



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 Through 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 pipeline 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	
Loop	MVI		X									
Loop	LOAD			X								
Loop	MUL				X							
Loop	ACC					X						
Loop	BXH LOOP						X					
Loop	LOAD							X				
Loop	MUL								X			
Loop	ACC									X		
Loop	BXH LOOP										X	
Loop	ST											X

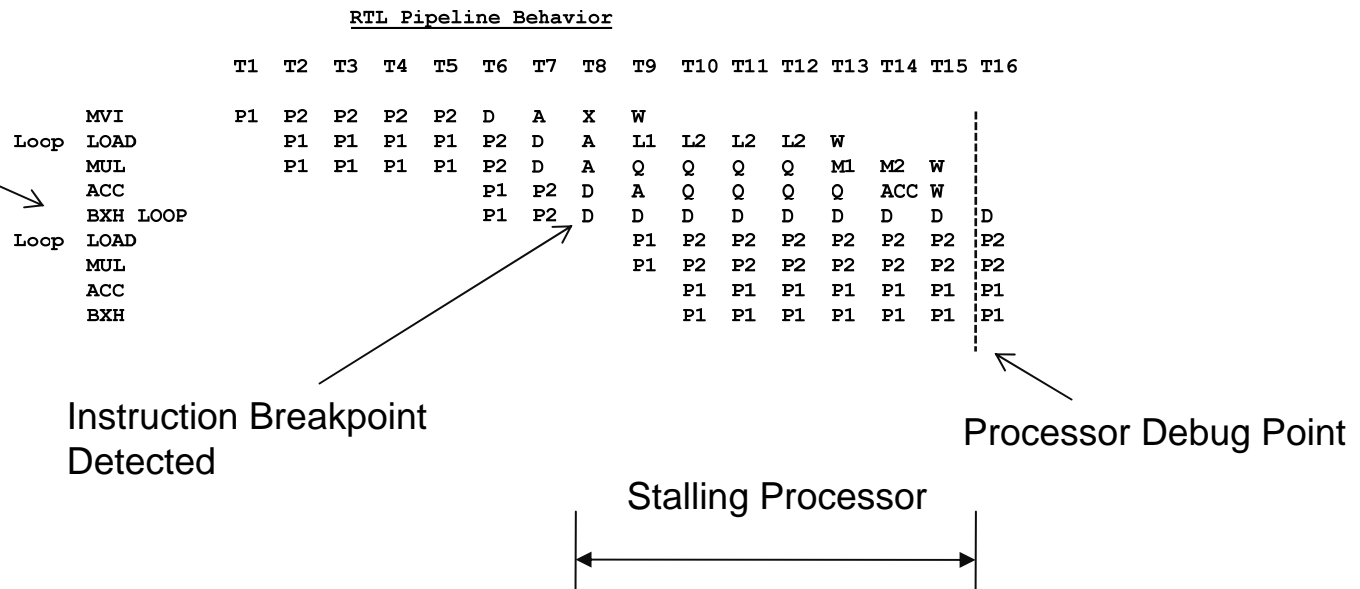
  

		RTL Pipeline Behavior																				
		T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	
Loop	MVI	P1	P2	P2	P2	P2	D	A	X	W												
Loop	LOAD		P1	P1	P1	P1	P2	D	A	L1	L2	L2	L2	W								
Loop	MUL		P1	P1	P1	P1	P2	D	A	Q	Q	Q	Q	M1	M2	W						
Loop	ACC						P1	P2	D	A	Q	Q	Q	Q	ACC	W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

# Reaching a “Debug Point”

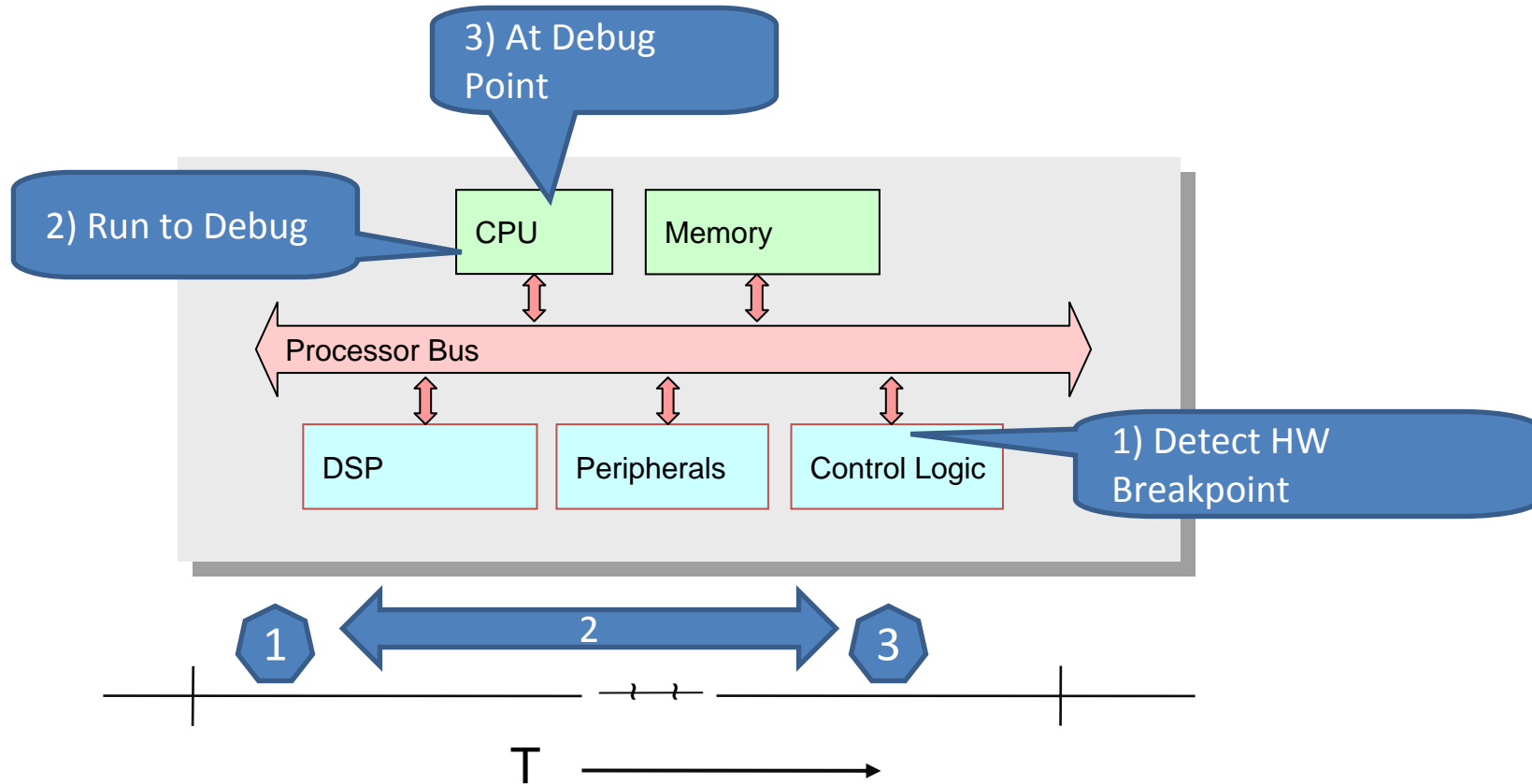
- Enable debugger interaction via “stalling” processor
- May be used to “trap” other functions (e.g., ARM’s semi-hosting function)
- Breakpoints are “extendable” with arbitrary delays and functions

Instruction Breakpoint  
Address





# HW / SW Debugging



# HW Breakpoint

The screenshot displays the Carbon SoC Designer Simulator interface. On the left, a 'Registers for mpcore[0].core0 (Current)' window shows a list of registers with R1 selected. The main workspace shows a block diagram with the following components and connections:

- mpcore[0] (ARM11MPCore)**: Contains a 'PeriphBase' block and an 'axiL\_m0' signal line.
- pl301\_a3bm\_pl301r1p2\_142 x2\_1[0] (pl301\_a3bm\_PL301r1p2\_A1)**: Receives 'axiL\_m0' and 'axiL\_s0' signals.
- axiv2\_mem[0] (AXIv2\_M...)**: Receives 'axiL\_s' and 'clk-in' signals.
- carbonsemihost[0] (CarbonSemihost)**: Receives 'semihost' and 'clk-in' signals.

A blue callout bubble points to the 'axiL\_m0' signal line with the text: **1) Detect HW Breakpoint**

Registers for mpcore[0].core0 (Current)

Register	Value
R0	0x00000000
R1	0x30100000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13	0x00000000
R14	0x00000000
R15	0x00803f30
CPSR	0x000001d3
folded_PC	0x00000000

Launching RealView Debugger via command : #home/sayde/Test/CortexM3/test/cpus/ARM11 MPCORE/MODELS/ARM11 MPCORE\_r2p0/test\_2cpu1axi/gcc

# Instruct Core0 to Debug Point

The screenshot displays the Carbon SoC Designer Simulator interface. On the left, a window titled "Registers for mpcore[0].core0 (Current)" shows a list of registers with their values. Register R1 is highlighted in orange, indicating it is the current register. The values for registers R0 through R15 are mostly 0x00000000, with R14 at 0x00803f30 and CPSR at 0x000001d3. The folded\_PC is 0x00000000.

The main window shows a block diagram of the system. A context menu is open over the "mpcore[0].core0" block, with the option "Launch RealView Debugger" selected. A blue callout bubble points to this option with the text "Run to Debug Point".

Register	Value
R0	0x00000000
R1	0x30100000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13	0x00000000
R14	0x00803f30
CPSR	0x000001d3
folded_PC	0x00000000

mpcore[0] (ARM11MPCore) | axi\_m0 | pi301\_a3bm\_pi301r1p2\_142 | axiv2\_mem[0] (AXIv2\_M ...)

mpcore[0].core0

mpcore[0].core1

Run to Debug Point

Launching RealView Debugger via command : /home/rsayde/Test/Cortex/M3/test/cpus/ARM11MPCORE/MODELS/ARM11MPCORE\_r2p0/test\_2cpu1axi/gcc  
Cycle 52 :  
Register BreakPoint hit for : core0.Current.R1

# Core0 at Debug Point

At Debug Point

```
66 Reset_Handler
67
68 ;--- Initialize stack pointer registers
69
70 IMPORT ||Image$$STACK$$ZI$$Limit||
71 LDR r1, =||Image$$STACK$$ZI$$Limit||
72
73 IF "MPCore" = {CPU} :LDR: "MPCoreNoVFP" = {CPU}
74 ;; First define the TOS depending on the CPUID
75 MRC r15, 0, r5, c0, 9
76 AND r5, r5, #0xf
77
78 LDR r2, =Offset_Stacks
79 MUL r2, r2, r5 ; decrease SP according to the CPU
80 SUB r1, r1, r2
81
82 ENDIF
83 ;--- The rest of the init should work as is
84 ; Enter each mode in turn and set up the stack pointer
85 MOV sp, r1
86 MOV r14, #0
87 MSR SPSR_cxsf, r14
88 SUB r1, r1, #Len_SVC_Stack
```

Register	Value
R0	0x00000000
R1	0x30100000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13	0x00000000
R14	0x00000000
R15	0x00000000
SP	0x00000000
LR	0x00000000
PC	0x00803F30
CPSR	nzcvq_ge3ge2ge1ge0_eAIFtSVC
USR	
IRQ	
FIQ	
SVC	
ABT	
UND	

```
Attached to stopped device
Stopped on Stop detected on target
Stopped at 0x00000000: _ENTRY__entry point>
> load/nv/r/mp /home/rsayde/Test/CortexM3/test/c
Loading file /home/rsayde/Test/CortexM3/test/cpus
/ARM11MPCORE/Applications/Sort/MPCoreNoVFP/smp_so
_ana_le_dbg.axf...
Note: current instruction address unchanged.
Warning: 0x021d0102: Error Target busy from targ
e : 8,*R
Stopped on Stop detected on target
Stopped at 0x00803F30: INIT_S\Reset_Handler Line
75
Stop> |
```

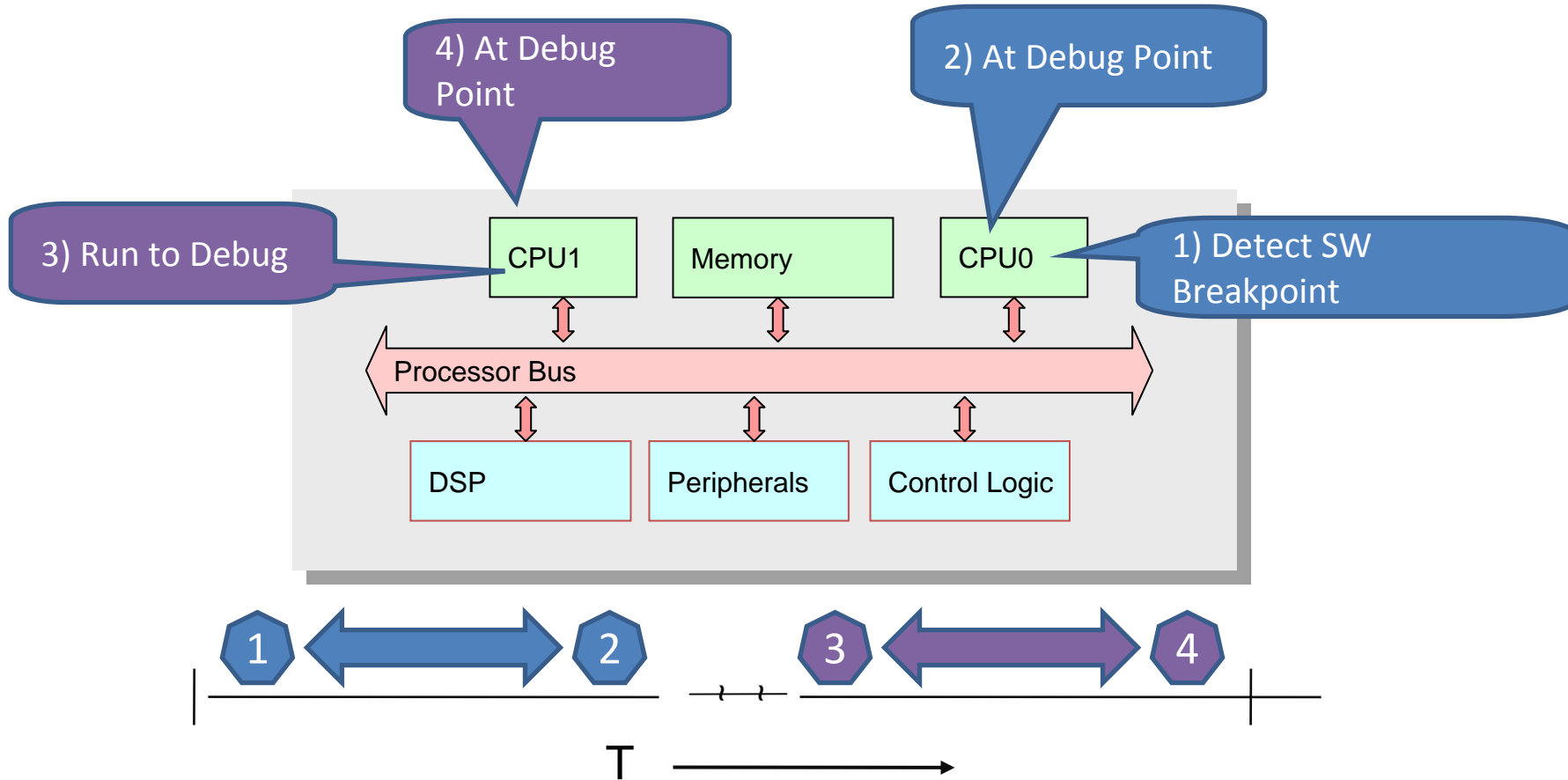
Register	Value
R0	0x00000000
R1	0x30100000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13	0x00000000
R14	0x00000000
R15	0x00803F30
CPSR	0x000001d3
folded_PC	0x00000000

Launching RealView Debugger via command : /home/rsayde/Test/CortexM3/ Cycle 52 : Register BreakPoint hit for: core0.Current.R1

# 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

Core0 At Debug Point

```
179      ORR    LDR    r0, =|Image$$APB$$Base|
180
181      MCR    r0, #PERIP_Size
182           p15, 0, r0, c15, c2, 4 ; Write Peripheral
183
184      ENDIF
185
186      IF "MPCore" = {CPU} :LOR: "MPCoreNoVFP" = {CPU}
187
188          CMP    r5, #0
189          BEQ    master_only_code
190          B      slave_only_code
191
192      master_only_code
193
194          ;; turn the SCU on
195      MOV    r0, #1
196      IMPORT scu_enable
197      BL     scu_enable
198
199      ENDIF
200
201      BL     init_mmu
```

Register	Value
R0	0x00000000
R1	0x300E9000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
SP	0x300FB000
LR	0x00000000
PC	0x00804010
CPSR	nZCvq_ge3ge2ge1ge0_eAIFITSYS
USR	
IRQ	
FIQ	
SVC	
ABT	
UND	

Register BreakPoint hit for: core0.Current.R1  
Cycle 323 :  
Disassembly BreakPoint hit for: Ext\*core0:0x04010

# Instruct Core1 to Debug Point

The image shows a screenshot of the Carbon SoC Designer Simulator interface. The main window displays assembly code for 'SemiHost1.mpcore'. A red arrow points to line 195, which contains the instruction 'MOV r0, #1'. A blue callout bubble with the text 'Run to Core1 Debug Point' points to this instruction. Below the code, a console window shows the execution log, including warnings and messages indicating that the program stopped at the debug point.

**Assembly Code (Line 195):**

```
195 MOV r0, #1
```

**Disassembly Table:**

Address	Opcode	Disassembly
0x00804008	0a000000	BEQ (pc)+8 ; 0x804010
0x0080400C	ea000009	B (pc)+0x2c ; 0x804038
0x00804010	e3a00001	MOV r0,#1
0x00804014	ebfff4f4	BL (pc)-0x2c8 ; 0x8013ec ; is a call
0x00804018	eb00002b	BL (pc)+0xd4 ; 0x8040cc ; is a call
0x0080401C	e1a00000	MOV r0,r0
0x00804020	eb00001a	BL (pc)+0x70 ; 0x804090 ; is a call
0x00804024	e321f01f	MSR CPSR_c,#0x1f
0x00804028	e1a00000	MOV r0,r0
0x0080402C	e1a00000	MOV r0,r0
0x00804030	e321f01f	MSR CPSR_c,#0x1f
0x00804034	eaffff46	B (pc)-0x3ee0 ; 0x800154
0x00804038	e321f01f	MSR CPSR_c,#0x1f
0x0080403C	e320f003	WFI
0x00804040	eb0000d5	BL (pc)+0x35c ; 0x80439c ; is a call
0x00804044	eb000011	BL (pc)+0x4c ; 0x804090 ; is a call
0x00804048	e1a00000	MOV r0,r0
0x0080404C	e320f000	NDP
0x00804050	ebffffd4	BL (pc)-0x38a8 ; 0x8007a8 ; is a call
0x00804054	e320f003	WFI
0x00804058	eafffffc	B (pc)-8 ; 0x804050
0x0080405C	e320f000	NDP
0x00804060	e320f003	WFI
0x00804064	e12ffffe	BX lr
0x00804068	e320f000	NDP

**Register Values:**

Register	Value
R0	0x00000000
R1	0x300e9000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13	0x300b0000
R14	0x00000000
R15	0x00804010
CPSR	0x800001df
folded_PC	0x00000000

**Console Output:**

```
Note: current instruction address unchanged.  
Warning: 0x021d0102; Error Target busy from target : 8,"A"  
Stopped on Stop detected on target  
Stopped at 0x00803f30; INIT_S\Reset_Handler Line 75  
> binstr \INIT_S\#195:0  
> go  
Warning: 0x021d0102; Error Target busy from target : 8,"A"  
Stopped at 0x00804010 due to SM Instruction  
Breakpoint  
Stopped at 0x00804010; INIT_S\Reset_Handler Line 195  
Stop>]
```



# Core1 at Debug Point

The image shows a screenshot of the Carbon SoC Designer Simulator interface. The main window is titled "Disassembly for mpcore[0].core1" and displays a list of assembly instructions with their addresses and opcodes. The instructions are as follows:

Address	Opcod	Disassembly
0x00804034	eaaff046	B {pc}-0x3ee0; 0x800154
0x00804038	e321f01f	MSR CPSR_c,#0x1f
0x0080403c	e320f003	WFI
0x00804040	eb0000d5	BL {pc}+0x35c; 0x80438c; is a call
0x00804044	eb000011	BL {pc}+0x4c; 0x804090; is a call
0x00804048	e1a00000	MOV r0,r0
0x0080404c	e320f000	NOP
0x00804050	ebffffd4	BL {pc}-0x38a8; 0x8007a8; is a call
0x00804054	e320f003	WFI
0x00804058	eaaffffc	B {pc}-8; 0x804050
0x0080405c	e320f000	NOP
0x00804060	e320f003	WFI
0x00804064	e12ffffe	BX lr
0x00804068	e320f000	NOP
0x0080406c	e320f002	WFE
0x00804070	e12ffffe	BX lr
0x00804074	e320f000	NOP
0x00804078	e320f004	SEV
0x0080407c	e12ffffe	BX lr
0x00804080	e320f000	NOP
0x00804084	30100000	ANDCC r0,r0,r0
0x00804088	00000000	ANDEQ r0,r0,r0
0x0080408c	00000000	ANDEQ r0,r0,r0
0x00804090	e3a00000	MOV r0,#0
0x00804094	ee070f15	MCR p15,#0x0,r0,c7,c5,#0

The left window shows source code for "mpcore0].core1" with assembly instructions like WFI and B. A blue callout bubble points to the WFI instruction at line 246, stating "Core1 At Debug Point". The bottom left window shows a console with error messages and a stop message: "Stopped on Stop detected on target. Stopped at 0x0080403c: INIT\_S/Reset\_Handler Line 246". The bottom right window shows a register window with "unknown Reset\_Handler(void)" and a data display window showing hexadecimal values.

# 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

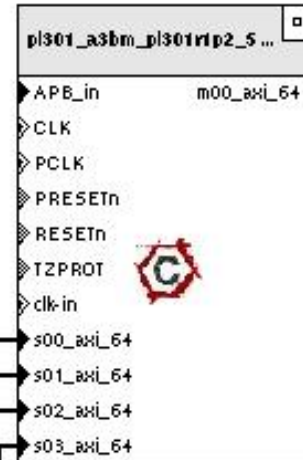
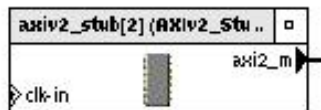
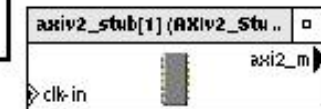
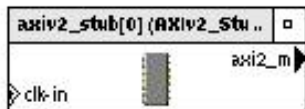
# Other Models

- 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
  - Semihosting
  - Debug (zero-time) transactions
  - Profiling software and caches
- Complete set of virtual model features



## 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 3<sup>rd</sup> party (e.g., ARM/Carbon)