ARM TrustZone

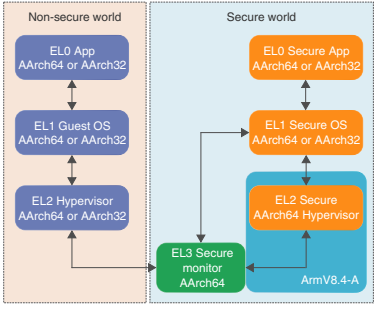
Alex Merkle

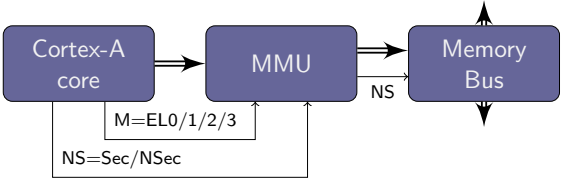
November 26, 2019

Overview

TrustZone Cortex-A

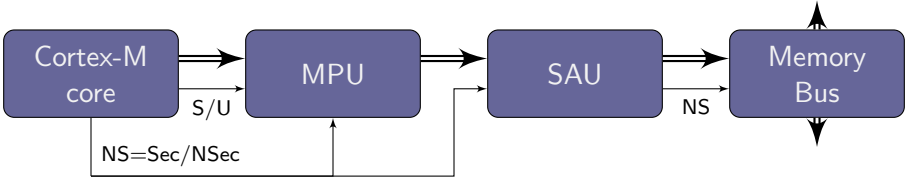
- Cortex-A cores feature multiple *Exception Level* EL0/1/2/3
  EL0 = User, EL1 = Supervisor, EL2 = Hypervisor, EL3 = Monitor
- Additionally there is a *Security State* Secure/NonSecure
- Both *Exception Level* and *Security State* control the MMU
- *Security State* is also connected to the SoC interconnect





TrustZone Cortex-M

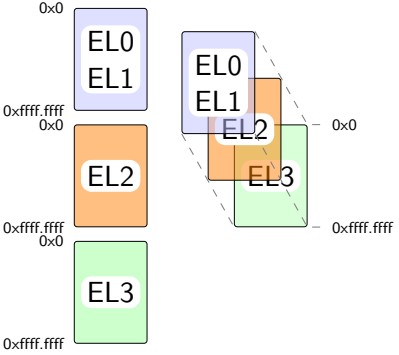
- Cortex-M cores feature Handler and Thread mode
  Handler = Supervisor , Thread = User/NonPrivileged
- Additionally there is a *Security State* Secure/NonSecure
- *Security State* controls the SAU
- *Security State* is also connected to the SoC interconnect



Overview

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

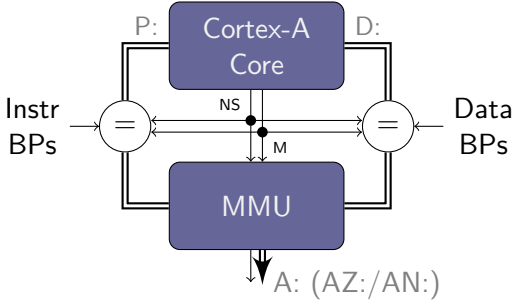
- TRACE32 Access Classes allow to access virtual memory of a specific mode
- TRACE32 combines the specified access class with the current CPU-Mode if the address is not fully qualified

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

[1] H: is a an alias for NH:
[2] starting from ARMv8.4

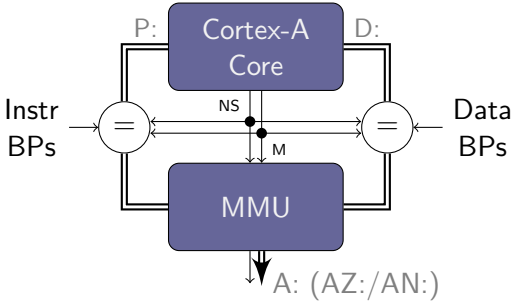
Onchip Breakpoints

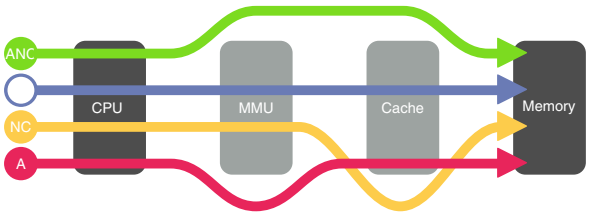
- Onchip Breakpoints can trigger on *Security State* Secure/NonSecure
- Onchip Breakpoints can trigger on *Exception Level* EL0+1/EL2/EL3
- No Onchip Breakpoints to physical memory



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

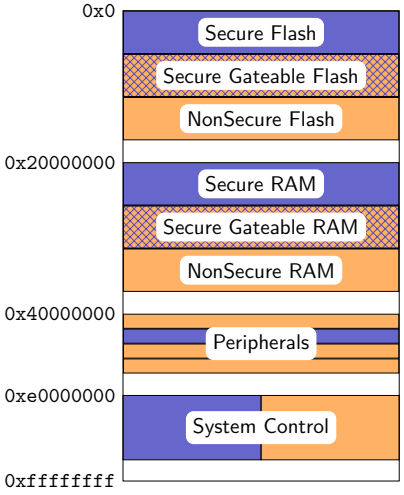
```
SYStem.Option ZONESPACES ON
Break.CONFIG.MatchZone   ON
```

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

          ON
      MatchZone
         OFF

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

Overview

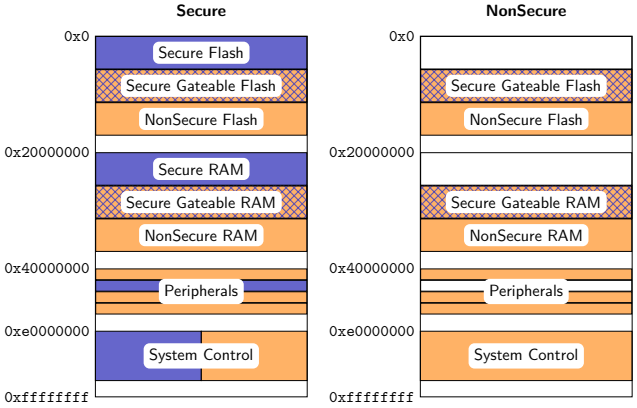
Memory Map

- 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



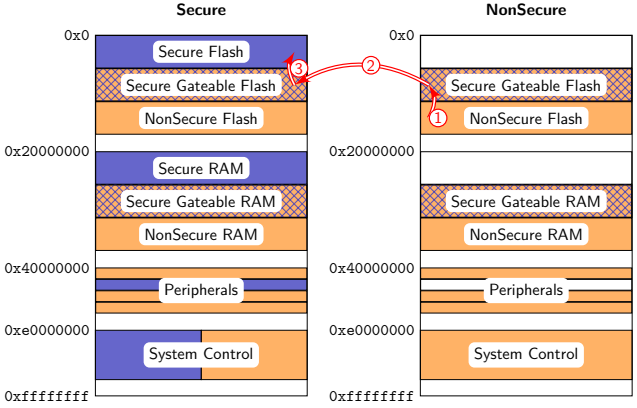
Memory Map

- The SAU works like a blinker/blinder for NonSecure



NonSecure to Secure switch

- Secure Gateable Region is used as a veneer



TRACE32 Access Classes

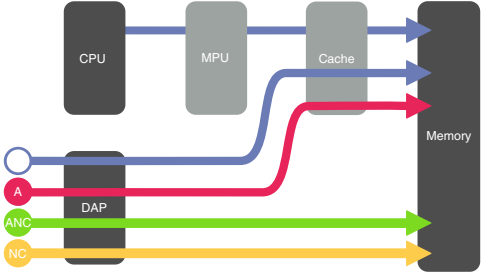
- TRACE32 Access Classes allow to access memory of a specific *Security State*
- TRACE32 combines the specified access class with the current CPU-Mode if the address is not fully qualified

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

[1] 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:
- Addresses for Cortex-M are always physical

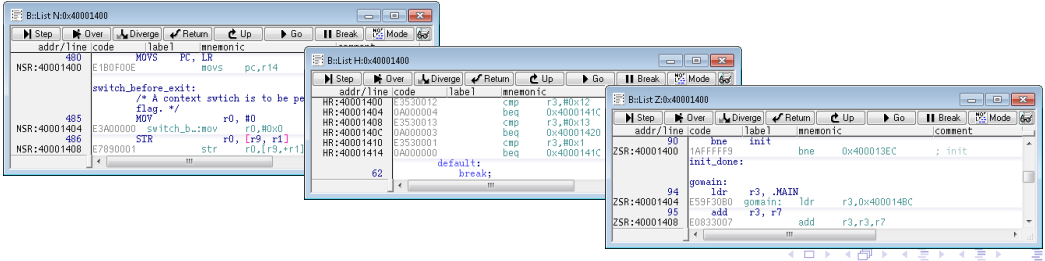


Overview

Symbols

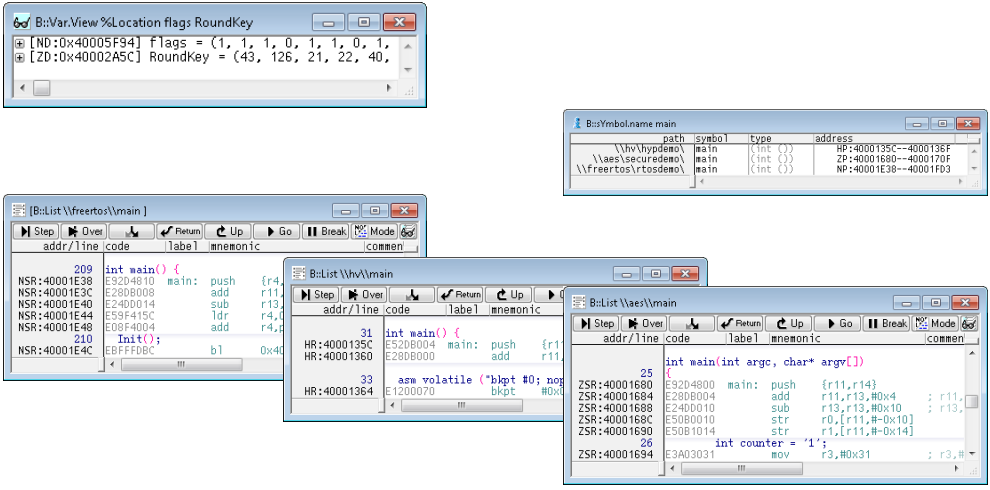
```
SYStem.Option ZONESPACES ON
Break.CONFIG.MatchZone   ON
; ---------------------------------------------------------
; load sYmbols
Data.LOAD.Elf <code_on_el3.elf>      M: /NoCODE
Data.LOAD.Elf <code_on_el2.elf>     NH: /NoCODE
Data.LOAD.Elf <code_on_el1_ns.elf> NS: /NoCODE
Data.LOAD.Elf <code_on_el1_s.elf>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

Different code and different sourcecode & memory for each zone



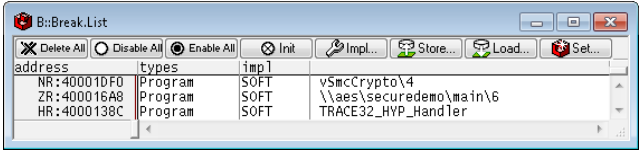
sYmbols

Functions & Variables are automatically accessed with the correct zone



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]
```

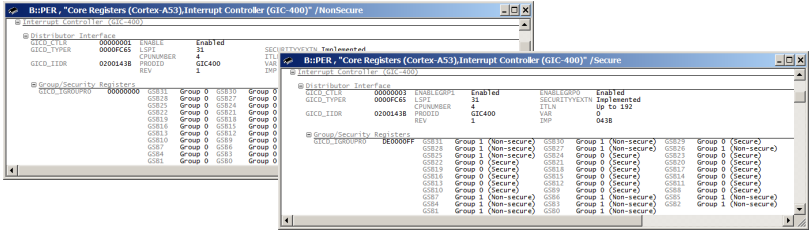


Access Memory

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



Overview

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

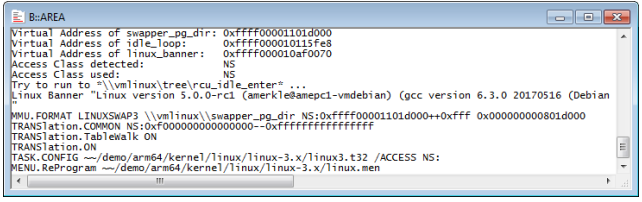
Debugging Linux + bl31

```
; --------------------------------------------------
; 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 ⇒ `SYStem.Mode Attach`
  - Debug symbols must be loaded ⇒ `Data.LOAD.Elf ...`
  - DO `~~/demo/<arm|arm64>/kernel/linux/boards/generic-template/detect_translation.cmm`



- 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`

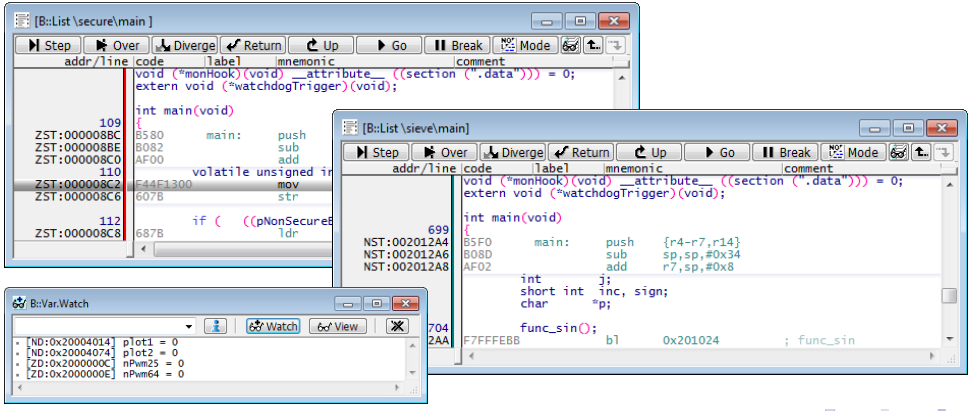
  triggered by the optee menu *Debug Trusted Application from Entry*.
- For TAs that are loaded persistently (no `TEEC_CloseSession`), possibly ambiguous Onchip-Breakpoints need to be used

Overview

Symbols

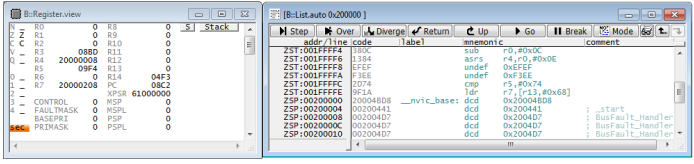
```
; -------------------------------------------------
; load sYmbols
Data.LOAD.Elf <secure.elf>     Z: /NoCODE
Data.LOAD.Elf <non_secure.elf> N: /NoCODE /NoCLEAR
```

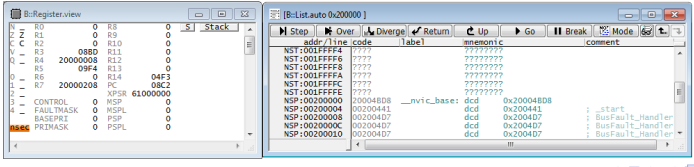
Functions & Variables are automatically accessed with the correct zone



Access Memory

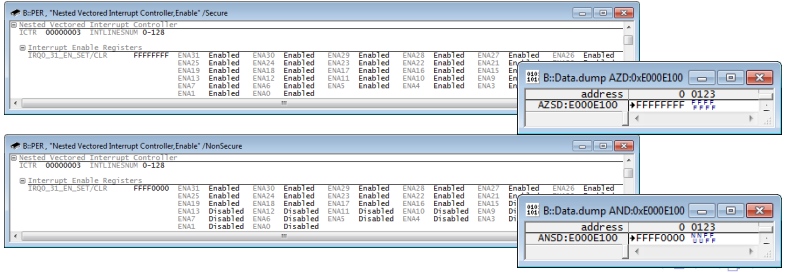
```
; ------------------------------------------------------
; access memory
Data.Set  <N|Z>:<addr> <value>
Data.Set  <addr> <value>
Data.dump <N|Z>:<addr>
Data.dump <addr>
```





Peripherals

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



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
```