Achim Nohl, CoWare Inc
Booth Number 3665 located in the North Hall

Getting started with Virtual Platforms: A Software Developer Prespective

Virtual Platform Workshop
DAC 09, San Francisco

Objective

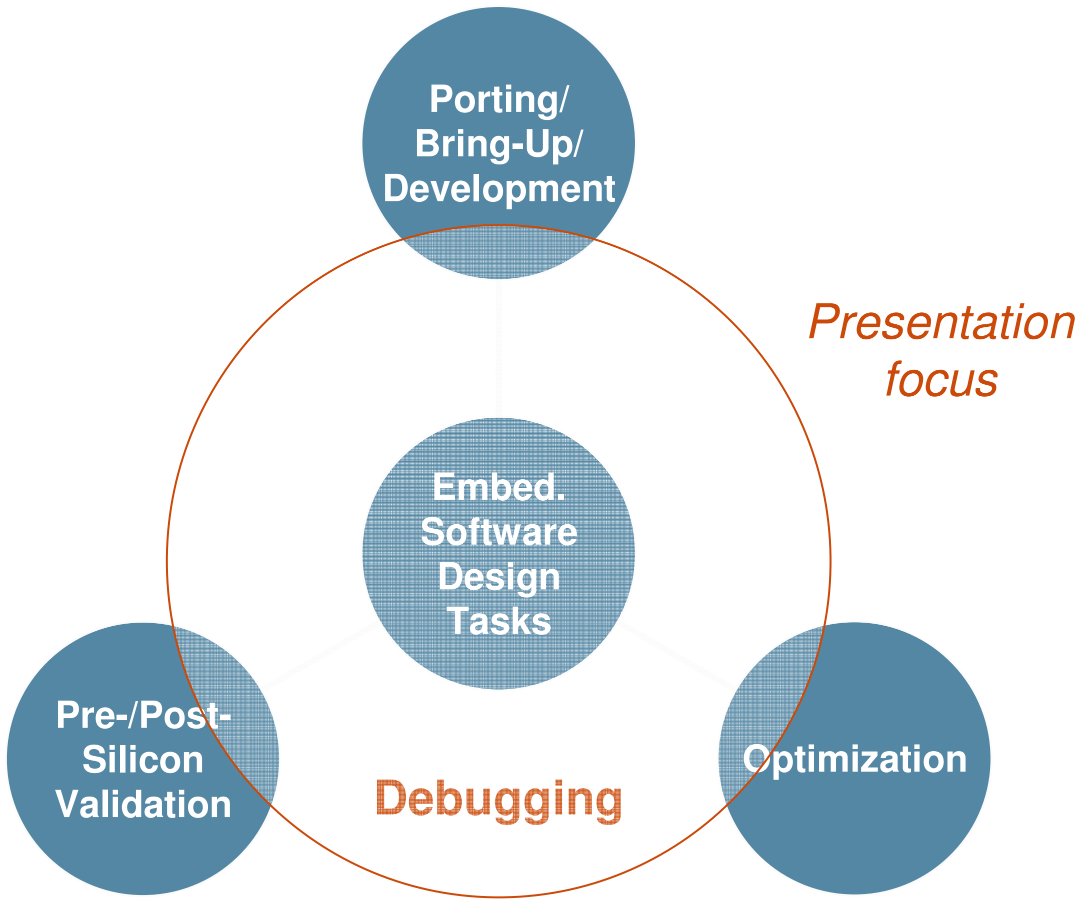
Objective: Provide illustrative examples on how Virtual Platforms are used for debugging.

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

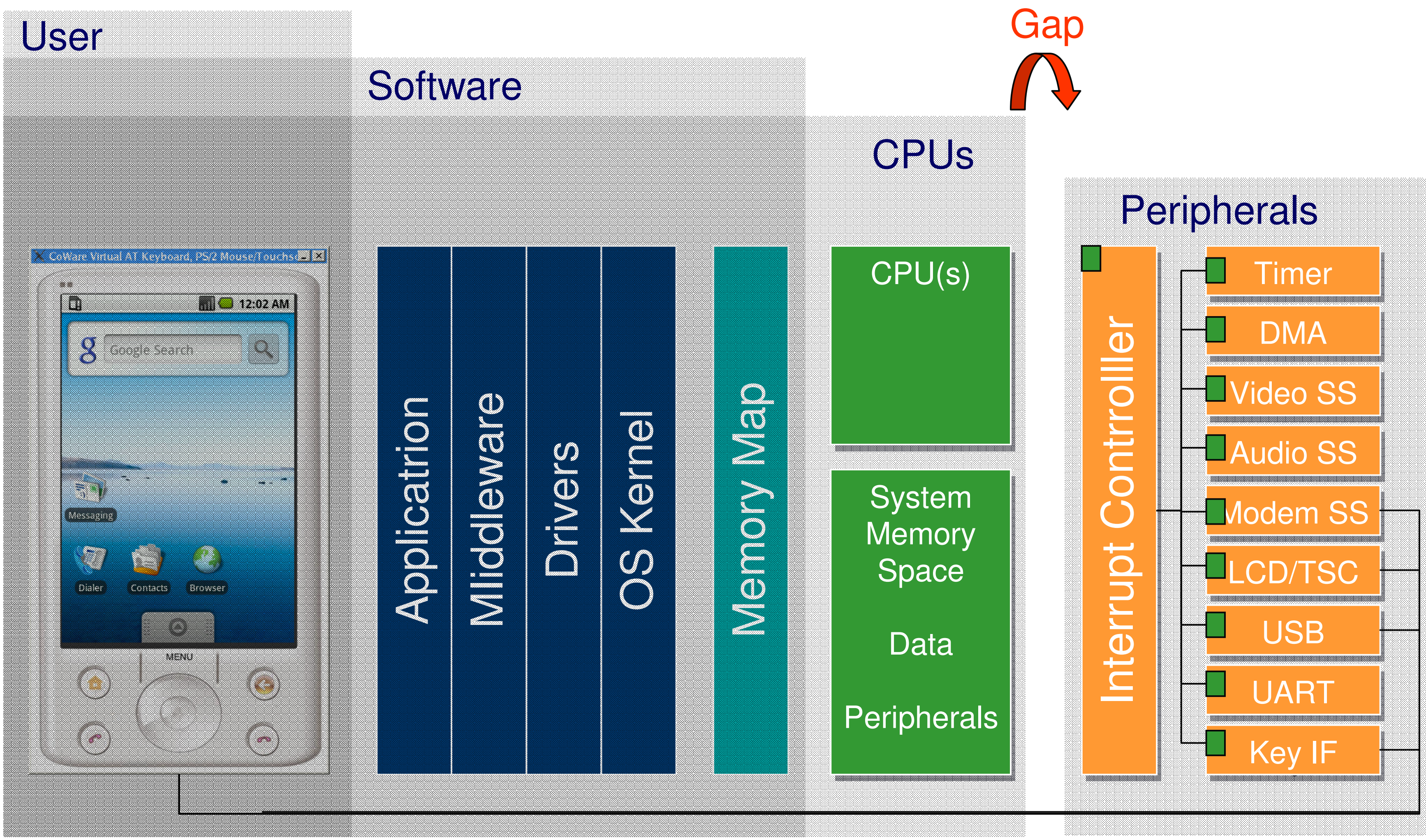
Outline

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

Embedded Software Design Tasks



Perspectives



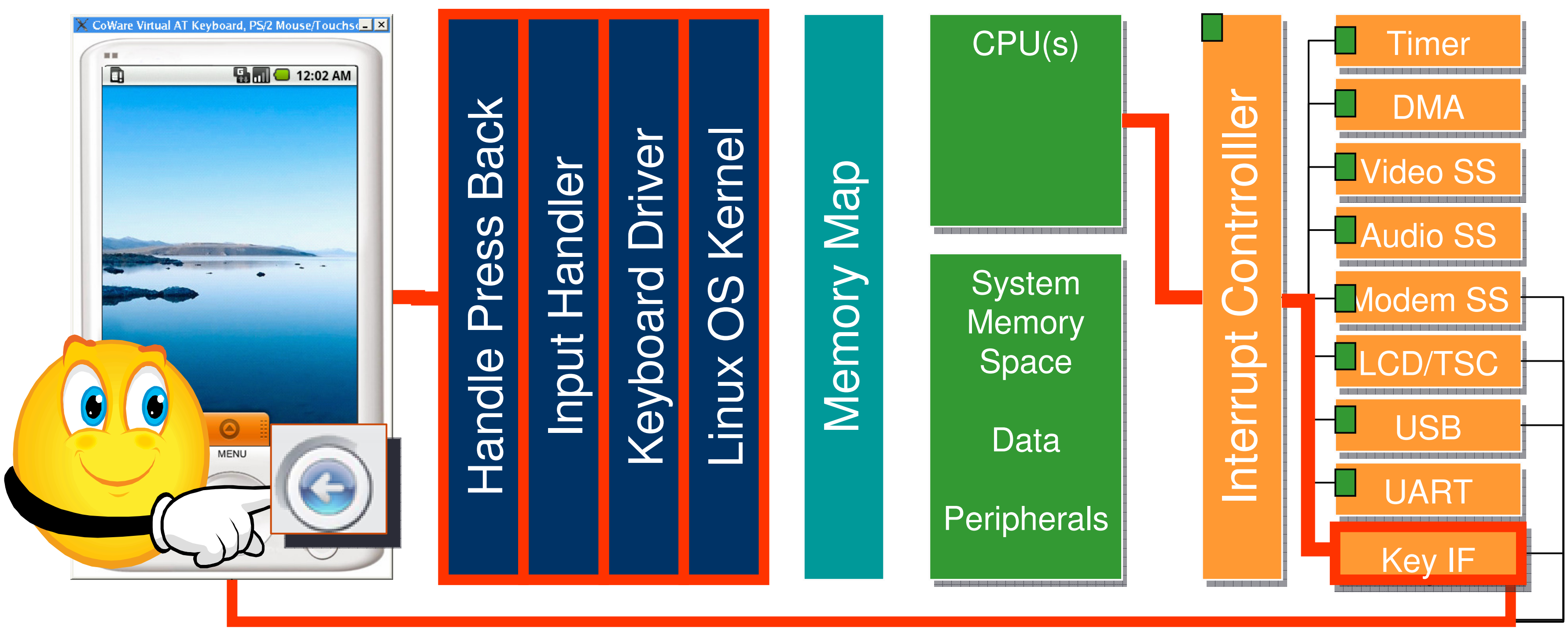
User's View

CoWare Virtual AT Keyboar...
12:02 AM
Google Search
Messaging
Dialer
Contacts
Browser
MENU

CoWare Virtual AT Keyboar...
12:02 AM
Alarm Clock
Browser
Calendar
Camcorder
Camera
Contacts
Dialer
Gallery
Messaging
Music
Settings
Spare Parts
MENU

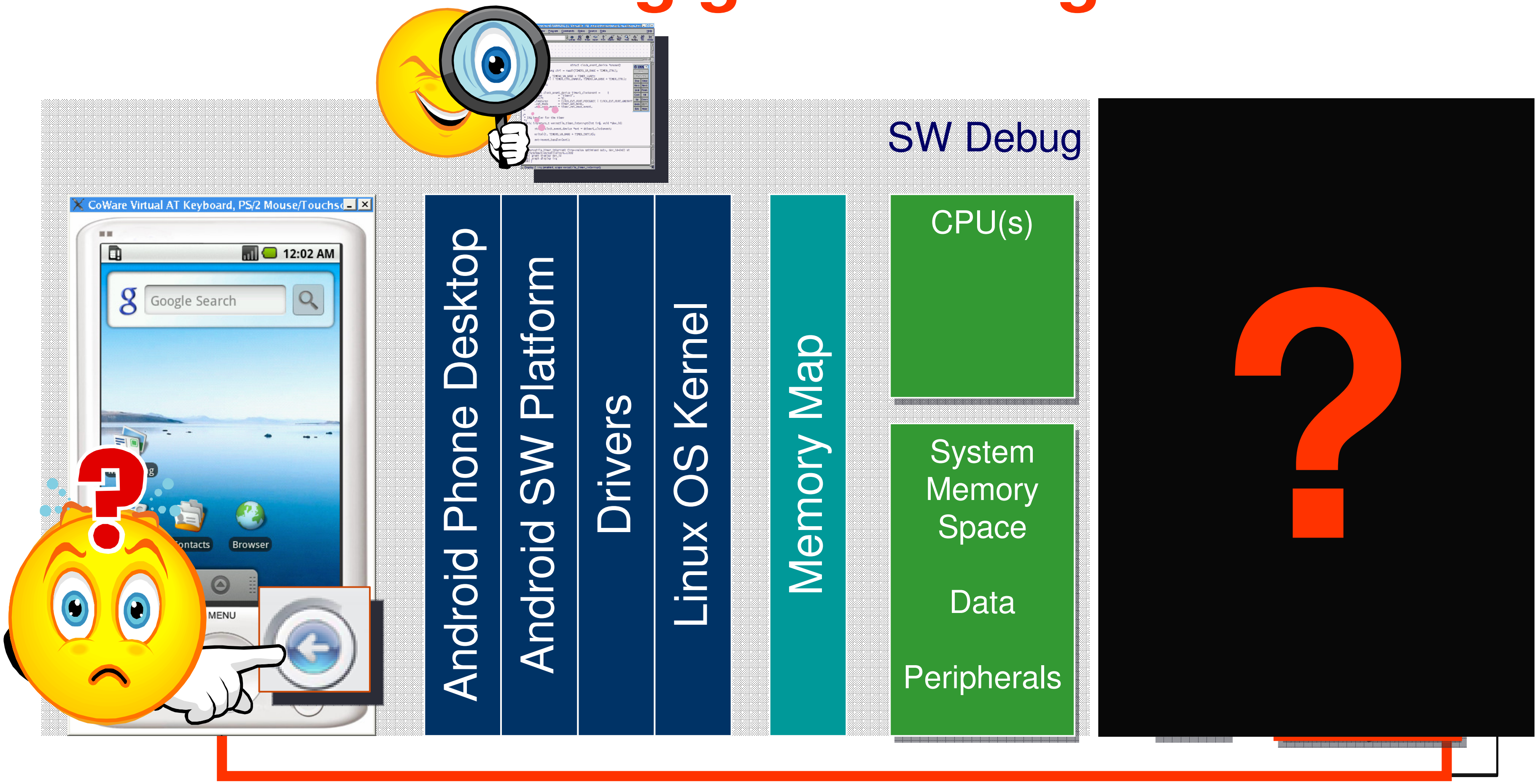
CoWare Virtual AT Keyboar...
12:02 AM
www.dac.com: 46th DAC Home: Desi...
46th DAC Keynote Sessions
Interested in the EXHIBITION?
Interested in the TECHNICAL CONFERENCE?
Interested in the MANAGEMENT DAY?
Where electronic design meets
MENU

Example:Device Key Press



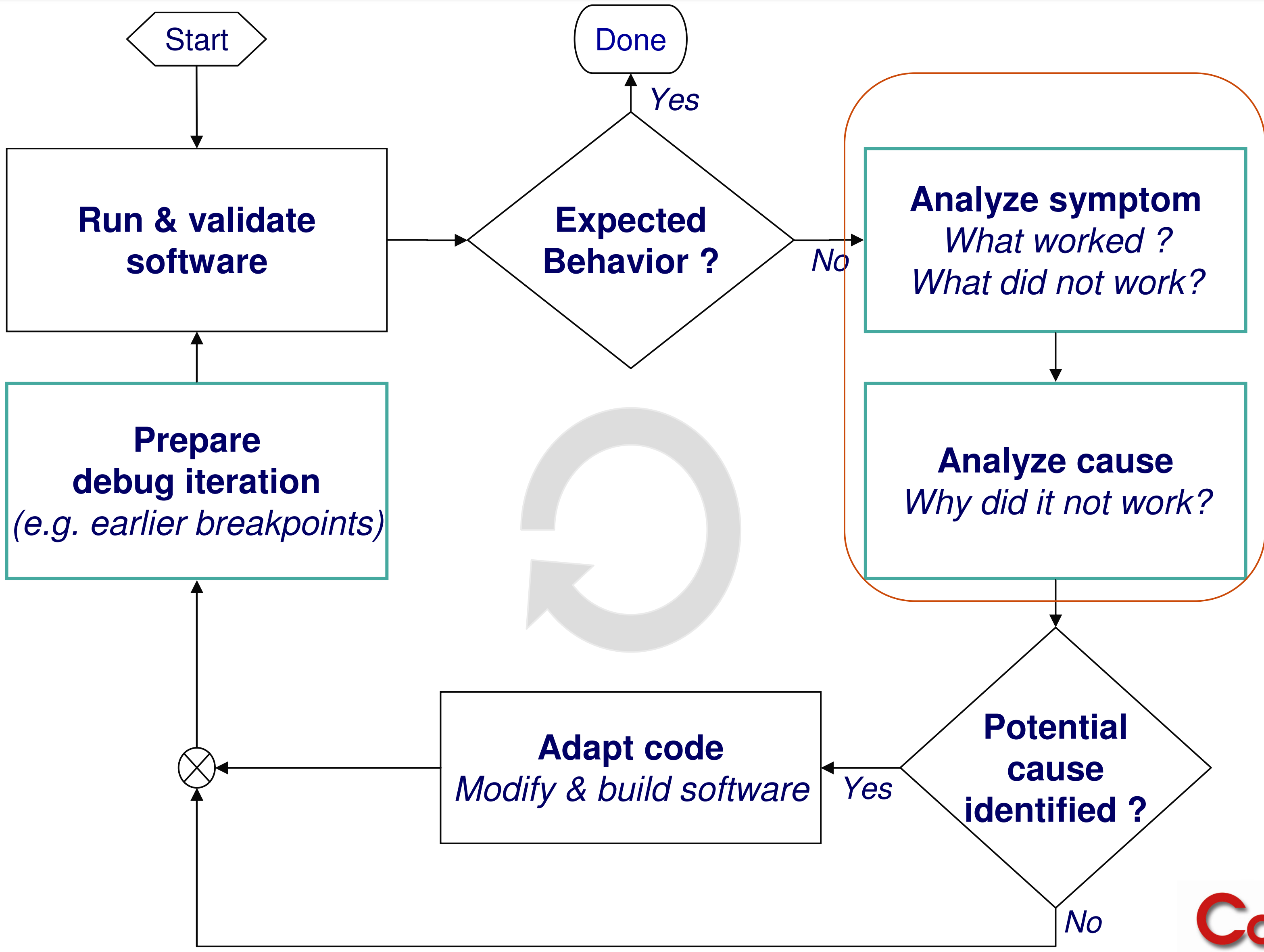
Example:Device Key Press

What if something goes wrong ?



CoWare

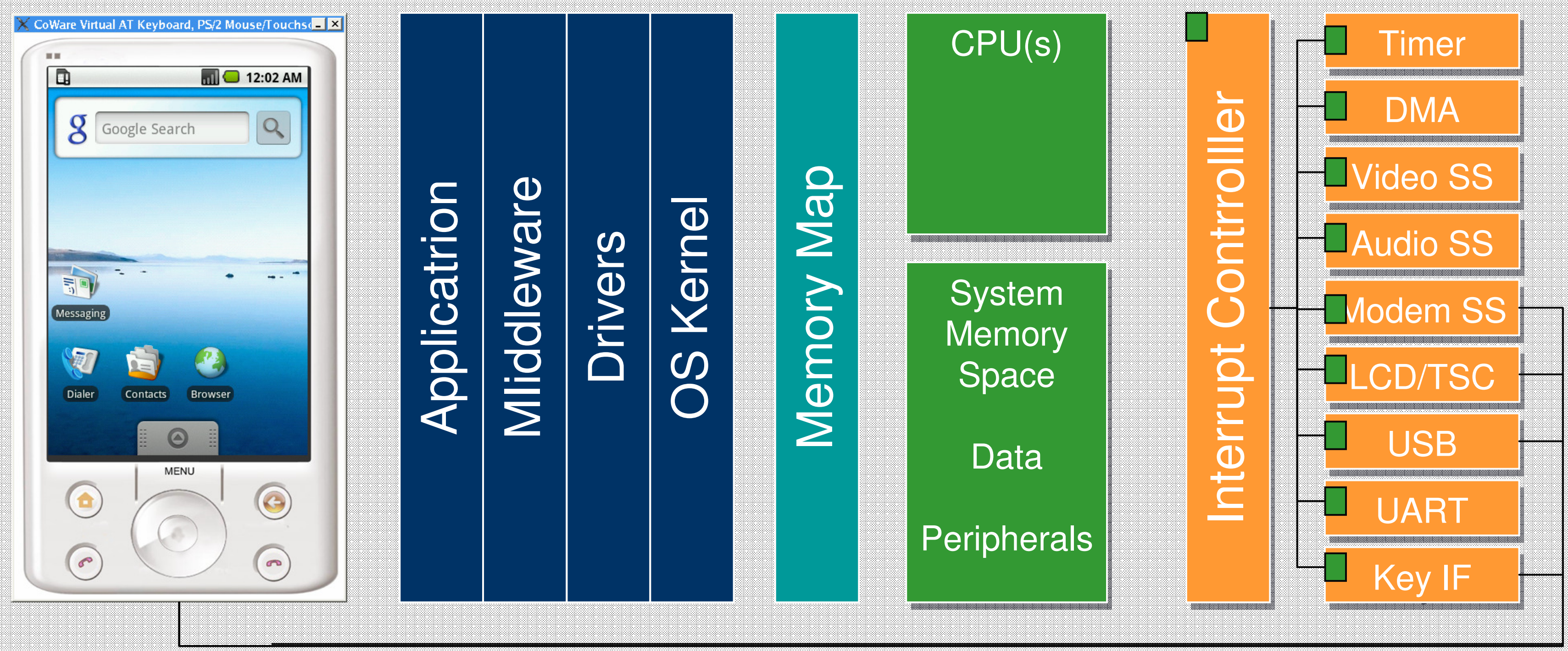
Debugging Process



Virtual Platform Perspective

Synchronous system control
Full and consistent system visibility

Virtual Platform Based Debug Perspective



CoWare®

Time and Space During Debug

Software initiatet system activity:

- Example: Timer dies
- What <u>code</u> should I look at and debug?

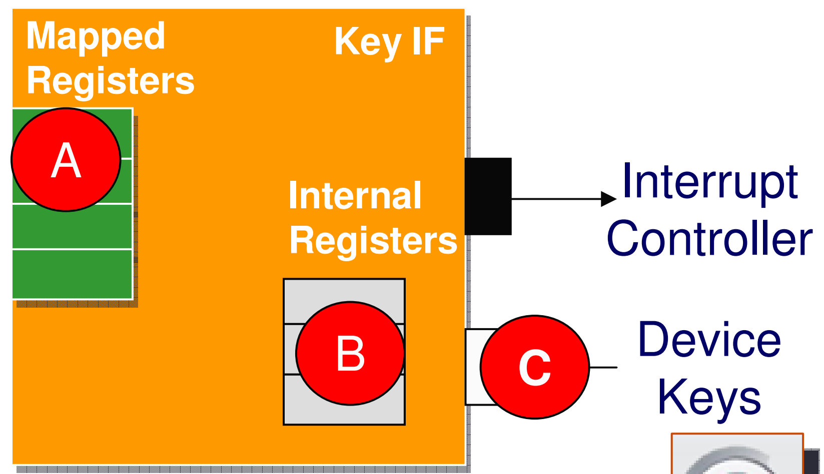
Hardware initiated system activity:

- Example: Device key press
- What <u>time</u> should I stop to debug?

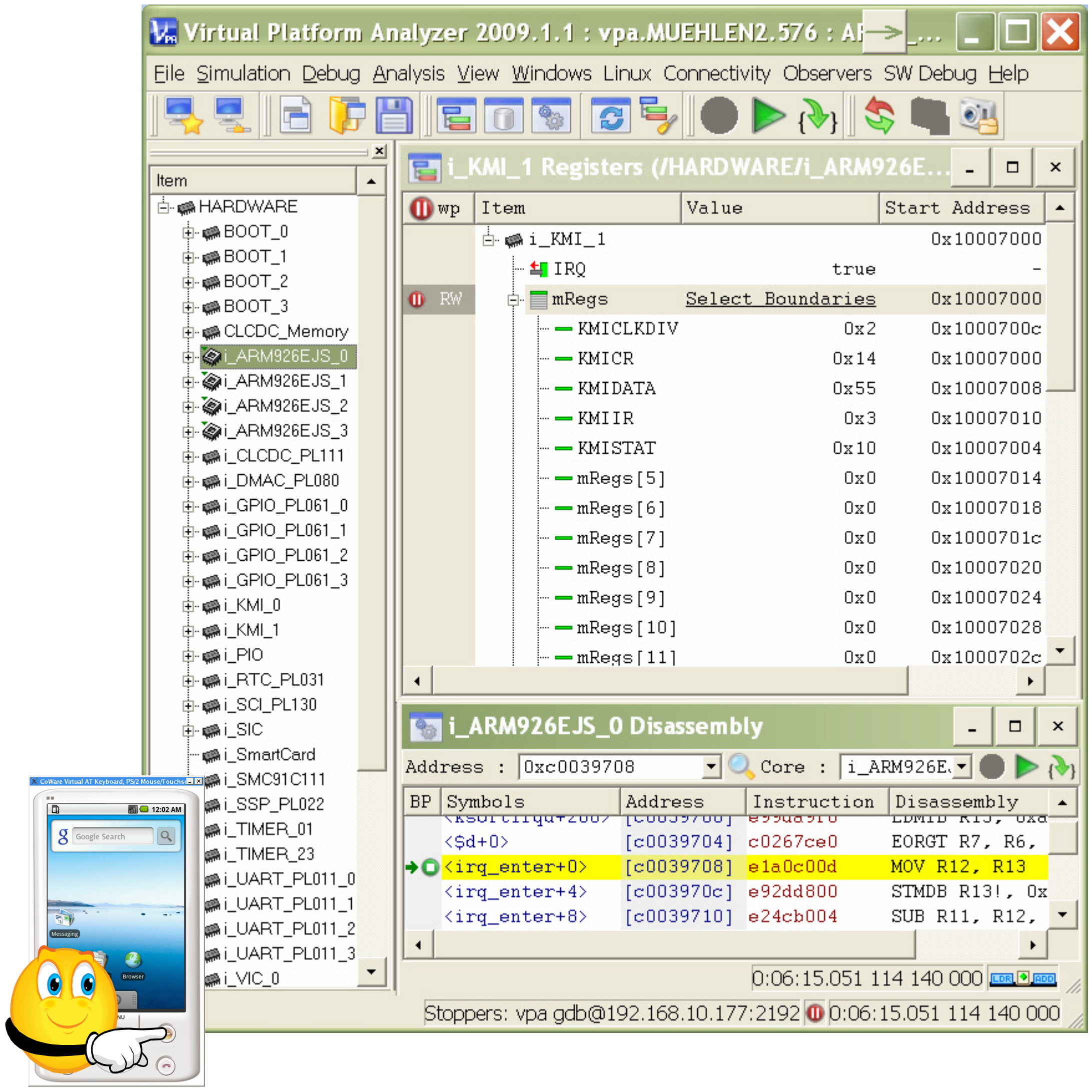
CoWare®

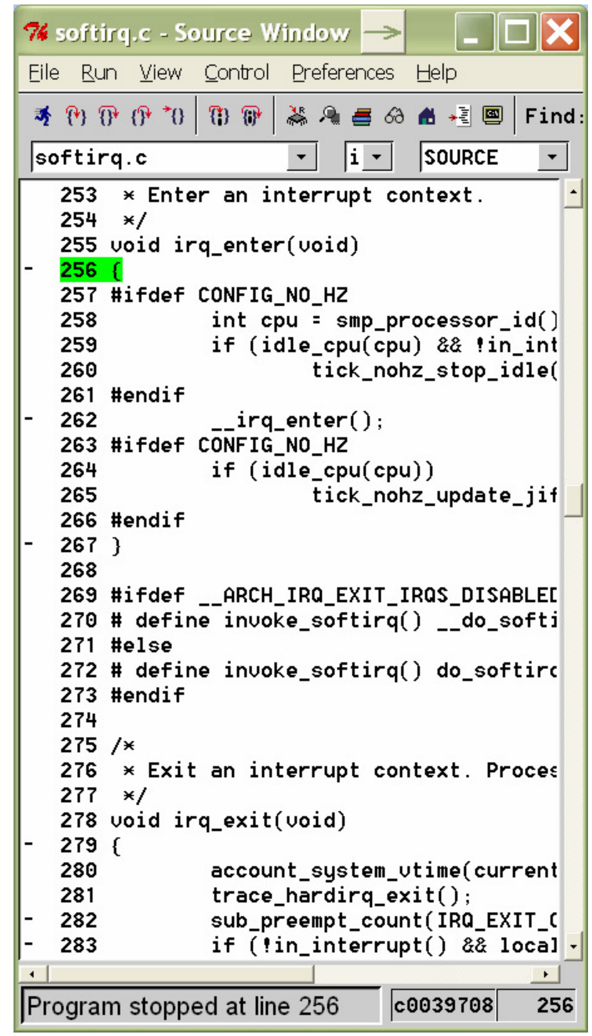
Watchpoints: Fast Track to the Problem

- **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**



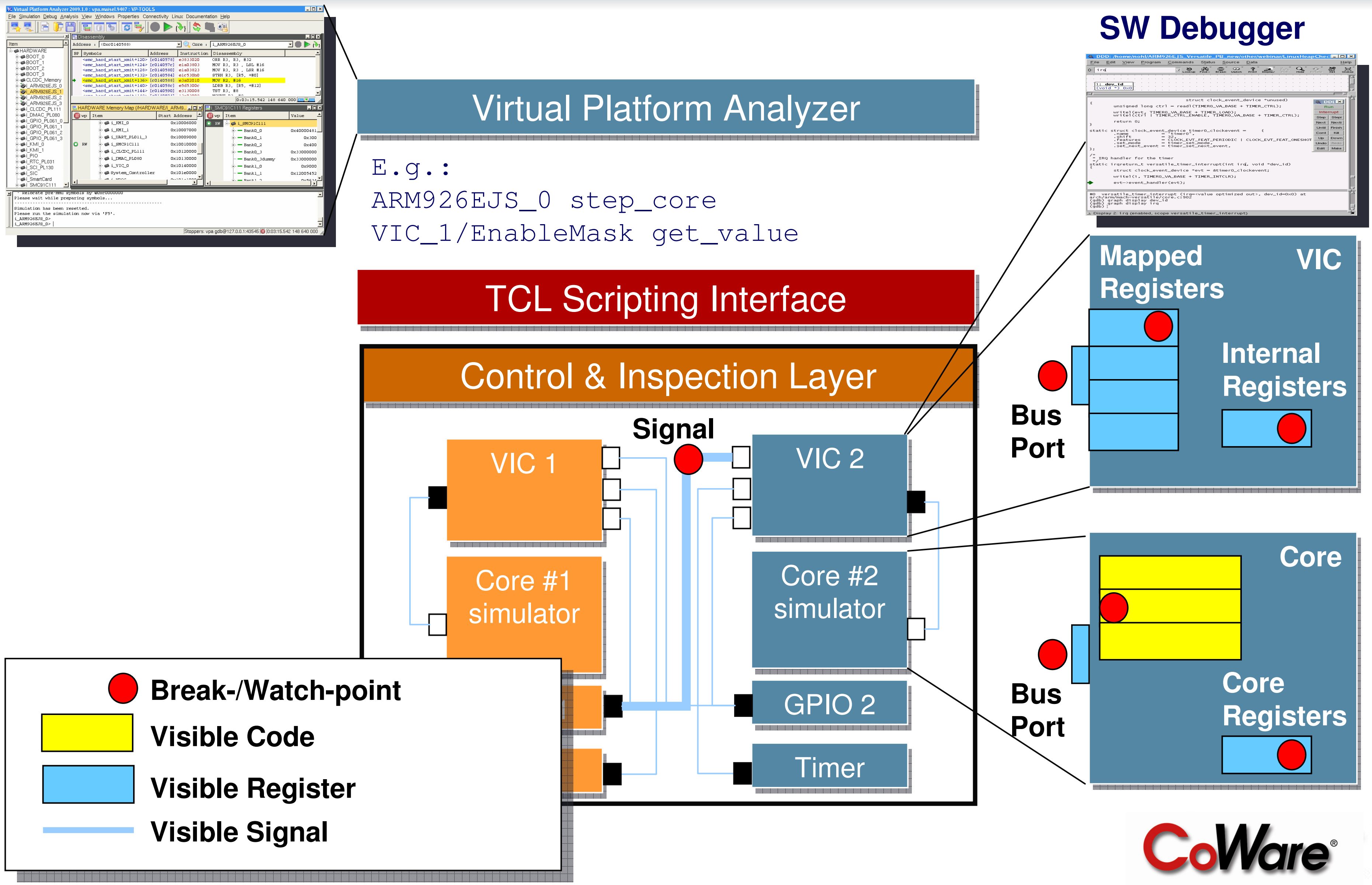
Watchpoints





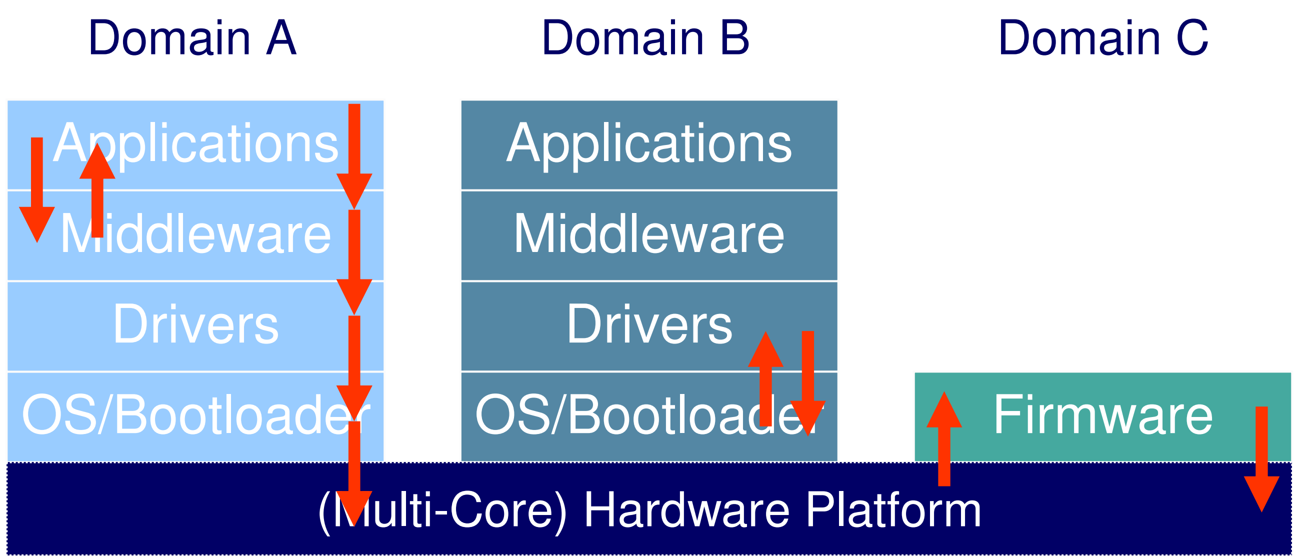
CoWare®

Platform Level Software Debugging



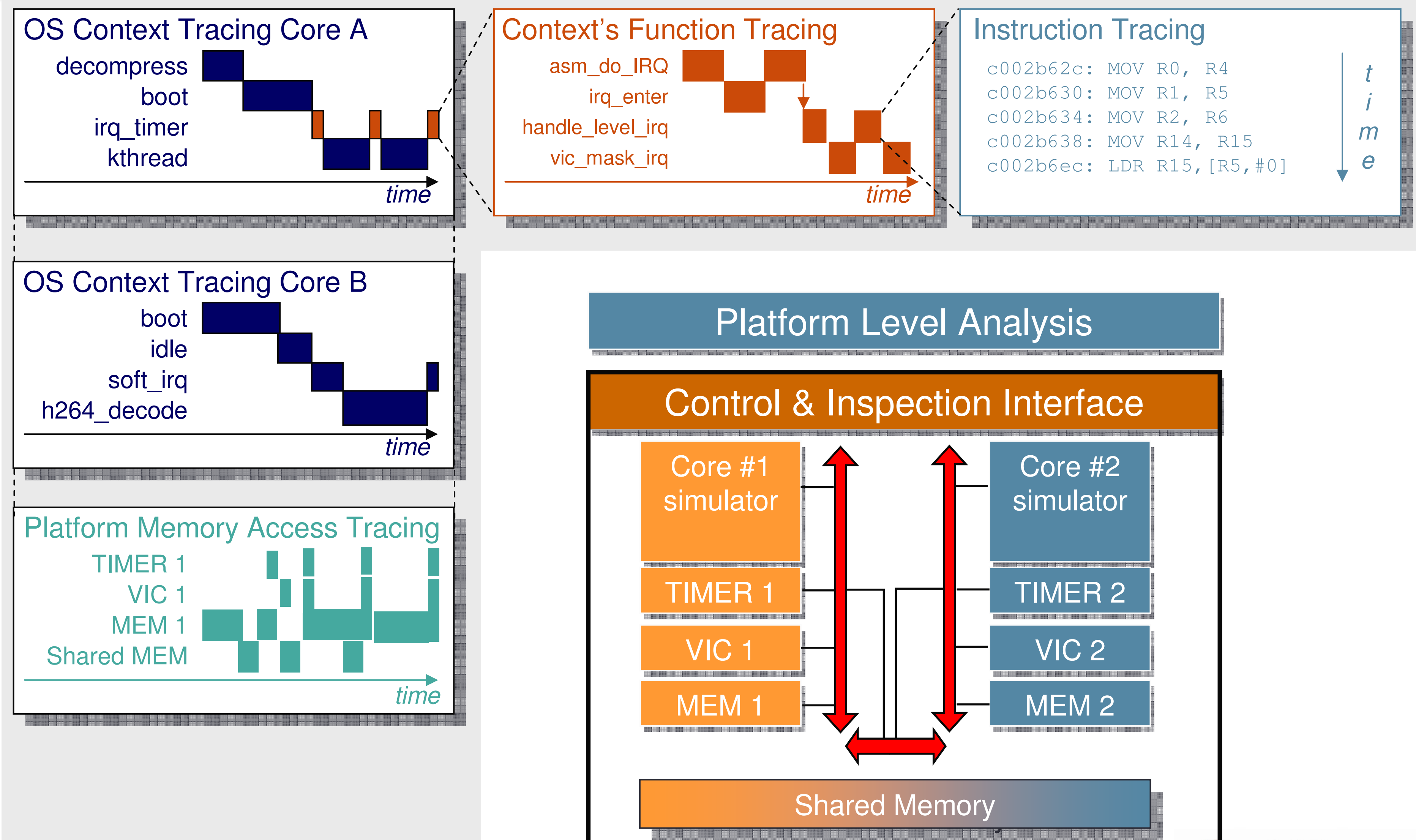
CoWare®

Platform Level Software Analysis



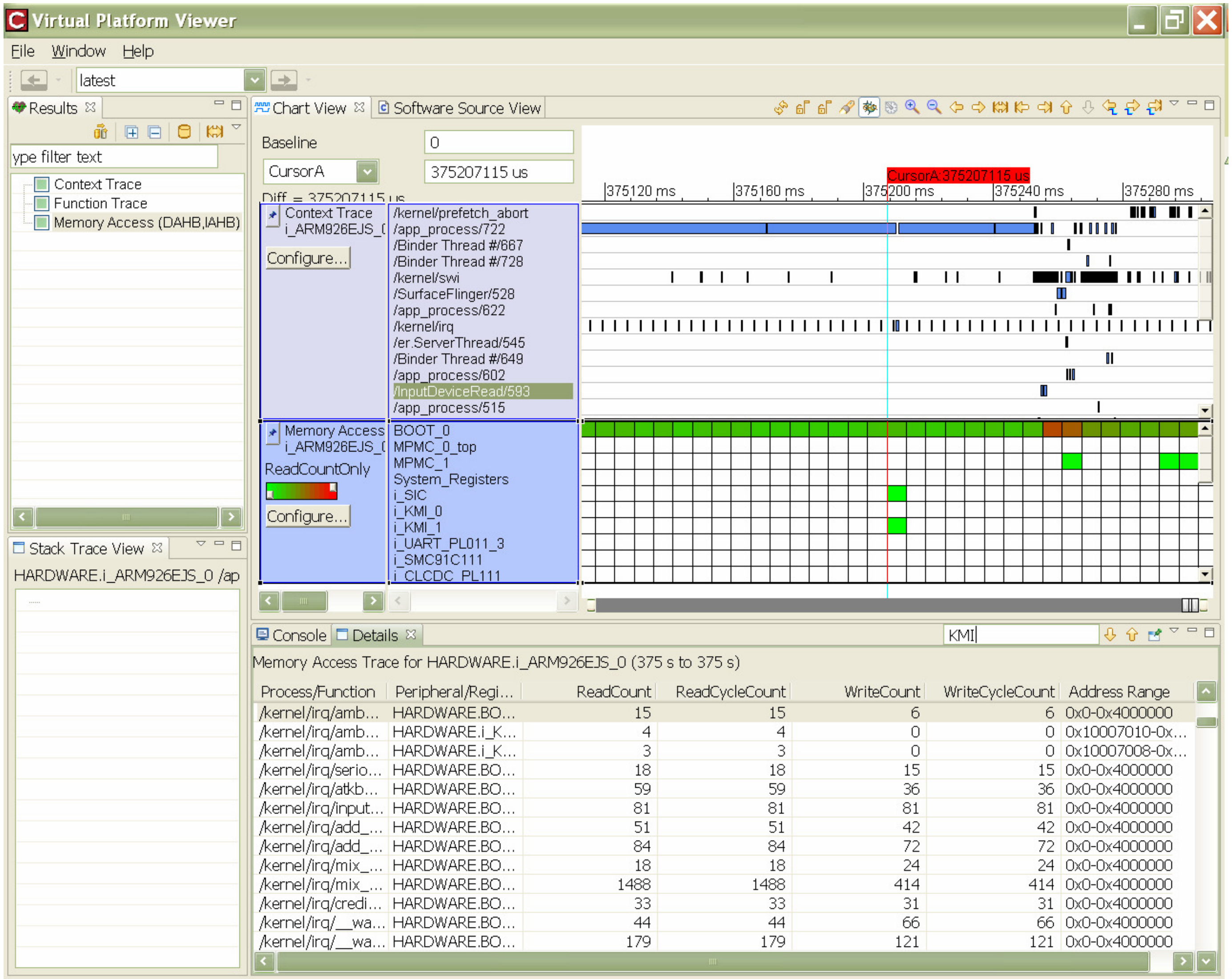
- **Debugging: Analyze a snaphot of the system state**
- **Challenge: Understand/analyze system history**
  - **Interaction between HW and SW entities over time**
- **Requierement: System level tracing of HW/SW**

Platform Level Software Analysis



CoWare®

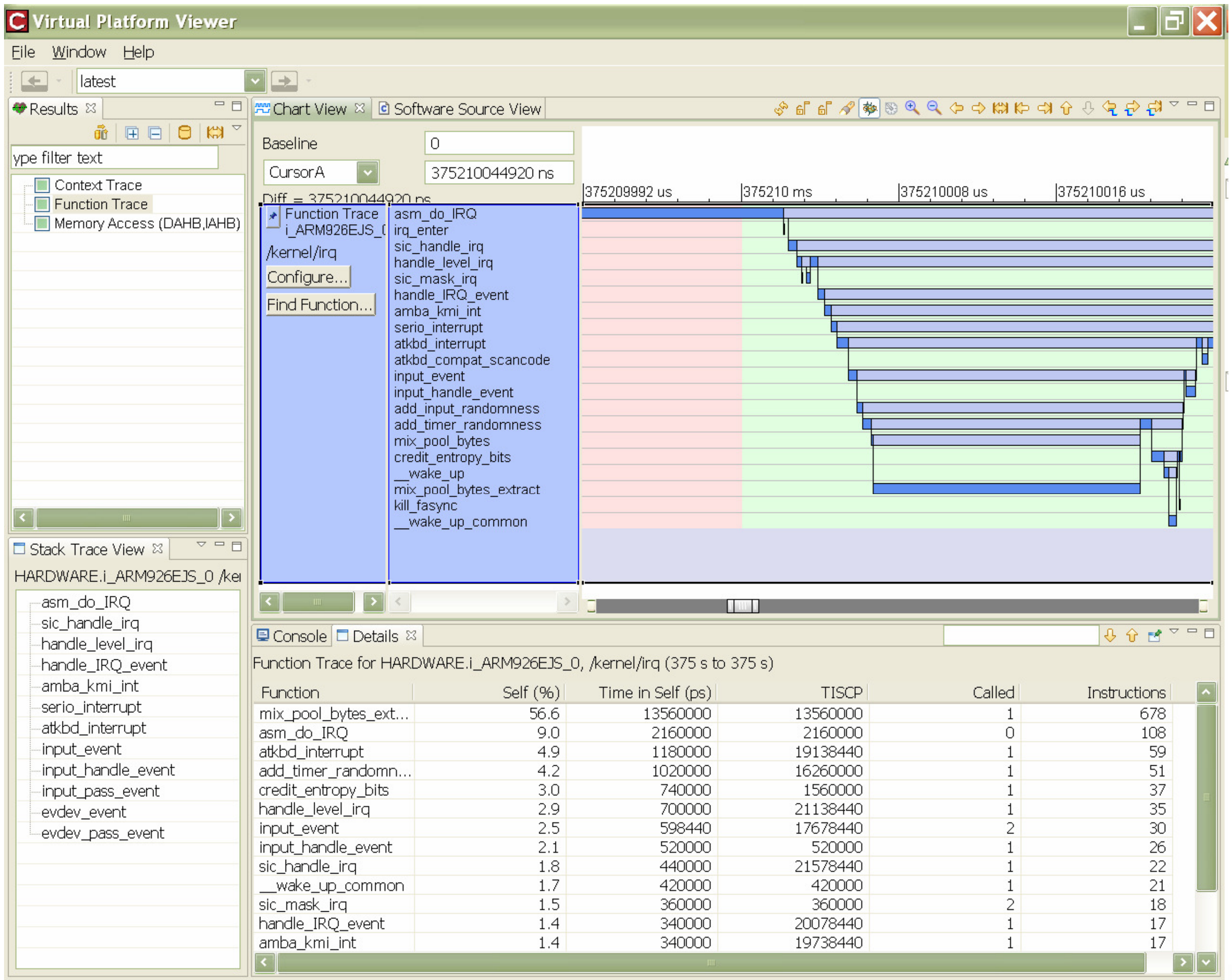
Platform Level Software Analysis



Process/Function	Peripheral/Regi...	ReadCount	ReadCycleCount	WriteCount	WriteCycleCount	Address Range
/kernel/irq/amb...	HARDWARE.BO...	15	15	6	6	0x0-0x4000000
/kernel/irq/amb...	HARDWARE.i_K...	4	4	0	0	0x10007010-0x...
/kernel/irq/amb...	HARDWARE.i_K...	3	3	0	0	0x10007008-0x...
/kernel/irq/serio...	HARDWARE.BO...	18	18	15	15	0x0-0x4000000
/kernel/irq/atkb...	HARDWARE.BO...	59	59	36	36	0x0-0x4000000
/kernel/irq/input...	HARDWARE.BO...	81	81	81	81	0x0-0x4000000
/kernel/irq/add_...	HARDWARE.BO...	51	51	42	42	0x0-0x4000000
/kernel/irq/add_...	HARDWARE.BO...	84	84	72	72	0x0-0x4000000
/kernel/irq/mix_...	HARDWARE.BO...	18	18	24	24	0x0-0x4000000
/kernel/irq/mix_...	HARDWARE.BO...	1488	1488	414	414	0x0-0x4000000
/kernel/irq/credi...	HARDWARE.BO...	33	33	31	31	0x0-0x4000000
/kernel/irq/__wa...	HARDWARE.BO...	44	44	66	66	0x0-0x4000000
/kernel/irq/__wa...	HARDWARE.BO...	179	179	121	121	0x0-0x4000000

CoWare®

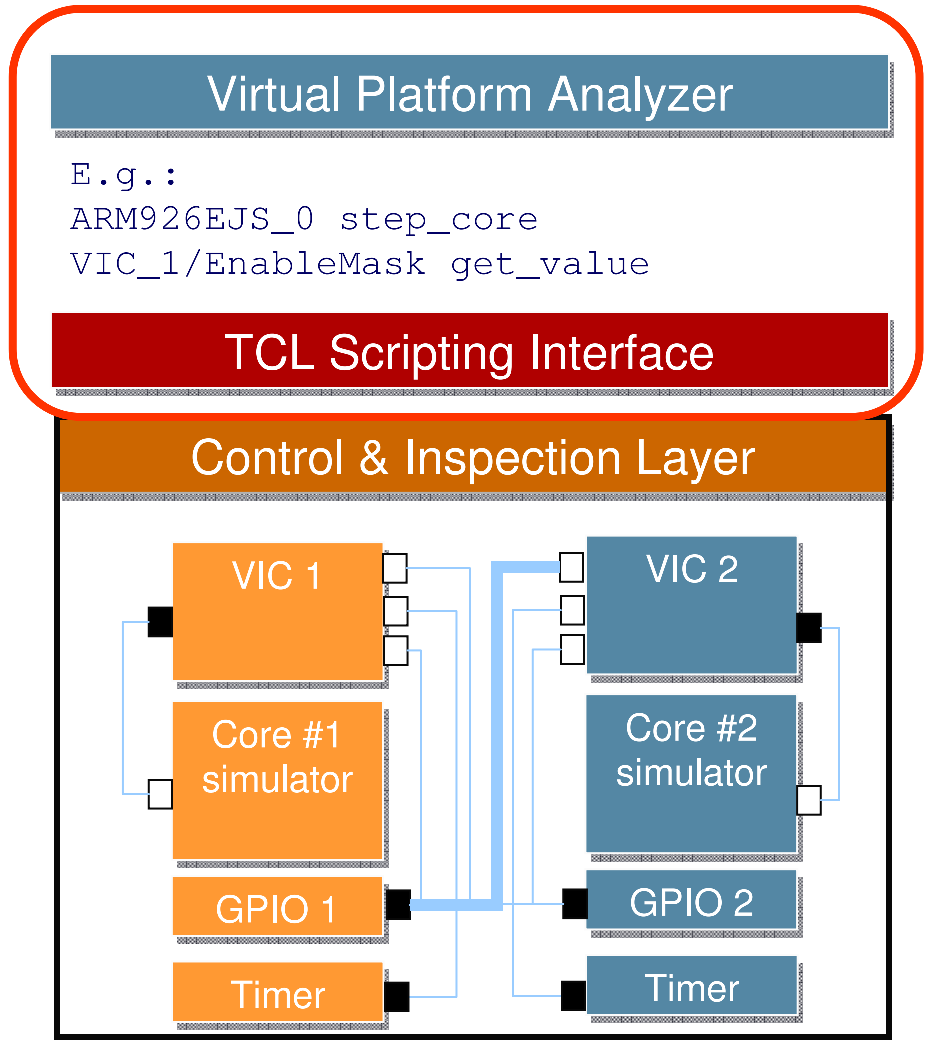
Platform Level Software Analysis



Virtual Platform Scripting

Scripting Use Cases:

- Determinstic repetition of scenarios
- Regressionizing
- Analyzing
- **Debugging**

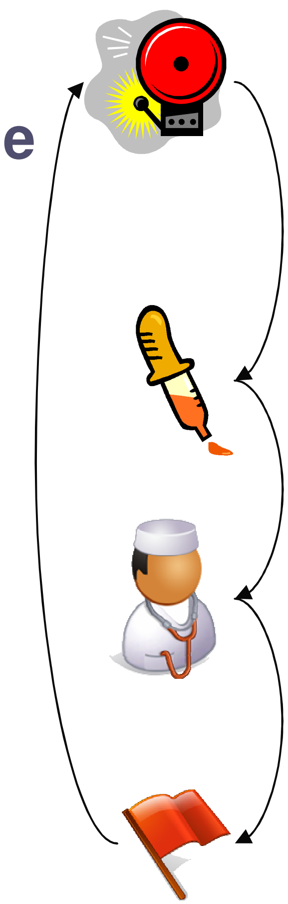


CoWare®

Scripting For Debug

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

```
terminal
Connected to session.
Uncompressing Linux..........................................................
.............. done, booting the kernel.
```

- 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!

2) VP Debug Helper Script (TCL)

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

# 2: Attach callback prodecure
$bp  set_callback tcl_emit_log_char

# 3: Callback procdure: 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]"
}
```

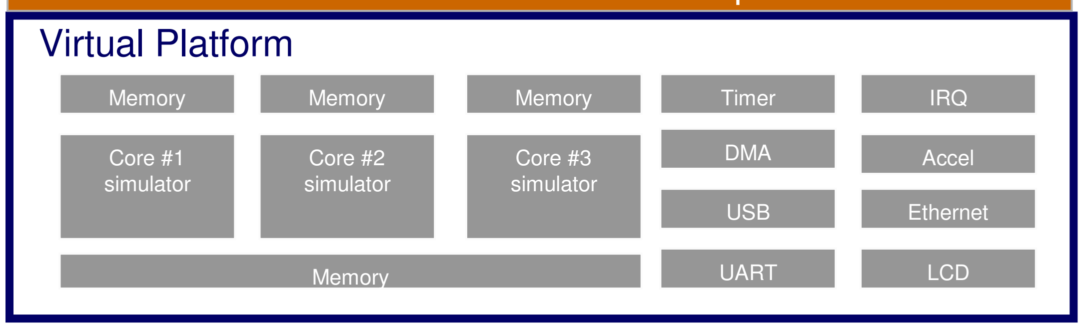
GO!

3) Kernel Debug Messages via VP

Terminal Trace for HARDWARE.i_ARM926EJS_0

Time (ps)	Text
2149216220000	<4>CPU0: D VIVT write-back cache
2149340860000	<4>CPU0: I cache: 4096 bytes, associativity 4, 32 byte lines, 32 sets
2149470620000	<4>CPU0: D cache: 4096 bytes, associativity 4, 32 byte lines, 32 sets
2149683180000	<4>Built 1 zonelists in Zone order. Total pages: 16256
2149847440000	<5>Kernel command line: root=/dev/mtdblock0 rw slram=test,0x8000000,
2158738360000	<1>Unhandled fault: alignment exception (0x801) at 0x0000000b
2158816720000	<4>Internal error: : 801 [#1] PREEMPT
2158871000000	<4>Modules linked in:
2158944740000	<4>CPU: 0 Not tainted (2.6.23 #161)
2159037200000	<4>PC is at vic_init+0x18/0xe0
2159146100000	<4>LR is at versatile_init_irq+0x1c/0xb8
2159283160000	<4>pc : [<c000ded8>] lr : [<c000e030>] psr:
2159326860000	<4>sp : c0231f80 ip : c0231fa8 fp : c0231fa4
2159425180000	<4>r10: 0001987c r9 : 41069263 r8 : 000198
2159541500000	<4>r7 : c0233e38 r6 : c001adf8 r5 : c0245ee0
2159653000000	<4>r3 : 00000000 r2 : ffffffff r1 : 00000000 r0 : ff

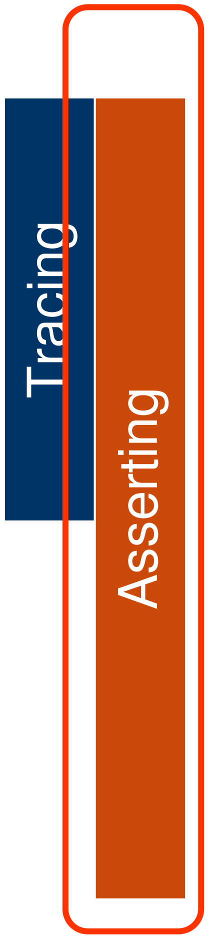
Virtual Platform Simulation Control & Inspection Interface



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

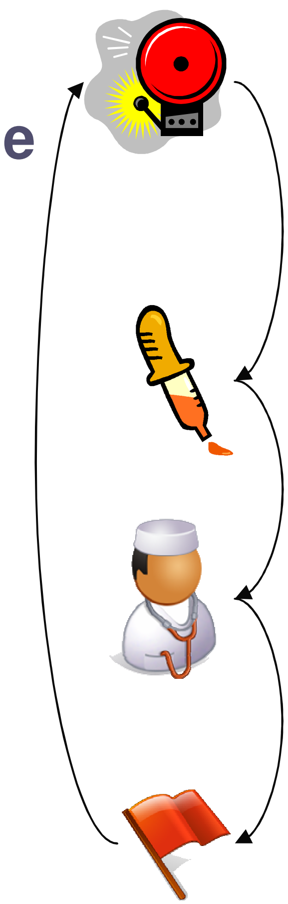
CoWare®

Scripting For Debug



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

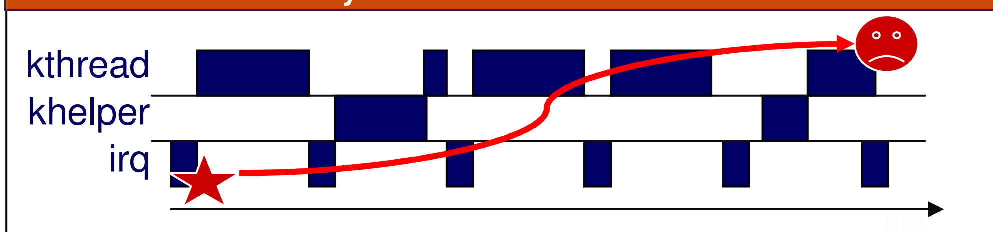
"My kernel shows sporadic kernel panic problems. How can I assert a memory corruption in the kernel?"

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	Delayed crash of kernel activities (scheduler, threads...)

History of a buffer overrun defect

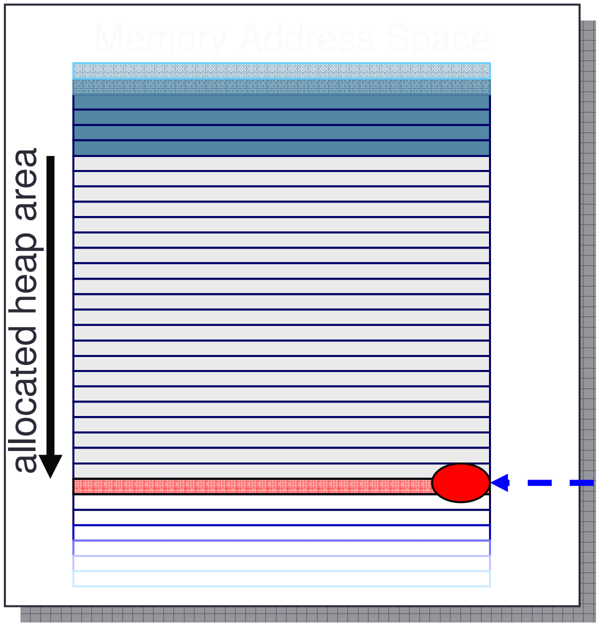


Typical code prone to a buffer overrun

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

CoWare®

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*

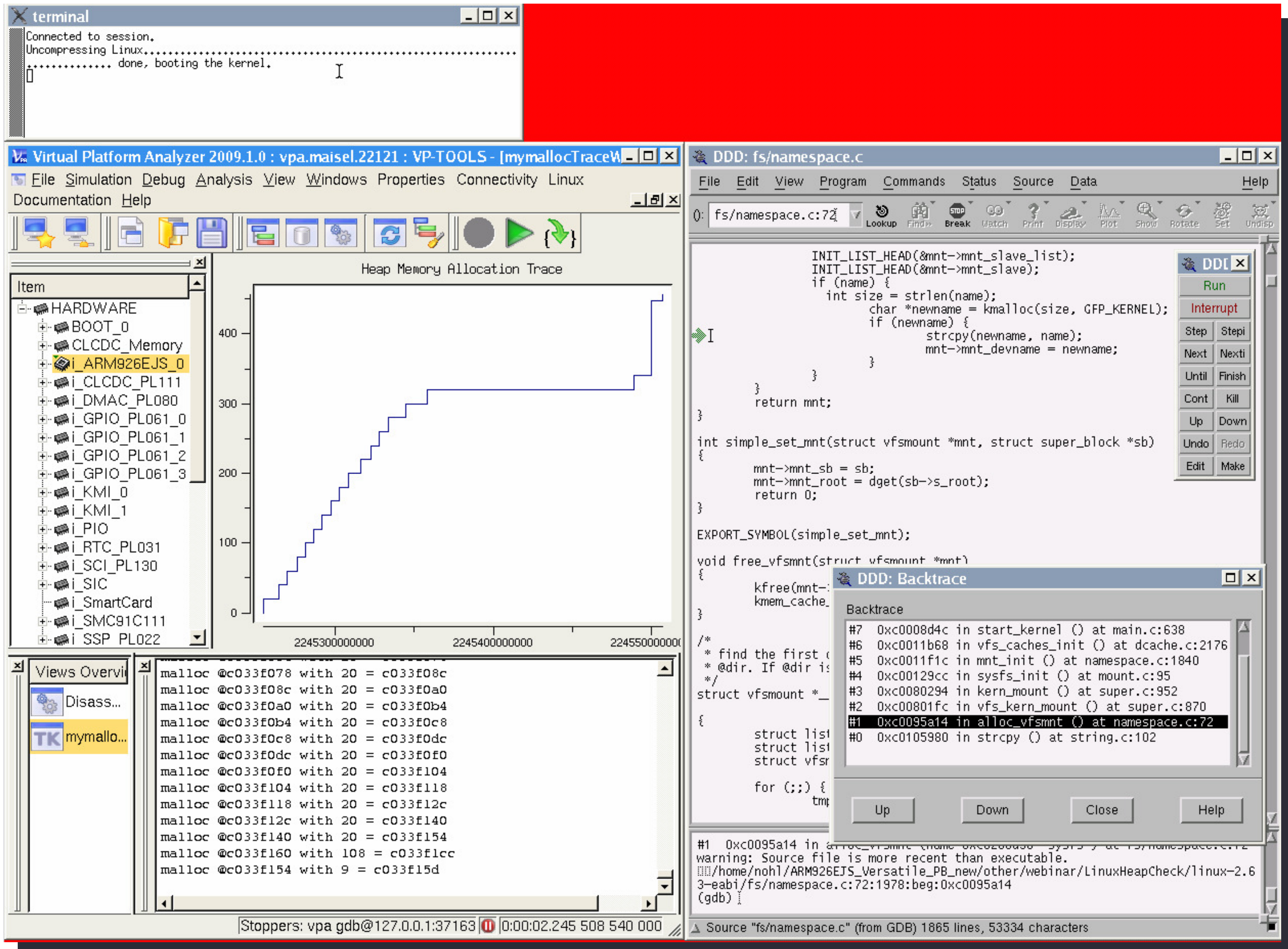
set watchpoint

```
proc callbackmallocExit {} {
    # 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_ARM926EJS_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_sw_watchpoint $next  write virtual
        }
    } else {
        create_sw_watchpoint $next write virtual
    }

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

    # Trace the allocation
    callbackmallocTrace $malloc_size
    return ""
}
```

CoWare®

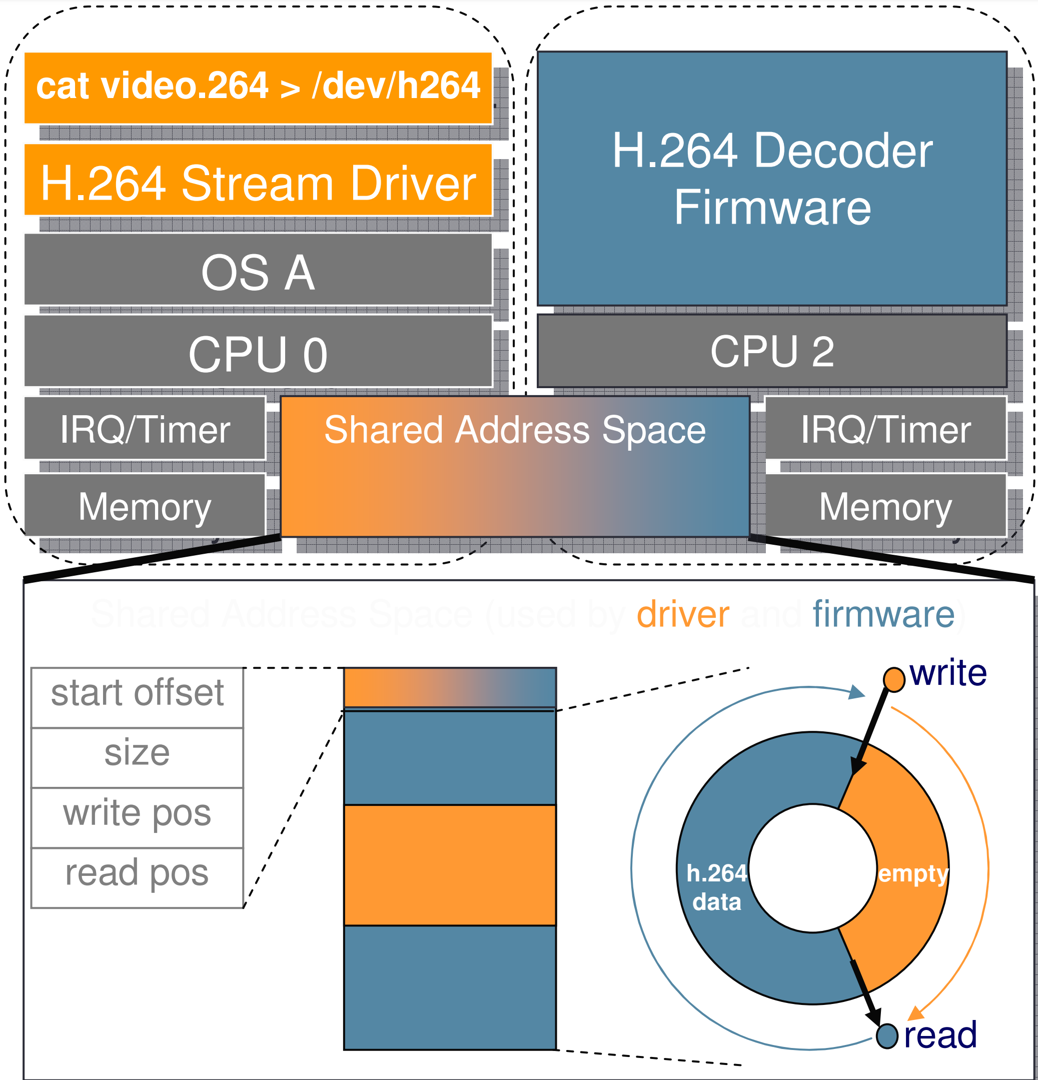
Kernel Memory Corruption

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

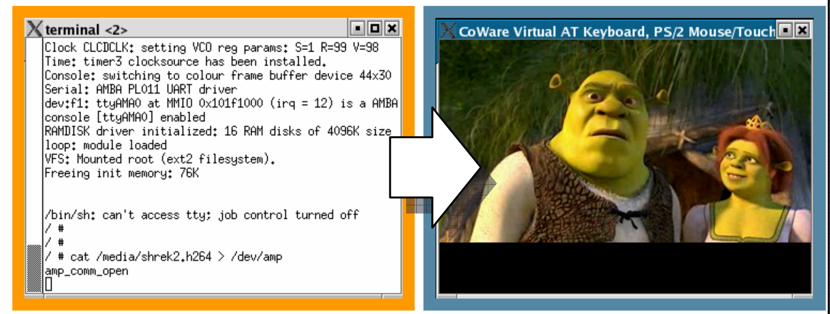


CoWare®

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

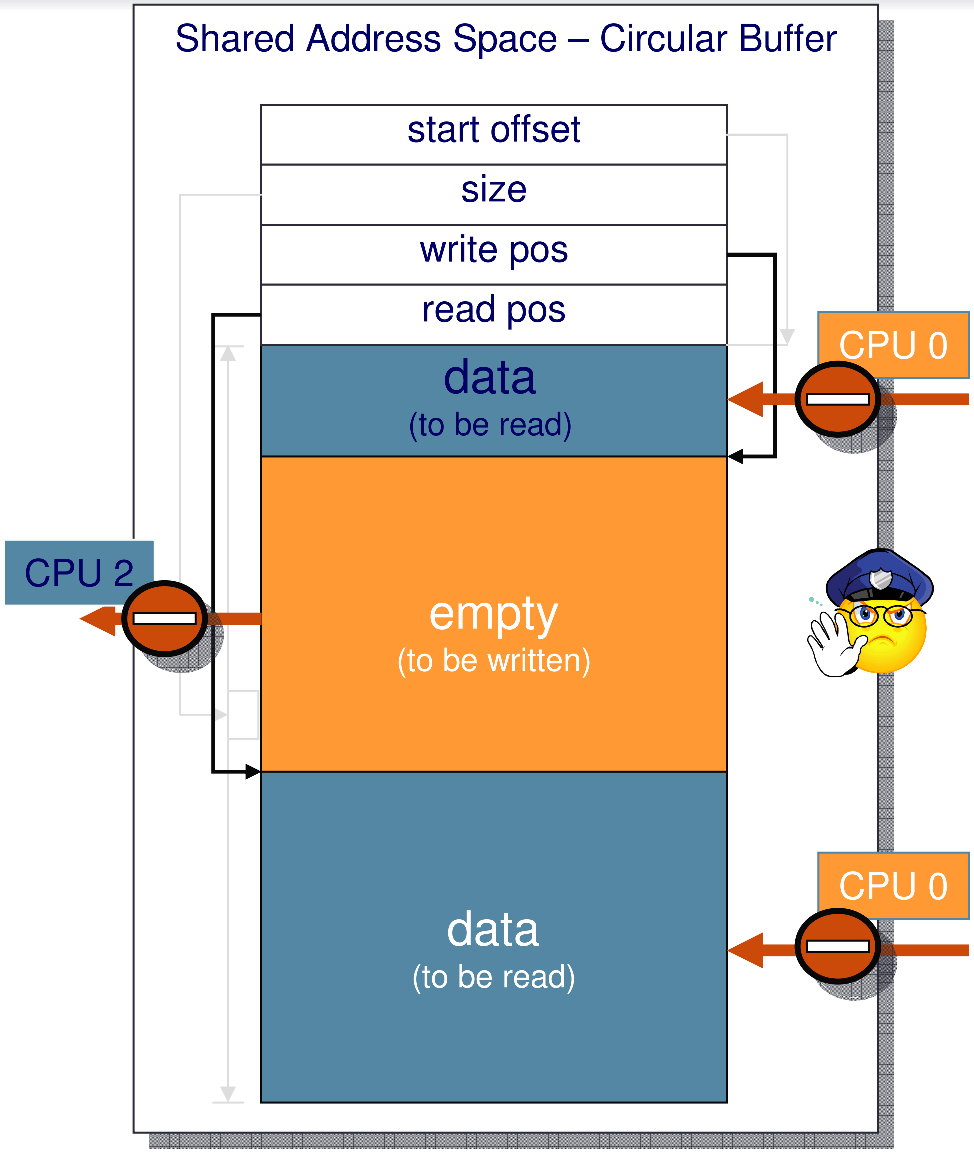
- Interrupts
- Mutex/Semaphores

Data streaming

- Circular buffer in shared memory

CoWare®

Multi-core Driver Example



Implementation issues

- Buffer overrun
- Race condition/data corruption
- Starvation

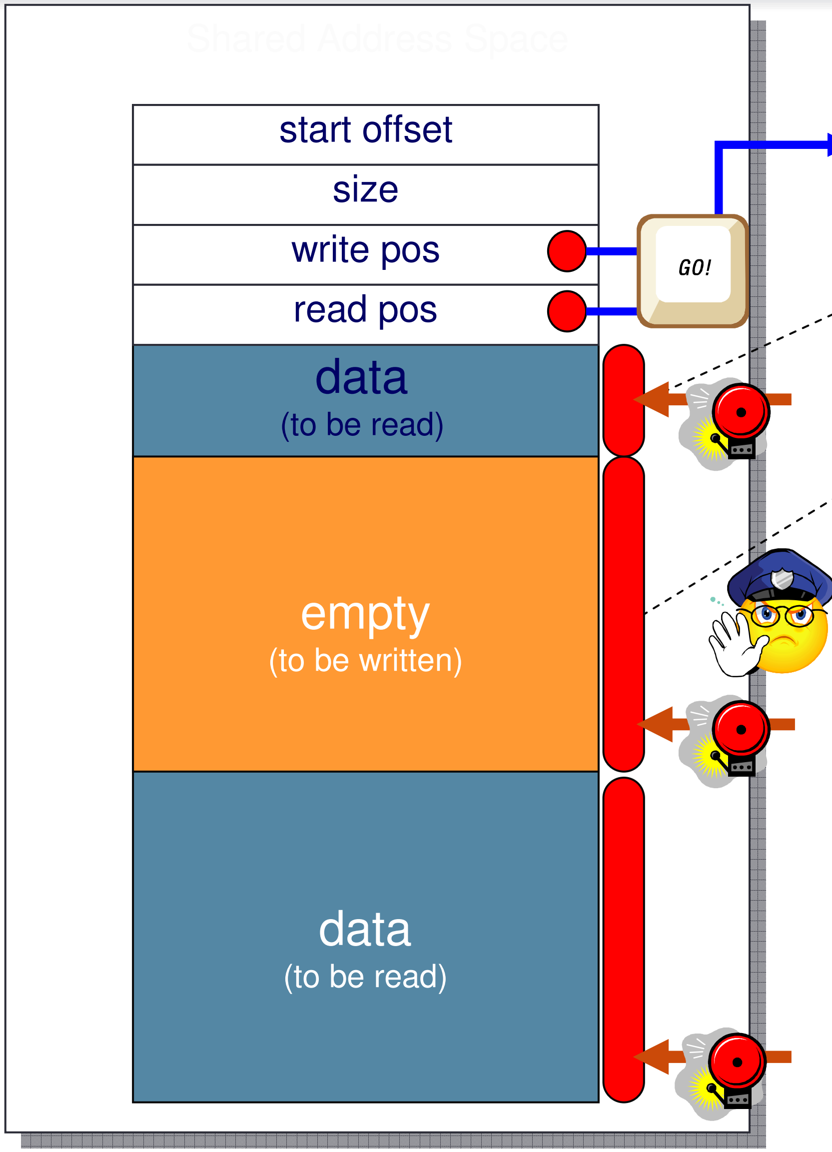
Debugging challenges

- 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
```

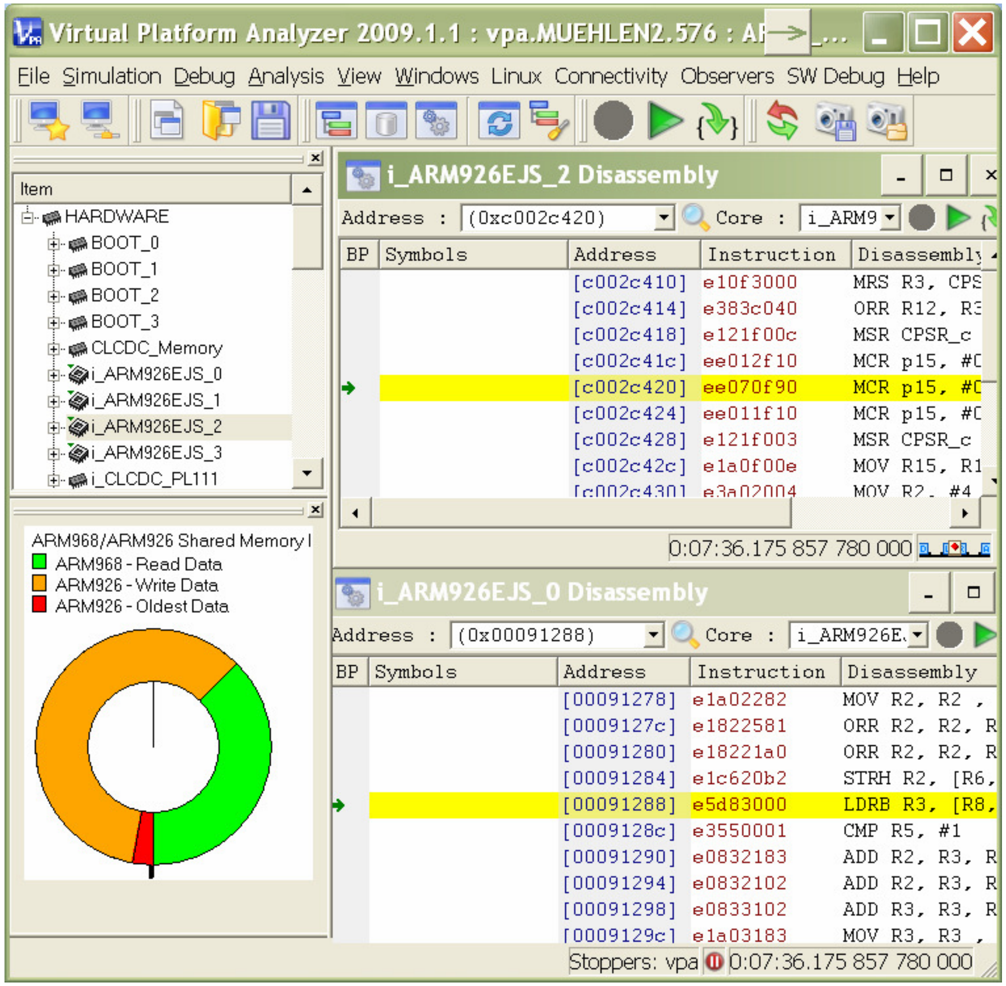
System level software assertion

- Watch-points on circular buffer pointers
- trigger updates of the watch-regions.

- Platform execution is suspended,
- as soon as memory is accessed illegally.

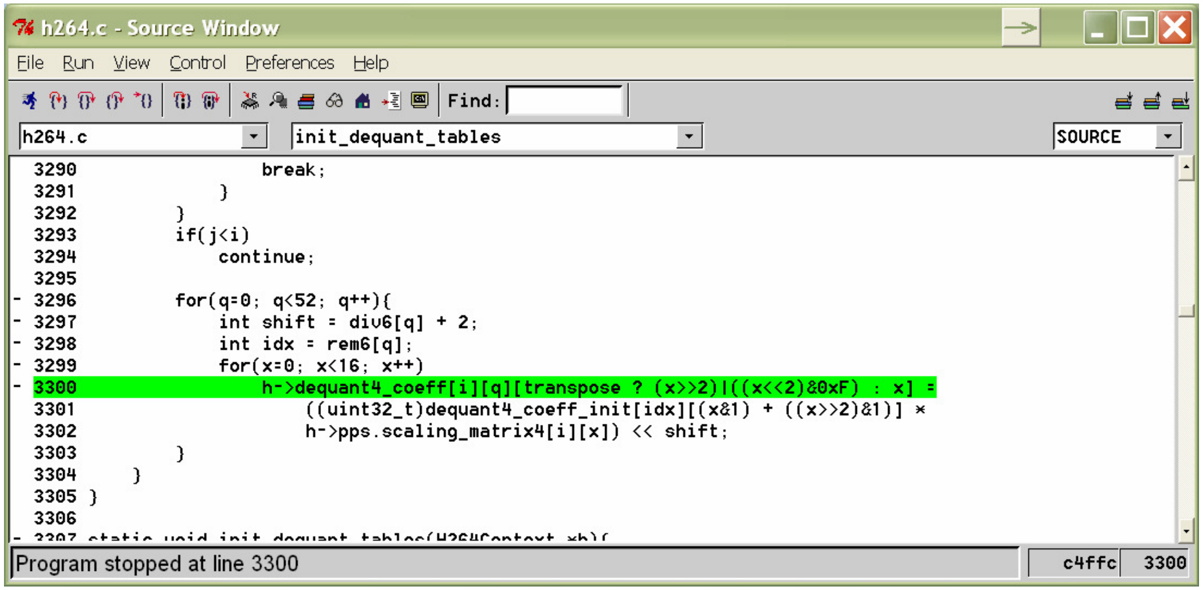
CoWare®

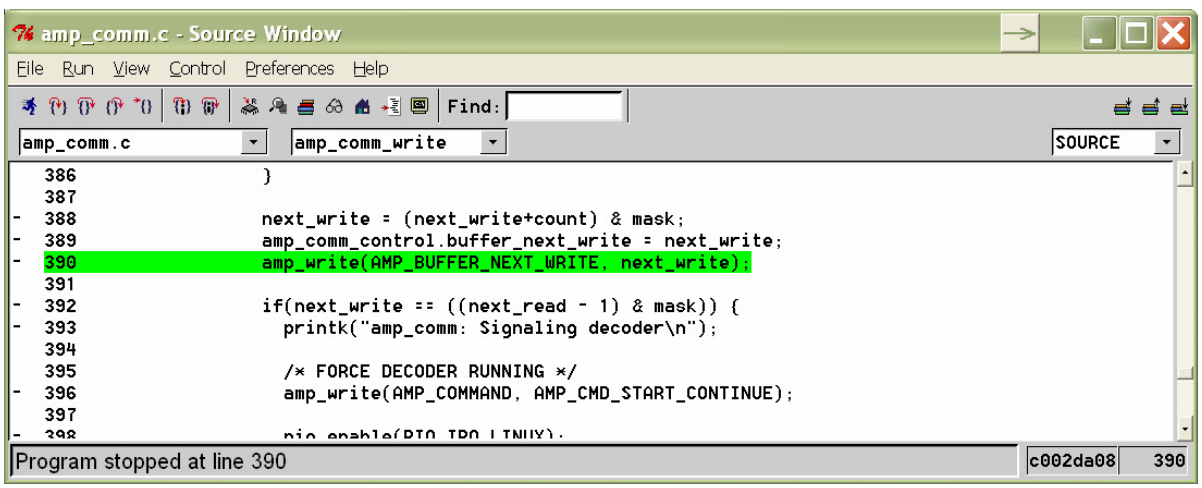
Multi-core Driver Example



CoWare®

Multi-core Driver Example





CoWare®

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.**

Thank You!