

# ARM TrustZone

Alex Merkle

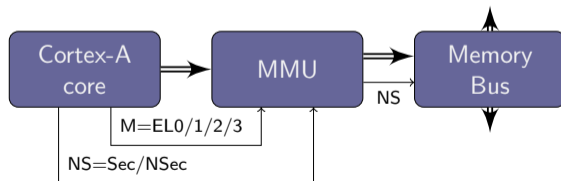
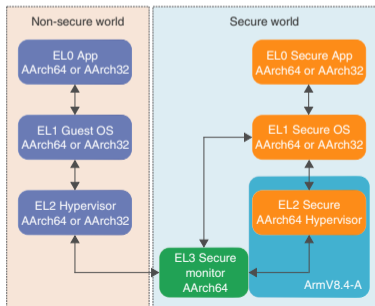
November 26, 2019

# Overview

- 1 TrustZone Basics
- 2 TrustZone Cortex-A
- 3 TrustZone Cortex-M
- 4 Usage Cortex-A
- 5 Example Cortex-A
- 6 Usage Cortex-M

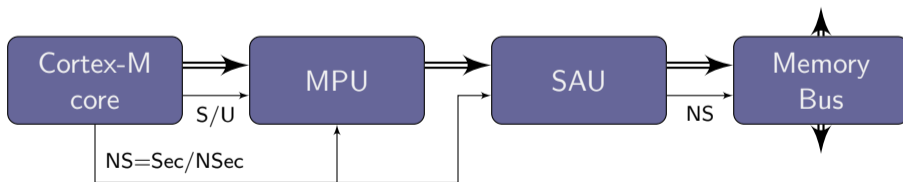
# TrustZone Cortex-A

- Cortex-A cores feature multiple *Exception Level* EL0/1/2/3  
EL0 = User, EL1 = Supervisor, EL2 = Hypervisor, EL3 = Monitor
- Additionally there is a *Security State* Secure/NonSecure
- Both *Exception Level* and *Security State* control the MMU
- *Security State* is also connected to the SoC interconnect



# TrustZone Cortex-M

- Cortex-M cores feature Handler and Thread mode  
Handler = Supervisor , Thread = User/NonPrivileged
- Additionally there is a *Security State* Secure/NonSecure
- *Security State* controls the SAU
- *Security State* is also connected to the SoC interconnect

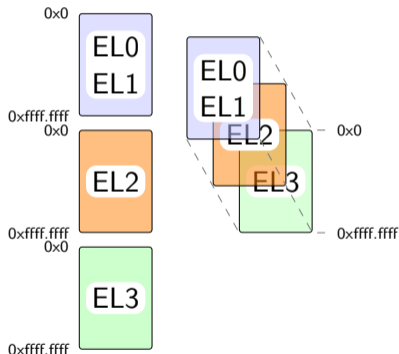


# Overview

- 1 TrustZone Basics
- 2 TrustZone Cortex-A**
- 3 TrustZone Cortex-M
- 4 Usage Cortex-A
- 5 Example Cortex-A
- 6 Usage Cortex-M

# Address Ambiguity

- As the *Exception Level & Security State* are input signal for the MMU there exists for every combination a own virtual address map
- Every *Exception Level & Security State* combination has a own Access-Class within TRACE32
- Reason: Address-Values are ambiguous!



# TRACE32 Access Classes

- TRACE32 Access Classes allow to access virtual memory of a specific mode
- TRACE32 combines the specified access class with the current CPU-Mode if the address is not fully qualified

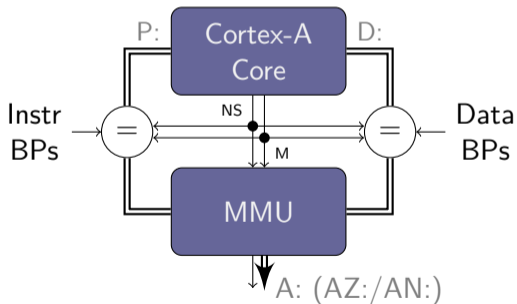
Mode (M)	TrustZone (NS)	
	NonSecure	Secure
EL0/User	NU:<addr>	ZU:<addr>
EL1/Supervisor	NS:<addr>	ZS:<addr>
EL2/Hypervisor	H:<addr> <sup>1</sup>	ZH:<addr> <sup>2</sup>
EL3/Monitor	M:<addr>	

<sup>1</sup>H: is a an alias for NH:

<sup>2</sup>starting from ARMv8.4

# Onchip Breakpoints

- Onchip Breakpoints can trigger on *Security State* Secure/NonSecure
- Onchip Breakpoints can trigger on *Exception Level* EL0+1/EL2/EL3
- No Onchip Breakpoints to physical memory

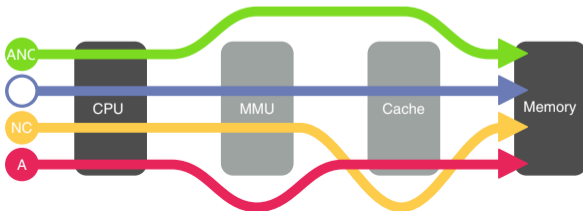
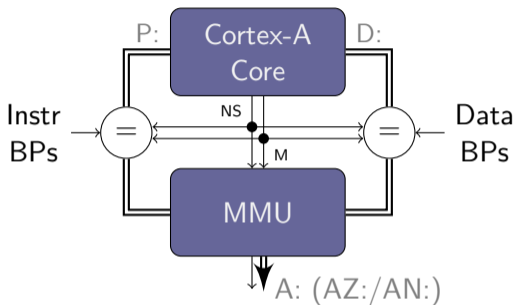




# Physical Memory

Access Class A: can be combined with *Security State* as it's connected to the SoC interconnect

AZ: <-> AN:



# TRACE32 behavior

- *Security State & Exception Level* combination is called *ZONE* in TRACE32
- Default - `SYSTEM.Option ZONESPACES OFF`
  - `sYmbol Database` is not strictly separated/symbol addresses may be ambiguous
- `SYSTEM.Option ZONESPACES ON`
  - `sYmbol Database` is strictly separated
- Default - `Break.CONFIG.MatchZONE OFF`
  - Onchip Breakpoints ignore *Security State* as well as *Exception Level*
- `Break.CONFIG.MatchZONE ON`
  - Onchip Breakpoints respect *Security State* as well as *Exception Level*

# TRACE32 behavior

## Configuration

```
SYStem.Option ZONESPACES ON
```

```
Break.CONFIG.MatchZone ON
```

```
Break.Set <Zone>:<Address> /Onchip
```

ON

MatchZone

OFF

```
Break.Set <Zone>:<Address> /Onchip
```

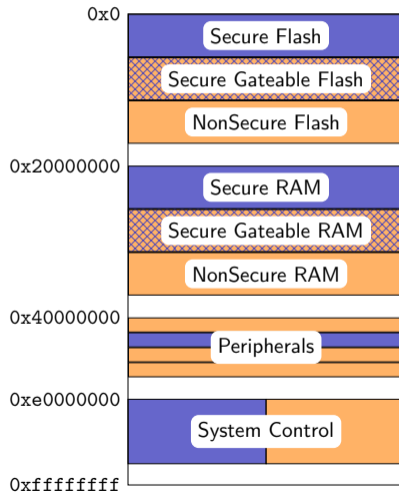
IGNORE

# Overview

- 1 TrustZone Basics
- 2 TrustZone Cortex-A
- 3 TrustZone Cortex-M**
- 4 Usage Cortex-A
- 5 Example Cortex-A
- 6 Usage Cortex-M

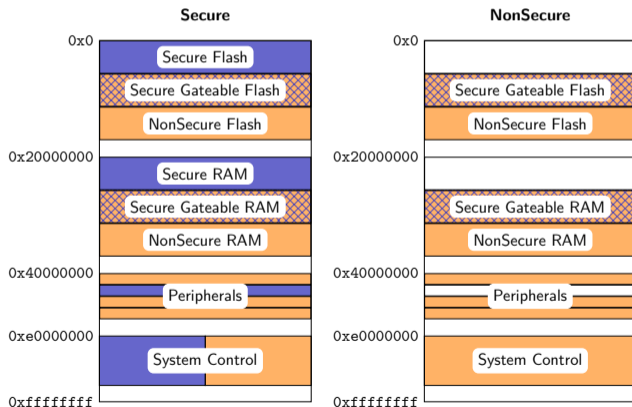
# Memory Map

- For Cortex-M addresses are non-ambiguous
- The *Security State* is a input signal for the SAU
- The SAU is software configurable and allows to restrict accesses into memory when the core is in *NonSecure* state
- Secure Gateable region contains sg instructions that allows *NonSecure* code to call *Secure* routines



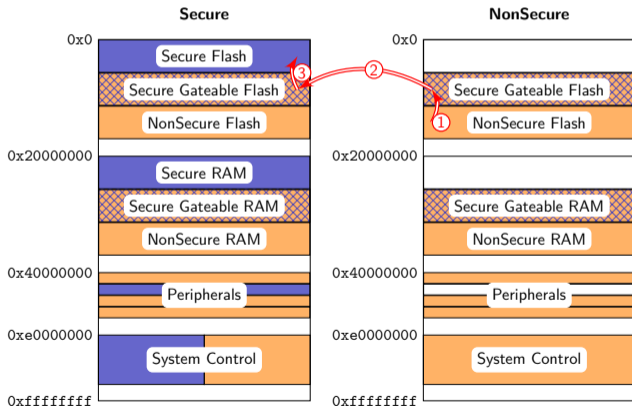
# Memory Map

- The SAU works like a blinker/blinder for NonSecure



# NonSecure to Secure switch

- Secure Gateable Region is used as a veneer



# TRACE32 Access Classes

- TRACE32 Access Classes allow to access memory of a specific *Security State*
- TRACE32 combines the specified access class with the current CPU-Mode if the address is not fully qualified

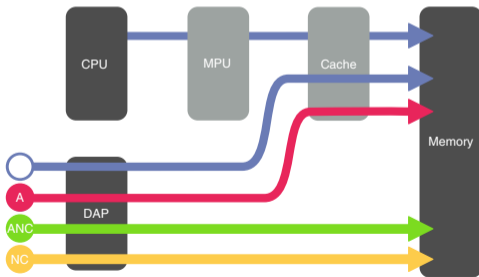
Mode (M)	TrustZone (NS)	
	NonSecure	Secure
EL0/User	NU:<addr> <sup>1</sup>	ZU:<addr> <sup>1</sup>
EL1/Supervisor	NS:<addr>	ZS:<addr>

<sup>1</sup>Cortex-M debugger accesses bypass MPU => U: is ignored



# Physical Memory

- Access Class A: can be combined with *Security State* as it's connected to the SoC interconnect  
AZ: <-> AN:  
AZ: <-> AN:
- Addresses for Cortex-M are always physical



# Overview

- 1 TrustZone Basics
- 2 TrustZone Cortex-A
- 3 TrustZone Cortex-M
- 4 Usage Cortex-A**
- 5 Example Cortex-A
- 6 Usage Cortex-M

# Symbols

```

SYSTEM.Option ZONESPACES ON
Break.CONFIG.MatchZone  ON
; -----
; load sYmbols
Data.LOAD.Elf <code_on_el3.elf>    M: /NoCODE
Data.LOAD.Elf <code_on_el2.elf>    NH: /NoCODE
Data.LOAD.Elf <code_on_el1_ns.elf> NS: /NoCODE
Data.LOAD.Elf <code_on_el1_s.elf>  ZS: /NoCODE

```

Different code and different sourcecode & memory for each zone

The image shows three overlapping windows of a debugger displaying assembly code for different zones:

- NSR:40001400:** Shows assembly code for the NSR zone, including instructions like `MOVS PC, LR` and `MOV r0, #0`. A comment indicates a context switch.
- HR:40001400:** Shows assembly code for the HR zone, including instructions like `cmp r3, #0x12` and `beq 0x4000141C`.
- ZSR:40001400:** Shows assembly code for the ZSR zone, including instructions like `bne init` and `ldr r3, .MAIN`.

# sYmbols

Functions & Variables are automatically accessed with the correct zone

```

B::Var.View %Location flags RoundKey
+ [ND:0x40005F94] fflags = (1, 1, 1, 0, 1, 1, 0, 1,
+ [ZD:0x40002A5C] RoundKey = (43, 126, 21, 22, 40,
  
```

```

B::Symbol.name main
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
  
```

```

[B::List \freertos\main]
Step Over Return Up Go Break Mode
addr/line code label mnemonic comment
209 int main() {
NSR:40001E38 E92D4810 main: push {r4,
NSR:40001E3C E28DB008 add r11,
NSR:40001E40 E24DD014 sub r13,
NSR:40001E44 E59F415C ldr r4,
NSR:40001E48 E08F4004 add r4,
210 Init();
NSR:40001E4C EBFFFDBC bl 0x40
  
```

```

B::List \hv\main
Step Over Return Up Go Break Mode
addr/line code label mnemonic comment
31 int main() {
HR:4000135C E52DB004 main: push {r1
HR:40001360 E28DB000 add r11,
33 asm volatile ("bkpt #0; nop
HR:40001364 E1200070 bkpt #0x0
  
```

```

B::List \aes\main
Step Over Return Up Go Break Mode
addr/line code label mnemonic comment
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 E50B0010 str r0,[r11,#-0x10]
ZSR:40001690 E50B1014 str r1,[r11,#-0x14]
26 int counter = '1';
ZSR:40001694 E3A03031 mov r3,#0x31 ; r3,#
  
```

# Breakpoints

```
Break.CONFIG.MatchZone ON
```

```
; -----
```

```
; set BPs
```

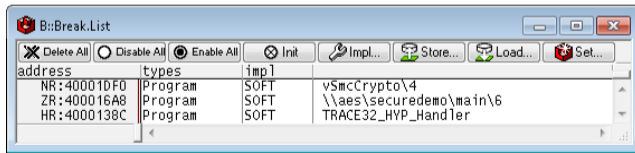
```
Break.Set M:<addr> [/Onchip] </Program|/ReadWrite|/Read|/Write>
```

```
Break.Set H:<addr> [/Onchip] </Program|/ReadWrite|/Read|/Write>
```

```
Break.Set NS:<addr> [/Onchip] </Program|/ReadWrite|/Read|/Write>
```

```
Break.Set ZS:<addr> [/Onchip] </Program|/ReadWrite|/Read|/Write>
```

```
Break.Set <symbol_name> [/Onchip]
```



# Access Memory

```
; -----  
; access memory  
Data.Set    M:<addr> <value>  
Data.Set    H:<addr> <value>  
Data.Set    NS:<addr> <value>  
Data.Set    ZS:<addr> <value>  
Data.dump   M:<addr>  
Data.dump   H:<addr>  
Data.dump   NS:<addr>  
Data.dump   ZS:<addr>  
; -----  
; access physical memory  
Data.Set    A:<addr> <value>  
Data.Set    AZ:<addr> <value>  
Data.Set    AN:<addr> <value>  
Data.dump   A:<addr>  
Data.dump   AZ:<addr>  
Data.dump   AN:<addr>
```

# Peripherals

```

; -----
; access physical memory
Data.Set <A|AZ|AN>:<addr> <value>
Data.dump <A|AZ|AN>:<addr>
; -----
; peripheral view
PER
PER , /NonSecure
PER , /Secure

```

The image shows two screenshots of a debugger window displaying the configuration of the Interrupt Controller (GIC-400) for NonSecure and Secure states.

**Left Screenshot: NonSecure State**

Window Title: B::PER, "Core Registers (Cortex-A53),Interrupt Controller (GIC-400)" /NonSecure

Tree View: Interrupt Controller (GIC-400)

- Distributor Interface
 

GICD_CTLR	00000001	ENABLE	Enabled
GICD_TYPER	0000FC65	LSPPI	31
GICD_IIDR	02001438	CPUNUMBER	4
		PROPID	GIC400
		REV	1
- Group/Security Registers
 

GICD_IGROUPPR0	00000000		
GSR11	Group 0	GSR30	Group 0
GSR28	Group 0	GSR27	Group 0
GSR25	Group 0	GSR24	Group 0
GSR22	Group 0	GSR21	Group 0
GSR19	Group 0	GSR18	Group 0
GSR16	Group 0	GSR15	Group 0
GSR13	Group 0	GSR12	Group 0
GSR10	Group 0	GSR9	Group 0
GSR7	Group 0	GSR6	Group 0
GSR4	Group 0	GSR3	Group 0
GSR1	Group 0	GSR0	Group 0

**Right Screenshot: Secure State**

Window Title: B::PER, "Core Registers (Cortex-A53),Interrupt Controller (GIC-400)" /Secure

Tree View: Interrupt Controller (GIC-400)

- Distributor Interface
 

GICD_CTLR	00000003	ENABLEGRP0	Enabled
GICD_TYPER	0000FC65	LSPPI	31
GICD_IIDR	02001438	CPUNUMBER	4
		PROPID	GIC400
		REV	1
		IMP	043B
- Group/Security Registers
 

GICD_IGROUPPR0	DE0000FF		
GSR31	Group 1 (Non-secure)	GSR30	Group 1 (Non-secure)
GSR28	Group 1 (Non-secure)	GSR27	Group 1 (Non-secure)
GSR25	Group 1 (Non-secure)	GSR24	Group 0 (Secure)
GSR22	Group 0 (Secure)	GSR21	Group 0 (Secure)
GSR19	Group 0 (Secure)	GSR18	Group 0 (Secure)
GSR16	Group 0 (Secure)	GSR15	Group 0 (Secure)
GSR13	Group 0 (Secure)	GSR12	Group 0 (Secure)
GSR10	Group 0 (Secure)	GSR9	Group 0 (Secure)
GSR7	Group 1 (Non-secure)	GSR6	Group 1 (Non-secure)
GSR4	Group 1 (Non-secure)	GSR3	Group 1 (Non-secure)
GSR1	Group 1 (Non-secure)	GSR0	Group 1 (Non-secure)

# Overview

- 1 TrustZone Basics
- 2 TrustZone Cortex-A
- 3 TrustZone Cortex-M
- 4 Usage Cortex-A
- 5 Example Cortex-A**
- 6 Usage Cortex-M



# Debugging ATF

```

; -----
; connect
SYStem.CPU 88F3720-A0
SYStem.Option ZONESPACES ON
Break.CONFIG.MatchZone ON
CORE.ASSIGN 1.
SYStem.Up

; -----
; load Symbols
; Bootloader 1: bl1 running in Monitor/EL3
Data.LOAD.Elf ----/atf/build/a3700/debug/bl1/bl1.elf M: /NoCODE
; Bootloader 2: bl2 running in Secure Supervisor/EL2
Data.LOAD.Elf ----/atf/build/a3700/debug/bl2/bl2.elf Z: /NoCODE /NoCLEAR
; Bootloader 3: bl31 running in Monitor/EL3
Data.LOAD.Elf ----/atf/build/a3700/debug/bl31/bl31.elf M: /NoCODE /NoCLEAR
; Bootloader 4: u-boot running in Hypervisor/EL2
Data.LOAD.Elf ----/u-boot/u-boot H: /NoCODE /NoCLEAR

Go bl1_entrypoint /Onchip /PROGRAM
WAIT !STATE.RUN()

Go bl2_entrypoint /Onchip /PROGRAM
WAIT !STATE.RUN()

Go bl31_entrypoint /Onchip /PROGRAM
WAIT !STATE.RUN()

Go __image_copy_start /Onchip /PROGRAM
WAIT !STATE.RUN() 10s

```

# Debugging ATF - Hints

- Keep the complete code in the bootsource (e.g. Flash / SD-Card)  
⇒ Boot sequence is automatized
- Use /Onchip Breakpoints to trigger at every bootstage entry  
⇒ Possibility to patch the code
- Use  
    TRANSlation.TableWalk ON  
    TRANSlation.ON  
to enable TRACE32 automatic MMU-TableWalk  
⇒ Software Breakpoints within every zone even though the .text segment is read-only (via MMU)

# Debugging ATF - Pitfalls

- Bootflow is SoC specific - check BSP
- To debug into the bootloaders requires to connect the debugger early after reset  
Highly SoC & Board specific!
- Not every SoC supports that!  
⇒ patch a endless loop into the bootimage (last resort)
- TRACE32 is preconfigured for SMP debugging after `SYSTEM.CPU`  
⇒ use `CORE.ASSIGN <x>` to select the bootcore(s) only
- ⇒ example scripts `~/demo/<arch>/hardware/<chip|board>`
- ⇒ [support@lauterbach.com](mailto:support@lauterbach.com)

# Debugging Linux + bl31

```

; -----
; connect
SYStem.CPU 88F3720-A0
SYStem.Option ZONESPACES ON
SYStem.Option MMUSPACES ON
Break.CONFIG.MatchZone ON
SYStem.Mode Attach
Break

; -----
; load Symbols
; bl31 running in Monitor/EL3
Data.LOAD.Elf ----/atf/build/a3700/debug/bl31/bl31.elf M: /NoCODE
; linux running in NonSecure Supervisor/EL1
Data.LOAD.Elf ----/linux/vmlinux NS: /NoCODE /NoCLEAR

; -----
; configure debugger address translation
MMU.FORMAT LINUXSWAP3 \\vmlinux\\swapper_pg_dir NS:<virtual range> <physical>
TRANSLation.COMMON NS:0xf000000000000000--0xffffffffffffffff
TRANSLation.TableWalk ON
TRANSLation.ON

TASK.CONFIG --/demo/arm64/kernel/linux/awareness/linux.t32 /ACCESS NS:
MENU.ReProgram --/demo/arm64/kernel/linux/awareness/linux.men

● Use --/demo/<arm|arm64>/kernel/linux/boards/generic-template/detect_translation.cmm
● Use --/demo/<arm|arm64>/kernel/linux/boards/generic-template/linux-attach.cmm

```

# detect\_translation.cmm

- First official release DVD2019.02
- Supports KAISER/MELTDOWN patch
- Steps
  - Linux must be up and running
  - Debugger is connected  $\Rightarrow$  SYSTEM.Mode Attach
  - Debug symbols must be loaded  $\Rightarrow$  Data.LOAD.Elf ...
  - DO `--/demo/<arm|arm64>/kernel/linux/boards/generic-template/detect_translation.cmm`

```

B::AREA
Virtual Address of swapper_pg_dir: 0xffff00001101d000
Virtual Address of idle_loop: 0xffff000010115fe8
Virtual Address of linux_banner: 0xffff000010af0070
Access Class detected: NS
Access Class used: NS
Try to run to "vmlinux\tree\rcu_idle_enter" ...
Linux Banner "Linux version 5.0.0-rc1 (amerkle@amepc1-vmdebian) (gcc version 6.3.0 20170516 (Debian
"
MMU.FORMAT LINUXSWAP3 vmlinux\swapper_pg_dir NS:0xffff00001101d000+0xffff 0x000000000801d000
TRANSLATION.COMMON NS:0xf000000000000000--0xffffffffffffffff
TRANSLATION.Tablewalk ON
TRANSLATION.ON
TASK.CONFIG --/demo/arm64/kernel/linux/linux-3.x/linux3.t32 /ACCESS NS:
MENU.ReProgram --/demo/arm64/kernel/linux/linux-3.x/linux.men
  
```

- See also: `--/demo/<arm|arm64>/kernel/linux/boards/generic-template/linux-attach.cmm`

# Debugging OP-TEE

```

; -----
; connect
SYStem.CPU ZYNQ-ULTRASCALE+-APU
SYStem.Option ZONESPACES ON
SYStem.Option MMUSPACES ON
Break.CONFIG.MatchZone ON
SYStem.Mode Attach
Break

; -----
; load Symbols
; op-tee running in Secure Supervisor/EL1
Data.LOAD.Elf out/arm-plat-zynqmp/core/tee.elf ZS: /NoCODE

; -----
; configure debugger address translation
TRANSLation.Create ZS:<optee virtual range> AZ:<optee physical>
TRANSLation.COMMON ZS:<optee virtual range>
TRANSLation.TableWalk ON
TRANSLation.ON

TASK.CONFIG    ~/demo/arm64/kernel/op-tee/optee.t32 /ACCESS ZS:
MENU.ReProgram ~/demo/arm64/kernel/op-tee/optee.men

```

# Debugging OP-TEE - Hints

- Op-Tee uses a paging approach with ambiguous TA address space  
⇒ `SYSTEM.Option MMUSPACES ON`
- The Op-Tee default translation needs to be configured manually, by Nov 2019 we do not support `MMU.FORMAT`  
⇒ `TRANSLation.Create`
- Use  
    `TRANSLation.TableWalk ON`  
    `TRANSLation.ON`  
to enable TRACE32 automatic MMU-TableWalk  
⇒ Software Breakpoints are available for TAs despite the `.text` segment is read-only (via MMU)

# Debugging OP-TEE - Limitations

Status November 2019

- The standard workflow foresees that a TA is
  - loaded - `TEEC_OpenSession`
  - debugged - `TEEC_InvokeCommand`
  - unloaded - `TEEC_CloseSession`

triggered by the optee menu *Debug Trusted Application from Entry*.

- For TAs that are loaded persistently (no `TEEC_CloseSession`), possibly ambiguous Onchip-Breakpoints need to be used



# Overview

- 1 TrustZone Basics
- 2 TrustZone Cortex-A
- 3 TrustZone Cortex-M
- 4 Usage Cortex-A
- 5 Example Cortex-A
- 6 Usage Cortex-M**

# Symbols

```

; -----
; load sSymbols
Data.LOAD.Elf <secure.elf>      Z: /NoCODE
Data.LOAD.Elf <non_secure.elf> N: /NoCODE /NoCLEAR

```

Functions & Variables are automatically accessed with the correct zone

The image displays three debugger windows illustrating symbol resolution across different memory zones:

- [B::List\secure\main]**: Shows assembly for the `main` function in the secure zone. The code includes a `monHook` function and the `main` function body. Address `109` is highlighted.
- [B::List\sieve\main]**: Shows assembly for the `main` function in the sieve zone. The code includes a `monHook` function and the `main` function body. Address `699` is highlighted.
- B::Var.Watch**: Shows a list of watched variables:
  - `[ND:0x20004014] plot1 = 0`
  - `[ND:0x20004074] plot2 = 0`
  - `[ZD:0x2000000C] nPwm25 = 0`
  - `[ZD:0x2000000E] nPwm64 = 0`

# Access Memory

; -----

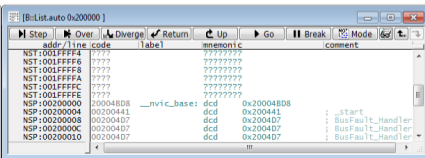
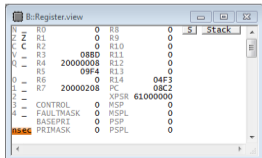
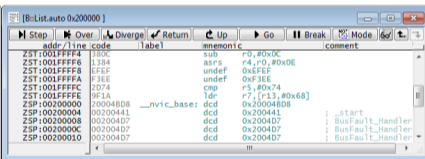
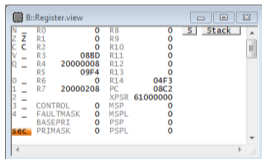
; access memory

Data.Set <N|Z>:<addr> <value>

Data.Set <addr> <value>

Data.dump <N|Z>:<addr>

Data.dump <addr>



# Peripherals

```

; -----
; access physical memory
Data.Set <A|AZ|AN>:<addr> <value>
Data.dump <A|AZ|AN>:<addr>
; -----
; peripheral view
PER
PER , /NonSecure
PER , /Secure

```

B:PER, "Nested Vectored Interrupt Controller,Enable" /Secure

Nested Vectored Interrupt Controller  
ICTR 00000003 INTLINESNUM 0-128

Interrupt Enable Registers

IRQ0_31_EN_SET/CLR	FFFFFFF	ENA31	Enabled	ENA30	Enabled	ENA29	Enabled	ENA28	Enabled	ENA27	Enabled	ENA26	Enabled
		ENA25	Enabled	ENA24	Enabled	ENA23	Enabled	ENA22	Enabled	ENA21	Enabled	ENA20	Enabled
		ENA19	Enabled	ENA18	Enabled	ENA17	Enabled	ENA16	Enabled	ENA15	Enabled	ENA14	Enabled
		ENA13	Enabled	ENA12	Enabled	ENA11	Enabled	ENA10	Enabled	ENA9	Enabled	ENA8	Enabled
		ENA7	Enabled	ENA6	Enabled	ENA5	Enabled	ENA4	Enabled	ENA3	Enabled	ENA2	Enabled
		ENA1	Enabled	ENA0	Enabled								

B::Data.dump AZD:0xE000E100

address	0 0123
AZD:E000E100	FFFFFFFF 0000

B:PER, "Nested Vectored Interrupt Controller,Enable" /NonSecure

Nested Vectored Interrupt Controller  
ICTR 00000003 INTLINESNUM 0-128

Interrupt Enable Registers

IRQ0_31_EN_SET/CLR	FFFFFF000	ENA31	Enabled	ENA30	Enabled	ENA29	Enabled	ENA28	Enabled	ENA27	Enabled	ENA26	Enabled
		ENA25	Enabled	ENA24	Enabled	ENA23	Enabled	ENA22	Enabled	ENA21	Enabled	ENA20	Enabled
		ENA19	Enabled	ENA18	Enabled	ENA17	Enabled	ENA16	Enabled	ENA15	Enabled	ENA14	Enabled
		ENA13	Disabled	ENA12	Disabled	ENA11	Disabled	ENA10	Disabled	ENA9	Disabled	ENA8	Disabled
		ENA7	Disabled	ENA6	Disabled	ENA5	Disabled	ENA4	Disabled	ENA3	Disabled	ENA2	Disabled
		ENA1	Disabled	ENA0	Disabled								

B::Data.dump AND:0xE000E100

address	0 0123
AND:E000E100	FFFFFF000 0000

# Flash programming

```
SYStem.CPU <cpu>
```

```
SYStem.Up
```

```
; -----
```

```
; Flash programming
```

```
; prepare flash programming (declarations)
```

```
DO ~/demo/arm/flash/<driver>.cmm PREPAREONLY
```

```
; ReProgram Flash
```

```
FLASH.ReProgram ALL
```

```
Data.LOAD.Elf <secure.elf> /NosYmbol
```

```
Data.LOAD.Elf <non_secure.elf> /NosYmbol
```

```
FLASH.ReProgram OFF
```

```
; -----
```

```
; load sYmbols
```

```
Data.LOAD.Elf <secure.elf> Z: /NoCODE
```

```
Data.LOAD.Elf <non_secure.elf> N: /NoCODE /NoCLEAR
```