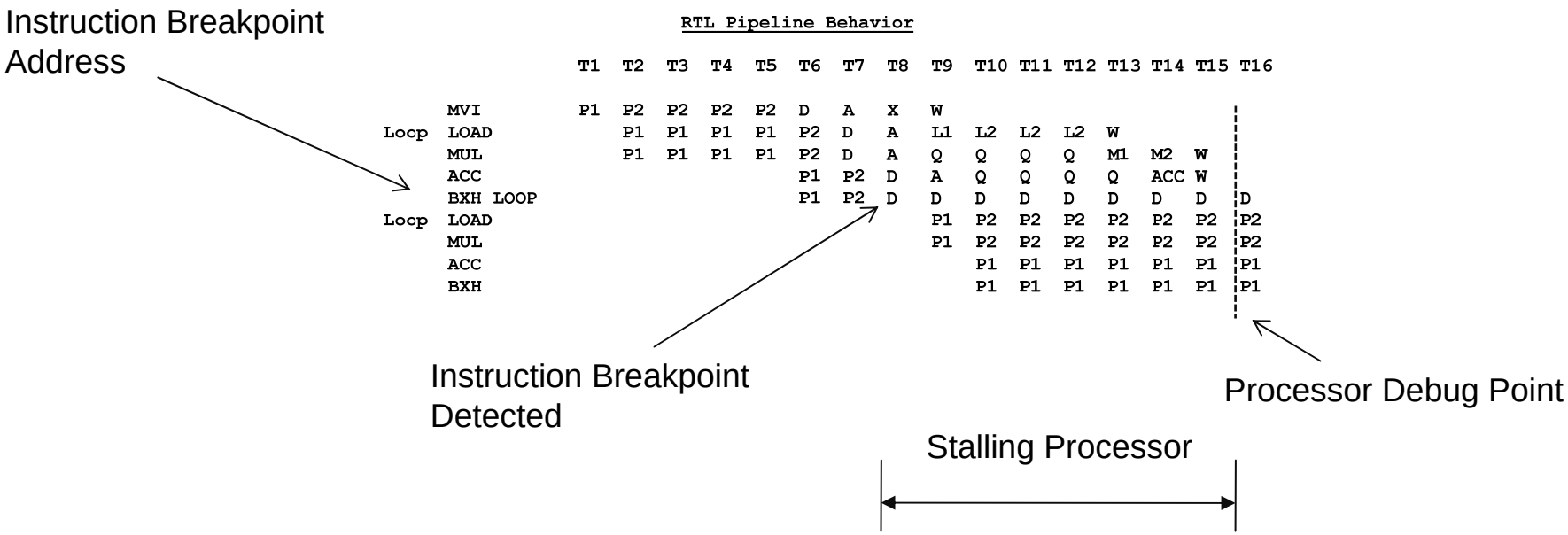
		T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
	MVI	X									
Loop	LOAD		X								
	MUL			X							
	ACC				X						
	BCH LOOP					X					
Loop	LOAD						X				
	MUL							X			
	ACC								X		
	BCH LOOP									X	
	ST										X

RTL Pipeline Behavior

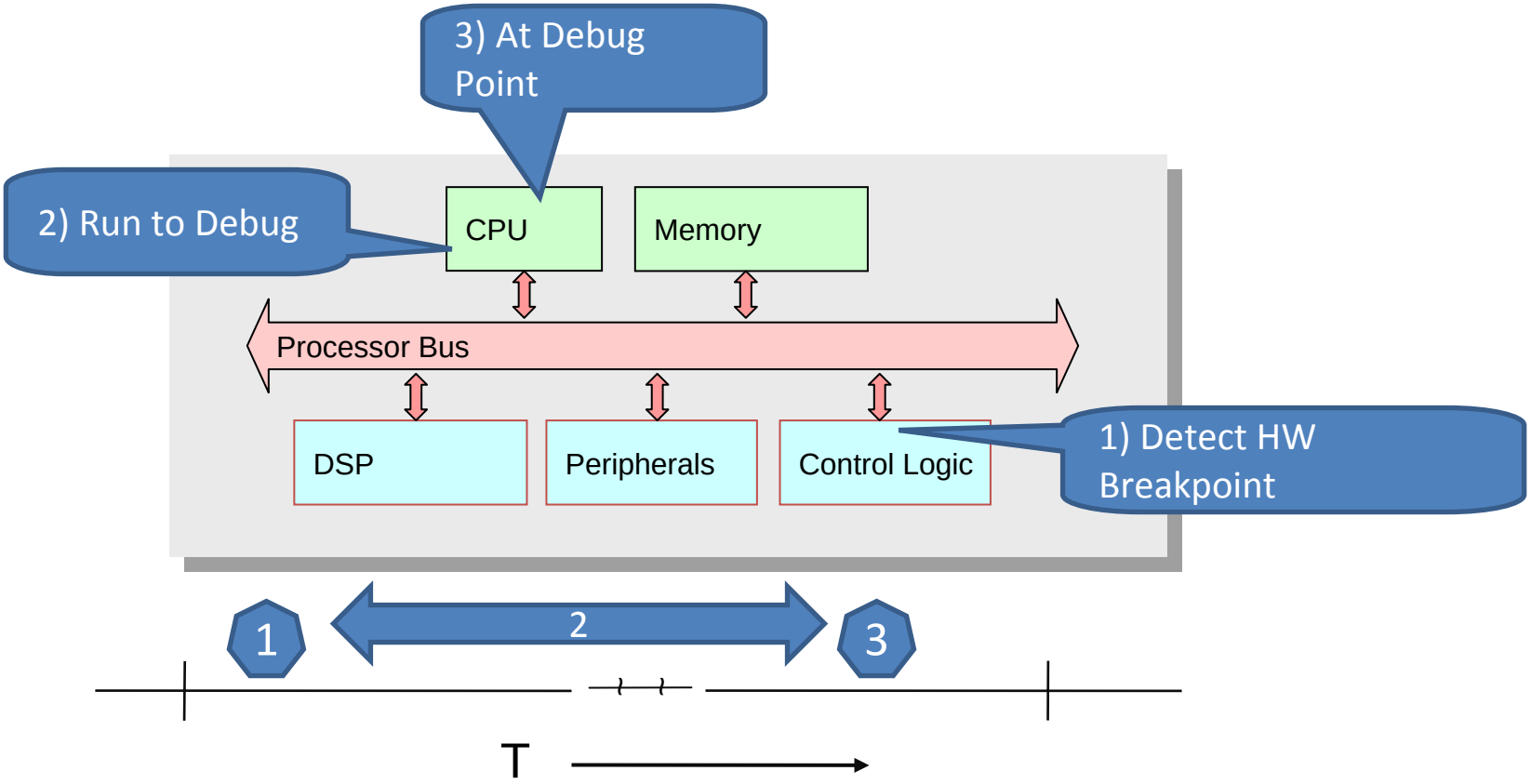
		T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20
	MVI	P1	P2	P2	P2	P2	D	A	X	W											
Loop	LOAD		P1	P1	P1	P1	P2	D	A	L1	L2	L2	L2	W							
	MUL		P1	P1	P1	P1	P2	D	A	Q	Q	Q	Q	M1	M2	W					
	ACC						P1	P2	D	A	Q	Q	Q	Q	ACC	W					
	BXH LOOP						P1	P2	D	A	X	W									
Loop	LOAD									P1	P2	D	A	L1	W						
	MUL									P1	P2	D	A	Q	M1	M2	W				
	ACC										P1	P2	D	A	Q	ACC	W				
	BXH										P1	P2	D	A	X	W					
	LOAD											~~P1~~	~~P2~~	~~D~~	~~A~~						
	MUL											~~P1~~	~~P2~~	~~D~~	~~A~~						
	ST															P1	P2	D	A	L1	W

Reaching a “Debug Point”

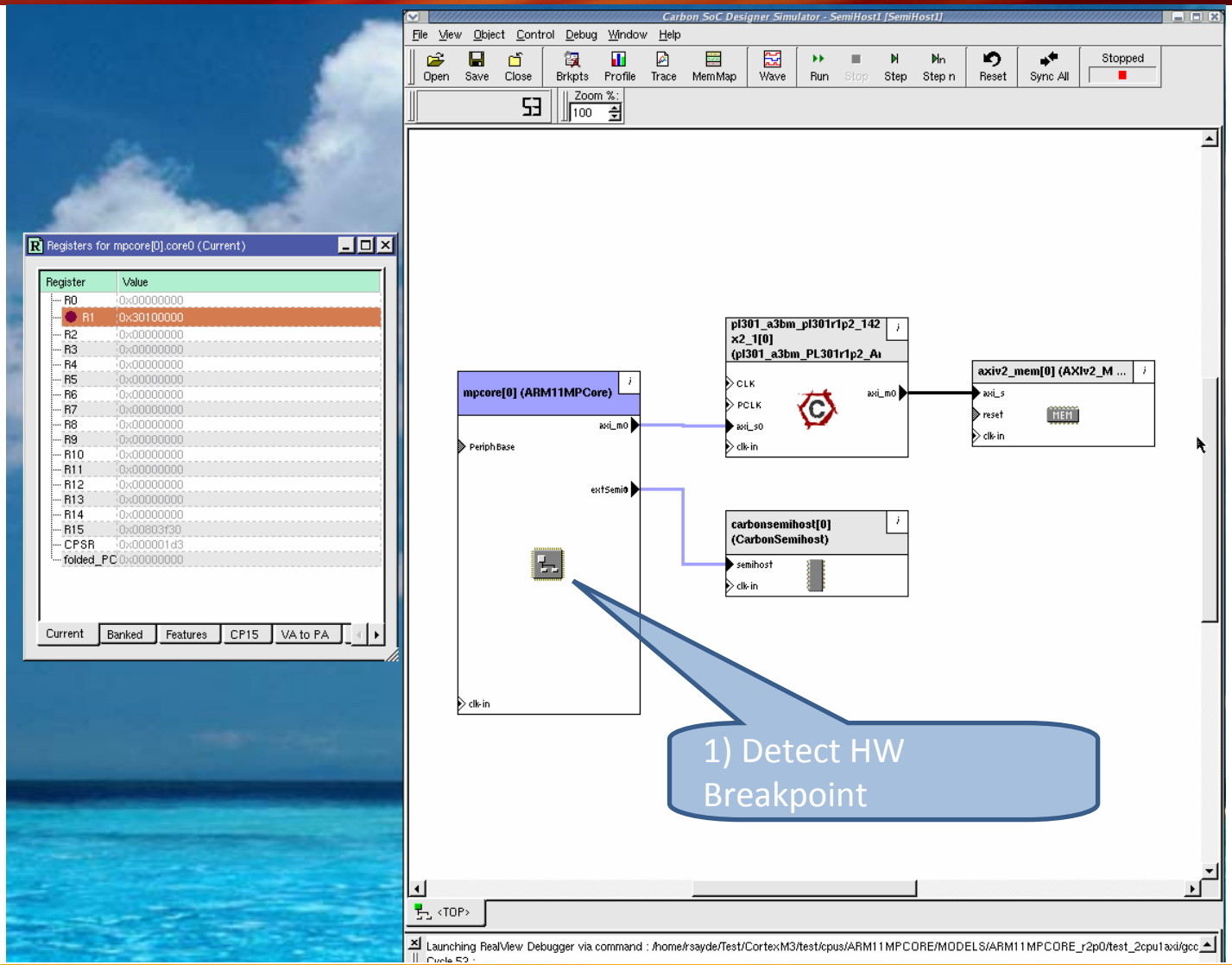
- Enable debugger interaction via “stalling” processor
- May be used to “trap” other functions (e.g., ARM’s semi-hosting function)
- Breakpoints are “extendable” with arbitrary delays and functions



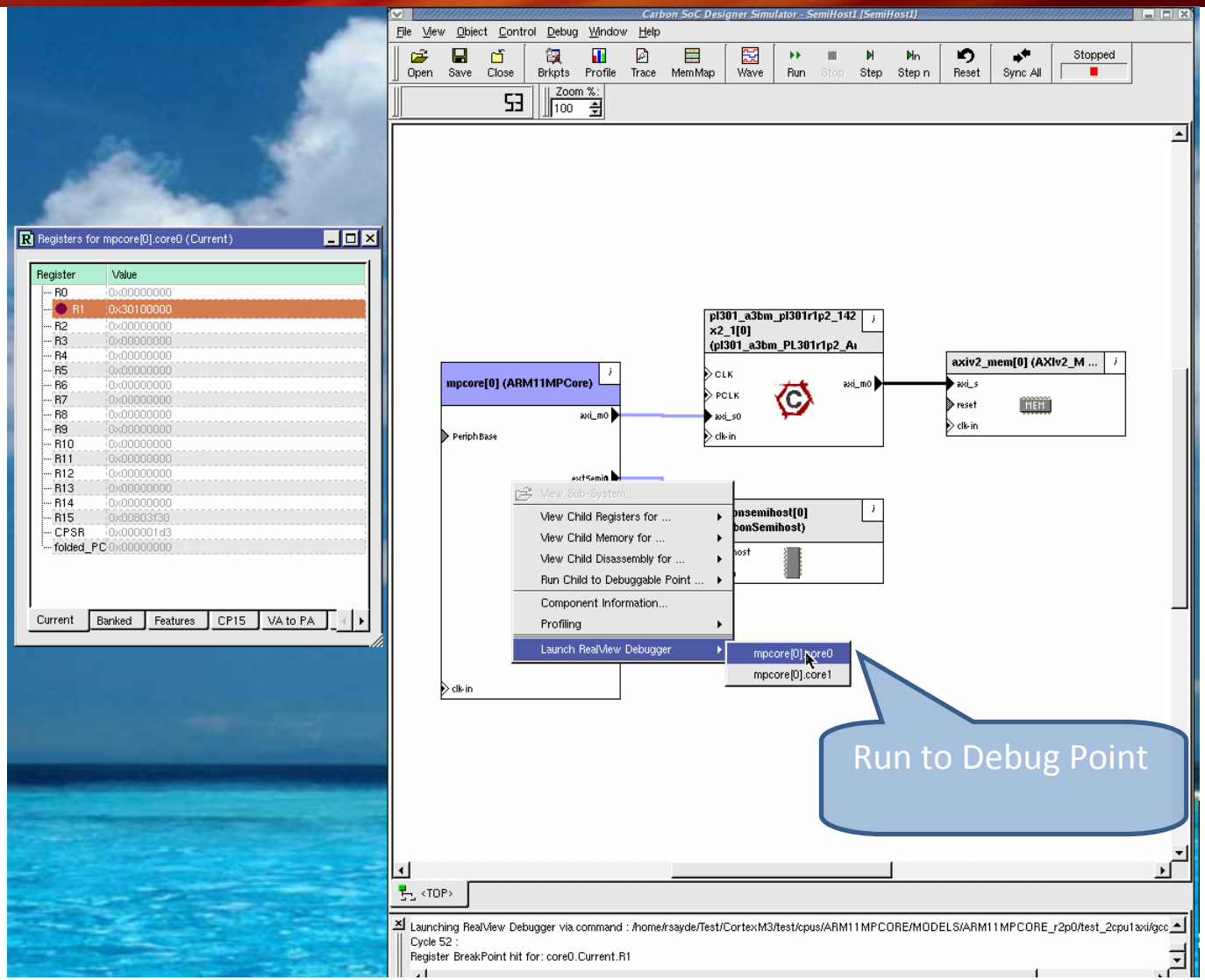
HW / SW Debugging



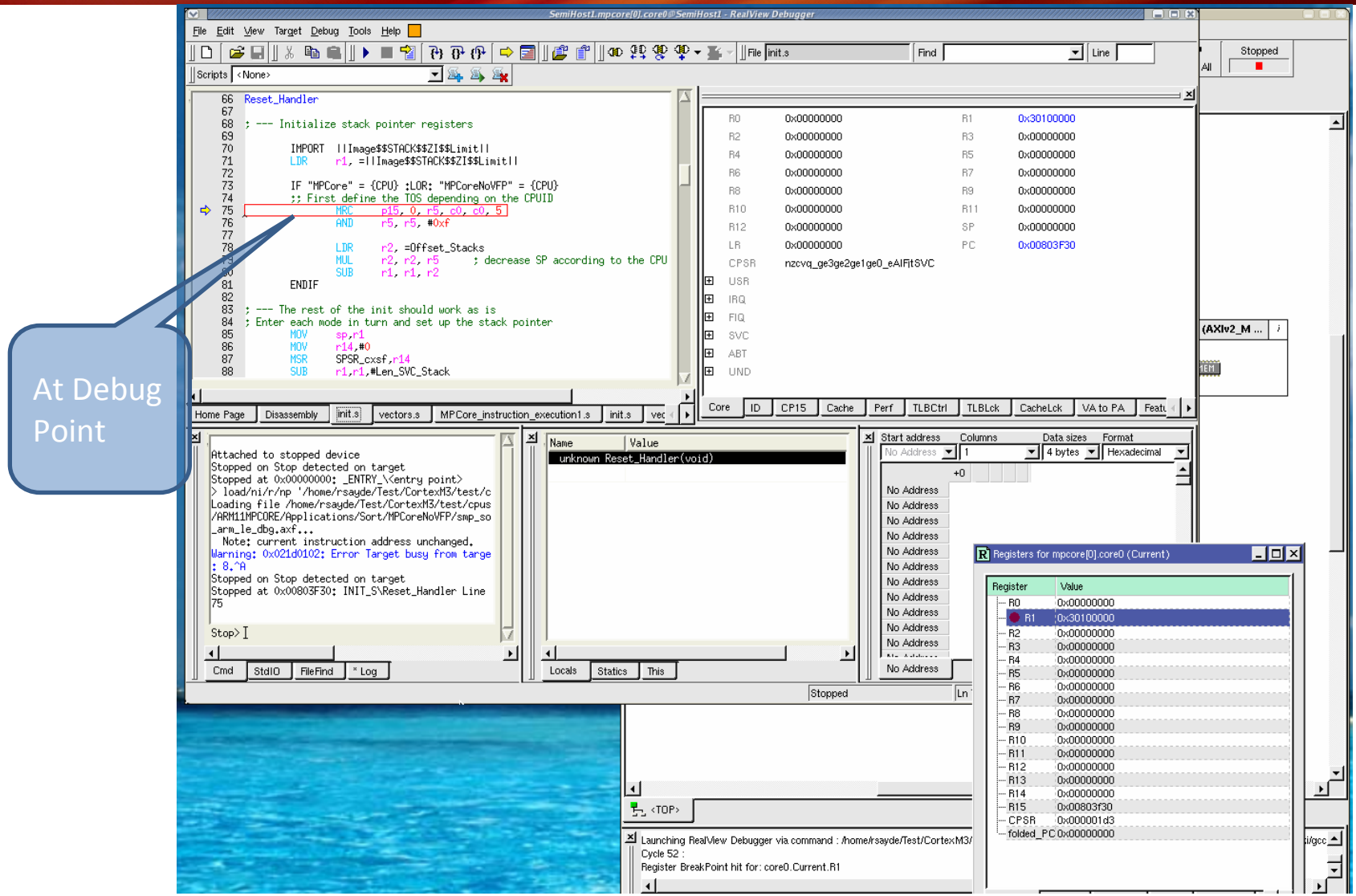
HW Breakpoint



Instruct Core0 to Debug Point



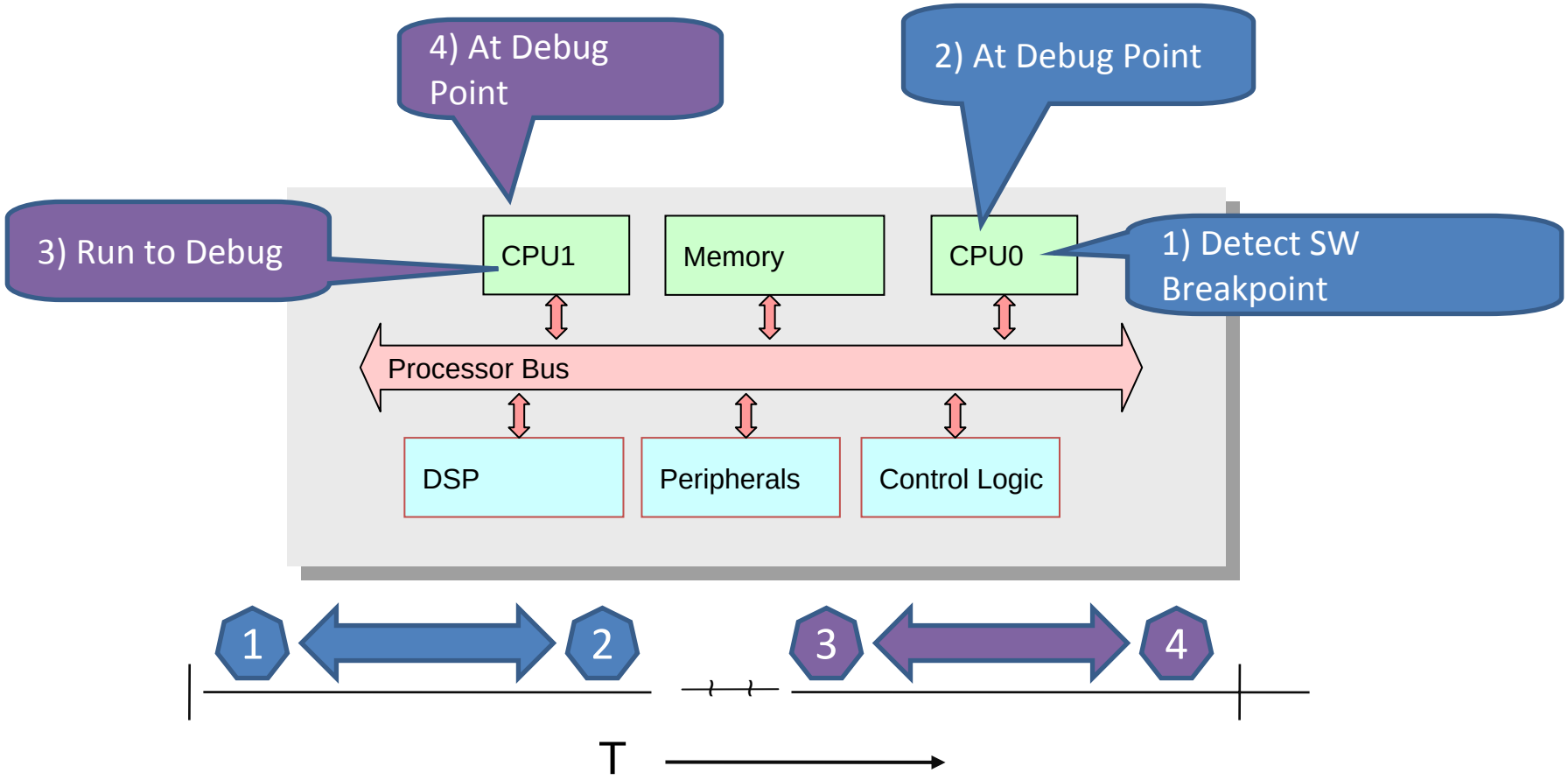
Core0 at Debug Point



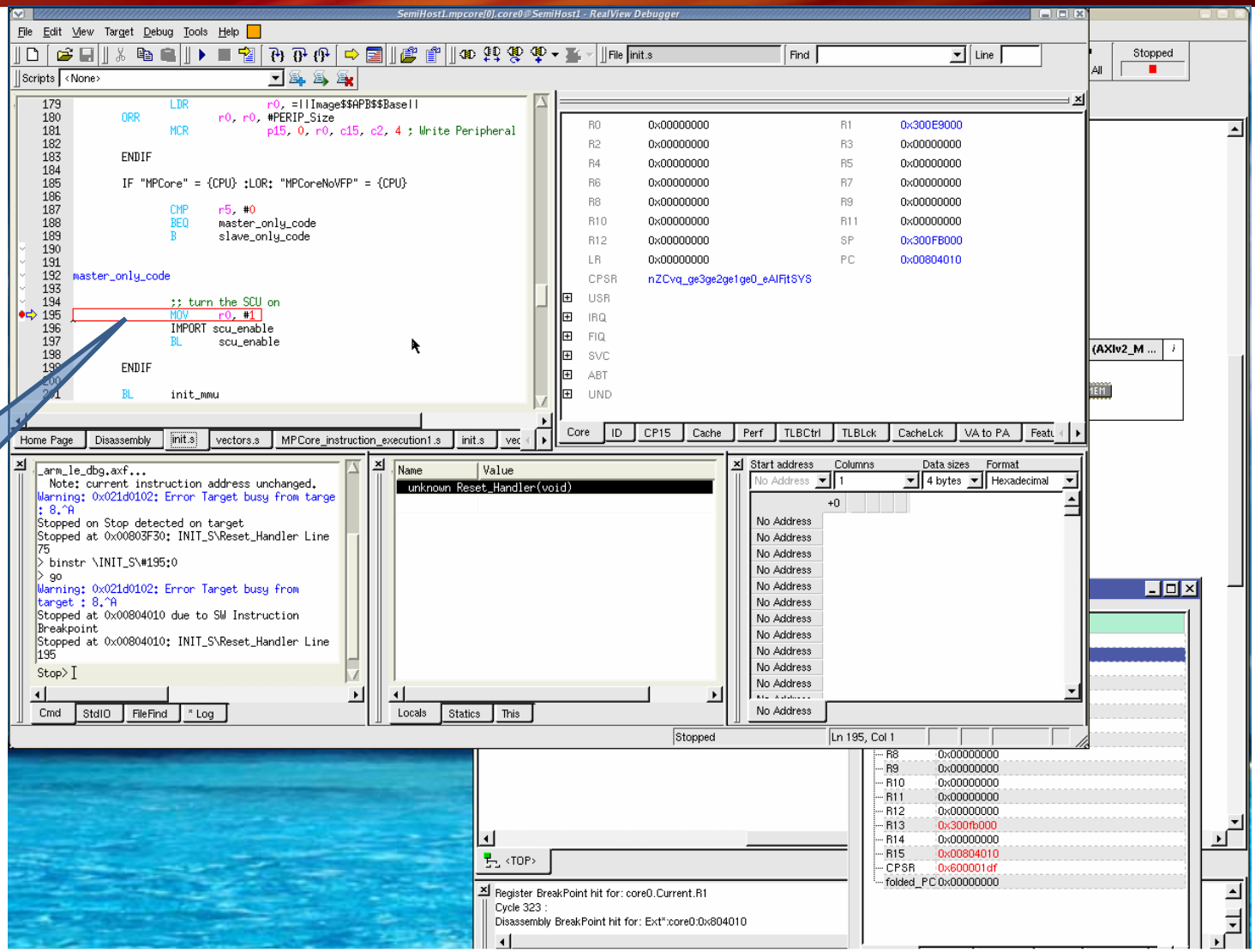
Multiprocessor Debugging

- Leverage model ability to reach a debug point
- Enable a debug point via a callback function
  - Pseudo Virtual Platform “external interrupt”
- Enables any processor to put other processors into a debug state, allowing separate debug sessions
- HW view via VP, SW views via Debuggers
  - TRUE HW/SW debugging capability

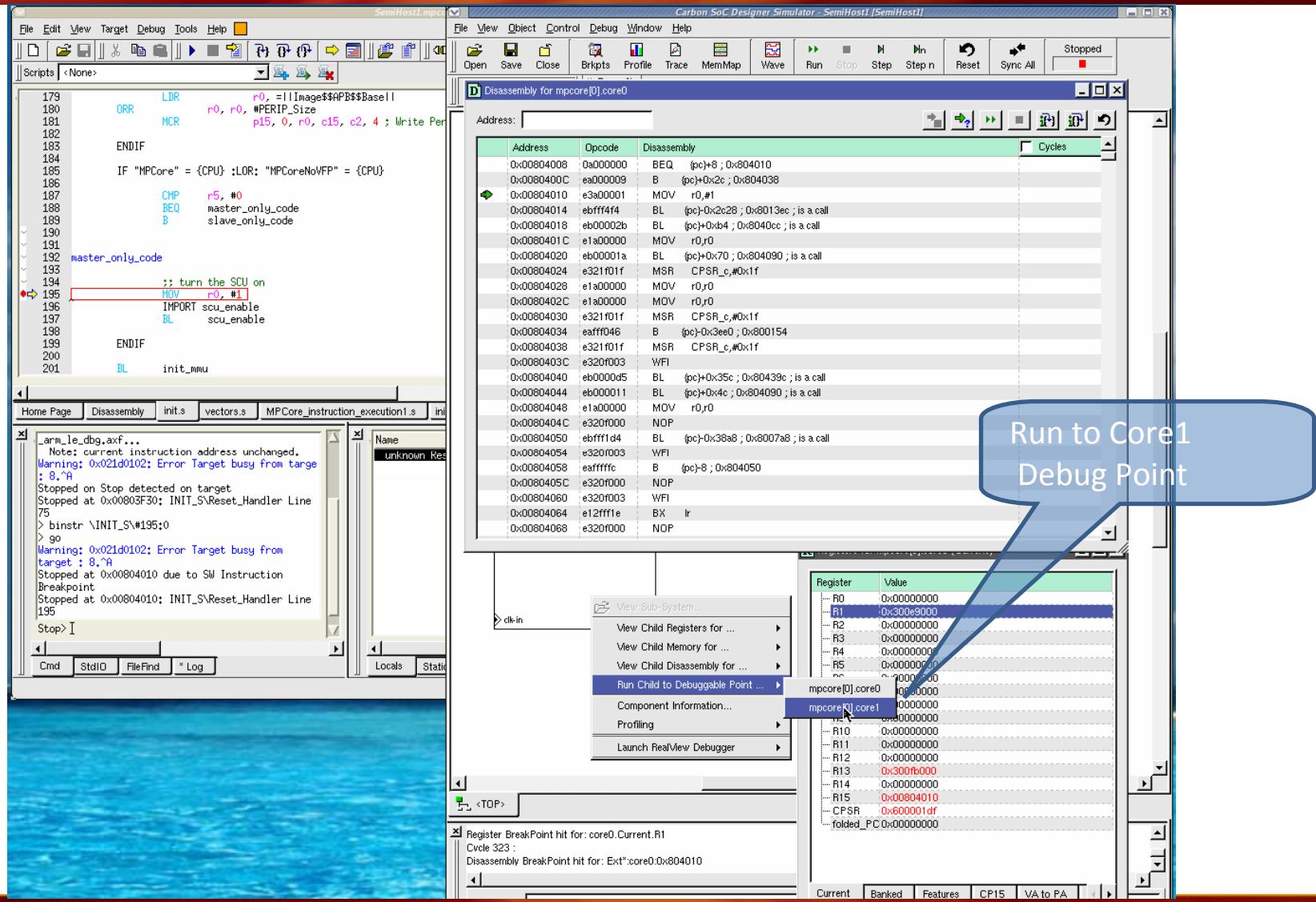
Multicore Debugging



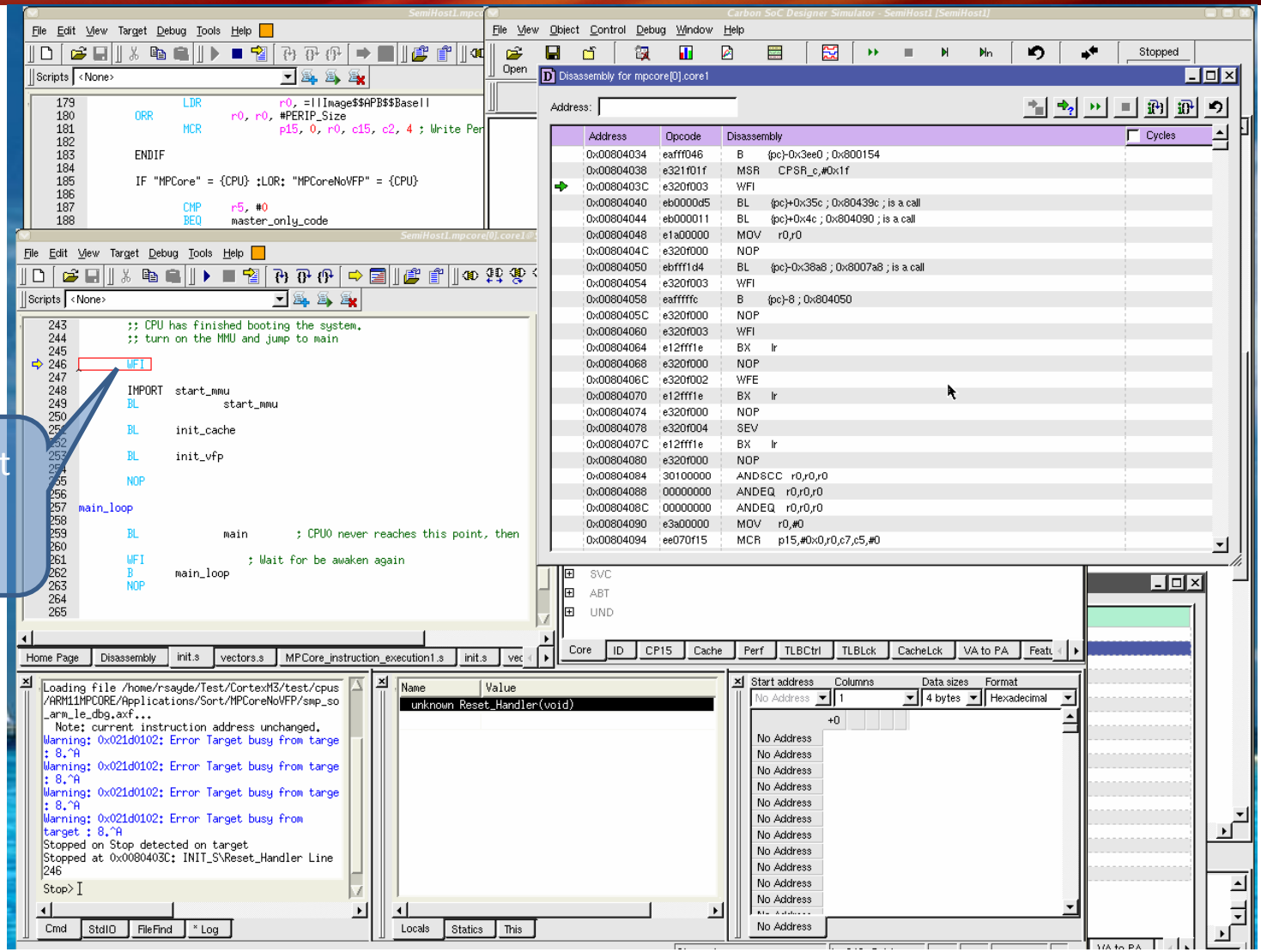
SW Breakpoint Core0



Instruct Core1 to Debug Point



Core1 at Debug Point



Programmer's View
Registers & Memory

- Required to integrate into PV virtual platform
- Isolate registers and memory functions via simulation
- Sequences of reads and writes -> common end and source points
- Leverage existing tools to provide register memory associations for virtual platform

Ports & Transactors

- Required to integrate into PV virtual platform
- Typically transactors or adaptors added to ports
- Facilitate system construction and analysis
- Common interface transactors available
- Leverage existing tools to integrate a core model with transactors and adaptors

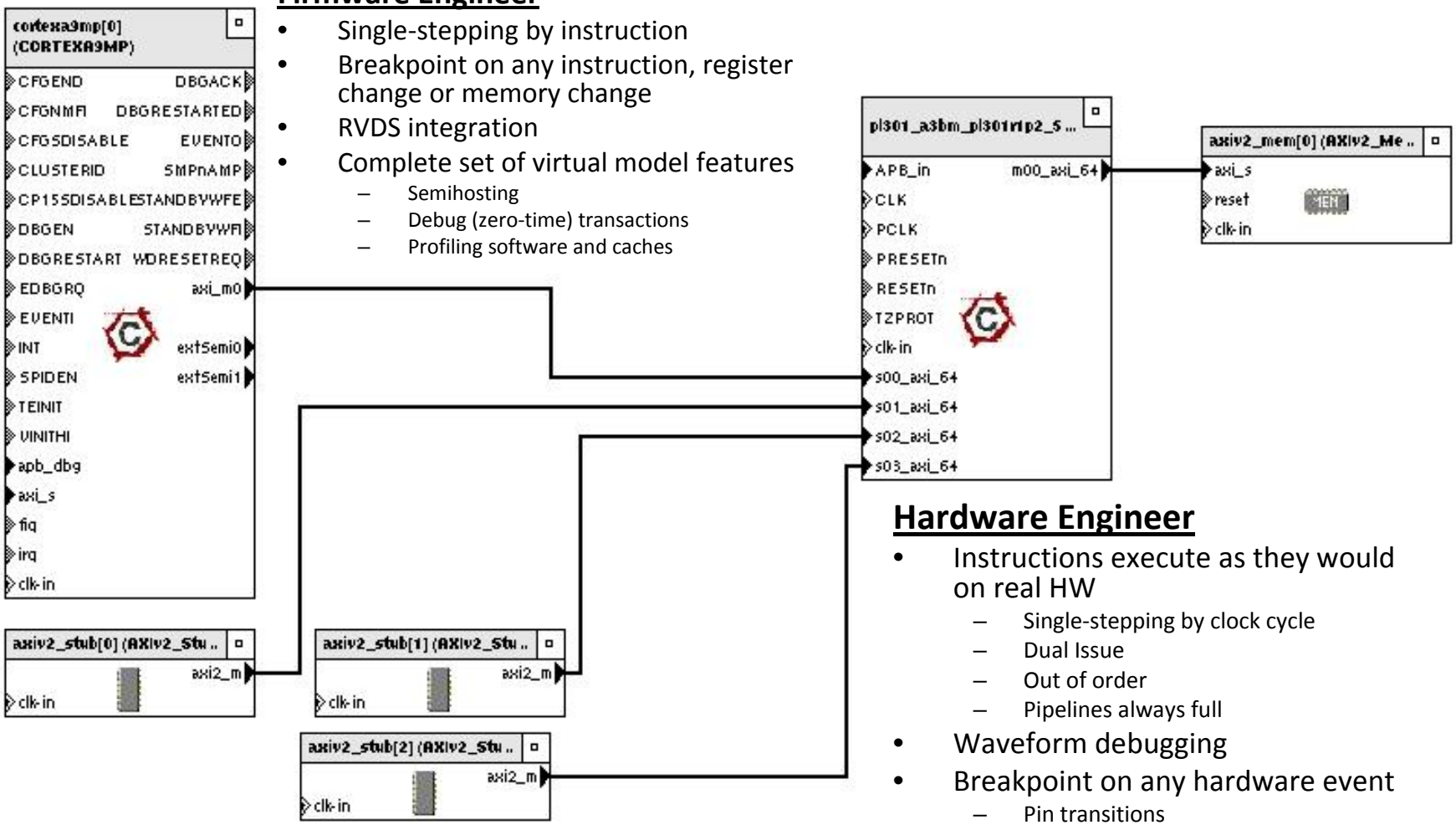
Other Models

- Other Processor Models – DSPs, Video Processors
  - Leverage CPU Model Development Flow
- Fabric
  - Understand system memory map
  - Understand transaction view of the system
  - Provide source or forwarding capability for Debugger requests for memory reads
- Memory blocks
  - Understand transaction view of the system
  - Prioritize write transactions over internal memory for Debugger requests

A9 Systems: In use today

Firmware Engineer

- Single-stepping by instruction
- Breakpoint on any instruction, register change or memory change
- RVDS integration
- Complete set of virtual model features
  - Semihosting
  - Debug (zero-time) transactions
  - Profiling software and caches



Hardware Engineer

- Instructions execute as they would on real HW
  - Single-stepping by clock cycle
  - Dual Issue
  - Out of order
  - Pipelines always full
- Waveform debugging
- Breakpoint on any hardware event
  - Pin transitions
  - Register changes

Summary

- RTL models support PV virtual platforms
  - Only true way to validate against original architectural & design decisions.
- PV view into RTL Models exists
- RTL level accuracy with full virtual platform ***visibility & control***
- More complicated models typically available through IP supplier or 3rd party (e.g., ARM/Carbon)