



Achim Nohl, CoWare Inc

Booth Number 3665 located in the North Hall

***Getting started with Virtual Platforms:
A Software Developer Perspective***
*Virtual Platform Workshop
DAC 09, San Francisco*

Objective

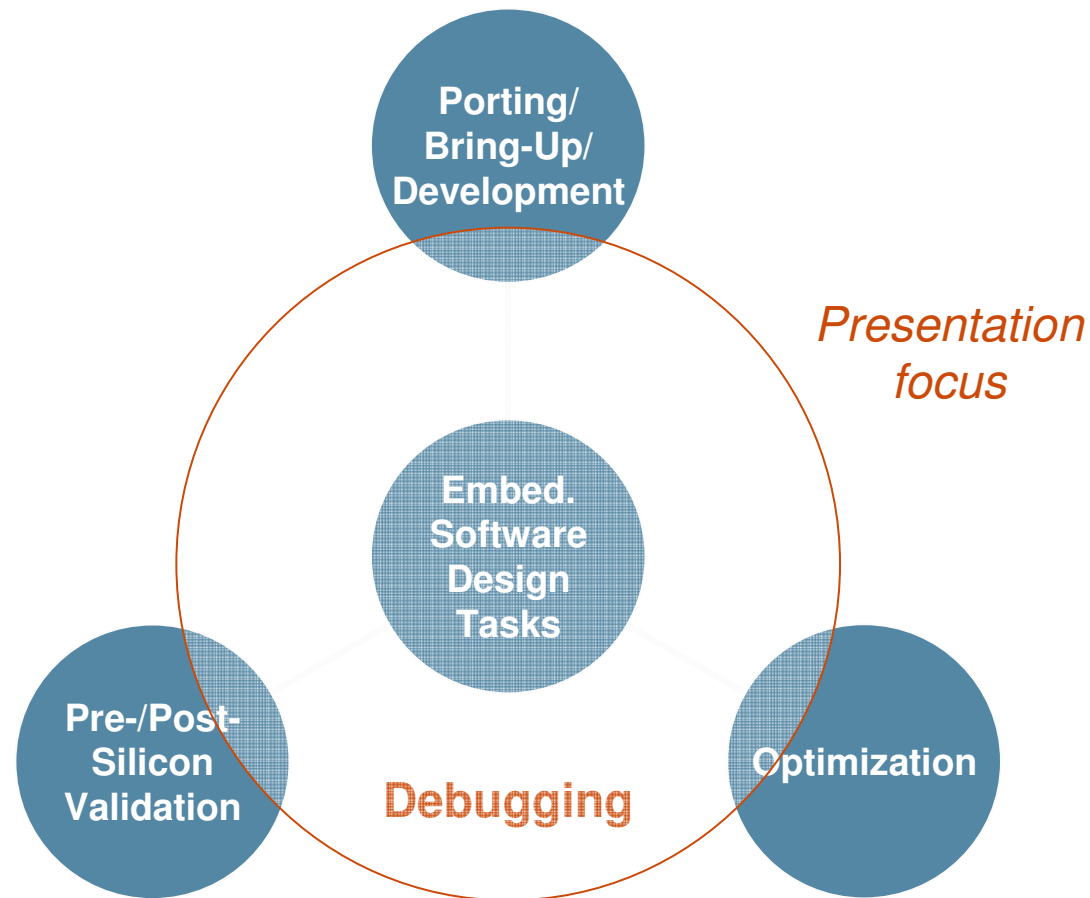
Objective: Provide illustrative examples on how Virtual Platforms are used for debugging.

Non-Objective: Provide a complete feature- and benefit-list of Virtual Platforms.

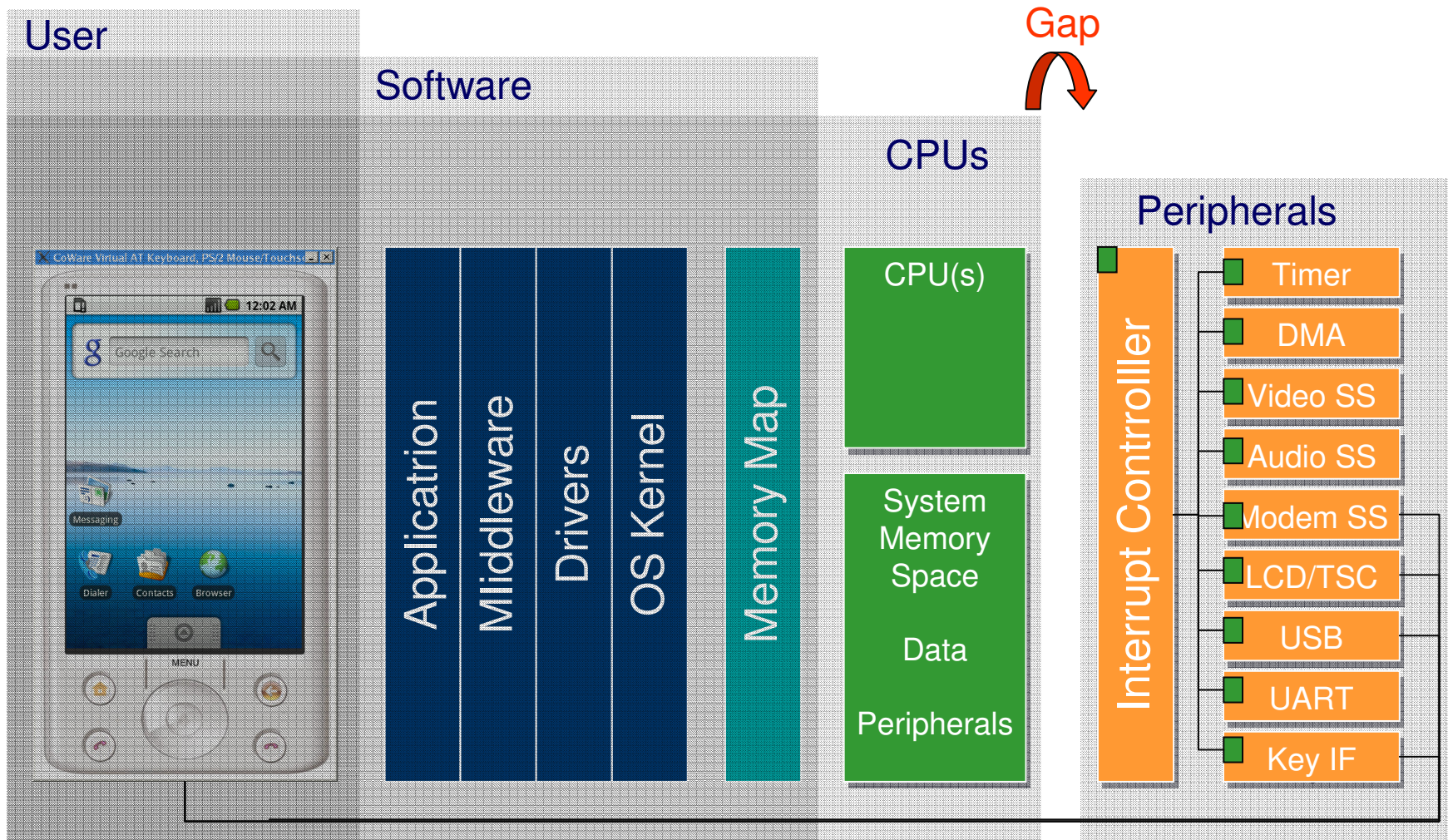
Outline

- **SW Developers's Views**
- **Debugging on the boundary of HW and SW**
- **Virtual Platform based Debugging**
- **Platform Level Software Analysis**
- **Virtual Platform Scripting**
 - Tracing
 - Software Assertions
- **Summary**

Embedded Software Design Tasks



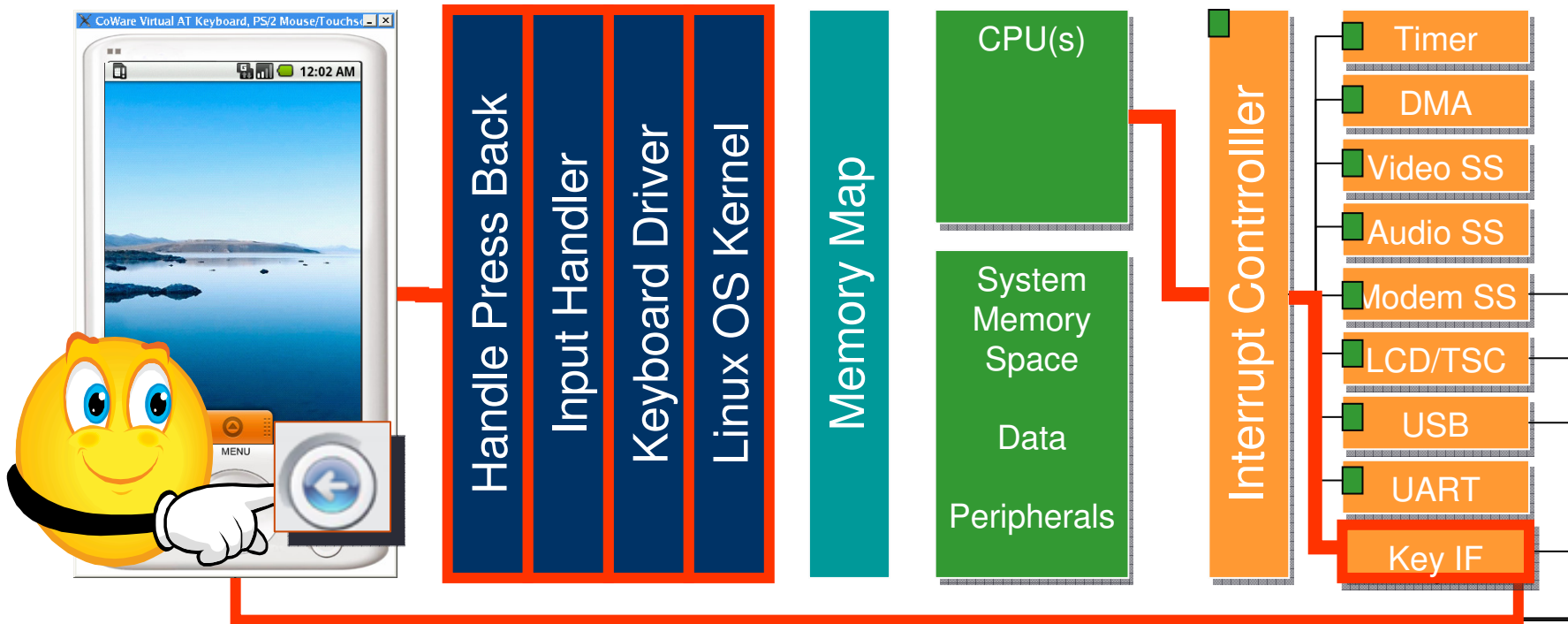
Perspectives



User's View

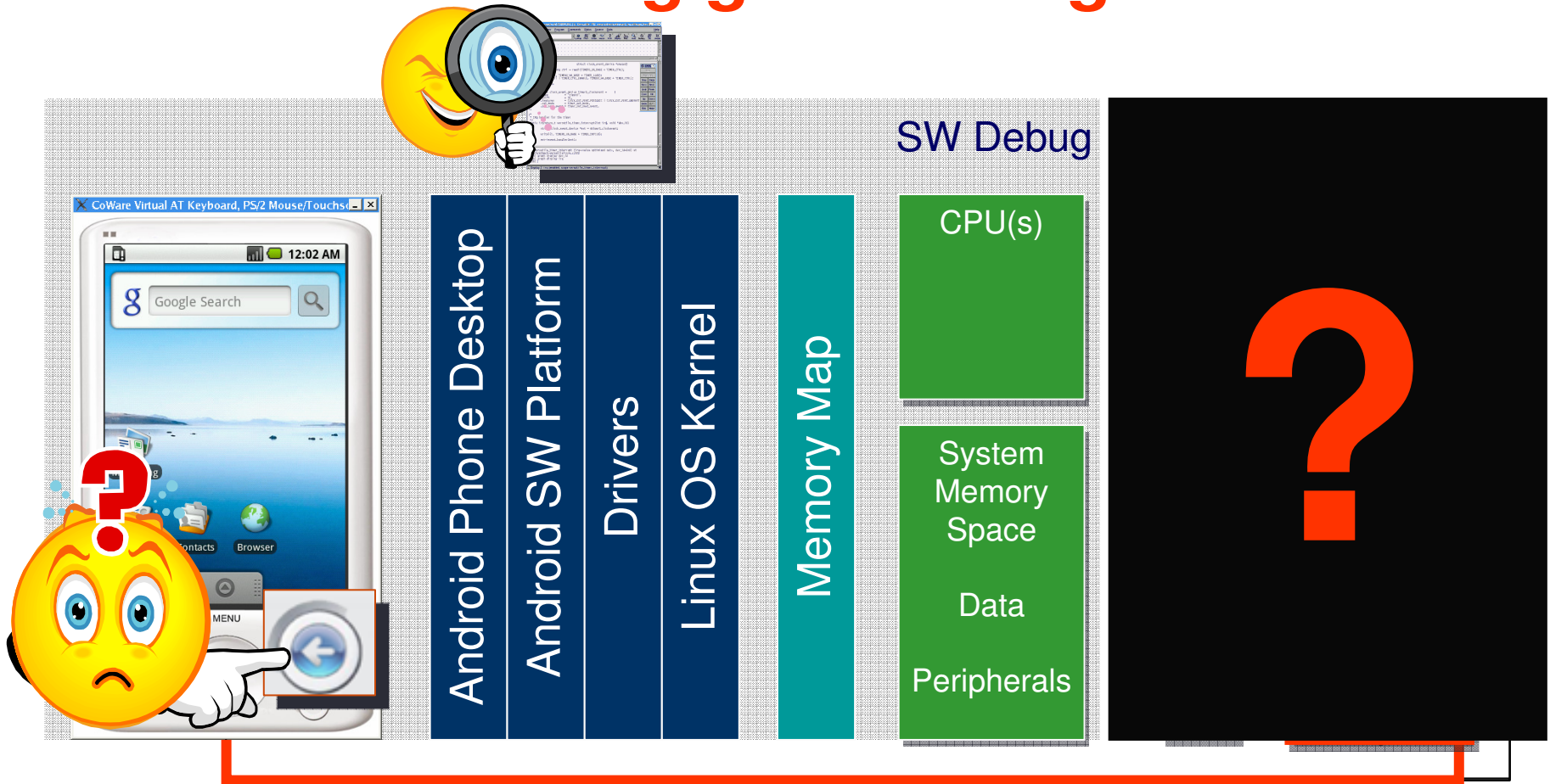


Example: Device Key Press

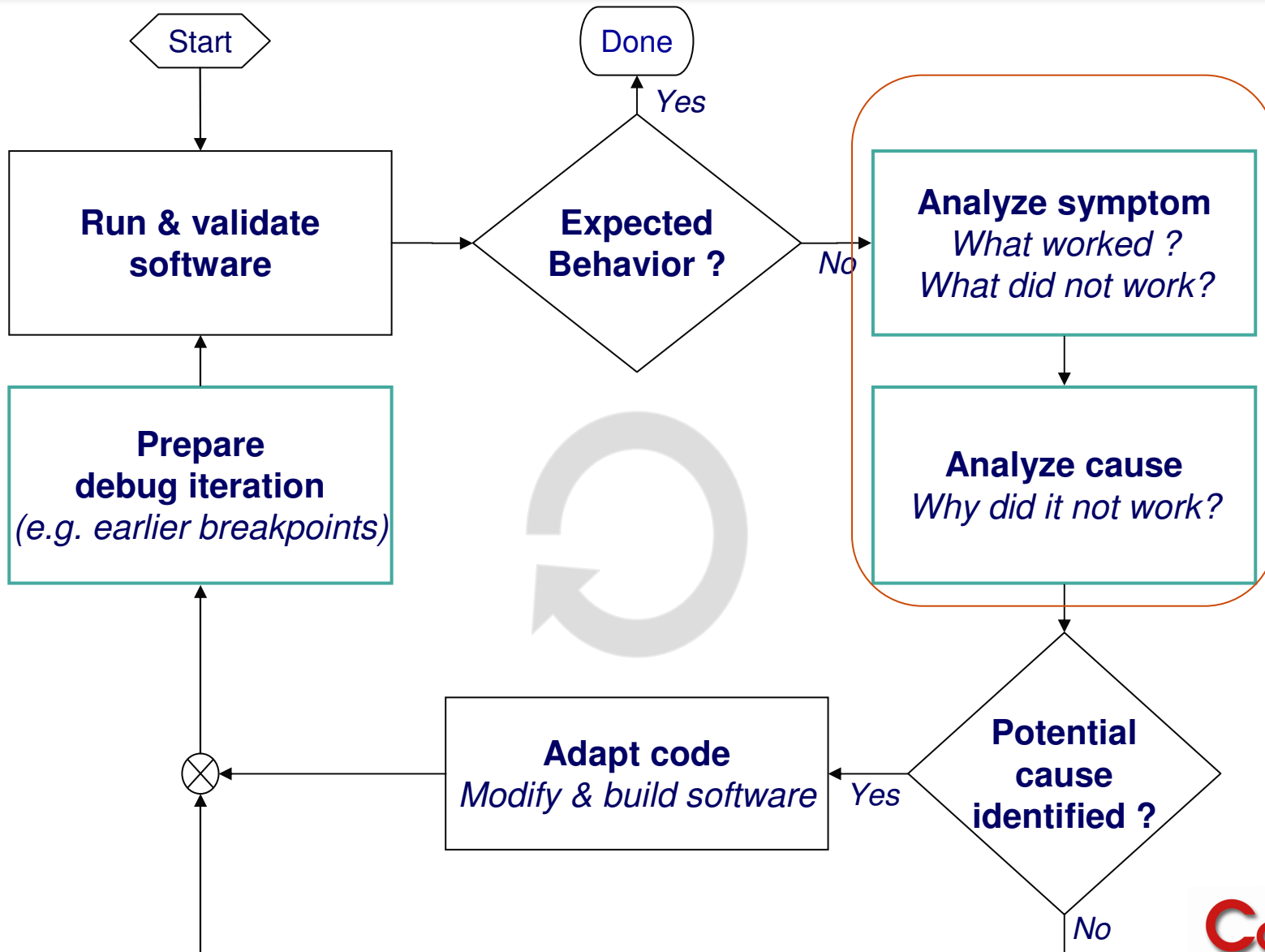


Example: Device Key Press

What if something goes wrong ?



Debugging Process

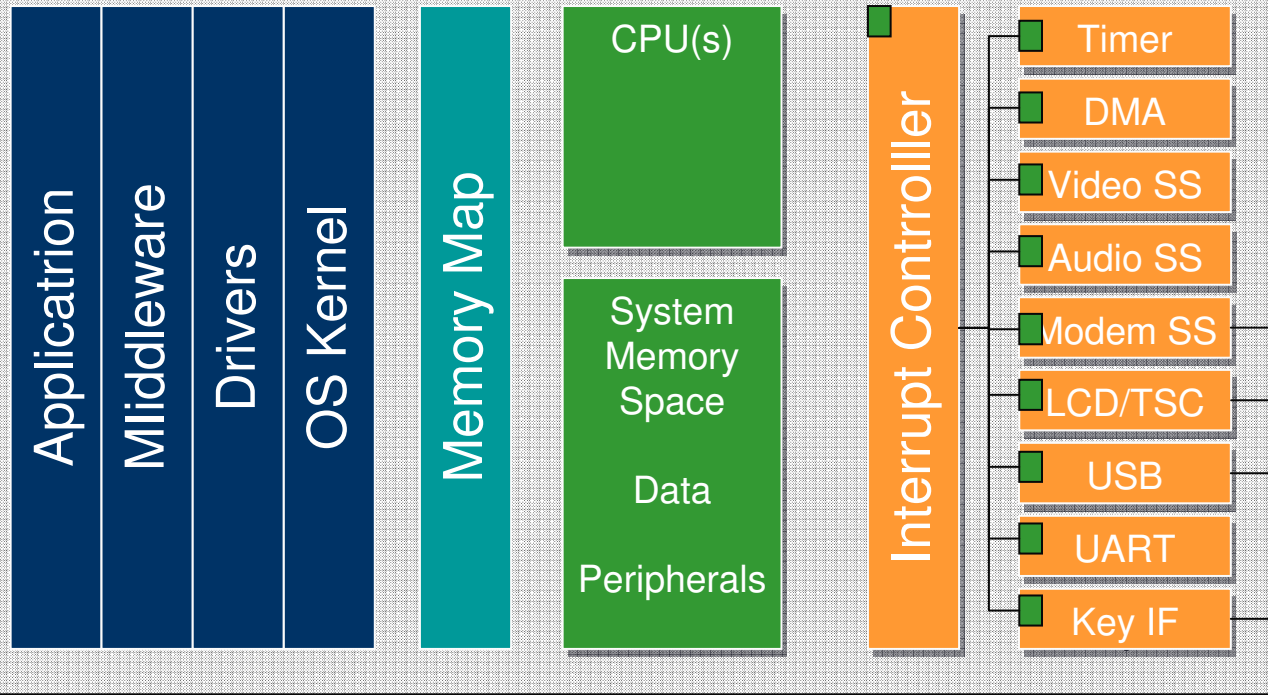
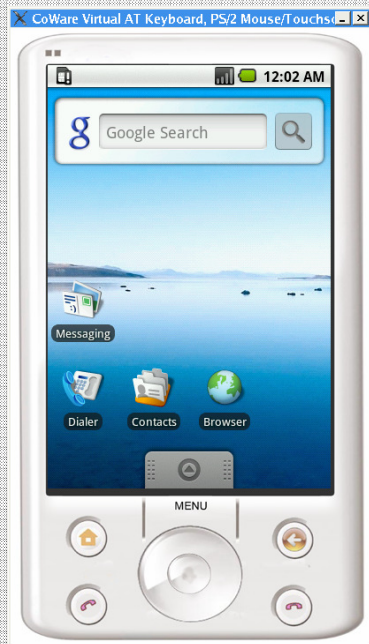


Virtual Platform Perspective



Synchronous system control
Full and consistent system visibility

Virtual Platform Based Debug Perspective



Time and Space During Debug

Software initiated system activity:

- Example: Timer dies
- What code should I look at and debug?

Hardware initiated system activity:

- Example: Device key press
- What time should I stop to debug?

Watchpoints: Fast Track to the Problem

■ Getting to the code:

– *Need to determine the code that configures/ corrupts the peripheral*

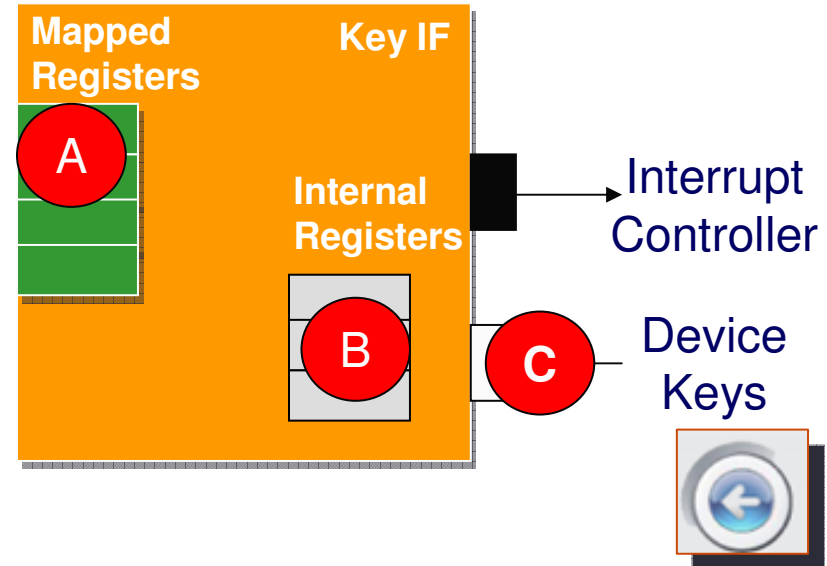
– **A: Software access watchpoint**

■ Getting to a point in time

– *Need to closely investigate the SW reaction on an event generated by the HW*

– **B: Hardware access watchpoint**

– **C: Signal watchpoint**



Watchpoints

Virtual Platform Analyzer 2009.1.1 : vpa.MUEHLEN2.576 : ARM926EJS_0

File Simulation Debug Analysis View Windows Linux Connectivity Observers SW Debug Help

Item

- HARDWARE
 - BOOT_0
 - BOOT_1
 - BOOT_2
 - BOOT_3
 - CLCDC_Memory
 - i_ARM926EJS_0**
 - i_ARM926EJS_1
 - i_ARM926EJS_2
 - i_ARM926EJS_3
 - i_CLCDC_PL111
 - i_DMxAC_PL080
 - i_GPIO_PL061_0
 - i_GPIO_PL061_1
 - i_GPIO_PL061_2
 - i_GPIO_PL061_3
 - i_KMI_0
 - i_KMI_1
 - i_PIO
 - i_RTC_PL031
 - i_SCL_PL130
 - i_SIC
 - i_SmartCard
 - i_SMC91C111
 - i_SSP_PL022
 - i_TIMER_01
 - i_TIMER_23
 - i_UART_PL011_0
 - i_UART_PL011_1
 - i_UART_PL011_2
 - i_UART_PL011_3
 - i_VIC_0

i_KMI_1 Registers (/HARDWARE/i_ARM926EJS_0)

wp	Item	Value	Start Address
	i_KMI_1		0x10007000
	IRQ	true	-
Rw	mRegs	Select Boundaries	0x10007000
	KMICLKDIV	0x2	0x1000700c
	KMICR	0x14	0x10007008
	KMIDATA	0x55	0x10007008
	KMIIR	0x3	0x10007010
	KMISTAT	0x10	0x10007004
	mRegs [5]	0x0	0x10007014
	mRegs [6]	0x0	0x10007018
	mRegs [7]	0x0	0x1000701c
	mRegs [8]	0x0	0x10007020
	mRegs [9]	0x0	0x10007024
	mRegs [10]	0x0	0x10007028
	mRegs [11]	0x0	0x1000702c

i_ARM926EJS_0 Disassembly

Address : 0xc0039708 Core : i_ARM926EJS_0

BP	Symbols	Address	Instruction	Disassembly
	<irq_enter+20>	[c0039700]	e99da910	LDMD R13, 0xc0039700
	<irq_enter+4>	[c0039704]	c0267ce0	EORGT R7, R6, #0
	<irq_enter+0>	[c0039708]	e1a0c00d	MOV R12, R13
	<irq_enter+4>	[c003970c]	e92dd800	STMDB R13!, 0xc003970c
	<irq_enter+8>	[c0039710]	e24cb004	SUB R11, R12, #4

Stoppers: vpa gdb@192.168.10.177:2192 0:06:15.051 114 140 000

softirq.c - Source Window

File Run View Control Preferences Help

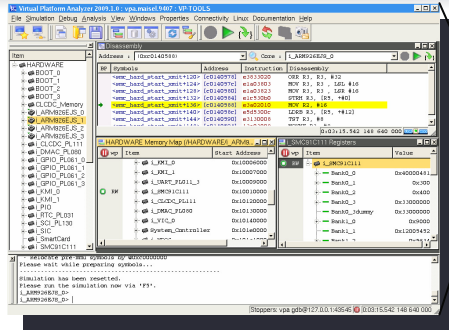
softirq.c i SOURCE

```
253 * Enter an interrupt context.
254 */
255 void irq_enter(void)
- 256 {
257 #ifdef CONFIG_NO_HZ
258     int cpu = smp_processor_id()
259     if (idle_cpu(cpu) && !in_int
260         tick_nohz_stop_idle(
261 #endif
262     __irq_enter());
263 #ifdef CONFIG_NO_HZ
264     if (idle_cpu(cpu))
265         tick_nohz_update_jif
266 #endif
267 }
268
269 #ifdef __ARCH_IRQ_EXIT_IRQS_DISABLED
270 # define invoke_softirq() __do_softirq
271 #else
272 # define invoke_softirq() do_softirq
273 #endif
274
275 /*
276 * Exit an interrupt context. Process
277 */
278 void irq_exit(void)
- 279 {
280     account_system_untime(current
281     trace_hardirq_exit();
- 282     sub_preempt_count(IRQ_EXIT_COUNT);
- 283     if (!in_interrupt() && local_irq_enabled())
```

Program stopped at line 256 c0039708 256



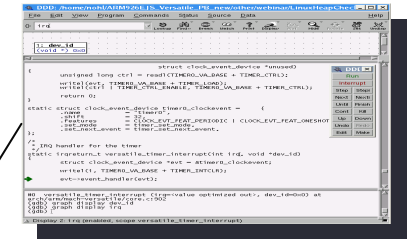
Platform Level Software Debugging



Virtual Platform Analyzer

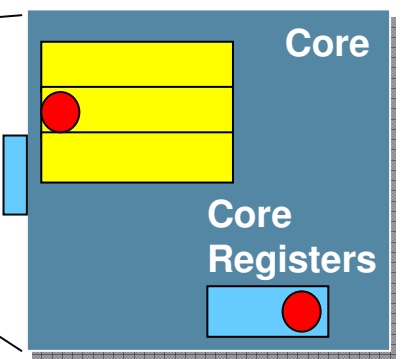
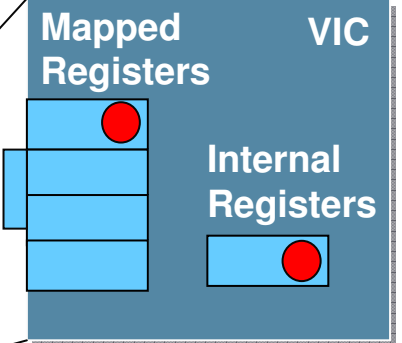
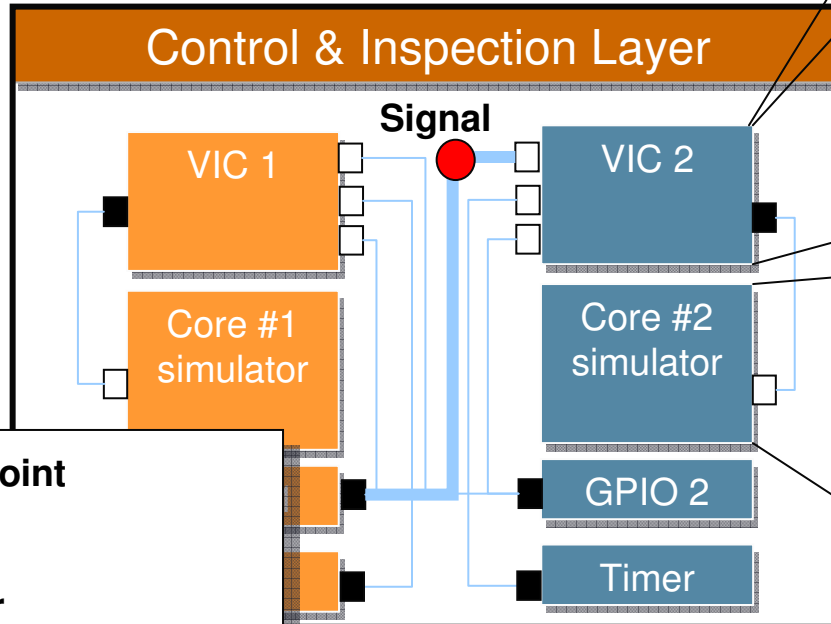
E.g.:
 ARM926EJS_0 step_core
 VIC_1/EnableMask get_value




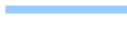
SW Debugger



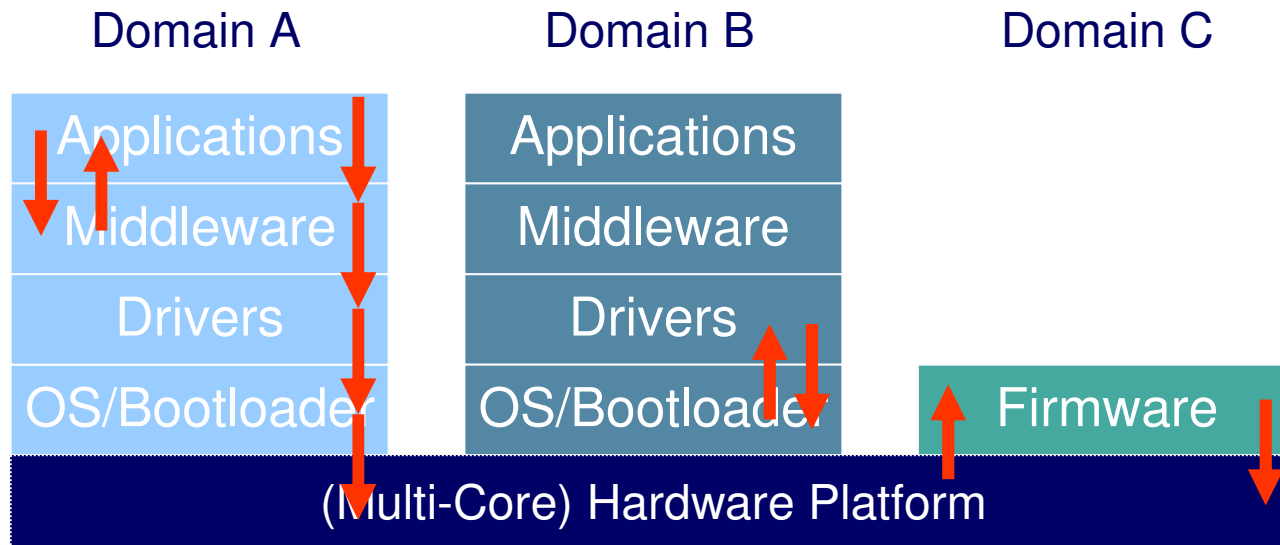
TCL Scripting Interface

Control & Inspection Layer



-  Break-/Watch-point
-  Visible Code
-  Visible Register
-  Visible Signal

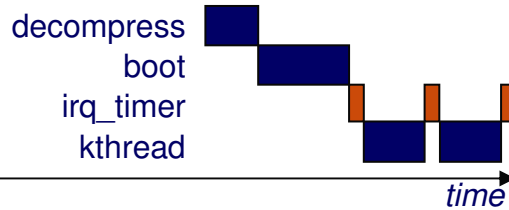
Platform Level Software Analysis



- **Debugging:** Analyze a snapshot of the system state
- **Challenge:** Understand/analyze system history
 - Interaction between HW and SW entities over time
- **Requirement:** System level tracing of HW/SW

Platform Level Software Analysis

OS Context Tracing Core A



Context's Function Tracing

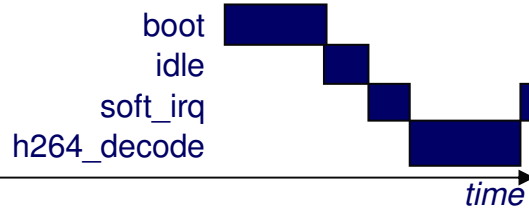


Instruction Tracing

```
c002b62c: MOV R0, R4  
c002b630: MOV R1, R5  
c002b634: MOV R2, R6  
c002b638: MOV R14, R15  
c002b6ec: LDR R15, [R5, #0]
```

time

OS Context Tracing Core B

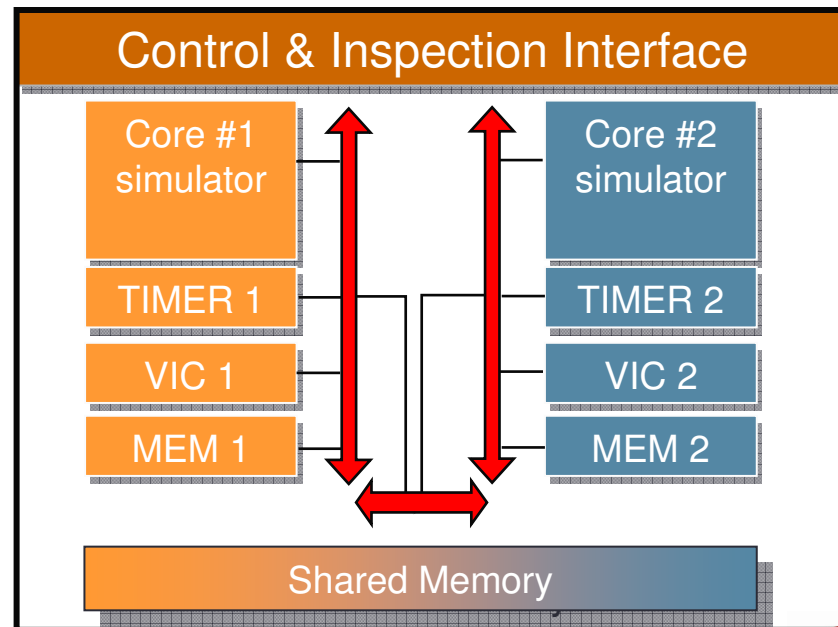


Platform Memory Access Tracing

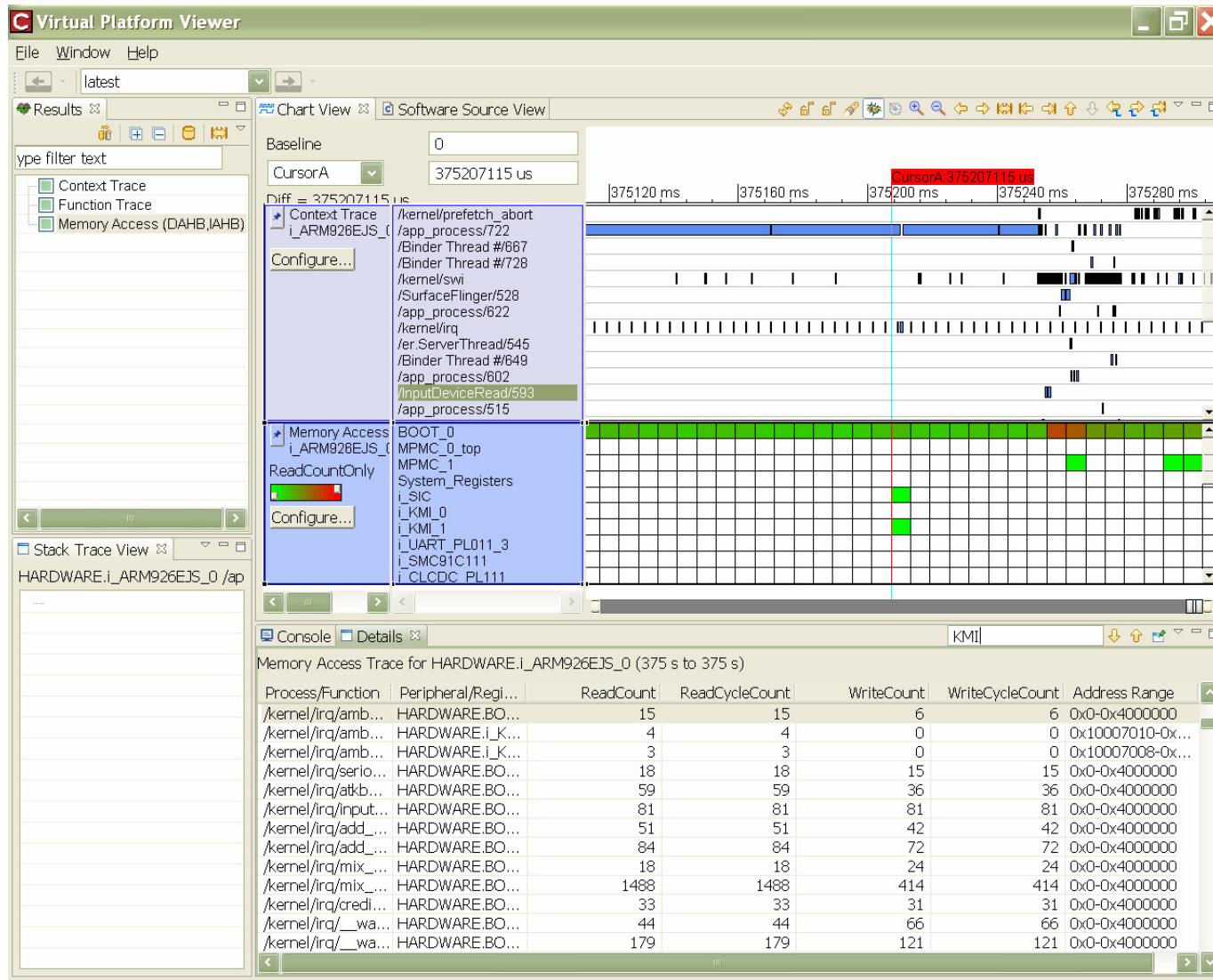


Platform Level Analysis

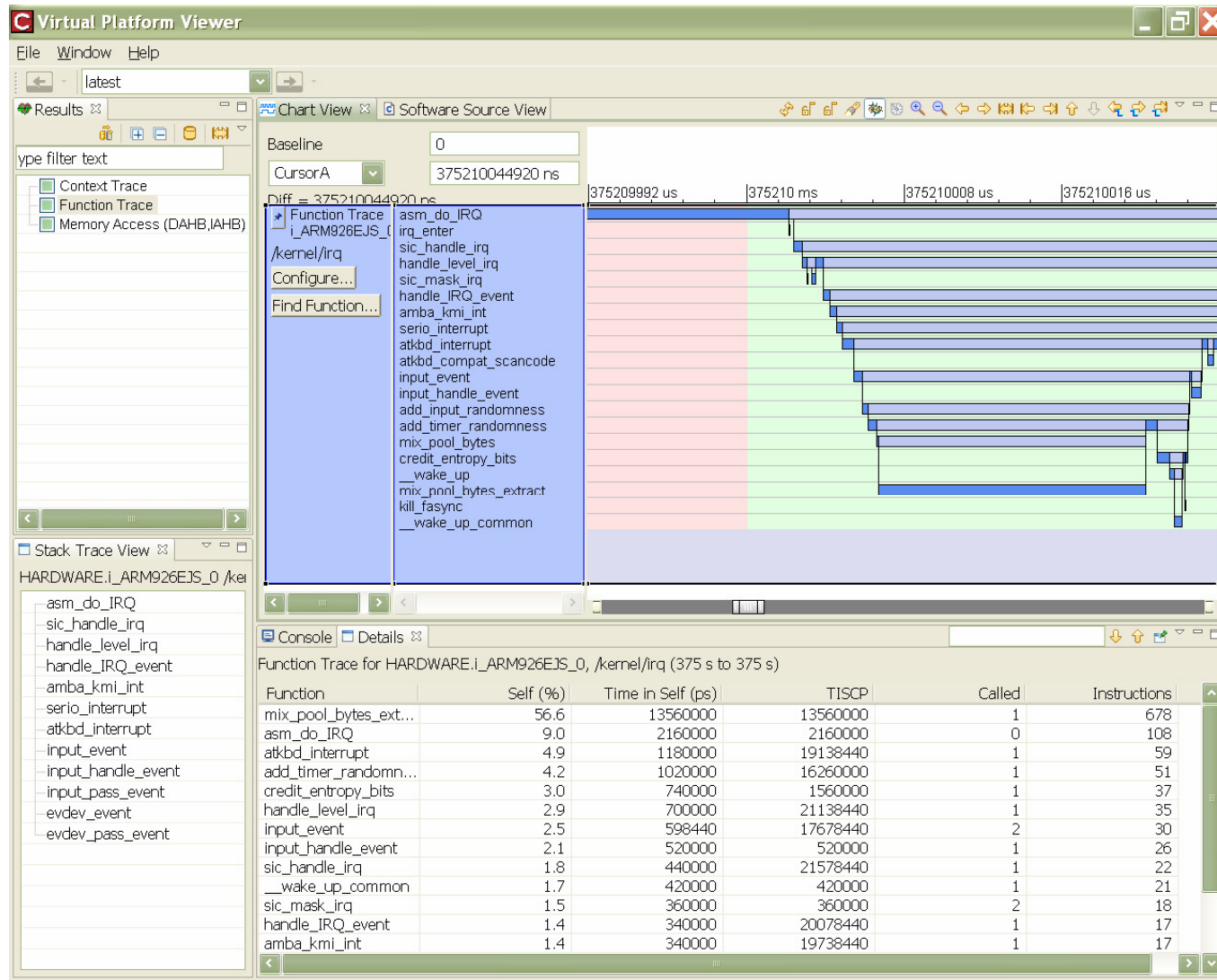
Control & Inspection Interface



Platform Level Software Analysis



Platform Level Software Analysis



Virtual Platform Scripting

Scripting Use Cases:

- Deterministic repetition of scenarios
- Regressionizing
- Analyzing
- Debugging

Virtual Platform Analyzer

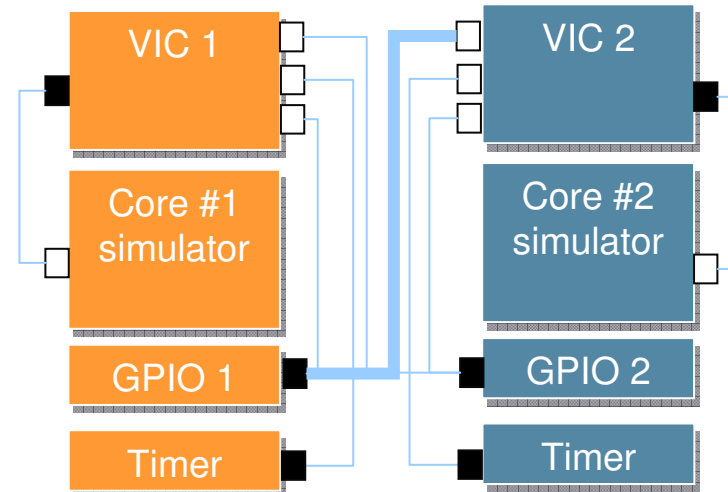
E.g.:

```
ARM926EJS_0 step_core
```

```
VIC_1/EnableMask get_value
```

TCL Scripting Interface

Control & Inspection Layer



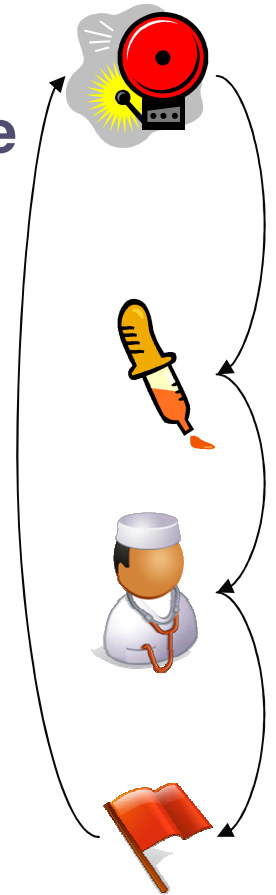
Scripting For Debug

Tracing

Asserting

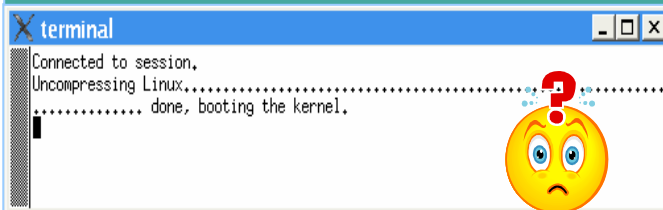
Principle:

- **Notify and react on system events**
 - Register, memory, pin access and change
 - Program control (e.g. Function call)
- **Inspect state**
 - Register, memory and pin values
- **Validate**
 - Assert correctness
- **Feedback assertion result**
 - Stop or carry state to next assertion



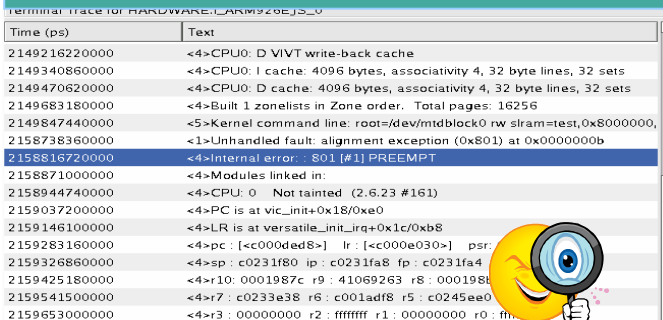
Typical Linux Boot Problem

1) Linux Boot UART Console



- Symptom: **It does not boot!**
- What worked, what did not work?
- No helpful kernel debug messages?
- UART driver not yet working!
- Debug messages would be helpful!

3) Kernel Debug Messages via VP



2) VP Debug Helper Script (TCL)

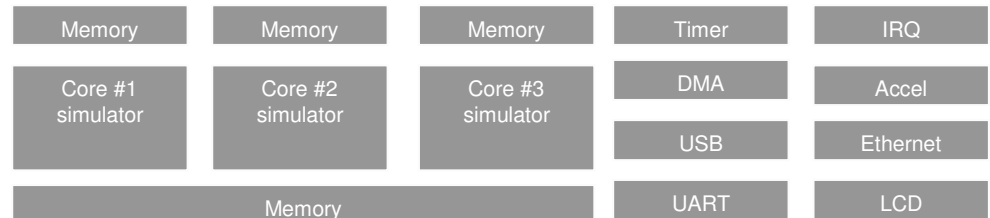
```
# 1: Create breakpoint at debug emit function
set bp [create_breakpoint "emit_log_char"]

# 2: Attach callback procedure
$bp set_callback tcl_emit_log_char

# 3: Callback procedure: print character from
      CPU register
proc tcl_emit_log_char {} {
    set c [ {i_ARM926EJS_0/R/R[0]} get_value]
    puts -nonewline "[format "%c" $c]"
}
```

Virtual Platform Simulation Control & Inspection Interface

Virtual Platform



Efficient debugging via non-intrusive control, visibility and scripted automation.
Increased debug productivity!

CoWare®

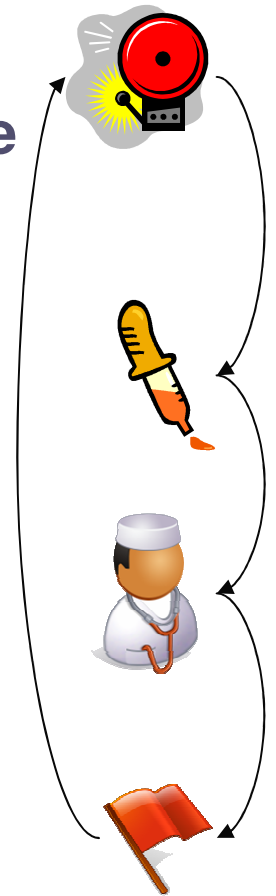
Scripting For Debug

Tracing

Asserting

Principle:

- **Notify and react on system events**
 - Register, memory, pin access and change
 - Program control (e.g. Function call)
- **Inspect state**
 - Register, memory and pin values
- **Validate**
 - Assert correctness
- **Feedback assertion result**
 - Stop or carry state to next assertion



Kernel Memory Corruption

“My kernel shows sporadic kernel panic problems.
How can I assert a memory corruption in the
kernel?”

Memory corruption defects severely increase the system vulnerability

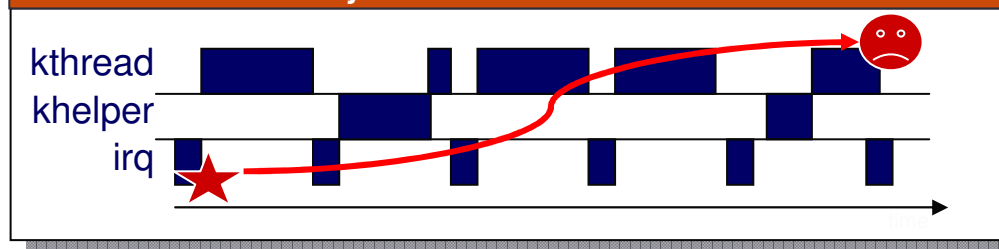
Kernel Memory Corruption Categories

Faulty heap memory mgmt.	Read un-initialized memory	Buffer overrun
--------------------------	----------------------------	----------------

Kernel Memory Corruption Symptoms

Memory leaks Kernel heap corruption	Most times immediate unpredictable kernel behavior	<u>Delayed</u> crash of kernel activities (scheduler, threads...)
--	---	--

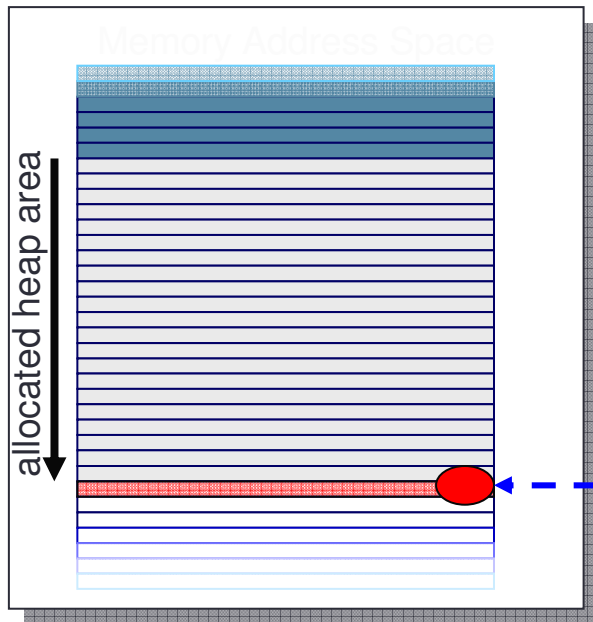
History of a buffer overrun defect



Typical code prone to a buffer overrun

```
char * tmp = (char*) malloc(strlen(str));  
strcpy(tmp, str);
```

Kernel Memory Corruption



Linux kernel function:

`slob_page_alloc(int size, int align, int addr)`

GO!

Virtual Platform Callback (Pseudo Code)

- Probe parameters: address, size
- Set watchpoint at address + size

set watchpoint

```
proc callbackallocExit () {
    # This callback gets called when the allocation is completed
    global malloc_size
    global malloc_pos
    global malloc_memory

    # The returned address will be stored in malloc_pos
    set malloc_pos [ (i,MMIO2EEJ3_0/R/R(0)) get_value ]

    # Store in table malloc_memory at pos malloc_pos the size of the allocated area
    set malloc_memory[(expr ($malloc_pos))] $malloc_size

    # The next address behind the allocated area will be stored in next
    set next [(expr ($malloc_pos + $malloc_size))]

    # Trace:
    puts "malloc @[format "%x" $malloc_pos] with $malloc_size = [format "%x" $next]"
    # If there is no memory allocated at the end of the allocated area,
    # then set a watchpoint to protect this address from being written
    if ( [ catch {} set sizeNext $malloc_memory($next) ] == 0 ) {
        if ( $sizeNext == 0 ) {
            create_sv_watchpoint $next write virtual
        }
    } else {
        create_sv_watchpoint $next write virtual
    }

    # This area is now allocated, remove a potential watchpoint here
    remove_sv_watchpoint $malloc_pos

    # Trace the allocation
    callbackallocTrace $malloc_size
    return
}
```

Kernel Memory Corruption

Demo: Linux SLOB (Simple List Of Blocks) Allocator – Virtual Platform Assertion

The screenshot displays the Virtual Platform Analyzer (VPA) interface. At the top left, a terminal window shows the boot process: "Connected to session. Uncompressing Linux... done, booting the kernel." A large red rectangle is overlaid on the terminal.

The main window is titled "Virtual Platform Analyzer 2009.1.0 : vpa.maisel.22121 : VP-TOOLS - [mymallocTraceW...". It features a menu bar (File, Simulation, Debug, Analysis, View, Windows, Properties, Connectivity, Linux, Documentation, Help) and a toolbar with icons for simulation control.

On the left, a tree view shows hardware components under "HARDWARE", including "BOOT_0", "CLCDC_Memory", "i_ARM926EJS_0", and various peripheral controllers like "i_CLCDC_PL111", "i_DMACH_PL080", "i_GPIO_PL061_0" through "3", "i_KMI_0" through "1", "i_PIO", "i_RTC_PL031", "i_SCI_PL130", "i_SIC", "i_SmartCard", "i_SMC91C111", and "i_SSP_PL022".

The central area displays a "Heap Memory Allocation Trace" graph. The y-axis represents memory size (0 to 400) and the x-axis represents memory address (224530000000 to 224550000000). The graph shows a series of steps, indicating memory allocation events.

At the bottom left, a "Views Overview" window shows a list of memory allocation events:

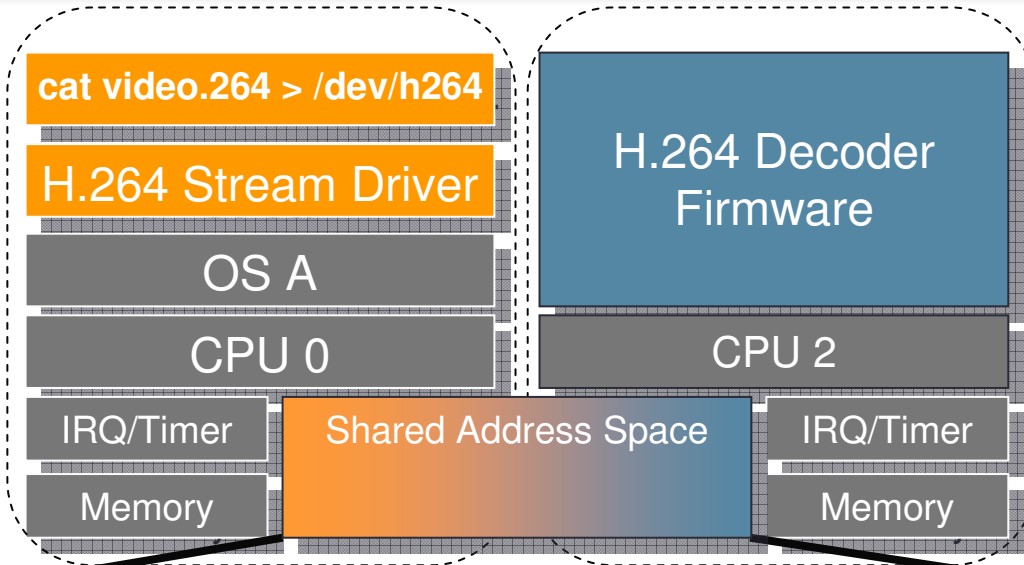
```
malloc @c033f078 with 20 = c033f08c
malloc @c033f08c with 20 = c033f0a0
malloc @c033f0a0 with 20 = c033f0b4
malloc @c033f0b4 with 20 = c033f0c8
malloc @c033f0c8 with 20 = c033f0dc
malloc @c033f0dc with 20 = c033f0f0
malloc @c033f0f0 with 20 = c033f104
malloc @c033f104 with 20 = c033f118
malloc @c033f118 with 20 = c033f12c
malloc @c033f12c with 20 = c033f140
malloc @c033f140 with 20 = c033f154
malloc @c033f160 with 108 = c033f1cc
malloc @c033f154 with 9 = c033f15d
```

The main window displays source code for "fs/namespace.c" with a cursor at line 72. The code includes functions like "INIT_LIST_HEAD", "simple_set_mnt", and "free_vfsmnt". A "DDD: Backtrace" window is open, showing the call stack:

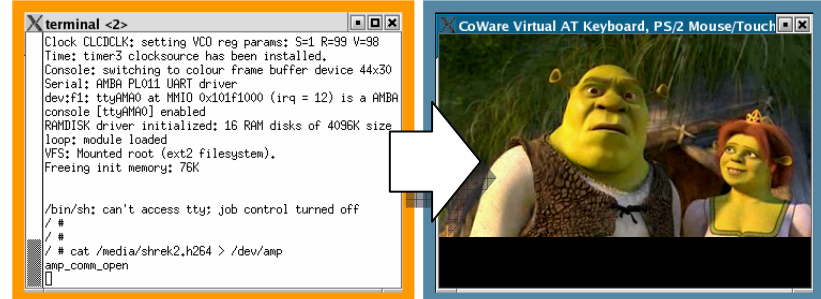
```
Backtrace
#7 0xc0008d4c in start_kernel () at main.c:638
#6 0xc0011b68 in vfs_caches_init () at dcache.c:2176
#5 0xc0011f1c in mnt_init () at namespace.c:1840
#4 0xc00129cc in sysfs_init () at mount.c:95
#3 0xc0080294 in kern_mount () at super.c:952
#2 0xc00801fc in vfs_kern_mount () at super.c:870
#1 0xc0095a14 in alloc_vfsmnt () at namespace.c:72
#0 0xc0105980 in strcpy () at string.c:102
```

The status bar at the bottom shows "Stoppers: vpa gdb@127.0.0.1:37163 | 0:00:02.245 508 540 000".

Multi-core Driver Example



H.264 Stream Device Example



Linux H.264 stream device

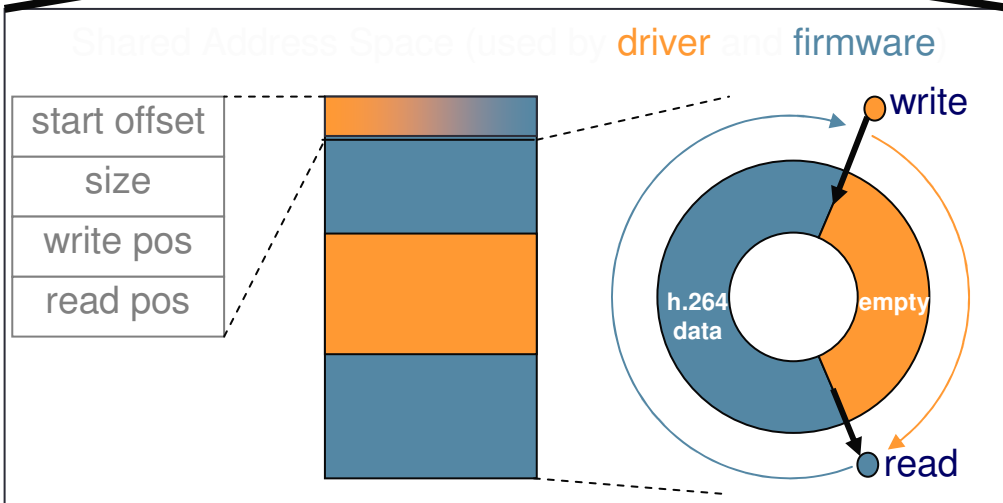
- Driver on CPU 0, provides data to
- H.264 decoder firmware on CPU 2

Control/Synchronization

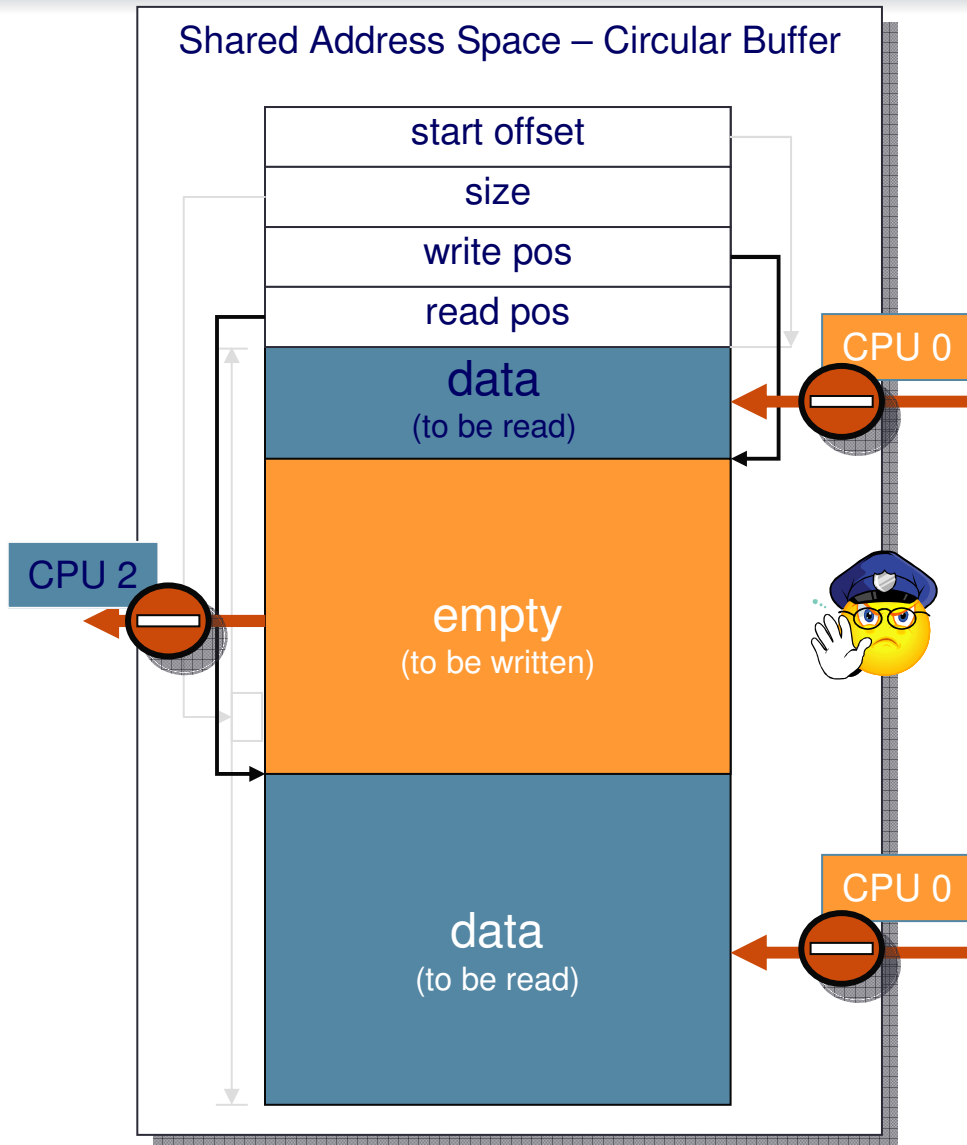
- Interrupts
- Mutex/Semaphores

Data streaming

- Circular buffer in shared memory



Multi-core Driver Example



Implementation issues

- Buffer overrun
- Race condition/data corruption
- Starvation

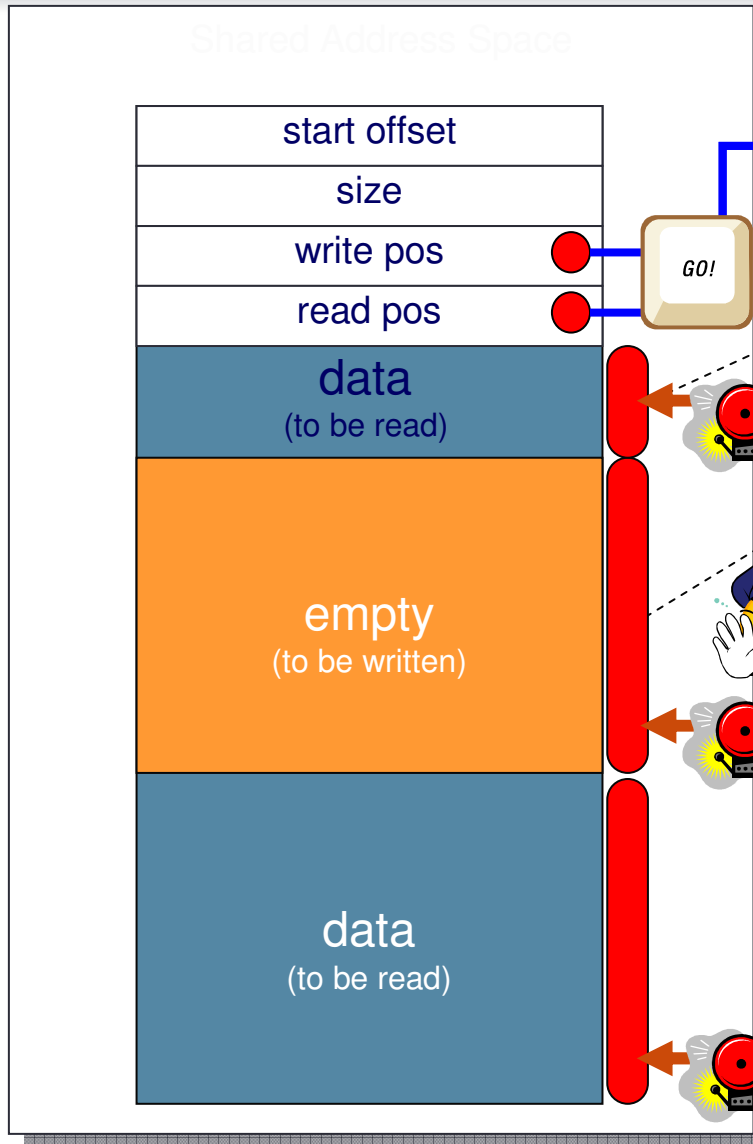
Debugging challenges

- Become aware of a defect
- Sporadic decoding errors
(e.g. frames dropped)

System level software assertion

- Protected address regions,
- through VP region watch-points
- Dynamically adjusted,
- on every circular buffer update

Multi-core Driver Example



Virtual Platform Callback Pseudo Code

```
update_watch_region_cpu_0  
update_watch_region_cpu_1
```

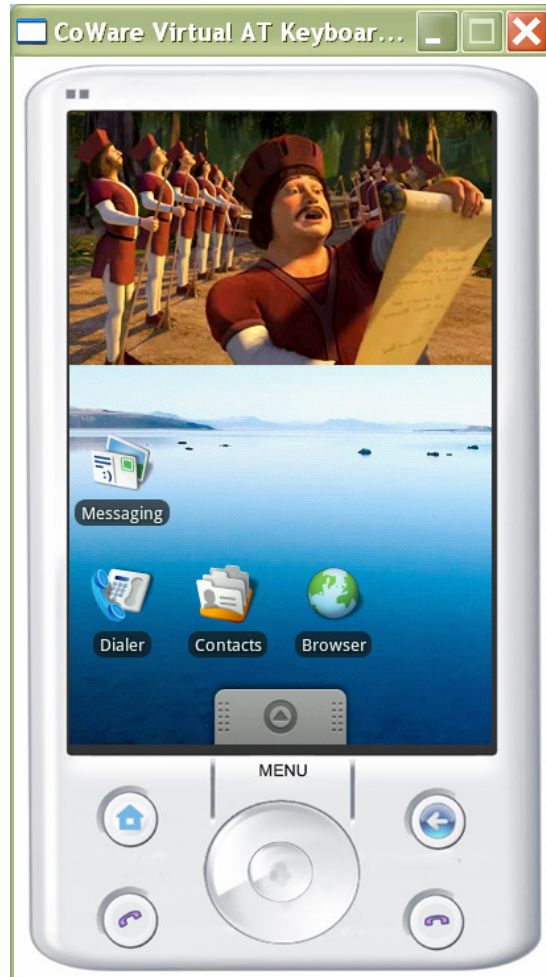
```
CPU_0 set_range_watchpoint read  
      $start_offset $write_pos-1
```

```
CPU_1 set_range_watchpoint write  
      $write_pos $read_pos-1
```

System level software assertion

- Watch-points on circular buffer pointers
- trigger updates of the watch-regions.
- Platform execution is suspended,
- as soon as memory is accessed illegally.

Multi-core Driver Example

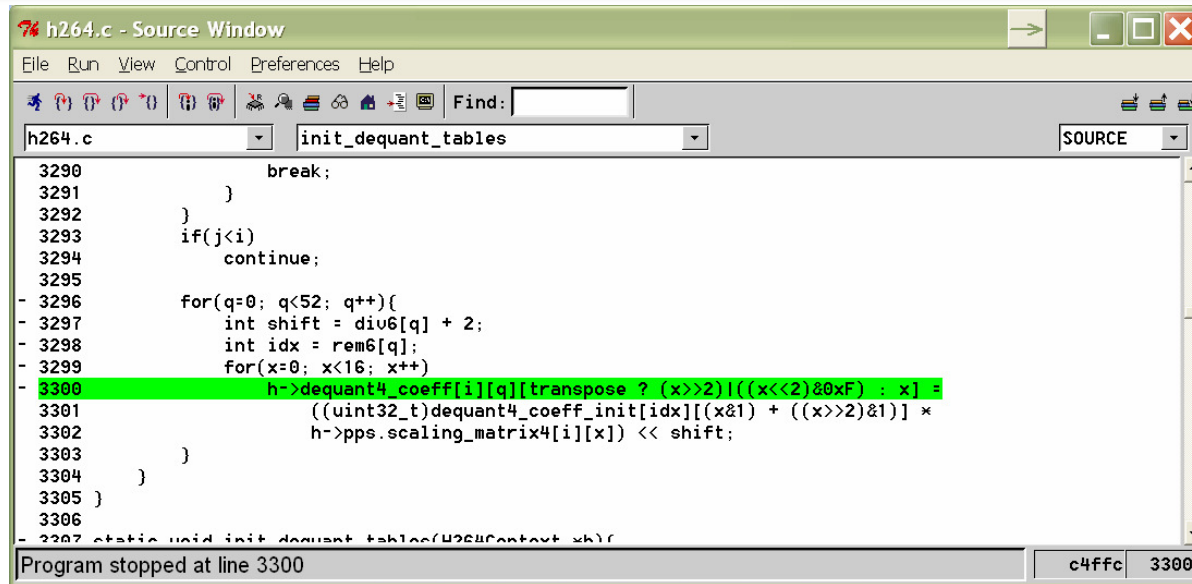


The image shows the "Virtual Platform Analyzer 2009.1.1" interface. The top menu includes "File", "Simulation", "Debug", "Analysis", "View", "Windows", "Linux", "Connectivity", "Observers", "SW Debug", and "Help". The left pane shows a tree view of hardware components, including "HARDWARE", "BOOT_0" through "BOOT_3", "CLCDC_Memory", and four "i_ARM926EJS" cores. Below this is a donut chart titled "ARM968/ARM926 Shared Memory I/O" with a legend: ARM968 - Read Data (green), ARM926 - Write Data (orange), and ARM926 - Oldest Data (red). The main area displays two disassembly windows. The top window, "i_ARM926EJS_2 Disassembly", shows instructions at address 0xc002c420, with the instruction "MCR p15, #0" highlighted in yellow. The bottom window, "i_ARM926EJS_0 Disassembly", shows instructions at address 0x00091288, with the instruction "LDRB R3, [R8, #0]" highlighted in yellow. The status bar at the bottom indicates "Stoppers: vpa 0:07:36.175 857 780 000".

BP	Symbols	Address	Instruction	Disassembly
		[c002c410]	e10f3000	MRS R3, CPS
		[c002c414]	e383c040	ORR R12, R3
		[c002c418]	e121f00c	MSR CPSR_c
		[c002c41c]	ee012f10	MCR p15, #0
→		[c002c420]	ee070f90	MCR p15, #0
		[c002c424]	ee011f10	MCR p15, #0
		[c002c428]	e121f003	MSR CPSR_c
		[c002c42c]	e1a0f00e	MOV R15, R1
		[c002c430]	e3a02004	MOV R2, #4

BP	Symbols	Address	Instruction	Disassembly
		[00091278]	e1a02282	MOV R2, R2,
		[0009127c]	e1822581	ORR R2, R2, R
		[00091280]	e18221a0	ORR R2, R2, R
		[00091284]	e1c620b2	STRH R2, [R6,
→		[00091288]	e5d83000	LDRB R3, [R8,
		[0009128c]	e3550001	CMP R5, #1
		[00091290]	e0832183	ADD R2, R3, R
		[00091294]	e0832102	ADD R2, R3, R
		[00091298]	e0833102	ADD R3, R3, R
		[0009129c]	e1a03183	MOV R3, R3,

Multi-core Driver Example

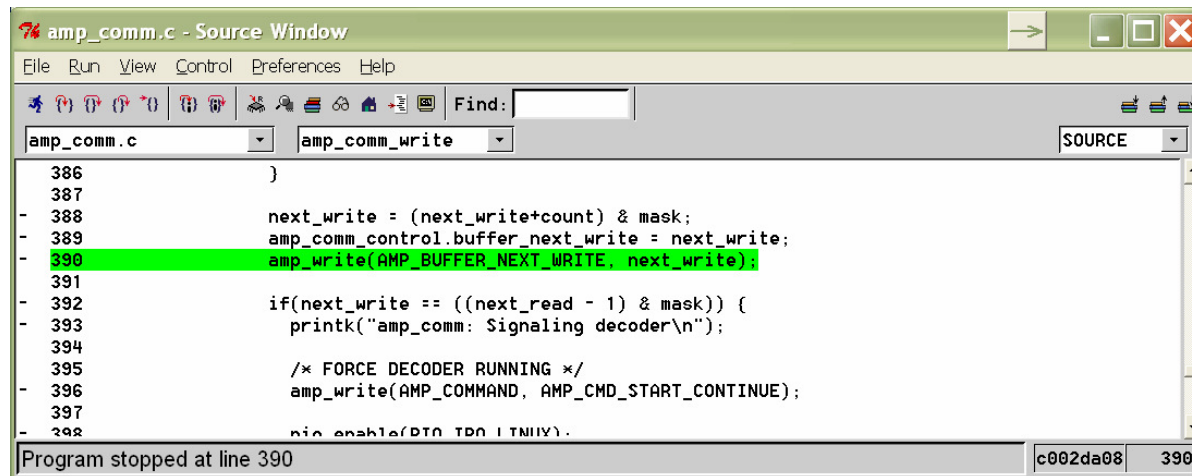


h264.c - Source Window

```
File Run View Control Preferences Help
h264.c init_dequant_tables SOURCE
3290     break;
3291     }
3292     }
3293     if(j<i)
3294         continue;
3295
- 3296     for(q=0; q<52; q++){
- 3297         int shift = div6[q] + 2;
- 3298         int idx = rem6[q];
- 3299         for(x=0; x<16; x++){
- 3300             h->dequant4_coeff[i][q][transpose ? (x>>2)|((x<<2)&0xF) : x] =
3301                 ((uint32_t)dequant4_coeff_init[idx][(x&1) + ((x>>2)&1)] *
3302                 h->pps.scaling_matrix4[i][x]) << shift;
3303         }
3304     }
3305 }
3306
- 3307 static void init_dequant_tables(H264Context *h){
```

Program stopped at line 3300

c4ffc 3300



amp_comm.c - Source Window

```
File Run View Control Preferences Help
amp_comm.c amp_comm_write SOURCE
386     }
387
- 388     next_write = (next_write+count) & mask;
- 389     amp_comm_control.buffer_next_write = next_write;
- 390     amp_write(AMP_BUFFER_NEXT_WRITE, next_write);
391
- 392     if(next_write == ((next_read - 1) & mask)) {
393         printk("amp_comm: Signaling decoder\n");
394
395         /* FORCE DECODER RUNNING */
- 396         amp_write(AMP_COMMAND, AMP_CMD_START_CONTINUE);
397
- 398         nic_enable(P10, TPO, LINIX);
```

Program stopped at line 390

c002da08 390

Summary

- We have used Virtual Platforms...
 - to identify, analyze and assert software defects.
 - by means of real-world hardware and software examples.
- We have seen Virtual Platforms...
 - provide non-intrusive and deterministic
 - Control & Visibility
 - enable novel debug solutions,
 - with less guessing and more analysis,
 - resulting in increased productivity,
 - for embedded software development.



CoWare[®]

Thank You!

CoWare®