FTF 2016
TECHNOLOGY FORUM

DEVICE AND U-BOOT BRING-UP USING CODEWARRIOR-ARMv8

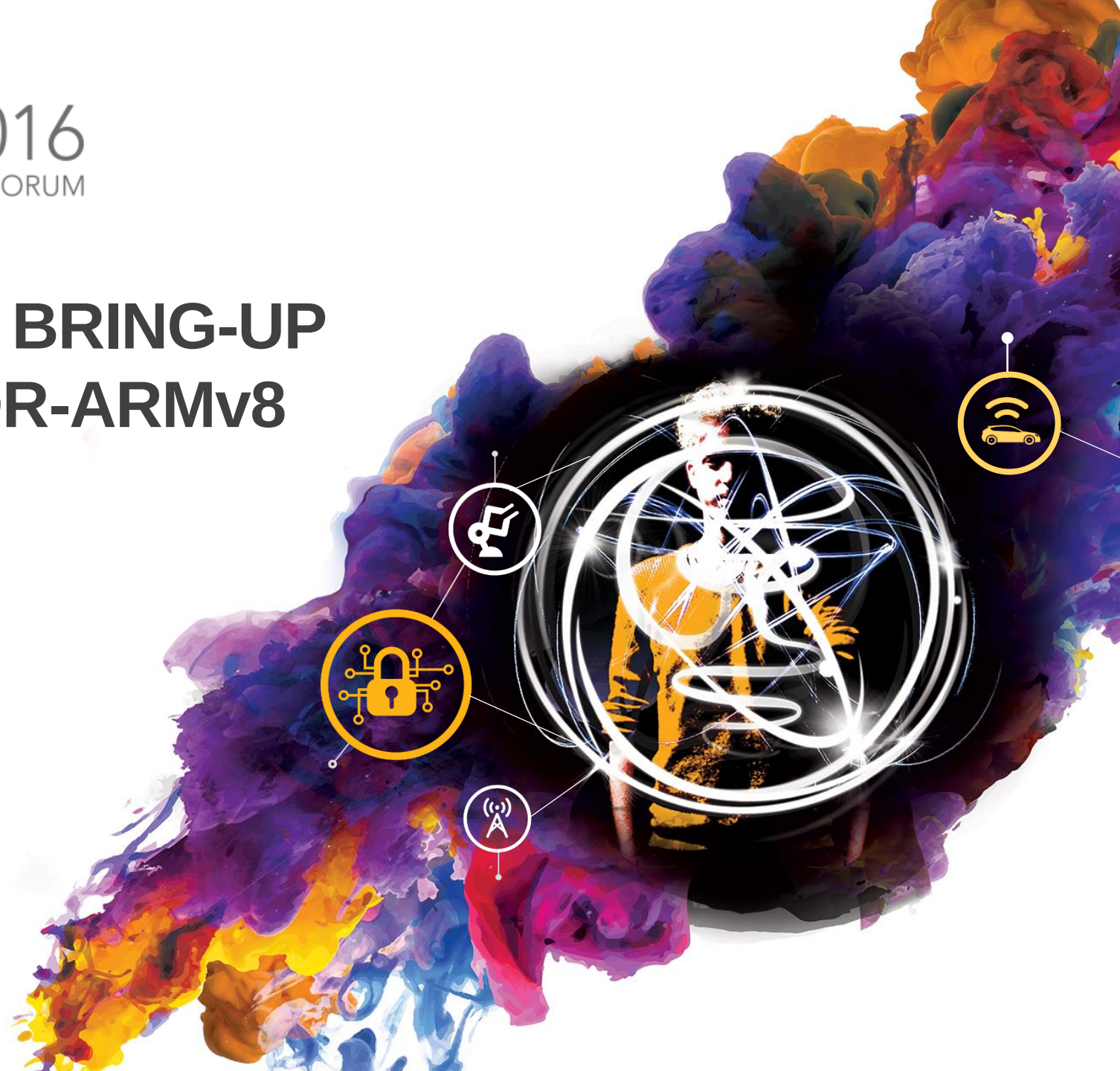
FTF-DES-N1834

ROBERT MCGOWAN, CHIEF ARCHITECT

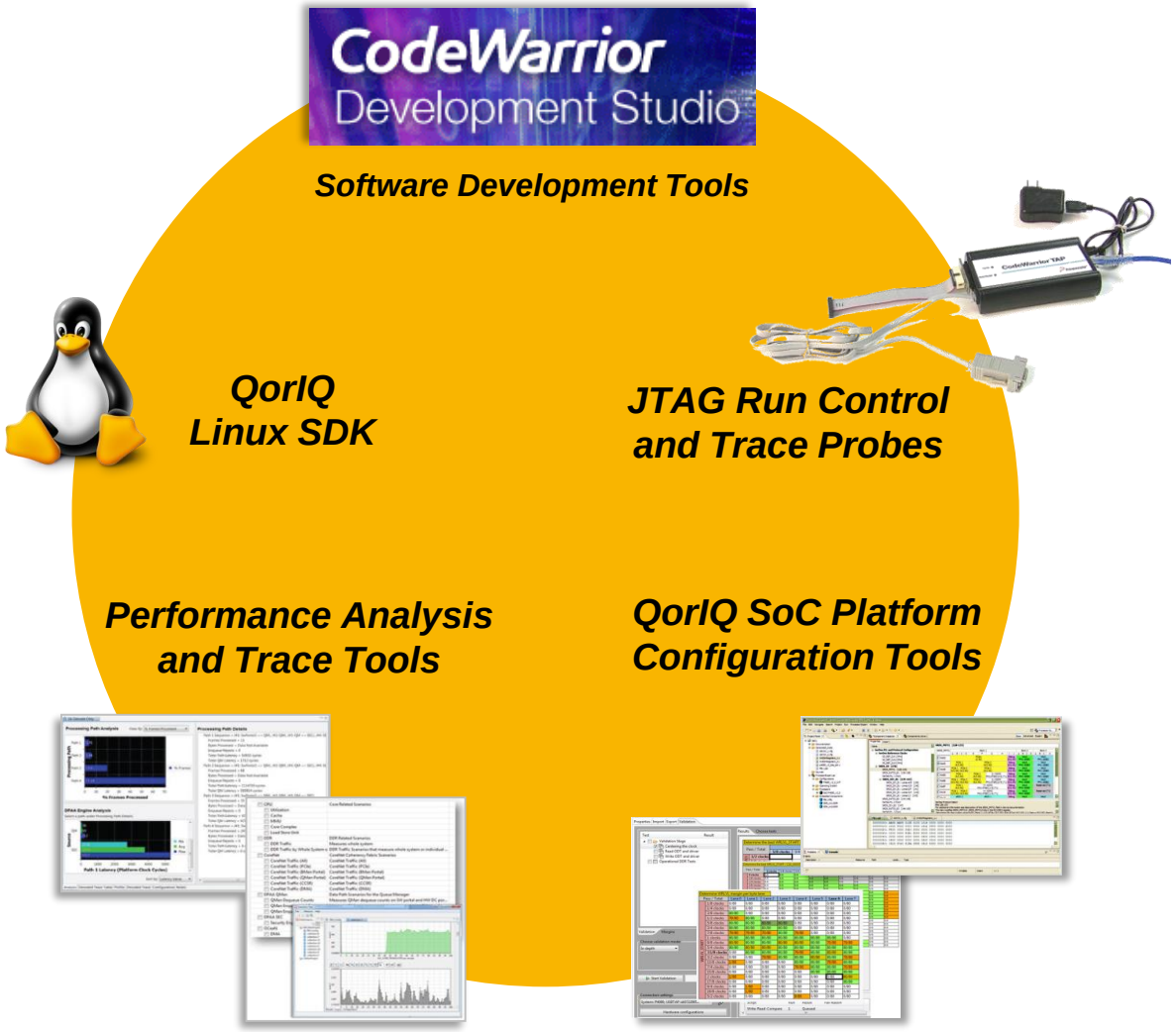
CATALIN UDMA

FTF-DES-N1834

MAY 16, 2016



Software and Tools Enablement for QorIQ LS-Series



Training and Hands-on Goals

- The following material has been developed so you...
  - …Become familiar with the LS2085A-RDB board
  - …Perform LS2085A-RDB board bring-up using CodeWarrior-ARMv8
  - …Perform U-Boot debug

AGENDA

- Lecture: Introduction/Overview
- Lecture: CodeWarrior
- Lecture: Board/Device Overview
- Activity: Create Bareboard Project
- Activity: Establish a CodeWarrior Connection
- Activity: Connection Diagnostic
- Activity: Flash Programmer
- Lecture: U-Boot Debug
- Activity: U-Boot Debug

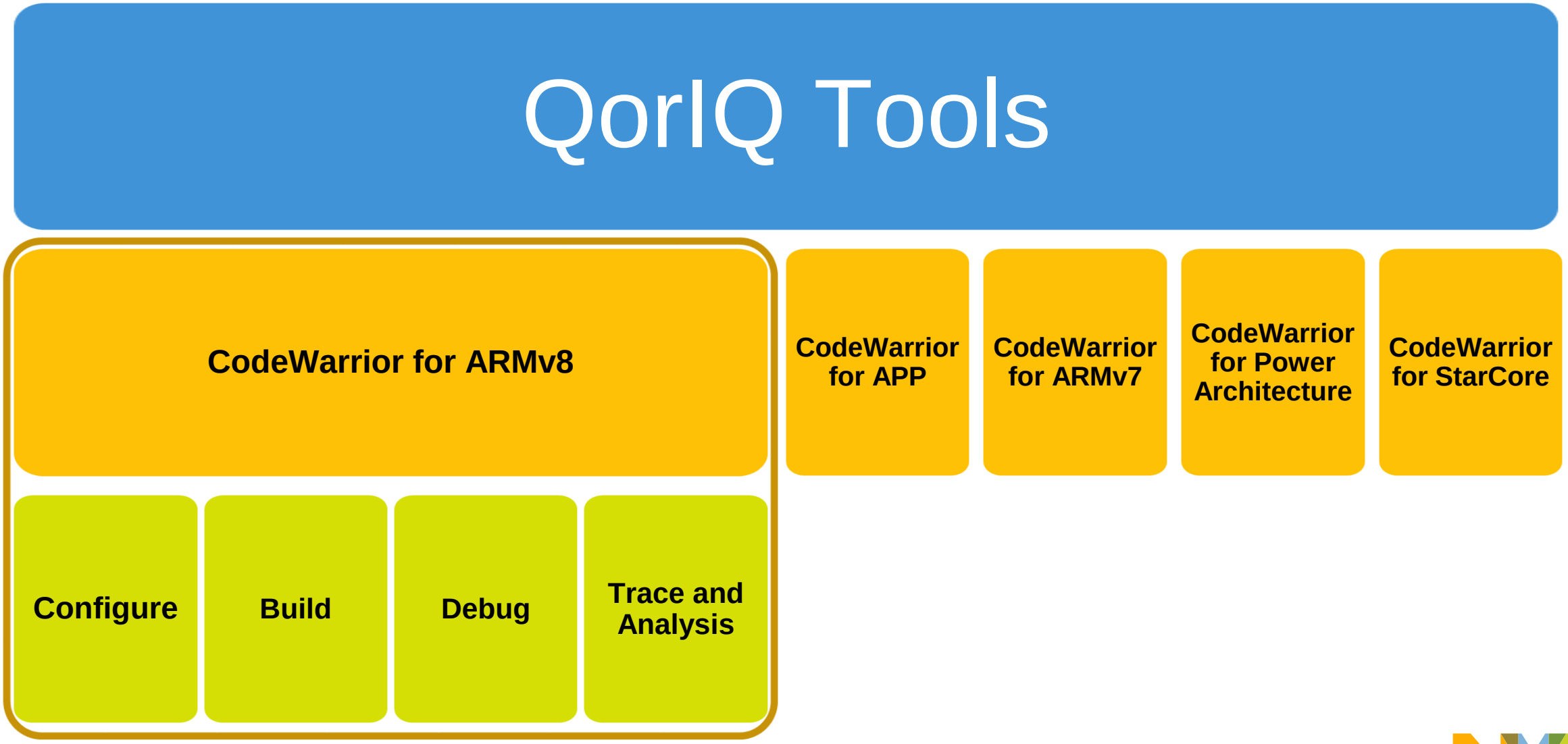
INTRODUCTION/ OVERVIEW

QorIQ TOOLS

CodeWarrior for ARMv8 ISA
CW-TAP

CODEWARRIOR

CodeWarrior Family



CodeWarrior Development Studio

A Complete Development Environment Under Eclipse

Eclipse IDE

- Configuration Wizards
- Plug-in Architecture
- 3rd party community

Build Tools

- C/C++ Compiler

Initialization Tools

- SOC platform initialization and configuration

Run Control

- CW-TAP

Debugger

- Multicore aware
- Cross-triggering
  - Run/Stop of targets simultaneously
- Access to all on-chip resources
- Linux awareness

Software Analysis – Trace and Profile

- Leverages chip capabilities
  - Profiling Unit
  - In system trace buffering
- Trace/Code/Performance Viewer
- Offline trace visibility

CodeWarrior Aids Debug Through Multiple Phases

- SoC and board bring-up
  - Single-core and multicore (AMP) bare-metal debugger
  - Device introspection: core and SoC registers, memory
  - U-Boot

- Linux OS development
  - SMP aware kernel debug
  - Device driver development and debug
  - Aligned with QorIQ SDK & Linaro GNU toolchain

CodeWarrior Aids Debug Through Multiple Phases (2)

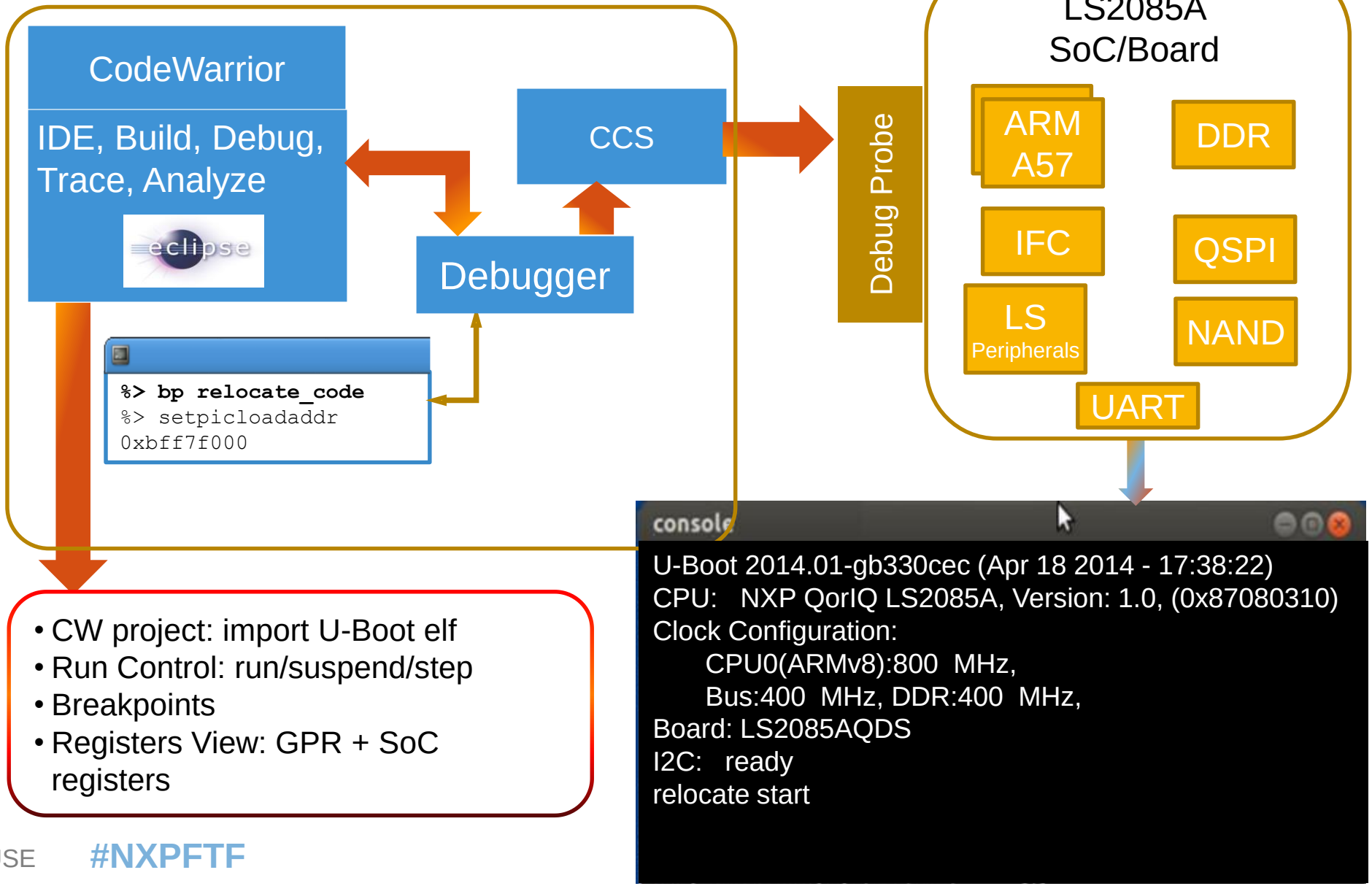
- Linux application development
  - GNU debugger compatible + extensions for Linux application debug
  - Linux target information: System Browser Linux kernel module development and debug
  - Aligned with QorIQ SDK and Linaro GNU toolchain
  - Target debug agent

- Performance Analysis
  - Core performance metrics and scenarios
  - SoC performance metrics and scenarios
  - Profiling from trace

CodeWarrior Aids Debug Through Multiple Phases (3)

- Non-intrusive debug through trace
  - Core and SoC trace sources: configuration, extraction, visibility
  - Post-mortem debugging: offline trace
  - Debug-print
  - Linux aware trace
  - Linux kernel trace
  - Code Coverage

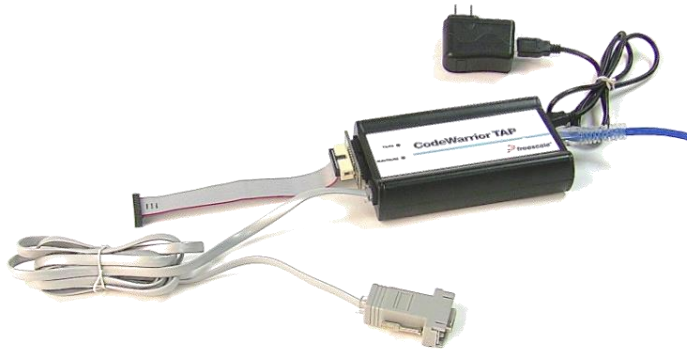
CodeWarrior: U-Boot Debug



- CW project: import U-Boot elf
- Run Control: run/suspend/step
- Breakpoints
- Registers View: GPR + SoC registers

CodeWarrior JTAG Probe

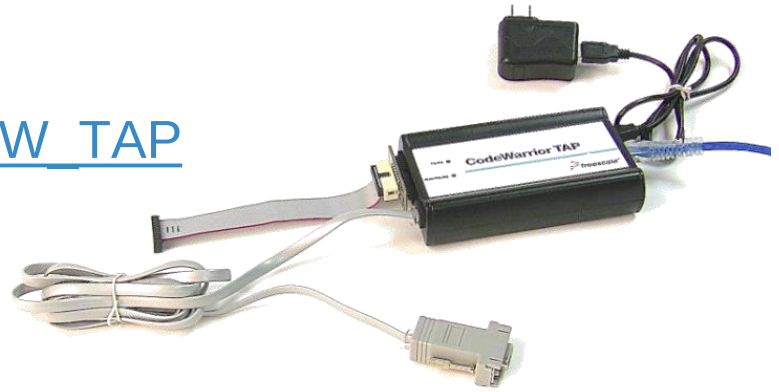
- CodeWarrior TAP
  - Provides connection between CodeWarrior and target device
  - Target run control
  - Serial port pass through
  - Interface to host
    - Locally over USB
    - Remotely over Ethernet
  - Buy separately (below $500)
- Reminder: no USB-TAP support



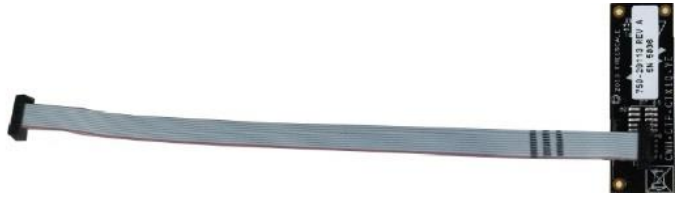
CodeWarrior TAP

- JTAG debugging and CodeWarrior run control requires:

Base Unit: (Part # CWH-CTP-BASE-HE)
www.nxp.com/webapp/sps/site/prod_summary.jsp?code=CW_TAP



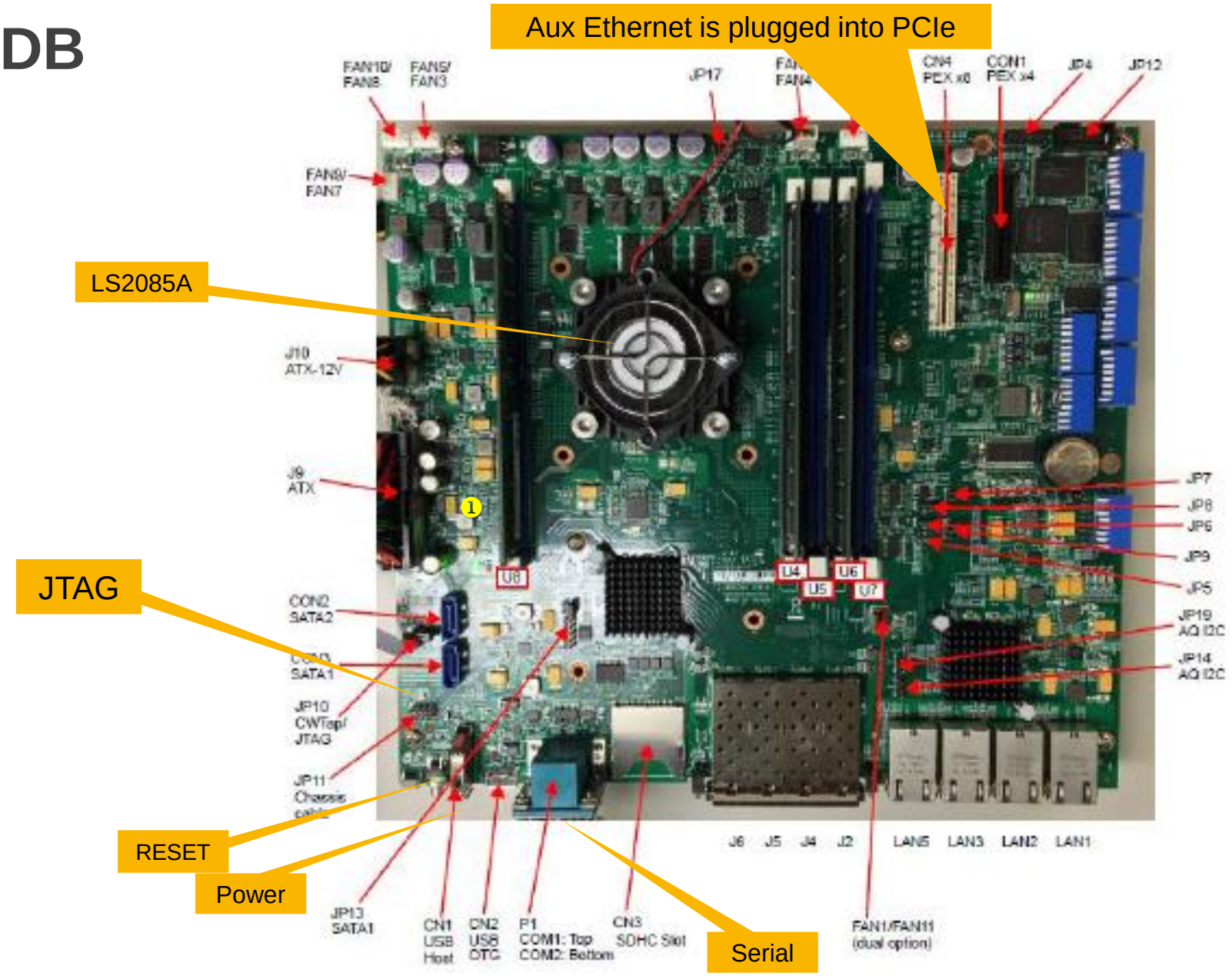
... and Probe Tip: (Part # CWH-CTP-CTX10-YE)
www.nxp.com/webapp/sps/site/prod_summary.jsp?code=CWH-CTP-CTX10-YE



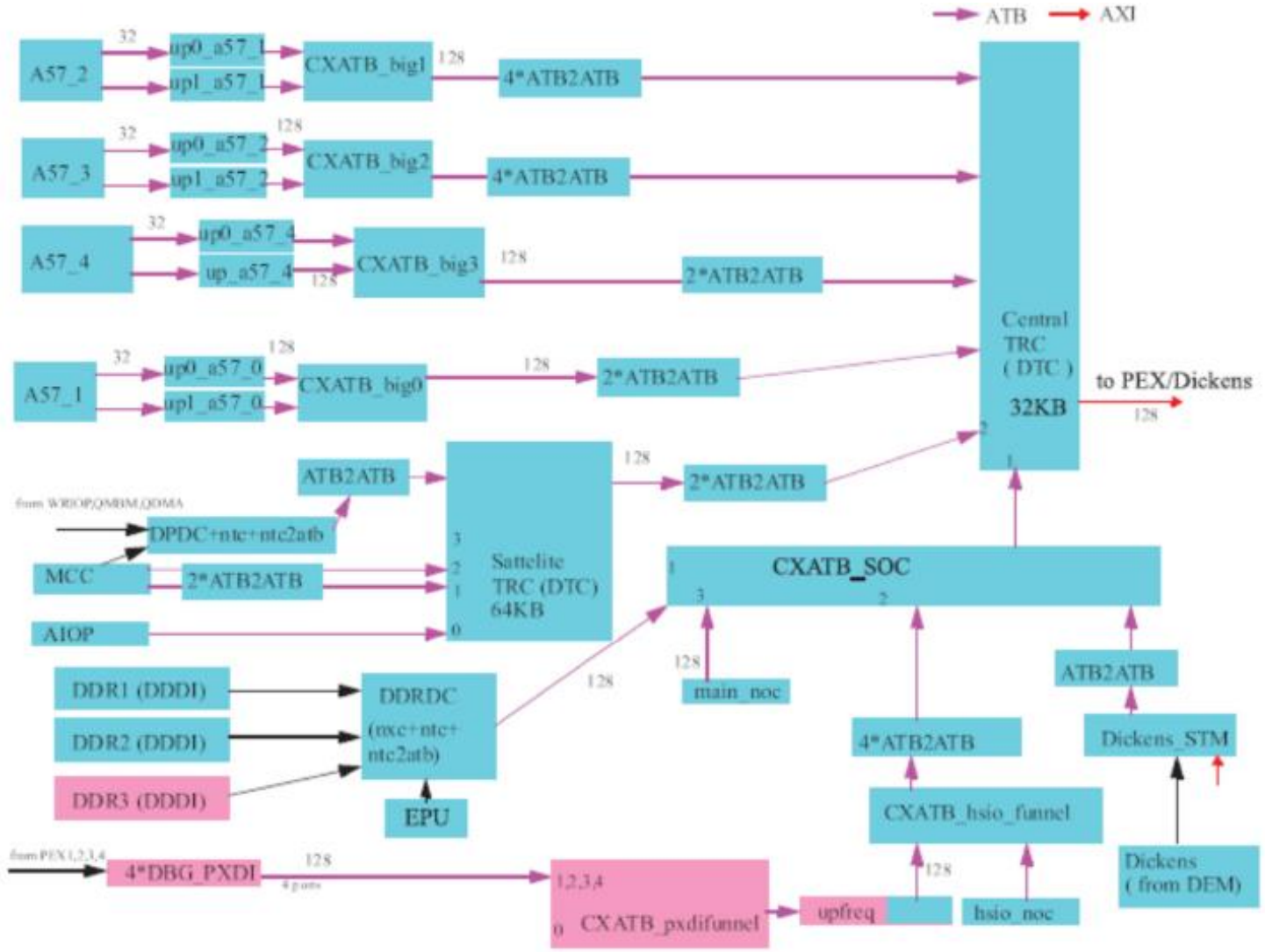
BOARD/DEVICE OVERVIEW

INTRODUCING THE LS2085A RDB

LS2085A RDB
Top View



QorIQ LS2085A Debug Block Diagram



Debug Features

- Run-Control debug features in cores
  - Cross-triggering between cores

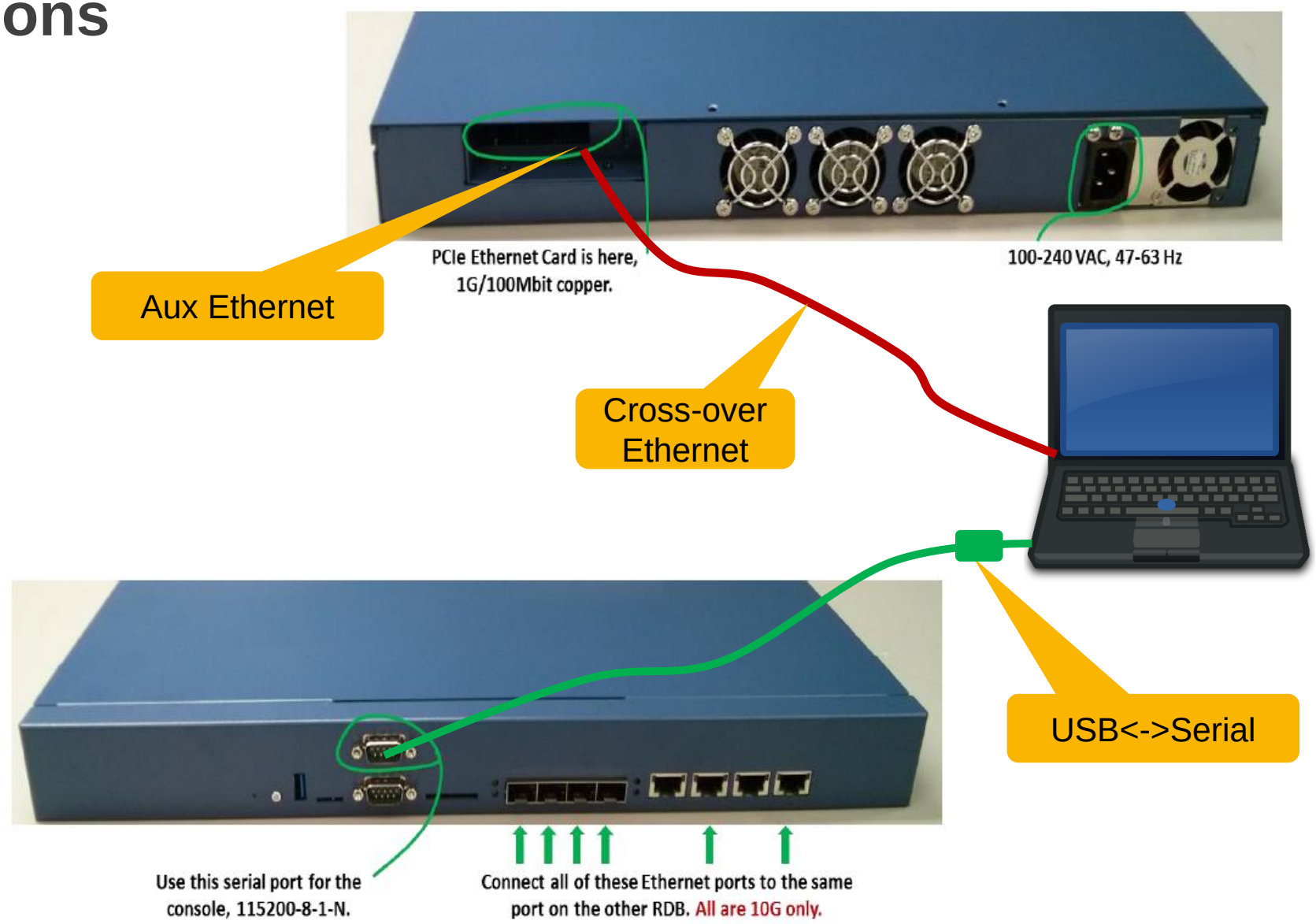
- Trace
  - Program trace (ETM)
  - System trace (STM)
  - Stored in internal memory or DDR
    - No external export via TPIU or Aurora

- EPU Performance Monitor

PREPARING THE ENVIRONMENT

What Has Been Done For You

Connections



Items That Have Been Setup for You

- Host OS
  - Best to use Linux on the host when developing Linux on the target
  - Multiple Linux OS supported
  - 64-bit Linux required
  - Used Mint 17.1 for class
- CodeWarrior for Networked Applications v2016.01
  - CodeWarrior for QorIQ ARMv8 ISA
- QorIQ Linux SDK for LS2085A RDB
  - Installed from ISOs – could also obtain from GIT
    - LS2085A-SDK-AARCH64-IMAGE-20160304-yocto
    - LS2085A-SDK-SOURCE-20160304-yocto
  - Did not use CACHE

Items That Have Been Setup for You

- Install on host
  - Yocto
  - Minicom / cutecom
    - 115200-8-N-1
  - Tftp server (not used in class)
  - telnet / putty (not used in class)

- Read RDB Quickstart Guide!
- Bitbake the SDK
- Install on target
  - Flash U-Boot

Class Information

- Linux Login
  - User: class
  - Password: codewarrior

- SDK is installed in ~/SDK
  - Need to use full path in tool: /home/class/SDK

- On desktop
  - Launcher to CodeWarrior – looks like rocket
  - shortcut to cutecom
  - Menu has link to terminal
    - Use for launch minicom

- No password on target Linux

RDB-LS2085A

SDK EAR6.0 Installed on LS2085A RDB

U-Boot Startup Messages

- Reset the RDB-LS2085A, interrupt the countdown
- Review the U-Boot output in the console window :

```
U-Boot 2015.10LS2085A-SDK+g3242b20 (Mar 21 2016 - 13:23:23 +0200)

SoC:  LS2085E (0x87010010)
Clock Configuration:
       CPU0(A57):1800 MHz  CPU1(A57):1800 MHz  CPU2(A57):1800 MHz
       CPU3(A57):1800 MHz  CPU4(A57):1800 MHz  CPU5(A57):1800 MHz
       CPU6(A57):1800 MHz  CPU7(A57):1800 MHz
       Bus:      600  MHz  DDR:      1866.667 MT/s     DP-DDR:   1600 MT/s
Reset Configuration Word (RCW):
       00: 48303830 48480048 00000000 00000000
       10: 00000000 00200000 00200000 00000000
       20: 01012980 00002580 00000000 00000000
       30: 00000e0b 00000000 00000000 00000000
       40: 00000000 00000000 00000000 00000000
       50: 00000000 00000000 00000000 00000000
       60: 00000000 00000000 00027000 00000000
       70: 412a0000 00000000 00000000 00000000
Model: Freescale Layerscape 2085a RDB Board
Board: LS2085E-RDB, Board Arch: V1, Board version: D, boot from vBank: 4
```

U-Boot Startup Messages

```
DDR    15 GiB (DDR4, 64-bit, CL=13, ECC on)
       DDR Controller Interleaving Mode: 256B
       DDR Chip-Select Interleaving Mode: CS0+CS1
DP-DDR 4 GiB (DDR4, 32-bit, CL=11, ECC on)
       DDR Chip-Select Interleaving Mode: CS0+CS1
Waking secondary cores to start from fff0b000
All (8) cores are up.
Using SERDES1 Protocol: 42 (0x2a)
Using SERDES2 Protocol: 65 (0x41)
Flash: 128 MiB
NAND:  2048 MiB
MMC:   FSL_SDHC: 0
AHCI 0001.0301 32 slots 1 ports 6 Gbps 0x1 impl SATA mode
flags: 64bit ncq pm clo only pmp fbss pio slum part ccc apst
Found 0 device(s).
SCSI:  Net:   crc32+
fsl-mc: Booting Management Complex ... SUCCESS
fsl-mc: Management Complex booted (version: 9.0.4, boot status: 0x1)
e1000: 68:05:ca:36:9c:7c
       DPMAC1@xgmii, DPMAC2@xgmii, DPMAC3@xgmii, DPMAC4@xgmii, DPMAC5@xgmii,
DPMAC6@xgmii, DPMAC7@xgmii, DPMAC8@xgmii, e1000#0 [PRIME]

Hit any key to stop autoboot:  0
```

Linux

- Linux is automatically booting
- If U-Boot countdown has been interrupted, boot Linux with command “boot”
- When Linux booting is complete:
  - Login with user root and no password
  - Configure eth0 to 192.168.1.100

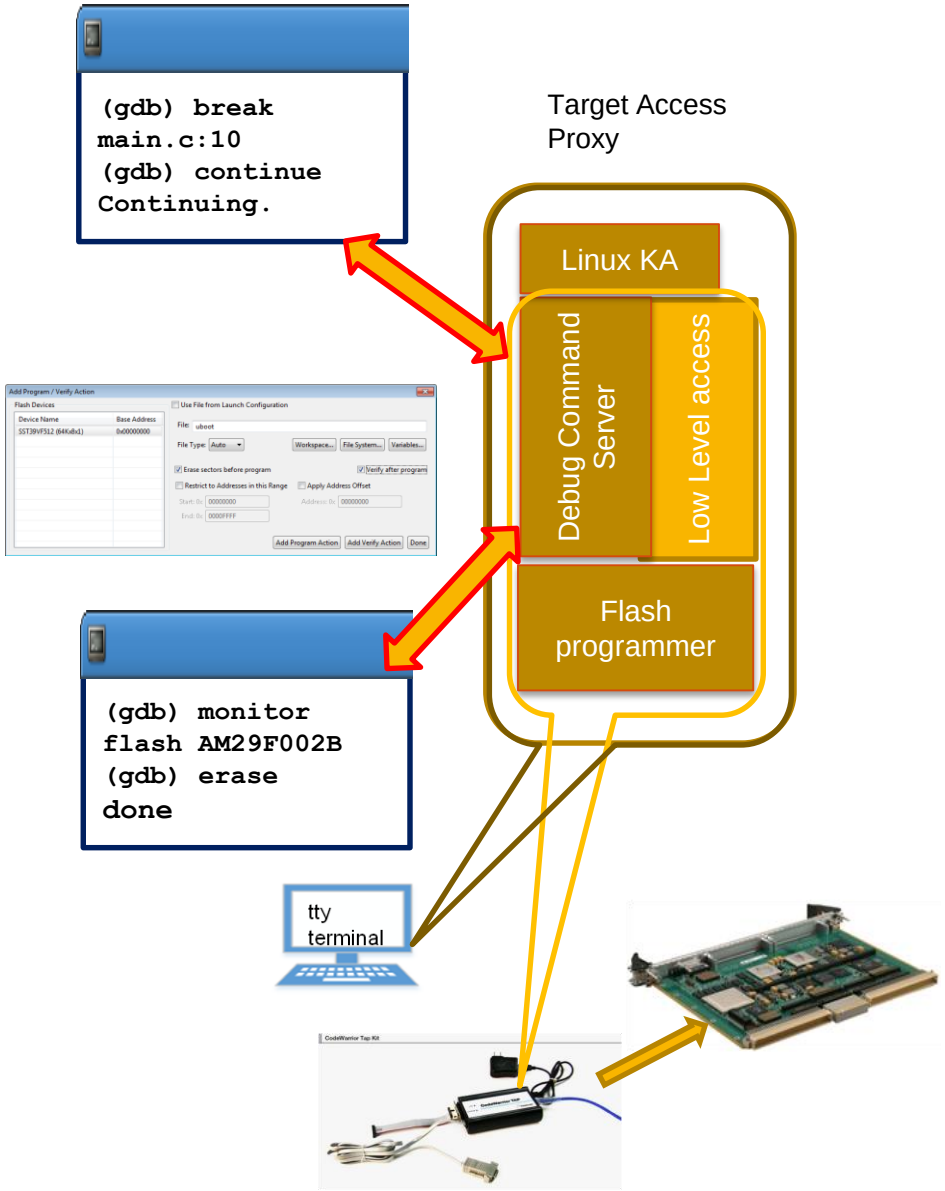
```
INIT: Entering runlevel: 5un-postinsts exists during rc.d purge
Configuring network interfaces... done.
Starting OpenBSD Secure Shell server: sshd
  generating ssh RSA key...
  generating ssh ECDSA key...
  generating ssh DSA key...
Poky (Yocto Project Reference Distro) 1.8.1 ls2085ardb /dev/ttyS1

ls2085ardb login: root
root@ls2085ardb:~# ifconfig eth0 192.168.1.100
root@ls2085ardb:~#
```

BARE-METAL DEBUG

Bare-Metal Debug

- Target interface to real hardware / simulator
- Lightweight debugger engine accessible from both GUI and command line
- Compatible with the GNU debugger front-end
- Standard set of memory/register access commands + monitor extensions
- Simultaneous connectivity with multiple clients
- Single- and Multicore support



ACTIVITY

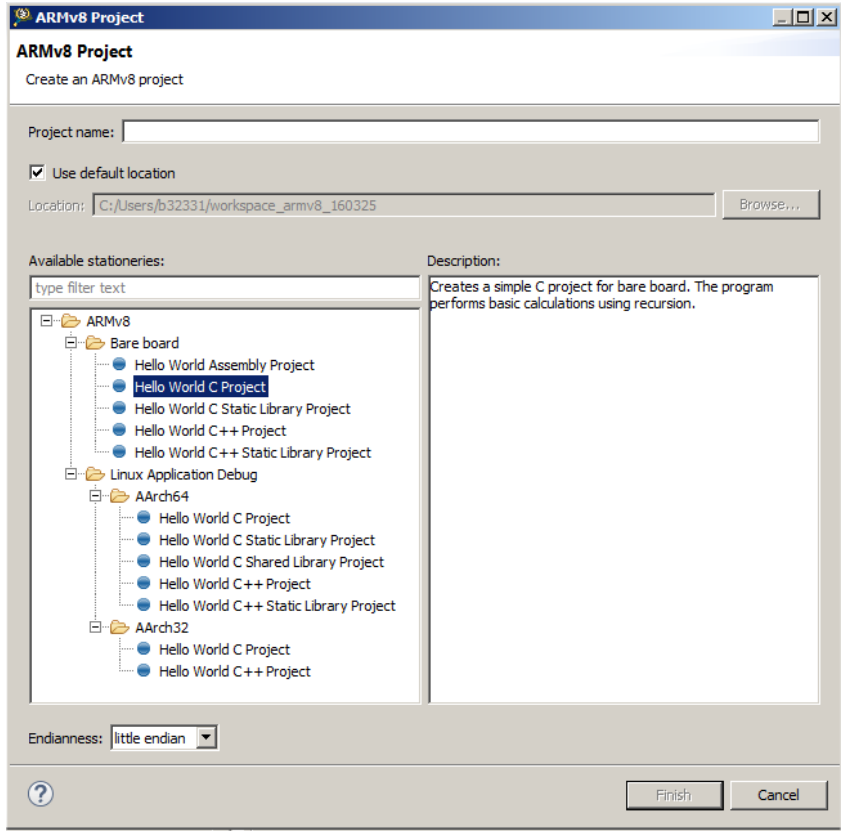
Create a CodeWarrior Bareboard Project

Activity – CodeWarrior Bareboard Debugging

Bareboard Debugging with CodeWarrior TAP Connection

Activity Summary:

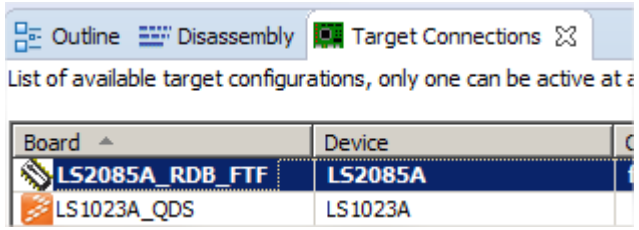
1. Launch CodeWarrior-ARMv8 from the desktop
2. Go to Workbench
3. ***File → New → ARMv8 Stationery***
4. Enter a ***Project Name***
5. Choose:
   - Bare board project type (ASM, C, C++)
6. Press ***Finish***

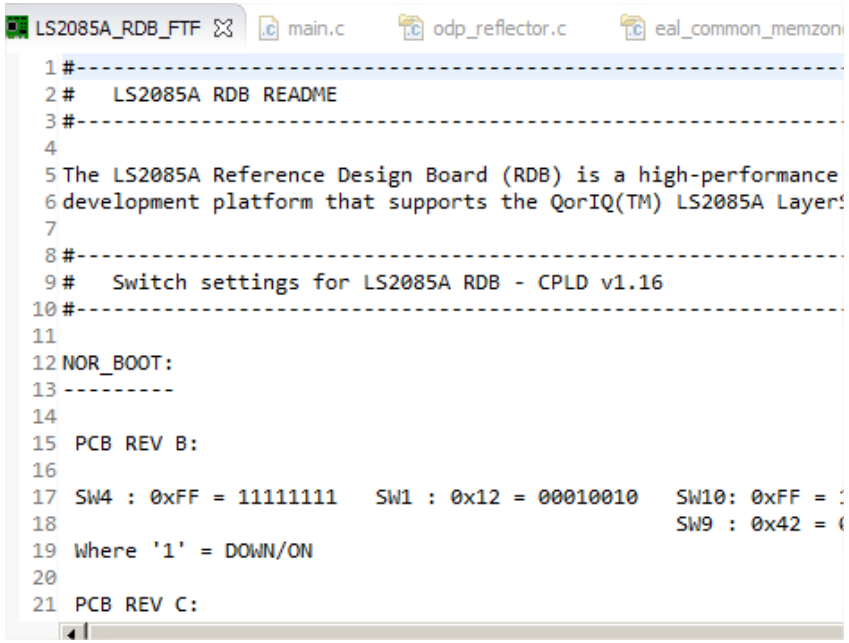


Create Bareboard Project – Activity (cont'd)

The project created can be used on any LS2 SoC and board supported:

- For any information about board hardware setup or project options please check the README section from Target Connection (double-click)





- You will find out about:
  - ***Boot*** options switch configurations
  - Default ***RCW***
  - ***Memory Map*** with accessible areas
  - ***SMP*** debug configuration
  - ***Console I/O*** or ***UART I/O*** selection
  - Other useful info

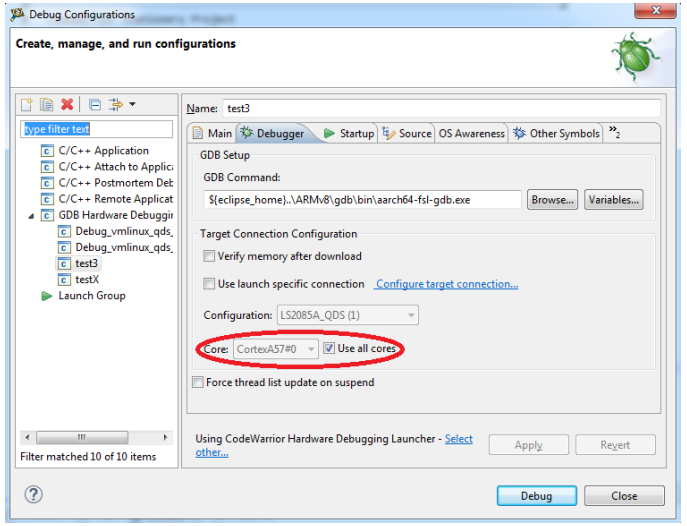
Create Bareboard Project – Activity (cont'd)

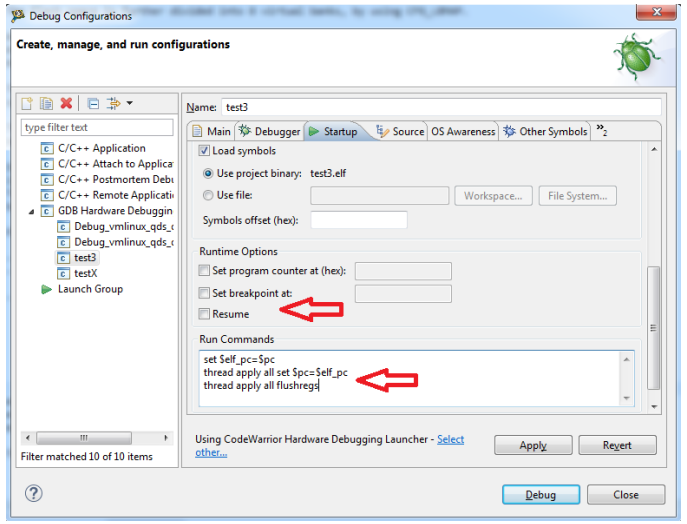
Activating **SMP**:

- Go to Debug Configuration
- Select *Debugger* tab
- Make sure "***Use all cores***" is checked

- Go to *Startup* tab
- Uncheck "***Set breakpoint at***" and "***Resume***" in the *Runtime Options* section

- Enter the following commands in the *Run Commands* section:
  - set $self_pc=$pc
  - thread apply all set $pc=$self_pc
  - thread apply all flushregs



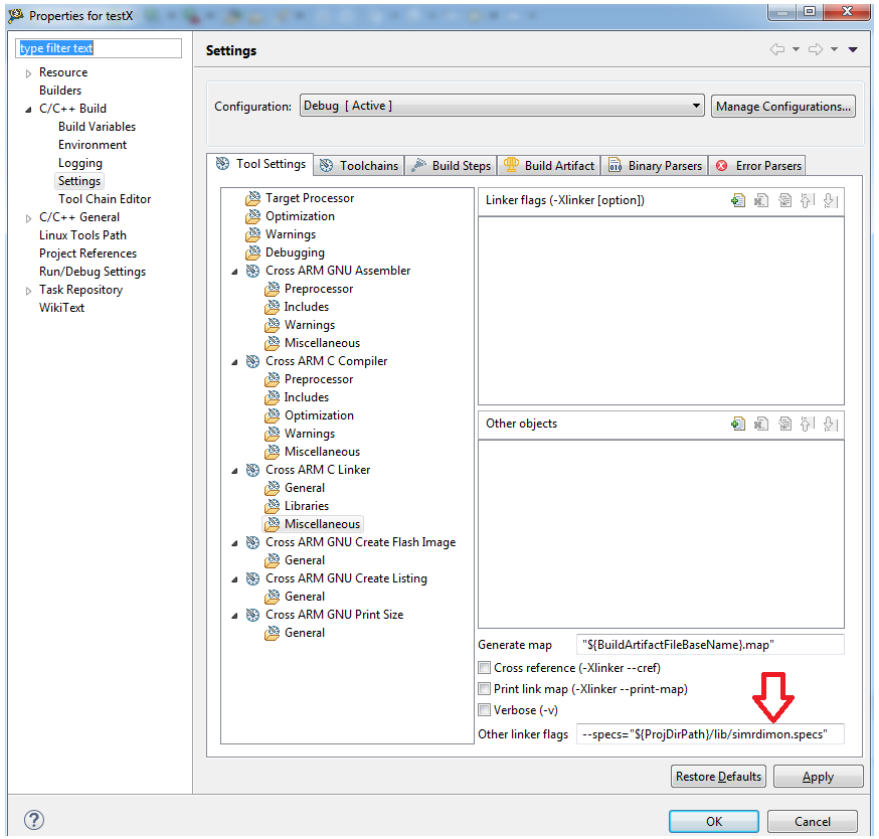


Create Bareboard Project – Activity (cont'd)

The default **I/O mode** is debugger console (in other words the *simrdimon* library is being used). You can change to **UART** I/O if needed by modifying the project build settings: navigate to

C/C++ Build -> Settings -> Cross ARM C (or C++) Linker -> Miscellaneous

- In *Other linker flags* section
- Replace *simrdimon.specs* with *uart.specs*
- Re-build the project



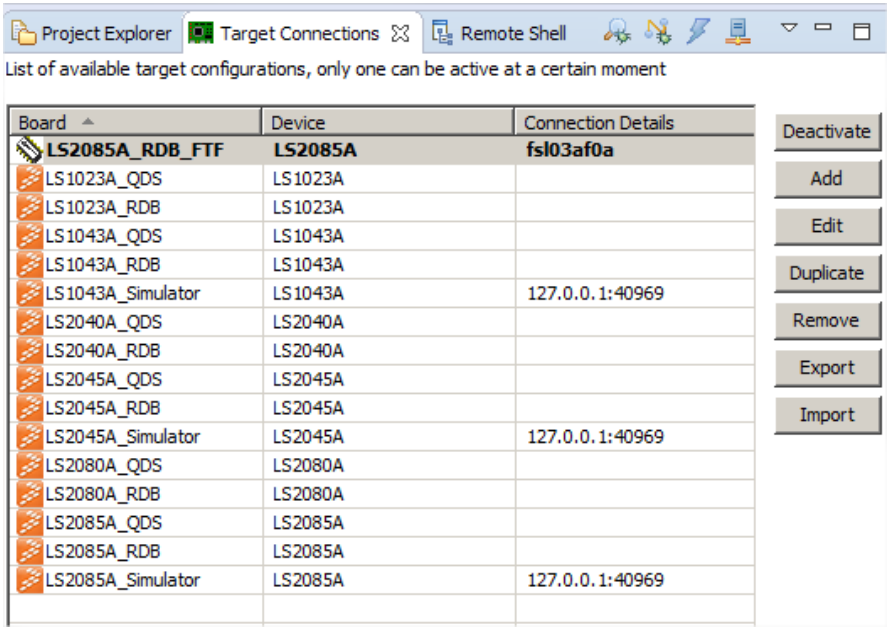
ACTIVITY

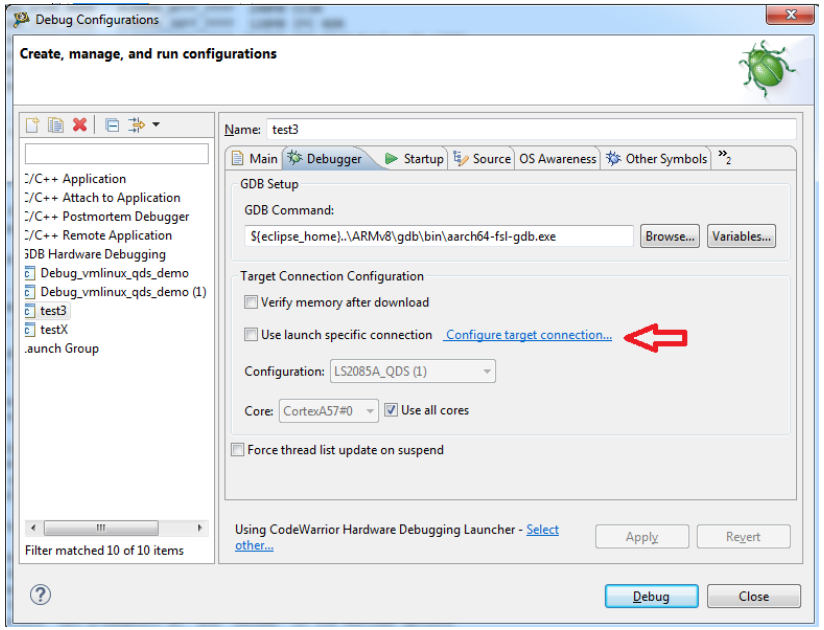
Establish a CodeWarrior Connection to the Target

Hardware Selection and Configuration Activity

Target Connection Configurator (TCC)

- For connection settings, target SoC and board selection go to:
- *Left side in C/C++ perspective or Right side in the Debug perspective* or
- *Debug Configurations -> Debugger -> Configure target connection*





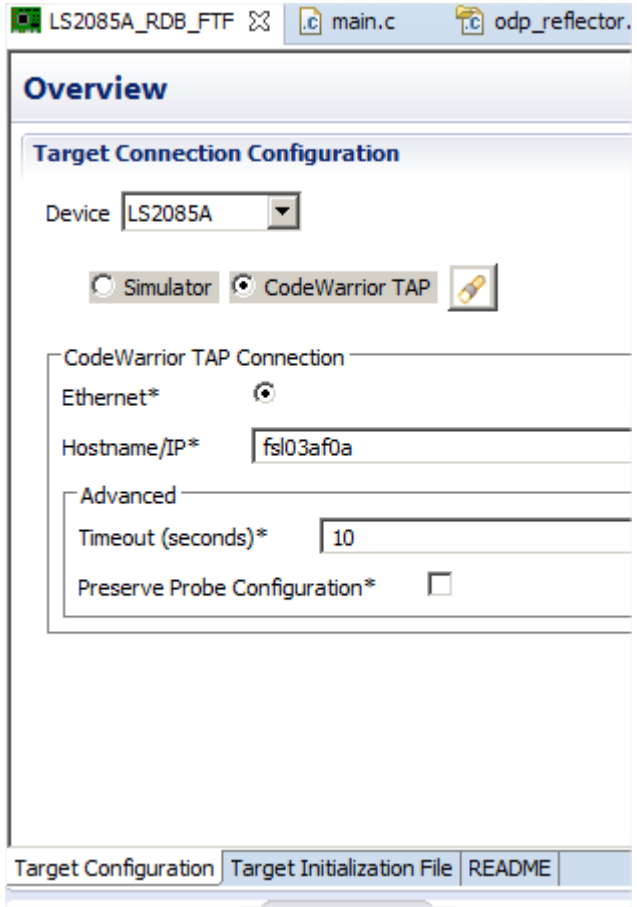
Hardware Selection and Configuration Activity

Target Connection Configurator (TCC) (cont'd)

- To select the SoC & Board configuration simply double-click one of the pre-defined templates ( ) or select one and click *Duplicate* or *Edit* to create your own configuration ( )

- You can change:
  - **device** type
  - **connection** settings
  - **debug probe** type

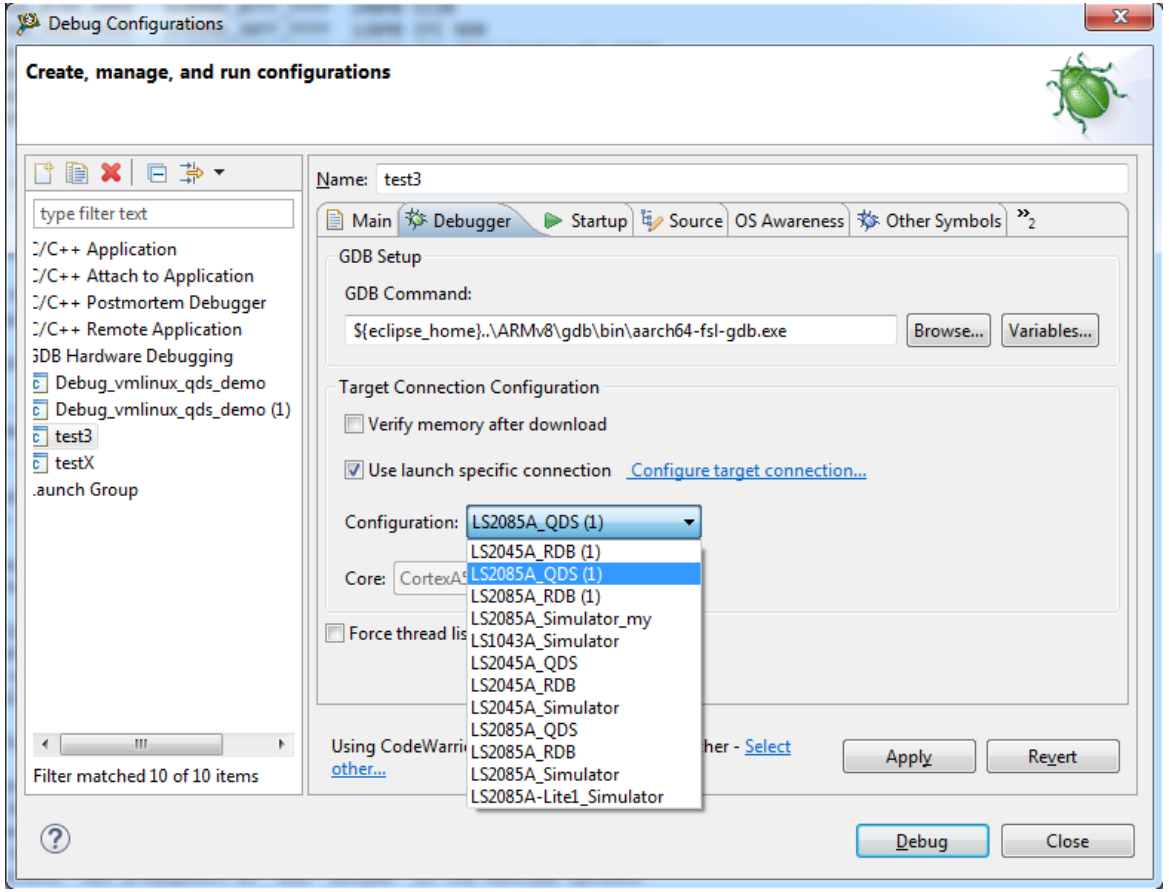
- Or tweak the **initialization file**



Hardware Selection and Configuration Activity

Target Connection Configurator (TCC) (cont'd)

- Selected configuration in TCC view will be the default one for all projects (launches), expect if *Use launch specific connect* is checked.
- In this case, any existing configuration can be selected:



ACTIVITY

Diagnose a CodeWarrior Connection to the Target

Diagnose a CodeWarrior Connection to the Target

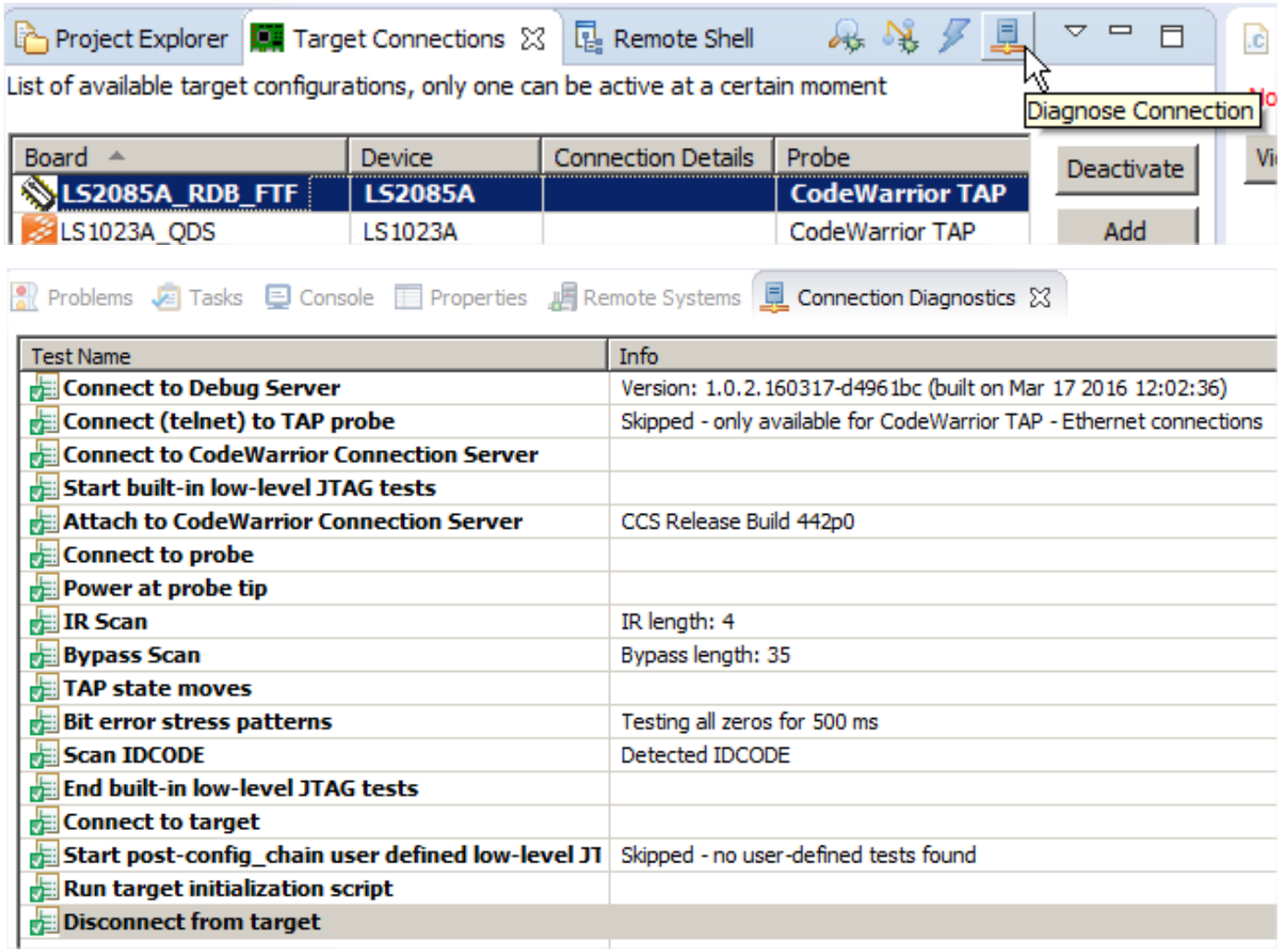
Summary

- Run a Diagnose check on the existing connection
- Introduce different failures in the connection (software configuration, disconnect the CWTAP USB cable, power-off board)
- For each scenario run Diagnostic tests, view the results and the suggested corrective actions

Diagnose Connection Feature

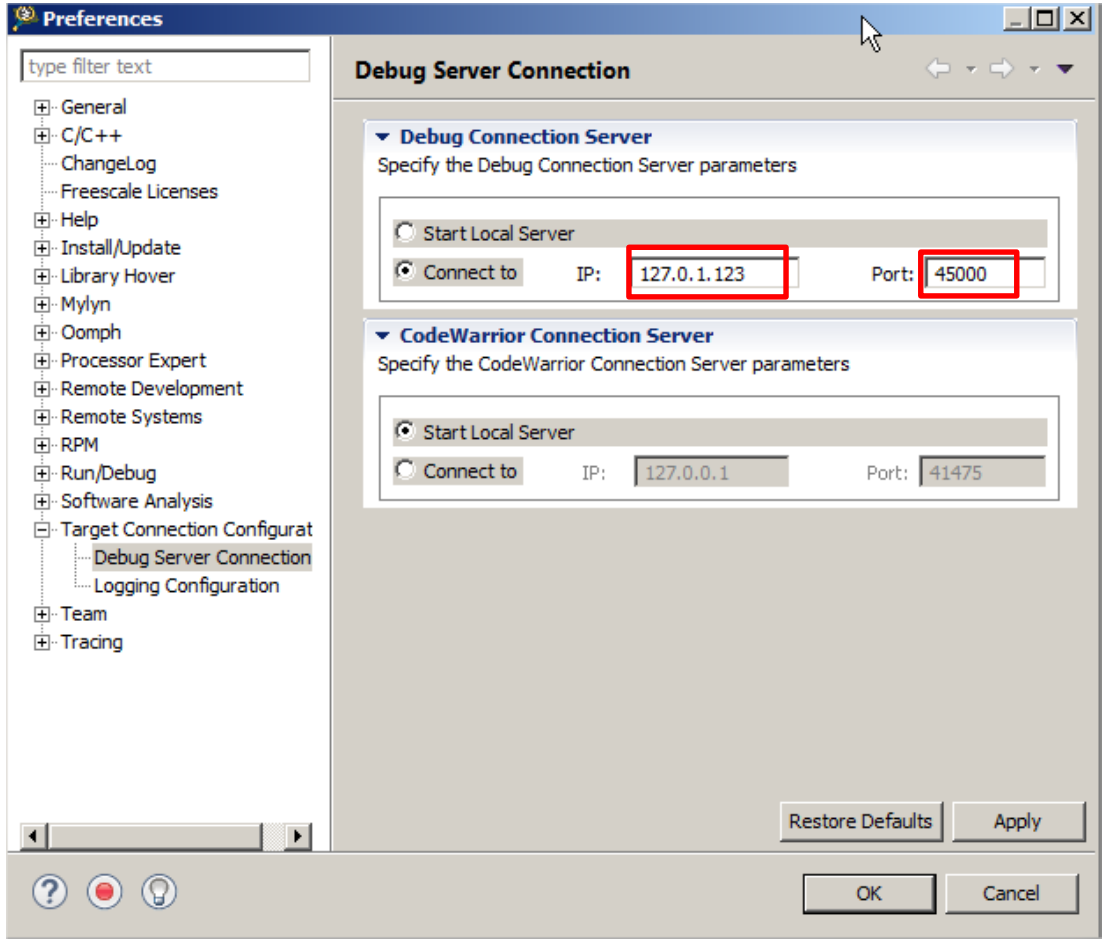
Check if the new created TCC is reliable

- Select Diagnose Connection button from TCC

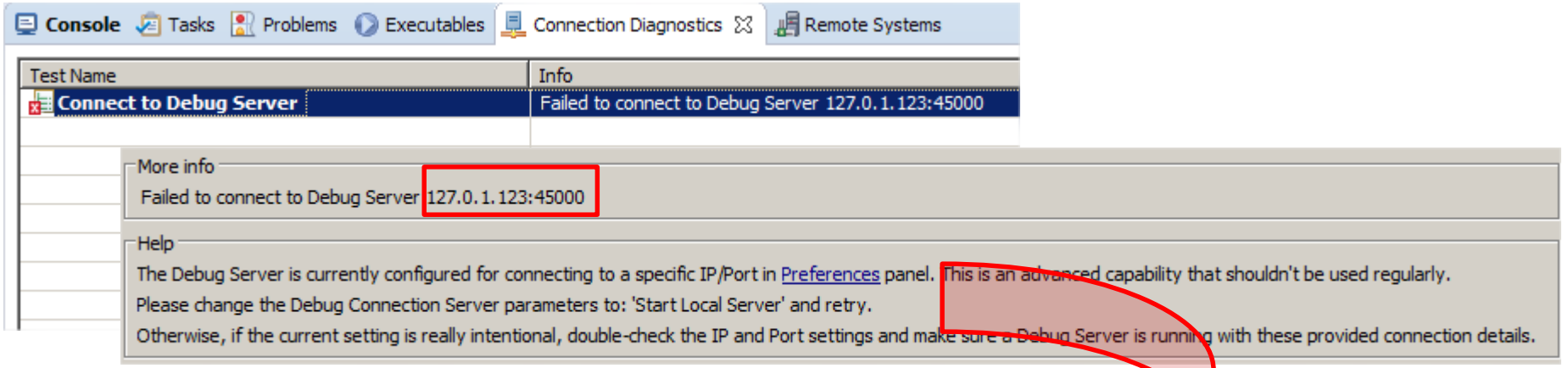


Connect to Debug Server (1)

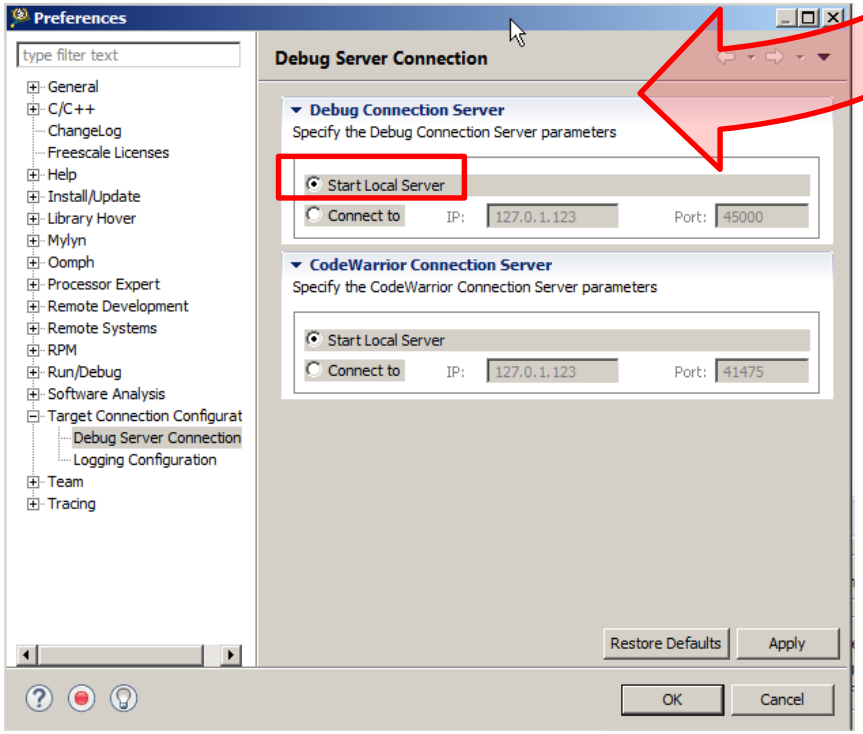
Introduce a failure – go to Window > Preferences > Target Connection Configuration > Debug Server Connection and change from Start Local Server to a custom IP/port as bellow



Connect to Debug Server (2)

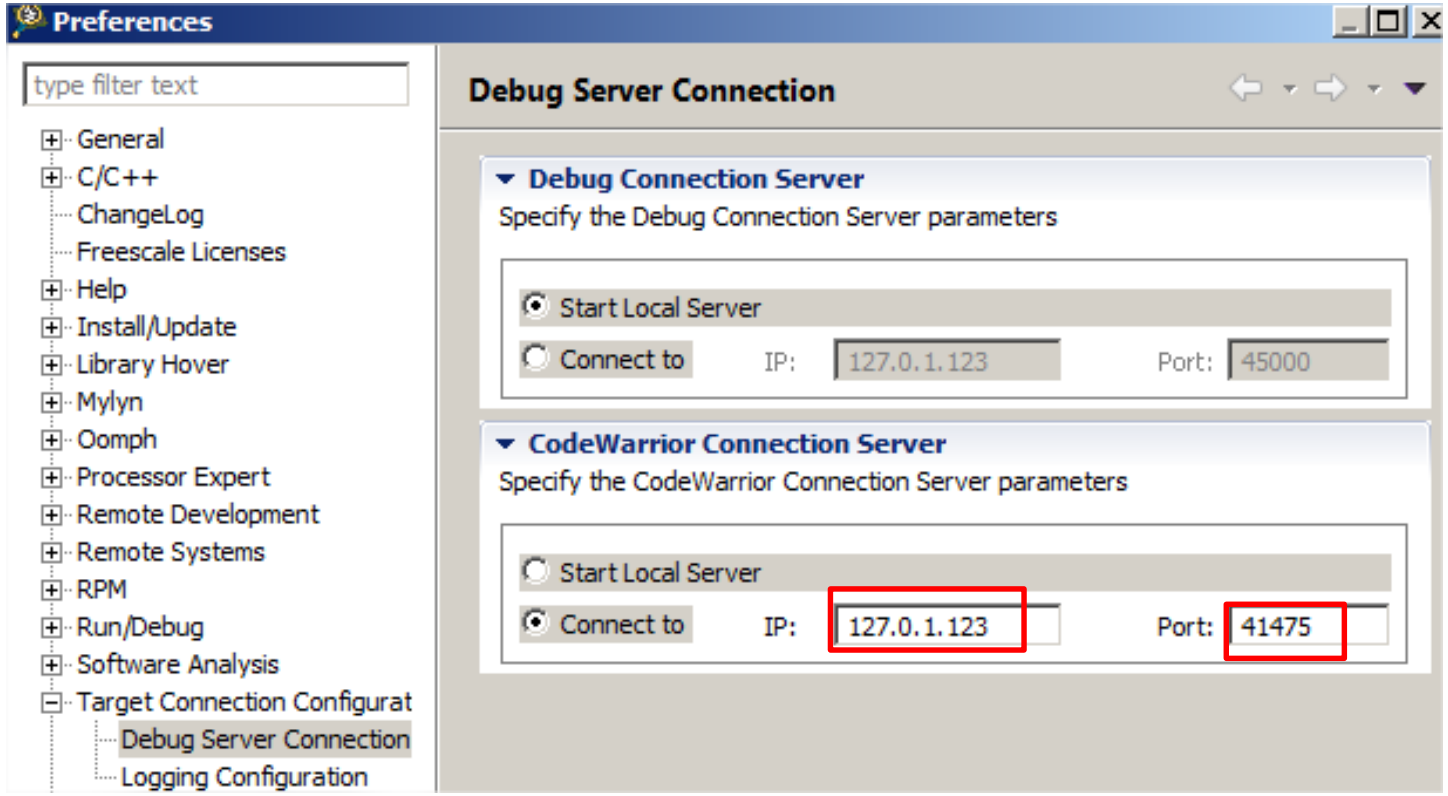


Check the Debug Server settings in the Preferences Panel (switch back to defaults)

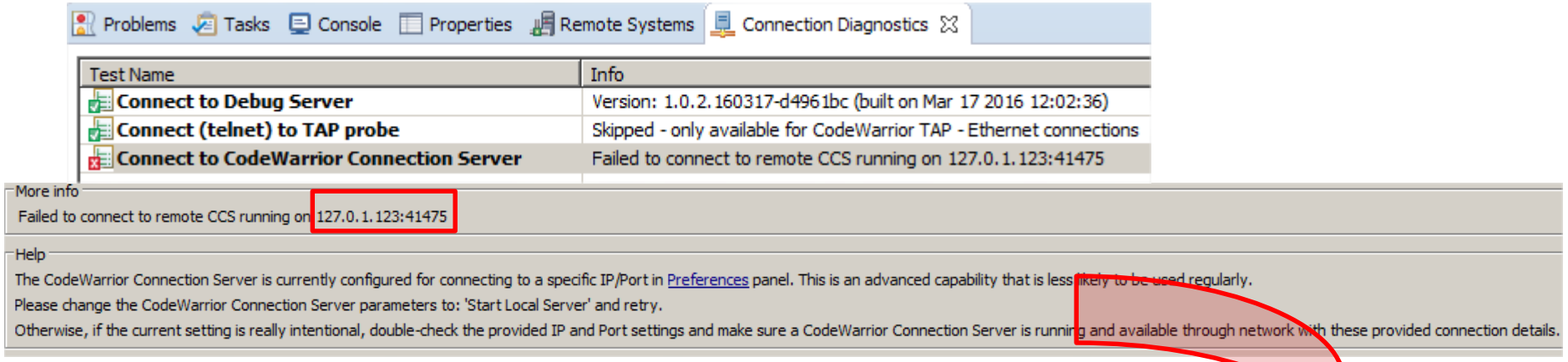


Connect to CodeWarrior Connection Server (1)

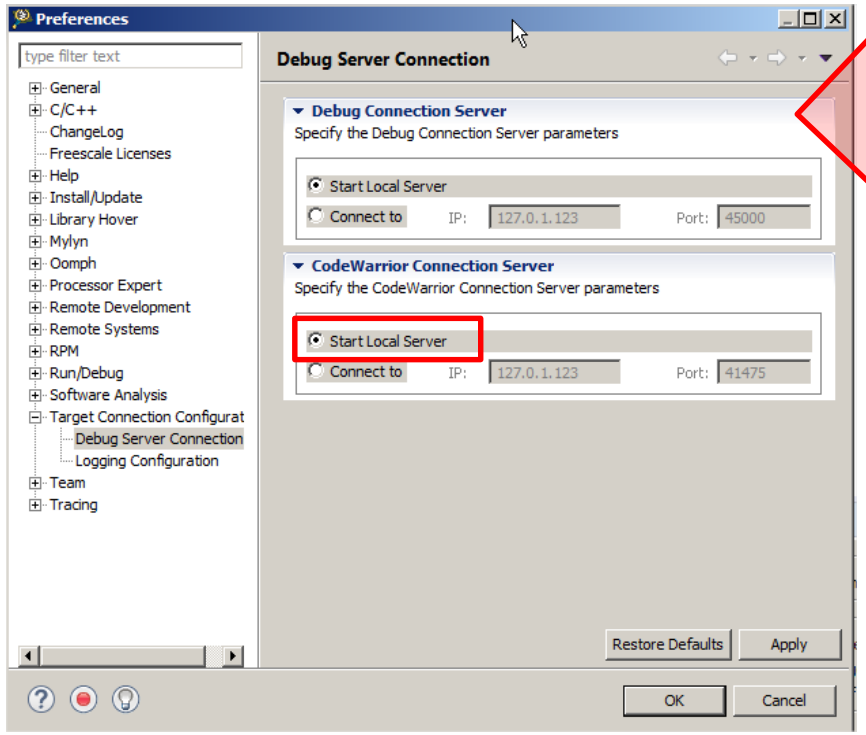
Introduce a failure – go to Window > Preferences > Target Connection Configuration > CodeWarrior Connection Server and change from Start Local Server to a custom IP/port as bellow



Connect to CodeWarrior Connection Server (2)



Check the Debug Server settings in Preferences Panel (switch back to defaults)



Connect to Probe (1)

Introduce a failure – power down the cwtap probe

Connect to Probe (2)

Problems | Tasks | Console | Properties | Remote Systems | Connection Diagnostics

Test Name	Info
Connect to Debug Server	Version: 1.0.2.160317-d4961bc (built on Mar 17 2016 12:02:36)
Connect (telnet) to TAP probe	Skipped - only available for CodeWarrior TAP - Ethernet connections
Connect to CodeWarrior Connection Server	
Start built-in low-level JTAG tests	
Attach to CodeWarrior Connection Server	CCS Release Build 442p0
Connect to probe	Could not connect to probe cwtap:

More info

Could not connect to probe cwtap:

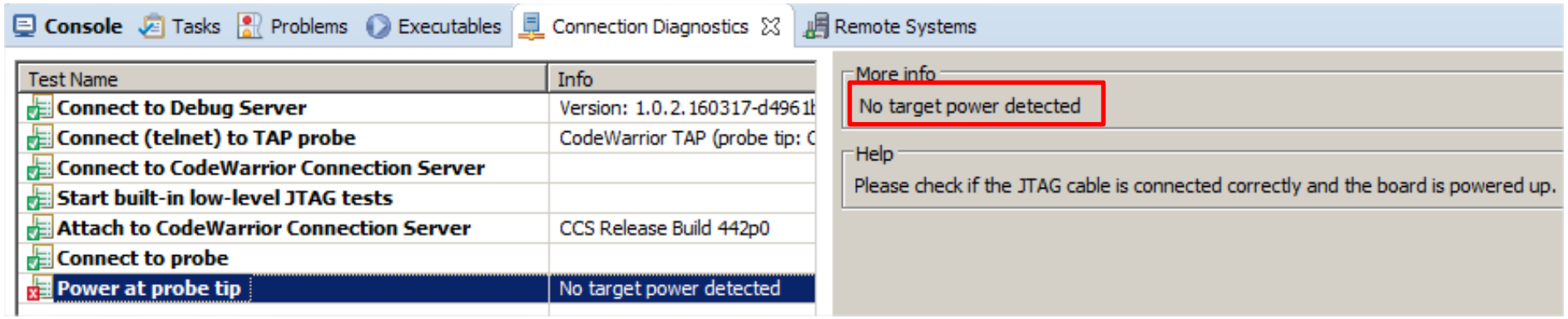
Help

- Check if the probe is properly connected
- Check probe settings (IP, Serial number) in Target Connection Configuration
(You can use "Discover probes" to detect available probes)
- Increase Timeout value
- Reset probe

Connect back the USB cable to CWTAP probe to get rid of the failure.

Power at Probe Tip

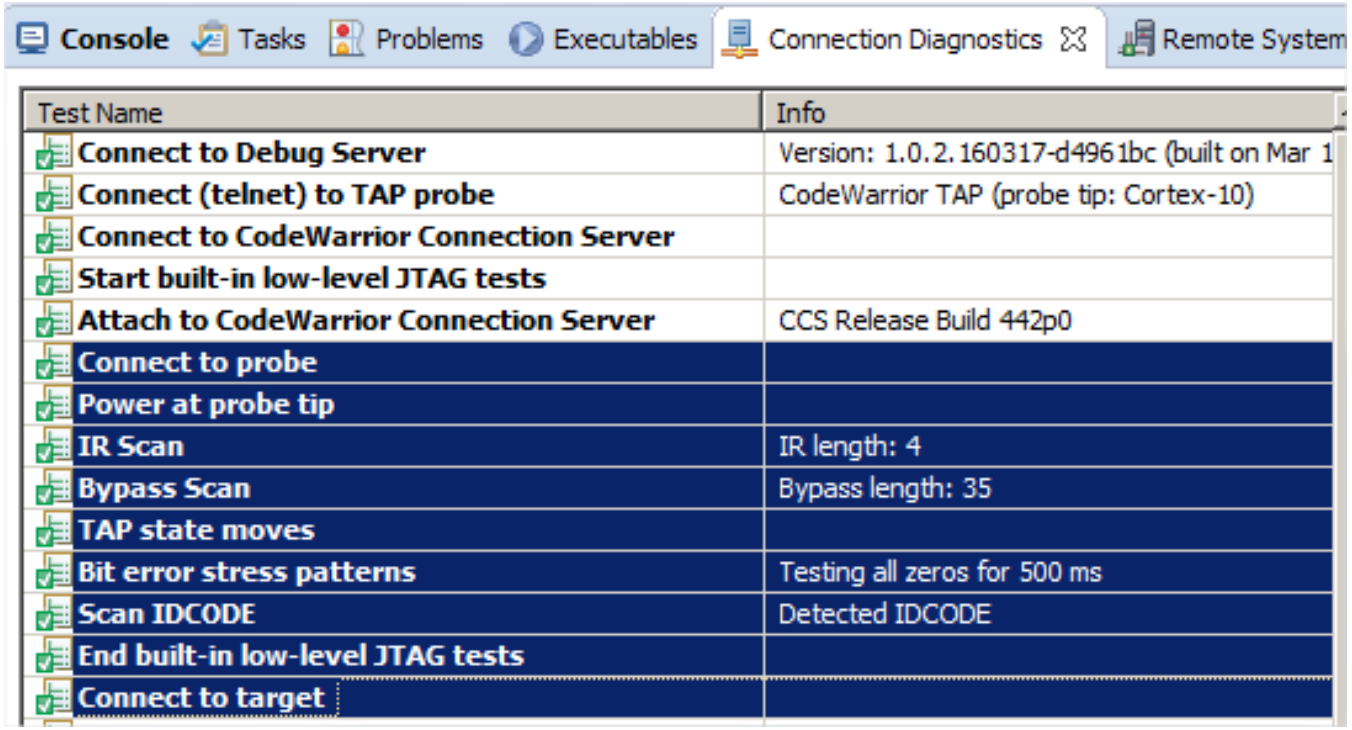
Introduce a failure – power down the LS2085ARDB board using the PWR button



To get rid of the problem, just power up the board again

Run Low-level JTAG Tests to Check Probe JTAG Reliability

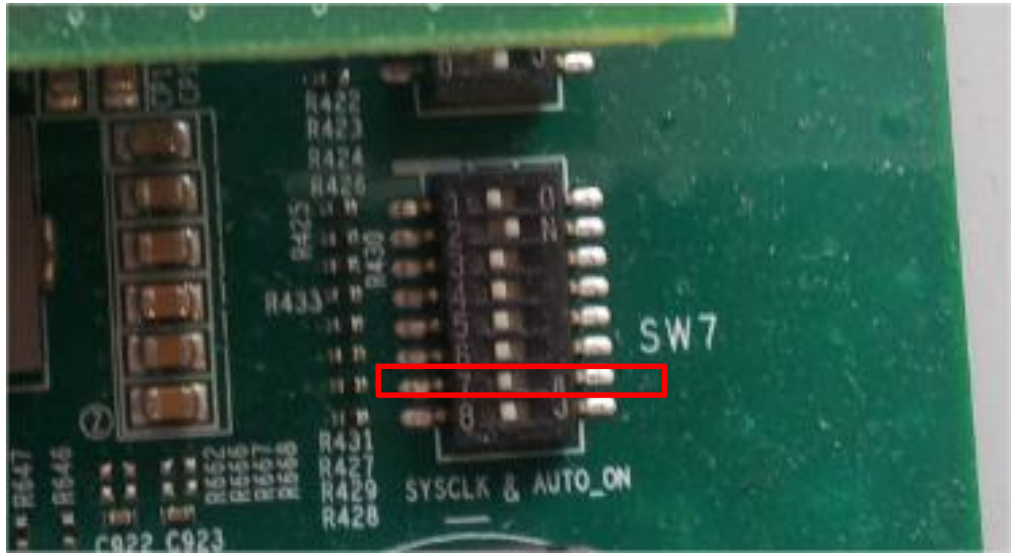
These tests are verifying if the JTAG is reliable using different low-level JTAG tests and also will “scan” the SoC after IDCODE to see if is ARM specific



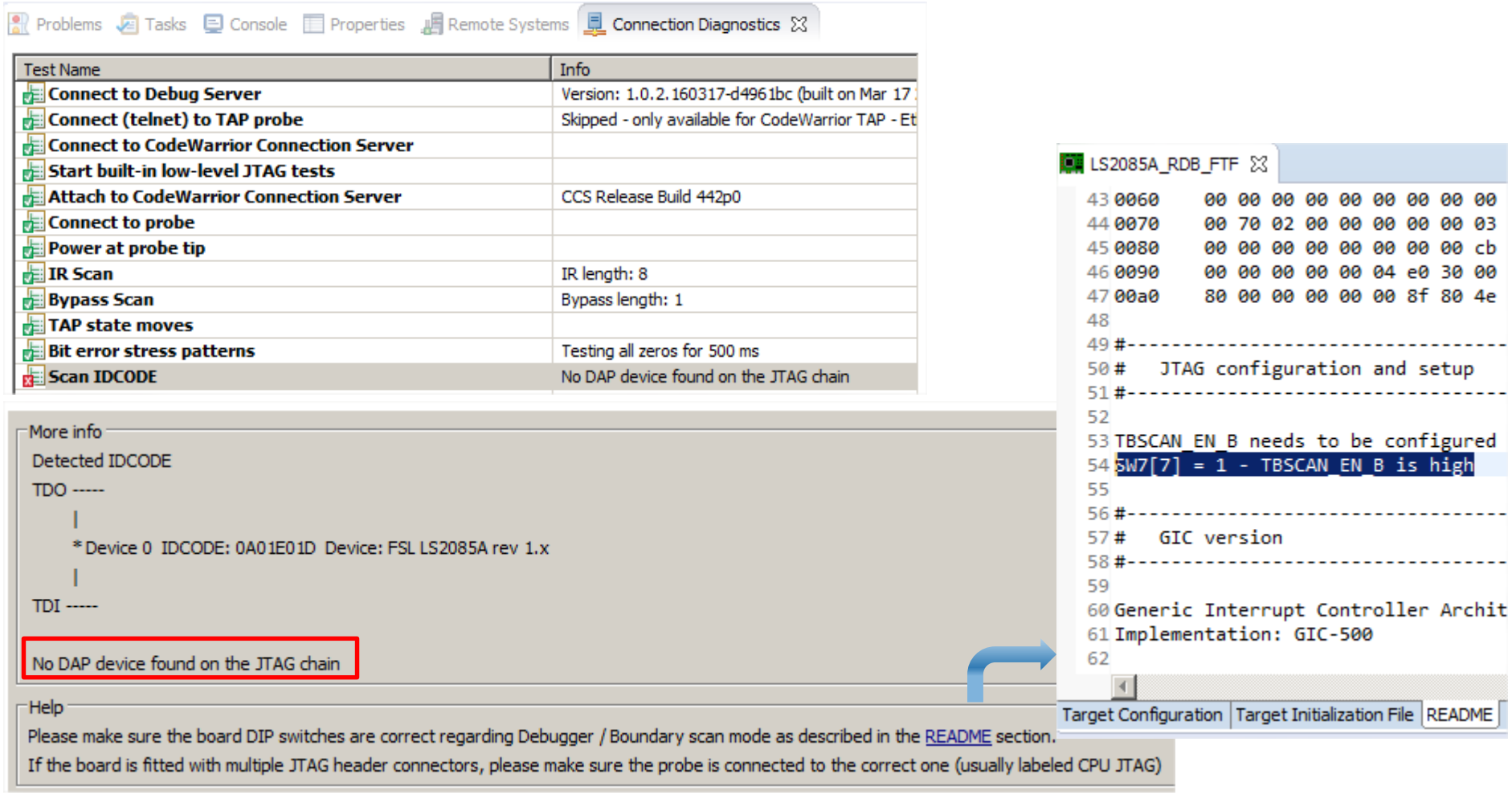
Test Name	Info
Connect to Debug Server	Version: 1.0.2.160317-d4961bc (built on Mar 1
Connect (telnet) to TAP probe	CodeWarrior TAP (probe tip: Cortex-10)
Connect to CodeWarrior Connection Server	
Start built-in low-level JTAG tests	
Attach to CodeWarrior Connection Server	CCS Release Build 442p0
Connect to probe	
Power at probe tip	
IR Scan	IR length: 4
Bypass Scan	Bypass length: 35
TAP state moves	
Bit error stress patterns	Testing all zeros for 500 ms
Scan IDCODE	Detected IDCODE
End built-in low-level JTAG tests	
Connect to target	

Scan IDCODE (1)

Introduce a failure – change the default ARM DAP (Debug Access Point) mode to NXP SAP (Service Access Point) setting SW7[7] = 0 (OFF)



Scan IDCODE (2)



To get rid of the problem change back to default ARM DAP (Debug Access Point) setting SW7[7] = 1 (ON)

ACTIVITY

Flashing U-Boot Image to the Target with CW Flash

FLASH PROGRAMMER

Flashing U-Boot Image to the Target with CW Flash

Summary

- Run Flash Programmer GUI
- Write U-Boot binary into NOR flash

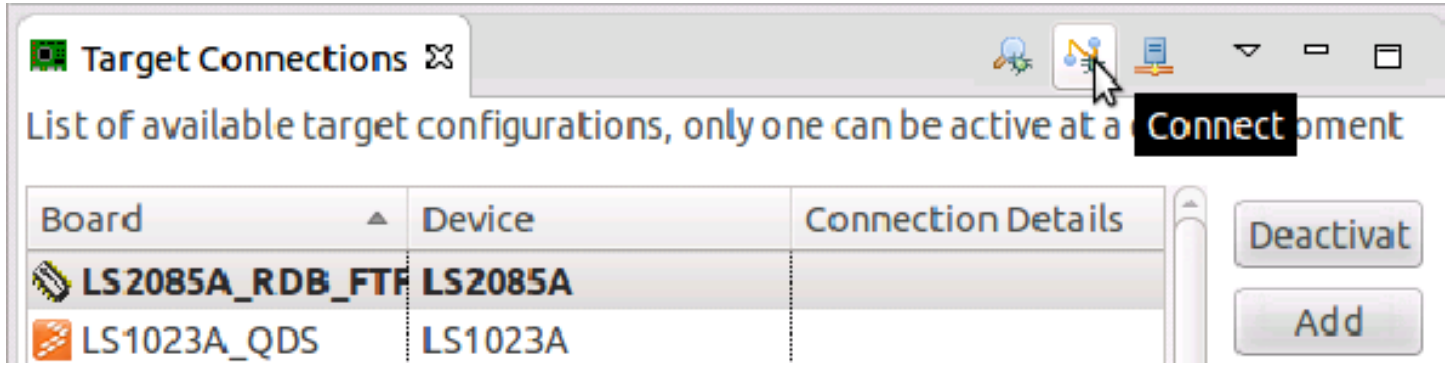
Flashing u-boot image to the Target Activity (1)

There are possible 3 ways to flash something in flash

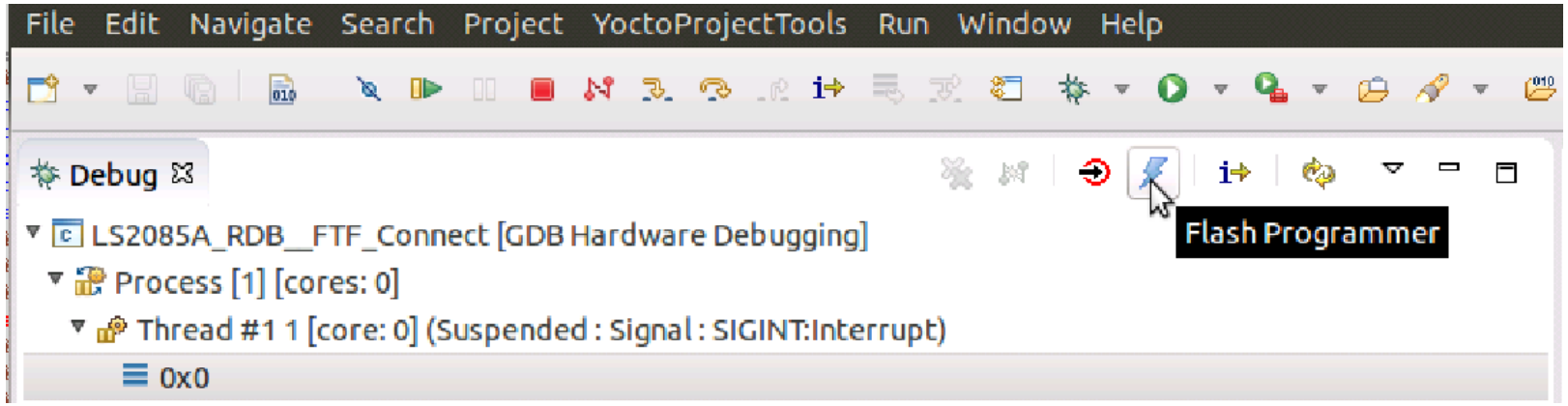
1. ***Flash Programmer GUI***
2. ***GDB CLI***
3. ***GDB Eclipse***

To use Flash Programmer GUI:

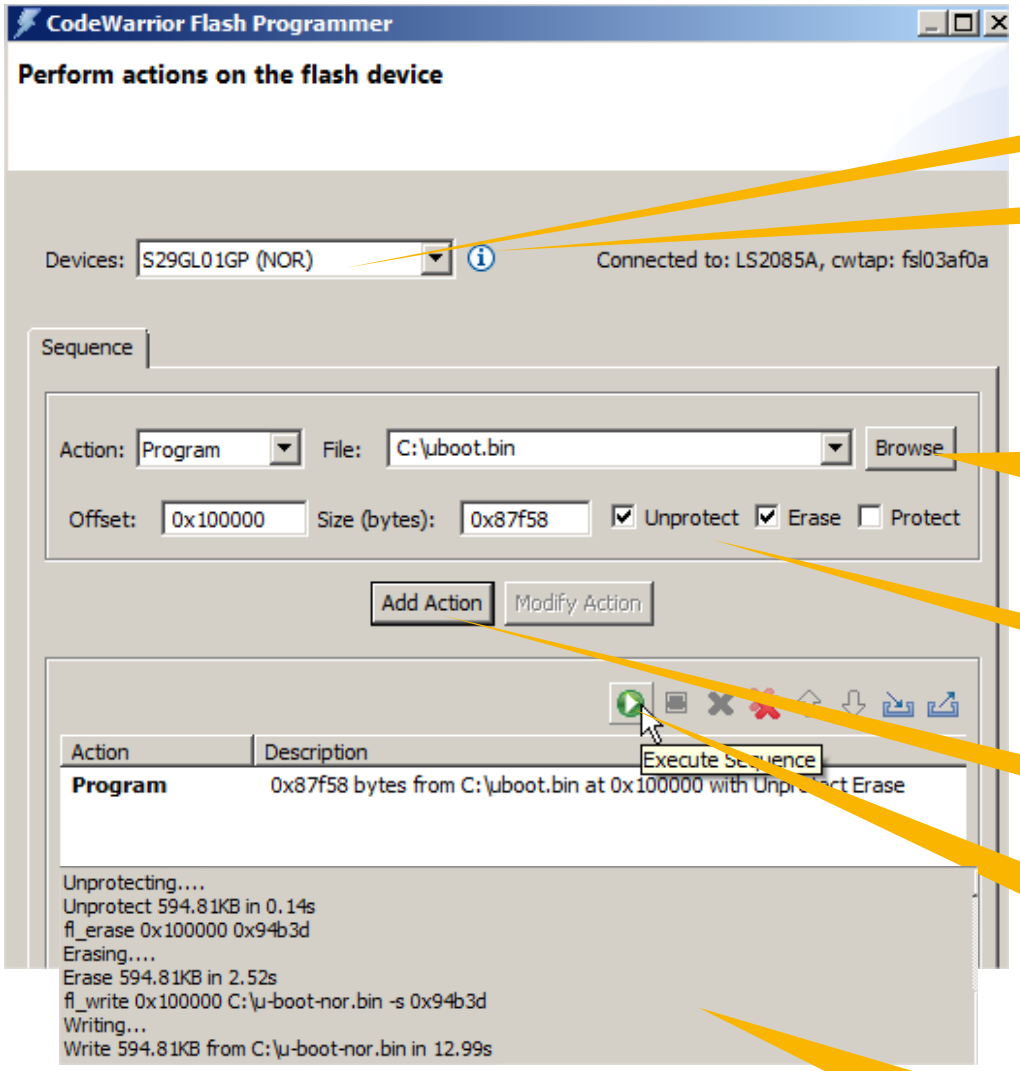
A. connect to target



B. Click Flash Programmer icon



Flashing U-Boot Image to the Target Activity (2)



Drop-down list with all supported flashes

Hover-information about current flash

Device Info
base address: 0x580000000
size: 0x8000000 (128.00MB)
sectors: 1024
sector size: 0x20000 (128.00KB)
geometry: 16x1
features: erase;erase_chip;dump;write;protect;unprotect;

Browse for U-Boot image from /home/class/SDK/LS2085A-SDK-20160304-yocto/build_ls2085ardb_release\tmp\deploy\images\ls2085ardb\u-boot-nor.bin

Check Erase and Unprotect

Add action

Execute it

Results in FP log

Using Flash from Command Line

1. Edit *CW_ARMv8/ARMv8/gdb_extensions/flash/cwflash.py* with your board and connection settings

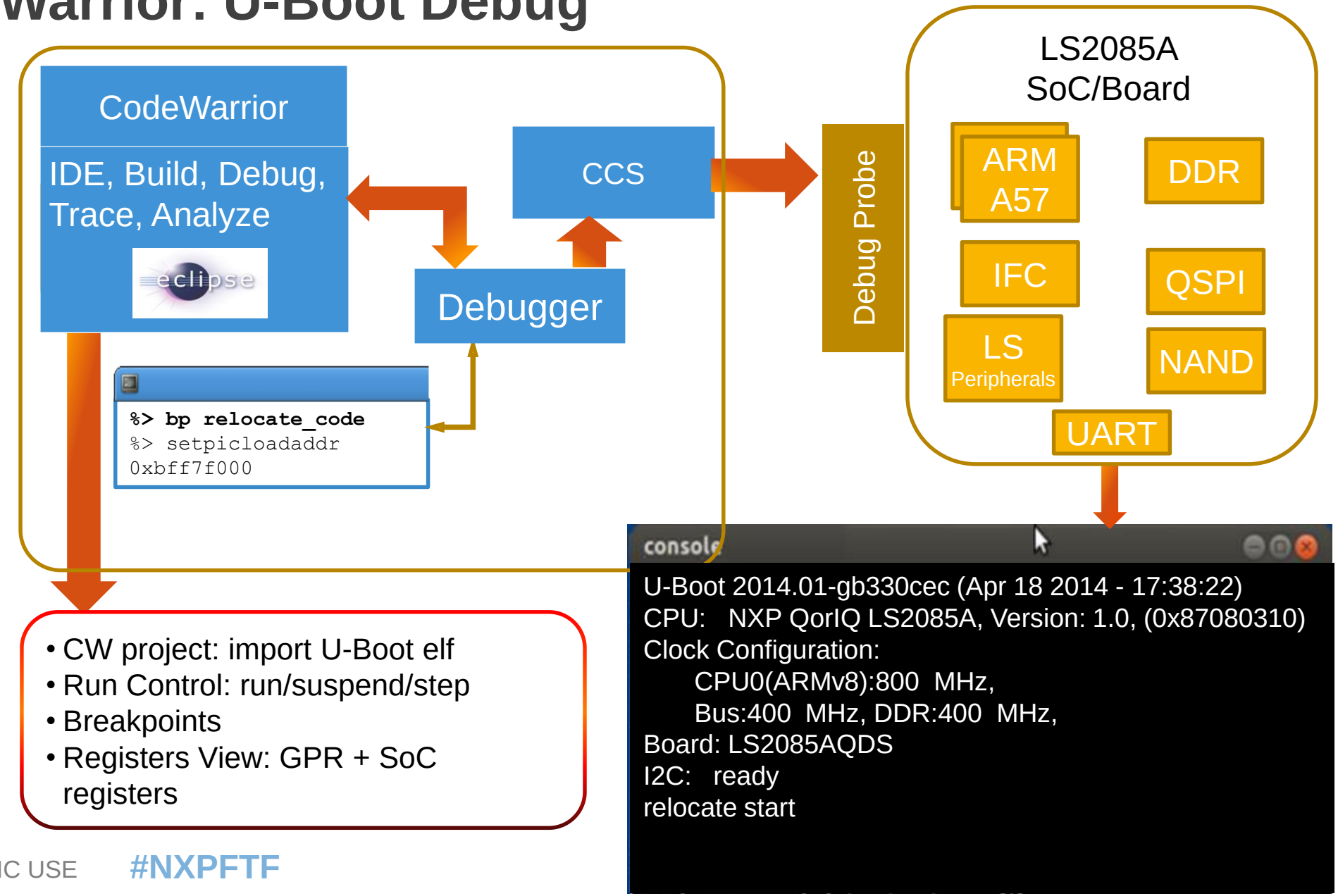
2. Start GDB console from *CW_ARMv8/ARMv8/gdb/bin/aarch64-fsl-gdb.bat*
   - *cd ../../gdb_extensions*
   - *source flash/cwflash.py*

3. Issue following command:
   - *fl_write --erase 0x100000 {u-boot_image_path}*

   - Wait a few seconds for the confirmation message
   - You are ready to debug U-Boot

U-BOOT DEBUG

CodeWarrior: U-Boot Debug



- CW project: import U-Boot elf
- Run Control: run/suspend/step
- Breakpoints
- Registers View: GPR + SoC registers

CodeWarrior: U-Boot Debug

- U-Boot bring-up and debugging
  - Import U-Boot ELF with symbol information
  - Debug from first U-Boot instruction (in flash)
  - Debug after U-Boot relocation in ram / relocate symbols
  - Debug to console prompt
  - Debug to kernel hand-off

- Registers View: GPR + SoC registers

- Debugging features:
  - Run control run/suspend/step
  - Breakpoints, in any ARMv8 EL mode
  - Disassembly, Memory view, Variable View, Expressions

- **Prerequisite**
  - U-Boot image (optionally with symbolic information, useful for source level debug)

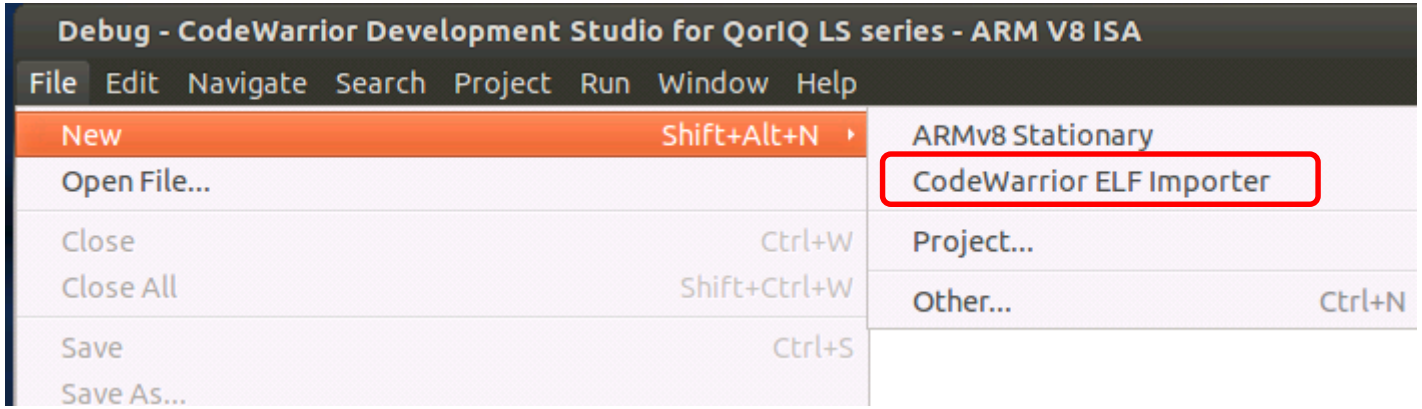
ACTIVITY

U-Boot – Debug

U-Boot Debug – Create Project – Activity

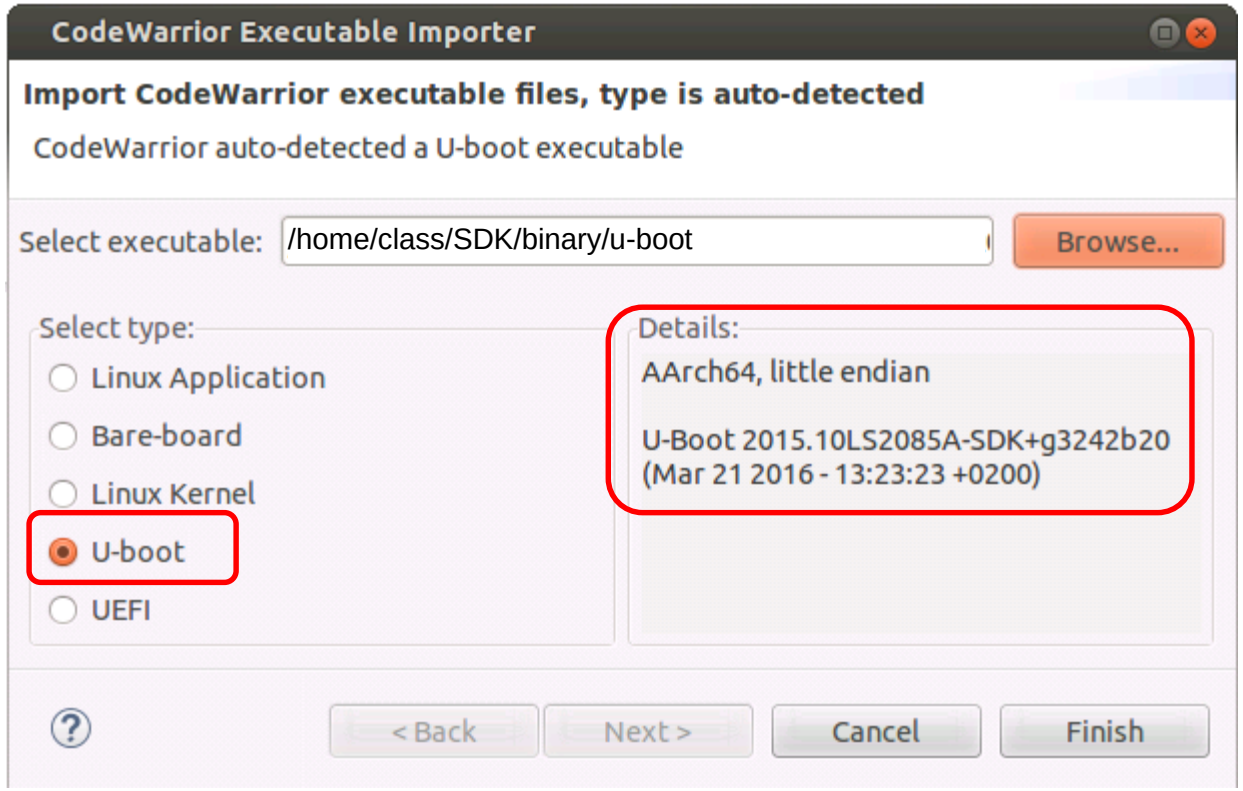
Select File > New > CodeWarrior ELF Importer

- Automatically detects the ELF type and applies the correct awareness settings



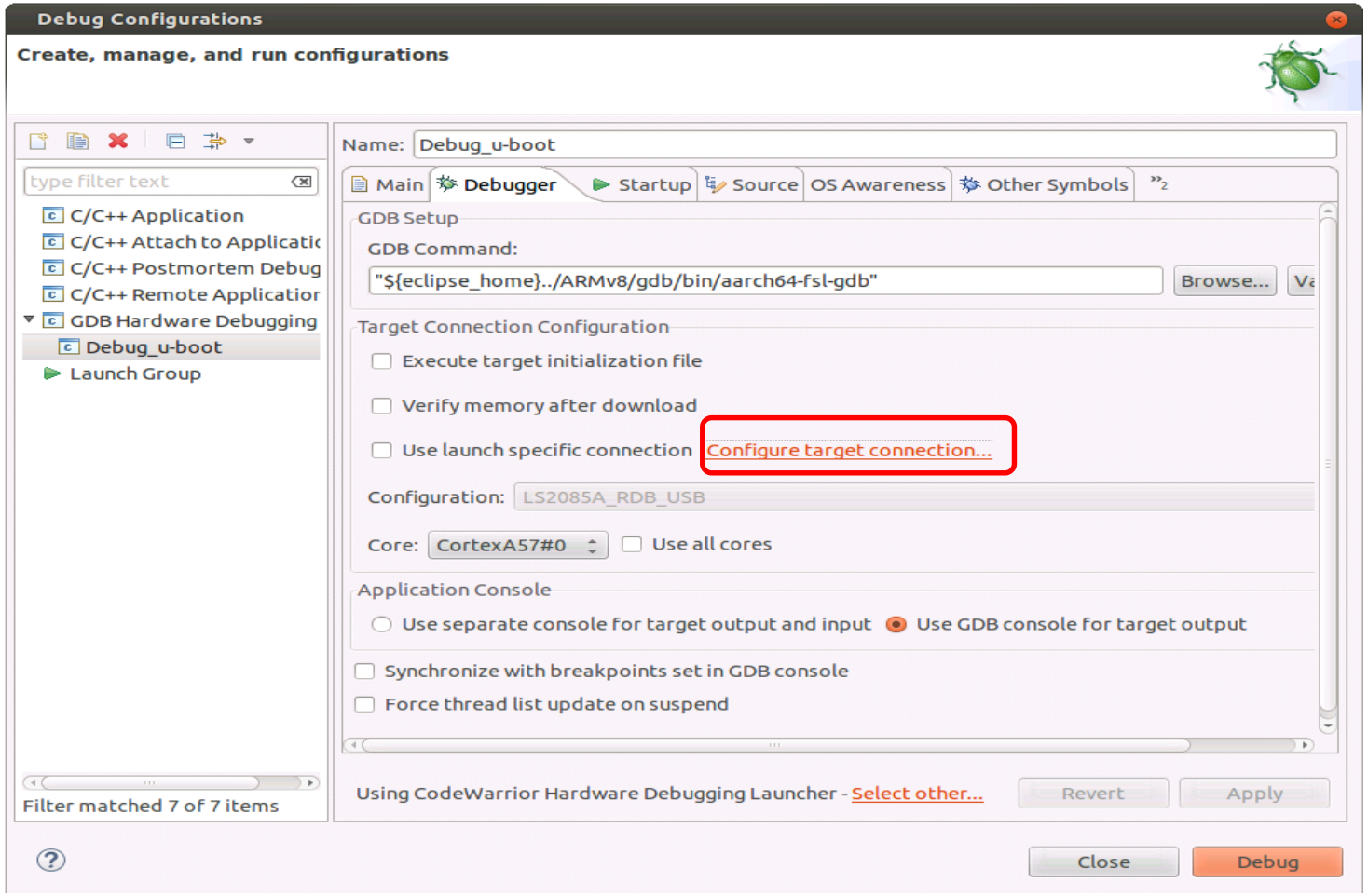
U-Boot Debug – Create Project – Activity

- Click Browse…
- Select your U-Boot file. The file must have debug information.
- CodeWarrior automatically:
  - Detect the ELF type as U-Boot
  - Show U-Boot summary information: U-Boot version, build time, configuration
  - Apply the right debugger settings for debugging U-Boot



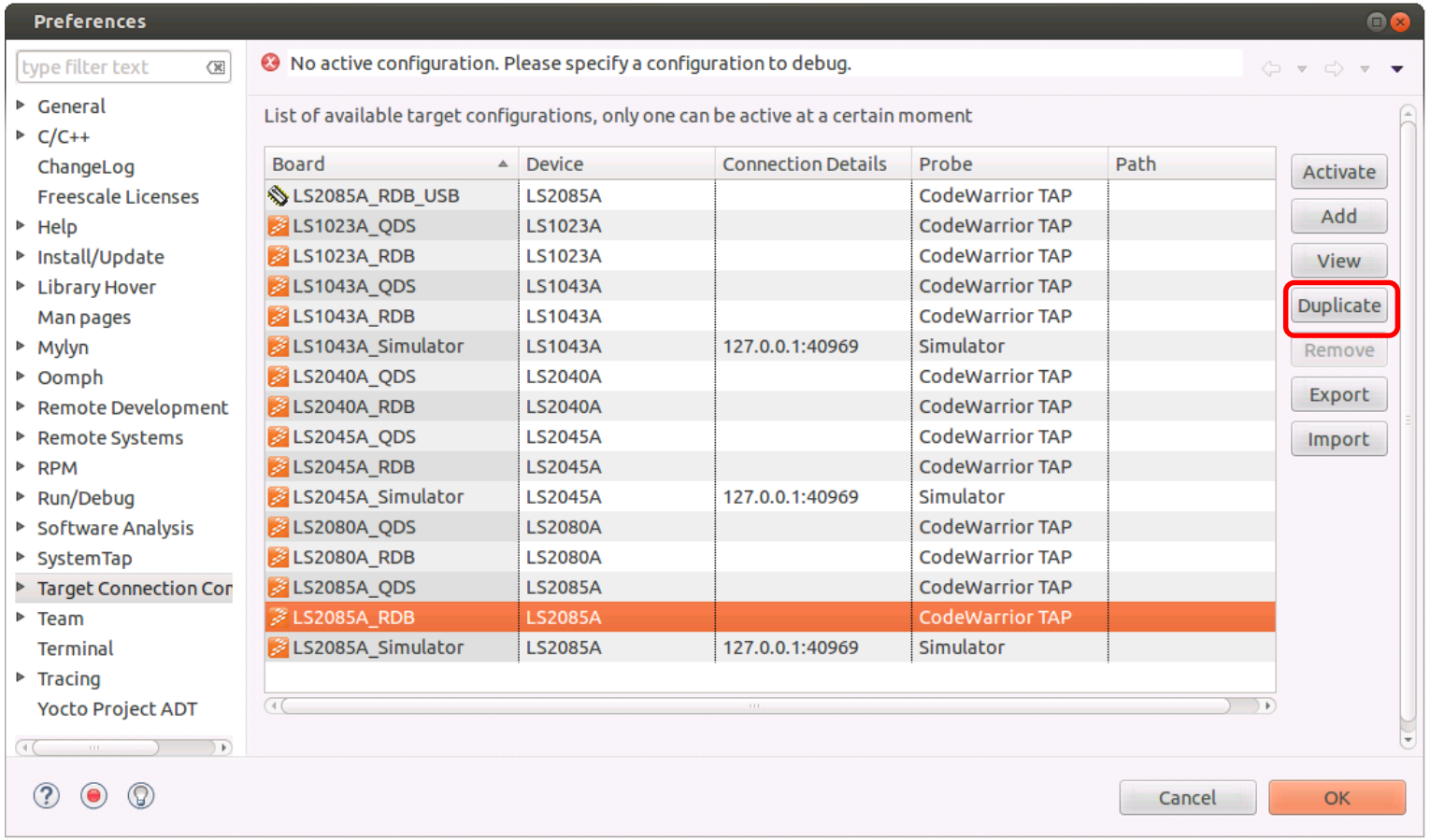
U-Boot Debug – Create Target Connection – Activity

If no target connection is set, configure one following the link “Configure target connection…”



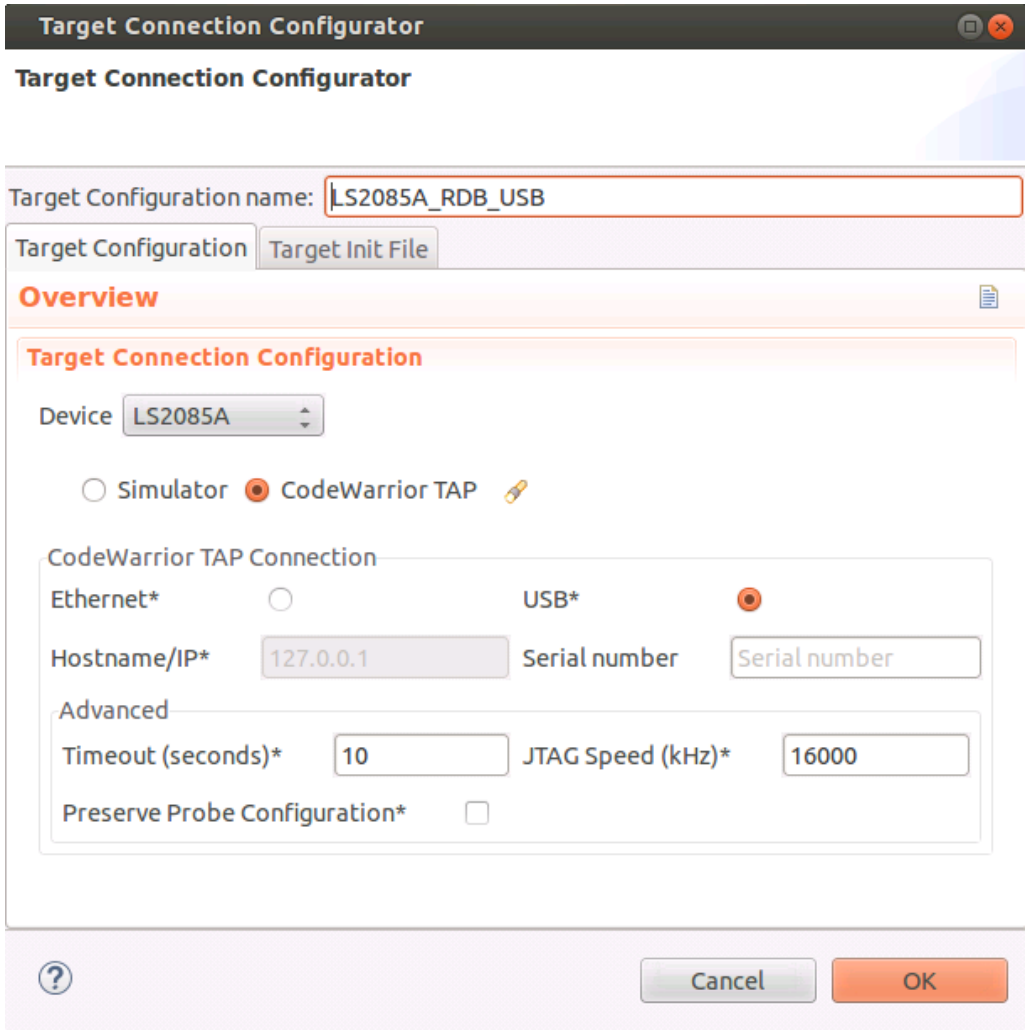
U-Boot Debug – Create Target Connection – Activity

- Select the desired target configuration template (e.g. LS2085A_RDB)
- Press “Duplicate” and set the new name
- Press Edit to change the template configuration



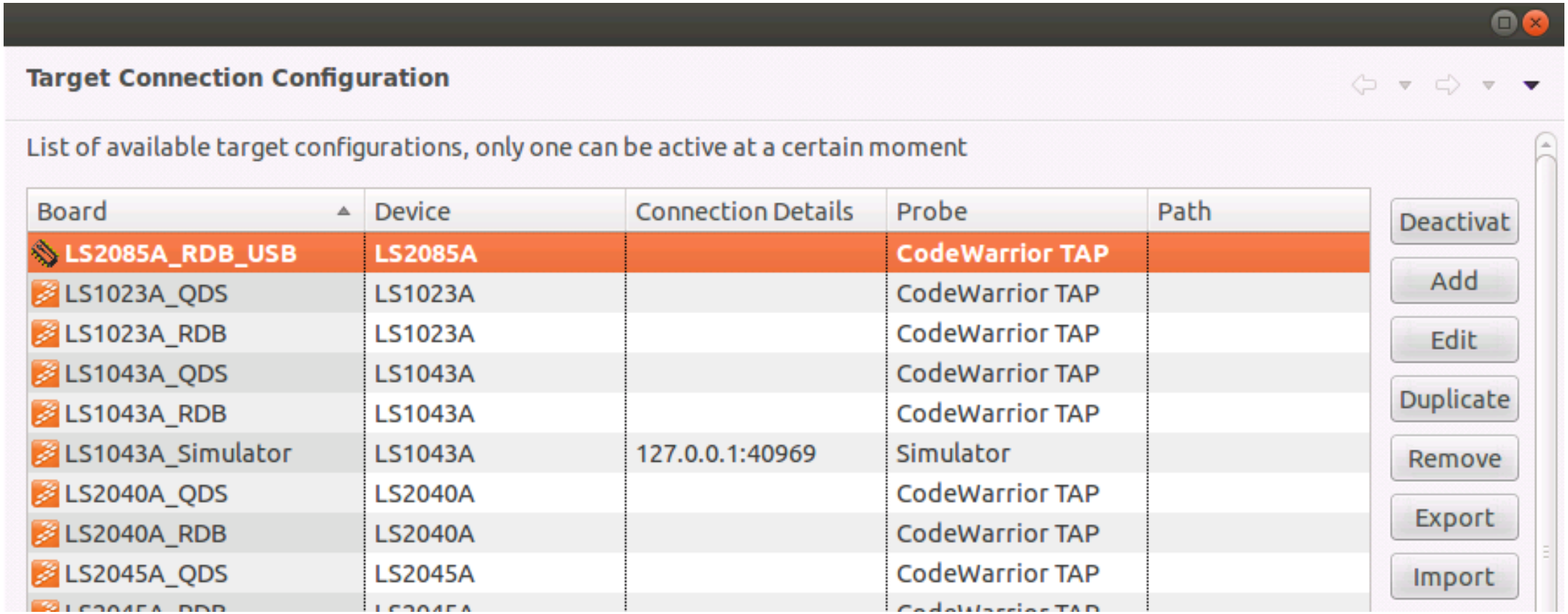
U-Boot Debug – Create Target Connection – Activity

- Edit the name (different than the template name)
- Select USB
- Press OK



U-Boot Debug – Create Target Connection – Activity

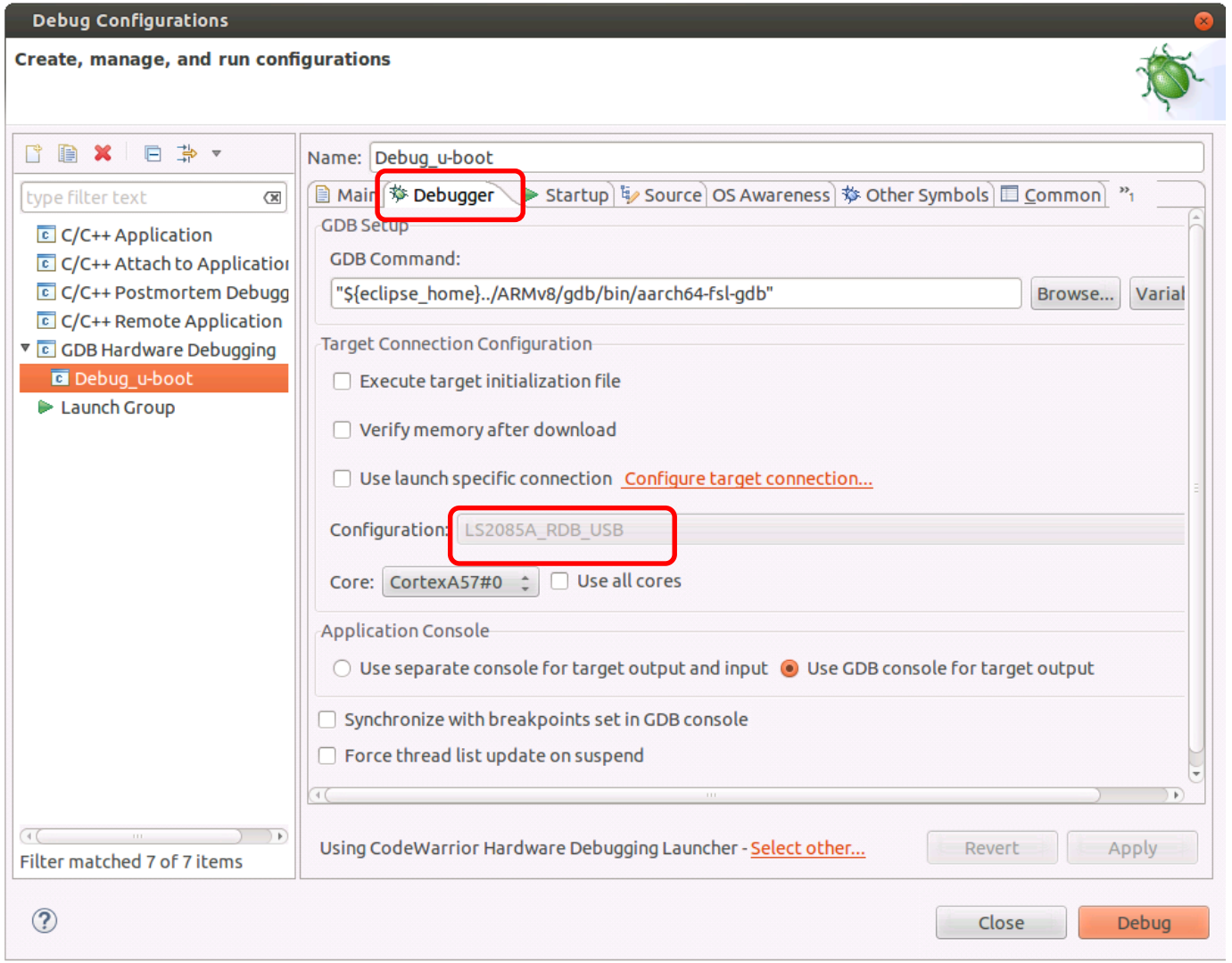
The newly created target configuration is set active



U-Boot Debug – Other Project Settings – Activity

Debugger Tab

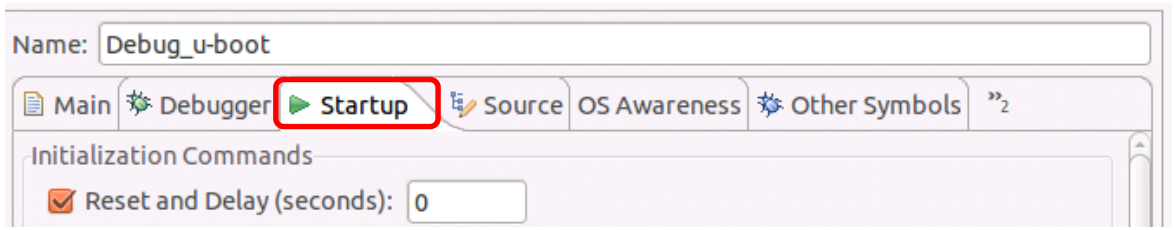
- Configuration automatically set to the default Target Connection



U-Boot Debug – Other Project Settings – Activity

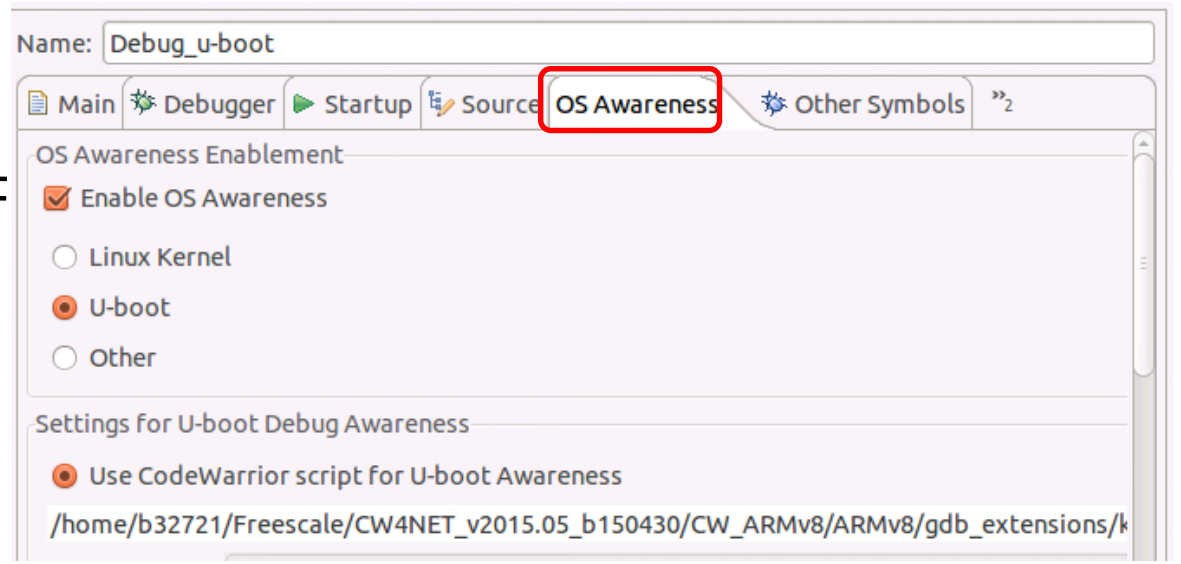
Startup Tab

- Uncheck Reset and Delay to attach to the current running U-Boot
- Check Reset and Delay = 0 to reset the target and debug U-Boot from the first instruction after reset



OS Awareness Tab

- U-Boot awareness settings have been automatically applied when CodeWarrior ELF Importer detects the U-Boot executable



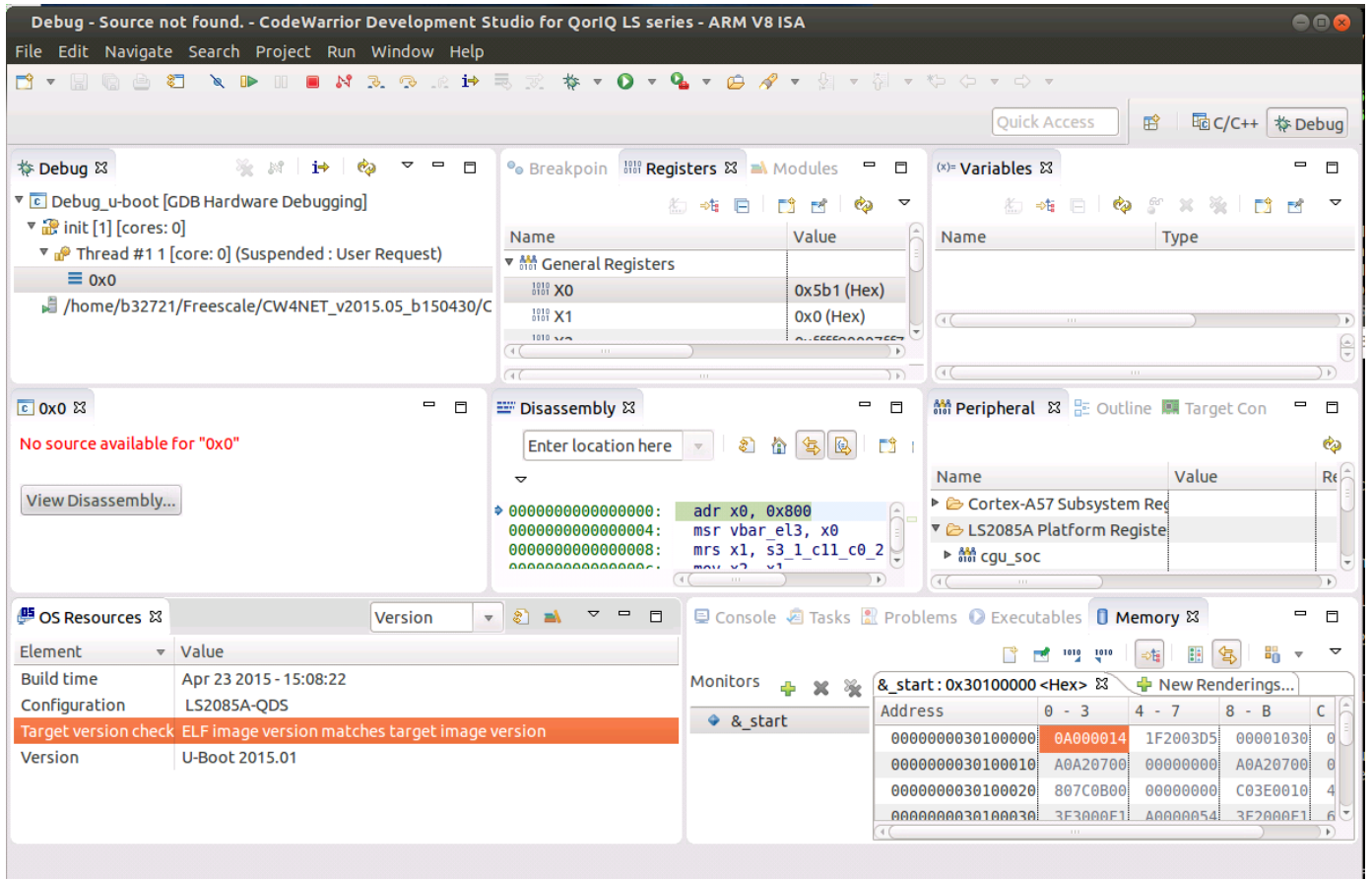
U-Boot Debug – Debug Overview – Activity

- **U-Boot Awareness**
  - A single U-Boot debug session while **the user is not aware about any stages or relocation offsets**.
  - The debugger automatically detects each U-Boot stage and performs the corresponding action.
  - The user can visualize meaningful U-Boot information about: U-Boot version, build time or memory information
  - Target image vs. ELF image version check

- Demonstrate a full U-Boot debug session
  - Debug from the first instruction after reset, to U-Boot entry point, running from Flash, after relocation to DDRAM and until U-Boot prompt is available
  - All is done in a single debug session with no other changes

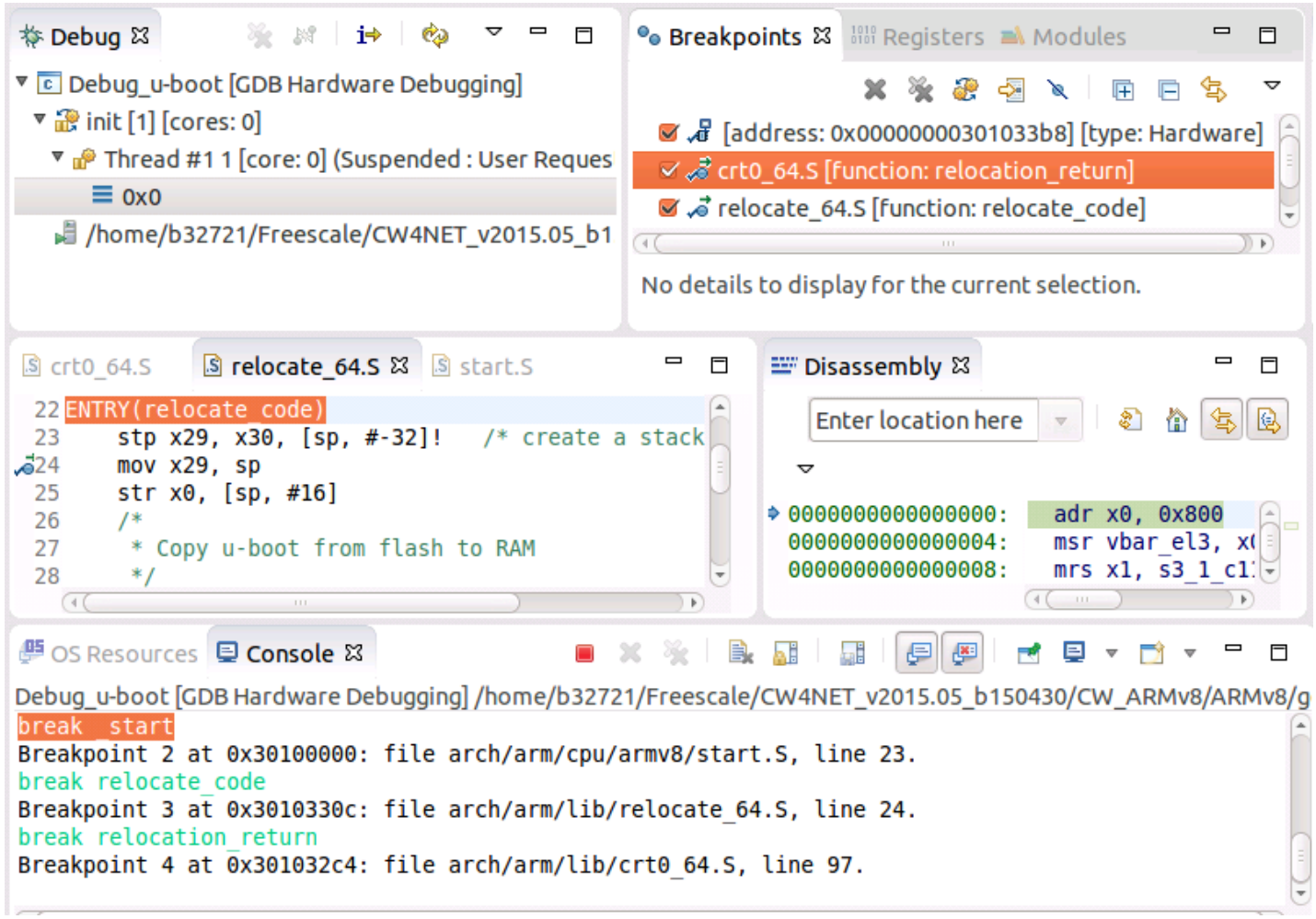
U-Boot Debug – Debug – Activity

- Click Launch Configuration > “Debug” button to start the debug session
- The target is reset and the debugger stops at the first instruction after reset
  - PC is 0x0 – BootRom
- No symbols or source file view available in U-Boot ELF – the U-Boot code has not been started yet



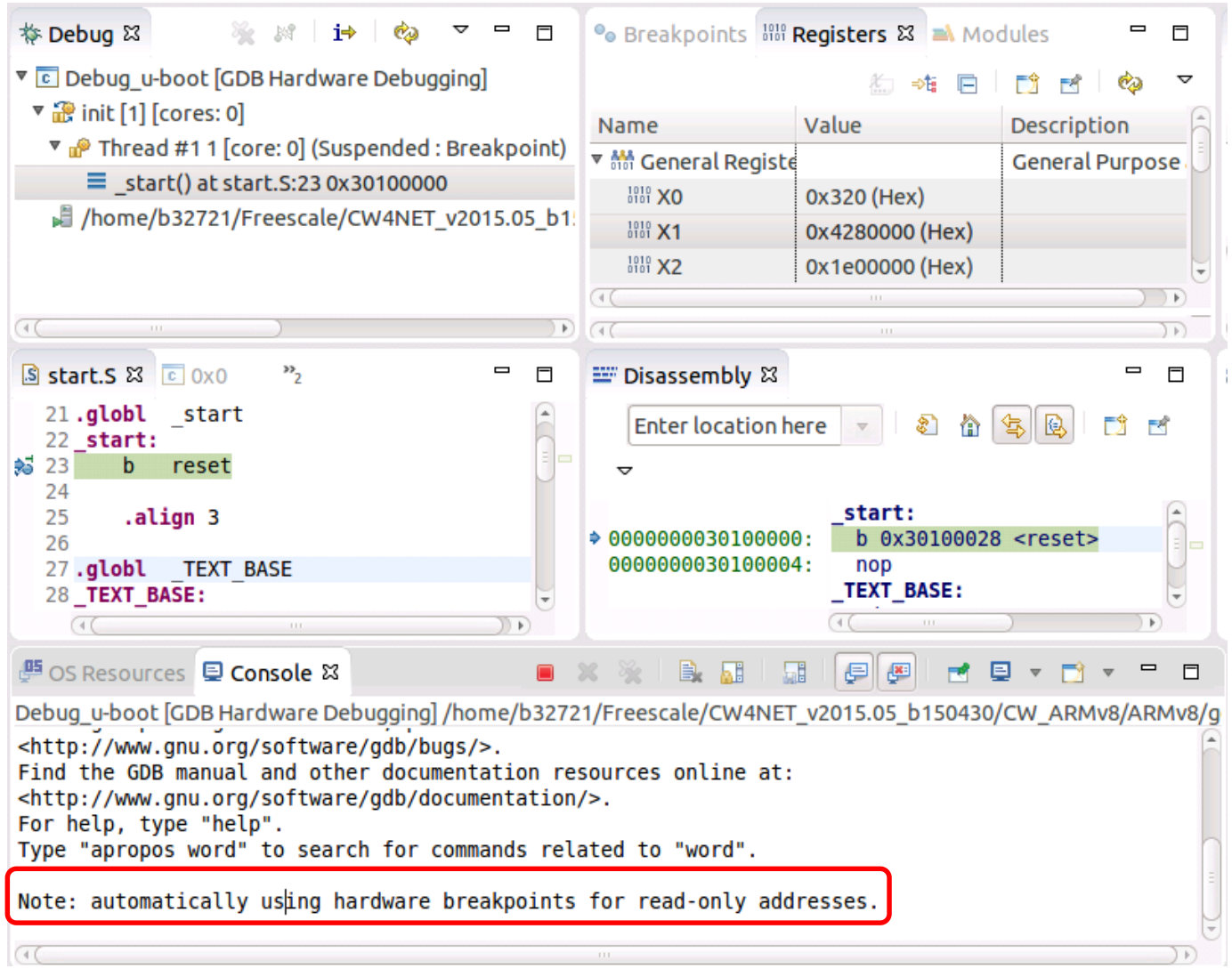
U-Boot Debug – Debug – Activity

- To demonstrate full U-Boot debug in a single session, set several breakpoints in some key points of booting process
- Set breakpoints from console or from file:
  - _start: U-Boot entry point
  - relocate_code: when running from Flash and relocation to DDRAM is started
  - relocation_return: first function executed from DDRAM



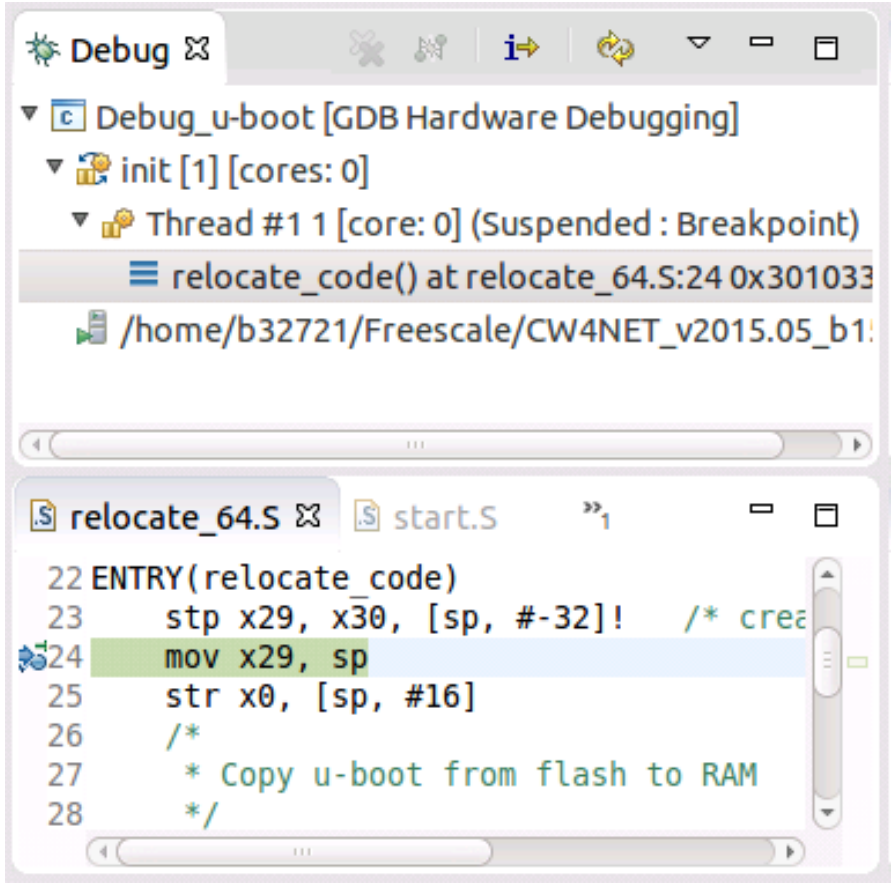
U-Boot Debug – Debug – Activity

- Entry point breakpoints hit
- Note: breakpoint set as SW breakpoint (default) to a Flash (read-only) address. U-Boot Awareness automatically changes the breakpoint type to HW for Flash address.



U-Boot Debug – Debug – Activity

- Start relocation breakpoint hit
- U-Boot running from Flash (address 0x3010xxxx) prepare for running from DDRAM
- Booting messages on U-Boot console

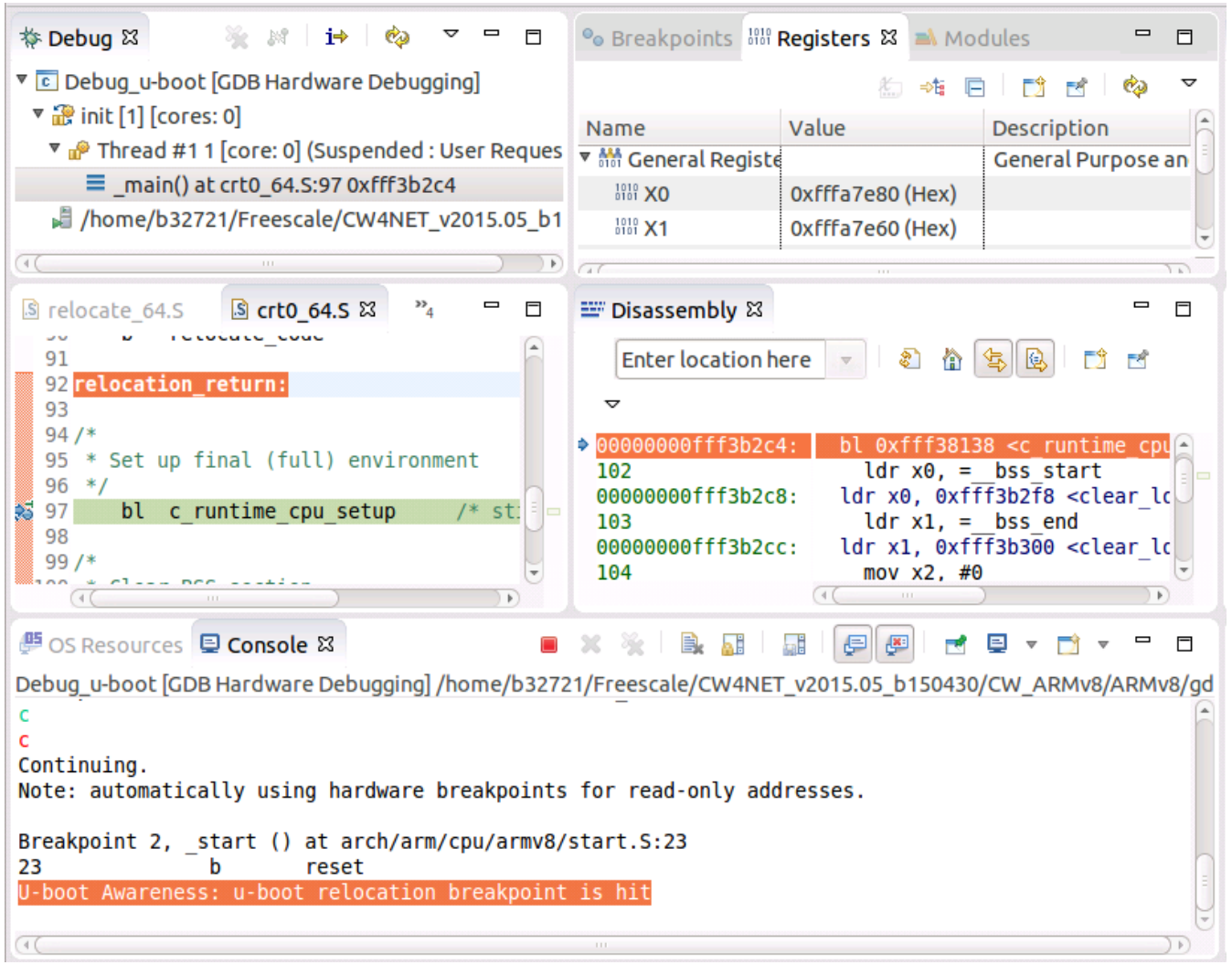



```
U-Boot 2015.01 (Apr 23 2015 - 15:08:22) LS2085A-QDS

Clock Configuration:
       CPU0(A57):1600 MHz  CPU1(A57):1600 MHz  CPU2(A57):1600 MHz
       CPU3(A57):1600 MHz  CPU4(A57):1600 MHz  CPU5(A57):1600 MHz
       CPU6(A57):1600 MHz  CPU7(A57):1600 MHz
       Bus:      600  MHz  DDR:      1333.333 MHz     DP-DDR:   1333.333 MHz
Reset Configuration Word (RCW):
       00: 40282830 40400040 00000000 00000000
       10: 00000000 00000000 00200000 00000000
       20: 00c12980 00002580 00000000 00000000
       30: 00000003 00000000 00000000 00000000
       40: 00000000 00000000 00000000 00000000
       50: 00000000 00000000 00000000 00000000
       60: 00000000 00000000 00027000 00000000
       70: 3f030007 00050000 00000000 00000000
Board:  LS2085A-QDS, Board Arch: V1, Board version: B, boot from vBank: 0
FPGA: v5 (LS2085AQDS_2015_0218_1606), build 132 on Wed Feb 18 22:06:38 2015
SERDES1 Reference : Clock1 = 156.25MHz Clock2 = 100 separate SSCGMHz
SERDES2 Reference : Clock1 = 100 separate SSCGMHz Clock2 = 100 separate SSCGMHz
I2C:   ready
DRAM:  Initializing DDR....using SPD
Detected UDIMM 18ASF1G72AZ-2G1A1
Detected UDIMM 18ASF1G72AZ-2G1A1
DP-DDR:  Detected UDIMM 18ASF1G72AZ-2G1A1
19.5 GiB
DDR    15.5 GiB (DDR4, 64-bit, CL=9, ECC on)
       DDR Controller Interleaving Mode: 256B
       DDR Chip-Select Interleaving Mode: CS0+CS1
DP-DDR 4 GiB (DDR4, 32-bit, CL=9, ECC on)
       DDR Chip-Select Interleaving Mode: CS0+CS1
```

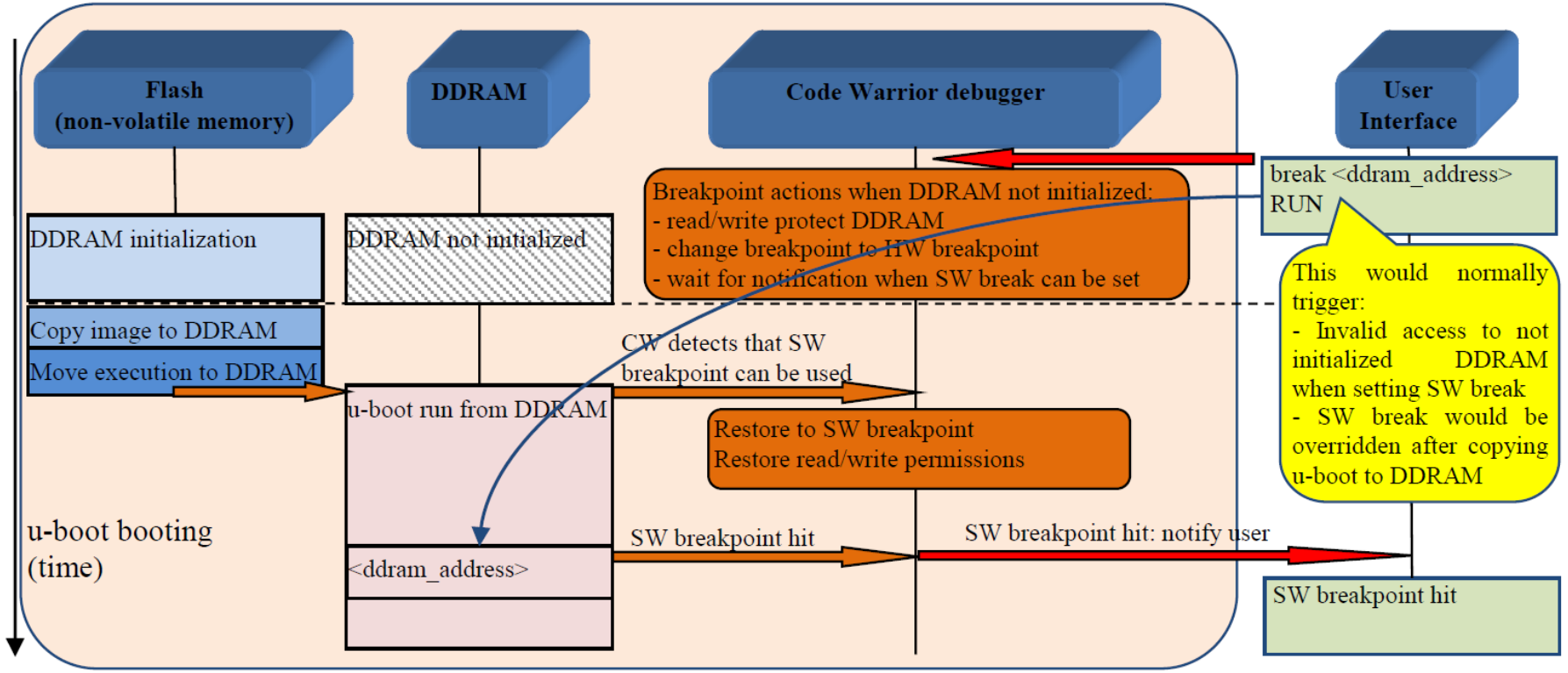
U-Boot Debug – Debug – Activity

- U-Boot running from DDRAM (relocation_return breakpoint hit)
- DDRAM Address 0xFFF3xxxx
- U-Boot Awareness breakpoint hit (see console) – debugger automatically handles the symbols relocation for DDRAM



U-Boot Debug – Debug – Activity

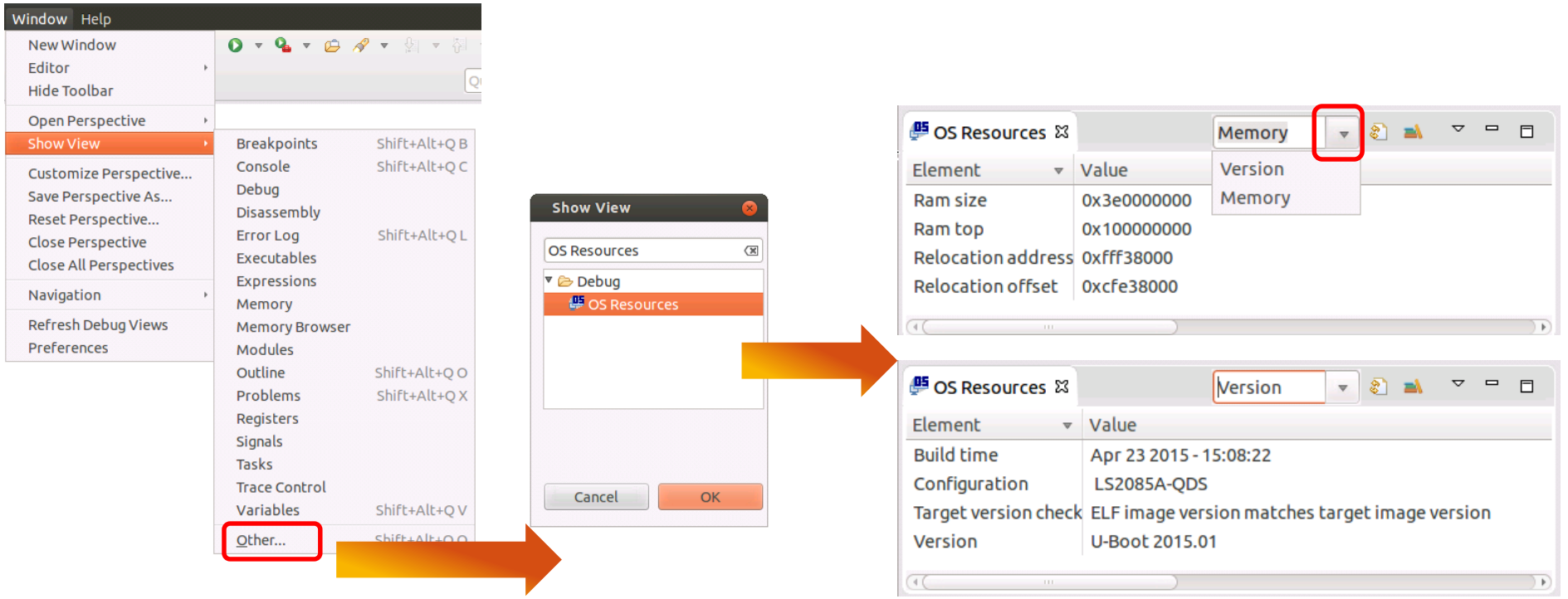
- U-Boot Awareness allows setting SW breakpoints to DDRAM before DDRAM initialization and before U-Boot relocation to DDRAM



U-Boot Debug – U-Boot information – Activity

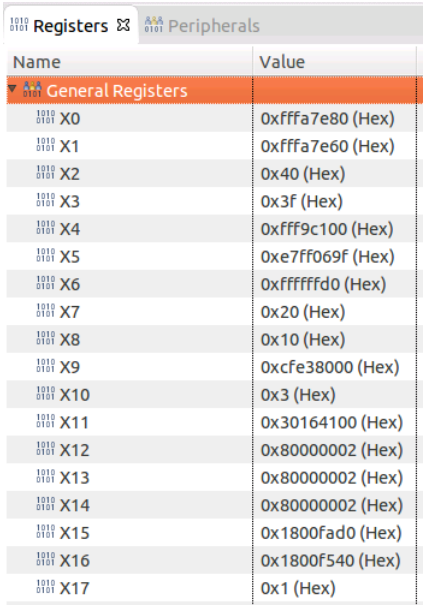
OS Resources: Windows > Show View > Other > Debugger > OS Resources

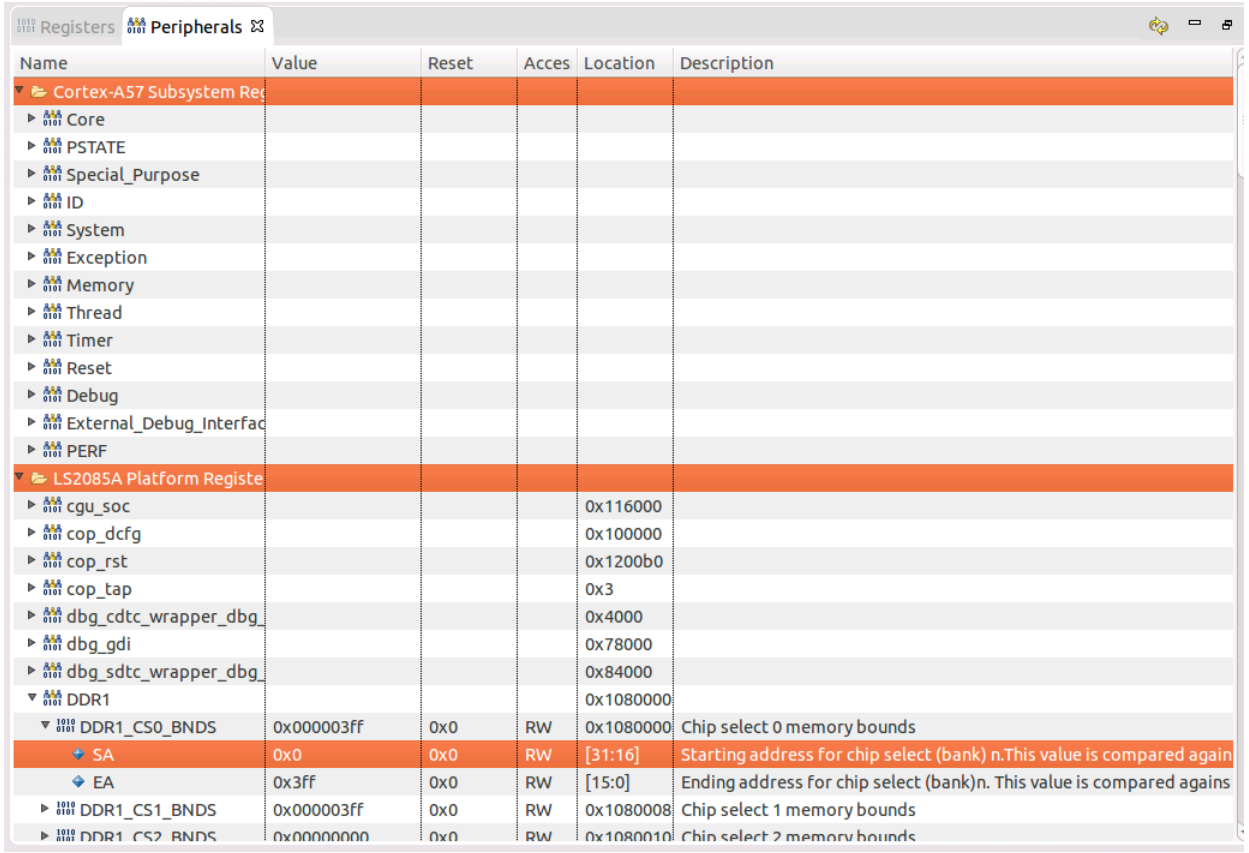
- Visualize meaningful U-Boot information about: U-Boot version, build time or memory information
- Target image vs. ELF image version check



U-Boot Debug – Register / Peripheral – Activity

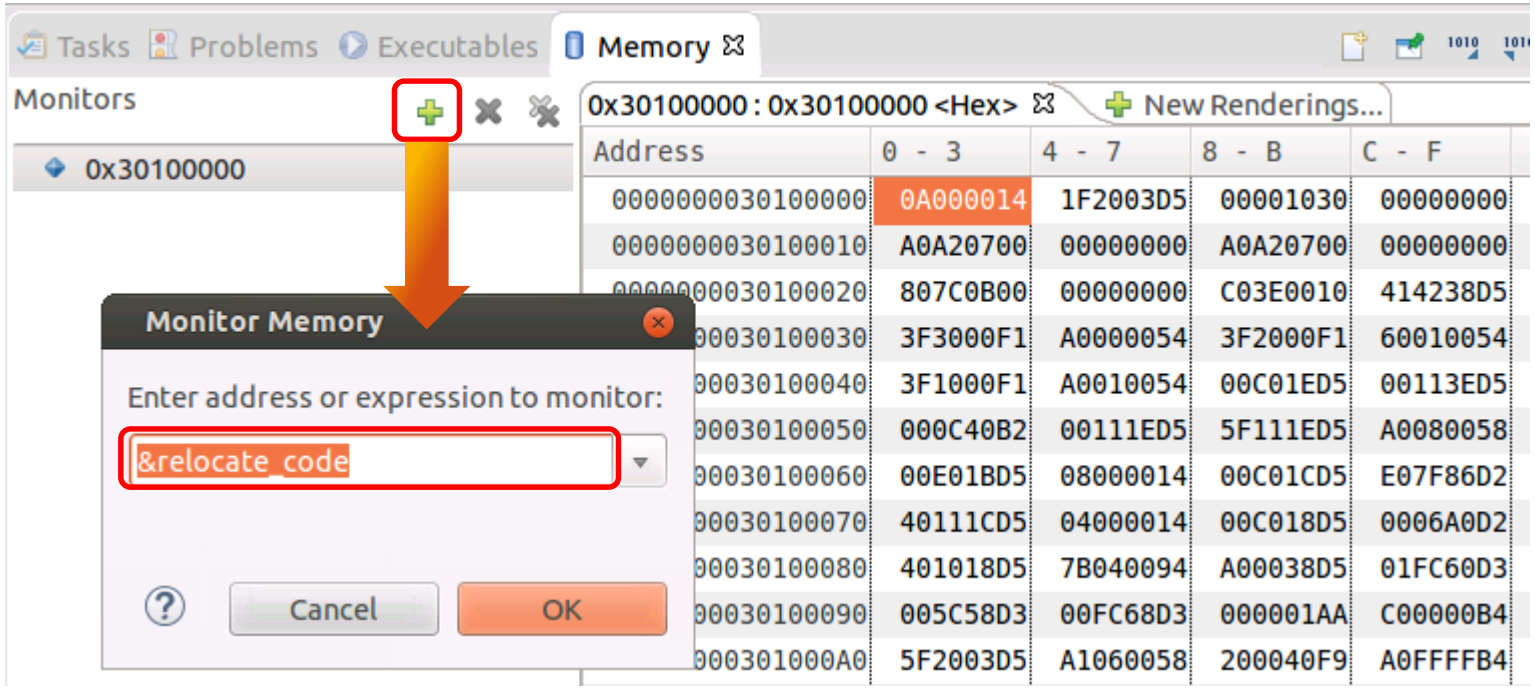
- Register view
- Peripheral: Windows > Show View > Other > Debugger > Peripherals





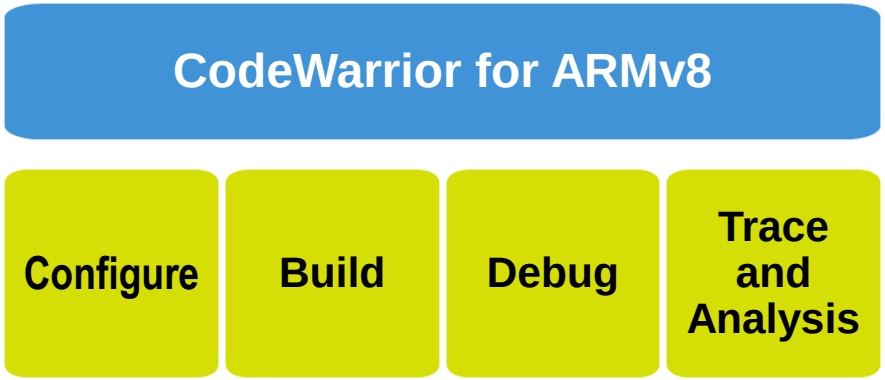
U-Boot Debug – Register / Peripheral – Activity

- Memory view: Window > Show View select “Memory”
- Add memory monitor
- Enter the address/expression



Summary

- This course has been a brief introduction into the LS2085 RDB board and the CodeWarrior tools available to debug the board
- Flash Programmer
- Connection Diagnostics
- U-Boot debug
- Digital Networking is introducing a new networking tools suite
  - CodeWarrior Development Studio for QorIQ LS Series – ARMv8 ISA
  - Tools covering Configuration, Build, Debug, and Analysis



http://www.nxp.com/codewarrior

Q & A

SECURE CONNECTIONS
FOR A SMARTER WORLD

ATTRIBUTION STATEMENT

PUBLIC USE

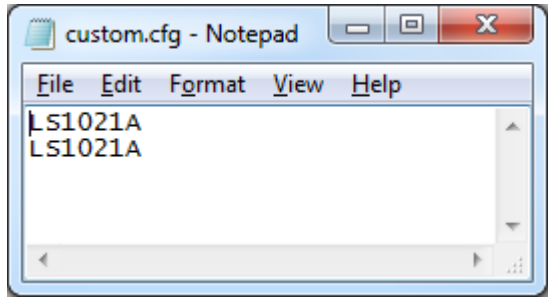
Company Public information is information that has been declared public knowledge by the owner and can freely be given to anyone inside or outside NXP without any possible damage to NXP.

BACKUP

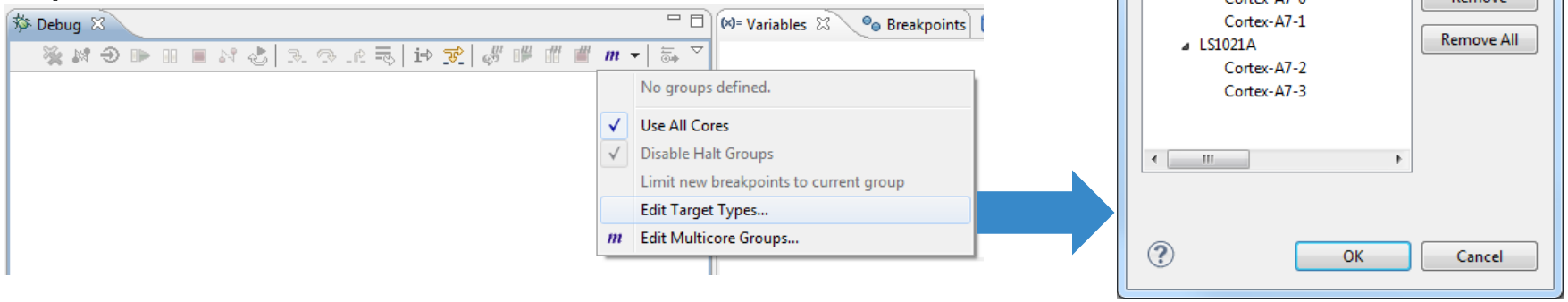
CUSTOM BOARD SETUP

Stock NXP Boards vs. Customer Designs

- For stock NXP boards, CW comes with all the necessary initialization / configuration files
- For custom designs, first, using a JTAG configuration file, define the list of devices on the JTAG chain.
  - As example, for a board with two QorIQ LS2085A processors, JTAG configuration file is:

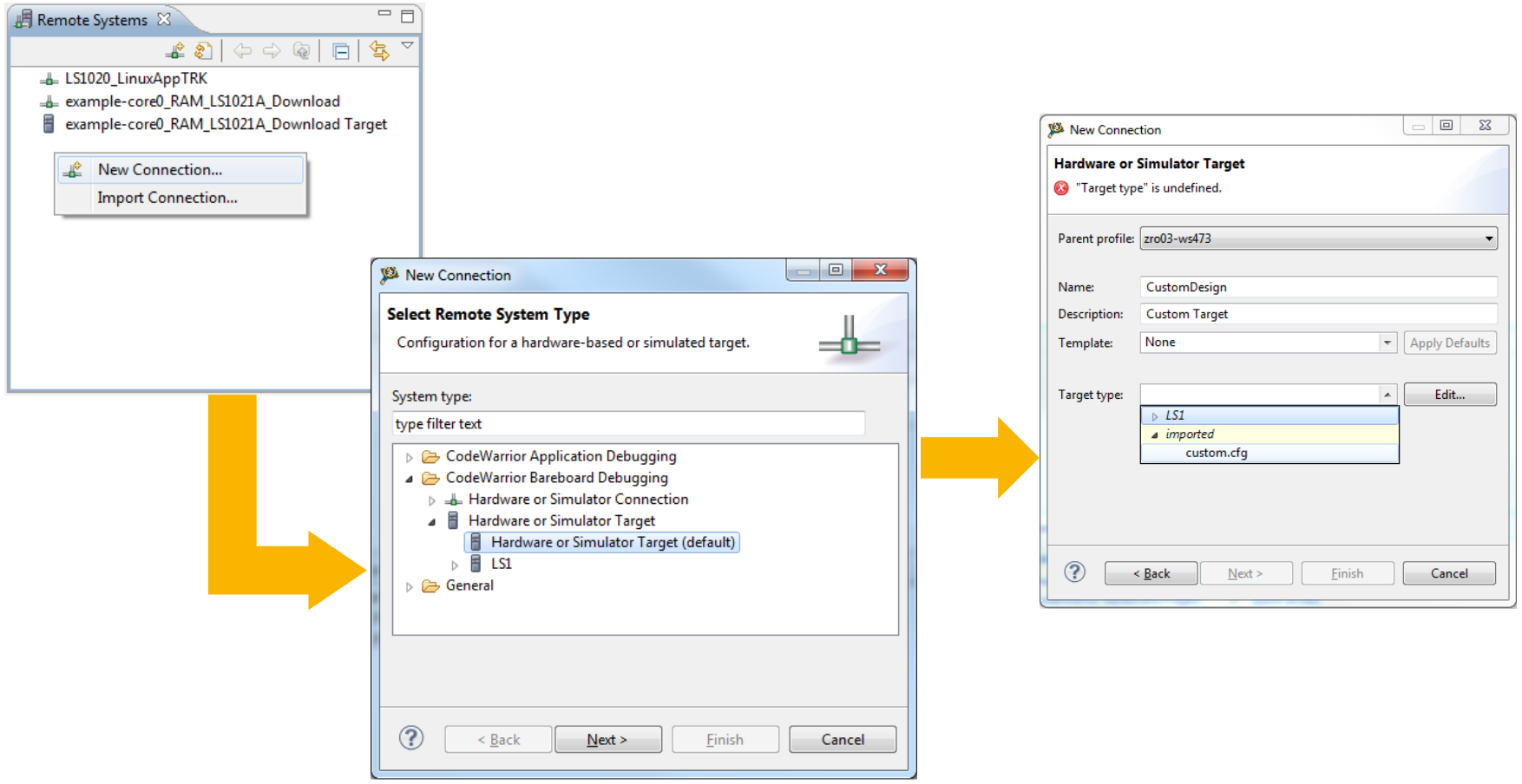


Import it.



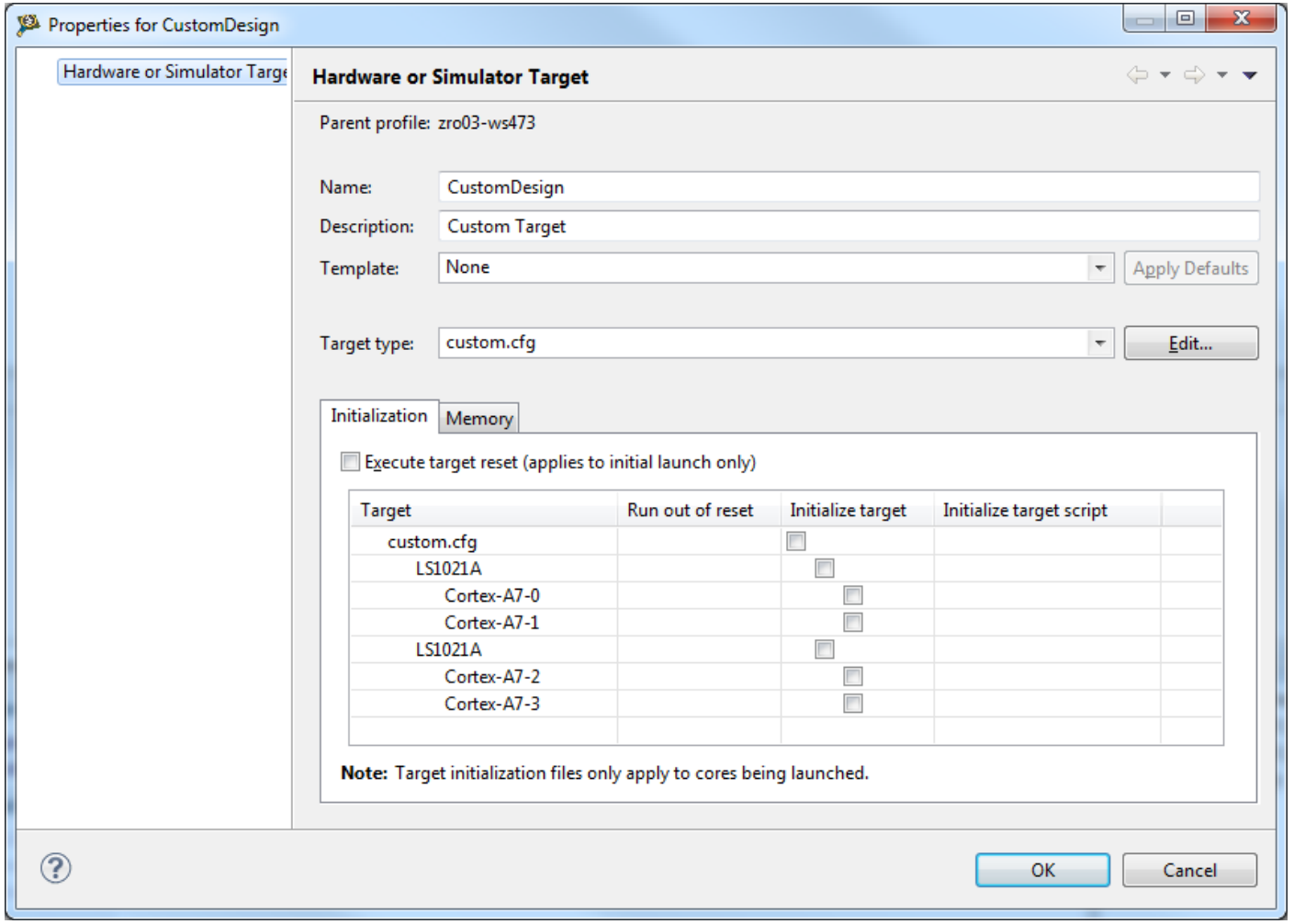
Creating a Custom Target System

- Create a new target system



Configure the Newly Created Target

- Check “Execute target reset” if you want to reset the SoC when first core connects.
- Select initialization files (for DDR, internal memories, flash controllers, etc.) and on what core(s) they should be executed.
- Select memory configuration files. Consider that the memory map may differ since, for example, some IP blocks may not be present in the new design.



New Connection and Debug

- Create a new connection and select for it the “Target” system that was just created.
- Apply the new connection to your project(s)