

# Arm Debugger: Attach or Up

Choose the correct debug scenario

Alex Merkle, Lauterbach Engineering GmbH & Co. KG

February 01, 2022





# AGENDA

1. Preface
2. Example U-Boot

# Motivation

- TRACE32 offers multiple commands to establish a connection to a target platform
- Goal of this document is to help choosing the correct approach for the use case
- Within this document we will use the terms *Attach* and *Up* scenarios
- The essential difference at a glance
  - *Attach* scenarios establish the debug connection but do **not** reset/restart the target platform
  - *Up* scenarios will reset/restart the target platform before establishing the debug connection
- This document will mainly put focus on *Up* scenarios



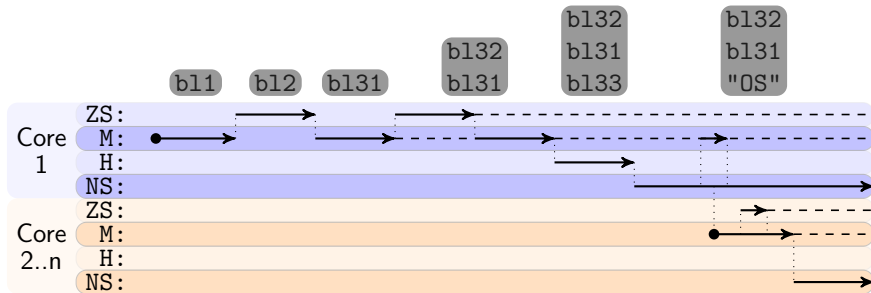
# Terms

Throughout this document the following TERMS are used (aligned with ARM TF-A)

- b11** 1st bootloader, ROM, loads secondary bootloader bl2 from external media
- b12** 2nd bootloader, RAM, may load multiple next stage bootloaders (bl31/bl32/bl33/...)
- b131** runtime firmware, RAM, keeps running in Monitor mode and offers firmware functionalities e.g. PSCI, SPD, ...
- b132** trusted OS, RAM, OS running in secure zone e.g. OP-Tee
- b133** nonsecure bootloader, RAM, typically u-boot, fastboot, coreboot bootloader offering first interactive shell
- OS** rich OS, RAM, e.g. Linux/Qnx/VxWorks or hypervisor Xen/L4Re/Qnx/Coqos



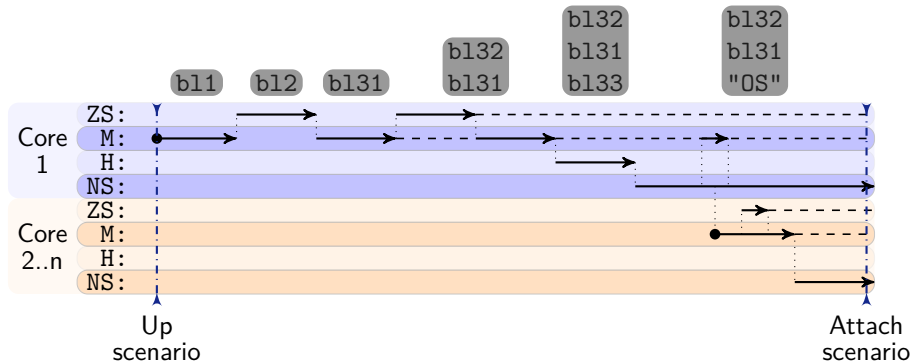
# typical TF-A bootflow



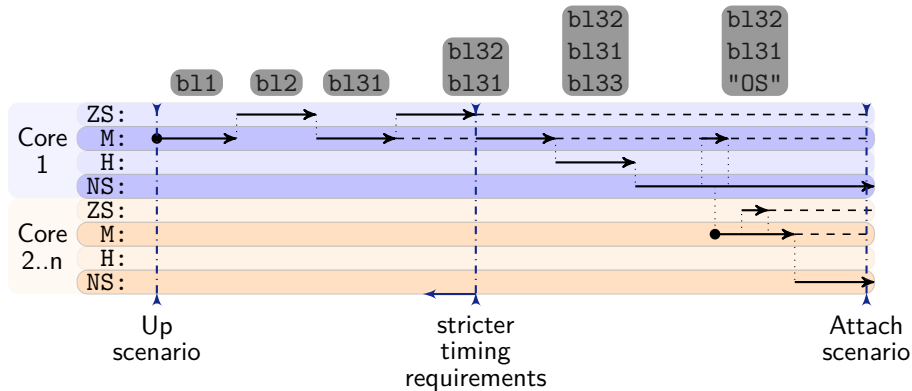
See also [https://www.lauterbach.com/projects\\_download/training\\_manuals/arm\\_trustzone.pdf](https://www.lauterbach.com/projects_download/training_manuals/arm_trustzone.pdf) for the meaning of access-classes ZS:/M:/H:/NS:.

Legend: ● Reset/Core start → Code alive, executing -- Code alive, inactive

# typical TF-A bootflow



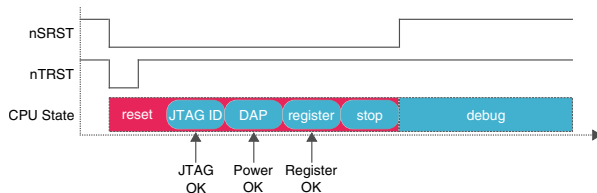
# typical TF-A bootflow



# ideal SYStem.Mode Up

- TRACE32 assumes that communication with the target platform can take place with nSRST/nReset line asserted
- TRACE32 can stop the core as soon as it starts

```
script.cmm
RESet
SYStem.CPU <>
SYStem.Option NoPRCRRReset ON
[CORE.ASSIGN <1.|2.>]
SYStem.Up
```



The more complex the target becomes this approach will not succeed anymore.

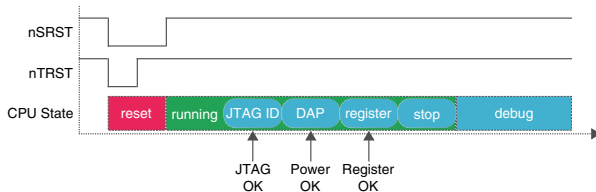




# relaxed SYStem.Mode Up

- TRACE32 offers diverse SYStem.Option's to relax the SYStem.Mode Up behavior
- SYStem.Option ResBreak OFF will force TRACE32 not to communicate with the target platform with nSRST/nReset asserted
- ⇒ Target platform will boot for *some* time

```
script.cmm  
RESet  
SYStem.CPU <>  
SYStem.Option ResBreak OFF  
SYStem.Option NoPRCRReset ON  
[CORE.ASSIGN <1.|2.>]  
SYStem.Up
```



## WaitReset

```
RESet
SYStem.CPU <>
SYStem.Option ResBreak OFF
SYStem.Option WaitReset <time>

SYStem.Option NoPRCRRReset ON
[CORE.ASSIGN <1.|2.>]
SYStem.Up
```

## IDCODE/DAPPWR/DBGREG

```
RESet
SYStem.CPU <>
SYStem.Option ResBreak OFF
SYStem.Option WaitIDCODE <time>
SYStem.Option WaitDAPPWR <time>
SYStem.Option WaitDBGREG <time>
SYStem.Option NoPRCRRReset ON
[CORE.ASSIGN <1.|2.>]
SYStem.Up
```

## SoC specific

```
RESet
SYStem.CPU <>

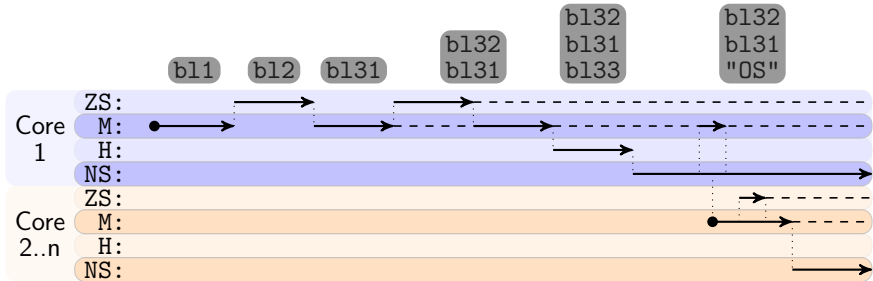
???

[CORE.ASSIGN <1.|2.>]
SYStem.Up
```

- More information can be found in *TRACE32: Help → Processor Architecture Manual*
- ARMv8: [~/pdf/debugger\\_armv8v9.pdf](#) chapter *Configure Debugger for SoC Specific Reset Behavior*
- Example scripts: [~/demo/arm/hardware/...](#) or *TRACE32: File → Search for Script*
- Complicated? Yes, SoC internal security, power-management, reset circuitry, ... implications become visible to the end-user

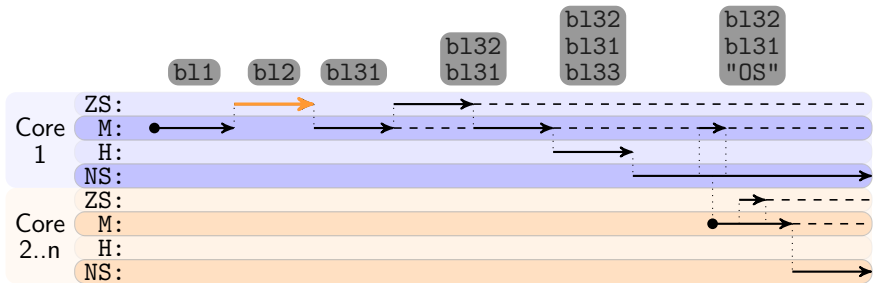
# Causality

- In order to debug a specific bootstage we need to connect before this stage begins



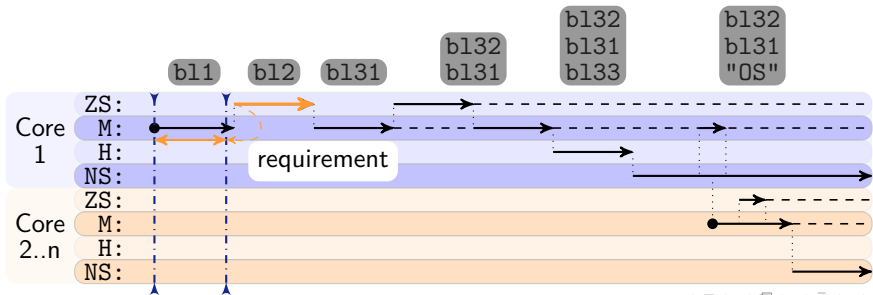
# Causality

- In order to debug a specific bootstage we need to connect before this stage begins
- Example: in order to debug bootloader b12 we need to connect before it starts



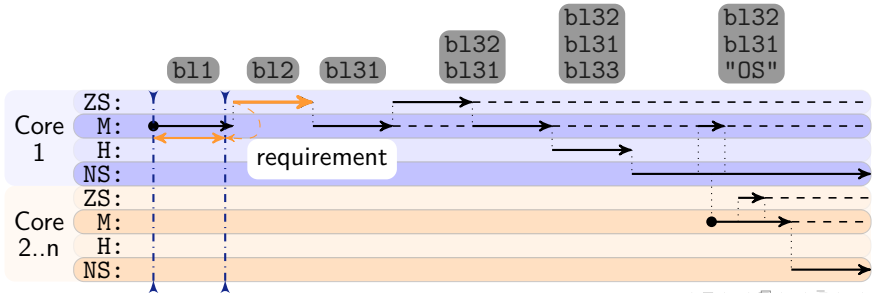
# Causality

- In order to debug a specific bootstage we need to connect before this stage begins
- Example: in order to debug bootloader b12 we need to connect before it starts
- Referring to our example it's enough as long as we stop while b11 stage (ROM), thus a short target boot time (buzzword ResBreak OFF) is okay



# Causality

- In order to debug a specific bootstage we need to connect before this stage begins
- Example: in order to debug bootloader b12 we need to connect before it starts
- Referring to our example it's enough as long as we stop while b11 stage (ROM), thus a short target boot time (buzzword ResBreak OFF) is okay
- Causality can be relaxed e.g. by putting a busy waiting loop into the bootloader



# CORE.ASSIGN

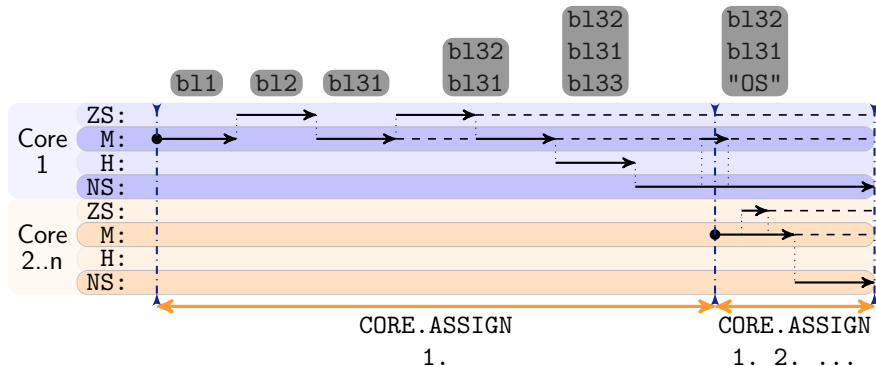
- Syntax: `CORE.ASSIGN <> [<>, [...]]`
- Conditions: only available and required if the selected `SYStem.CPU <>` selects a multicore cluster e.g. 8x Cortex-A55
- `CORE.ASSIGN` allows to select physical cores of the multicore cluster and merges them into one SMP system
  - `CORE.ASSIGN 3.` selects physical core 3
  - `CORE.ASSIGN 1. 2. 3.` selects physical cores 1 2 3 as one SMP system
- assigned cores become logical cores in the debug session

<code>CORE.ASSIGN</code>	2.	4.	6.	8.	physical numbering $1 \leq x \leq \text{corenumber}$
<code>CORE.select</code>	0.	1.	2.	3.	logical numbering $0 \leq x \dots$
- Rule of thumb: Key for success not only when debugging boot-scenarios, assign only cores which are up and running
- Reason: accessing unpowered/unlocked resources from the debug logic might trigger unrecoverable errors, highly target platform specific behavior



# CORE.ASSIGN

Rule of thumb: Key for success not only when debugging boot-scenarios, assign only cores which are up and running





## Homogeneous Multicore

- All cores use the same core architecture and type
- CORE.ASSIGN is linear indexed

```
B::core.assign  
assignment: 1, 2, 3, 4 (CA55, CA55, CA55, CA55)
```

Bootcore is most likely  
CORE.ASSIGN 1.

## Heterogeneous Multicore

- Cores use the same architecture but different types
- CORE.ASSIGN is tick/tock indexed  
ARM big.LITTLE, DynamIQ

```
B::core.assign  
assignment: 1, 2, 3, 4 (CA76, CA55, CA76, CA55)
```

Bootcore is either  
CORE.ASSIGN 1.  
or  
CORE.ASSIGN 2.

⇒ Check example scripts ~/demo/arm/hardware/...

Correct setting may depend on BOOTMODE lines or BOOTIMAGE, target platform specific!

# Breakpoints

- use ONCHIP Breakpoints to stop at every entry point of a new bootloader stage
- starting from the entry point, software breakpoints may be used within that bootloader stage
- Why ONCHIP breakpoints?
  - Task of a bootloader is to load the code for the next bootstages  
⇒ software breakpoints set in the next bootstages may get overwritten by the current bootloader stage
  - Task of a bootloader is to initialize hardware e.g. DRAM  
⇒ software breakpoints can't be set into untrained DRAM
- Syntax: `Break.Set <address|symbol> /Onchip`
- Further ONCHIP Breakpoint like features available - Onchip Triggers or ARM ETM



# Code Breakpoints

## Core logic - single address

```
Break.Set 0x80080000 /Program /Onchip
Break.Set _head /Program /Onchip
Break.Set sYmbol.SECADDRESS(.head.text) /Program /Onchip
```

See also: [~/pdf/debugger\\_armv8v9.pdf](~/pdf/debugger_armv8v9.pdf) - Breakpoints

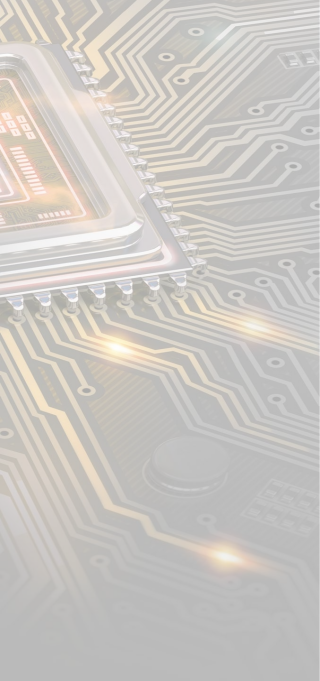
## ETM logic - address range [optional]

```
ETM.StoppingBreakpoints ON
Break.Set <range> /Program /Onchip
Break.Set 0x80000000++0x0ffffff /Program /Onchip
```

## Onchip Triggers - exception level

```
TrOnchip.Set <NSEL1|NSEL2|SEL3|SEL1> <OFF|ON> ; trigger as soon as CPU enters a specific mode
TrOnchip.Set NSEL1 ON ; trigger as soon as CPU enters NonSecure EL1
; may be also read as Break.Set NS:0x0--0xffffffff /Program /Onchip
```

Reminder: All those address comparisons use virtual addresses.



## 1. Preface

## 2. Example U-Boot

## Scenario: Connect in u-boot shell

Use cases:

- debug u-boot command
- load/patch OS code using TRACE32
- debug OS boot
- remote control of u-boot shell

Can be solved as a *Attach* and *Up* scenario.

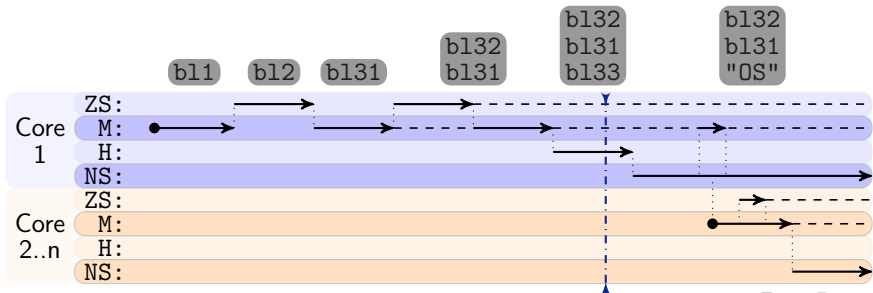
```

B:TERM#1
U-Boot 2019.07-rc4-gc17c4a29 (Jan 30 2020 - 17:07:25 +0000) vexpress_aemv8a fvp
aarch64 semi, Build: jenkins-armllt-platforms-release-77

DRAM: 2 GiB
Flash: 64 MiB
*** Warning - bad CRC, using default environment

In: serial_pl01x
Out: serial_pl01x
Err: serial_pl01x
Net: SMC91111-0
Error: SMC91111-0 address not set.

Hit any key to stop autoboot: 0
sem_open: ERROR fd -1 for file 'devtree.dtb'
semhosting load failed, try booting with contents of DRAM
Bad Linux ARM64 Image magic!
fvp#
  
```



# Manual Attach

- connect serial terminal e.g. PUTTY/minicom
- press reset button/power cycle platform
- as soon as terminal string shows up, invoke  
    D0 script  
    in TRACE32
- script.cmm:
  1. reset settings of TRACE32
  2. configure TRACE32 for target platform
  3. connect and stop

- script.cmm

  1. RESet
  2. SYStem.CPU <>  
[CORE.ASSIGN <1.|2.>]
  3. SYStem.Mode Attach  
Break



# TERM.TRIGGER

> script.cmm:

1. reset settings of TRACE32
2. configure TRACE32 for target platform
3. ensure disconnect
4. connect terminal/serial console
5. setup trigger for string *Hit any key*  
RESET target platform (assert nReset line)
6. stop as soon as string appears in terminal

⇒ an *Up* scenario without SYStem.Mode Up

```
script.cmm
1.  RESet
2.  SYStem.CPU <>
    [CORE.ASSIGN <1.|2.>]
3.  SYStem.Down
4.  TERM.METHOD #1 COM ....
    TERM.view
5.  TERM.TRIGGER #1 "Hit any key"
    SYStem.ResetOut
6.  SCREEN.WAIT TERM.TRIGGERED(#1)
    SYStem.Mode Attach
    Break
```



# Using Symbols

## > script.cmm:

1. reset settings of TRACE32
2. configure TRACE32 for target platform
3. use SYStem.Mode Up
4. load debug symbols only
5. wait till temporary onchip breakpoint on start symbol of bootloader triggers
6. wait till temporary breakpoint in relocation procedure triggers
7. let bootloader perform relocation
8. wait till temporary breakpoint on key-press check triggers

```
script.cmm

1.  RESet

2.  SYStem.CPU <>
    [CORE.ASSIGN <1.|2.>]
    [SYStem.Option ...]

3.  SYStem.Up

4.  [Break.CONFIG MATCHZONE ON]
    Data.LOAD Elf u-boot /NoCODE

5.  Go __image_copy_start /Onchip /Program
    WAIT !STATE.RUN()

6.  Go relocate_done
    WAIT !STATE.RUN()

7.  Go.Up
    WAIT !STATE.RUN()

    DO ~/demo/<arch>/bootloader/uboot/
        load_uboot_symbols_reloc.cmm

8.  Go tstc
    WAIT !STATE.RUN()
```





# Using Symbols

> script.cmm:

1. reset settings of TRACE32
2. configure TRACE32 for target platform
3. use SYStem.Mode Up
4. load debug symbols only
5. wait till cpu enters NonSecure EL2
6. wait till temporary breakpoint in relocation procedure triggers
7. let bootloader perform relocation
8. wait till temporary breakpoint on key-press check triggers


```
script.cmm
1.  RESet
2.  SYStem.CPU <>
    [CORE.ASSIGN <1.|2.>]
    [SYStem.Option ...]
3.  SYStem.Up
4.  Data.LOAD.Elf u-boot /NoCODE
5.  TrOnchip.Set NSEL2 ON
    Go
    WAIT !STATE.RUN()
    TrOnchip.Set NSEL2 OFF
6.  Go relocate_done
    WAIT !STATE.RUN()
7.  Go.Up
    WAIT !STATE.RUN()

    DO ~/demo/<arch>/bootloader/uboot/
        load_uboot_symbols_reloc.cmm
8.  Go tstc
    WAIT !STATE.RUN()
```





Questions?



Thank You!

Arm Debugger: Attach or Up