Rudi Dienstbeck ▪ June 07, 2016

ARM® TrustZone® and Hypervisor Debugging

ARM® CORTEX®-A/-R EXPERT DAY

Meet the Debug Experts

LAUTERBACH
DEVELOPMENT TOOLS

Agenda

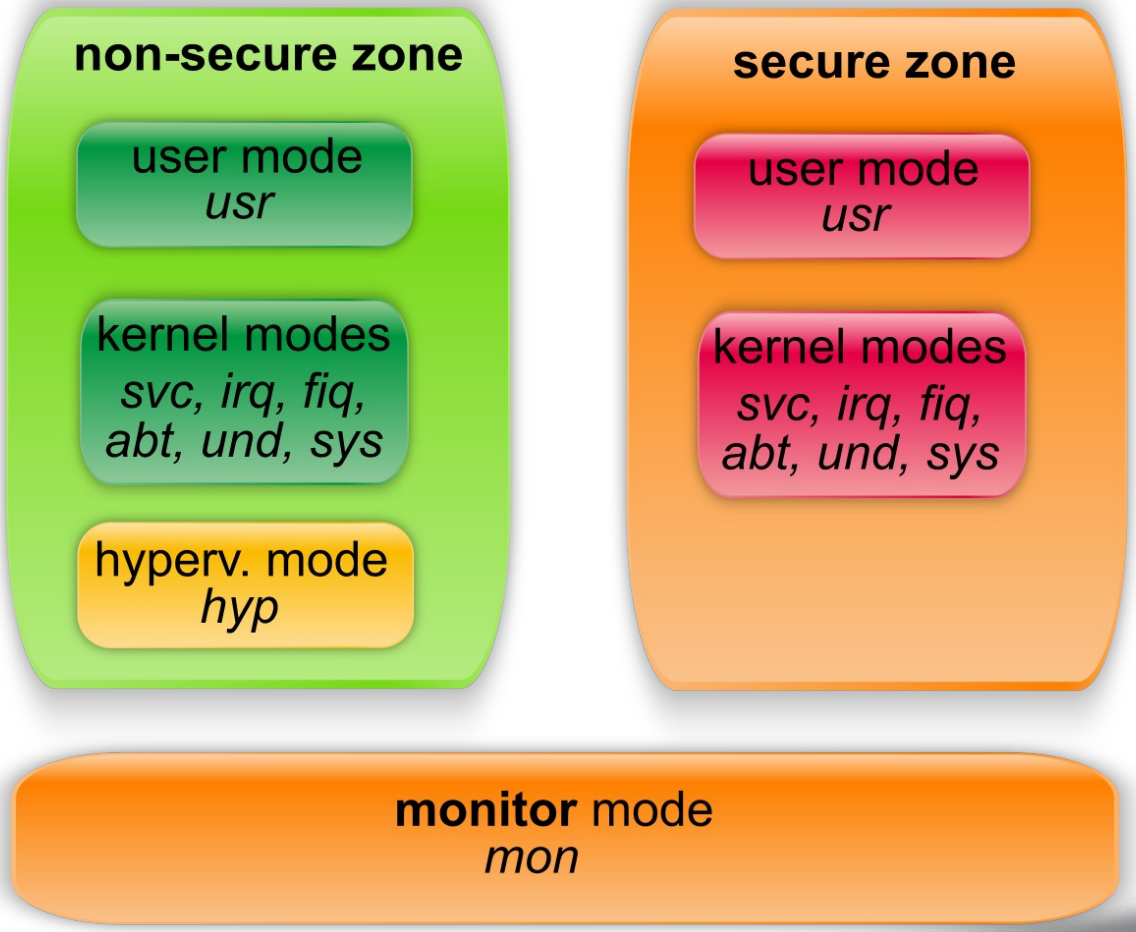
- TrustZone And CPU Modes In TRACE32
- Default Behavior Of The Debugger
- Special TrustZone Support
- Debugging Through The Zones
- Outlook To Multiple Guests

Agenda

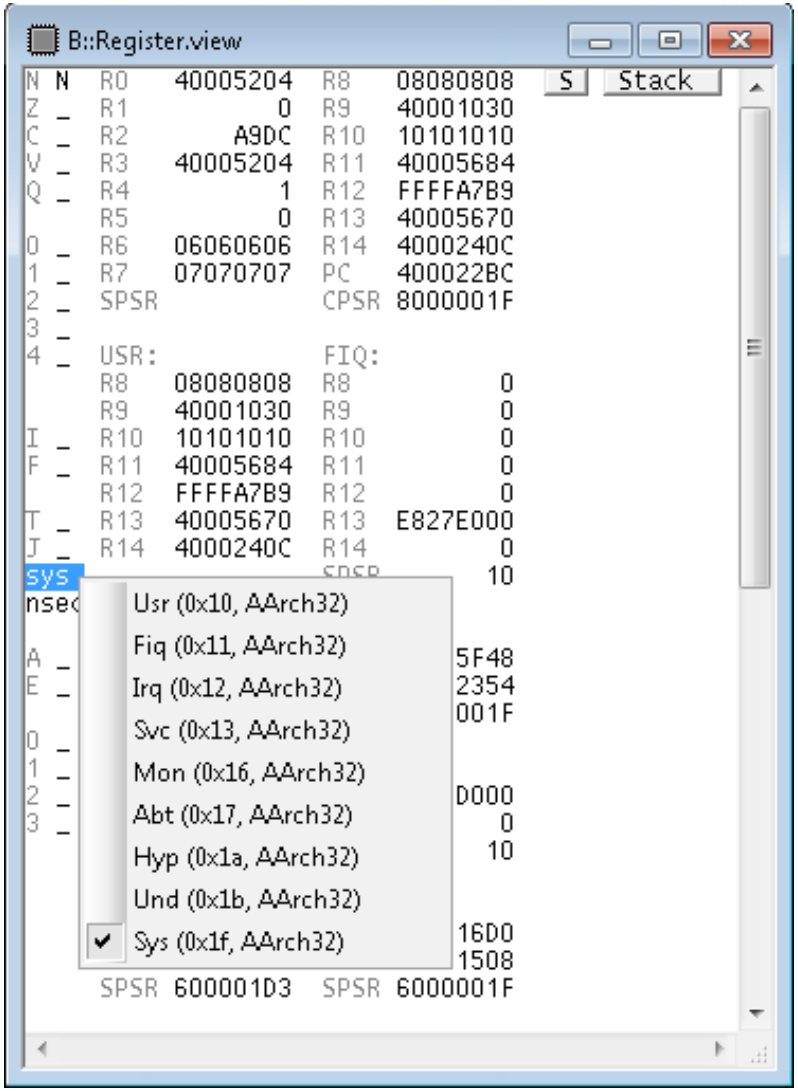
- ▶ **TrustZone And CPU Modes In TRACE32**
- Default Behavior Of The Debugger
- Special TrustZone Support
- Debugging Through The Zones
- Outlook To Multiple Guests

TrustZone And CPU Modes In TRACE32

CPU Modes - Core View



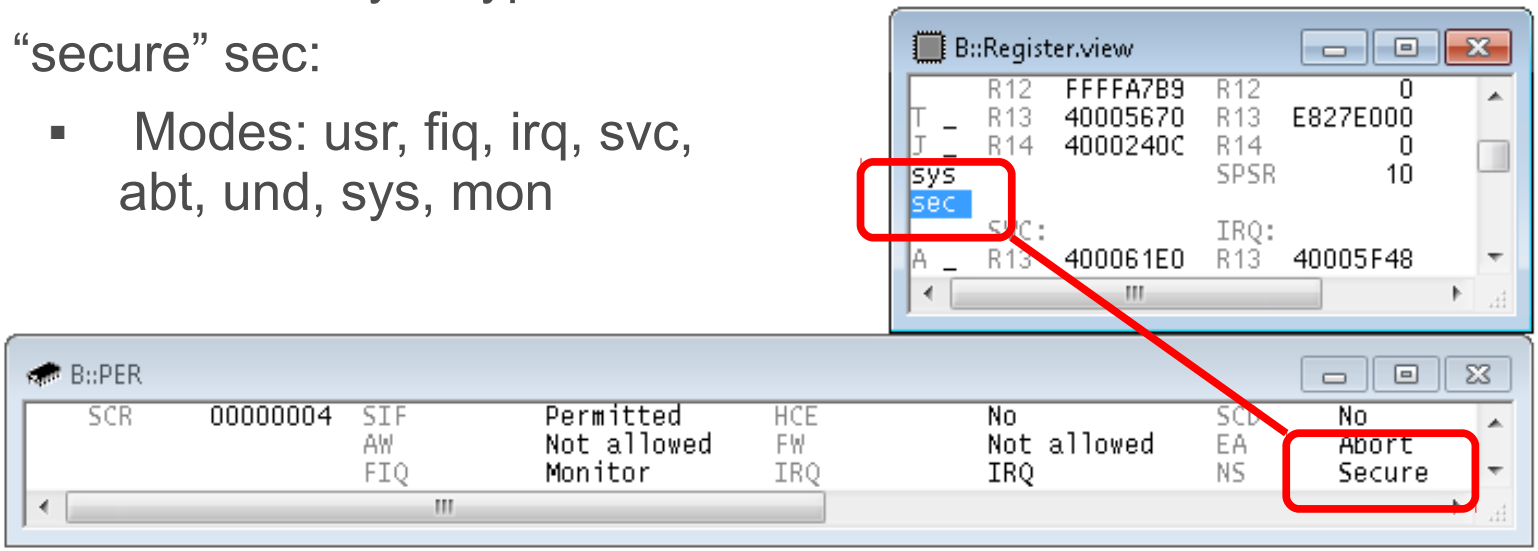
TrustZone And CPU Modes In TRACE32



- CPU modes in Register window
  - Register modes in CPSR bits 0..4
    - “user mode”:
      - usr
    - “kernel modes”:
      - fiq, irq, svc, abt, und, sys
    - “hypervisor mode”:
      - hyp (only non-secure)
    - “monitor mode”:
      - mon (only secure)

TrustZone And CPU Modes In TRACE32

- CPU modes in Register window
  - TrustZone in SCR bit 0
    - “non-secure” nsec:
      - Modes: usr, fiq, irq, svc, abt, und, sys, hyp
    - “secure” sec:
      - Modes: usr, fiq, irq, svc, abt, und, sys, mon



TrustZone And CPU Modes In TRACE32

CPU Modes - Memory Access View

non-secure zone

modes: **usr, svc, irq, fiq, abt, und, sys**

access class: **N:**

MMU translation: **NonSecPageTable IntermedPageTable**

hypervisor zone

mode: **hyp**

access class: **H:**

MMU translation: **HypPageTable**

secure zone

modes: **usr, svc, irq, fiq, abt, und, sys**

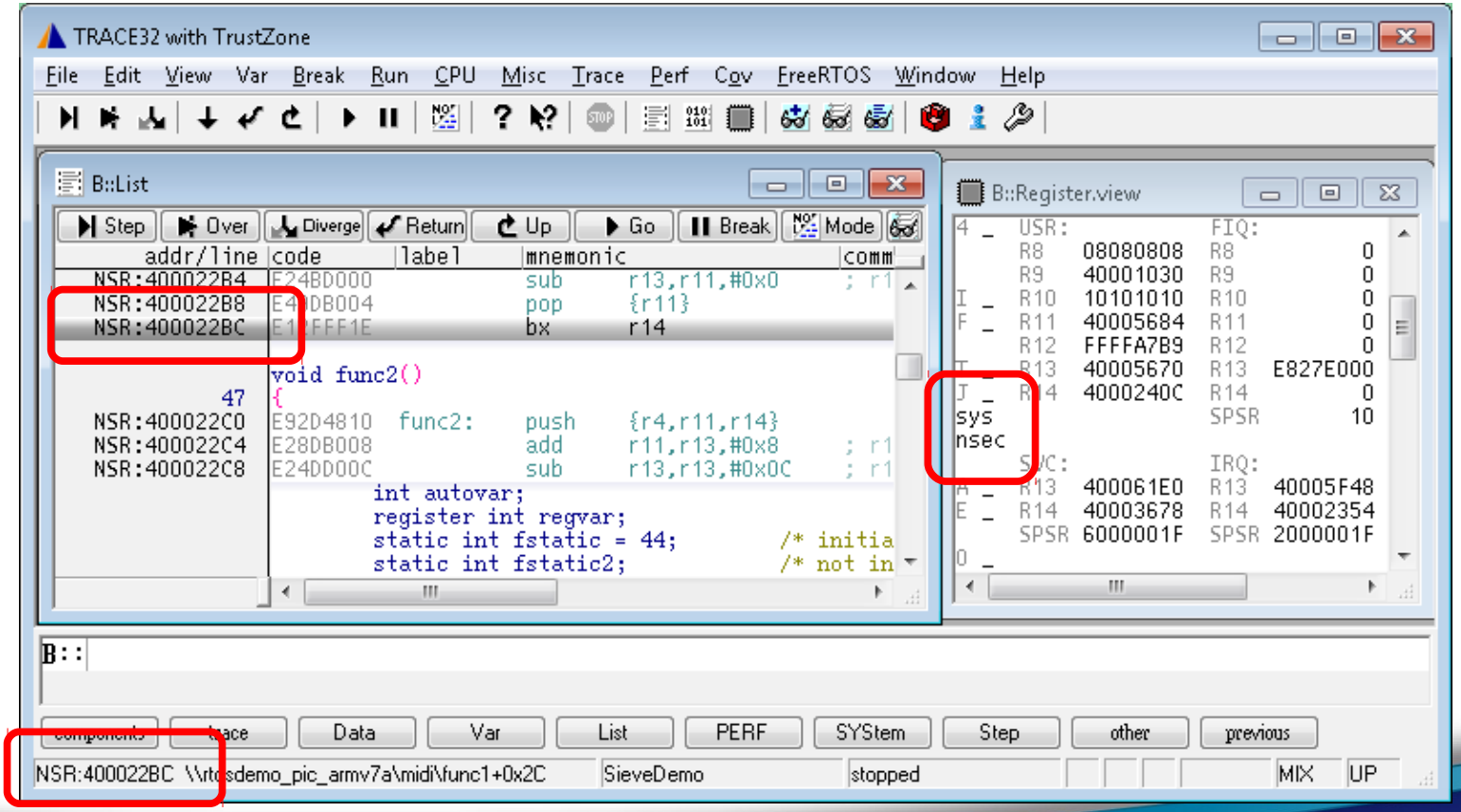
access class: **Z:**

MMU translation: **SecPageTable**

mode: **mon**

TrustZone And CPU Modes In TRACE32

- Memory addresses with “access class”
  - Access class relates to memory access depending on CPU mode and zone
  - PC shown with current access class in “**`List.auto`**” and status line



TrustZone And CPU Modes In TRACE32

- Memory addresses with “access class”
  - If not explicitly specified, windows follow current CPU mode
  - Overriding is possible: see examples below

B::Data.dump 0x40000000

address	0	4	01234567
NSD:40000000	E1A00000	E1A00000	NNAE NNAE UU01 UU01
NSD:40000008	E1A00000	EAFFFFFB	NNAE FFFE UU01 BFFA
NSD:40000010	87BE51E0	7B567081	EQBS SpV{ 0E71
NSD:40000018	9791B3A9	991D6FAC	AB99A 9317C 019 D3

B::Data.dump ZS:0x40000000

address	0	4	01234567
ZSD:40000000	E1A00000	E1A00000	NNAE NNAE UU01 UU01
ZSD:40000008	E1A00000	EAFFFFFB	NNAE FFFE UU01 BFFA
ZSD:40000010	95ADA736	559F08B7	GAA9 BB9U 7D57 SF
ZSD:40000018	B7F55AED	74DD3B3B	EZFB ;;Dt DZ57 DD

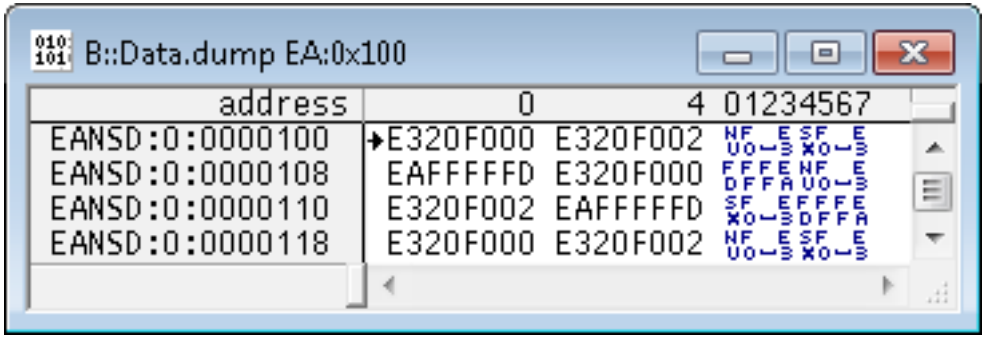
B::List U:sieve

Step | Over | Up | Go

addr/line	code	label	mnemonic	
	int sieve()			
186	{			
NUR:40002898	E92D0870	sieve:	push	{r4-r6,
NUR:4000289C	E28DB00C		add	r11,r13
NUR:400028A0	E24DD008		sub	r13,r13
NUR:400028A4	E59F30A4		ldr	r3,0x40

TrustZone And CPU Modes In TRACE32

- Mapping of access classes for memory accesses
  - D: Data, *P: Program*, R: aRm code, T: Thumb code
  - S: kernel modes (“Supervisor”), U: User mode
  - N: Non-secure, Z: secure+monitor (“Zone”), H: Hypervisor
    - These access classes map MMU tables (see next slides)
  - A: physical (“Absolute”) access (see next slides)
  - E: run-time access (“Emulation”, aka dual-port, see next slides)

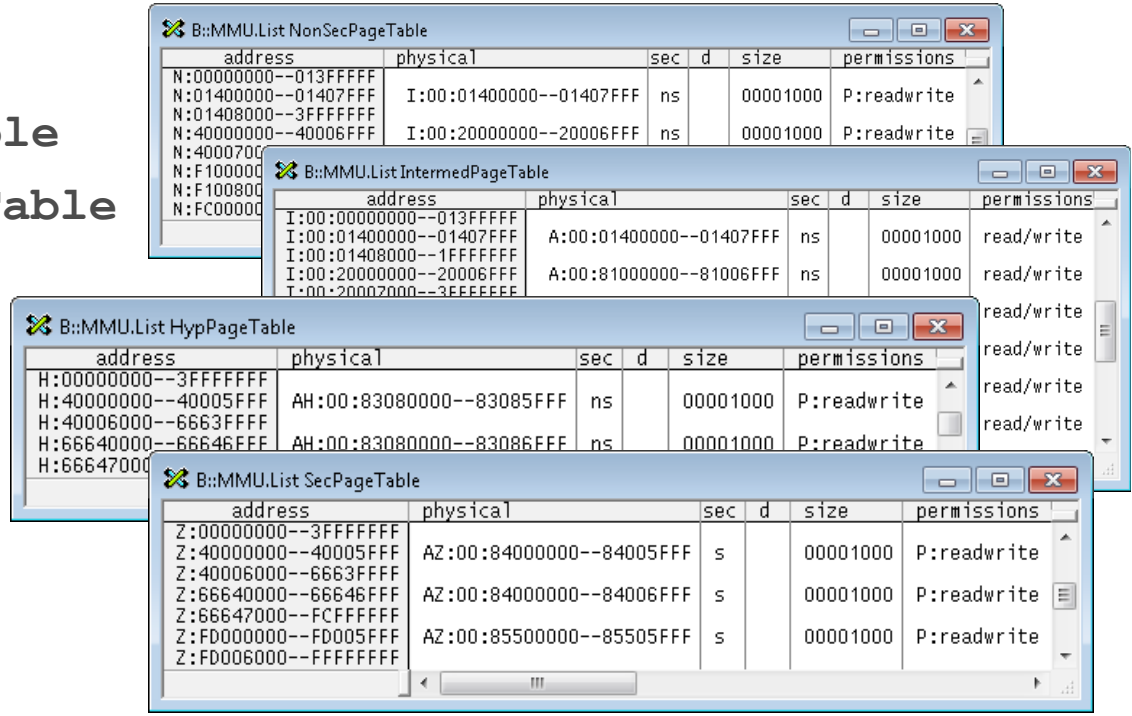


TrustZone And CPU Modes In TRACE32

- Mapping of access classes for memory accesses
  - Combinations: e.g.
    - ZUT:
      - Secure zone, user mode, thumb code
    - ANSD:
      - Non-secure zone, supervisor mode, data, physical address
    - EAHR:
      - Dual-port access to physical hypervisor arm code – BUT:
      - Note: “H” not visible at bus with “E”,
      - Note: MMU/caching issues with “E”
  - C: “CPU” access = current CPU mode access
  - Several others, not mentioned here (e.g. coprocessor access)
    - See debugger_arm.pdf, chapter “Access Classes”

TrustZone And CPU Modes In TRACE32

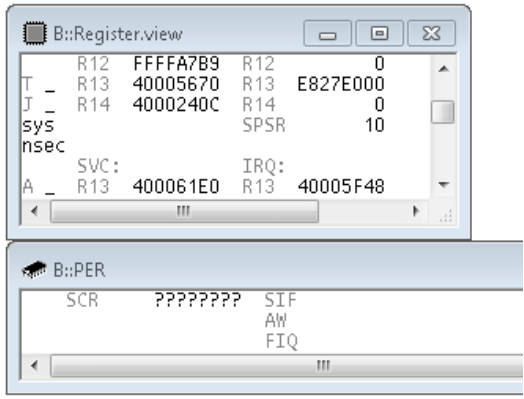
- MMU mapping of zones and access classes
  - Non-secure zone (N:)
    - **`MMU.List NonSecPageTable`**
    - **`MMU.List IntermedPageTable`**
  - Hypervisor (H:)
    - **`MMU.List HypPageTable`**
  - Secure zone (Z:)
    - **`MMU.List SecPageTable`**
  - "Current" CPU mode
    - **`MMU.List PageTable`**

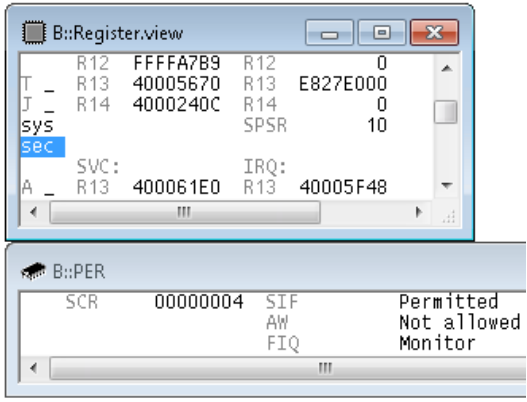


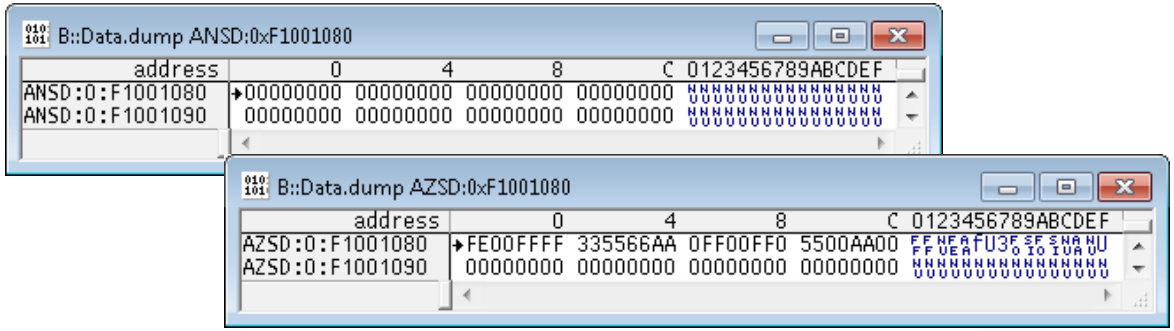
- *Note: `MMU.List` without parameters shows fixed, manual entries!*

TrustZone And CPU Modes In TRACE32

- "Physical" addresses / access class A:
  - Physical is unambiguous? No it isn't!
    - AZSD: is different from ANUP!
  - S/U state visible on AMBA bus → distinction possible!
  - Zone also distinguishable on physical bus:
    - AN: might not "see" physical secure memory
    - AZ: might see different memory/peripheral than AN:
    - While in non-secure mode, override with AZ:
      - If CPU lets us do it (secure debug enabled)

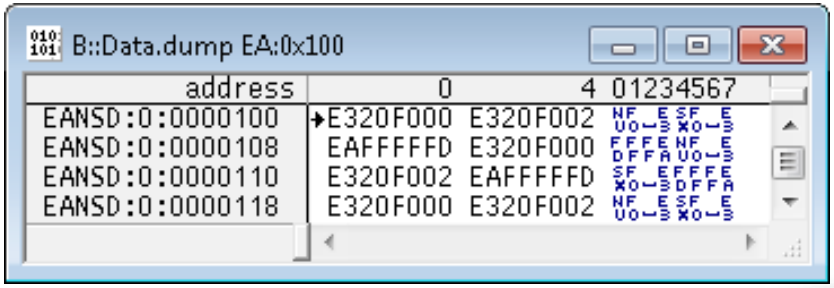






TrustZone And CPU Modes In TRACE32

- Run-Time access class E:
  - E.g. **`SYStem.MemAccess DAP`**
    - DAP-Access to ARM internal bus (APB/AHB/AXI)
    - Caution: Cache invisible!
      - With write-back cache, you'll see old/invalid data!
    - MMU Translation with debugger may not be possible!
  - E.g. **`SYStem.CpuAccess Enable`**
    - Debugger stops CPU shortly to read data
    - Caution: heavy run-time impact!



TrustZone And CPU Modes In TRACE32

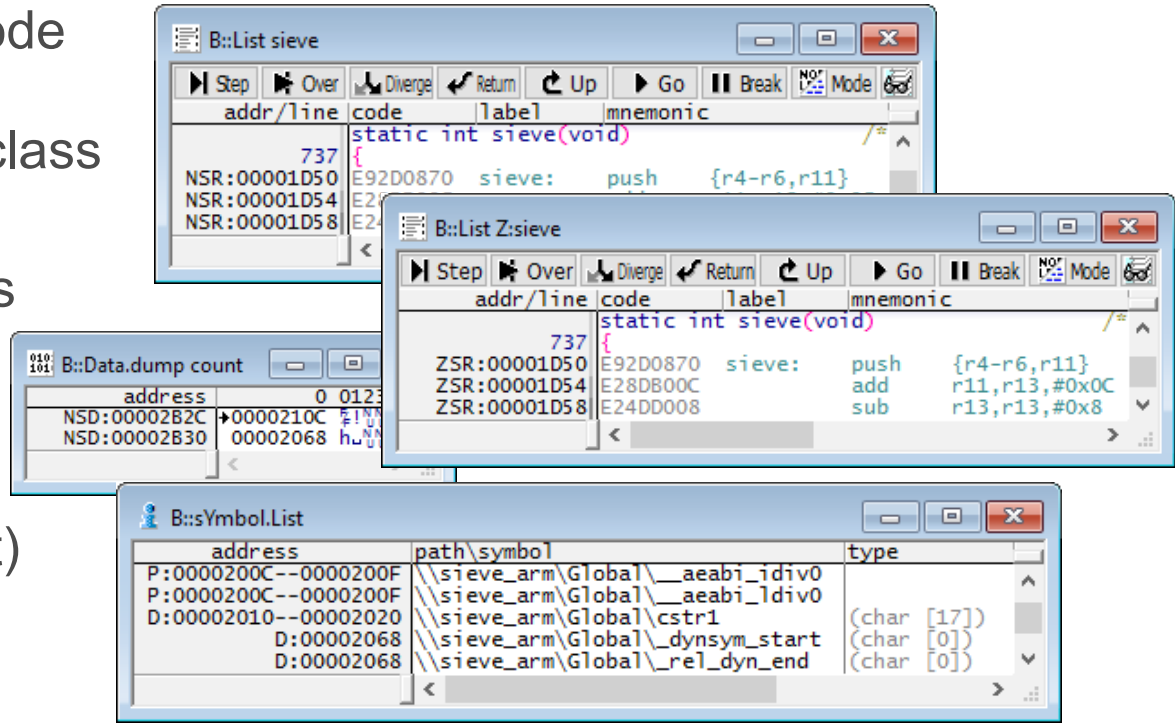
- Memory visibility with zones
  - Depending on CPU hardware implementation,
    secure zone may be inaccessible!
    (Secure debug disabled)
  - Depending on CPU mode,
    some memory may be invisible in non-secure zone!
    See slide about physical access (AN:/AZ: difference)

Agenda

- TrustZone And CPU Modes In TRACE32
- ▶ Default Behavior Of The Debugger
- Special TrustZone Support
- Debugging Through The Zones
- Outlook To Multiple Guests

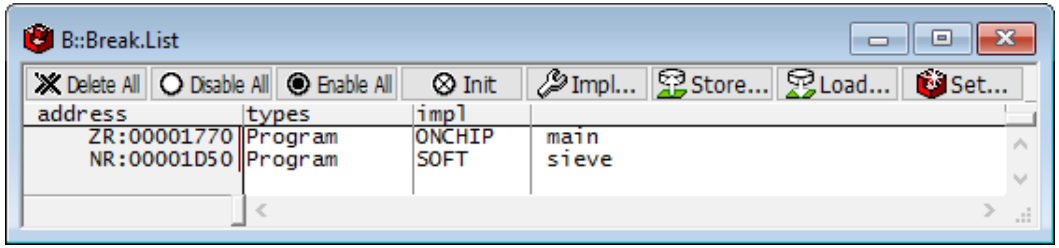
Default Behavior Of The Debugger

- Symbols are independent of the CPU mode and TrustZone
  - List window always shows code matching to symbol address, regardless of zone / access class
  - Debugger accesses variables with the current CPU mode
  - Symbols are only divided in P: and D: (see sYmbol.List)
  - Access class override is possible!



Default Behavior Of The Debugger

- Breakpoints are independent of the CPU mode and TrustZone
  - Software breakpoints are written to the code with current CPU mode
  - Onchip breakpoints react regardless of current CPU mode



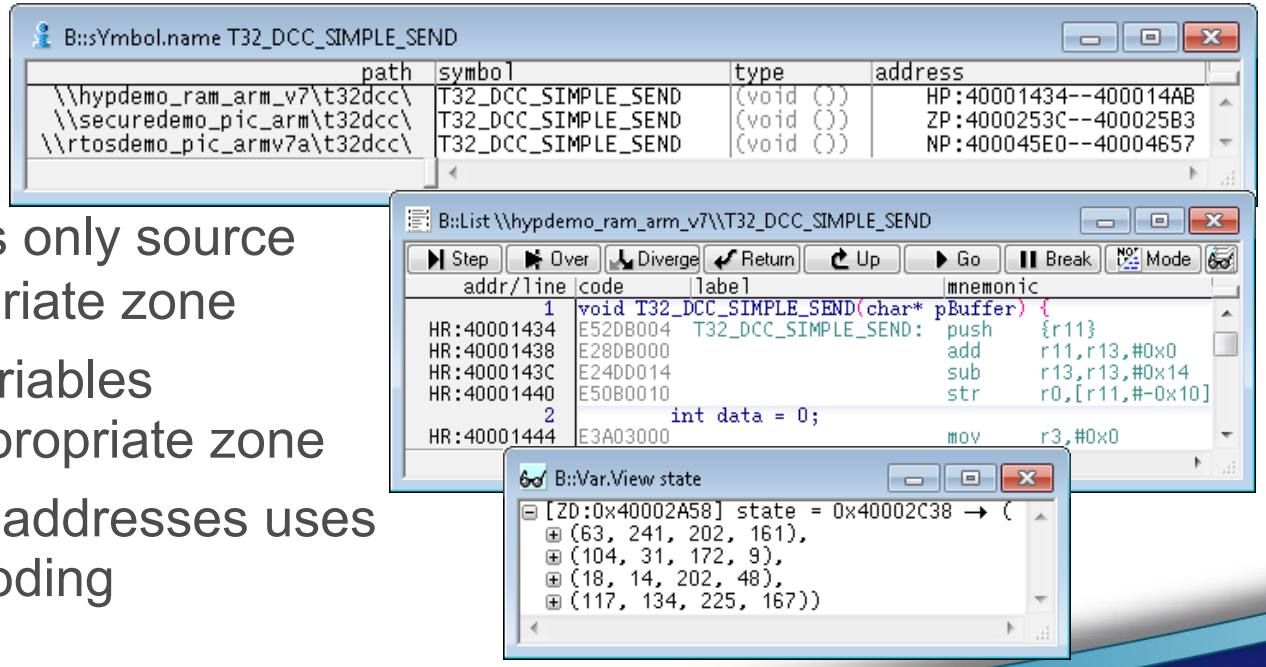
- CAUTION:
  - Each zone (non-secure, secure, hypervisor) has an own MMU translation!
  - If the translation is different for each zone, symbols may not match!
  - Software breakpoints may be set into wrong code!
  - Onchip breakpoints may halt where not desired

Agenda

- TrustZone And CPU Modes In TRACE32
- Default Behavior Of The Debugger
- ▶ Special TrustZone Support
- Debugging Through The Zones
- Outlook To Multiple Guests

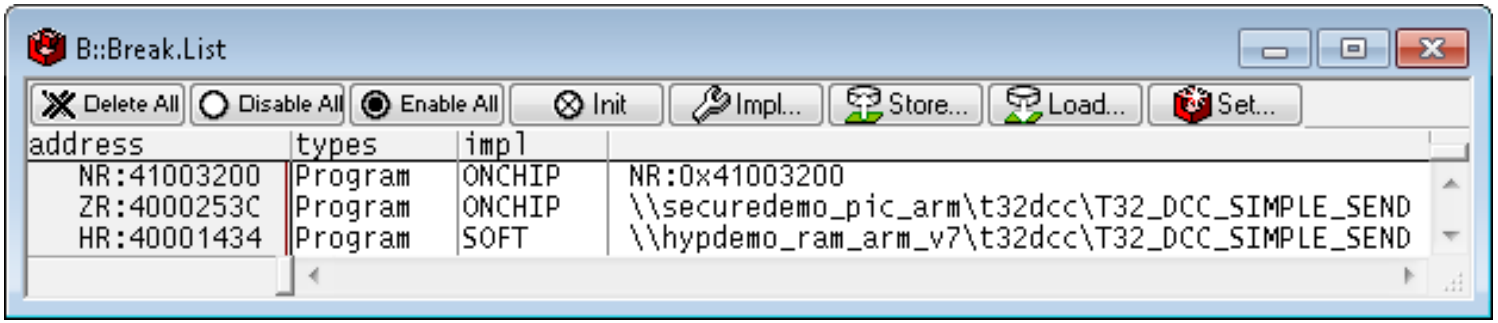
Special TrustZone Support

- Support of TrustZones with so called “Zone Spaces”
  - **`SYStem.Option ZoneSpaces ON`**
- Symbols are mapped to a specific zone (N: / H: / Z:)
  - E.g.: **`Data.LOAD.Elf symbols.elf H:0`**
  - **`sYmbol.name`** shows symbols mapped to zones
  - **`List`** window matches only source code loaded to appropriate zone
  - Accessing symbols/variables automatically uses appropriate zone
  - Accessing symbols or addresses uses appropriate MMU decoding



Special TrustZone Support

- Breakpoints are “zone aware”
  - Software breakpoints set on symbols are automatically set in correct code
  - Software breakpoints on addresses follow the access class
  - Onchip breakpoints react in specified zone/access class only

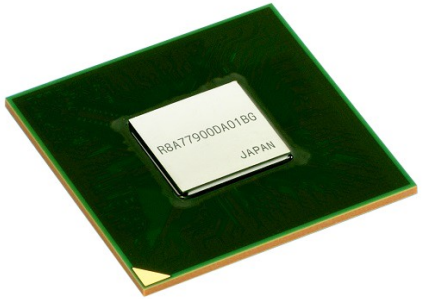


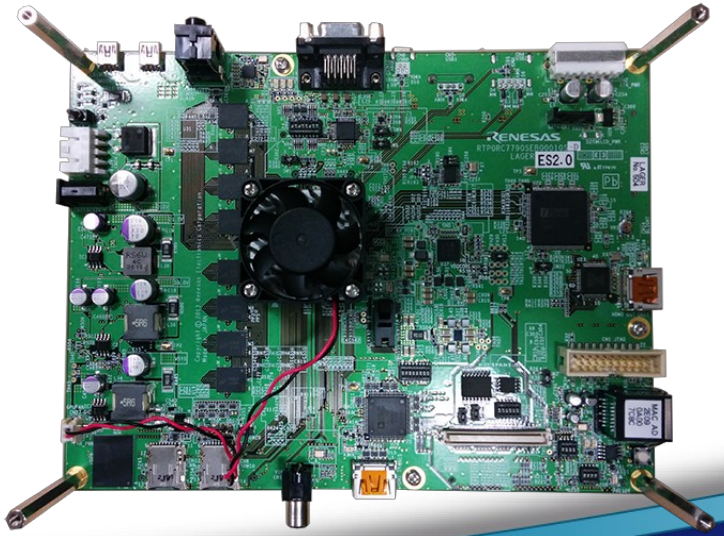
Agenda

- TrustZone And CPU Modes In TRACE32
- Default Behavior Of The Debugger
- Special TrustZone Support
- ▶ Debugging Through The Zones
- Outlook To Multiple Guests

Debugging Through The Zones

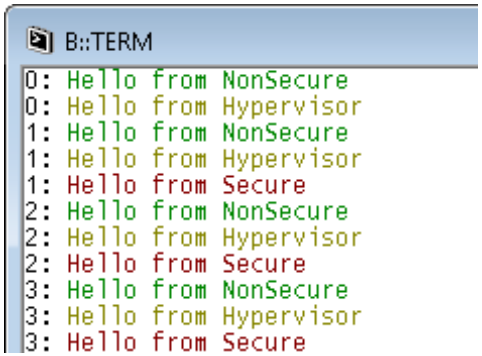
- Demo hardware
  - TRACE32 PowerDebug PRO + PowerTrace PX + Debug Cable Cortex-A/R + ETM Preprocessor
  - Renesas Lager Board
  - CPU: R-CarH2 (Quad Cortex-A15, Quad Cortex-A7)





Debugging Through The Zones

- Demo application running on Cortex-A7
  - small hand-written monitor for switching secure/non-secure
    - MMU mapping: `Z:0x40001200--0x400012ff AZ:0x84001200`
  - small hand-written hypervisor
    - MMU mapping: `H:0x40001100--0x40001fff AH:0x83081000`
  - Decryption (AES) demo in secure mode
    - MMU mapping: `Z:0x40001300--0x40002fff AZ:0x84001300`
  - FreeRTOS running in non-secure mode
    - MMU mapping: `N:0x40001000--0x40006fff I:0x20001000 AN:0x81001000`
    - FreeRTOS Awareness set to work on non-secure mode only `TASK.ACCESS N:`



Debugging Through The Zones

- Demo application running on Cortex-A7
  - Overlapping virtual addresses in all three zones!
    MMU tables are set up to map each zone to a different physical address

B::MMU.List NonSecPageTable

address	physical	sec	d	size	permissions
N:00000000--013FFFFF					
N:01400000--01407FFF	I:00:01400000--01407FFF	ns		00001000	P:readwrite
N:01408000--3FFFFFFF					
N:40000000--40006FFF	I:00:20000000--20006FFF	ns		00001000	P:readwrite
N:40007000--F0FFFFFF					
N:F1000000--F1007FFF	I:00:F1000000--F1007FFF	ns		00001000	P:readwrite
N:F1008000--FBFFFFFF					
N:FC000000--FC001FFF	I:00:40000000--40001FFF	ns		00001000	P:readwrite

B::MMU.List HypPageTable

address	physical	sec	d	size	permissions
H:00000000--3FFFFFFF					
H:40000000--40005FFF	AH:00:83080000--83085FFF	ns		00001000	P:readwrite
H:40006000--6663FFFF					
H:66640000--66646FFF	AH:00:83080000--83086FFF	ns		00001000	P:readwrite
H:66647000--FFFFFFFF					

B::MMU.List SecPageTable

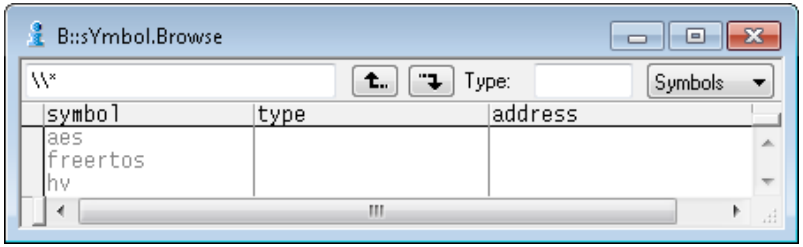
address	physical	sec	d	size	permissions
Z:00000000--3FFFFFFF					
Z:40000000--40005FFF	AZ:00:84000000--84005FFF	s		00001000	P:readwrite
Z:40006000--6663FFFF					
Z:66640000--66646FFF	AZ:00:84000000--84006FFF	s		00001000	P:readwrite
Z:66647000--FCFFFFFF					
Z:FD000000--FD005FFF	AZ:00:85500000--85505FFF	s		00001000	P:readwrite
Z:FD006000--FFFFFFFF					

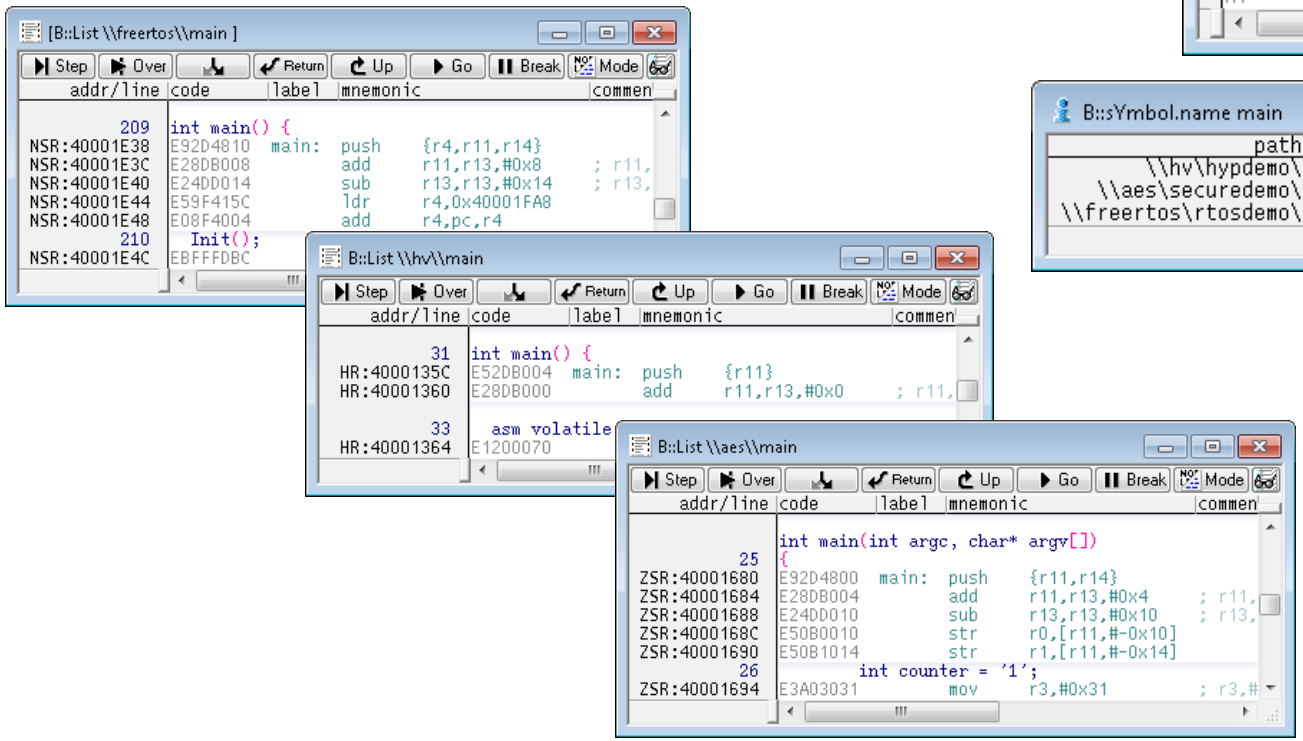
B::MMU.List IntermedPageTable

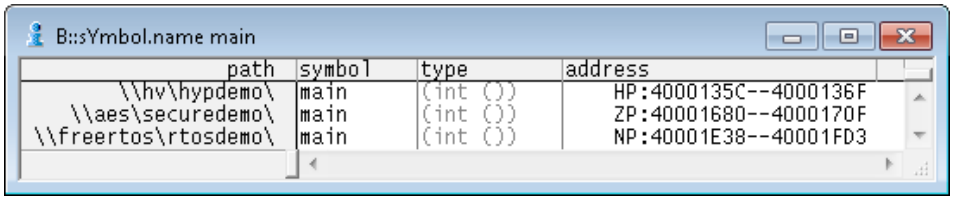
address	physical	sec	d	size	permissions
I:00:00000000--013FFFFF					
I:00:01400000--01407FFF	A:00:01400000--01407FFF	ns		00001000	read/write
I:00:01408000--1FFFFFFF					
I:00:20000000--20006FFF	A:00:81000000--81006FFF	ns		00001000	read/write
I:00:20007000--3FFFFFFF					
I:00:40000000--40001FFF	A:00:82008000--82009FFF	ns		00001000	read/write
I:00:40002000--A00FFFFF					
I:00:A0100000--A0163FFF	A:00:A0100000--A0163FFF	ns		00001000	read/write
I:00:A0164000--A01FFFFF					
I:00:A0200000--A0263FFF	A:00:A0200000--A0263FFF	ns		00001000	read/write
I:00:A0264000--F0FFFFFF					
I:00:F1000000--F1007FFF	A:00:F1000000--F1007FFF	ns		00001000	read/write
I:00:F1008000--FFFFFFFF					

Debugging Through The Zones

- Accessing symbols
  - Different set of symbols for each zone
    - E.g.: each zone has its own “main”



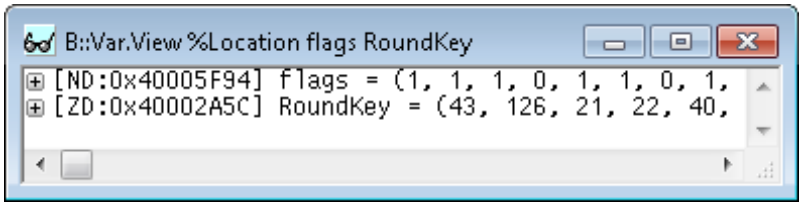




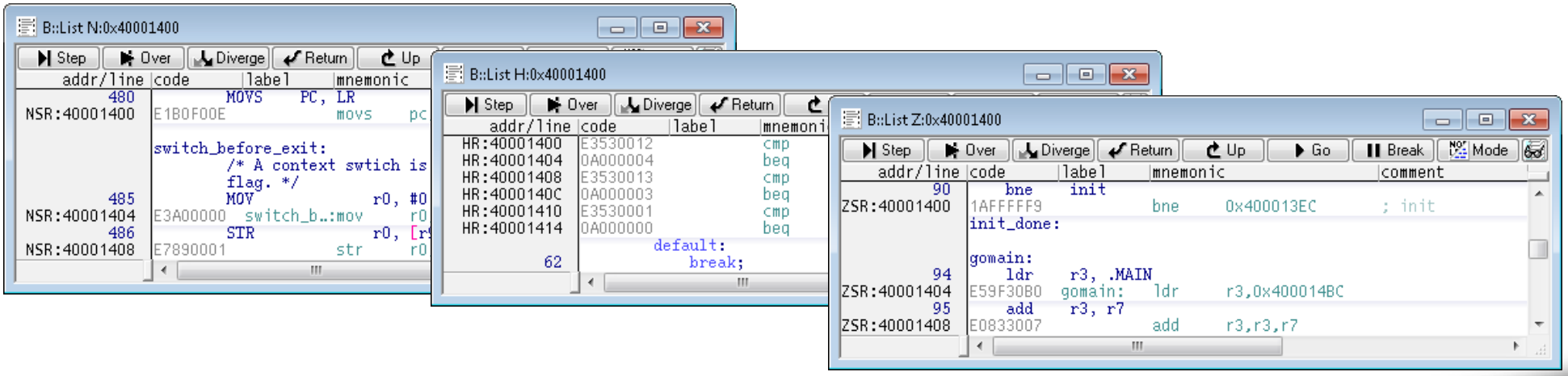
LAUTERBACH DEVELOPMENT TOOLS

Debugging Through The Zones

- Accessing symbols
  - Variables automatically point to the correct zone

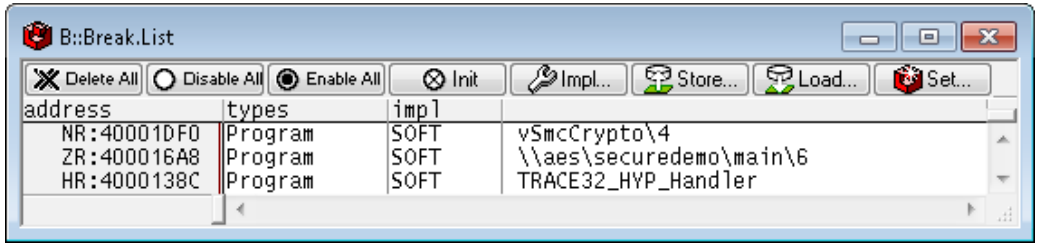


- Different code and different source for each zone

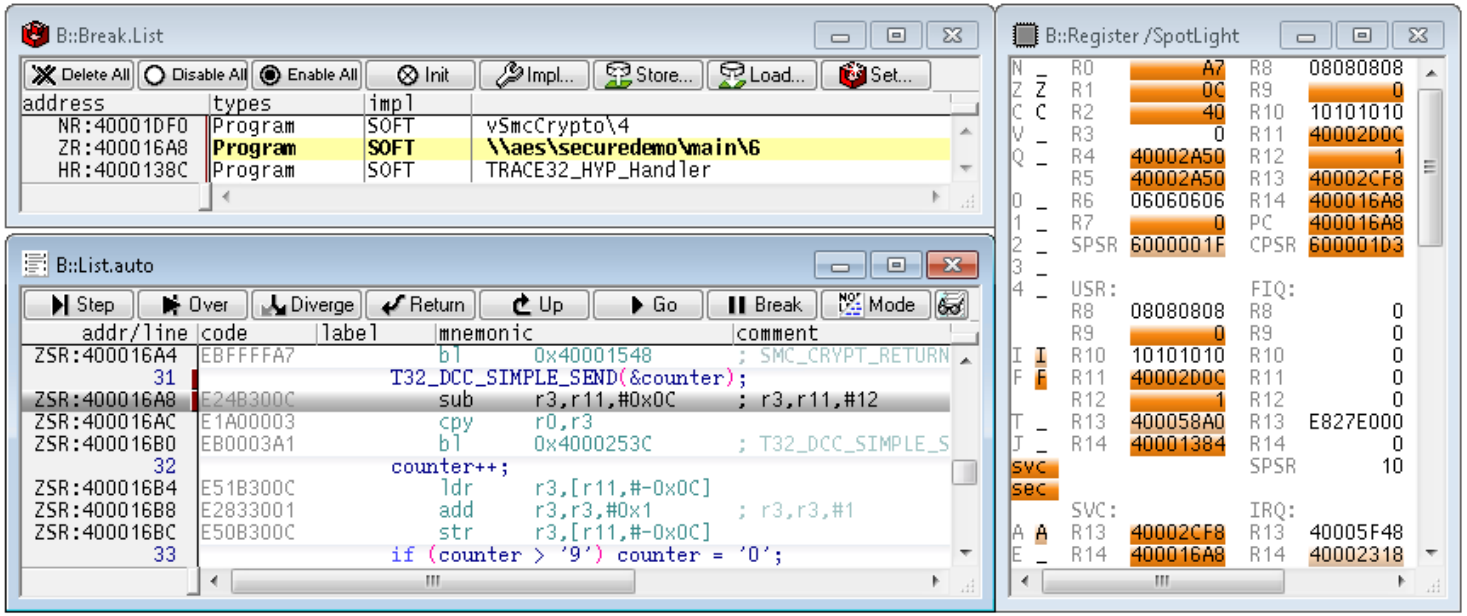


Debugging Through The Zones

- Breakpoints can be set in any zone

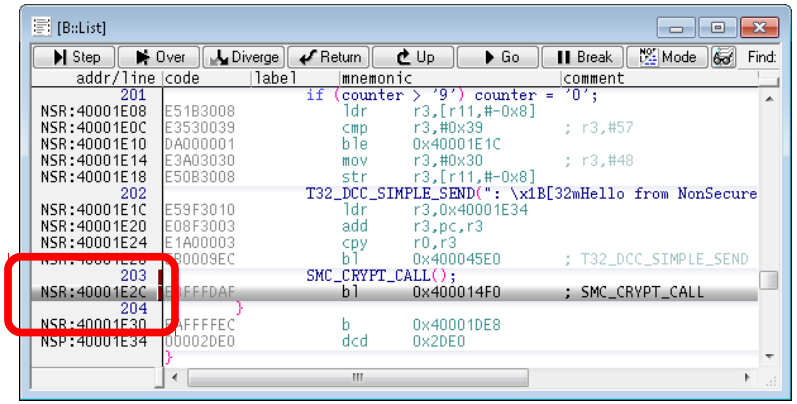


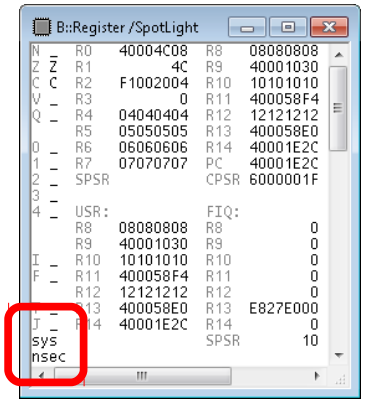
- **`List.auto`** and **`Register`** windows follow current zone



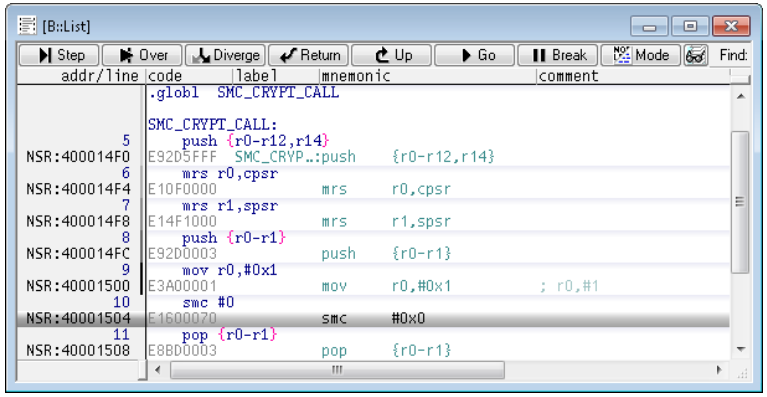
Debugging Through The Zones

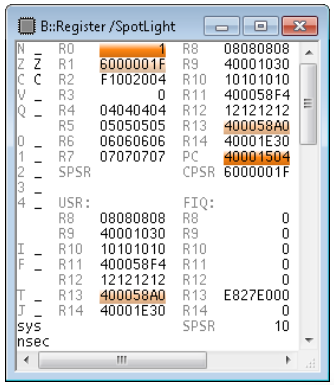
- Stepping through the mode changes
  - We start in non-secure zone





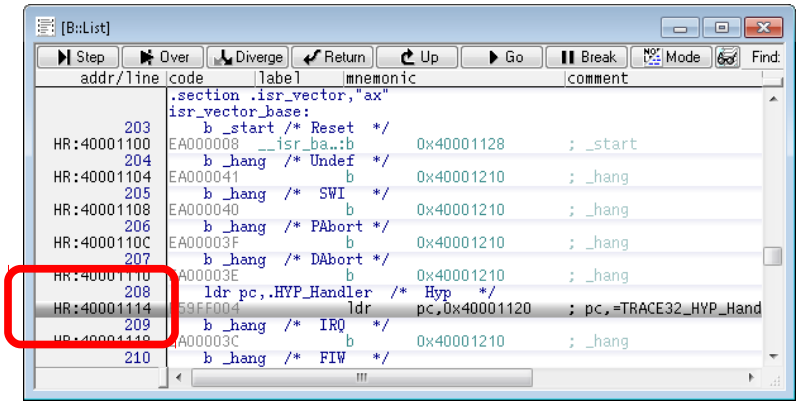
  - Non-secure executes “smc #0”

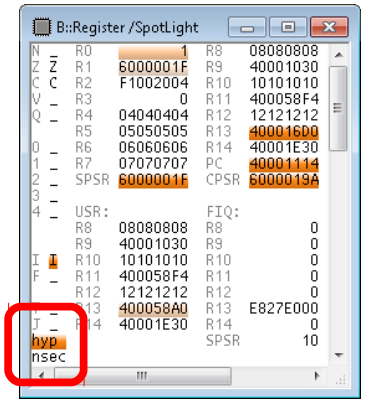




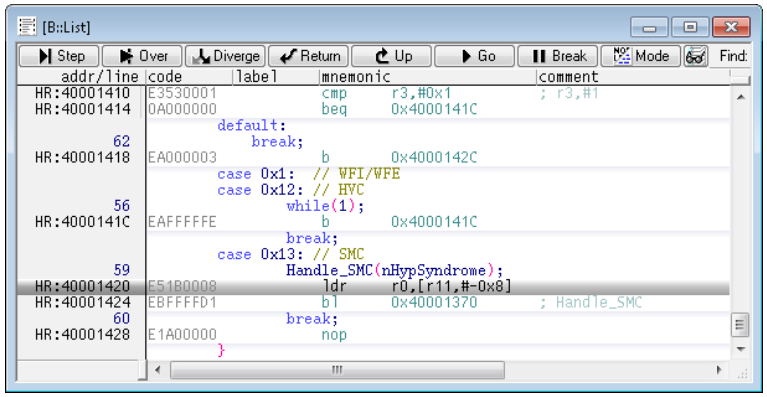
Debugging Through The Zones

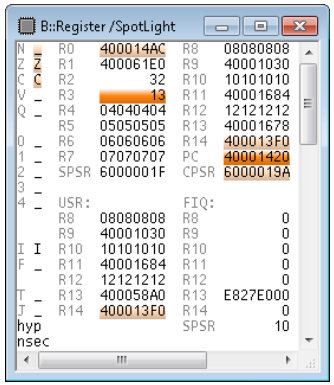
- Stepping through the mode changes
  - Single step into hypervisor (set **`TrOnchip.HENTRY ON`**)





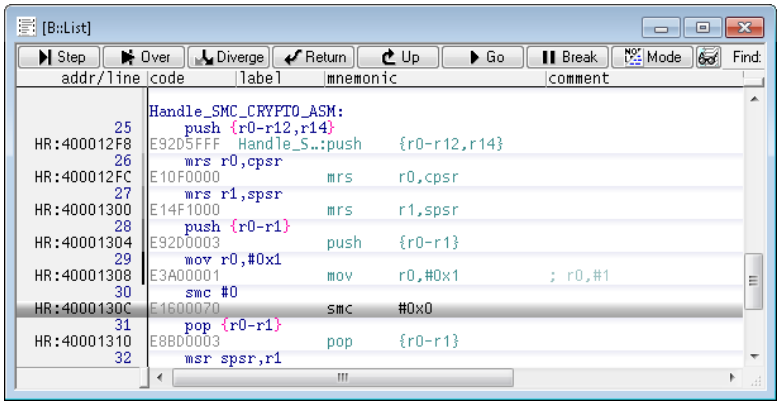
  - Debug hypervisor until switch to monitor

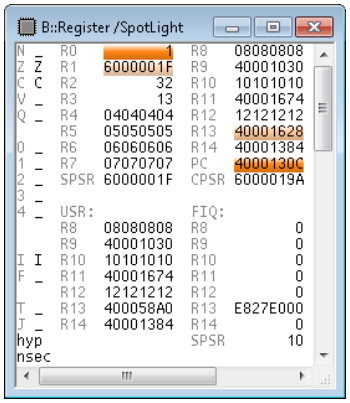




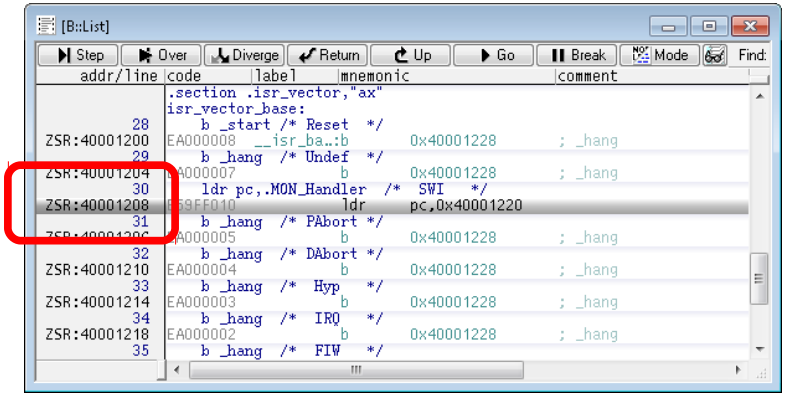
Debugging Through The Zones

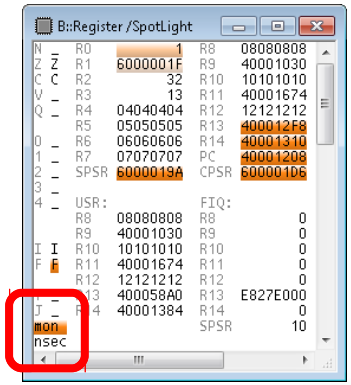
- Stepping through the mode changes
  - Hypervisor executes “smc #0”





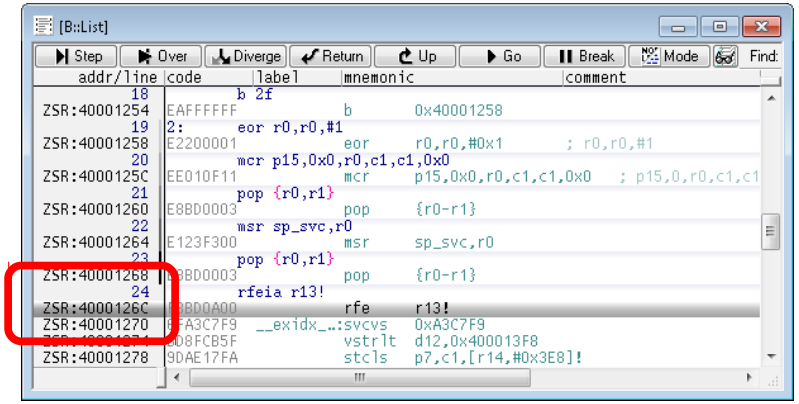
  - Step into monitor mode (note: “nsec” not valid due to monitor mode)

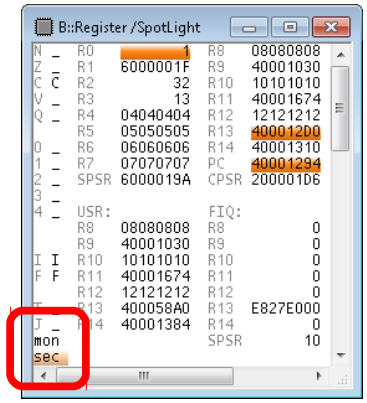




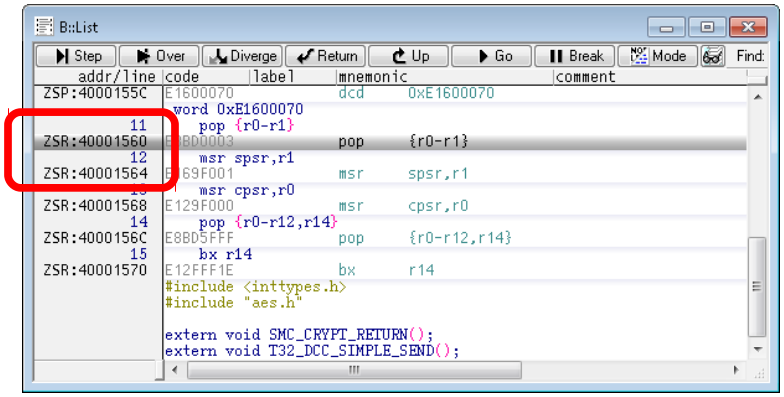
Debugging Through The Zones

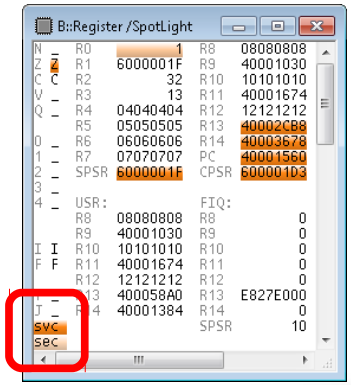
- Stepping through the mode changes
  - Monitor sets up secure mode and executes “rfe”





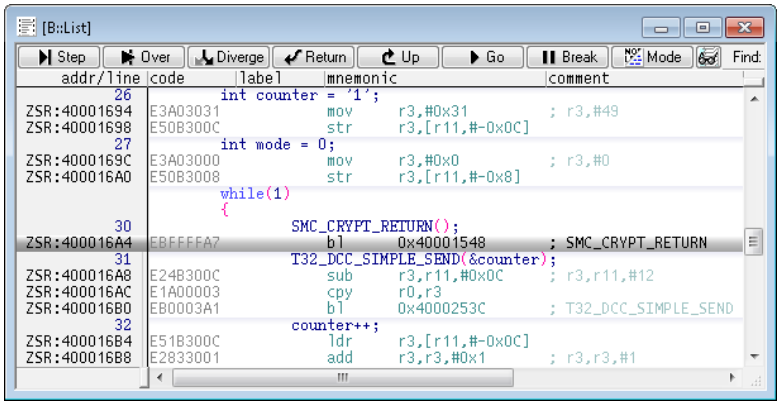
  - Step into secure zone

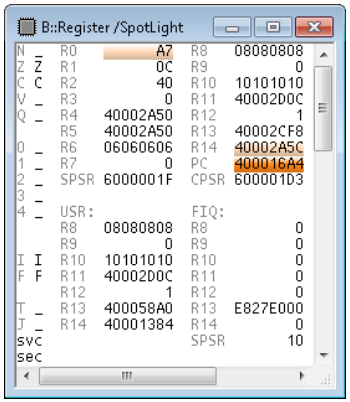




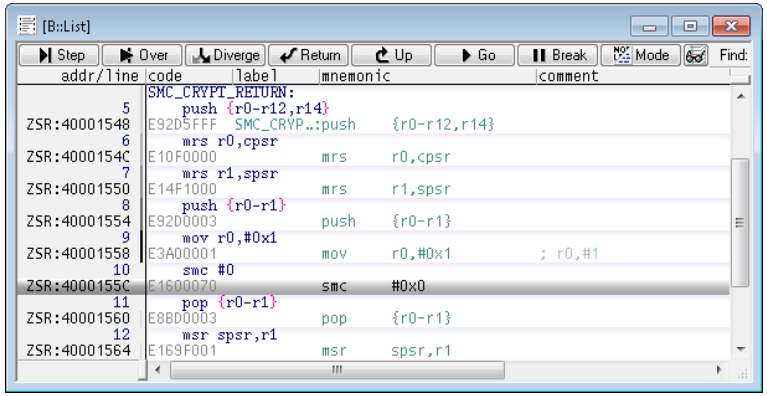
Debugging Through The Zones

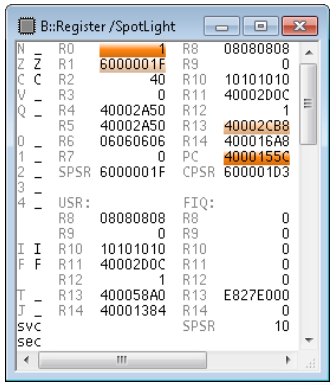
- Stepping through the mode changes
  - Debug secure zone until switch to monitor





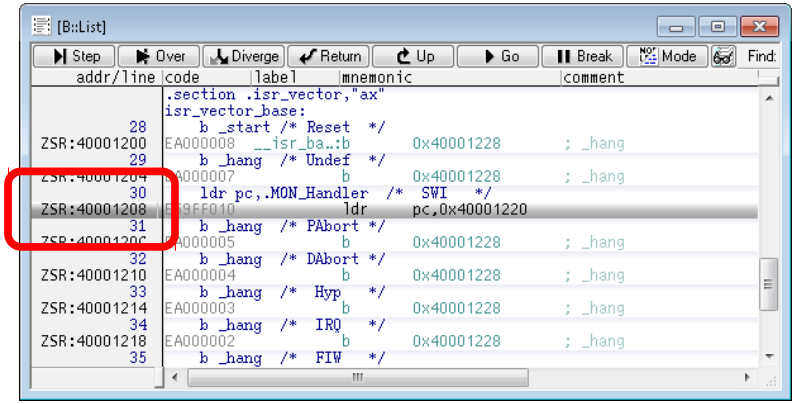
  - Secure executes “smc #0”

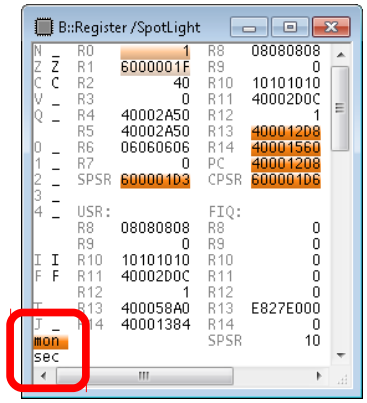




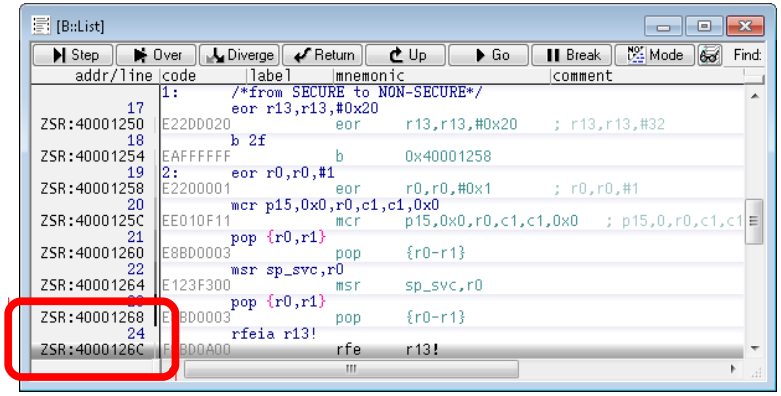
Debugging Through The Zones

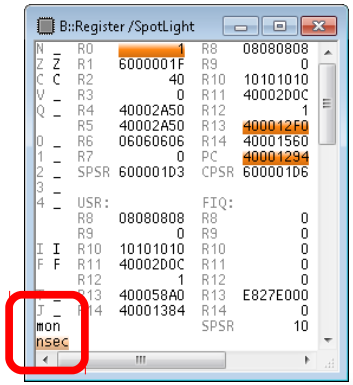
- Stepping through the mode changes
  - Step into monitor mode





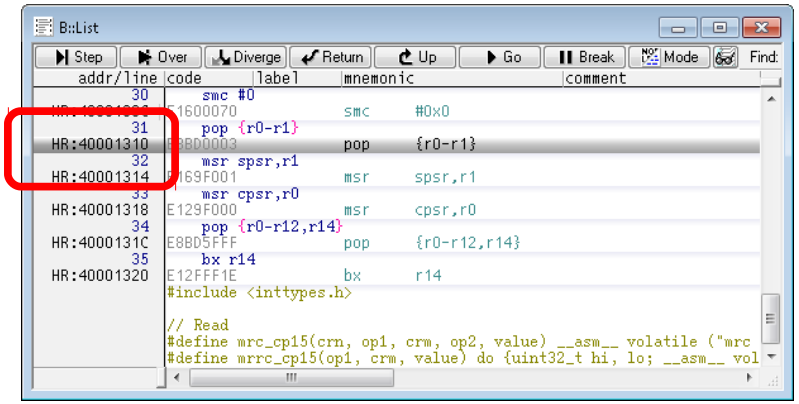
  - Monitor sets up non-secure mode and executes “rfe”

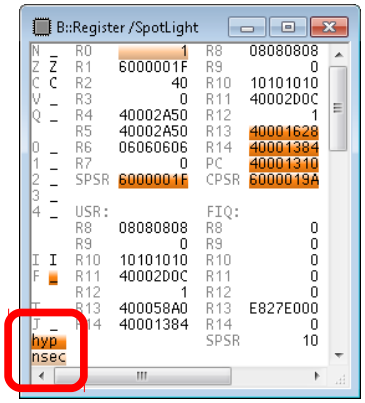




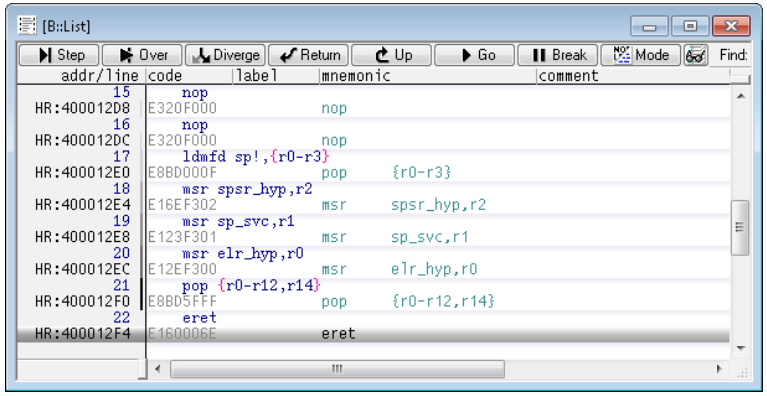
Debugging Through The Zones

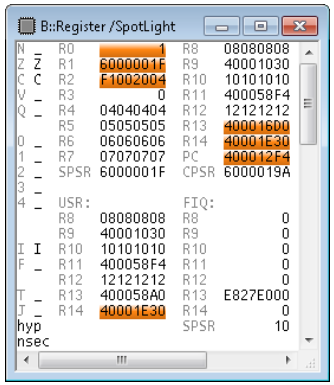
- Stepping through the mode changes
  - Monitor mode returns to hypervisor





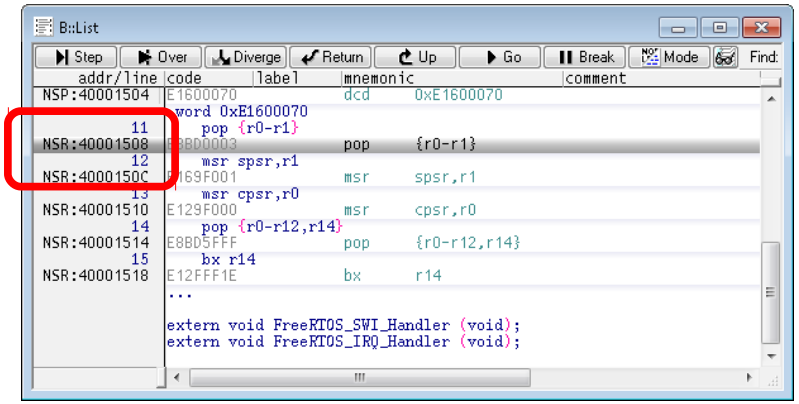
  - Hypervisor executes “eret”

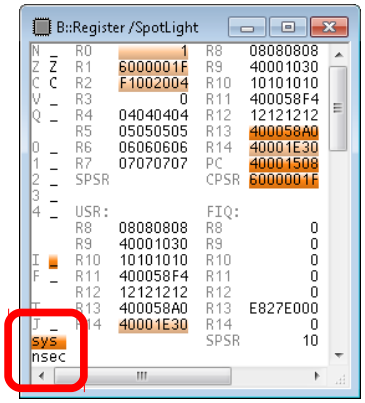




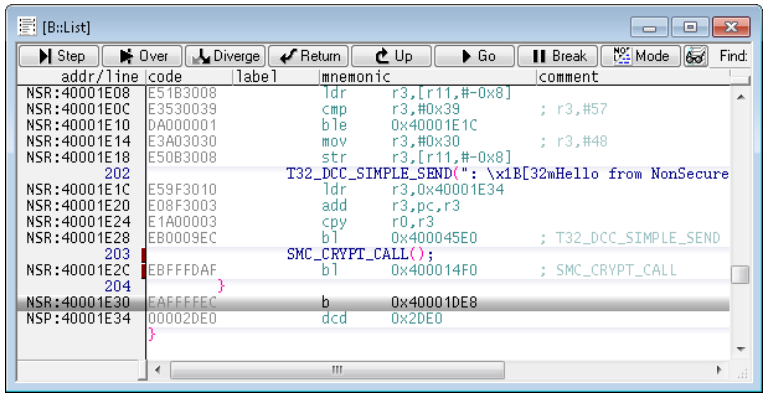
Debugging Through The Zones

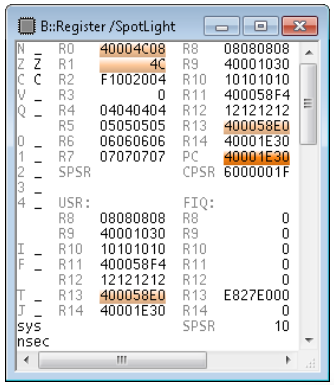
- Stepping through the mode changes
  - Step into non-secure mode





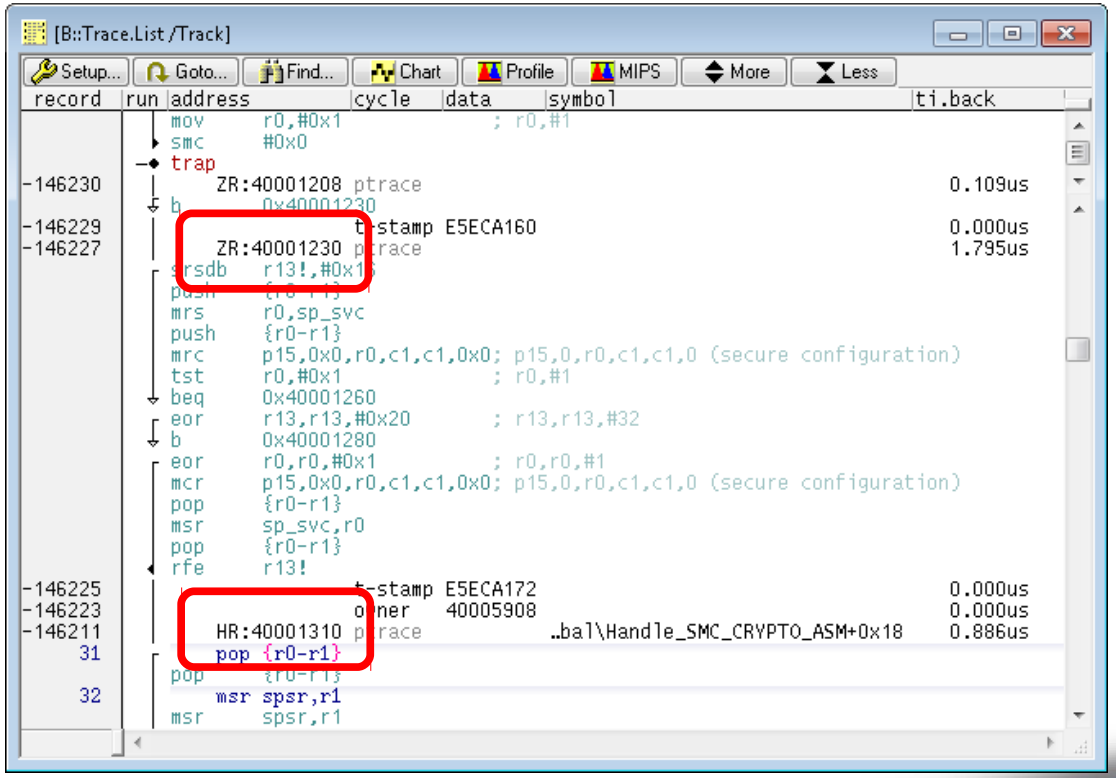
  - Return from subroutine, and we're back where we started!





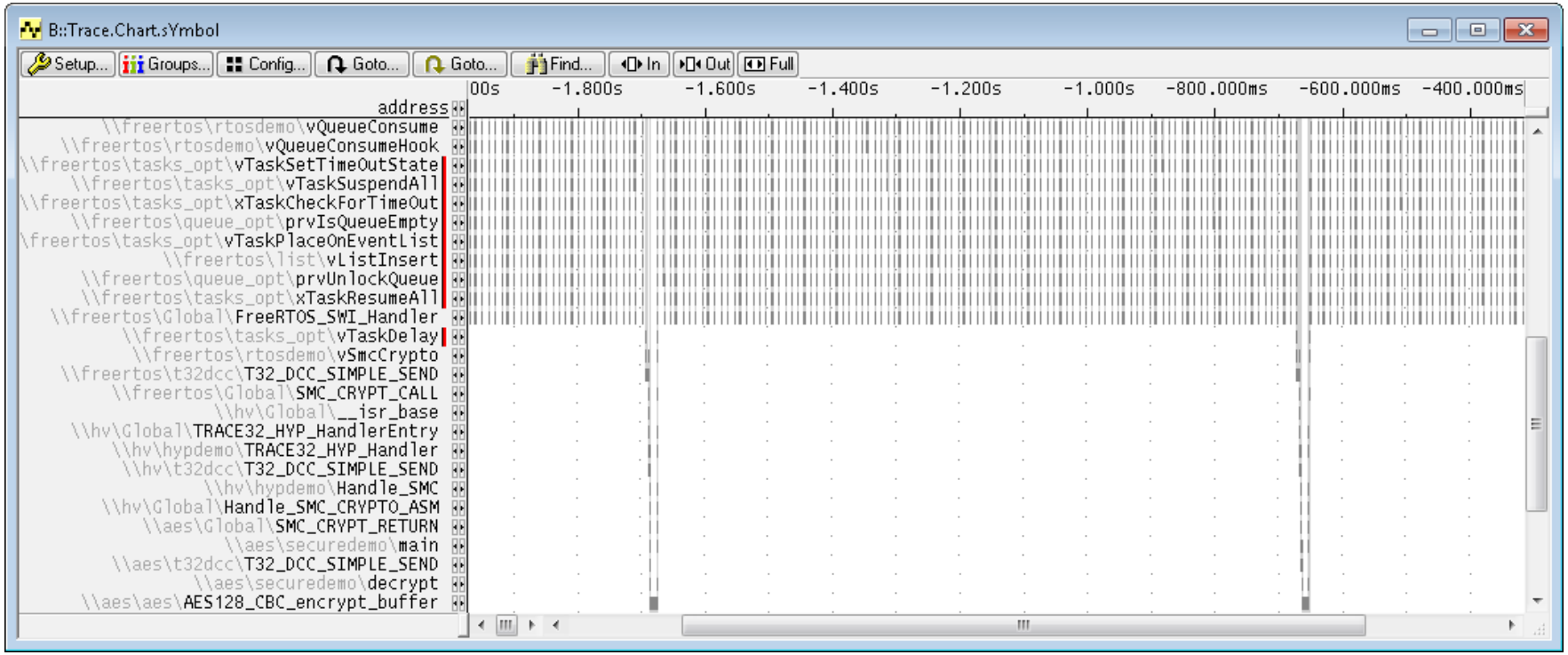
Debugging Through The Zones

- Tracing the zones
  - The ETM stream contains information about which zone is active
  - TRACE32 detects the zone switches and adjusts the access classes to the addresses in the trace.



Debugging Through The Zones

- Tracing the zones
  - Symbol time chart over zone switches



Debugging Through The Zones

- Tracing the zones
  - Statistics of recorded functions of all zones
    (interrupt) = hypervisor; (unknown) = secure zone (unknown to RTOS)

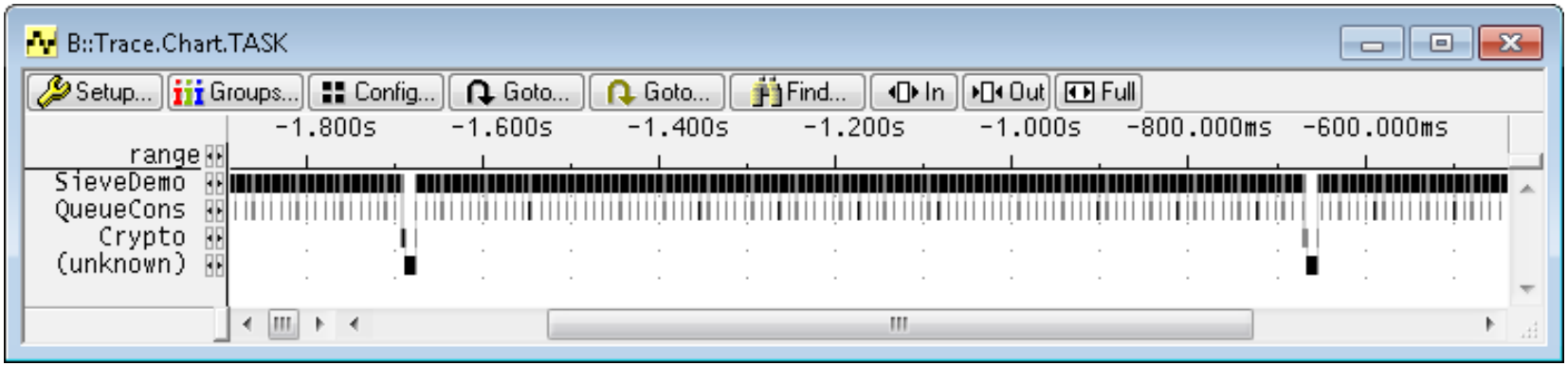
B::Trace.STATistic.TREE TASK DEFault

Setup... | Groups... | Config... | Goto... | Detailed | Nesting | Chart

task	tree	total	min	max	avr	count	intern%	1% 2%
SieveDemo	(root)	2.119s	-	2.119s	2.119s	-	6.550%	
SieveDemo	sieve	1.102s	37.614us	46.356us	40.370us	27288.(1/0)	49.233%	
SieveDemo	xTaskGetTickCount	6.459ms	0.224us	0.832us	0.237us	27288.	0.288%	
SieveDemo	func2	300.657ms	10.463us	13.197us	11.018us	27288.(0/1)	9.981%	
SieveDemo	func9	221.002ms	7.631us	9.476us	8.099us	27287.	5.273%	
SieveDemo	func13	331.460ms	11.672us	15.582us	12.147us	27287.	3.941%	
SieveDemo	xQueueGenericSend	10.908ms	40.062us	42.999us	41.317us	264.	0.041%	
(interrupt)	(root)	-	-	-	-	-	0.000%	
(interrupt)	pVectorTable+0x18	45.218ms	18.245us	43.470us	20.507us	2205.	0.470%	
(interrupt)	vApplicationIRQHandler	33.355ms	14.267us	29.313us	15.127us	2205.	0.433%	
(interrupt)	FreeRTOS_Tick_Handler	23.650ms	9.982us	25.022us	10.725us	2205.	0.620%	
(interrupt)	vTaskSwitchContext	1.335ms	4.440us	5.381us	4.871us	274.	0.059%	
(interrupt)	__isr_base+0x14	4.000ms	2.000ms	2.001ms	2.000ms	2.	<0.001%	
(interrupt)	TRACE32_HYP_Handler	3.997ms	1.998ms	1.999ms	1.998ms	2.	<0.001%	
QueueCons	(root)	42.356ms	-	42.356ms	42.356ms	-	0.010%	
QueueCons	xQueueGenericReceive	42.003ms	157.399us	161.478us	158.502us	265.(1/1)	0.098%	
QueueCons	vQueueConsumeHook	112.028us	0.281us	0.855us	0.424us	264.	0.005%	
Crypto	(root)	3.840ms	-	3.840ms	3.840ms	-	<0.001%	
Crypto	vTaskDelay	91.329us	45.854us	45.854us	30.443us	3.(1/1)	<0.001%	
Crypto	T32_DCC_SIMPLE_SEND	3.742ms	2.464us	1.926ms	935.506us	4.	0.167%	
(unknown)	(root)	23.459ms	-	23.459ms	23.459ms	-	<0.001%	
(unknown)	ZR:0x40001208	7.677us	1.495us	2.193us	1.919us	4.	<0.001%	
(unknown)	T32_DCC_SIMPLE_SEND	3.554ms	50.572us	1.726ms	888.535us	4.	0.158%	
(unknown)	decrypt	19.893ms	9.944ms	9.949ms	9.946ms	2.	<0.001%	
(unknown)	AES128_CBC_encrypt_buffer	19.874ms	9.936ms	9.938ms	9.937ms	2.	0.001%	

Debugging Through The Zones

- Tracing the zones
  - Task runtime measurements, if one zone contains an RTOS
    - Here: FreeRTOS running in non-secure
    - (unknown) = secure zone, not known to FreeRTOS

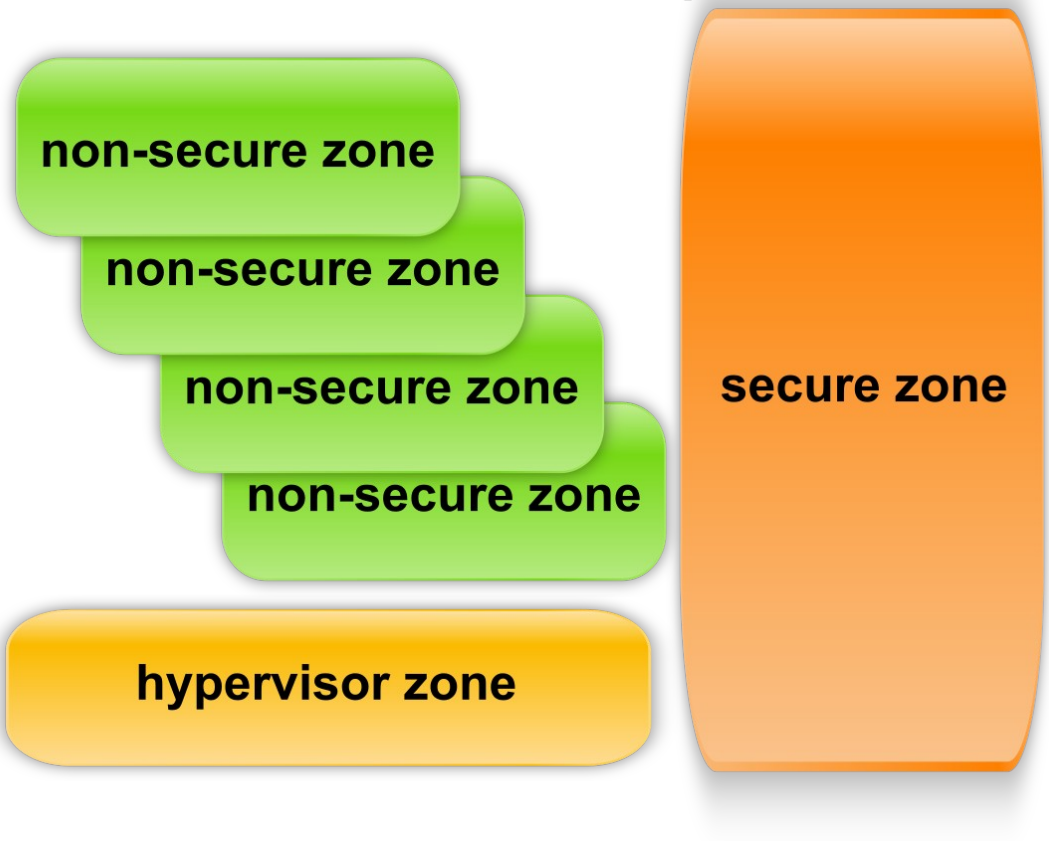


Agenda

- TrustZone And CPU Modes In TRACE32
- Default Behavior Of The Debugger
- Special TrustZone Support
- Debugging Through The Zones
- ► Outlook To Multiple Guests

Outlook To Multiple Guests

CPU Modes - Multiple Guests



Outlook To Multiple Guests

- Ongoing efforts to support multiple guests
  - Work in progress
  - Multiple guests in access class N: (non-secure zone)
- Introduction of a “machine id”
  - Each guest gets its own machine id
  - Will be an addition to the virtual address
- Symbol handling separate for each “machine”
  - Just like the zones, but extended to several guests
- OS Awareness for each machine
  - Loading several OS awareness at the same time

Outlook To Multiple Guests

- Two-stage MMU support
  - Debugger does its own MMU translation and table walk
  - Access to hypervisor and guest simultaneously
  - Access to *every* guest simultaneously
  - Access to *every* process within every guest simultaneously
  - Prerequisite: “Hypervisor Awareness” plus RTOS Awareness
    - HV Awareness provides “VTTB” of guests
    - RTOS awareness provides “TTB” of processes

Summary

- TRACE32 separates the zones for you
- Debugging simultaneously in each zone, even with overlapping MMU
- Debugging the zone switches
- Tracing decodes the zones accordingly

TRACE32® ready for ARM® TrustZone®!

Thank You!

Rudi Dienstbeck

rudolf.dienstbeck@lauterbach.com
Phone: +49 8102 9876 175

Questions?