

ARM® TrustZone® and Hypervisor Debugging

ARM® CORTEX®-A/R EXPERT DAY

Meet the Debug Experts

Agenda

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

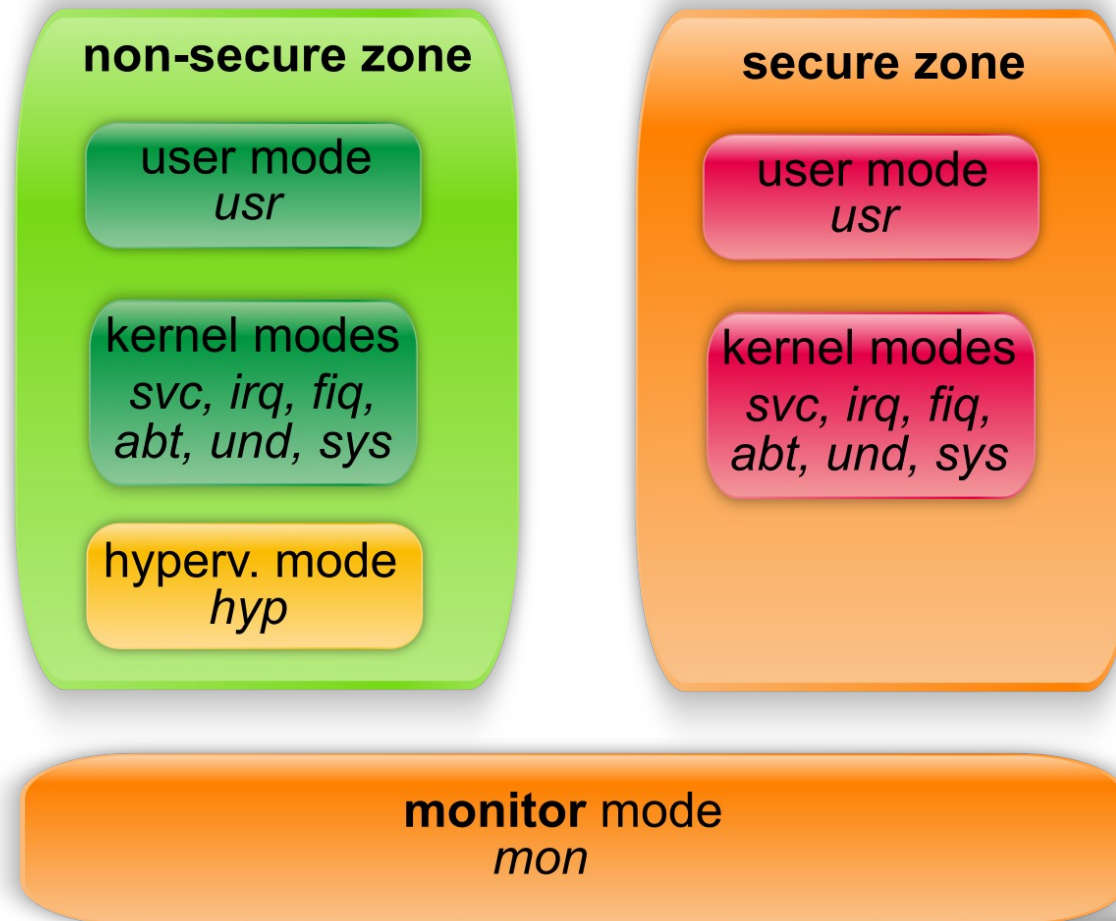
Agenda

► TrustZone And CPU Modes In TRACE32

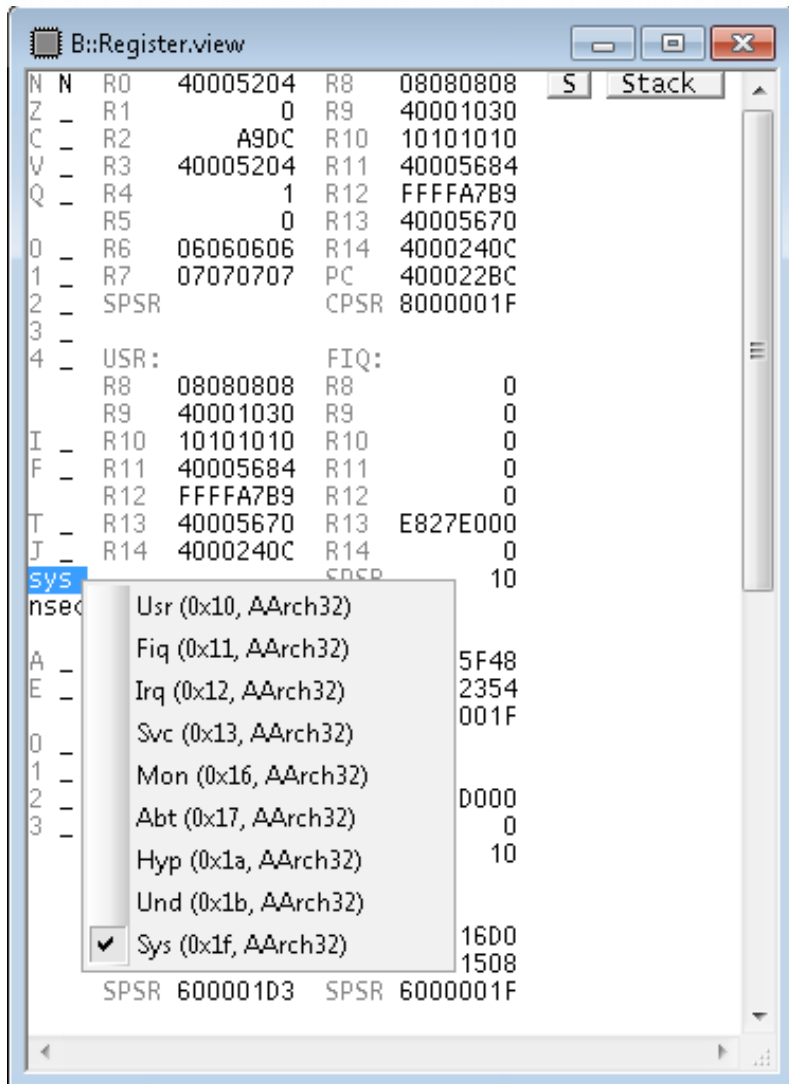
- Default Behavior Of The Debugger
- Special TrustZone Support
- Debugging Through The Zones
- Outlook To Multiple Guests

TrustZone And CPU Modes In TRACE32

CPU Modes - Core View



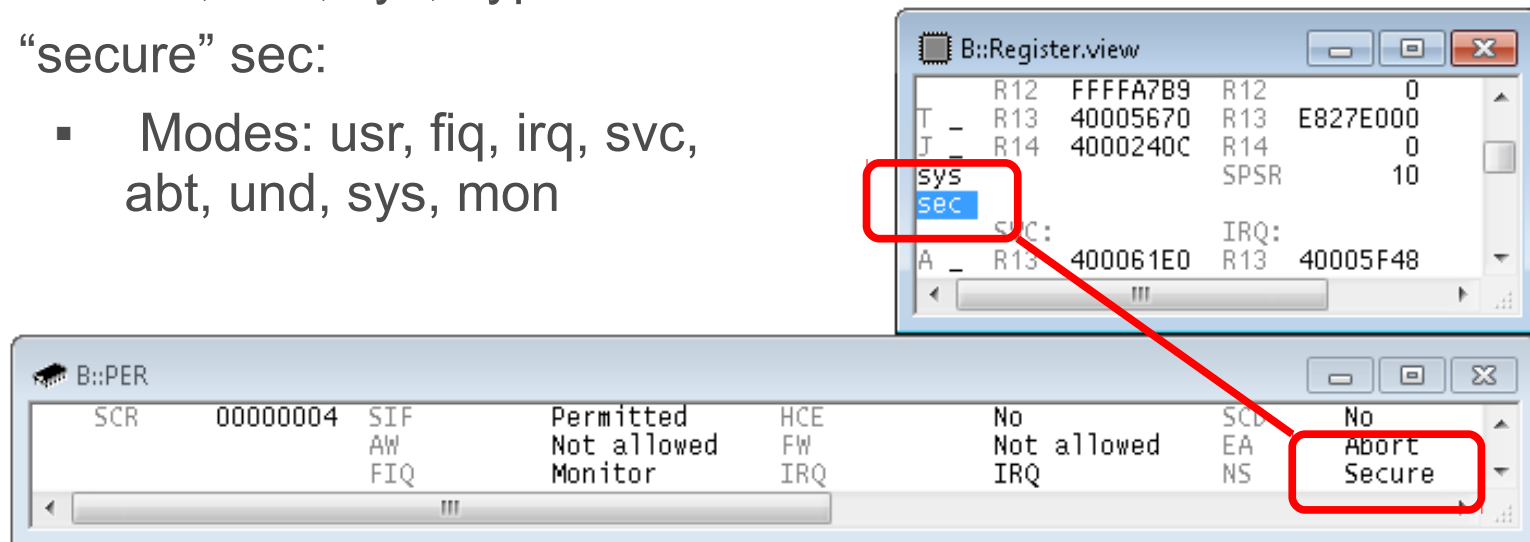
TrustZone And CPU Modes In TRACE32



- 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**

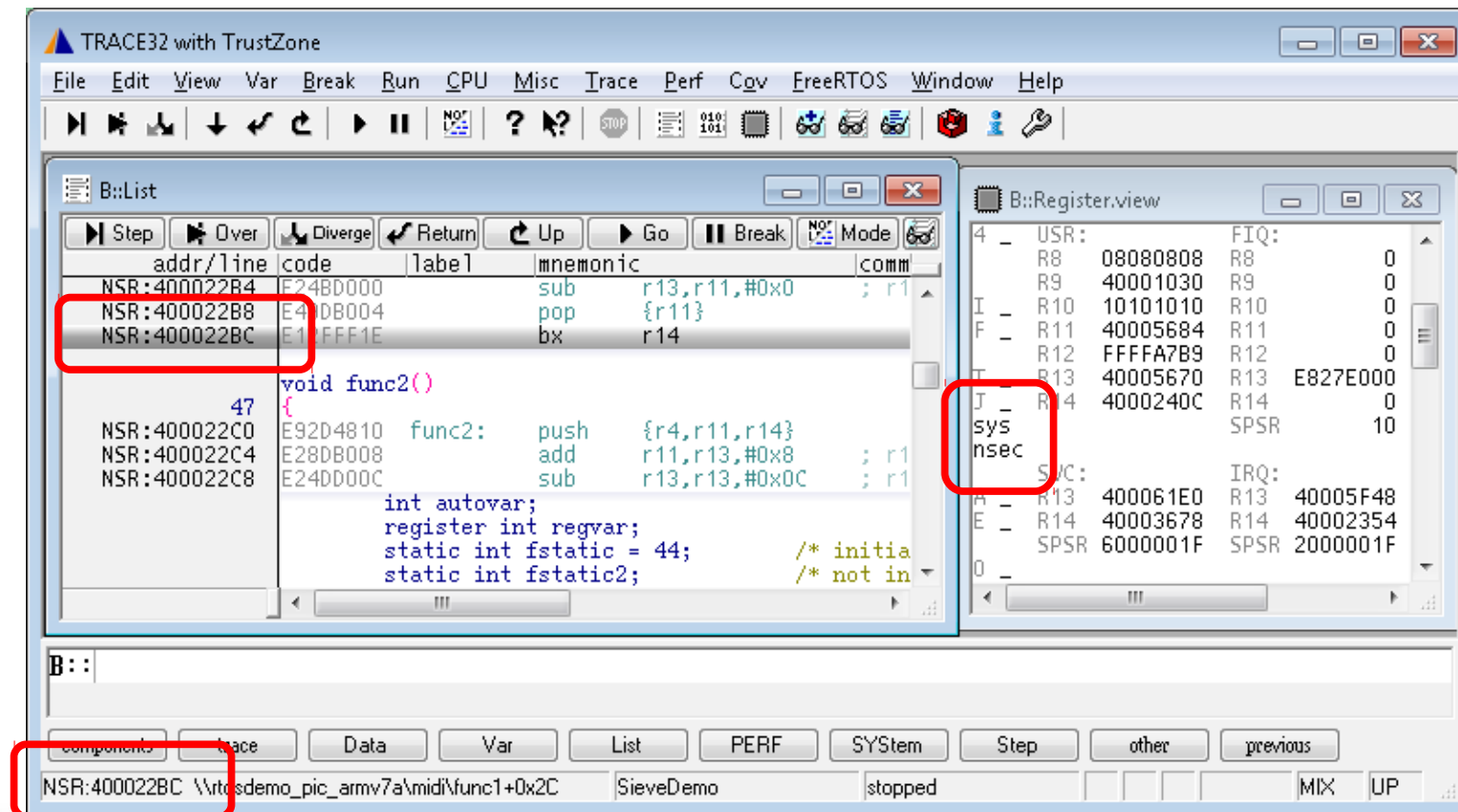
access class: **Z:**

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

010: 101 B::Data.dump 0x40000000

address	0	4	01234567
NSD:40000000	→E1A00000	E1A00000	NNAENNAE UU01UU01
NSD:40000008	E1A00000	EAF FFFFB	NNAEFFFE UU01BFFA
NSD:40000010	87BE51E0	7B567081	EQE881pV{ 8B33A019
NSD:40000018	9791B3A9	991D6FAC	9317C0B3

010: 101 B::Data.dump ZS:0x40000000

address	0	4	01234567
ZSD:40000000	→E1A00000	E1A00000	NNAENNAE UU01UU01
ZSD:40000008	E1A00000	EAF FFFFB	NNAEFFFE UU01BFFA
ZSD:40000010	95ADA736	559F08B7	6A82B82U 7D575FU
ZSD:40000018	B7F55AED	74DD3B3B	EZF575bt BZ5775bt

B::List U:sieve

	addr/line	code	label	mnemonic
	186	int sieve()		
		{		
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)

010: B::Data.dump EA:0x100

address	0	4	01234567
EANSD:0:0000100	→E320F000	E320F002	NF ES UO XS
EANSD:0:0000108	EAF FFFFD	E320F000	FFF NF DFFA UO XS
EANSD:0:0000110	E320F002	EAF FFFFD	SF ES XO XS DFFA
EANSD:0:0000118	E320F000	E320F002	NF ES UO XS

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`

The image shows four overlapping screenshots of the TRACE32 MMU mapping tables. Each table has columns for address, physical, sec, d, size, and permissions.

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--F1000000					
N:F1000000--F1008000					
N:F1008000--FC000000					

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
T:00:20007000--3FFFFFFF					

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--FC000000					

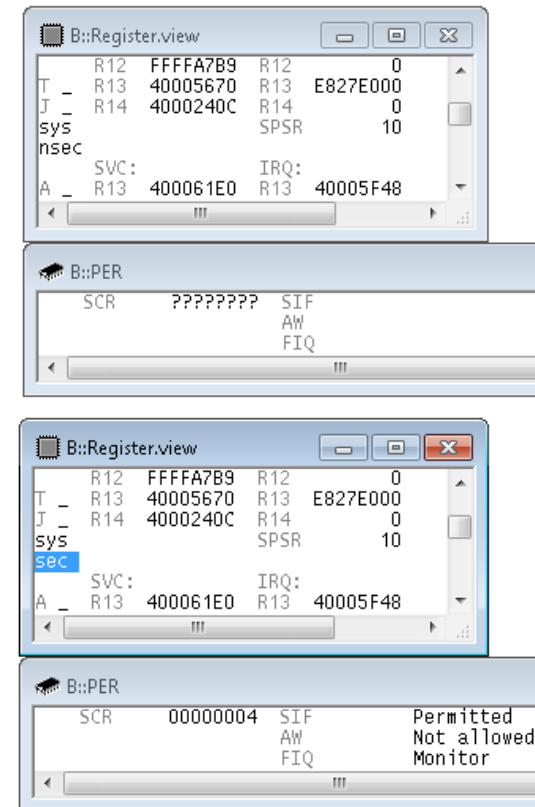
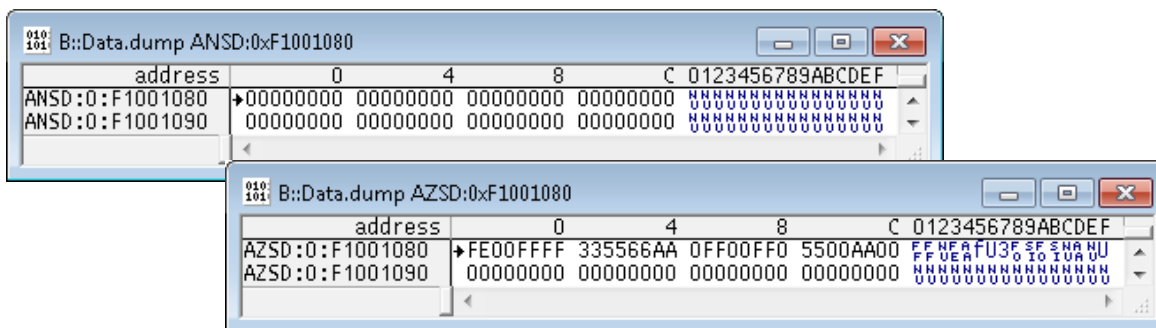
B::MMU.List SecPageTable

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--FCFFFFFFF					
Z:FD000000--FD005FFF	AZ:00:85500000--85505FFF	s		00001000	P:readwrite
Z:FD006000--FFFFFFF					

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

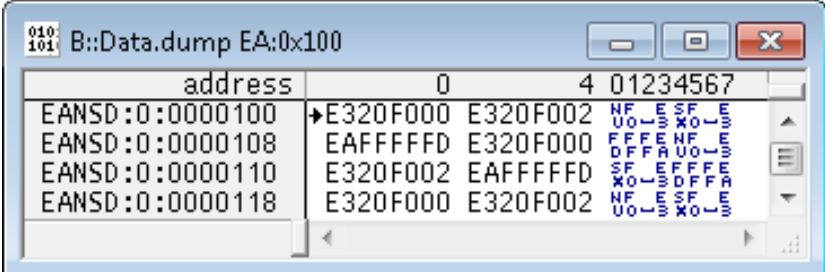
TrustZone And CPU Modes In TRACE32

- “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!



The screenshot shows a debugger window titled "B::Data.dump EA:0x100". It displays a table of memory dump data with columns for address, data, and disassembly. The data is organized into four rows, each representing a 4-byte memory location. The addresses are EANSD:0:0000100, EANSD:0:0000108, EANSD:0:0000110, and EANSD:0:0000118. The data values are E320F000, E320F002, E320F000, E320F002, E320F002, E320F000, E320F000, and E320F002. The disassembly column shows instructions like "HF UO LE SF" and "FFFA UO LE".

address	0	4	01234567
EANSD:0:0000100	E320F000	E320F002	HF UO LE SF
EANSD:0:0000108	E320F000	E320F000	FFFA UO LE
EANSD:0:0000110	E320F002	E320F000	FFFA UO LE
EANSD:0:0000118	E320F000	E320F002	HF UO LE SF

TrustZone And CPU Modes In TRACE32

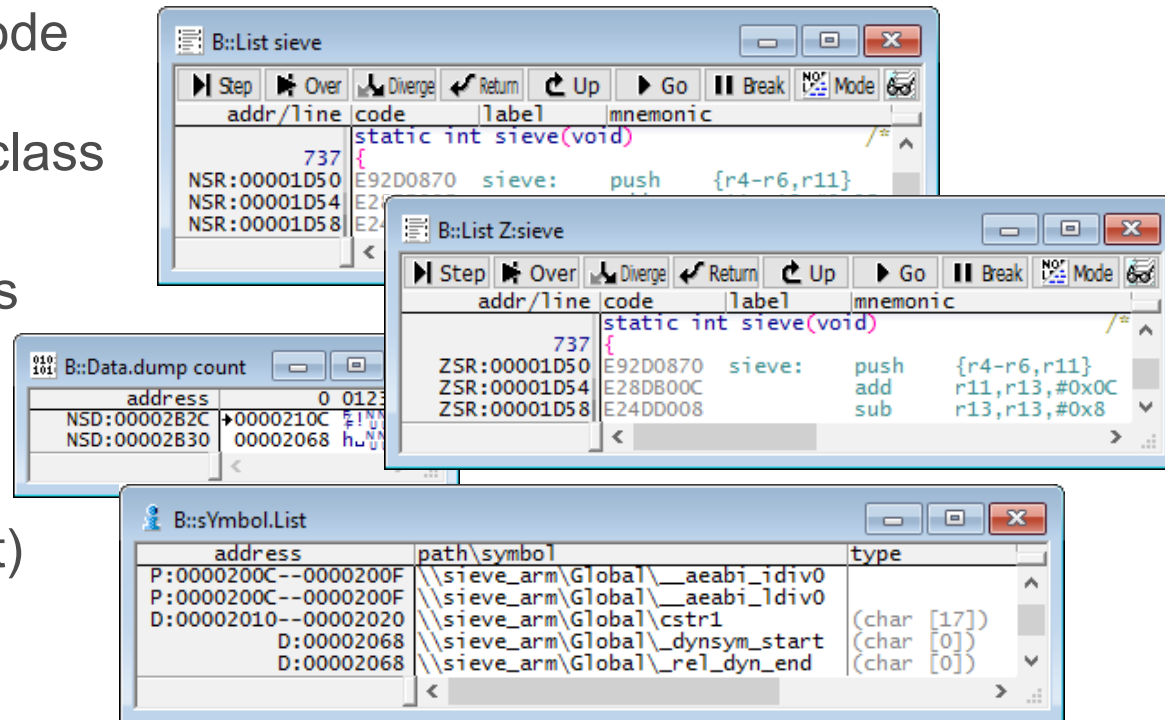
- Memory visibility with zones
 - Depending on CPU hardware implementation, secure zone may be inaccessible! (Secure debug disabled)
 - Depending on CPU mode, some memory may be invisible in non-secure zone! See slide about physical access (AN:/AZ: difference)

Agenda

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

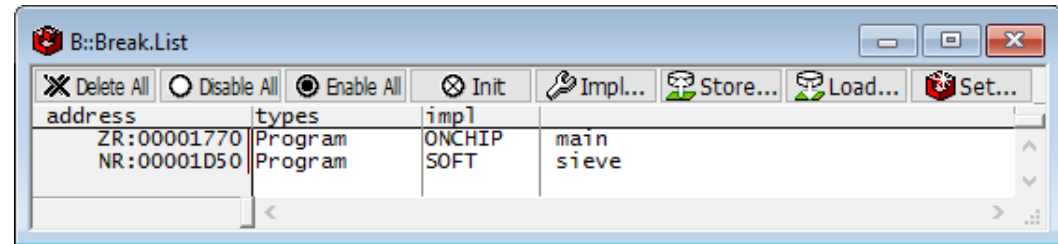
Default Behavior Of The Debugger

- 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



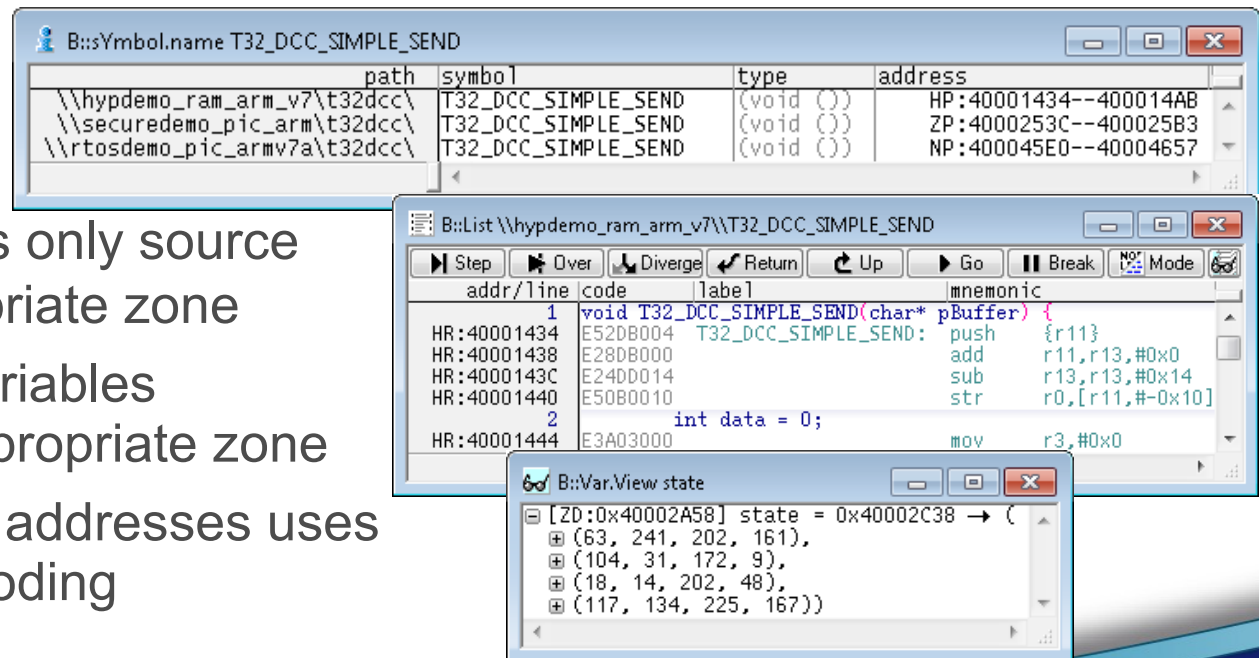
- **CAUTION:**
 - Each zone (non-secure, secure, hypervisor) has an own MMU translation!
 - If the translation is different for each zone, symbols may not match!
 - Software breakpoints may be set into wrong code!
 - Onchip breakpoints may halt where not desired

Agenda

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

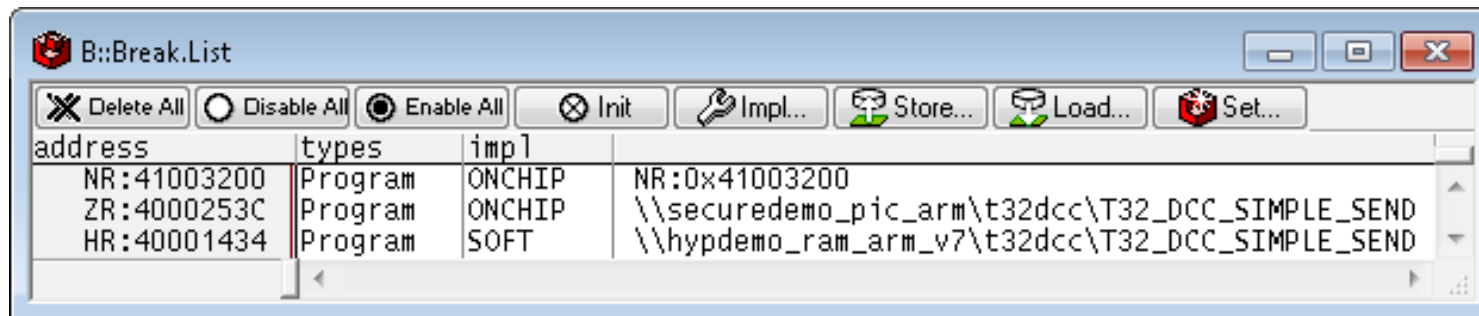
Special TrustZone Support

- 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

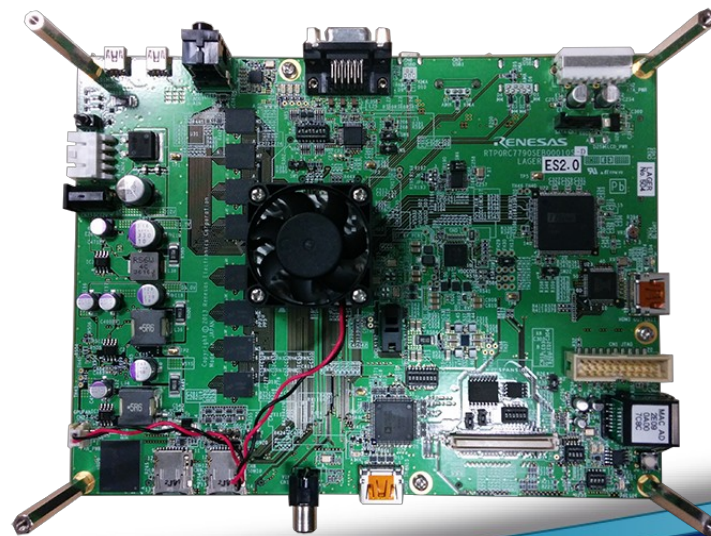
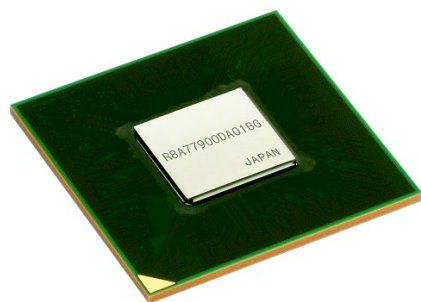


Agenda

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

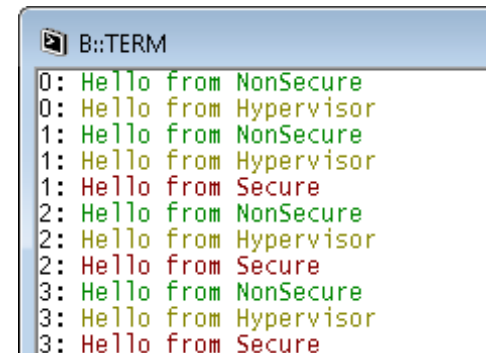
Debugging Through The Zones

- 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:



```
B::TERM
0: Hello from NonSecure
0: Hello from Hypervisor
1: Hello from NonSecure
1: Hello from Hypervisor
1: Hello from Secure
2: Hello from NonSecure
2: Hello from Hypervisor
2: Hello from Secure
3: Hello from NonSecure
3: Hello from Hypervisor
3: Hello from Secure
```


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 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--A00FFFFFFF					
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--F0FFFFFFF					
I:00:F1000000--F1007FFF	A:00:F1000000--F1007FFF	ns		00001000	read/write
I:00:F1008000--FFFFFFF					

B::MMU.List HypPageTable

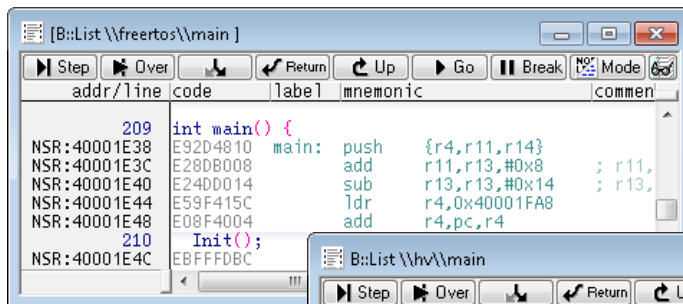
address	physical	sec	d	size	permissions
H:00000000--3FFFFFFF					
H:40000000--40005FFF	AH:00:83080000--83085FFF	ns		00001000	P:readwrite
H:40006000--6663FFFF					
H:66640000--66646FFF	AH:00:83080000--83086FFF	ns		00001000	P:readwrite
H:66647000--FFFFFFF					

B::MMU.List SecPageTable

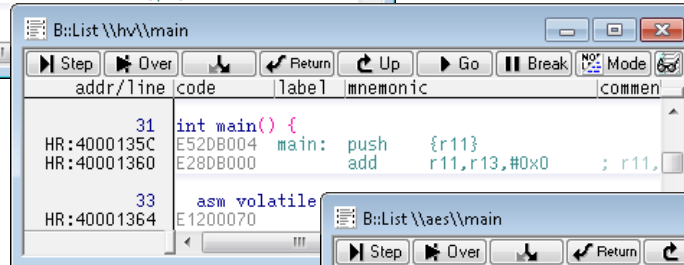
address	physical	sec	d	size	permissions
Z:00000000--3FFFFFFF					
Z:40000000--40005FFF	AZ:00:84000000--84005FFF	s		00001000	P:readwrite
Z:40006000--6663FFFF					
Z:66640000--66646FFF	AZ:00:84000000--84006FFF	s		00001000	P:readwrite
Z:66647000--FCFFFFFFF					
Z:FD000000--FD005FFF	AZ:00:85500000--85505FFF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

Debugging Through The Zones

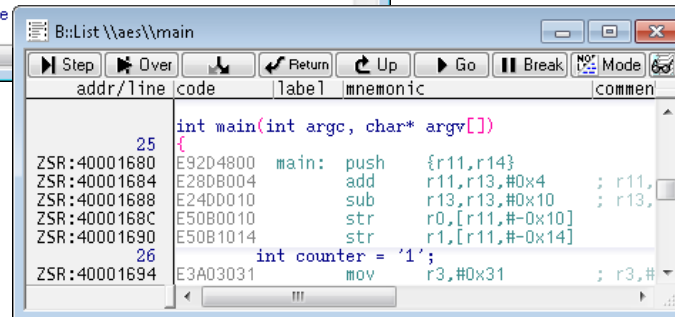
- Accessing symbols
 - Different set of symbols for each zone
 - E.g.: each zone has its own “main”



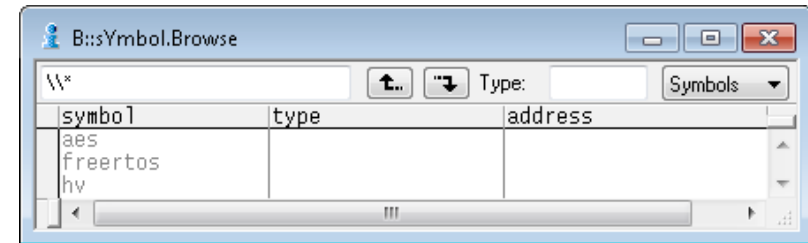
```
[B::List \\freertos\\main ]
Step Over Return Up Go Break Mode
addr/line code label mnemonic comment
NSR:40001E38 209 int main() {
NSR:40001E38 E92D4810 main: push {r4,r11,r14}
NSR:40001E3C E28DB008 add r11,r13,#0x8 ; r11,
NSR:40001E40 E24DD014 sub r13,r13,#0x14 ; r13,
NSR:40001E44 E59F415C ldr r4,0x40001FA8
NSR:40001E48 E08F4004 add r4,pc,r4
NSR:40001E4C 210 Init();
NSR:40001E4C EBFFFDBC
```



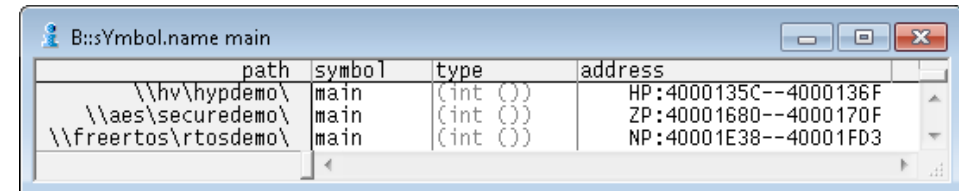
```
B::List \\hv\\main
Step Over Return Up Go Break Mode
addr/line code label mnemonic comment
HR:4000135C 31 int main() {
HR:4000135C E52DB004 main: push {r11}
HR:40001360 E28DB000 add r11,r13,#0x0 ; r11,
HR:40001364 33 asm volatile
HR:40001364 E1200070
```



```
B::List \\aes\\main
Step Over Return Up Go Break Mode
addr/line code label mnemonic comment
ZSR:40001680 25 int main(int argc, char* argv[])
ZSR:40001680 E92D4800 main: push {r11,r14}
ZSR:40001684 E28DB004 add r11,r13,#0x4 ; r11,
ZSR:40001688 E24DD010 sub r13,r13,#0x10 ; r13,
ZSR:4000168C E5080010 str r0,[r11,#-0x10]
ZSR:40001690 E5081014 str r1,[r11,#-0x14]
ZSR:40001694 26 int counter = '1';
ZSR:40001694 E3A03031 mov r3,#0x31 ; r3,#
```



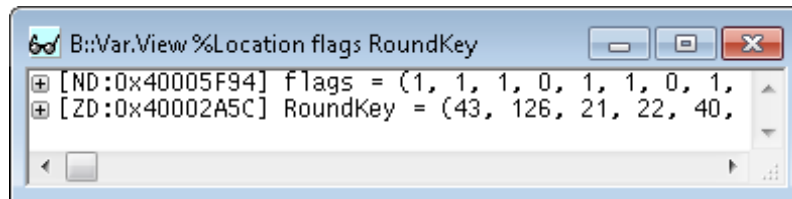
symbol	type	address
aes		
freertos		
hv		



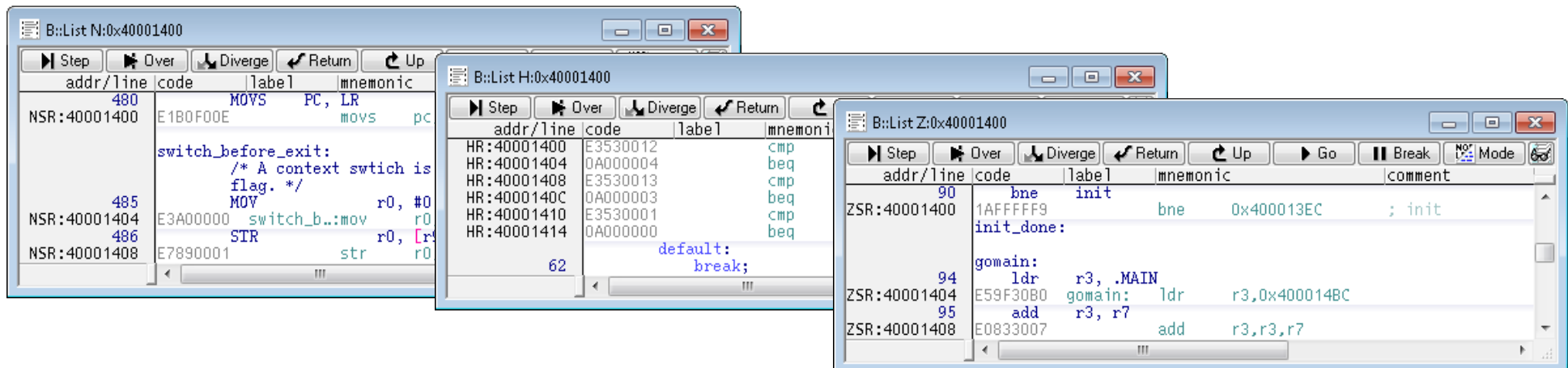
path	symbol	type	address
\\hv\\hypdemo\\	main	(int ())	HP:4000135C--4000136F
\\aes\\securedemo\\	main	(int ())	ZP:40001680--4000170F
\\freertos\\rtosdemo\\	main	(int ())	NP:40001E38--40001FD3

Debugging Through The Zones

- Accessing symbols
 - Variables automatically point to the correct zone

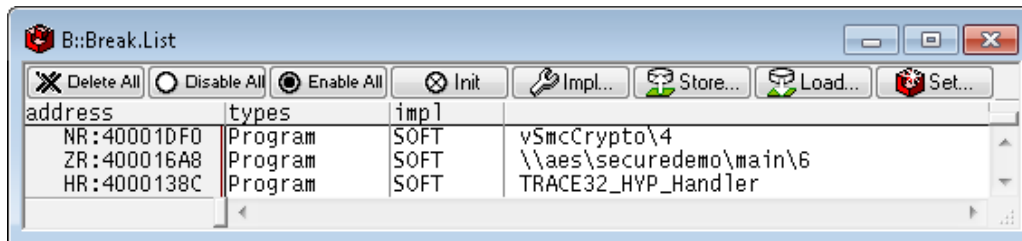


- Different code and different source for each zone

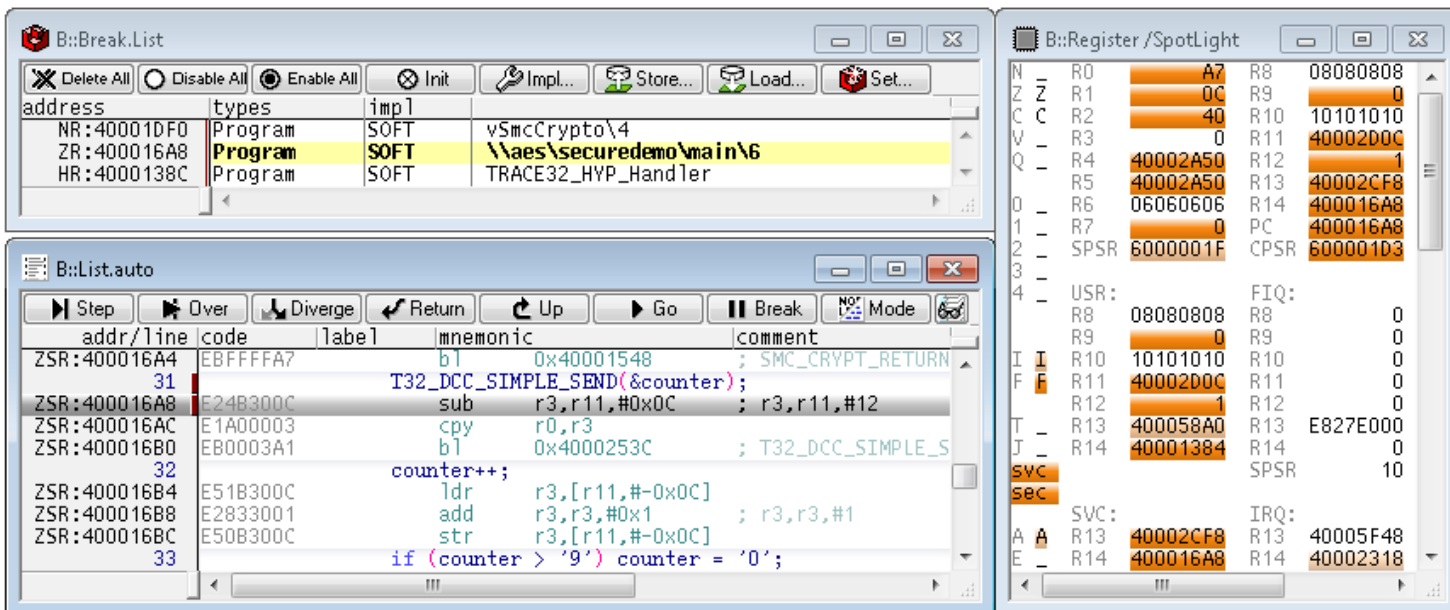


Debugging Through The Zones

- 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

[B::List]

addr/line	code	label	mnemonic	comment
201			if (counter > '9') counter = '0';	
NSR:40001E08	E51B3008		ldr r3,[r11,#-0x8]	
NSR:40001E0C	E3530039		cmp r3,#0x39	; r3,#57
NSR:40001E10	DA000001		bte 0x40001E1C	
NSR:40001E14	E3A03030		mov r3,#0x30	; r3,#48
NSR:40001E18	E50B3008		str r3,[r11,#-0x8]	
202			T32_DCC_SIMPLE_SEND(":\x1B[32mHello from NonSecure	
NSR:40001E1C	E59F3010		ldr r3,0x40001E34	
NSR:40001E20	E08F3003		add r3,pc,r3	
NSR:40001E24	E1A00003		cpy r0,r3	
NSR:40001E28	E0009EC		b1 0x400045E0	; T32_DCC_SIMPLE_SEND
203			SMC_CRYPT_CALL();	
NSR:40001E2C	FFFFDAF		b1 0x400014F0	; SMC_CRYPT_CALL
204				
NSR:40001E30	FAFFFEFC		b 0x40001DE8	
NSP:40001E34	00002DE0		dcd 0x2DE0	

B::Register/SpotLight

N	Z	C	V	Q	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	SPSR	CPSR
R0	40004C08	R8	08080808																			
R1	4C	R9	40001030																			
R2	F1002004	R10	10101010																			
R3	0	R11	400058F4																			
R4	04040404	R12	12121212																			
R5	05050505	R13	400058E0																			
R6	06060606	R14	40001E2C																			
R7	07070707	PC	40001E2C																			
SPSR		CPSR	6000001F																			

- Non-secure executes “smc #0”

[B::List]

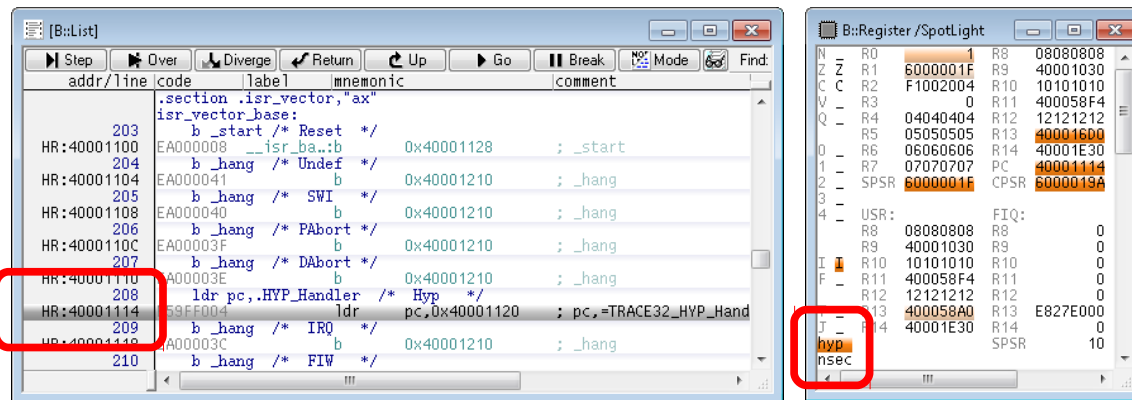
addr/line	code	label	mnemonic	comment
5			SMC_CRYPT_CALL:	
NSR:400014F0	E92D5FFF	SMC_CRYPT_CALL:push	{r0-r12,r14}	
6			mrs r0,cpsr	
NSR:400014F4	E10F0000		mrs r0,cpsr	
7			mrs r1,spsr	
NSR:400014F8	E14F1000		mrs r1,spsr	
8			push {r0-r1}	
NSR:400014FC	E92D0003		push {r0-r1}	
9			mov r0,#0x1	; r0,#1
NSR:40001500	E3A00001		mov r0,#0x1	
10			smc #0	
NSR:40001504	E1600070		smc #0x0	
11			pop {r0-r1}	
NSR:40001508	E8BD0003		pop {r0-r1}	

B::Register/SpotLight

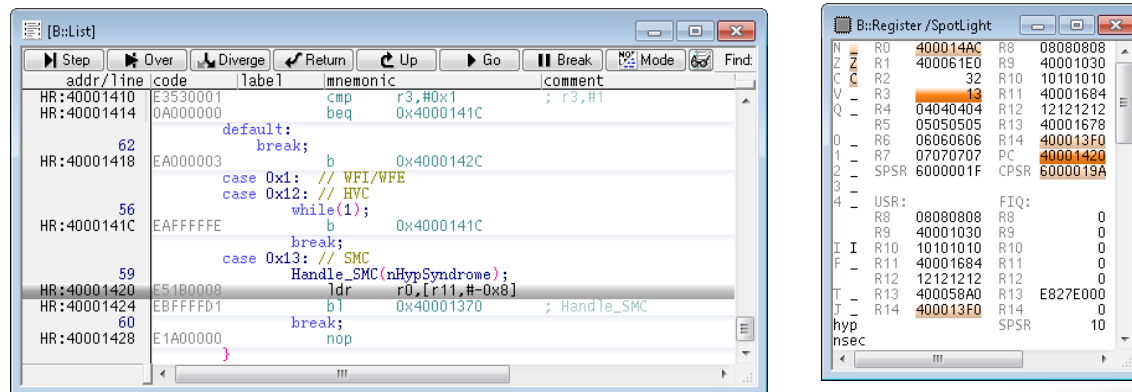
N	Z	C	V	Q	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	SPSR	CPSR
R0	6000001F	R8	08080808																			
R1	4C	R9	40001030																			
R2	F1002004	R10	10101010																			
R3	0	R11	400058F4																			
R4	04040404	R12	12121212																			
R5	05050505	R13	400058E0																			
R6	06060606	R14	40001E30																			
R7	07070707	PC	40001504																			
SPSR		CPSR	6000001F																			

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”

[B::List] window showing assembly code for Handle_SMC_CRYPTO_ASM. The instruction at address 4000130C is 'smc #0' (E1600070), which is highlighted. The [B::Register/SpotLight] window on the right shows the current state of registers, with R0 containing 00000000 and CPSR containing 6000019A.

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

[B::List] window showing assembly code for the .section .isr_vector, "ax" block. The instruction at address 40001208 is 'ldr pc, MON_Handler' (E9FF0100), which is highlighted with a red box. The [B::Register/SpotLight] window on the right shows the current state of registers, with R0 containing 00000000 and CPSR containing 6000019A. The 'MON' and 'nsec' fields in the register window are also highlighted with a red box.

Debugging Through The Zones

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

The screenshot shows the debugger interface with two windows. The left window, titled "[B::List]", displays assembly code for the ZSR memory region. The right window, titled "B::Register /SpotLight", shows the current register values.

Assembly Code (ZSR:40001254 - ZSR:40001278):

addr/line	code	label	mnemonic	comment
18	EAF FFFF	b 2f		
19	2: EAF FFFF		eor r0,r0,#1	0x40001258
20	E22 0000		eor r0,r0,#0x1	; r0,r0,#1
21	EE0 10F1		mcr p15,0x0,r0,c1,c1,0x0	; p15,0,r0,c1,c1
22	E8B D000		pop {r0,r1}	
23	MSR sp_svc,r0		msr sp_svc,r0	
24	E12 3F30		pop {r0,r1}	
25	E8B D000		pop {r0,r1}	
26	E8B D000		rfeia r13!	
27	E8B D000		rfe r13!	
28	E8B D000		__exidx__	0xA3C7F9
29	E8B D000		vstrlt d12,0x400013F8	
30	E8B D000		stc1s p7,c1,[r14,#0x3E8]!	

Register Values (B::Register /SpotLight):

Register	Value	Register	Value
R0	00000001	R8	08080808
R1	6000001F	R9	40001030
R2	32	R10	10101010
R3	13	R11	40001674
R4	04040404	R12	12121212
R5	05050505	R13	40001200
R6	06060606	R14	40001310
R7	07070707	PC	40001294
SPSR	6000019A	CPSR	200001D6

- Step into secure zone

The screenshot shows the debugger interface with two windows. The left window, titled "[B::List]", displays assembly code for the ZSP memory region. The right window, titled "B::Register /SpotLight", shows the current register values.

Assembly Code (ZSP:4000155C - ZSP:40001570):

addr/line	code	label	mnemonic	comment
11	E16 0000		word 0xE1600070	
12	E8B D000		pop {r0-r1}	
13	MSR spsr,r1		msr spsr,r1	
14	MSR cpsr,r0		msr cpsr,r0	
15	E12 9F00		pop {r0-r12,r14}	
16	E8B D000		pop {r0-r12,r14}	
17	BX r14		bx r14	
18	E12 FFF1		bx r14	

Register Values (B::Register /SpotLight):

Register	Value	Register	Value
R0	00000001	R8	08080808
R1	6000001F	R9	40001030
R2	32	R10	10101010
R3	13	R11	40001674
R4	04040404	R12	12121212
R5	05050505	R13	40002C88
R6	06060606	R14	40003678
R7	07070707	PC	40001560
SPSR	6000001F	CPSR	600001D3

Debugging Through The Zones

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

[B::List]

addr/line	code	label	mnemonic	comment
26			int counter = '1';	
ZSR:40001694	E3A03031		mov r3,#0x31	; r3,#49
ZSR:40001698	E50B300C		str r3,[r11,#-0x0C]	
27			int mode = 0;	
ZSR:4000169C	E3A03000		mov r3,#0x0	; r3,#0
ZSR:400016A0	E50B3008		str r3,[r11,#-0x08]	
30			while(1)	
			{	
ZSR:400016A4	EBFFFA7		SMC_CRYPT_RETURN();	
			b1 0x40001548	; SMC_CRYPT_RETURN
ZSR:400016A8	E24B300C		T32_DCC_SIMPLE_SEND(&counter);	
ZSR:400016AC	E1A00003		sub r3,r11,#0x0C	; r3,r11,#12
ZSR:400016B0	EB0003A1		cpy r0,r3	; T32_DCC_SIMPLE_SEND
			b1 0x4000253C	
ZSR:400016B4	E51B300C		counter++;	
ZSR:400016B8	E2833001		ldr r3,[r11,#-0x0C]	
			add r3,r3,#0x1	; r3,r3,#1

B::Register/SpotLight

N	Z	C	V	Q	R0	R1	R2	R3	R4	R5	R6	R7	SPSR	R8	R9	R10	R11	R12	R13	R14	PC	FIQ	CPSR
					08080808	0C	40	0	40002A50	40002A50	06060606	0	6000001F	08080808	0	10101010	40002D0C	1	40002CF8	40002A5C	400016A4	600001D3	

- Secure executes “smc #0”

[B::List]

addr/line	code	label	mnemonic	comment
5			SMC_CRYPT_RETURN:	
ZSR:40001548	E92D5FFF	SMC_CRYPT::push	{r0-r12,r14}	
6			mrs r0,cpsr	
ZSR:4000154C	E10F0000		mrs r0,cpsr	
7			mrs r1,spsr	
ZSR:40001550	E14F1000		mrs r1,spsr	
8			push {r0-r1}	
ZSR:40001554	E92D0003		push {r0-r1}	
9			mov r0,#0x1	
ZSR:40001558	E3A00001		mov r0,#0x1	; r0,#1
10			smc #0	
ZSR:4000155C	E1600070		smc #0x0	
11			pop {r0-r1}	
ZSR:40001560	E8BD0003		pop {r0-r1}	
12			msr spsr,r1	
ZSR:40001564	E169F001		msr spsr,r1	

B::Register/SpotLight

N	Z	C	V	Q	R0	R1	R2	R3	R4	R5	R6	R7	SPSR	R8	R9	R10	R11	R12	R13	R14	PC	FIQ	CPSR
					6000001F	40	0	0	40002A50	40002A50	06060606	0	6000001F	08080808	0	10101010	40002D0C	1	40002CF8	400016A8	4000155C	600001D3	

Debugging Through The Zones

- Stepping through the mode changes
 - Step into monitor mode

The [B::List] window shows the following assembly code:

```

28  .section .isr_vector, "ax"
29  isr_vector_base:
30  b_start /* Reset */
31  ZSR:40001200 EA000008 __isr_base: 0x40001228 ; _hang
32  b_hang /* Undef */
33  ZSR:40001204 A0000007 b 0x40001228 ; _hang
34  ldr pc, .MON_Handler /* SWI */
35  ZSR:40001208 E9FF0110 ldr pc, 0x40001220
36  b_hang /* PAbort */
37  ZSR:4000120C A0000005 b 0x40001228 ; _hang
38  b_hang /* DAbort */
39  ZSR:40001210 EA000004 b 0x40001228 ; _hang
40  b_hang /* Hyp */
41  ZSR:40001214 EA000003 b 0x40001228 ; _hang
42  b_hang /* IRQ */
43  ZSR:40001218 EA000002 b 0x40001228 ; _hang
44  b_hang /* FIQ */
45  ZSR:4000121C A0000001 b 0x40001228 ; _hang

```

The B::Register/SpotLight window shows the following register values:

Register	Value
R0	08080808
R1	6000001F
R2	40
R3	0
R4	40002A50
R5	40002A50
R6	06060606
R7	0
SPSR	60000103
CPSR	60000106
PC	40001208

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

The [B::List] window shows the following assembly code:

```

17  /*from SECURE to NON-SECURE*/
18  ZSR:40001250 E22D0020 eor r13, r13, #0x20 ; r13, r13, #32
19  b 2f
20  ZSR:40001254 EAF00000 b 0x40001258
21  eor r0, r0, #1
22  ZSR:40001258 E2200001 eor r0, r0, #0x1 ; r0, r0, #1
23  mcr p15, 0x0, r0, c1, c1, 0x0
24  ZSR:4000125C EE010F11 mcr p15, 0x0, r0, c1, c1, 0x0 ; p15, 0, r0, c1, c1
25  pop {r0, r1}
26  ZSR:40001260 E8BD0003 pop {r0-r1}
27  mcr sp_svc, r0
28  ZSR:40001264 E123F300 mcr sp_svc, r0
29  pop {r0, r1}
30  ZSR:40001268 E8BD0003 pop {r0-r1}
31  rfeia r13!
32  ZSR:4000126C E8BD0003 rfe r13!

```

The B::Register/SpotLight window shows the following register values:

Register	Value
R0	08080808
R1	6000001F
R2	40
R3	0
R4	40002A50
R5	40002A50
R6	06060606
R7	0
SPSR	60000103
CPSR	60000106
PC	40001224

Debugging Through The Zones

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

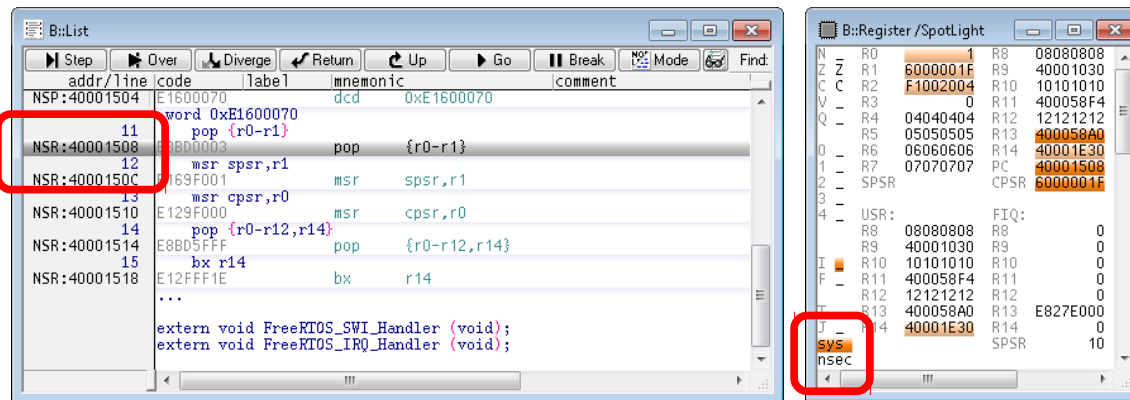
The B::List window shows assembly code with a red box highlighting the instruction at address 40001310: `pop {r0-r1}`. The B::Register/SpotLight window shows the current state of registers, with a red box highlighting the `hyp` and `nsec` fields in the CPSR register, indicating the processor is in Hypervisor mode.

- Hypervisor executes “eret”

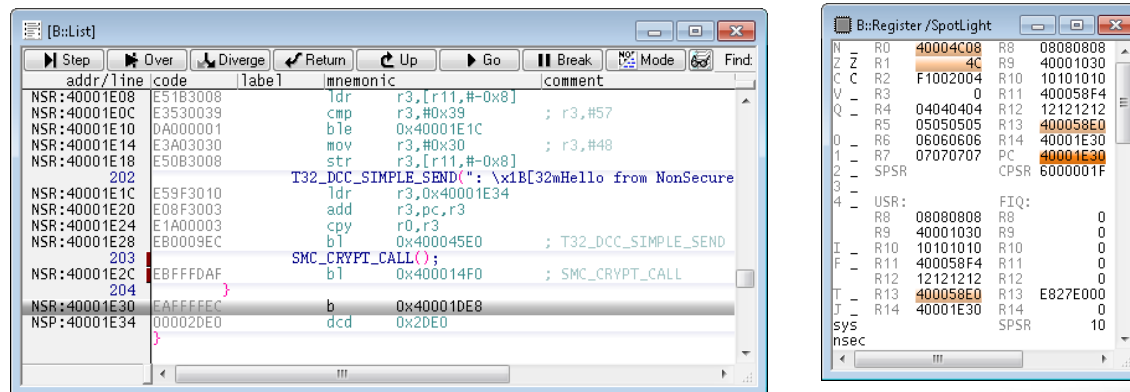
The B::List window shows the execution of the `eret` instruction at address 400012F4. The B::Register/SpotLight window shows the updated state of registers, with the `hyp` field in the CPSR register now set to 1, indicating the processor is in Hypervisor mode.

Debugging Through The Zones

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

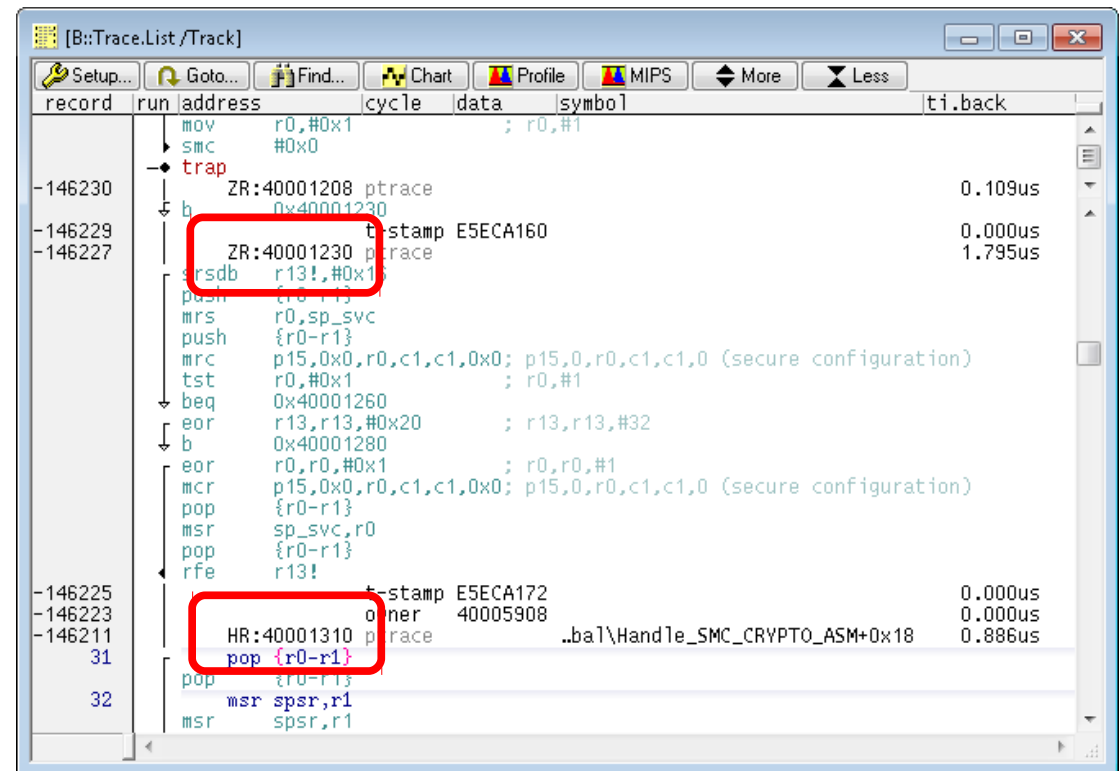


- Return from subroutine, and we're back where we started!



Debugging Through The Zones

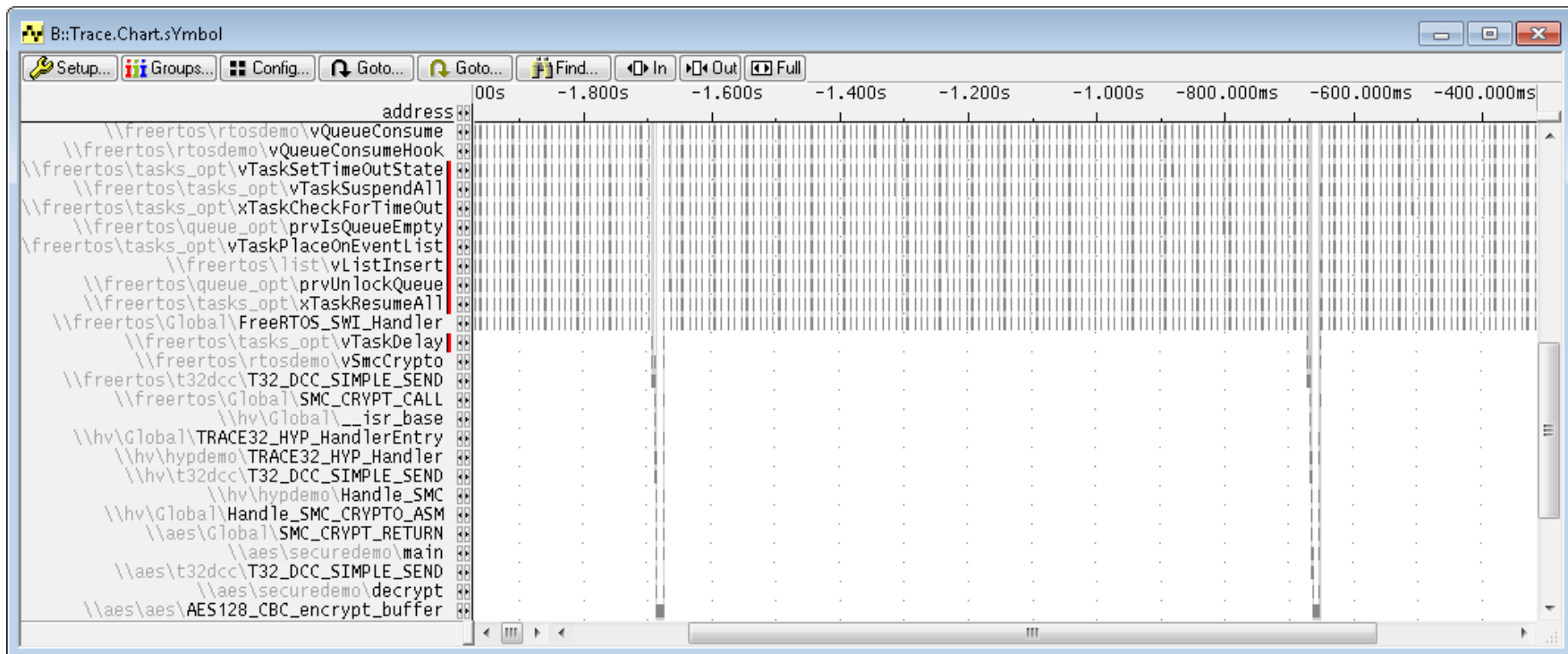
- Tracing the zones
 - The ETM stream contains information about which zone is active
 - TRACE32 detects the zone switches and adjusts the access classes to the addresses in the trace.



record	run	address	cycle	data	symbol
-146230		mov r0,#0x1			
-146229		smc #0x0			
-146227		trap			
-146230		ZR:40001208		ptrace	0.109us
-146229		b		0x40001230	0.000us
-146227		ZR:40001230		ptrace	1.795us
-146225		rsdb r13!,#0x1			
-146223		push {r0-r1}			
-146211		mrs r0,sp_svc			
-146211		push {r0-r1}			
-146211		mrc p15,0x0,r0,c1,c1,0x0; p15,0,r0,c1,c1,0 (secure configuration)			
-146211		tst r0,#0x1			
-146211		beq 0x40001260			
-146211		eor r13,r13,#0x20			
-146211		b 0x40001280			
-146211		eor r0,r0,#0x1			
-146211		mcr p15,0x0,r0,c1,c1,0x0; p15,0,r0,c1,c1,0 (secure configuration)			
-146211		pop {r0-r1}			
-146211		msr sp_svc,r0			
-146211		pop {r0-r1}			
-146211		rfe r13!			
-146225		tstamp E5ECA172			0.000us
-146223		owner 40005908			0.000us
-146211		HR:40001310		ptrace	0.886us
-146211		pop {r0-r1}			
-146211		pop {r0-r1}			
-146211		msr spsr,r1			
-146211		msr spsr,r1			

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)

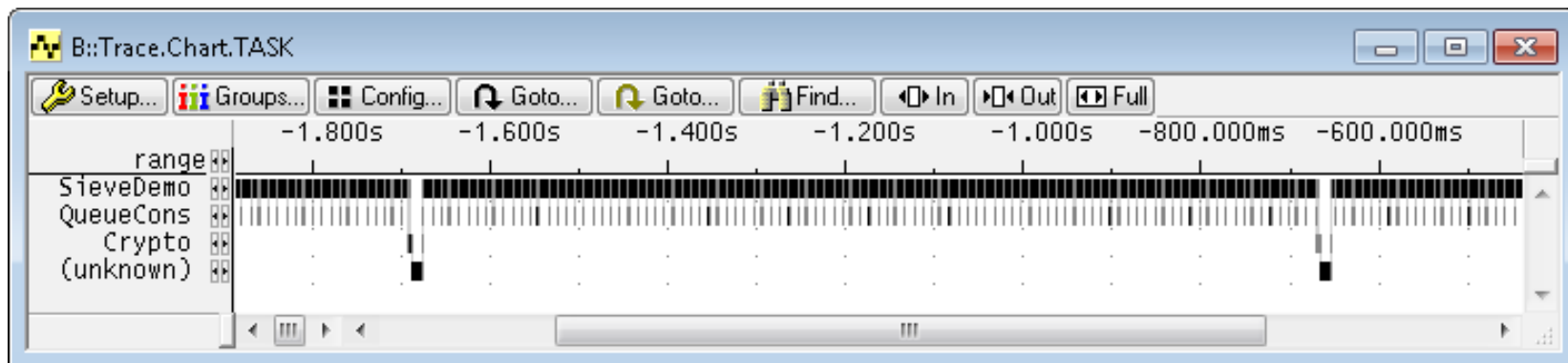
B::Trace.Statistic.TREE TASK DEFAULT

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

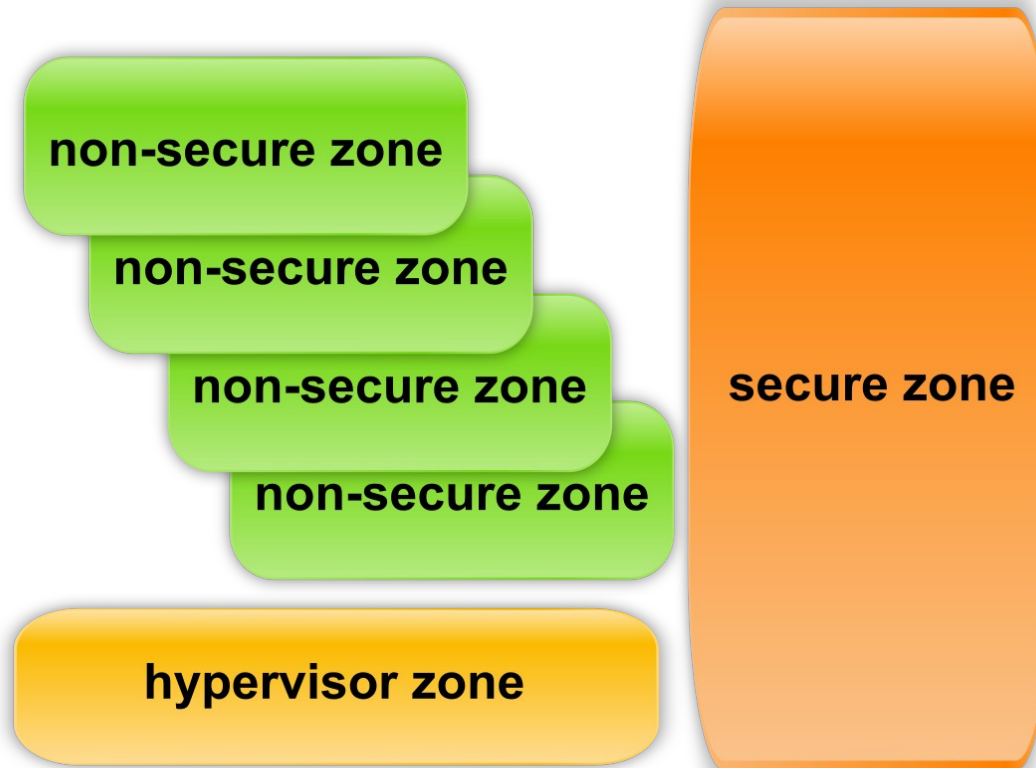


Agenda

- 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?