FTF 2016
TECHNOLOGY FORUM

DEBUG LINUX APPLICATIONS USING CODEWARRIOR FOR ARM®V8

ROBERT MCGOWAN | CHIEF ARCHITECT
CATALIN UDMA
FTF-DES-1836
MAY 18, 2016

PUBLIC USE

AGENDA

- Lecture: Introduction/Overview
- Lecture: CodeWarrior
- Lecture: Board/Device Overview
- Lecture: Linux Application Debug
- Activity: Linux Application Debug Download
- Activity: Linux Application Debug Attach
- Lecture: ODP Reflector Application
- Activity: Run ODP Reflector Application
- Activity: Debug ODP Reflector Application

INTRODUCTION/ OVERVIEW

Software and Tools Enablement for QorIQ LS-Series

CodeWarrior
Development Studio

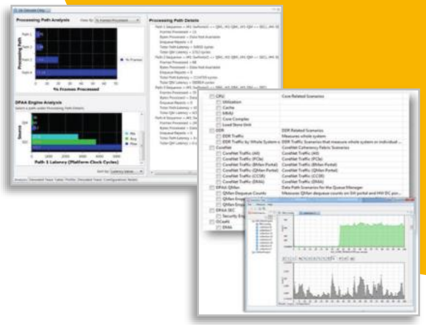
Software Development Tools

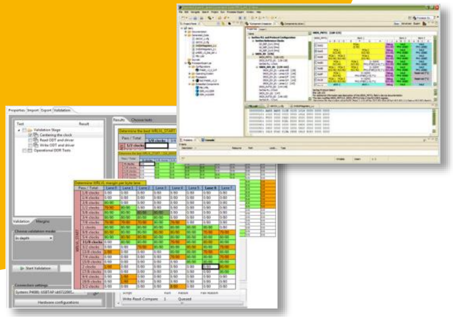
**QorIQ
Linux SDK**

***JTAG Run Control
and Trace Probes***

***Performance Analysis
and Trace Tools***

***QorIQ SoC Platform
Configuration Tools***





Software Products and Services

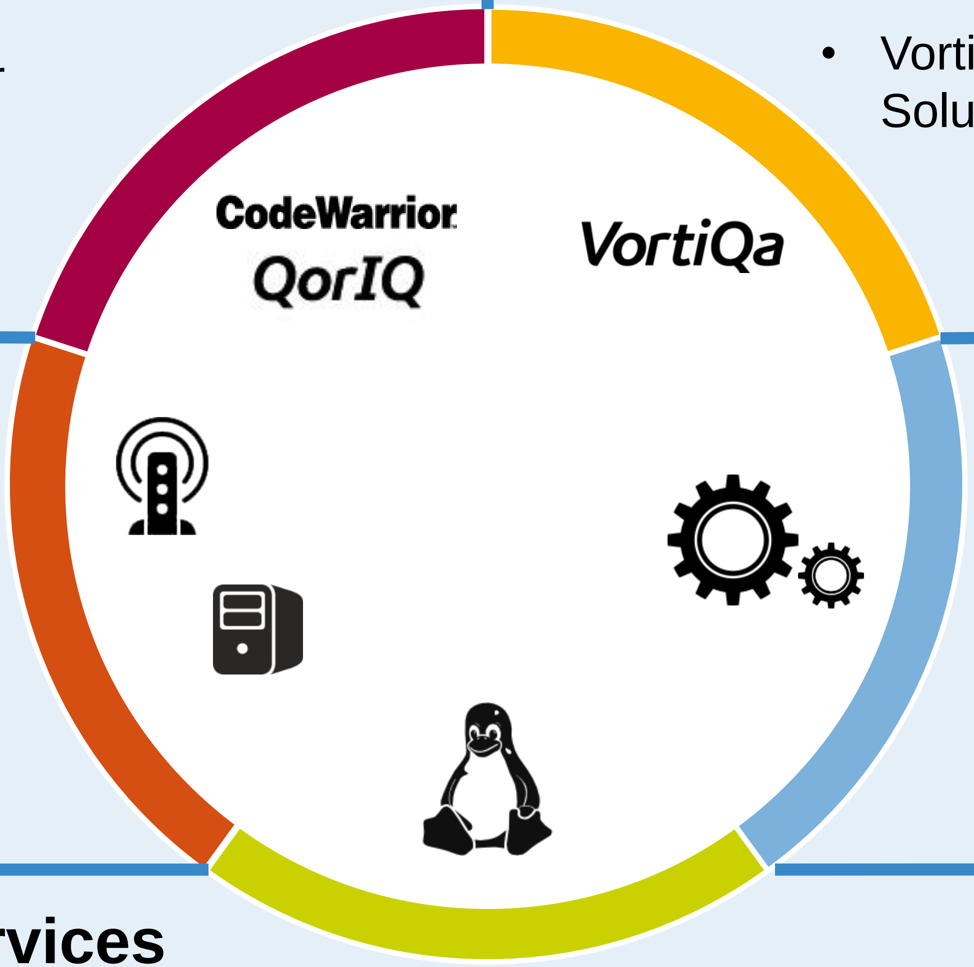
Visit us in the Tech Lab – #247

Development Tools

- CodeWarrior

Runtime Products

- VortiQa Software Solutions



Solutions Reference

- IOT Gateway
- OpenWRT+

Integration Services

- Security Consulting
- Hardened Linux

Linux® Services

- Commercial Support
- Performance Tuning

Accelerate **Customer Time-to-Market**

Deliver **Commercial Software, Support, Services and Solutions**

Simplify **Software Engagement with NXP**

Create **Success!**

Find us online at www.nxp.com/networking-services

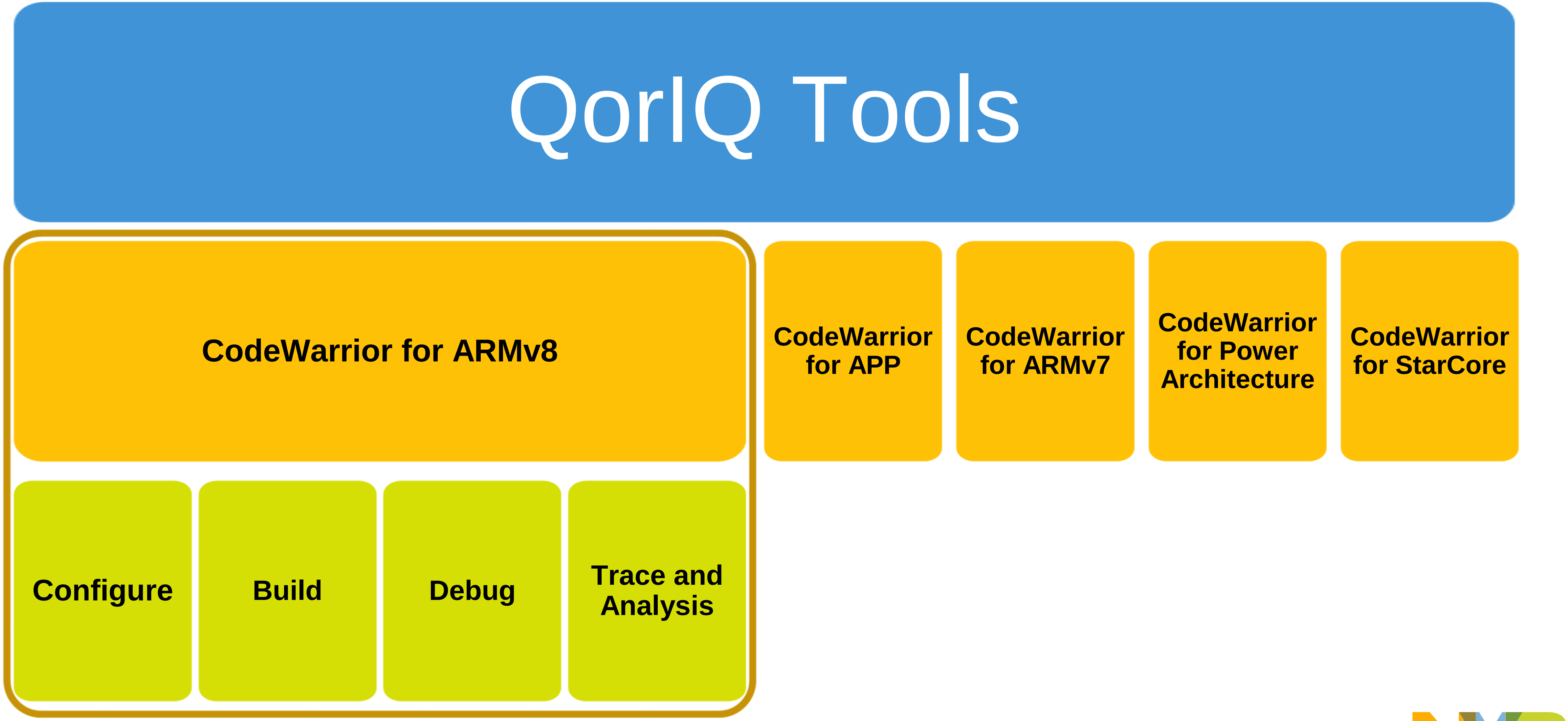
Training and Hands-on Goals

- The following material has been developed to help you…
  - ... Become familiar with debugging Linux applications using CodeWarrior
  - ... Learn about the networking capabilities of the QorIQ LS2085A platform
  - ... Configure and use Linux QorIQ networking resources with demo application
  - ... Debug a demo Linux networking application

QorIQ TOOLS

CodeWarrior for ARMv8 ISA
CW-TAP

CodeWarrior Family



QorIQ Tools

CodeWarrior for ARMv8

Configure

Build

Debug

Trace and Analysis

CodeWarrior for APP

CodeWarrior for ARMv7

CodeWarrior for Power Architecture

CodeWarrior for StarCore

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 & Linaro GNU toolchain
  - Target debug agent

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

CodeWarrior Aids Debug Through Multiple Phases (3)

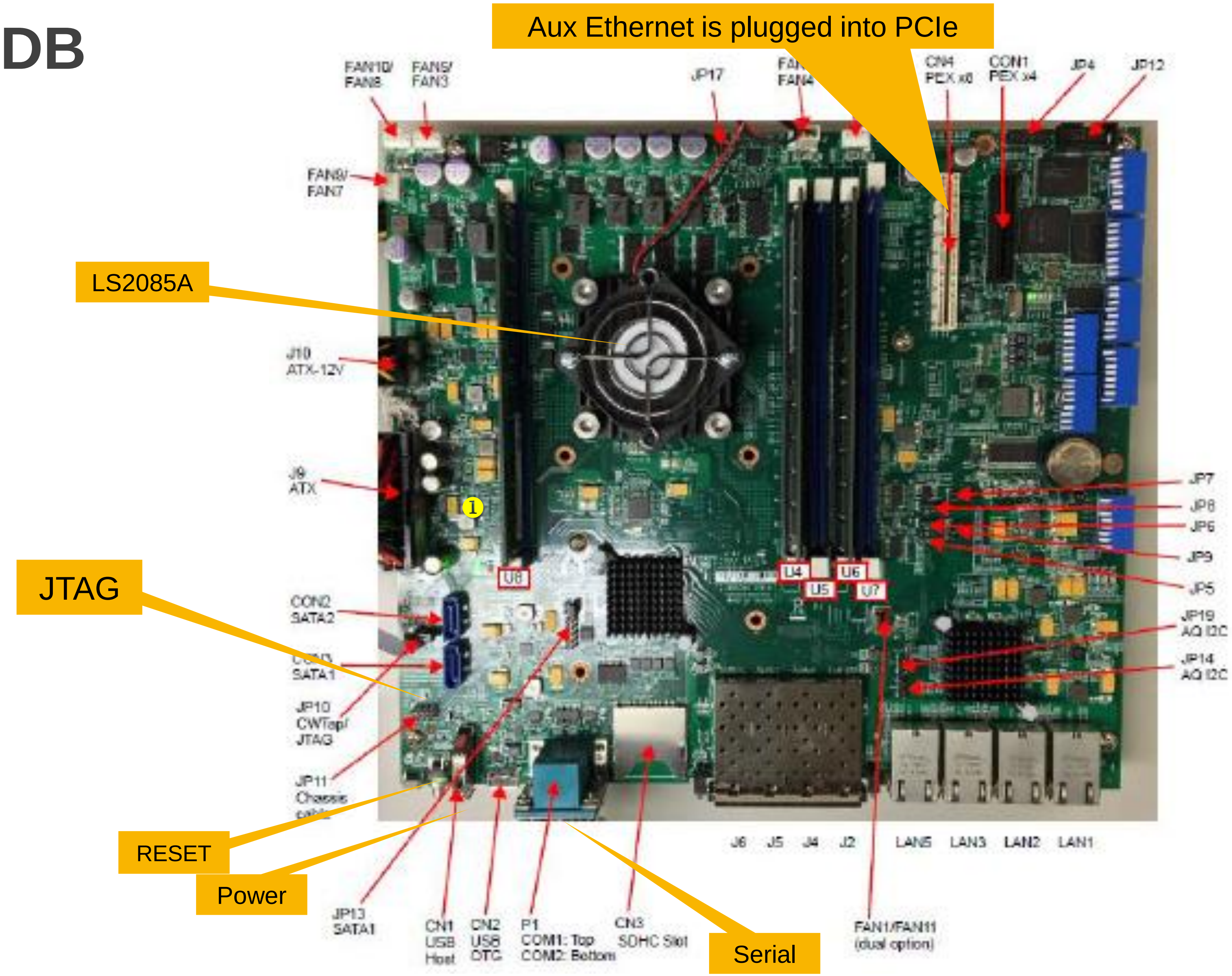
- Non-intrusive debug through trace
  - Core and SoC trace sources: configuration, extraction, visibility
  - Post-mortem debugging: offline trace
  - Debug-print
  - Linux aware trace
  - Linux kernel trace
  - Code Coverage

BOARD/DEVICE OVERVIEW

INTRODUCING THE LS2085A RDB

LS2085A RDB

Top View



Aux Ethernet is plugged into PCIe

LS2085A

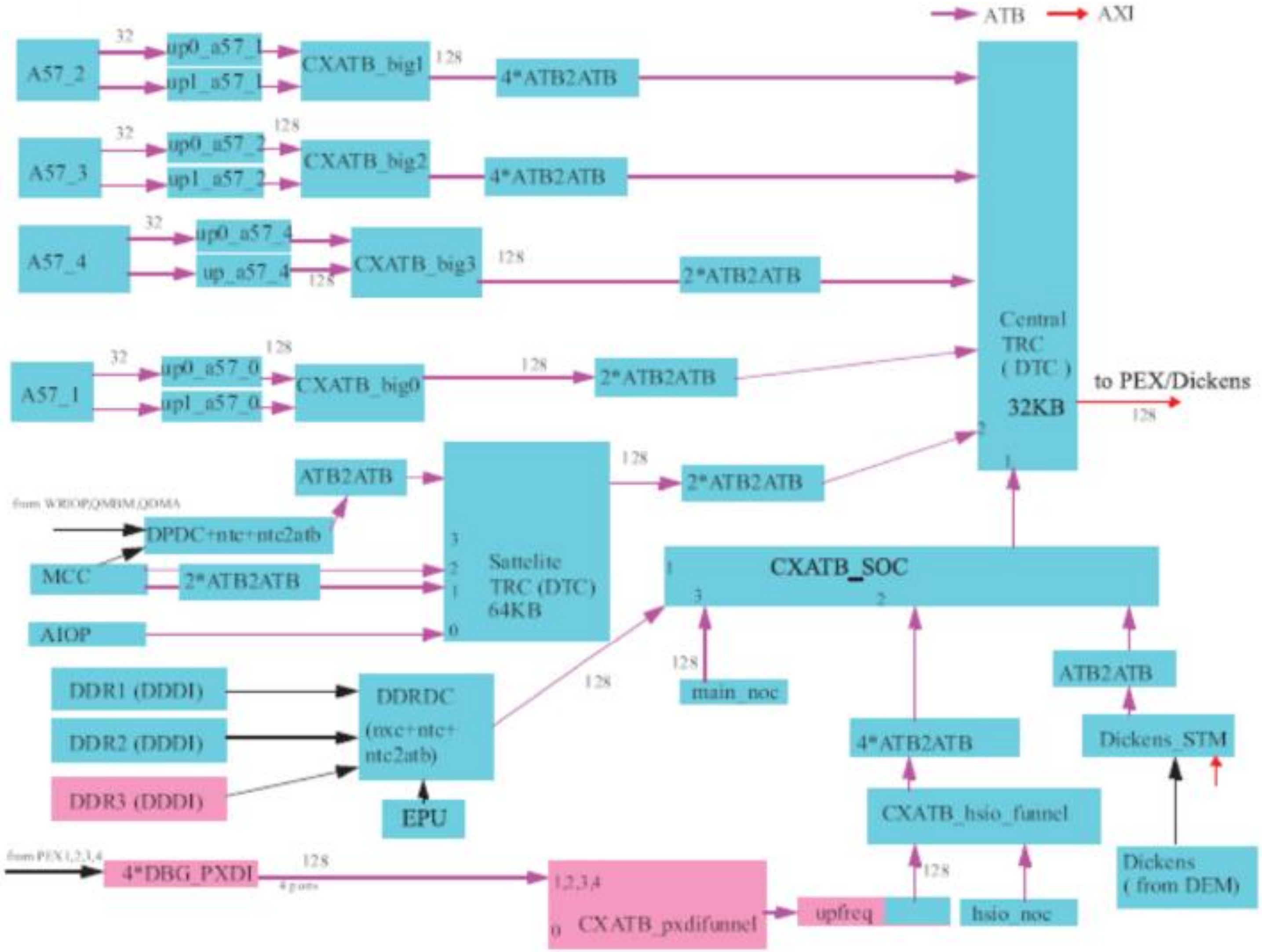
JTAG

RESET

Power

Serial

QorIQ LS2085A Debug Block Diagram



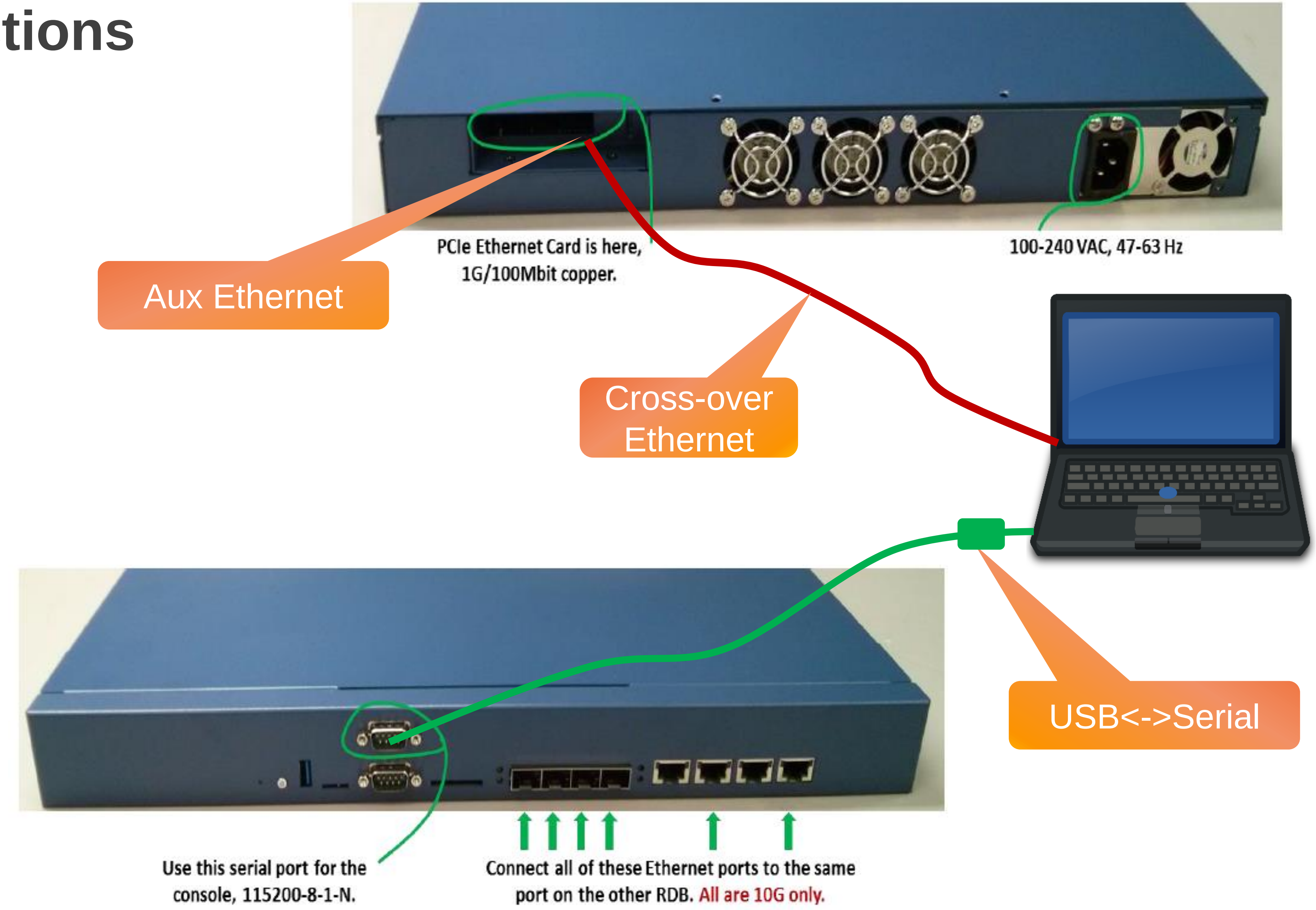
Debug Features

- 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



PCIe Ethernet Card is here, 1G/100Mbit copper.

100-240 VAC, 47-63 Hz

Aux Ethernet

Cross-over Ethernet

USB<->Serial

Use this serial port for the console, 115200-8-1-N.

Connect all of these Ethernet ports to the same port on the other RDB. All are 10G only.

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 ARMv8 ISA
- Linux SDK for QorIQ 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
```

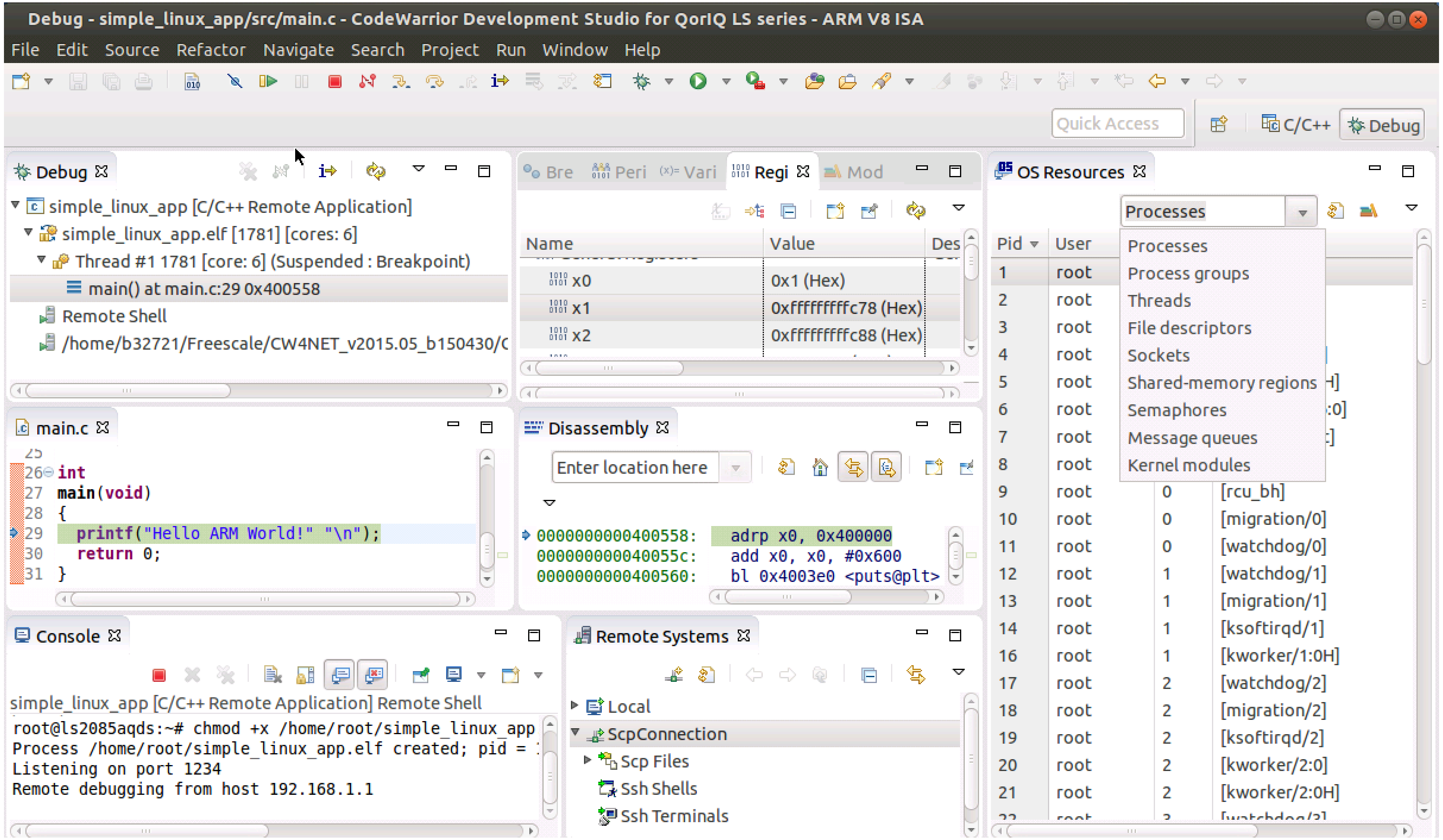
Linux

- 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:~#
```

LINUX APPLICATION DEBUG

CodeWarrior – Debugging ARM Target



- simple_linux_app [C/C++ Remote Application]
  - simple_linux_app.elf [1781] [cores: 6]
    - Thread #1 1781 [core: 6] (Suspended : Breakpoint)
      - main() at main.c:29 0x400558
  - Remote Shell
  - /home/b32721/Freescale/CW4NET_v2015.05_b150430/C

Name	Value	Des
x0	0x1 (Hex)	
x1	0xfffffffffc78 (Hex)	
x2	0xfffffffffc88 (Hex)	

```
25
26 int
27 main(void)
28 {
29   printf("Hello ARM World!" "\n");
30   return 0;
31 }
```

```
0000000000400558:   adrp x0, 0x400000
000000000040055c:   add x0, x0, #0x600
0000000000400560:   bl 0x4003e0 <puts@plt>
```

simple_linux_app [C/C++ Remote Application] Remote Shell

```
root@ls2085aqds:~# chmod +x /home/root/simple_linux_app
Process /home/root/simple_linux_app.elf created; pid = 1
Listening on port 1234
Remote debugging from host 192.168.1.1
```

- Local
- ScpConnection
  - Scp Files
  - Ssh Shells
  - Ssh Terminals

Pid	User		
1	root		
2	root		
3	root		
4	root		
5	root		H]
6	root		:0]
7	root		]
8	root		
9	root	0	[rcu_bh]
10	root	0	[migration/0]
11	root	0	[watchdog/0]
12	root	1	[watchdog/1]
13	root	1	[migration/1]
14	root	1	[ksoftirqd/1]
16	root	1	[kworker/1:0H]
17	root	2	[watchdog/2]
18	root	2	[migration/2]
19	root	2	[ksoftirqd/2]
20	root	2	[kworker/2:0]
21	root	2	[kworker/2:0H]

Two Ways to Run GDB

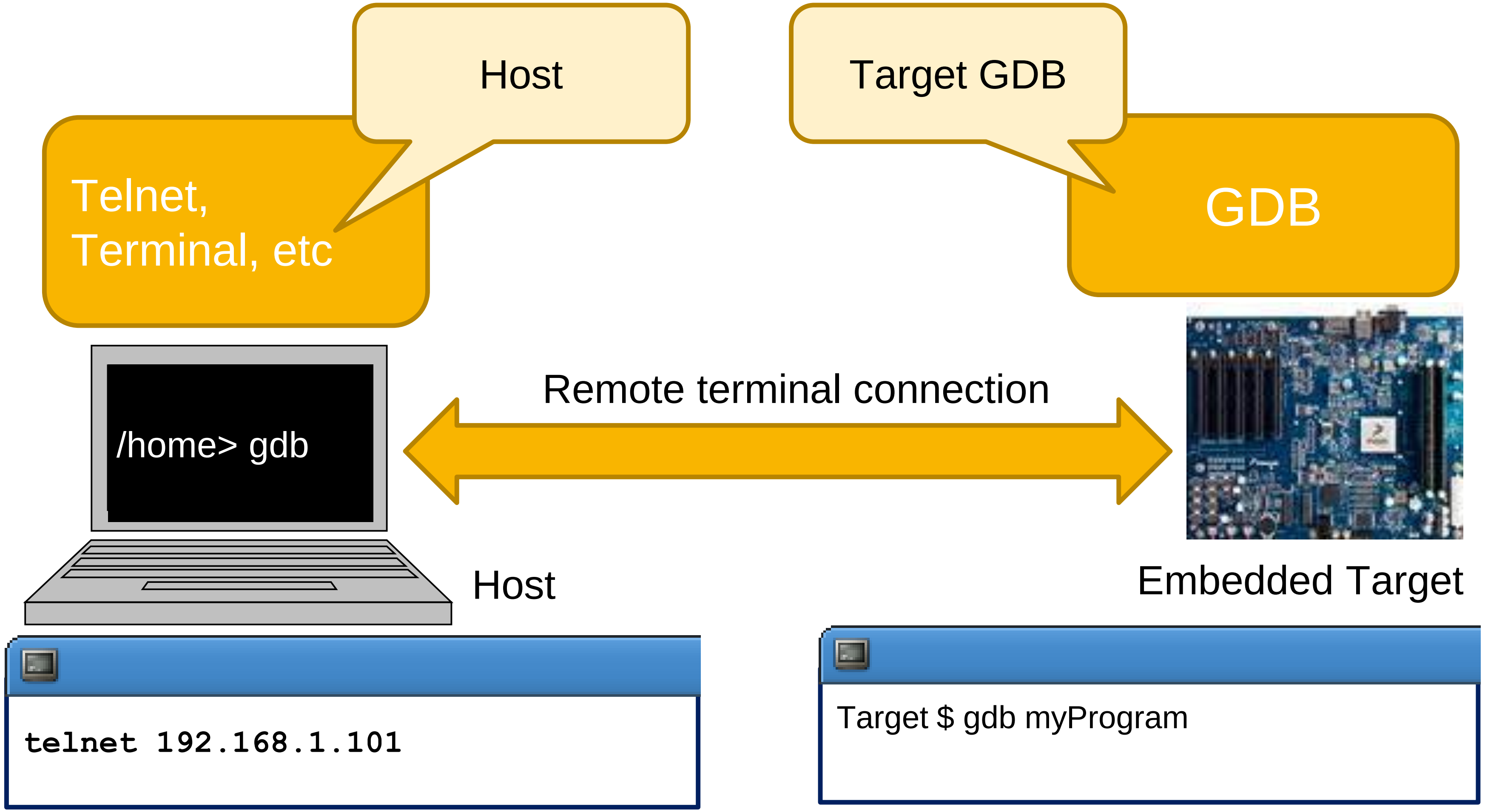
Native (host)

- GDB runs on the target (DUT)
  - E.g. Target OS: Linux
- Debugs an application running on the same system
- Interface with the target system using other applications
  - telnet into the target system to run GDB from the Linux command prompt

Target (self-hosted)

- GDB runs on the development host
  - Host OS and Target OS are not necessarily the same
- Remotely debugs an application running on the target
  - Socket connection or UART connection over the OS's drivers and interface carries GDB commands and responses
  - Host GDB communicates with target GDB server

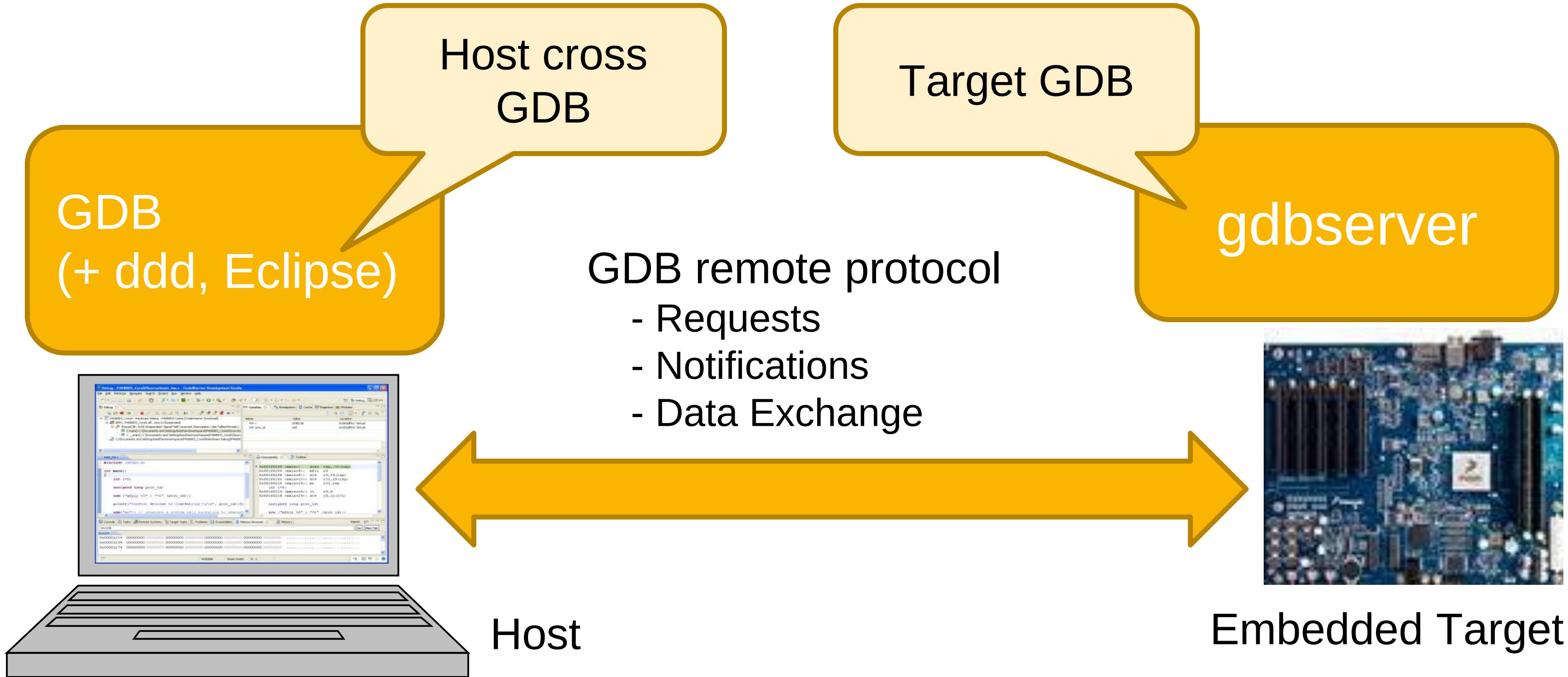
GDB Self-Hosted Target Debugging ARM Target



```
telnet 192.168.1.101
```

```
Target $ gdb myProgram
```

GDB Host Remote Debugging ARM Target



Host cross GDB

Target GDB

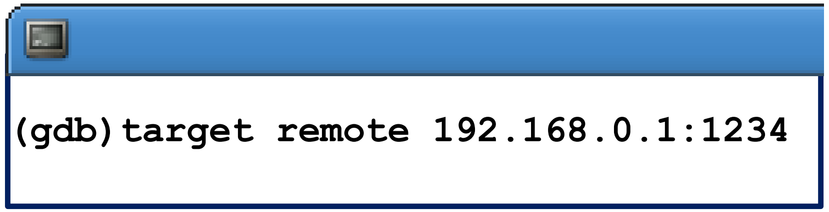
GDB
(+ ddd, Eclipse)

gdbserver

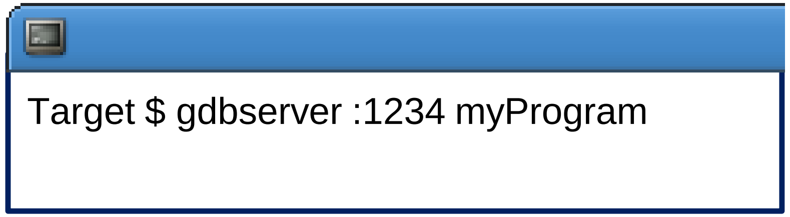
GDB remote protocol
- Requests
- Notifications
- Data Exchange

Host

Embedded Target



```
(gdb)target remote 192.168.0.1:1234
```



```
Target $ gdbserver :1234 myProgram
```

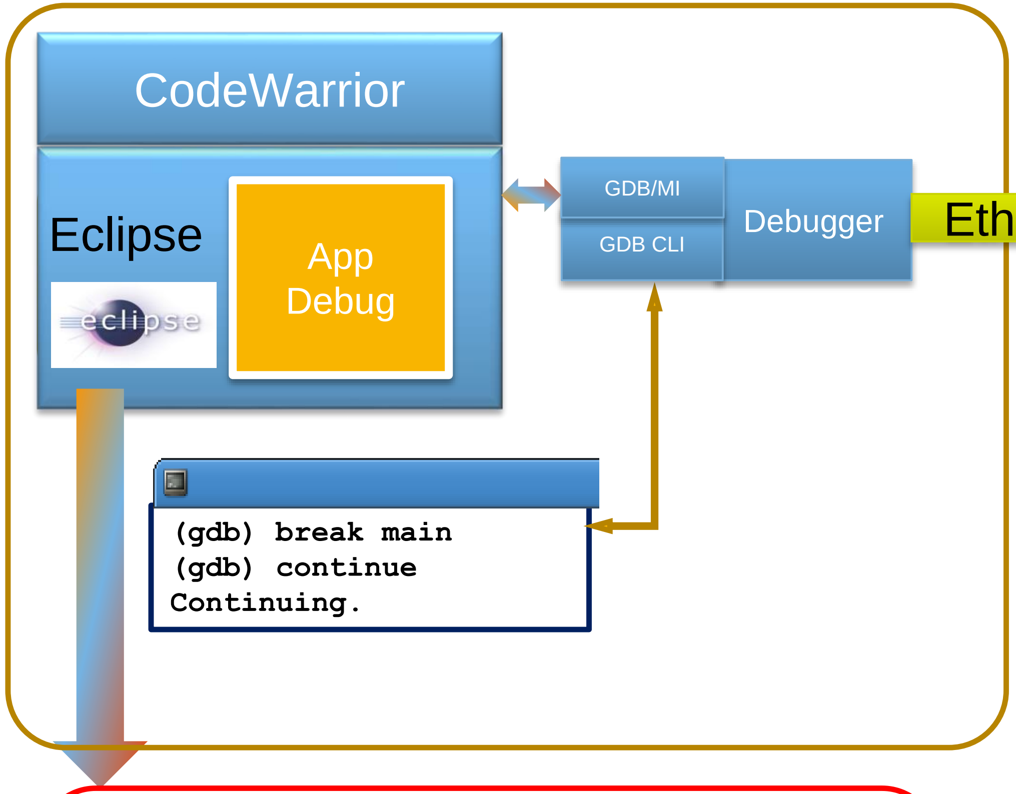
Linux Application Debug – Capabilities

- **gdbserver** Debug agent
  - User-space application
  - Uses **ptrace**
- Debug **scenarios** supported
  - **Download**, start & debug application from main
  - **Attach** to a running process
- Features
  - Read/write memory, registers, variables
  - **Threads creation/death detection**
  - Shared libraries awareness
  - Configurable signal policies
  - I/O redirection
- **OS Resources**
- CodeWarrior – GDB server interaction
  - Ethernet connection
  - Serial connection

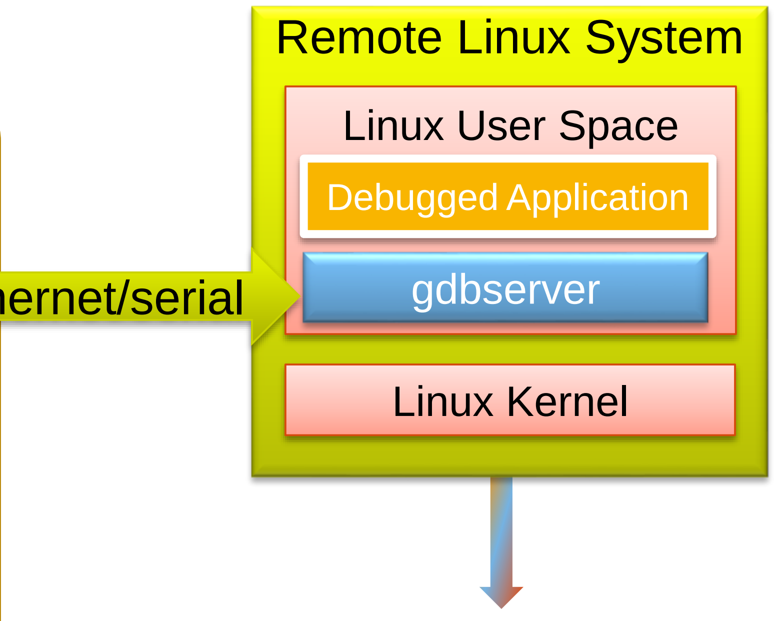
Linux Application Debug – Prerequisites

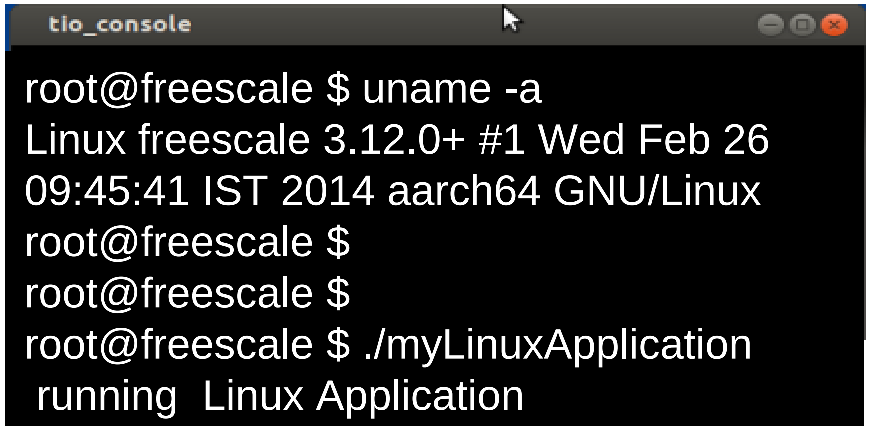
- **LS2085A RDB** board
- **Linux** running on the target
- **Network connectivity** inside Linux
- **GDB server** debug agent on the target
- Ways of putting GDB server on the target
  - GDB server is included by default in the SDK image – no change required
  - Compile GDB Agent separately
    - **bitbake –c cleansstate gdb**
    - **bitbake gdb**
    - Use SCP to put GDBAgent on the target (we'll find the **ELF** in /home/class/ LS2085A-SDK-20160304-yocto/build_ls2085ardb_release/tmp/work/ aarch64-fsl-linux/gdb/7.8.2+fsl-r0/build/gdb/gdbserver/gdbserver)

Linux Application Debug



```
(gdb) break main
(gdb) continue
Continuing.
```





```
root@freescale $ uname -a
Linux freescale 3.12.0+ #1 Wed Feb 26
09:45:41 IST 2014 aarch64 GNU/Linux
root@freescale $
root@freescale $
root@freescale $ ./myLinuxApplication
 running  Linux Application
```

- CodeWarrior project: project wizard
- Eclipse DSF – automatic download and launch application
- Run Control: run/suspend/step
- Breakpoints
- Registers View: GPR registers

ACTIVITY

Linux Application Debug – Simple Example

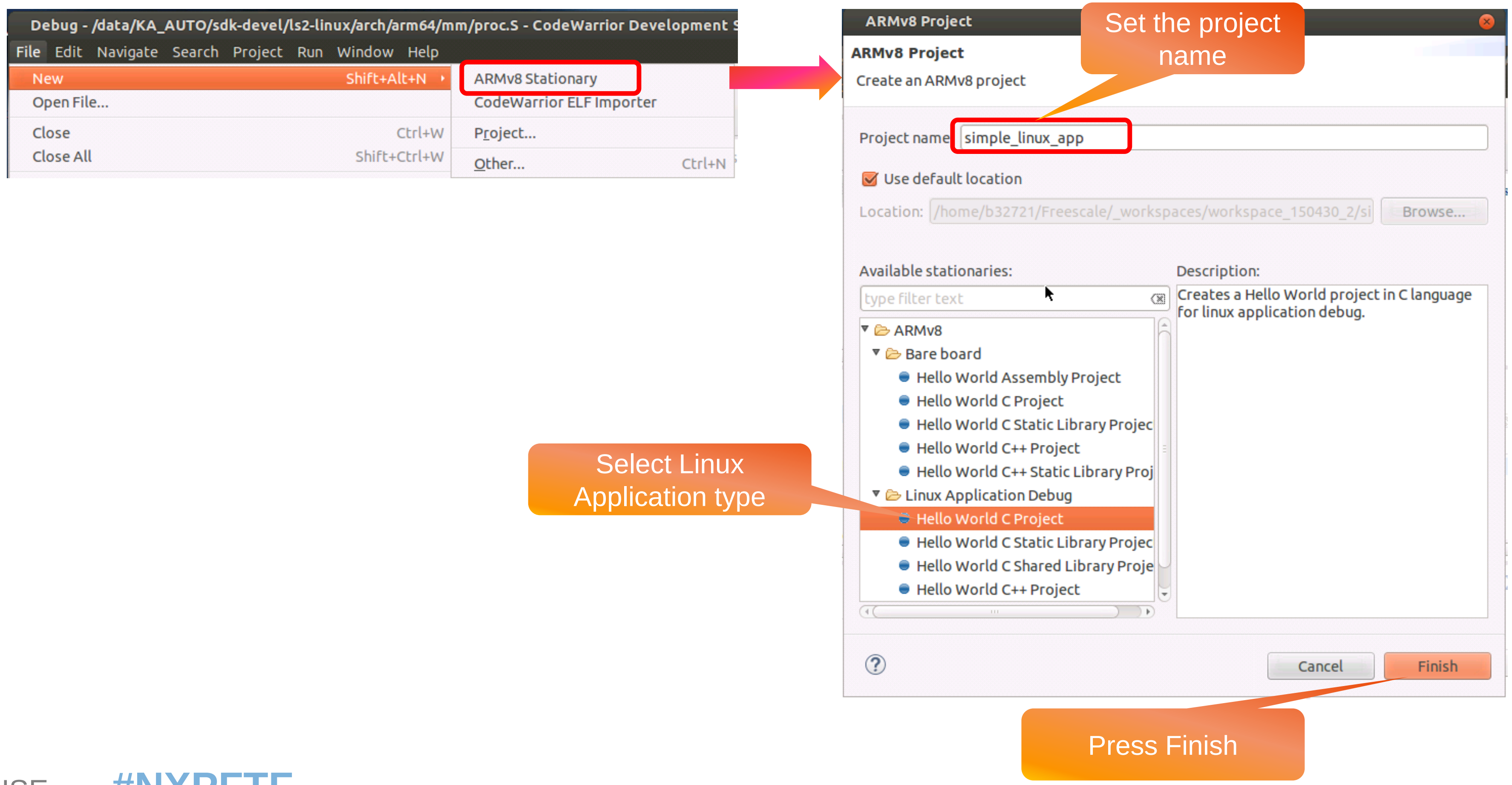
Linux Application Debug – Simple Example

Summary:

- Create a simple CodeWarrior project
- Configure the project to debug the remote target:
  - Remote IP to 192.168.1.100
  - Set sysroot for remote target
- Start the Debug session

Debugging a Simple Linux Application Debug Project – Activity

- Select File > New > ARMv8 Stationary
- Set the project name, select Linux Application and press Finish button



Debug - /data/KA_AUTO/sdk-devel/ls2-linux/arch/arm64/mm/proc.S - CodeWarrior Development S

ARMv8 Project

ARMv8 Project

Create an ARMv8 project

Project name: simple_linux_app

☑ Use default location

Location: /home/b32721/Freescale/_workspaces/workspace_150430_2/si

Available stationaries:

type filter text
ARMv8
Bare board
Hello World Assembly Project
Hello World C Project
Hello World C Static Library Projec
Hello World C++ Project
Hello World C++ Static Library Proj
Linux Application Debug
Hello World C Project
Hello World C Static Library Projec
Hello World C Shared Library Proje
Hello World C++ Project
Browse...
Cancel
Finish

Description:

Creates a Hello World project in C language for linux application debug.

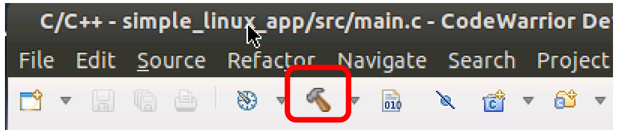
Set the project name

Select Linux Application type

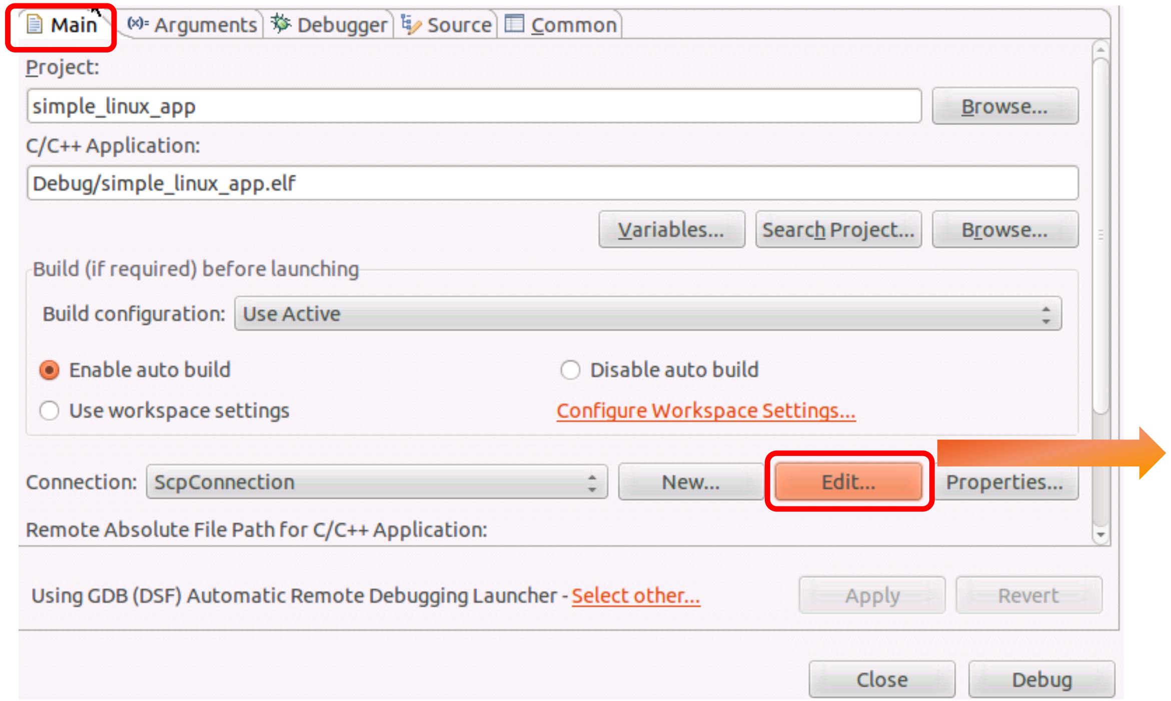
Press Finish

Debugging a Simple Linux Application Debug Project – Activity

- Build Project


C/C++ - simple_linux_app/src/main.c - CodeWarrior De
File Edit Source Refactor Navigate Search Project

- Configure project settings: Run -> Debug Configurations and select the project from C/C++ Remote Application



Project:
simple_linux_app
Browse...

C/C++ Application:
Debug/simple_linux_app.elf
Variables... Search Project... Browse...

Build (if required) before launching

Build configuration: Use Active

● Enable auto build ○ Disable auto build

○ Use workspace settings Configure Workspace Settings...

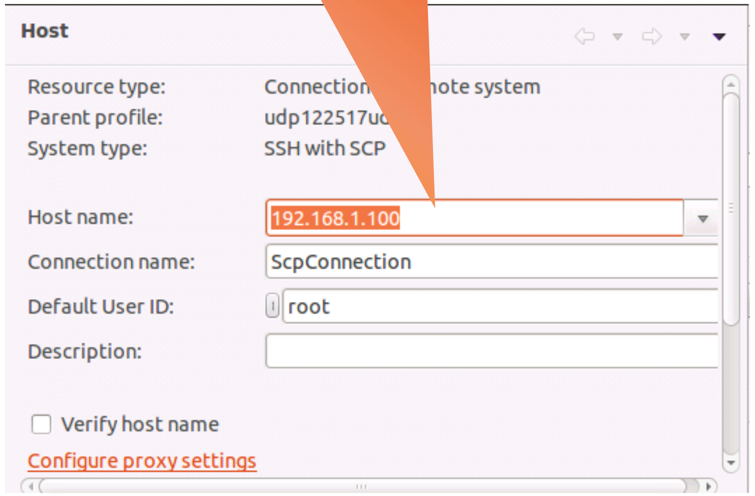
Connection: ScpConnection New... Edit... Properties...

Remote Absolute File Path for C/C++ Application:

Using GDB (DSF) Automatic Remote Debugging Launcher - Select other... Apply Revert

Close Debug

Set the IP address of the remote Linux target
192.168.1.100


Host

Resource type: Connection ... ote system
Parent profile: udp122517u...
System type: SSH with SCP

Host name: 192.168.1.100
Connection name: ScpConnection
Default User ID: root
Description:

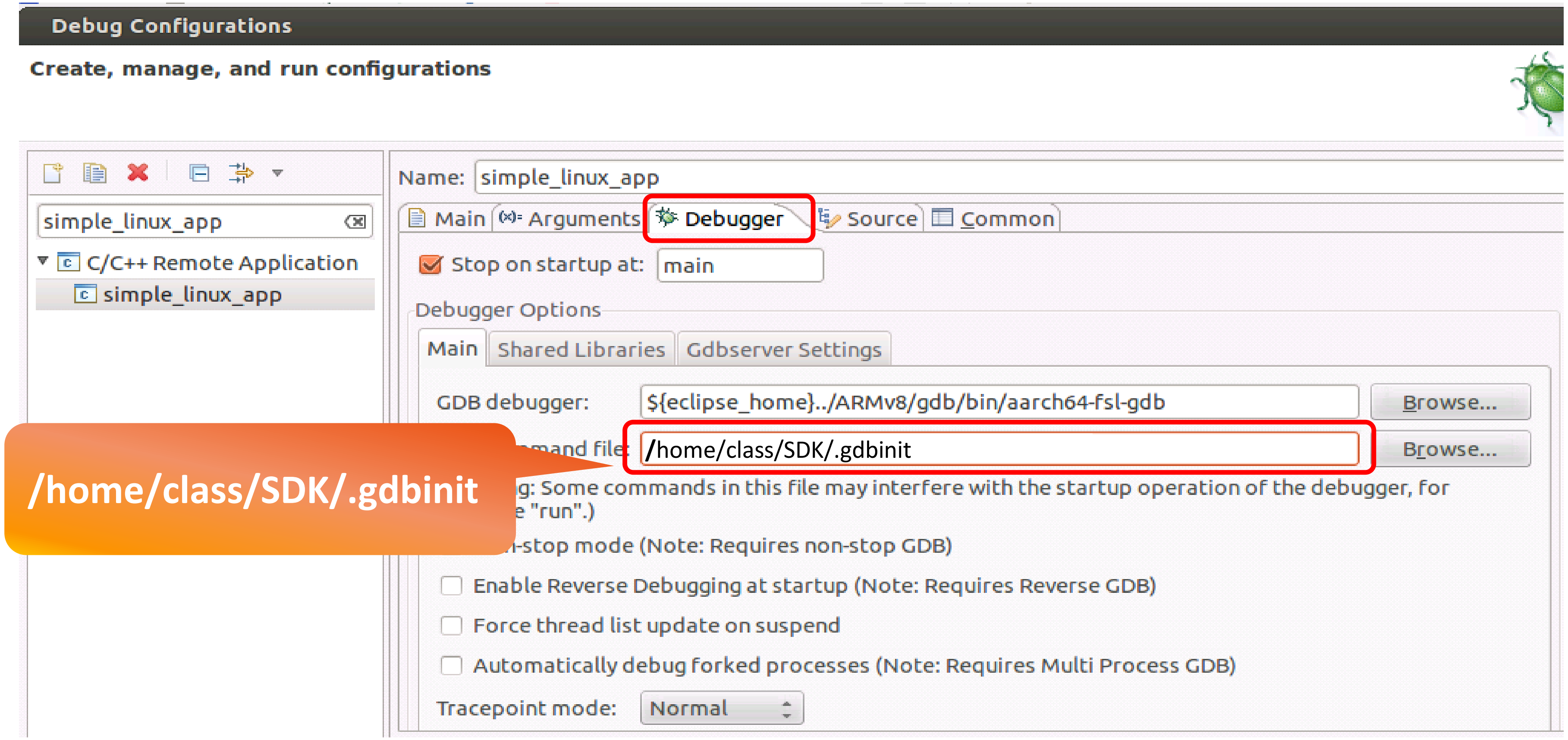
□ Verify host name

Configure proxy settings

Debugging a Simple Linux Application Debug Project – Activity

Set the gdb initialization file where the sysroot is set. .gdbinit file contains:

```
set sysroot /home/class/SDK/LS2085A-SDK-20160304-yocto/build_ls2085ardb_release/tmp/sysroots/ls2085ardb
```



Debug Configurations

Create, manage, and run configurations

simple_linux_app

- C/C++ Remote Application
  - simple_linux_app

Name: simple_linux_app

Main | Arguments | Debugger | Source | Common

☑ Stop on startup at: main

Debugger Options

Main | Shared Libraries | Gdbserver Settings

GDB debugger: ${eclipse_home}../ARMv8/gdb/bin/aarch64-fsl-gdb — Browse...

mand file: /home/class/SDK/.gdbinit — Browse...

g: Some commands in this file may interfere with the startup operation of the debugger, for e "run".)

-stop mode (Note: Requires non-stop GDB)

□ Enable Reverse Debugging at startup (Note: Requires Reverse GDB)

□ Force thread list update on suspend

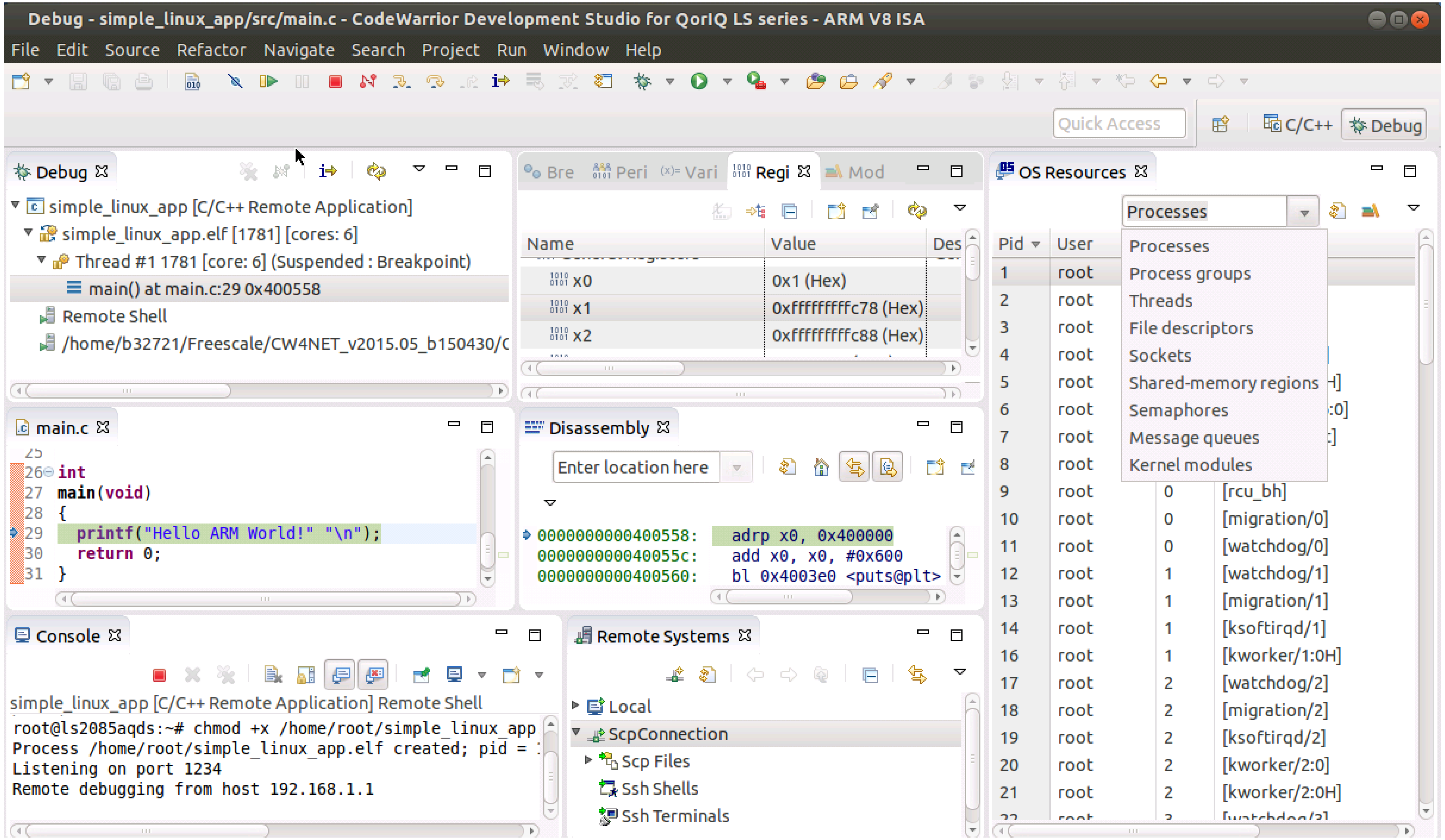
□ Automatically debug forked processes (Note: Requires Multi Process GDB)

Tracepoint mode: Normal

/home/class/SDK/.gdbinit

Debugging a Simple Linux Application Debug Project – Activity

Press Debug button and perform usual debugging



simple_linux_app [C/C++ Remote Application]
simple_linux_app.elf [1781] [cores: 6]
Thread #1 1781 [core: 6] (Suspended : Breakpoint)
main() at main.c:29 0x400558
Remote Shell
/home/b32721/Freescale/CW4NET_v2015.05_b150430/C

Name	Value	Des
x0	0x1 (Hex)	
x1	0xfffffffffc78 (Hex)	
x2	0xfffffffffc88 (Hex)	

Processes
Process groups
Threads
File descriptors
Sockets
Shared-memory regions
Semaphores
Message queues
Kernel modules

Pid	User		
1	root		
2	root		
3	root		
4	root		
5	root		H]
6	root		:0]
7	root		]
8	root		
9	root	0	[rcu_bh]
10	root	0	[migration/0]
11	root	0	[watchdog/0]
12	root	1	[watchdog/1]
13	root	1	[migration/1]
14	root	1	[ksoftirqd/1]
16	root	1	[kworker/1:0H]
17	root	2	[watchdog/2]
18	root	2	[migration/2]
19	root	2	[ksoftirqd/2]
20	root	2	[kworker/2:0]
21	root	2	[kworker/2:0H]
22	root	3	[watchdog/3]

```
int
main(void)
{
  printf("Hello ARM World!" "\n");
  return 0;
}
```

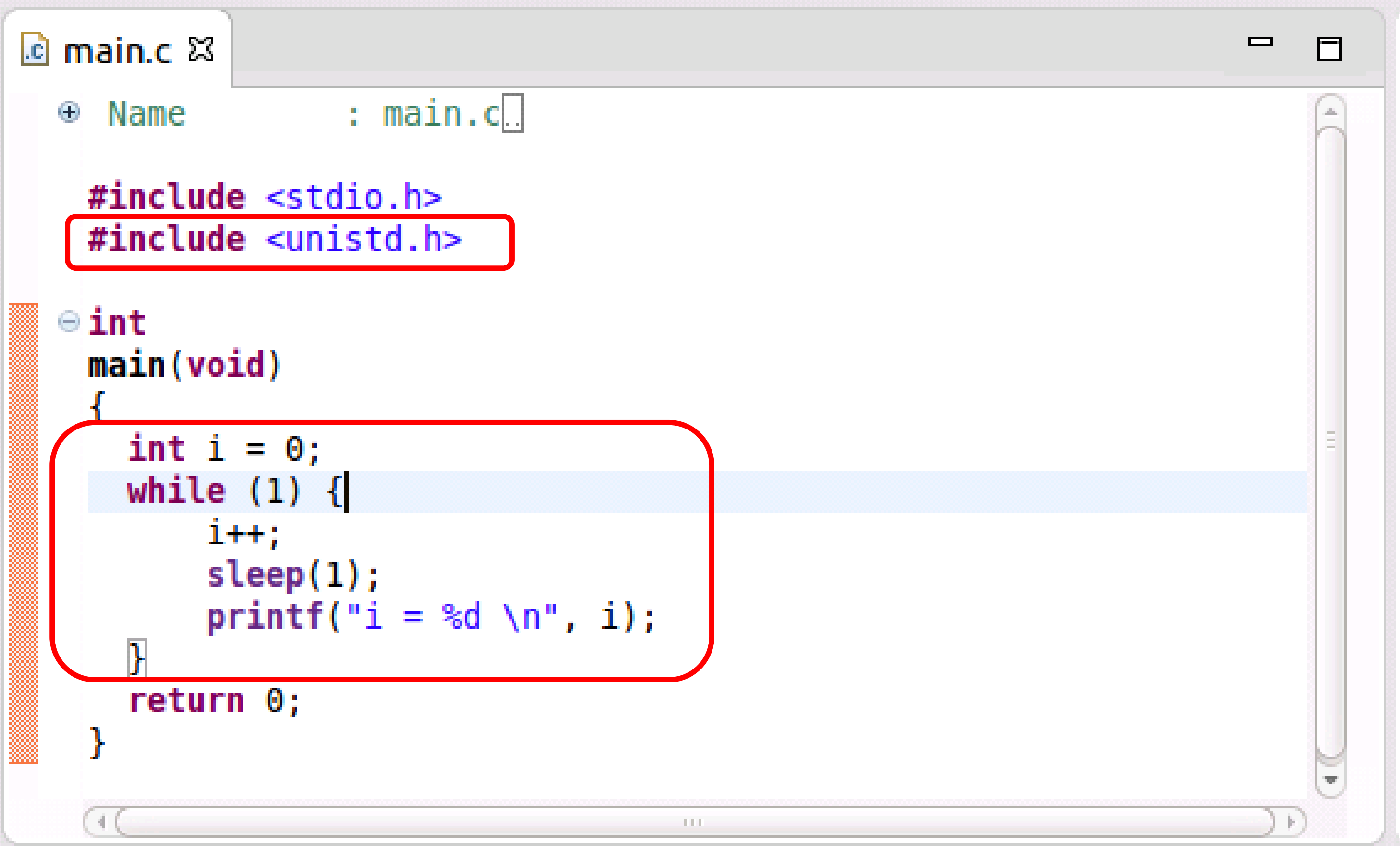
```
0000000000400558:   adrp x0, 0x400000
000000000040055c:   add x0, x0, #0x600
0000000000400560:   bl 0x4003e0 <puts@plt>
```

simple_linux_app [C/C++ Remote Application] Remote Shell

```
root@ls2085aqds:~# chmod +x /home/root/simple_linux_app
Process /home/root/simple_linux_app.elf created; pid = :
Listening on port 1234
Remote debugging from host 192.168.1.1
```

Debugging a Simple Linux Application Debug Project – Activity

Change the Linux application: add an infinite loop



```
Name          : main.c

#include <stdio.h>
#include <unistd.h>

int
main(void)
{
    int i = 0;
    while (1) {
        i++;
        sleep(1);
        printf("i = %d \n", i);
    }
    return 0;
}
```

Debugging a Simple Linux Application Debug Project – Activity

- Notes:
  - CodeWarrior automatically connects to the remote target (over ssh) start the gdbserver on the configured port, debugging the current application
  - No need for the user to connect to target and configure or run programs
  - OS Resources Window provides system information: processes, threads, sockets, shared memory…
  - From CodeWarrior you can open a terminal/shell to target

ACTIVITY

Attach to an Existing Linux Process

Attach to an Existing Linux Process

Summary:

- Start the gdbserver for attaching to application
- Manually start the application
- Configure the project to debug the remote target:
  - Remote IP to 192.168.1.100
  - Set sysroot for remote target
- Start the Debug session: attach to the existing application

Attach to a Running Linux Application Example – Activity (prerequisites)

- The application **simple_linux_app** should be copied on target (using a CodeWarrior download session or manually using **scp**)
- Assume a **ssh/telnet console** is active on the target board

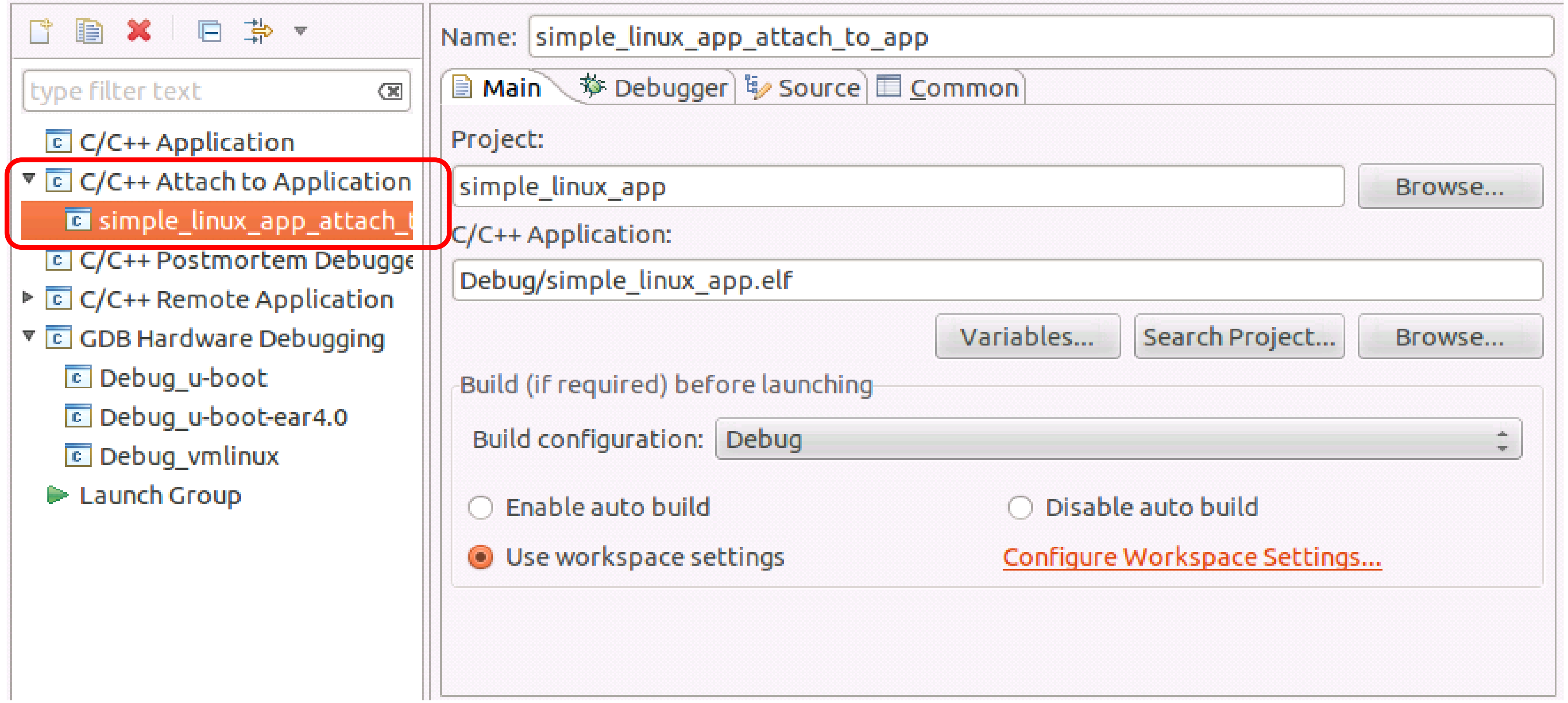
- Manually start the gdbserver to allow attaching to any linux application:
- Run the application on target

```
root@ls2085ardb:~# gdbserver --multi :1234 &
[1] 1737
Listening on port 1234
root@ls2085ardb:~# ./simple_linux_app.elf
```

- gdbsever and application are running on target.

Attach to a Running Linux Application Example – Activity

- Open **Debug Configurations: Run → Debug Configurations**
- For C/C++ Attach to Application: create a new launch
- The Main tab will automatically be completed



type filter text

- C/C++ Application
- C/C++ Attach to Application
  - simple_linux_app_attach_t
- C/C++ Postmortem Debugge
- C/C++ Remote Application
- GDB Hardware Debugging
  - Debug_u-boot
  - Debug_u-boot-ear4.0
  - Debug_vmlinux
- Launch Group

Name: simple_linux_app_attach_to_app

Main | Debugger | Source | Common

Project:
simple_linux_app

Browse...

C/C++ Application:
Debug/simple_linux_app.elf

Variables... | Search Project... | Browse...

Build (if required) before launching

Build configuration: Debug

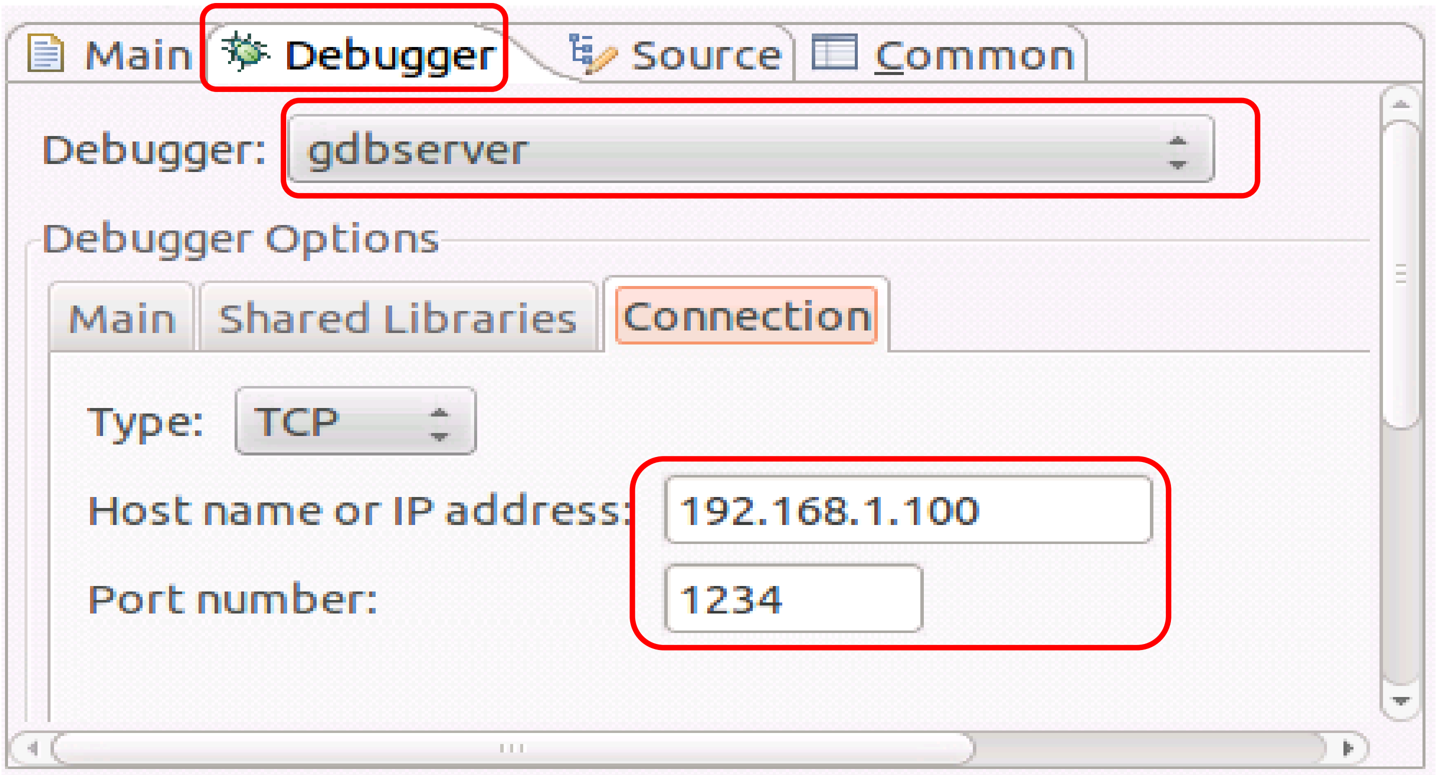
○ Enable auto build ○ Disable auto build

● Use workspace settings

Configure Workspace Settings...

Attach to a Running Linux Application Example – Activity

- In the Debugger tab:
  - Main sub-tab: add gdbinit file
  - Connection sub-tab: set the target parameters: IP address and port
- Press Debug button to start debugging



Debugger: gdbserver

Debugger Options

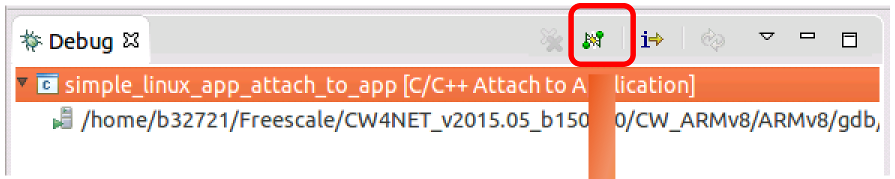
Type: TCP

Host name or IP address: 192.168.1.100

Port number: 1234

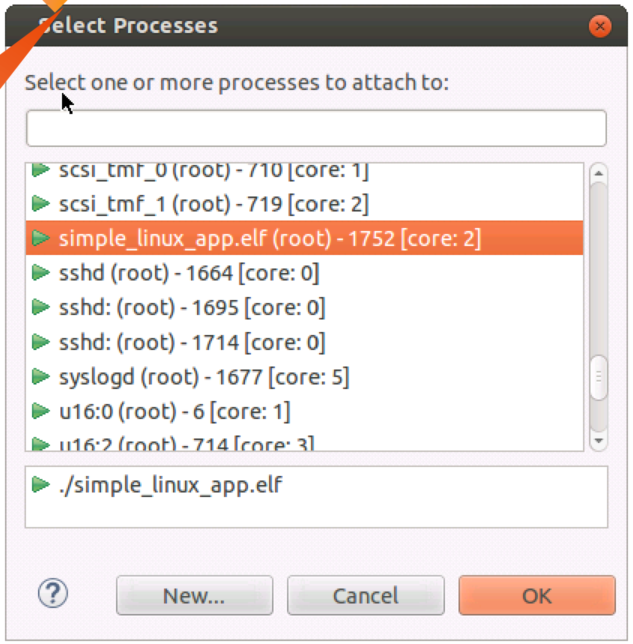
Attach to a Running Linux Application Example – Activity

- Hit “Connect to a process” icon to open the pop-up dialog for selecting the application



simple_linux_app_attach_to_app [C/C++ Attach to A lication]
/home/b32721/Freescale/CW4NET_v2015.05_b150 0/CW_ARMv8/ARMv8/gdb,

- From the pop-up dialog select the relevant running application
- The debugger will attach and user will be able to **suspend** and debug the application as usual

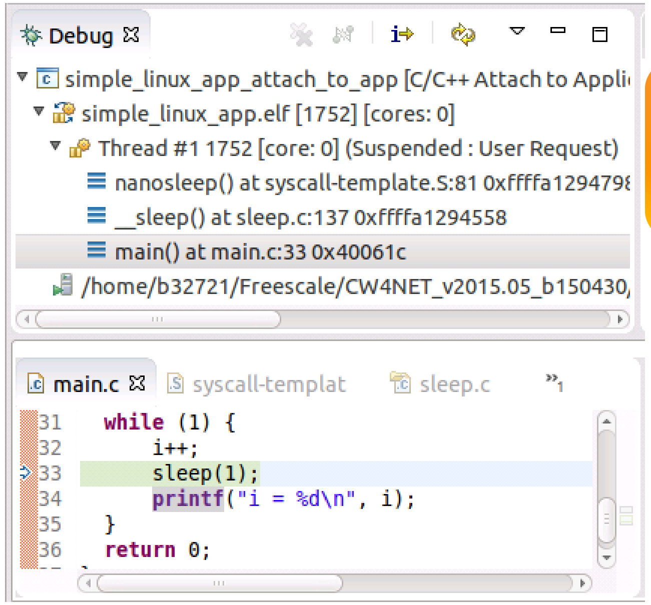


Select one or more processes to attach to:

```
scsi_tmf_0 (root) - 710 [core: 1]
scsi_tmf_1 (root) - 719 [core: 2]
simple_linux_app.elf (root) - 1752 [core: 2]
sshd (root) - 1664 [core: 0]
sshd: (root) - 1695 [core: 0]
sshd: (root) - 1714 [core: 0]
syslogd (root) - 1677 [core: 5]
u16:0 (root) - 6 [core: 1]
u16:2 (root) - 714 [core: 3]
```

./simple_linux_app.elf

You can filter processes list
simple_*



```
simple_linux_app_attach_to_app [C/C++ Attach to Appli
  simple_linux_app.elf [1752] [cores: 0]
    Thread #1 1752 [core: 0] (Suspended : User Request)
      nanosleep() at syscall-template.S:81 0xffffa129479
      __sleep() at sleep.c:137 0xffffa1294558
      main() at main.c:33 0x40061c
  /home/b32721/Freescale/CW4NET_v2015.05_b150430,
```

```
	while (1) {
		i++;
		sleep(1);
		printf("i = %d\n", i);
	}
	return 0;
```

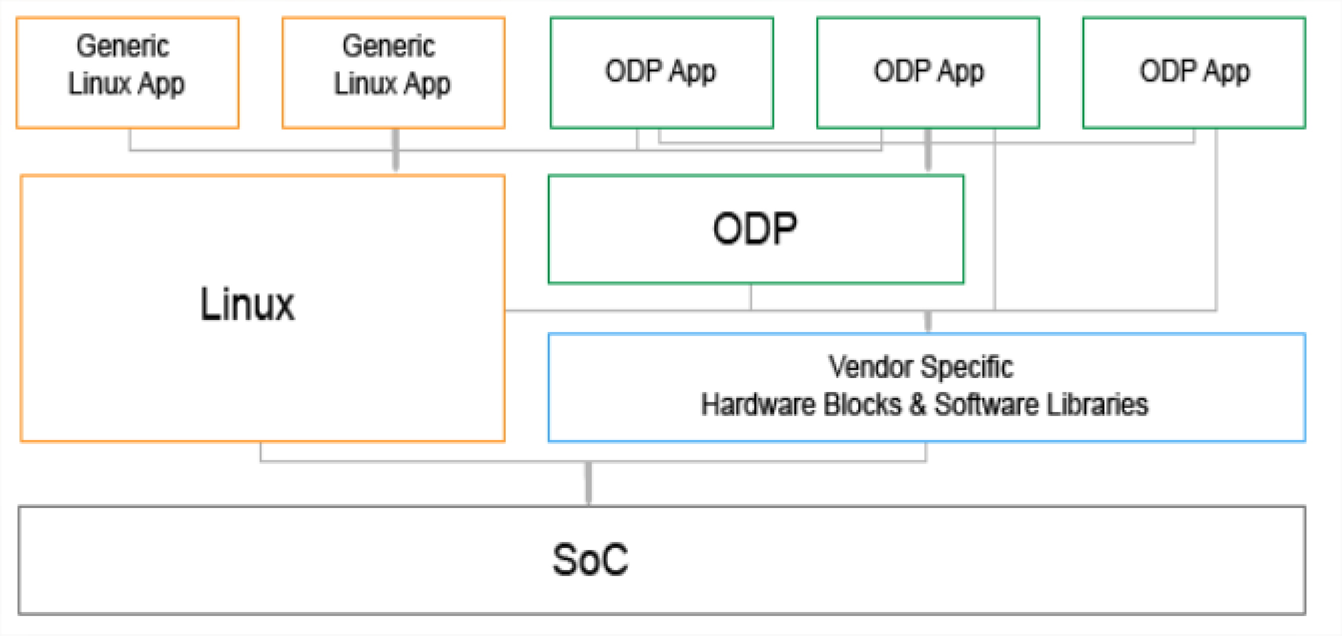
ODP REFLECTOR DEBUG

- Demonstrate the ODP reflector usage and debug capabilities with CodeWarrior
  - Introduction to ODP
  - Hardware setup
  - ODP reflector software configuration
  - ODP reflector import in CodeWarrior and debug

Introduction to ODP

What is ODP?

- The **Open Data Plane** (ODP) project has been established to produce an open-source, cross-platform set of application programming interfaces (APIs) for the networking data plane
- ODP provides a data plane application programming environment that is easy to use, high performance and portable between networking SoCs



Introduction to ODP Reflector Application

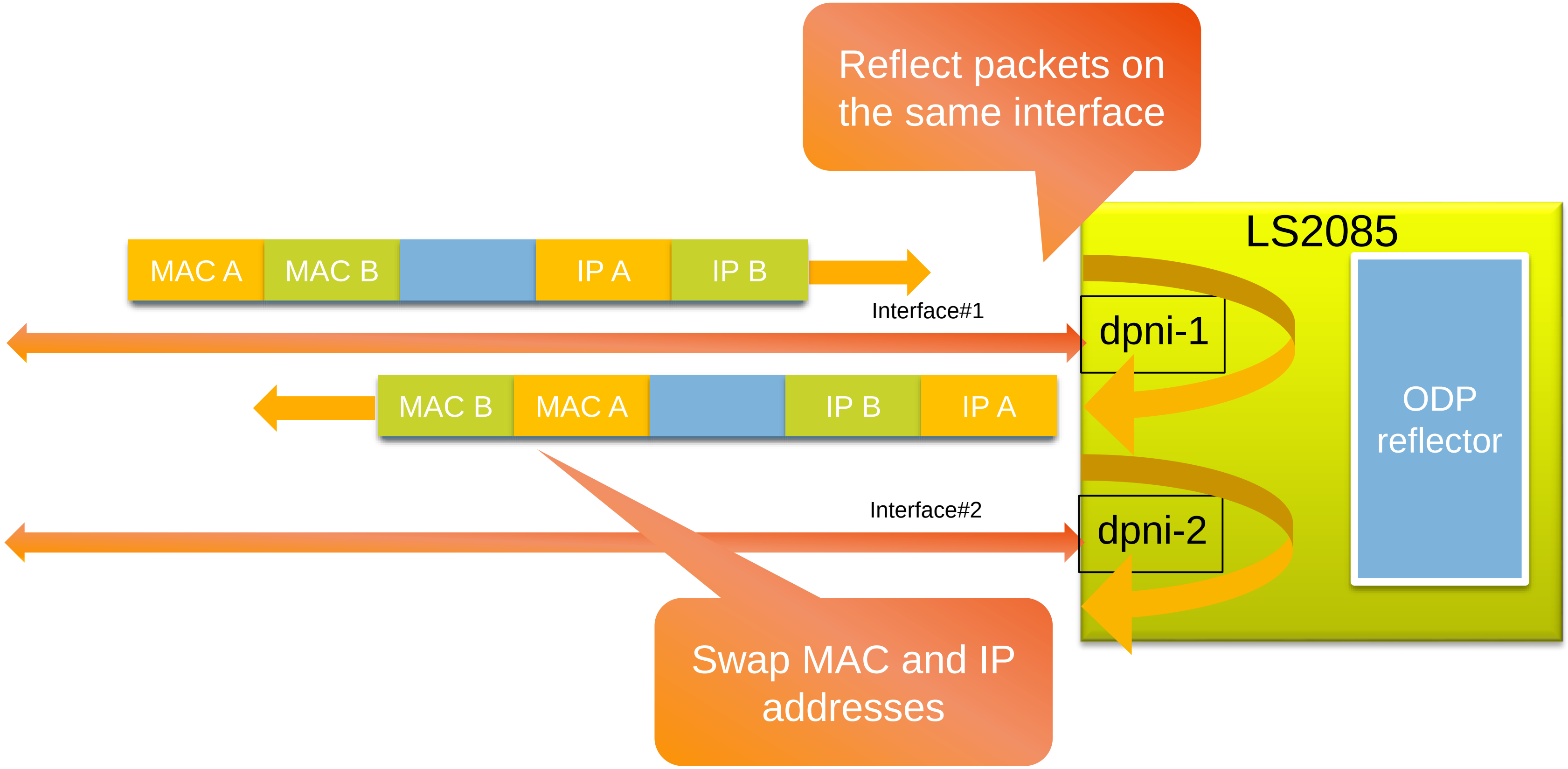
Linux user space demo application to demonstrate ODP and networking capabilities of QorIQ LS2085A processor.

ODP Reflector performs several functions:

- Received scheduled packets are reflected back onto the same interface where the packets were originally received
- The source and destination MAC and IP addresses are swapped in received packet
- Works for all Ethernet interfaces that are defined in the resource container used by the application
- Multiple threads can be spawned for each network interface for I/O operation. In multicore environment, threads are affined with multiple cores. For single core environment, all threads are affined with the same core

Reference: SDK documentation: Open_Data_Plane_Example_Applicatins_User_Manual_RevB.pdf

Introduction to ODP Reflector Application

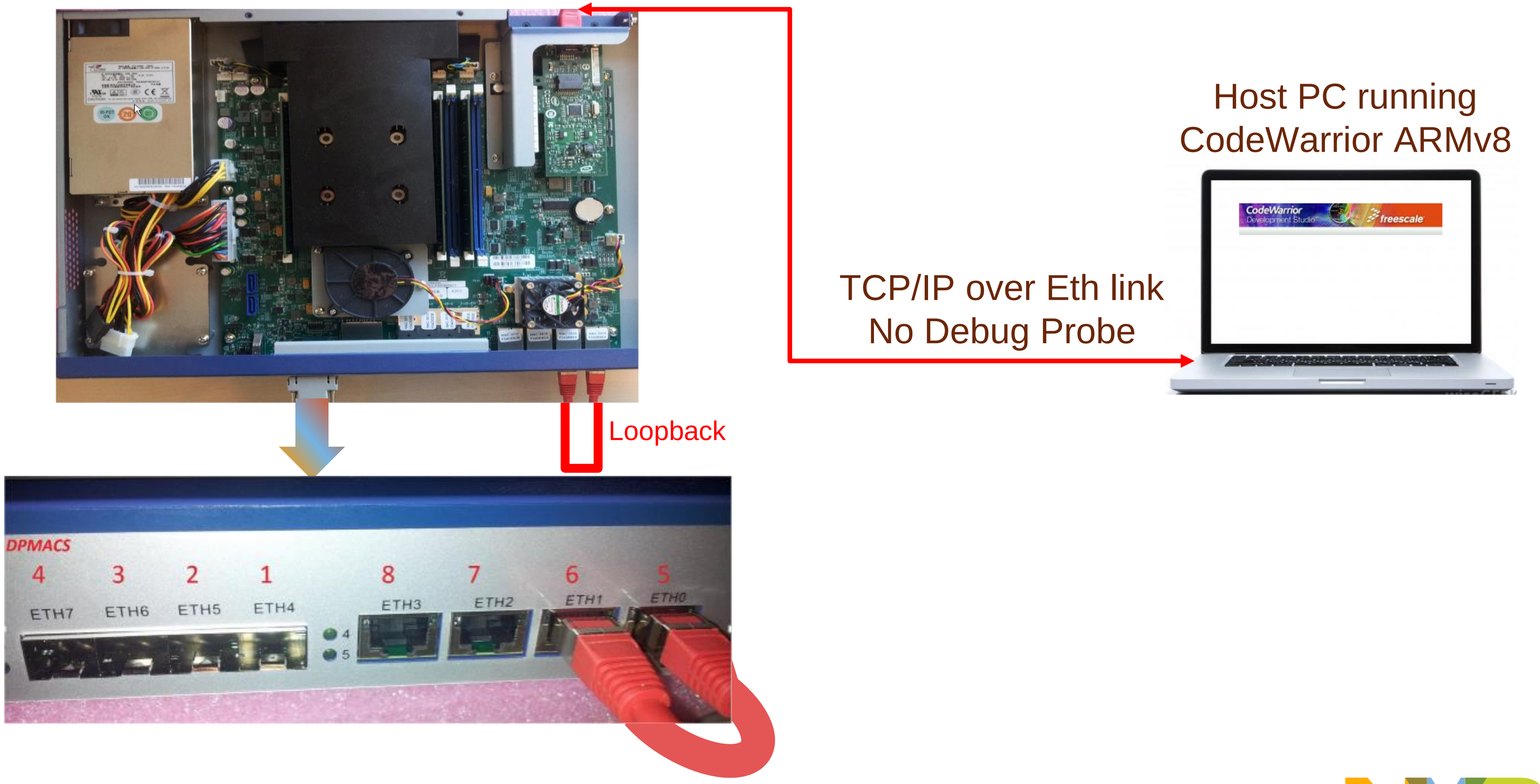


Configuration Options:

- one or more Eth interface. DPNI = Data Path Network Interface
- Scheduling options (PULL or PUSH mode)
- Number of CPU to use

ODP Reflector – Hardware Setup Using Only One Board

For full details and steps describing the hardware and software setup please check *AN5269*



LS2085A-RDB

Host PC running
CodeWarrior ARMv8

TCP/IP over Eth link
No Debug Probe

Loopback

ODP Reflector – Software Installation Prerequisites

Linux SDK for QorIQ LS2085A EAR 6.0

(the following steps were already done on the class machines)

- Install SDK on the host Linux machine

```
$ sudo mount -o loop LS2085A-SDK-20160304-yocto /mnt/cdrom
$ /mnt/cdrom/install -> install SDK in /home/class/SDK
$ cd /home/class/SDK/LS2085A-SDK-20160304-yocto
$ source ./poky/fsl-setup-poky -m ls2085ardb
```

- Configure ODP reflector to build the reflector with debug symbols

```
In file /home/class/SDK/LS2085A-SDK-20160304-yocto/meta-fsl-
networking/recipes-dpaa2/odp/odp.inc, add the following line:

CFLAGS = "-pipe -ggdb -feliminate-unused-debug-types"
```

- Build ODP reflector application

```
$ bitbake odp
$ bitbake fsl-image-kernelitb -> optional to build full distribution
```

ODP Reflector – Target Configuration Prerequisites

Configure LS2085A-RDB:

- Setup flash with images from Linux SDK for QorIQ LS2085A EAR 6.0
    - U-boot
    - Linux Kernel and rootfs
    - Data Path Layout: default configure Linux interface **ni0** for MAC5 port
    - Documentation: *QorIQ LS2085A EAR 6.0 Deployment Guide*

- Management port on PCI card configured in target Linux as interface **eth0**. It is connected with an Ethernet cable to Linux host computer.

- ODP reflector application build with debug symbols available in rootfs
    - After adding debug symbols for ODP, build distribution and deploy to target

ACTIVITY

Run ODP Reflector Application

Run ODP Reflector Application

Summary

- Configure the target resources
- Verify configuration using resource management tool
- Start the ODP Reflector Application
- View the results: ping working through physical loopback and ODP Reflector

ODP Reflector – Using ODP Reflector Application

Configuration

1 Set ip to ni0 interface used for Linux Container

2 Add arp entry – all traffic to 6.6.6.10 will be redirect to dpni1 (which dmpac.6 – 000000000006)

3 Set ip to eth0 interface used by communication with CW

4 Allocate a new dpni (dpni.1) to dpmac.6 using restool via dynamic_dpl.sh utility script

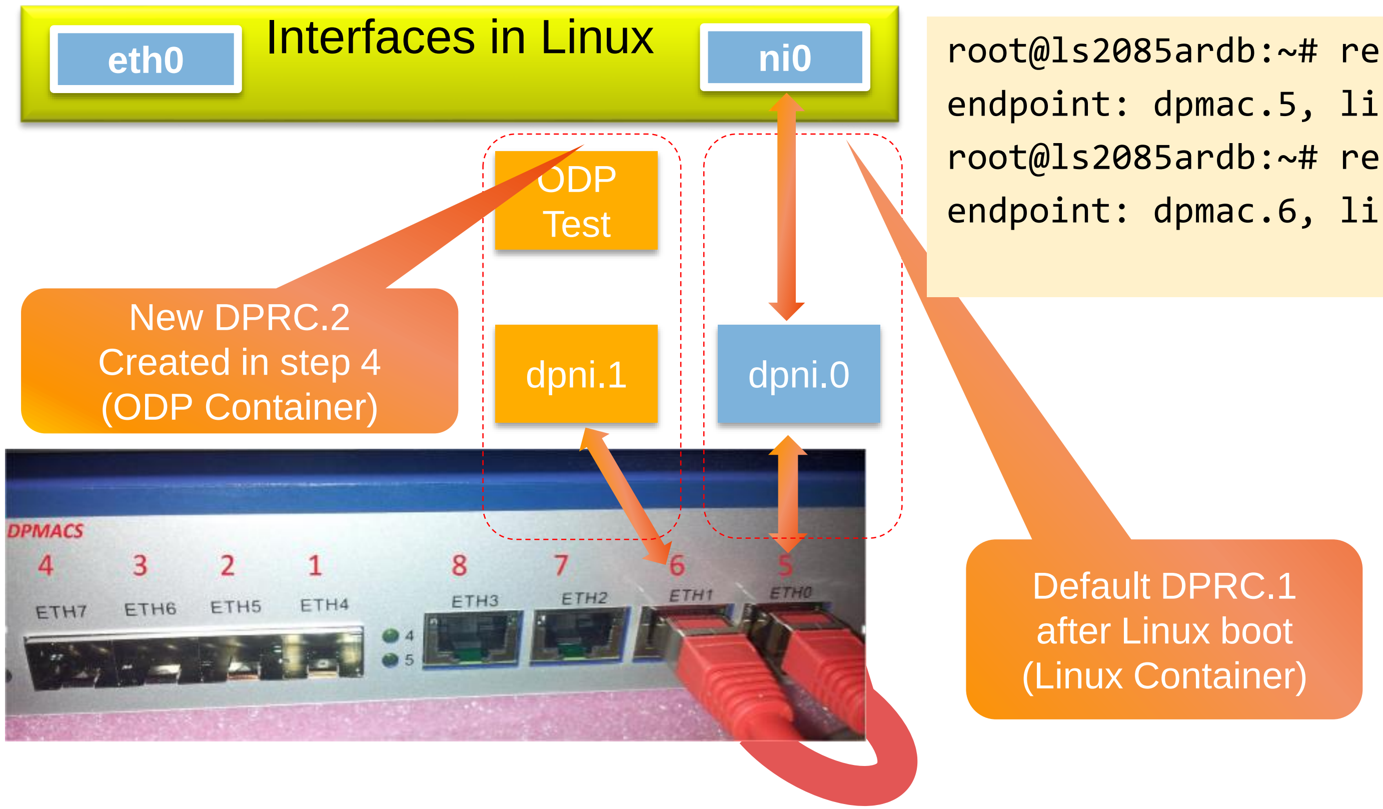
DPRC = Data Path Resource Container

```
1 ifconfig ni0 6.6.6.1 up

2 arp -s 6.6.6.10 000000000006

3 ifconfig eth0 192.168.1.100

4 /usr/odp/scripts/dynamic_dpl.sh dpmac.6
…
dprc.2 Created
dpmac.6 <--------connected------> dpni.1 (00:00:00:00:0:6)
USE  dprc.2  FOR YOUR APPLICATIONS
```

ODP Reflector – Using ODP Reflector Application

Verification

Using **restool:** DPAA resource management tool to verify the DPNI status



```
root@ls2085ardb:~# restool dpni info dpni.0
endpoint: dpmac.5, link is up
root@ls2085ardb:~# restool dpni info dpni.1
endpoint: dpmac.6, link is down
```

ODP Reflector – Using ODP Reflector Application

Starting ODP reflector application

1. Set the ODP container
2. Start the odp_reflector on dpni.1 in PULL mode, using all 8 CPUs

1

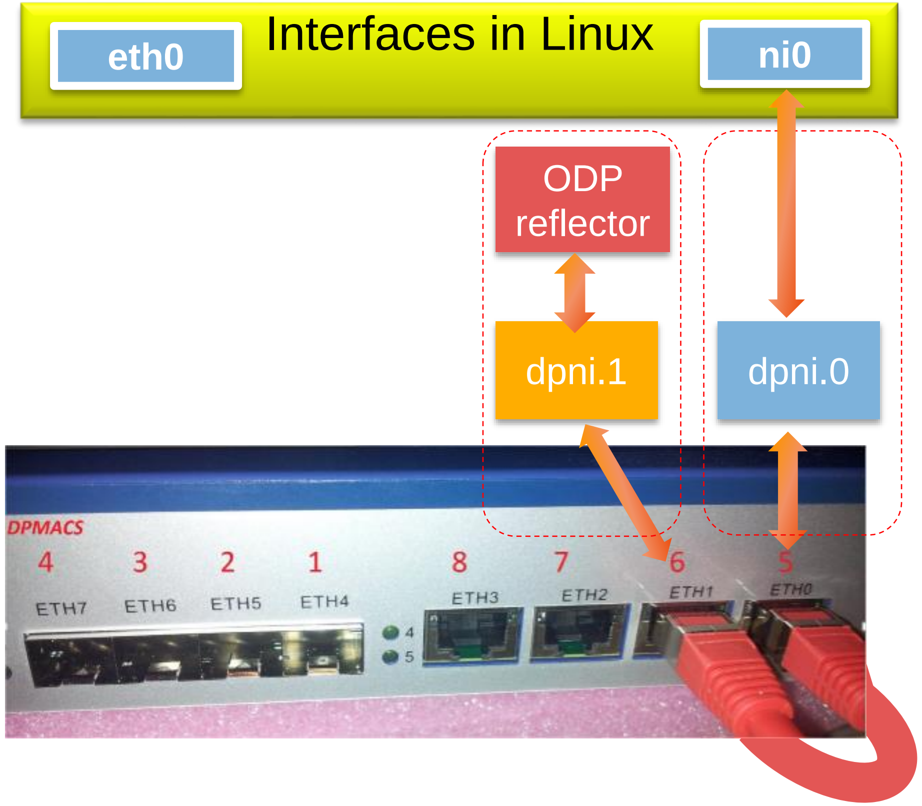
```
root@ls2085ardb:~# export DPRC=dprc.2
```

2

```
root@ls2085ardb:~# /usr/odp/bin/odp_reflector -i dpni-1 -m 0 -c 8 &
Initializing NADK framework with following parameters:
        Resource container :dprc.2
…
setup_pkt_nadk 55-NOTICE-port =>  dpni-1 being created
setup_pkt_nadk 66-NOTICE-setup FQ 0
Port dpni-1 = Mac 00.00.00.00.00.06
<enter>
```

ODP Reflector – Using ODP Reflector Application

Verification after ODP reflector has been started



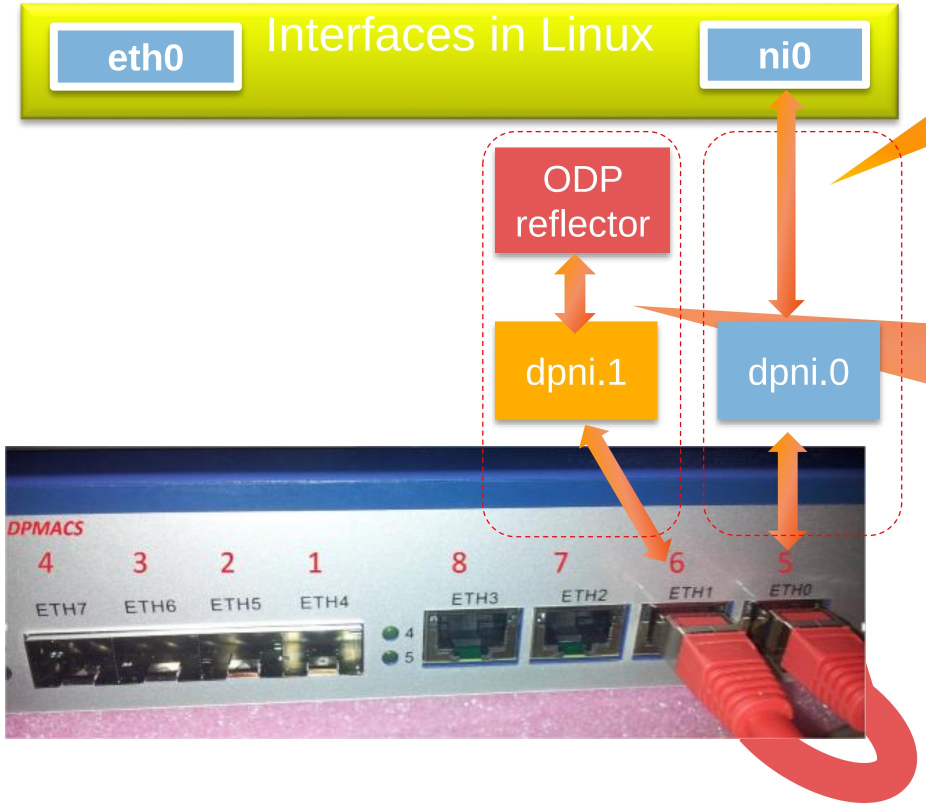
```
root@ls2085ardb:~# restool dpni info dpni.0
endpoint: dpmac.5, link is up
root@ls2085ardb:~# restool dpni info dpni.1
endpoint: dpmac.6, link is up
```

ODP reflector is connected to dpni.1

ODP Reflector – Using ODP Reflector Application

View the results

- Summary for setup configuration: MAC and IP addresses



Linux Container

IP : 6.6.6.1

```
ifconfig ni0 6.6.6.1 up
```

ODP Reflector Container

IP : 6.6.6.10

MAC: 00:00:00:00:00:06

```
arp -s 6.6.6.10 000000000006
```

ODP Reflector – Using ODP Reflector Application

View the results

- Start a network packet capture (tcpdump) to inspect packets sent and received on **ni0** interface
- Send ping request to IP:

1 - **Echo request is sent from Linux on ni0 interface**

- **IP address of the ODP Reflector Container**
- **Using MAC address of dpmac.5**

```
root@ls2085ardb:~# tcpdump -i ni0 &
<enter>
root@ls2085ardb:~# ping 6.6.6.10 -c 1
1 IP 6.6.6.1 > 6.6.6.10: ICMP echo request, id 1953, seq 1
```

ODP Reflector – Using ODP Reflector Application

View the results

- Start a network packet capture (tcpdump) to inspect packets sent and received on **ni0** interface
- Send ping request to IP:
  1 - **Echo request is sent from Linux on ni0 interface**
  2 - Echo request is reflected back swapping MAC and IP addresses

```
  root@ls2085ardb:~# tcpdump -i ni0 &
  <enter>
  root@ls2085ardb:~# ping 6.6.6.10 -c 1
1 IP 6.6.6.1 > 6.6.6.10: ICMP echo request, id 1953, seq 1
2 IP 6.6.6.10 > 6.6.6.1: ICMP echo request, id 1953, seq 1
```

ODP Reflector – Using ODP Reflector Application

View the results

- Start a network packet capture (tcpdump) to inspect packets sent and received on **ni0** interface
- Send ping request to IP:
  1 - **Echo request is sent from Linux on ni0 interface**
  2 - Echo request is reflected back swapping MAC and IP addresses
  3 - Linux networking stack on **ni0** responds to the received echo request sending an echo replay

```
  root@ls2085ardb:~# tcpdump -i ni0 &
  <enter>
  root@ls2085ardb:~# ping 6.6.6.10 -c 1
1 IP 6.6.6.1 > 6.6.6.10: ICMP echo request, id 1953, seq 1
2 IP 6.6.6.10 > 6.6.6.1: ICMP echo request, id 1953, seq 1
3 IP 6.6.6.1 > 6.6.6.10: ICMP echo reply, id 1953, seq 1
```

ODP Reflector – Using ODP Reflector Application

View the results

- Start a network packet capture (tcpdump) to inspect packets sent and received on **ni0** interface
- Send ping request to IP:

1 - **Echo request is sent from Linux on ni0 interface**

2 - Echo request is reflected back swapping MAC and IP addresses

3 - Linux networking stack on **ni0** responds to the received echo request sending an echo replay

4 - Echo reply is reflected back swapping MAC and IP. **Linux receives the echo reply on ni0**

```
  root@ls2085ardb:~# tcpdump -i ni0 &
  <enter>
  root@ls2085ardb:~# ping 6.6.6.10 -c 1
1 IP 6.6.6.1 > 6.6.6.10: ICMP echo request, id 1953, seq 1
2 IP 6.6.6.10 > 6.6.6.1: ICMP echo request, id 1953, seq 1
3 IP 6.6.6.1 > 6.6.6.10: ICMP echo reply, id 1953, seq 1
4 IP 6.6.6.10 > 6.6.6.1: ICMP echo reply, id 1953, seq 1
```

ACTIVITY

ODP Reflector Debug using CodeWarrior

ODP Reflector Debug Using CodeWarrior

Summary:

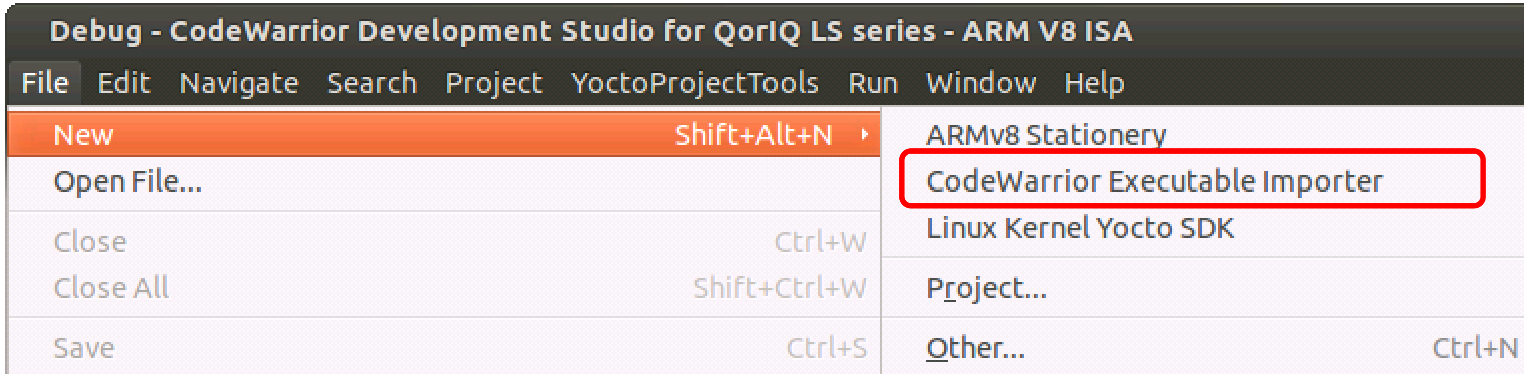
- Create the CodeWarrior project for ODP reflector debug (import executable)
- Configure the project to debug the remote target:
  - Remote IP to 192.168.1.100
  - Set sysroot for remote target
  - Configure the project to run reflector in the same way as for starting from Linux console

```
# export DPRC=dprc.2
# /usr/odp/bin/odp_reflector -i dpni-1 -m 0 -c 8 &
```

- Start the Debug session

ODP Reflector – Create CodeWarrior Project

- Open CodeWarrior using fsl_eclipse.sh script from CW_ARMv8
  - File > Import > C/C++ > CodeWarrior Executable Importer > Next

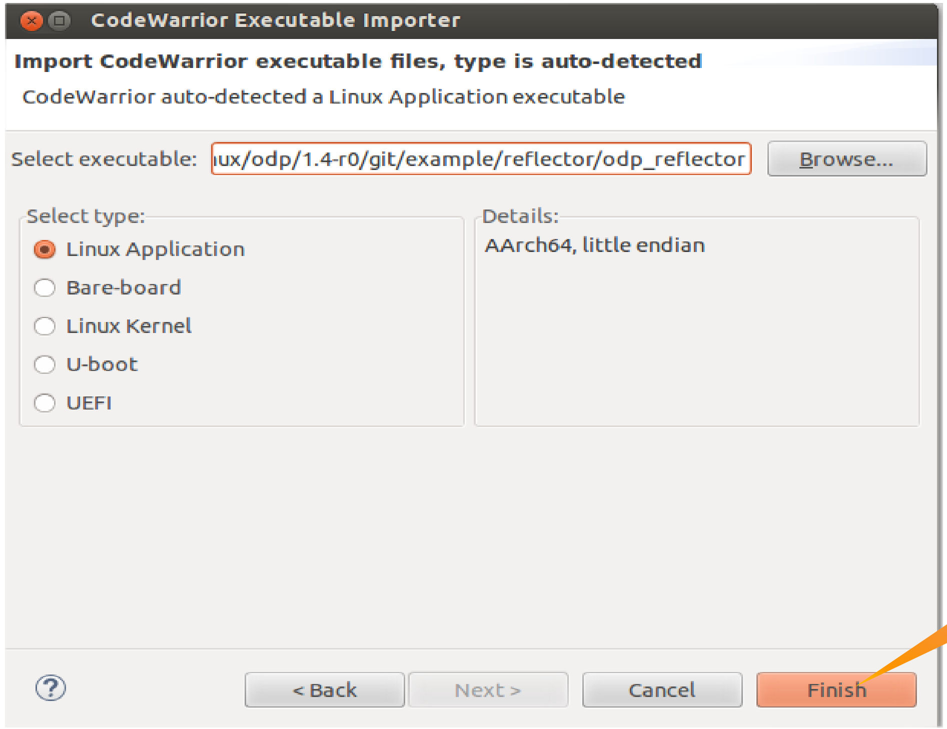


ODP Reflector – Create CodeWarrior Project

Select odp_reflector elf

/home/class/SDK/LS2085A-SDK-20160304-yocto/build_ls2085ardb_release/tmp/work/aarch64-fsl-linux/odp/1.4-r0/git/example/reflector/odp_reflector

CodeWarrior automatically detects the elf type and will make the settings for a Linux Application debug flow



CodeWarrior Executable Importer

Import CodeWarrior executable files, type is auto-detected

CodeWarrior auto-detected a Linux Application executable

Select executable: ıux/odp/1.4-r0/git/example/reflector/odp_reflector

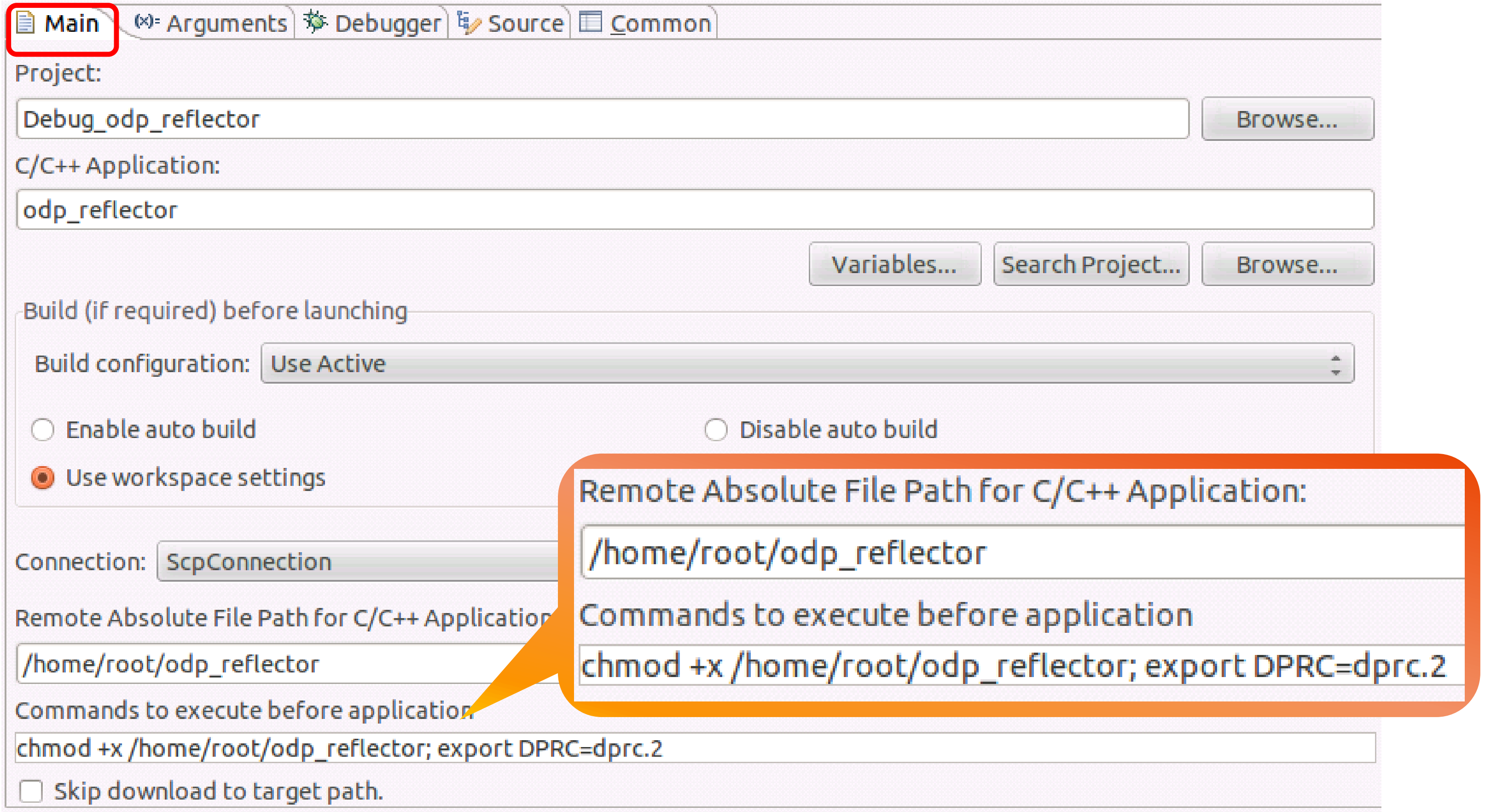
Select type:
- ● Linux Application
- ○ Bare-board
- ○ Linux Kernel
- ○ U-boot
- ○ UEFI

Details: AArch64, little endian

Click Finish

ODP Reflector – Create CodeWarrior Project

- Add the commands that will be executed before starting odp_reflector



Project:
Debug_odp_reflector

C/C++ Application:
odp_reflector

Build (if required) before launching

Build configuration: Use Active

○ Enable auto build
○ Disable auto build
● Use workspace settings

Connection: ScpConnection

Remote Absolute File Path for C/C++ Application:
/home/root/odp_reflector

Commands to execute before application
chmod +x /home/root/odp_reflector; export DPRC=dprc.2

□ Skip download to target path.

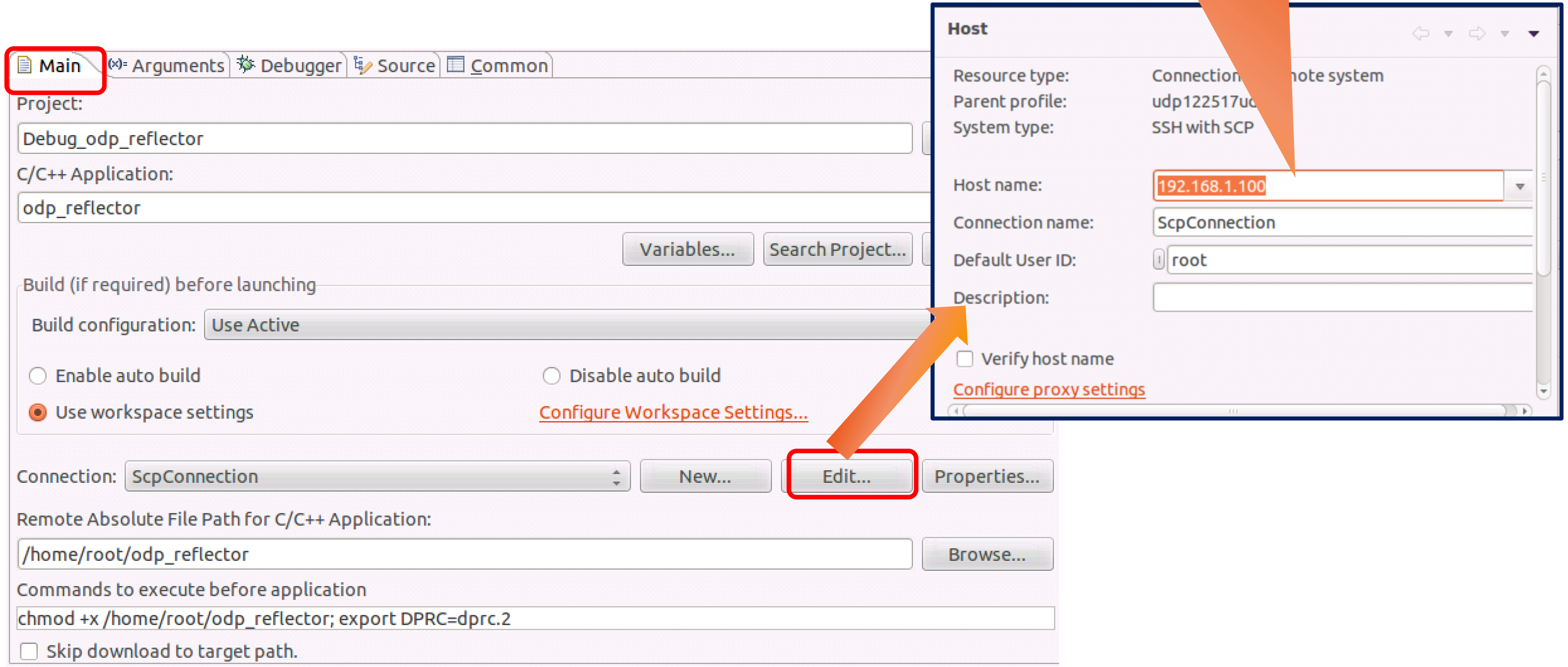
Remote Absolute File Path for C/C++ Application:
/home/root/odp_reflector

Commands to execute before application
chmod +x /home/root/odp_reflector; export DPRC=dprc.2

ODP Reflector – Configuration in CodeWarrior

- Set the Host name / IP of the Linux target (eth0) 192.168.1.100

Set the IP address of the remote Linux target
192.168.1.100



Project:
Debug_odp_reflector

C/C++ Application:
odp_reflector

Build (if required) before launching

Build configuration: Use Active

○ Enable auto build
○ Disable auto build
● Use workspace settings
Configure Workspace Settings...

Connection: ScpConnection

Remote Absolute File Path for C/C++ Application:
/home/root/odp_reflector

Commands to execute before application
chmod +x /home/root/odp_reflector; export DPRC=dprc.2

□ Skip download to target path.

Host

Resource type:	Connection ote system
Parent profile:	udp122517u
System type:	SSH with SCP
Host name:	192.168.1.100
Connection name:	ScpConnection
Default User ID:	root
Description:	

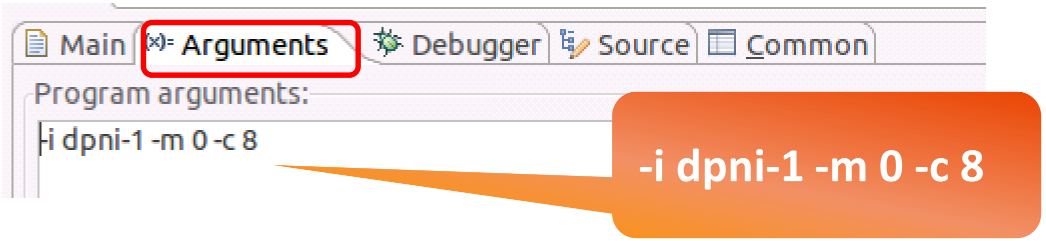
□ Verify host name

Configure proxy settings

ODP Reflector – Configuration in CodeWarrior

Set the odp_reflector arguments:

-i dpni-1 -m 0 -c 8

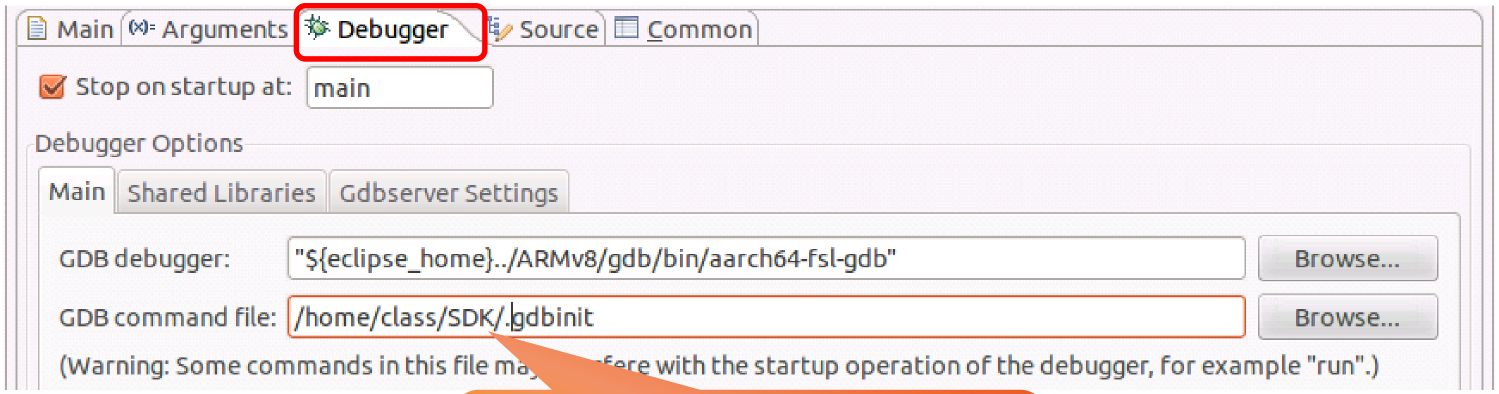


Program arguments:

```
-i dpni-1 -m 0 -c 8
```

-i dpni-1 -m 0 -c 8

Set the gdb initialization file where the sysroot is set



☑ Stop on startup at: main

GDB debugger: "${eclipse_home}../ARMv8/gdb/bin/aarch64-fsl-gdb"

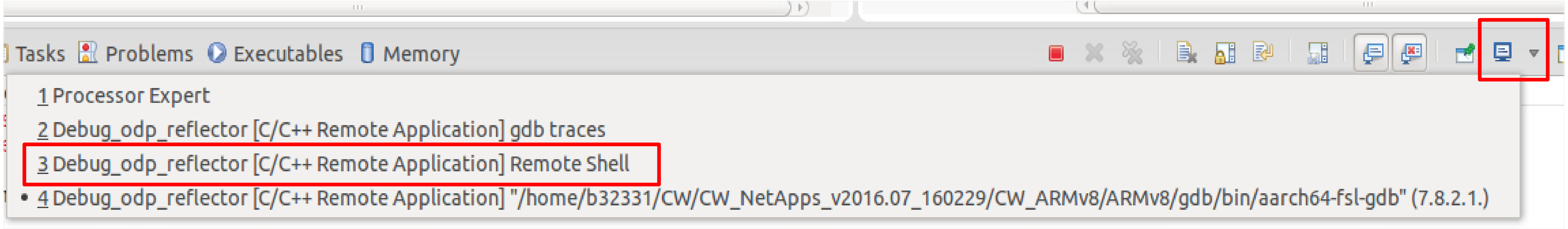
GDB command file: /home/class/SDK/.gdbinit

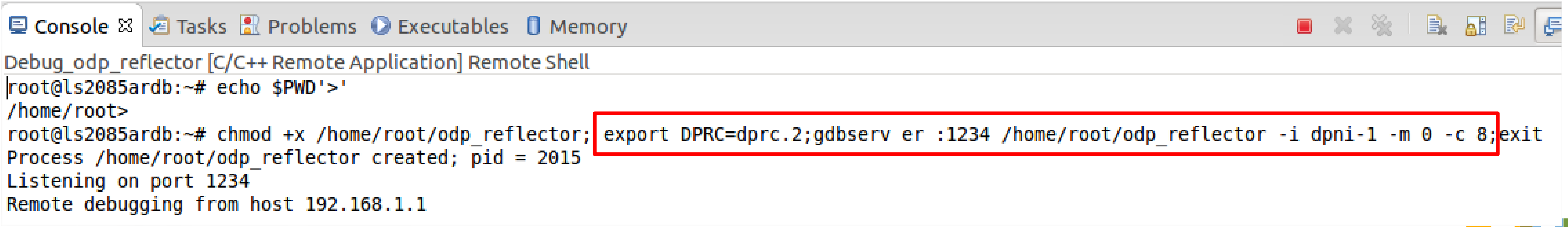
(Warning: Some commands in this file ma fere with the startup operation of the debugger, for example "run".)

/home/class/SDK/.gdbinit

ODP Reflector – Debug Using CodeWarrior

- Prerequisites: configure Linux target for ODP reflector
- Run the reflector hitting Debug button
  (Run > Debug Configurations > C/C++ Remote Application > Debug).
- A pop-up login window will appear – user ID for Linux target is root and there is no password. Just click OK for sending the values
- **ODP Reflector and started**


Tasks
Problems
Executables
Memory
1 Processor Expert
2 Debug_odp_reflector [C/C++ Remote Application] gdb traces
3 Debug_odp_reflector [C/C++ Remote Application] Remote Shell
• 4 Debug_odp_reflector [C/C++ Remote Application] "/home/b32331/CW/CW_NetApps_v2016.07_160229/CW_ARMv8/ARMv8/gdb/bin/aarch64-fsl-gdb" (7.8.2.1.)

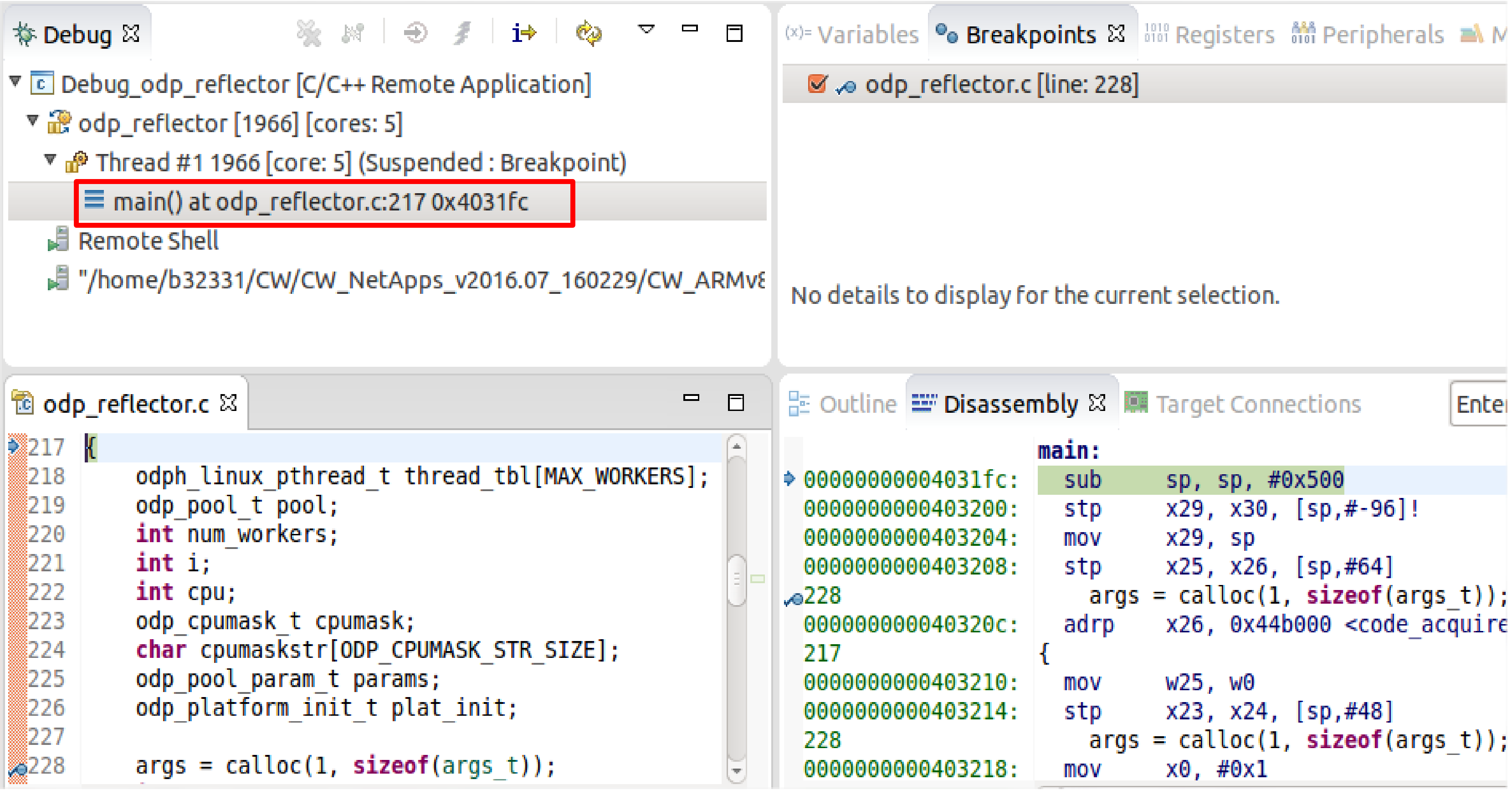


Debug_odp_reflector [C/C++ Remote Application] Remote Shell

```
root@ls2085ardb:~# echo $PWD'>'
/home/root>
root@ls2085ardb:~# chmod +x /home/root/odp_reflector; export DPRC=dprc.2;gdbserv er :1234 /home/root/odp_reflector -i dpni-1 -m 0 -c 8;exit
Process /home/root/odp_reflector created; pid = 2015
Listening on port 1234
Remote debugging from host 192.168.1.1
```

ODP Reflector – Debug Using CodeWarrior

- Debug ODP reflector from main



```
Debug_odp_reflector [C/C++ Remote Application]
  odp_reflector [1966] [cores: 5]
    Thread #1 1966 [core: 5] (Suspended : Breakpoint)
      main() at odp_reflector.c:217 0x4031fc
  Remote Shell
  "/home/b32331/CW/CW_NetApps_v2016.07_160229/CW_ARMv8
```

```
odp_reflector.c [line: 228]

No details to display for the current selection.
```

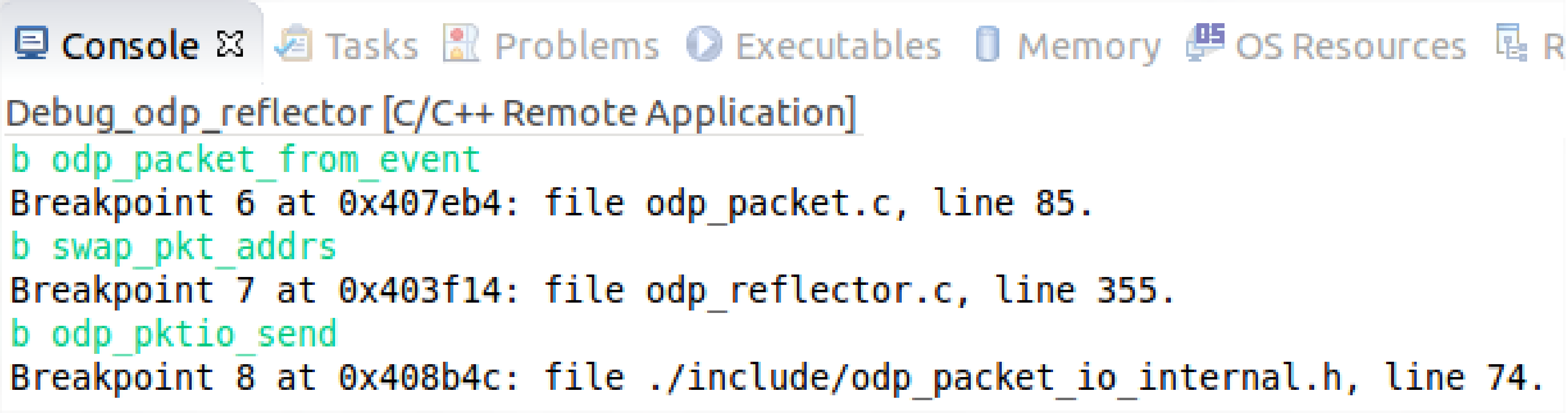
```
217  {
218      odph_linux_pthread_t thread_tbl[MAX_WORKERS];
219      odp_pool_t pool;
220      int num_workers;
221      int i;
222      int cpu;
223      odp_cpumask_t cpumask;
224      char cpumaskstr[ODP_CPUMASK_STR_SIZE];
225      odp_pool_param_t params;
226      odp_platform_init_t plat_init;
227
228      args = calloc(1, sizeof(args_t));
```

```
                  main:
00000000004031fc:   sub      sp, sp, #0x500
0000000000403200:   stp      x29, x30, [sp,#-96]!
0000000000403204:   mov      x29, sp
0000000000403208:   stp      x25, x26, [sp,#64]
228                   args = calloc(1, sizeof(args_t));
000000000040320c:   adrp     x26, 0x44b000 <code_acquire
217               {
0000000000403210:   mov      w25, w0
0000000000403214:   stp      x23, x24, [sp,#48]
228                   args = calloc(1, sizeof(args_t));
0000000000403218:   mov      x0, #0x1
```

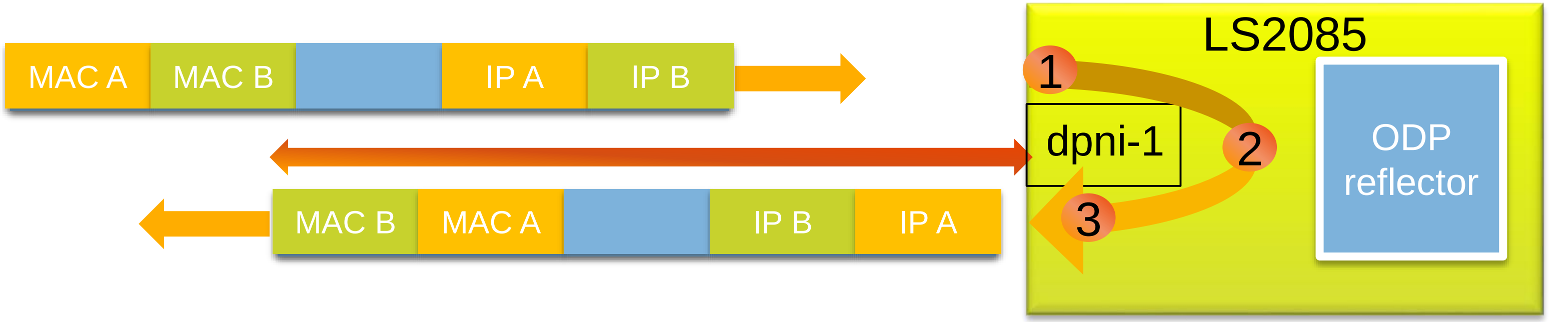
ODP Reflector – Debug Some Relevant Points

- Set some breakpoints in some key points of ODP Reflector application

1 - odp_packet_from_event
2 - swap_pkt_addrs
3 - odp_pktio_send

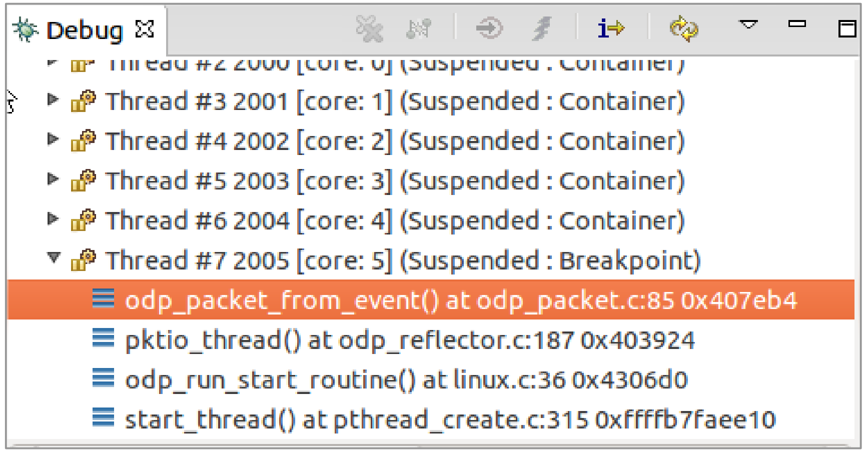


```
Debug_odp_reflector [C/C++ Remote Application]
b odp_packet_from_event
Breakpoint 6 at 0x407eb4: file odp_packet.c, line 85.
b swap_pkt_addrs
Breakpoint 7 at 0x403f14: file odp_reflector.c, line 355.
b odp_pktio_send
Breakpoint 8 at 0x408b4c: file ./include/odp_packet_io_internal.h, line 74.
```

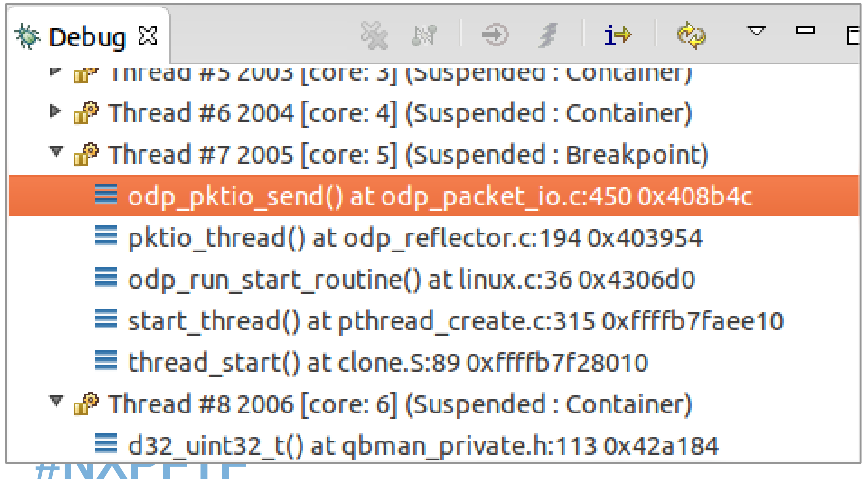


ODP Reflector – Debug Some Relevant Points

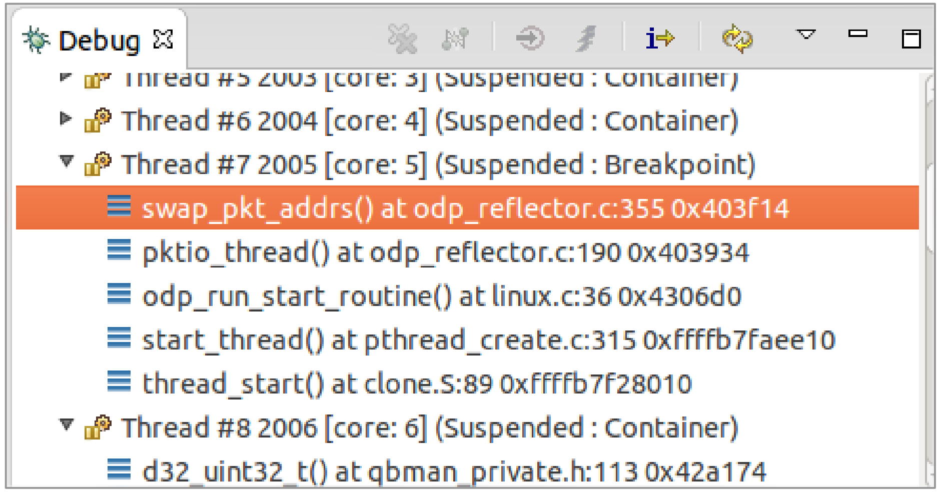
- Resume the application
- Generate traffic using “ping 6.6.6.10 -c 1”
- breakpoints hit 2 times each



```
Thread #2 2000 [core: 0] (Suspended : Container)
Thread #3 2001 [core: 1] (Suspended : Container)
Thread #4 2002 [core: 2] (Suspended : Container)
Thread #5 2003 [core: 3] (Suspended : Container)
Thread #6 2004 [core: 4] (Suspended : Container)
Thread #7 2005 [core: 5] (Suspended : Breakpoint)
    odp_packet_from_event() at odp_packet.c:85 0x407eb4
    pktio_thread() at odp_reflector.c:187 0x403924
    odp_run_start_routine() at linux.c:36 0x4306d0
    start_thread() at pthread_create.c:315 0xffffb7faee10
```



```
Thread #5 2003 [core: 3] (Suspended : Container)
Thread #6 2004 [core: 4] (Suspended : Container)
Thread #7 2005 [core: 5] (Suspended : Breakpoint)
    odp_pktio_send() at odp_packet_io.c:450 0x408b4c
    pktio_thread() at odp_reflector.c:194 0x403954
    odp_run_start_routine() at linux.c:36 0x4306d0
    start_thread() at pthread_create.c:315 0xffffb7faee10
    thread_start() at clone.S:89 0xffffb7f28010
Thread #8 2006 [core: 6] (Suspended : Container)
    d32_uint32_t() at qbman_private.h:113 0x42a184
```

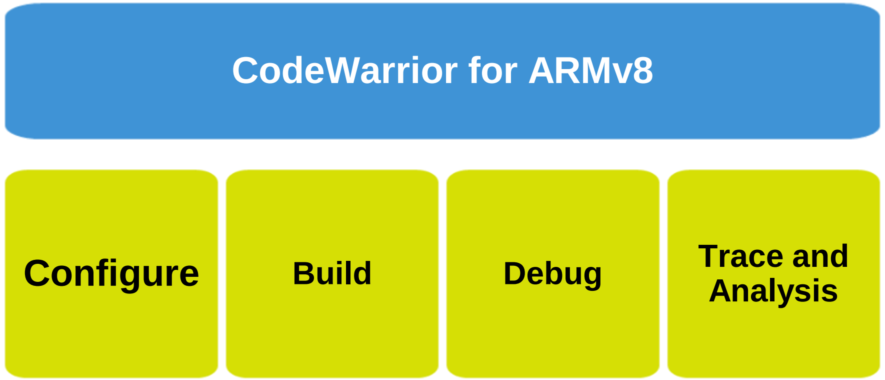


```
Thread #5 2003 [core: 3] (Suspended : Container)
Thread #6 2004 [core: 4] (Suspended : Container)
Thread #7 2005 [core: 5] (Suspended : Breakpoint)
    swap_pkt_addrs() at odp_reflector.c:355 0x403f14
    pktio_thread() at odp_reflector.c:190 0x403934
    odp_run_start_routine() at linux.c:36 0x4306d0
    start_thread() at pthread_create.c:315 0xffffb7faee10
    thread_start() at clone.S:89 0xffffb7f28010
Thread #8 2006 [core: 6] (Suspended : Container)
    d32_uint32_t() at qbman_private.h:113 0x42a174
```

SECURE CONNECTIONS
FOR A SMARTER WORLD

Summary

- This course has been a brief introduction into the QorIQ LS2085-RDB board and the CodeWarrior tools available to debug Linux application
- Linux application debug
- Configure and use Linux QorIQ networking resources
- Debug a demo Linux networking application
- 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