MultiZone® Security

User Guide

i.MX8MQ-MEK

Version	Date	Changes
0.8.0	April 25, 2020	Initial Draft
0.8.1	April 29, 2020	Restructure outline and install flow
0.9.0	May 4, 2020	Add script and completed debug
0.9.1	May 4, 2020	Minor edits to “Security and Performance Assessment”
0.9.2	May 5, 2020	Add reference to the Segger J-Link Mini probe
1.0.0	May 6, 2020	Add MultiZone API and Toolchain command line sections
1.0.1	November 29, 2021	Update MultiZone patent info

Copyright Notice Copyright © 2020, Hex Five Security, Inc. All rights reserved. Information in this document is provided as is, with all faults. Hex Five Security expressly disclaims all warranties, representations and conditions of any kind, whether express or implied, including, but not limited to, the implied warranties or conditions of merchantability, fitness for a particular purpose and non-infringement. Hex Five Security does not assume any liability rising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation indirect, incidental, special, exemplary, or consequential damages.

Please remember that export/import and/or use of strong cryptography software, providing cryptography hooks, or even just communicating technical details about cryptography software is illegal in some parts of the world. When you import this software to your country, re-distribute it from there or even just email technical suggestions, or even source patches to the authors or other people, you are strongly advised to pay close attention to any laws or regulations which apply to you. Hex Five Security, Inc. and the authors of the software included in this SDK are not liable for any violations you make here. So be careful, it is your responsibility.

Hex Five Security reserves the right to make changes without further notice to any products herein.

Arm®, Cortex®-M, and TrustZone® are registered trademarks of Arm Limited
MultiZone® and HEX-Five® are registered trademarks of Hex Five Security, Inc.
MultiZone® Security is protected by patents US 11,151,262 and PCT/US2019/038774

Contents

Hardware and Software Requirements

NXP® i.MX8QM-MEK

This version of MultiZone is designed for the NXP i.MX 8 Multisensory Enablement Kit (MEK). The platform is endowed with a i.MX 8QuadMax application processor which features an heterogeneous set of different Arm processors and microcontrollers: Cortex-A72, Cortex-A53, and Cortex-M4F.

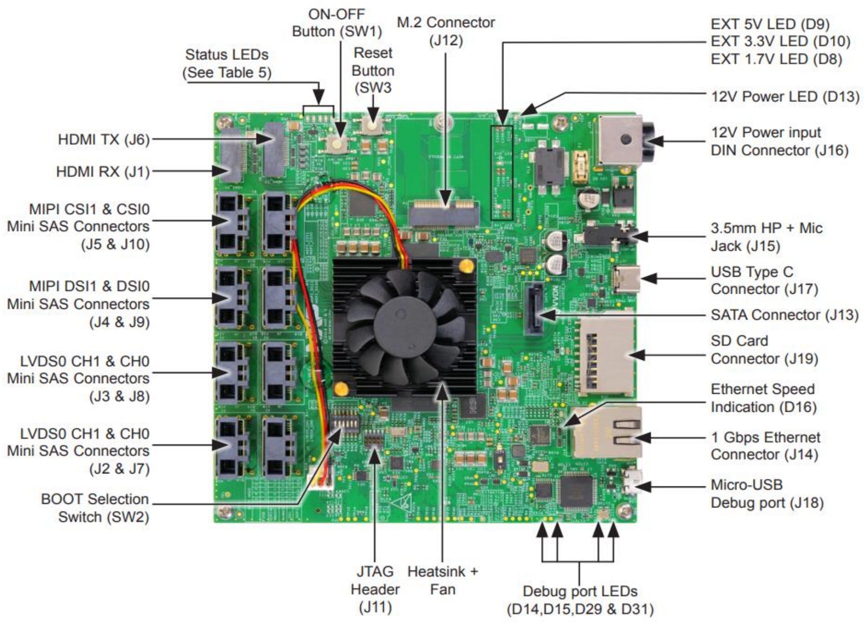

ON-OFF Button (SW1)
M.2 Connector (J12)
EXT 5V LED (D9)
EXT 3.3V LED (D10)
EXT 1.7V LED (D8)
Status LEDs (See Table 5)
Reset Button (SW3
12V Power LED (D13)
HDMI TX (J6)
HDMI RX (J1)
12V Power input DIN Connector (J16)
MIPI CSI1 & CSI0 Mini SAS Connectors (J5 & J10)
3.5mm HP + Mic Jack (J15)
USB Type C Connector (J17)
MIPI DSI1 & DSI0 Mini SAS Connectors (J4 & J9)
SATA Connector (J13)
SD Card Connector (J19)
LVDS0 CH1 & CH0 Mini SAS Connectors (J3 & J8)
Ethernet Speed Indication (D16)
LVDS0 CH1 & CH0 Mini SAS Connectors (J2 & J7)
1 Gbps Ethernet Connector (J14)
Micro-USB Debug port (J18)
BOOT Selection Switch (SW2)
JTAG Header (J11)
Heatsink + Fan
Debug port LEDs (D14,D15,D29 & D31)

Figure 1. Main interface features of the NXP i.MX8QM-MEK board [1].

Figure 1 depicts the top-view of the i-MX8QM-MEK evaluation platform. The i.MX 8 processor family is built with a high-level integration to support graphics, video, image processing, audio, and voice functions, and is ideal for safety-certifiable and efficient performance requirements.

The boot mode configuration can be selected through the "BOOT Selection Switch (SW2), located in the bottom left corner of the board (consult Figure X). To use the standard SD card boot option, one must switch SW2 to OFF, OFF, ON, ON, OFF, OFF (from 1-6 bit) (Figure 2).

POS-6	POS-5	POS-4	POS-3	POS-2	POS-1	BOOT DEVICE
0	0	0	0	0	0	**BOOT From Fuse**
0	0	0	1	0	0	Serial Download
0	0	1	0	0	0	EMMC0
0	0	1	1	0	0	SD1
0	1	1	0	0	0	Octal SPI

Figure 2. i.MX8QM-MEK SW2 boot switch configuration [1].

Figure 3. lists all the push buttons and switches available on the board and their respective functions.

ITEM	DESCRIPTION
SW1	MEK ON/OFF button • Press and hold f or 0.5 sec to turn ON, press and hold f or 5 sec to turn OFF.
SW2	MEK BOOT selection switch • Used f or boot conf iguration according to SCU boot mode.
SW3	MEK RESET button • Pressing of the button will reset the sy stem and begin a boot sequence

Figure 3. i.MX8QM-MEK button functions [1].

J-Link Mini Debug Probe

This hardware is optional and required only to debug the trusted applications running in zones. It is not required to build, deploy, or run any part of the SDK. The Segger J-Link Mini is a popular low-cost option for JTAG/SWD programming and debugging. The Mini version offers the same functionality as the Base version for Cortex-M processors in a practical reduced form factor. The probe comes with the necessary connectors and can be easily sourced online - see [7].

MultiZone® Security Trusted Execution Environment

MultiZone® is a quick and safe way to add security and separation to Arm® Cortex-M® devices that lack hardware isolation mechanisms like TrustZone® or that need more than one secure world.

Cortex-M processors are widely used in general purpose microcontrollers and are often embedded in System on Chip (SoC) devices that collectively ship in billions of units annually. Securing these devices has become increasingly difficult as complex new requirements are often met with the addition of readily available third-party software. Legacy designs lack the physical resources necessary to provide separation of trusted and untrusted functionality, thus leading to larger attack surface and increased likelihood of vulnerability. In response, Hex Five has created a software only solution in MultiZone providing security and separation without the need to redesign existing hardware and software, and eliminating the complexity associated with managing a hybrid hardware/software security scheme.

MultiZone provides hardware-enforced software-defined separation of multiple functional areas within the same chip. MultiZone is completely self-contained, exposes an extremely small attack surface, and it is policy-driven, meaning that no coding or security expertise are required. With MultiZone Security open source software, third party binaries, and legacy code can be configured in minutes to achieve unprecedented levels of safety and security.

Overview of the MultiZone SDK folder structure

The MultiZone SDK is organized as shown in Figure 4. The trusted applications are called 'zones' and each respective source code tree included in folders named **`zone1`** to **`zone4`**. The Board Support Package folder named bsp contains platform-dependent shared code including drivers, memory layout, newlib implementation, and the MultiZone policies defined in the configuration file **`multizone.cfg`**. The **`multizone.jar`** toolchain extension merges the MultiZone runtime, the policies defined in the **`multizone.cfg`** file, and the zones binaries to output a single signed image named **`multizone.bin`** according to the directives in the **`Makefile`** script.

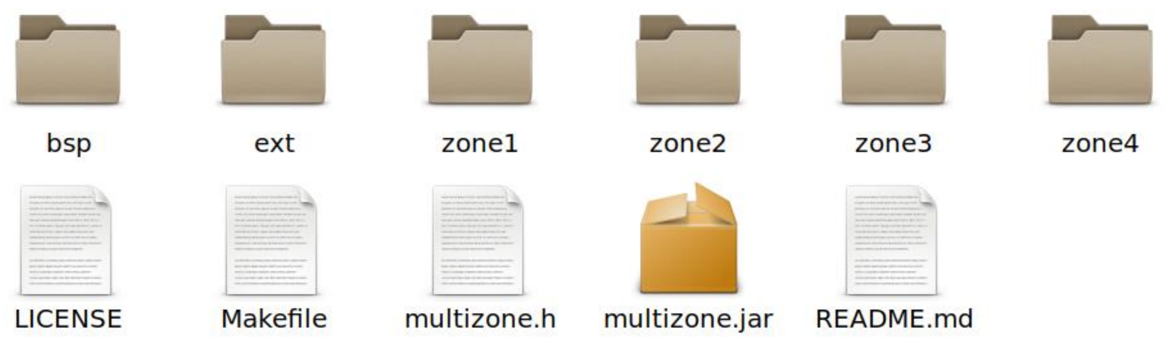


Figure 4. MultiZone SDK folder structure for i.MX8.

The folder **`ext`** contains MultiZone external dependencies: the NXP firmware, the image making tool, the openOCD programmer/debugger, the QNX image, and the MultiZone test application for QNX. Third party dependencies are not included in the Hex Five repository and need to be downloaded and installed by the user according to the instructions included in this guide.

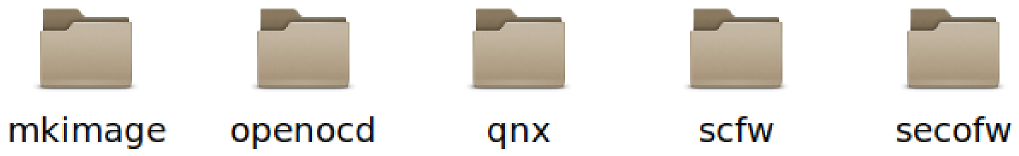


Figure 5. MultiZone external dependencies.

BlackBerry QNX 7.0

The QNX® Software Development Platform (SDP) 7.0 provides a complete and comprehensive development environment for QNX Neutrino-based devices and systems. SDP 7.0 incorporates a multi-level security model that delivers BlackBerry's best-in-class security technologies, which help guard systems against malfunctions, malware, and cyber security breaches. Building on existing certifications including ISO 26262, IEC 61508 and IEC 62304, QNX SDP 7.0 brings a proven safety pedigree.

To complement the SDP toolchain, QNX also provides QNX Software Center. This software mechanism is used to manage discovery, delivery and dependencies of QNX development products in a centralized fashion. The tool is able to alert developers with new software updates, patches and new product releases. Follow the steps here to set up your QNX environment and test the HexFive MultiZone Security test application.

System Setup

This chapter describes how to build the QNX and MultiZone SDK for i.MX8QM from scratch. All steps involved in this User Guide require a Linux development host running Ubuntu 18.04.4 LTS. The target hardware is an i.MX8QM-MEK development board. The remaining sections describe how to install the required tools to build and debug the software and to create a bootable image for the target.

Linux Development Environment

Open a terminal and type the following commands to install the required packages:

```
$ sudo apt update
$ sudo apt install build-essential default-jre git gtkterm
```

Note: *Git is required only to download the imx-mkimg tool from the codeaurora.org website. GtkTerm is entirely optional and required only to connect to the reference application via UART. It is not required to build, debug, and load any software components.*

Arm GNU 8 Embedded Toolchain

Download and install the GNU Arm Embedded Toolchain for Cortex-M:

```
$ cd ~
$ wget http://developer.arm.com/-/media/Files/downloads/gnu-rm/8-
2019q3/RC1.1/gcc-arm-none-eabi-8-2019-q3-update-linux.tar.bz2
$ tar -xvf gcc-arm-none-eabi-8-2019-q3-update-linux.tar.bz2
$ export GNU_ARM=~/gcc-arm-none-eabi-8-2019-q3-update/bin
$ export PATH=$PATH:~/gcc-arm-none-eabi-8-2019-q3-update/bin
```

MultiZone SDK

Login to GitHub.com using an Hex Five authorized user id and download the MultiZone SDK zip at https://github.com/hex-five/multizone-sdk-imx8/archive/master.zip

Note: this is a private repository. Request access via email at info@hex-five.com

Extract the zip file, rename the folder, and take note of the folder location for later use:

```
$ cd ~
$ unzip multizone-sdk-imx8-master.zip
$ mv multizone-sdk-imx8-master multizone-sdk-imx8
```

NXP Image making tool

Download the public repository and build the executable mkimage_imx8:

```
$ cd ~/multizone-sdk-imx8/ext/mkimage
$ git clone --single-branch --branch imx_4.19.35_1.1.0
https://source.codeaurora.org/external/imx/imx-mkimage
$ cd imx-mkimage
$ make
```

NXP Security Controller Firmware

Download and extract the SECOFW binary:

```
$ cd ~/multizone-sdk-imx8/ext/secofw
$ wget http://www.nxp.com/lgfiles/NMG/MAD/YOCTO/firmware-imx-8.1.bin
$ chmod u+x firmware-imx-8.1.bin
$ ./firmware-imx-8.1.bin
```

NXP System Controller Firmware

Download from a browser (NXP login required) and copy to ~/multizone-sdk-imx8/ext/scfw:

```
http://www.nxp.com/webapp/Download?colCode=L4.19.35_1.1.0_SCFKIT-
1.2.7.1&appType=license
```

Extract the SCFW porting kit:

```
$ cd ~/multizone-sdk-imx8/ext/scfw
$ tar -xvzf imx-scfw-porting-kit-1.2.7.1.tar.gz
$ cd packages
$ chmod u+x imx-scfw-porting-kit-1.2.7.1.bin
$ ./imx-scfw-porting-kit-1.2.7.1.bin
$ cd imx-scfw-porting-kit-1.2.7.1/src
$ tar -xvzf scfw_export_mx8qm_b0.tar.gz
```

Patch and build the MultiZone modified version of the NXP firmware (Gnu Arm Toolchain required):

```
$ export GNU_ARM=~/gcc-arm-none-eabi-8-2019-q3-update/bin
$ export PATH=$PATH:~/gcc-arm-none-eabi-8-2019-q3-update/bin
$ cd ~/multizone-sdk-imx8/ext/scfw
$ patch packages/imx-scfw-porting-kit-
1.2.7.1/src/scfw_export_mx8qm_b0/platform/board/mx8qm_mek/board.c <
board.patch
$ cd packages/imx-scfw-porting-kit-1.2.7.1/src/scfw_export_mx8qm_b0
$ make qm R=b0 B=mek CROSS_COMPILE=arm-none-eabi-
```

OpenOCD On-Chip Debugger

The OpenOCD is the on-chip debugger (version X) used by MultiZone to debug and load binaries. A prebuilt version of the tool is located in the **`ext/openocd`** folder.

Note: openOCD is optional and required only for debugging the TEE applications.

Expand the archive and add environment variables:

```
$ cd ~/multizone-sdk-imx8/ext/openocd
$ tar -xvzf imx8-openocd-20200504.tar.xz
$ export OPENOCD=~/multizone-sdk-imx8/ext/openocd/openocd-imx8qm/bin
$ export PATH=$PATH:~/multizone-sdk-imx8/ext/openocd/openocd-
imx8qm/bin
```

Eclipse IDE 2019-12

The Eclipse IDE will be used to debug step-by-step MultiZone on the i-MX8QM-MEK platform. After installing the standard package, the “GNU ARM & RISC-V C/C++ Cross Development Tools” plugin needs to be installed, which will allow to create a debug configuration which uses MultiZone SDK versions of OpenOCD and GNU Arm toolchain (arm-none-eabi-gdb debugger).

Note: *The Eclipse IDE is optional and required only for debugging the TEE applications.*

1) Download to your home directory the Eclipse package by the following the below link:

```
https://www.eclipse.org/downloads/download.php?file=/oomph/epp/
```

```
2019-12/R/eclipse-inst-linux64.tar.gz
```

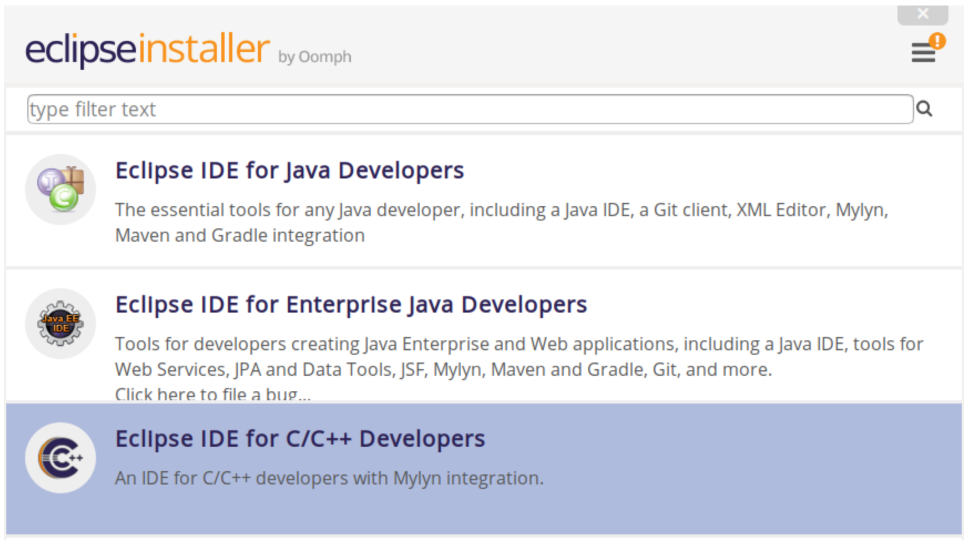
2) Extract the package:

```
$ cd ~
$ tar -xvf eclipse-inst-linux64.tar.gz
```

3) Navigate to the extracted eclipse folder and run the installer executable:

```
$ cd ~/eclipse-installer/
$ ./eclipse-inst
```

4) The Eclipse installer will prompt a new window to choose the product to be installed (Figure X). Select the "Eclipse IDE for C/C++ Developers" and press Next.



Eclipse IDE for Java Developers
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration

Eclipse IDE for Enterprise Java Developers
Tools for developers creating Java Enterprise and Web applications, including a Java IDE, tools for Web Services, JPA and Data Tools, JSF, Mylyn, Maven and Gradle, Git, and more.
Click here to file a bug...

Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers with Mylyn integration.

Figure 6. Eclipse installer window.

5) Press **Install** button on the next window and accept the license agreement. This process can take a few minutes to complete.

6) After the installation, hit the **Launch** button to launch the Eclipse IDE. Select a directory as a workspace and press **Launch**.

7) In the main window of Eclipse go to Help and press "Install New Software..." option.

8) The windows in Figure X will be open. Type in "Work with" bar on the following website and select the package "GNU ARM & RISC-V C/C++ Cross Development Tools" to be installed. Hit **Next**, **Next**, and finally accept the license and begin to start the installation by clicking the **Finish** button.

```
http://gnu-mcu-eclipse.netlify.com/v4-neon-updates
```

Note: if you get prompted with a security warning regarding unsigned content, proceed to "Install anyway".

9) Hit the **`Restart Now`** button

Bootable media

Follow these steps to initialize a new SD card:

Connect the SD card, or an SD card reader, to your computer and take note of the device name:

```
$ lsblk
```

Important: *make sure you identify the correct device name as the next steps will cause permanent data loss if directed to the wrong device - you are advised.*

Unmount the volume in case the SD card is already in use:

```
$ sudo umount /dev/sdX1
```

Run the fdisk command and substitute sdX with your device name:

```
$ sudo fdisk /dev/sdX
```

Create a new empty DOS partition table with the command 'o':

```
Command (m for help): o
Created a new DOS disklabel with disk identifier 0xXXXXXXXX.
```

Create a new primary partition with the commands 'n' and 'p' (accept default values):

```
Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
```

Change partition type to FAT32 and mark it active with the commands ‘t’ ‘c’ ‘a’:

```
Command (m for help): t
Selected partition 1
Hex code (type L to list all codes): c
Changed type of partition 'Linux' to 'W95 FAT32 (LBA)'.
Command (m for help): a
Selected partition 1
The bootable flag on partition 1 is enabled now.
```

Write the changes to the media with the command ‘w’:

```
Command (m for help): w
The partition table has been altered.
```

Format the SD card:

```
$ sudo mkfs.vfat /dev/sdX1
```

Note: *you may need to reinsert the media to make the changes visible to the Linux kernel.*

BlackBerry QNX 7.0

This section covers installation and configuration of the QNX Environment. The first part describes how to install the QNX Software Development environment on Ubuntu 18.04.4 LTS - a QNX license is required. The second part shows how to configure and build the target QNX image including the MultiZone QNX test application.

Installation

QNX Software Center

Log in to your myQNX account on the QNX website, select the Developers tab at the top of the page, then click the QNX Software Center link. Click the link for Linux hosts and download the QNX Software Center installer, qnx-setup-nnnnnnnnnnnn-lin.run to your home directory.

Run the installer:

```
$ cd ~
$ chmod a+x qnx-setup-nnnnnnnnnnnn-lin.run
$ ./qnx-setup-nnnnnnnnnnnn-lin.run
```

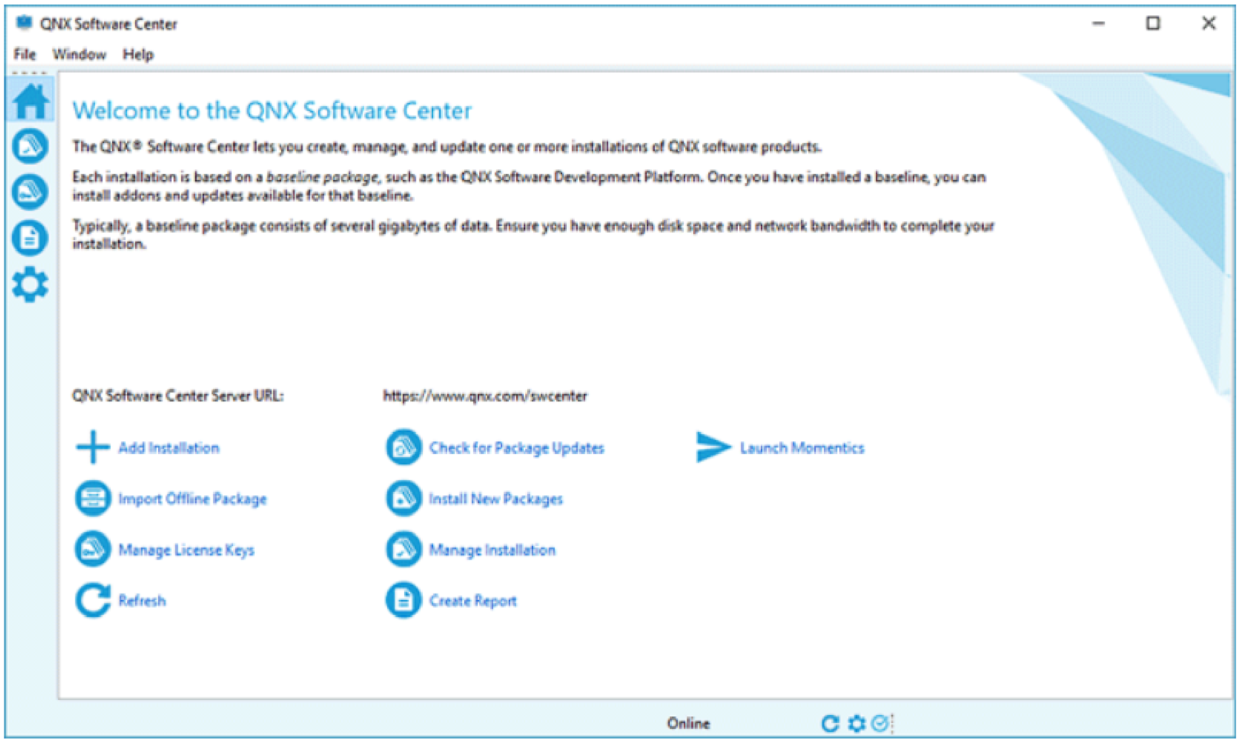
Note: *Make sure you run the installer in user mode - i.e. do not sudo.*

The installer automatically launches the QNX Software Center. If needed, you can later restart the QNX Software Center with the command:

```
$ ~/qnx/qnxsoftwarecenter/qnxsoftwarecenter
```

QNX Software Development Platform SDP 7.0

After launching the QNX Software Center you will be prompted to login in to your myQNX account. Enter your credentials and the following welcome screen will be shown.



Welcome to the QNX Software Center

The QNX® Software Center lets you create, manage, and update one or more installations of QNX software products.

Each installation is based on a *baseline package*, such as the QNX Software Development Platform. Once you have installed a baseline, you can install addons and updates available for that baseline.

Typically, a baseline package consists of several gigabytes of data. Ensure you have enough disk space and network bandwidth to complete your installation.

QNX Software Center Server URL: https://www.qnx.com/swcenter

Figure 7. Welcome screen of QNX Software Center.

Select **Add Installation** from the Welcome screen.

Expand the QNX Software Development Platform 7.0 group and install the QNX SDP 7.0 by clicking in the QNX Software Development Platform 7.0 package.

Click **Next** when you're ready to proceed with the installation.

Select installation folder ~/qnx700 and target architecture aarch64le , then click Next.

The QNX Software Center may take a few minutes to calculate the installation requirements. Make sure that the item you want to install is checked, then click **Next**.

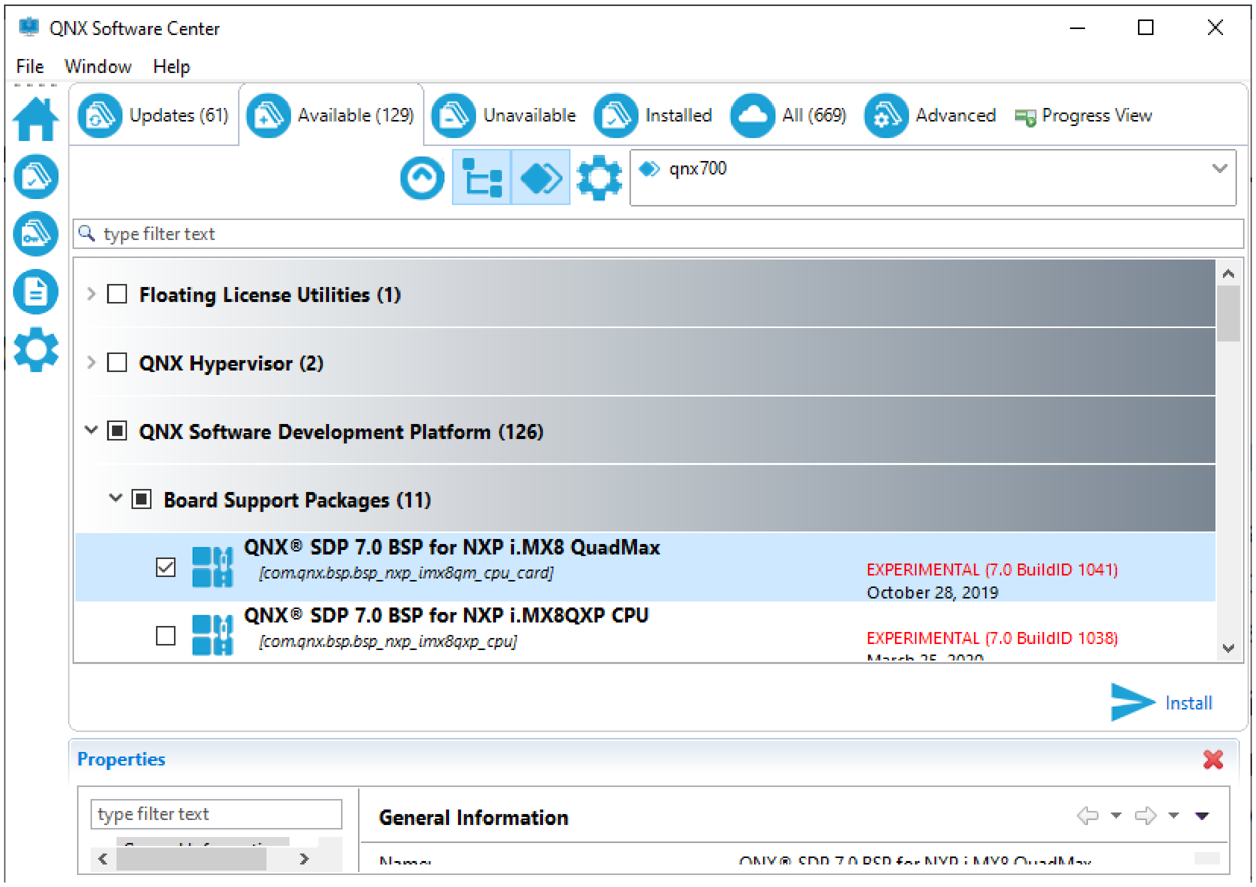
Review the list of packages that the QNX Software Center will install, then click **Next**.

Choose a license for the product being installed, then click **Finish** to start the installation process.

QNX i.MX8QM-MEK Board Support Package

Click the "Available" tab and select these 2 packages:

- ❏ "Board Support Packages" > "QNX SDP 7.0 BSP for NXP i.MX8 QuadMax"
- ❏ "BSP & Drivers" > "QNX® SDP 7.0 SMMUMAN NXP i.MX8 SoC Support"


QNX Software Center

File Window Help

Updates (61) Available (129) Unavailable Installed All (669) Advanced Progress View

qnx700

type filter text

- Floating License Utilities (1)
- QNX Hypervisor (2)
- QNX Software Development Platform (126)
  - Board Support Packages (11)
    - [✓] QNX® SDP 7.0 BSP for NXP i.MX8 QuadMax [com.qnx.bsp.bsp_nxp_imx8qm_cpu_card] EXPERIMENTAL (7.0 BuildID 1041) October 28, 2019
    - [ ] QNX® SDP 7.0 BSP for NXP i.MX8QXP CPU [com.qnx.bsp.bsp_nxp_imx8qxp_cpu] EXPERIMENTAL (7.0 BuildID 1038)

Install

Properties

type filter text

General Information

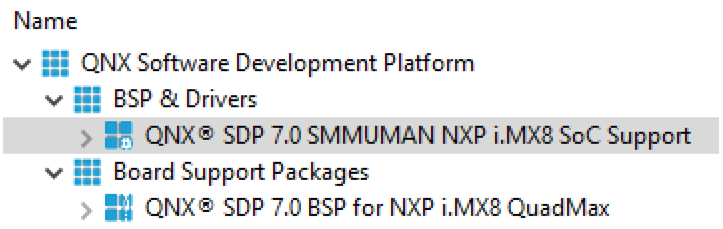

Name
- QNX Software Development Platform
  - BSP & Drivers
    - QNX® SDP 7.0 SMMUMAN NXP i.MX8 SoC Support
  - Board Support Packages
    - QNX® SDP 7.0 BSP for NXP i.MX8 QuadMax

Figure 8. “Available” screen on QNX Software Center to download the 2 BSP packages.

Click **Install** to download the required packages and keep note of the installation directory of the **bsp**.

Navigate to the QNX SDP home directory and set up the QNX environment variables:

```
$ cd ~/qnx700
$ source qnxsdp-env.sh
```

Open the bsp folder, and extract the BSP archive to the working directory **bsp_working_dir** and set an environment variable to point to this folder:

```
$ mkdir bsp_working_dir
$ export BSP_ROOT_DIR=~/qnx700/bsp_working_dir
```

Extract the BSP under the directory created in the previous step:

```
$ unzip bsp/BSP_nxp-imx8qm-xxx.zip -d bsp_working_dir
```

QNX Target Configuration

Setup Ethernet IP address and SSH connectivity

Generate SSH keys:
Note: make sure the QNX environment is sourced

```
$ ssh-keygen -A -f $QNX_TARGET
```

Create an empty directory named var/chroot/sshd in the target folder:

```
$ mkdir -p $QNX_TARGET/var/chroot/sshd
```

Setup permissions for the sshd daemon:

```
$ echo PermitRootLogin yes >> $QNX_TARGET/etc/ssh/sshd_config
$ echo ChrootDirectory / >> $QNX_TARGET/etc/ssh/sshd_config
```

Setup static IP address and sshd initialization in the by adding the following 2 lines to the startup script bsp_working_dir/images/imx8qm-cpu-mek.build just before “Start the main shell”:

```
ifconfig fec0 192.168.10.1
[+session] ksh -c /usr/sbin/sshd
```

Note: you may want to replace 192.168.10.1 with an IP address suitable to your network

Add the ssh daemon executable to QNX file system by adding the **`/usr/sbin/sshd=sshd`** entry to the bsp_working_dir/images/imx8qm-cpu-mek.build file, right after inetd and telnetd:

```
###########################################################################
## uncomment for network services (telnet) support
```

```
###########################################################################
/usr/sbin/inetd=inetd
/usr/sbin/telnetd=telnetd
/usr/sbin/sshd=sshd
```

Add the ssh configuration to the file bsp_working_dir/images/imx8qm-cpu-mek.build:

```
############################################################################
## SSH
############################################################################
[uid=0 gid=0 perms=0600] /etc/ssh=${QNX_TARGET}/etc/ssh
[uid=0 gid=0 perms=0700] /var/chroot/sshd=${QNX_TARGET}/var/chroot/sshd
[uid=0 gid=0 perms=0644] /etc/profile = {
    PATH=/proc/boot:/bin:/usr/bin:/sbin:/usr/sbin
}
```

Note: copy these lines exactly as indicated above - i.e. with new lines.

The above configuration will enable ssh access to the target with the command:

```
$ ssh root@192.168.10.1
$ password: root
```

Disable the QNX watchdog

Edit the **bsp_working_dir/images/imx8qm-cpu-mek.build** file to disable the watchdog. Edit line #12 and remove the '-W' parameter:

```
startup-imx8x-qm-cpu-mek -s
```

Configure Boot Loader

Edit the **bsp_working_dir/src/hardware/ipl/boards/imx8qm-cpu/board.h** file and change line #68 into:

```
#define IMX_QNX_IMAGE_LOAD_FROM          'M'
```

Note: don't forget the single quotes.

MultiZone Test Application

The multizone-qnx-test application demonstrates the enforcement of the MultiZone policies to the QNX cluster. This utility provides 4 interactive commands (load, store, dump, erase) to read and write any physical memory location in the system. For a detailed explanation of the syntax see section xxx.

Build the test application:

```
$ cd ~/multizone-sdk-imx8/ext/qnx
$ source ~/qnx700/qnxsdp-env.sh
$ make
```

Include the test application in the QNX file system image by adding this one line to the **`bsp_working_dir/images/imx8qm-cpu-mek.build`** file:

```
[uid=0 gid=0] /multizone-qnx-test = /home/<user>/multizone-sdk-
imx8/ext/qnx/multizone-qnx-test
```

QNX Build

The QNX image consists of two files: ipl-imx8qm-cpu-mek.bin (IPL) and qnx-ifs-cpu-mek (IFS). The IPL is required to build the MultiZone TEE and therefore copied to the MutiZone SDK folder.

Build the image

```
$ source ~/qnx700/qnxsdp-env.sh
$ export BSP_ROOT_DIR=~/qnx700/bsp_working_dir
$ cd $BSP_ROOT_DIR && make clean && make
$ cd $BSP_ROOT_DIR/images && make clean
$ cd $BSP_ROOT_DIR && make install
$ cd $BSP_ROOT_DIR/src && make install
$ cd $BSP_ROOT_DIR/images && make
$ cp ipl-imx8qm-cpu-mek.bin ~/multizone-sdk-imx8/ext/qnx
```

Copy the image files to the SD card

```
$ cd $BSP_ROOT_DIR/images
$ sudo dd if=ipl-imx8qm-cpu-mek_b0.imx of=/dev/sdX bs=1k seek=32
$ cp qnx-ifs-cpu-mek /media/<user>/<sdcardid>/qnx-ifs
```

Note: See section xxx for SD card initialization. dd is a privileged command and you may need sudo access. Replace /dev/sdX with the correct device name of your SD card or reader.

Verify the QNX installation

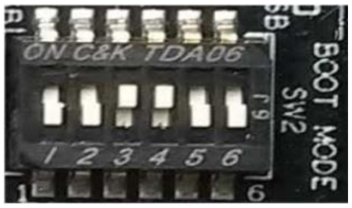


Figure 9. SW2 DIP switch to boot using the SD card on the i.MX8QM MEK.

1) Unmount the SD card from the development computer and insert it into the board slot J19.
2) Verify that SW2 is configured to boot from the SD card - see image above.
3) Connect port J16 to the 12V power adapter.
4) Connect USB port J18 to your computer.
5) Connect Ethernet port J14 to your network.
6) Identify the serial port device on your computer - i.e. /dev/ttyUSB0.
7) Test serial connectivity at 115200/8/N/1 - you may need to press enter a few times.
8) Test Internet connectivity: ping 192.168.10.1.
9) Test ssh connectivity (accept the board fingerprint): ssh root@192.168.10.1 / password ‘root’.
10) Test the MultiZone application: ./multizone-qnx-test

The multizone-qnx-test application allows the issuing of read and write commands to physical memory addresses 64-bit long. Both the address and value must be entered in hex notation:

Load: read data from the memory address `addr`
Syntax: `load addr`
Example: `load 0x100000`

Store: write `value` to memory address `addr`

Syntax: `store addr value`

Example: `store 0x100000 0x100`

Dump: read `length` bytes starting at address `addr`

Syntax: `dump addr length`

Example: `dump 0x100000 0x100`

Erase: writes `length` zeros starting at address `addr`

Syntax: `erase addr length`

Example: `erase 0x100000 0x100`

MultiZone Security Trusted Execution Environment

MultiZone Security SDK defines hardware separation policies by mapping devices, interrupts, and memory regions to zones. This can be accomplished by modifying the MultiZone.cfg file.

```
# Copyright(C) 2020 Hex Five Security, Inc. - All Rights Reserved

# MultiZone reserved memory: 8K @0x1FFE0000, 6K @0x20000000

Tick = 10 # ms

Zone = 1
dev  = M4_0, M4_0_UART
base = 0x1FFE8000; size =  32K; rwx = rx # PROGRAM
base = 0x20004000; size =   4K; rwx = rw # DATA

Zone = 2
dev  = M4_0, M4_0_SEMA42
base = 0x1FFF0000; size =   8K; rwx = rx # PROGRAM
base = 0x20005000; size =   4K; rwx = rw # DATA

Zone = 3
dev  = M4_0, M4_0_RGPIO
base = 0x1FFF2000; size =   8K; rwx = rx # PROGRAM
base = 0x20006000; size =   4K; rwx = rw # DATA

Zone = 4
dev  = M4_0, M4_0_TPM
base = 0x1FFF4000; size =   8K; rwx = rx # PROGRAM
base = 0x20007000; size =   4K; rwx = rw # DATA

Zone = 5
dev  = M4_1, M4_1_RGPIO, M4_1_UART
base = 0x38000000; size =  64M; rwx = rwx # M4_1_MEM

Zone = 6
dev  = A53, A72
base = 0x000000000; size = 448M; rwx = rwx; # OCM
base = 0x060000000; size = 256M; rwx = rwx; # PCIe0
base = 0x070000000; size = 256M; rwx = rwx; # PCIe1
base = 0x080000000; size =   2G; rwx = rwx; # DDR (low)
base = 0x880000000; size =  30G; rwx = rwx; # DDR (high)
```

Listing 1. Example of a MultiZone.cfg file defining mapping policies.

Listing 1 exemplifies a configuration file for the i.MX8QM-MEK board. It defines 4 zones for the Cortex-M4 core 0 and assigns their hardware resources; command `dev` is responsible to define these parameters. The `Tick` option refers to the nanokernel system timer and its period. The base option defines the memory regions for code (#`PROGRAM`) and data (#`DATA`).

Build the MultiZone SDK image

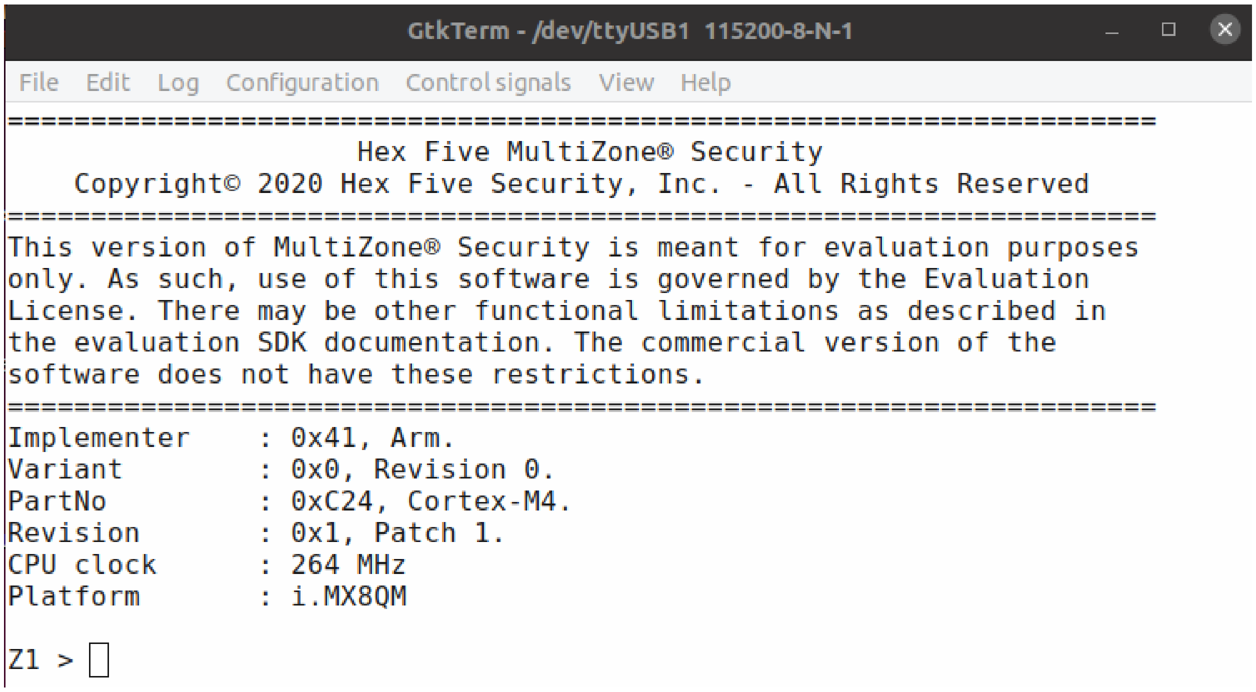
```
$ cd ~/multizone-sdk-imx8
$ export GNU_ARM=~/gcc-arm-none-eabi-8-2019-q3-update/bin
$ make clean && make
```

Copy to SD card:

```
$ sudo dd if=multizone.imx of=/dev/sdX bs=1k seek=32
```

Verify the MultiZone installation

1. Unmount the SD card from the development computer and insert it into the board slot J19.
2. Verify that SW2 is configured to boot from the SD card - see image above.
3. Connect port J16 to the 12V power adapter.
4. Connect the USB port J18 to your computer.
5. Identify the serial port device on your computer - i.e. /dev/ttyUSB1.
6. Test serial connectivity at 115200/8/N/1 - you may need to press enter a few times.
7. Refer to section "Security and Performance Assessment" for detailed test cases.


GtkTerm - /dev/ttyUSB1 115200-8-N-1
File Edit Log Configuration Control signals View Help

```
======================================================================
                     Hex Five MultiZone® Security
     Copyright© 2020 Hex Five Security, Inc. - All Rights Reserved
======================================================================
This version of MultiZone® Security is meant for evaluation purposes
only. As such, use of this software is governed by the Evaluation
License. There may be other functional limitations as described in
the evaluation SDK documentation. The commercial version of the
software does not have these restrictions.
======================================================================
Implementer    : 0x41, Arm.
Variant        : 0x0, Revision 0.
PartNo         : 0xC24, Cortex-M4.
Revision       : 0x1, Patch 1.
CPU clock      : 264 MHz
Platform       : i.MX8QM

Z1 >
```

Figure 10. MultiZone Test Application Console in Zone 1.

Debugging Trusted Applications

This section describes how to debug the trusted applications using OpenOCD, GDB, and Eclipse IDE. First we create a standard i.MX8QM-MEK bootable image with a dummy binary for the Cortex-M4 core0. MultiZone runtime is loaded to the core upon starting the debug session. Then we explain how to set up the debug environment with OpenOCD, GDB, and Eclipse IDE. The instructions in this section refer to the Segger J-Link Mini probe, which is not included in the NXP MEK kit and that needs to be sourced separately - see [7].

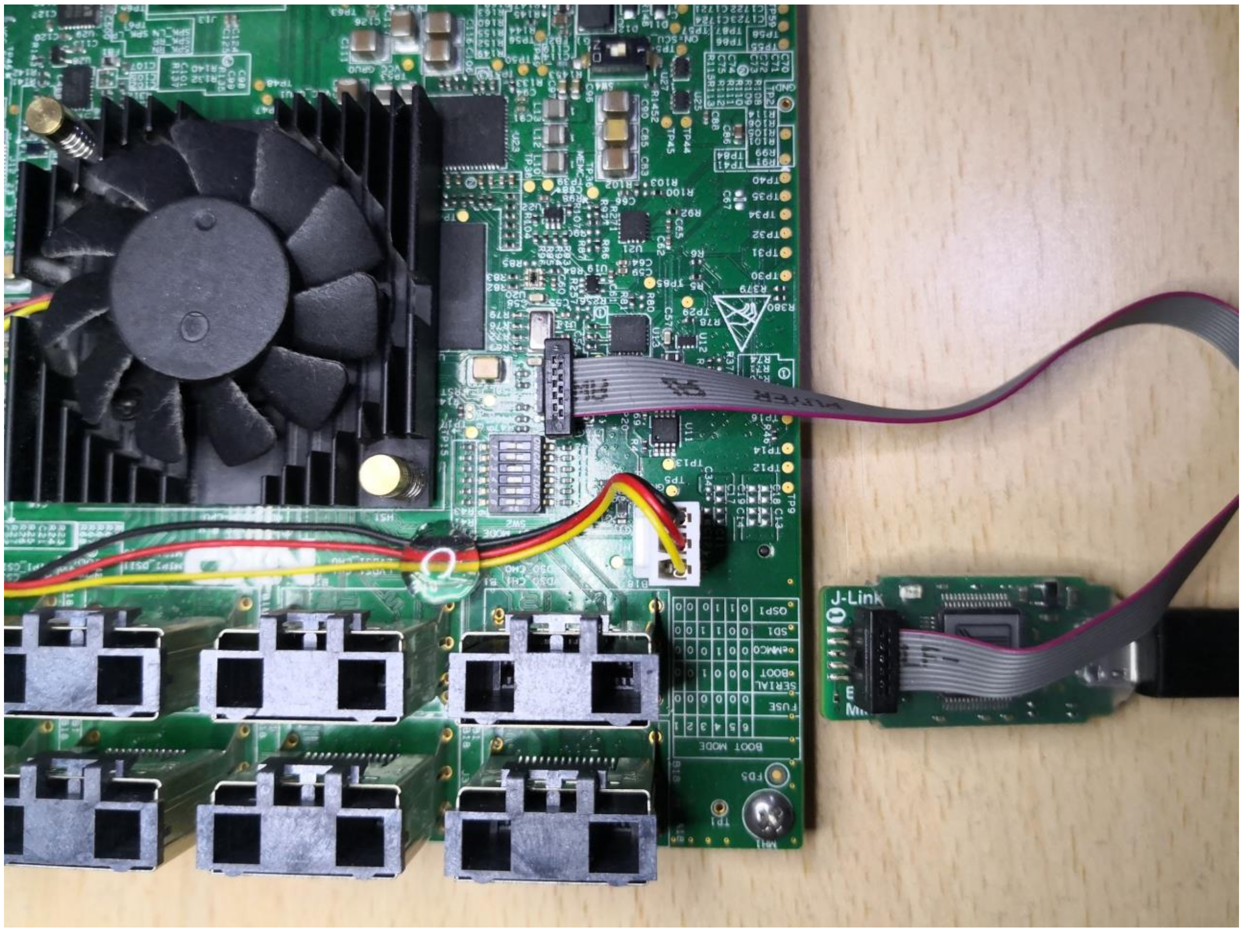
Get a proper i.MX8QM-MEK Bootable Image to enable debugging.

The **`ext/openocd/multizone-sdk-imx8-openocd`** directory under MultiZone SDK has a specific image file to burn on the SD Card (**`debug.bin`**) to enable debugging.

Copy this file to the SD card:

```
$ sudo dd if=./ext/openocd/debug.bin of=/dev/mmcblk0 bs=1k seek=32 skip=0
```

Connect the debug probe to the board, using the JTAG cable, as depicted below:



Boot the target.

Establish a Debug Session by using OpenOCD and GDB

1) Run OpenOCD with the appropriate configuration file in another terminal.

```
$ ./openocd -f ./ext/openocd/openocd-imx8/tcl/interface/jlink.cfg -f
./ext/openocd/openocd-imx8/tcl/target/imx8qm.cfg -c "adapter speed 1000;
reset_config srst_only; targets"
```

2) Check the OpenOCD output:

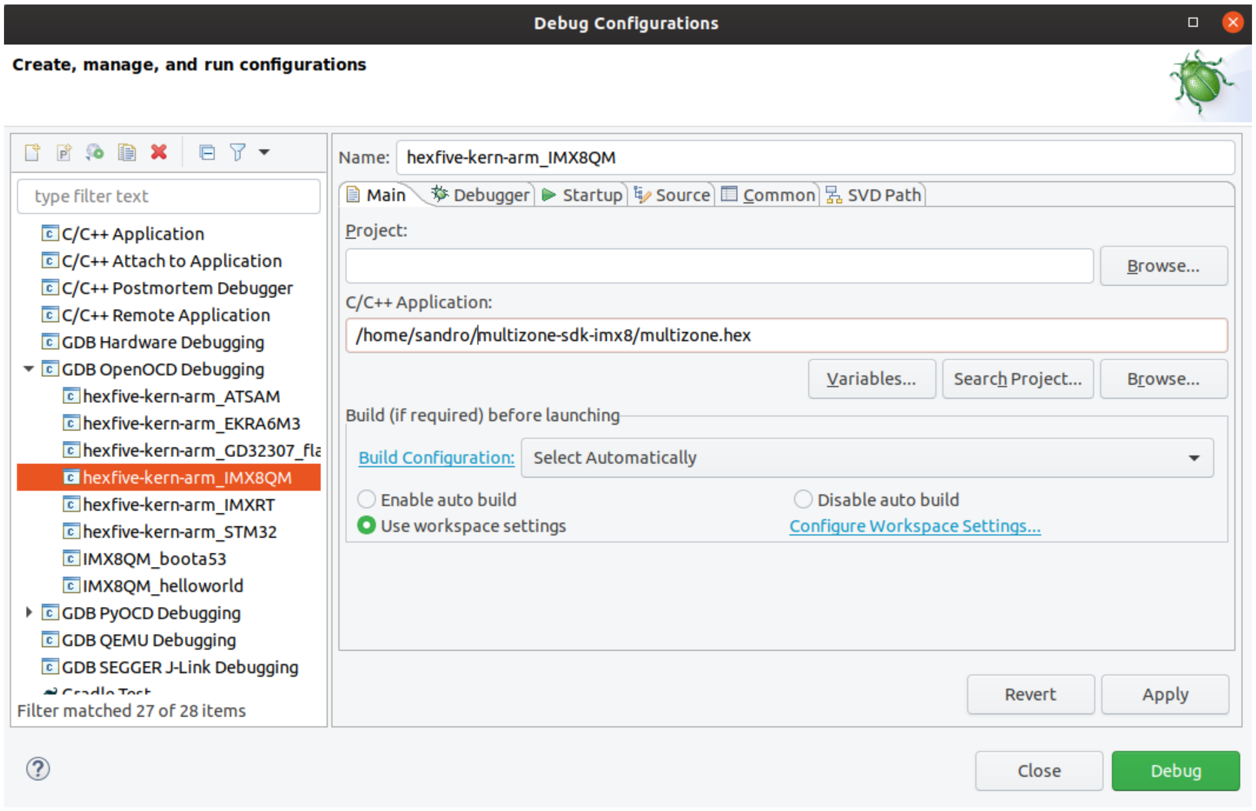
```
$ ./openocd -f ./ext/openocd/multizone-sdk-imx8-openocd/tcl/interface/jlink.cfg -
f ./ext/openocd/multizone-sdk-imx8-openocd/tcl/target/imx8qm.cfg -c "adapter
speed 1000; reset_config srst_only; targets"
Open On-Chip Debugger 0.10.0+dev-01055-gd3669692 (2020-05-03-19:50)
Licensed under GNU GPL v2
For bug reports, read
       http://openocd.org/doc/doxygen/bugs.html
Info : auto-selecting first available session transport "jtag". To override use
'transport select <transport>'.
Info : Hardware thread awareness created
    TargetName         Type       Endian TapName            State
--  ------------------ ---------- ------ ------------------ ------------
 0  imx8qm.a53.0       aarch64    little imx8qm.cpu         unknown
 1  imx8qm.a53.1       aarch64    little imx8qm.cpu         unknown
 2  imx8qm.a53.2       aarch64    little imx8qm.cpu         unknown
 3  imx8qm.a53.3       aarch64    little imx8qm.cpu         unknown
 4  imx8qm.a72.0       aarch64    little imx8qm.cpu         unknown
 5  imx8qm.a72.1       aarch64    little imx8qm.cpu         unknown
 6  imx8qm.scu         cortex_m   little imx8qm.cpu         unknown
 7* imx8qm.m4_0        cortex_m   little imx8qm.cpu         unknown
 8  imx8qm.m4_1        cortex_m   little imx8qm.cpu         unknown

Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : J-Link EDU Mini V1 compiled Jan  7 2020 16:53:19
Info : Hardware version: 1.00
Info : VTarget = 1.702 V
Info : clock speed 1000 kHz
Info : JTAG tap: imx8qm.cpu tap/device found: 0x1890101d (mfg: 0x00e (Freescale
(Motorola)), part: 0x8901, ver: 0x1)
Error: Timeout during WAIT recovery
Info : DAP transaction stalled (WAIT) - slowing down
Error: Timeout during WAIT recovery
Error: imx8qm.a53.0: examination failed

Info : Listening on port 5551 for gdb connections
Info : Listening on port 3333 for gdb connections
Info : Listening on port 3334 for gdb connections
Info : Listening on port 3335 for gdb connections
```

3) Open Eclipse and go to “Run”->”Debug Configurations”

4) Double click “GDB OpenOCD Debugging” on the left panel to create a new debug configuration.

5) Follow the Figures 11, 12, 13, and 14 to configure the new debug configuration. Modify all user dependent properties on directory paths on each “Main”, “Debugger”, and “Startup” tabs.

6) Then click “Debug” to start the debug session.



Name: hexfive-kern-arm_IMX8QM

Project:

C/C++ Application: /home/sandro/multizone-sdk-imx8/multizone.hex

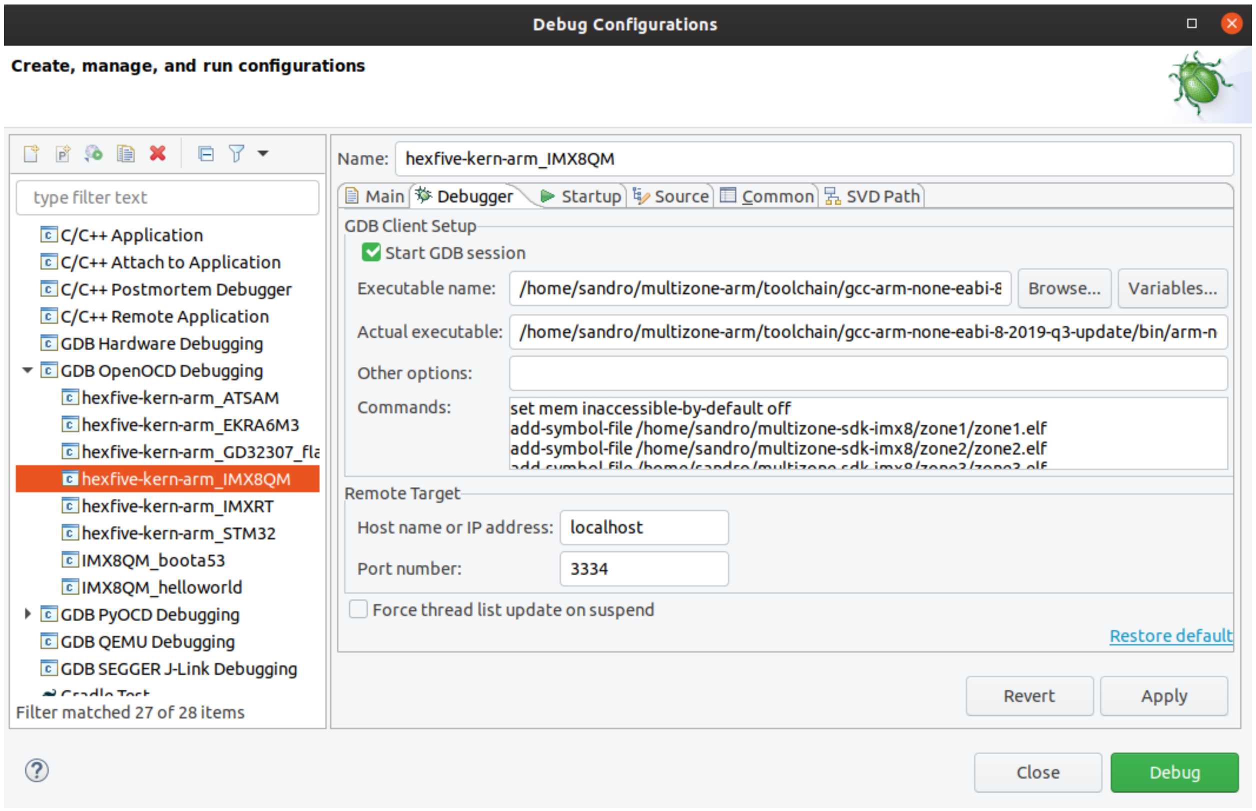
Build (if required) before launching

Build Configuration: Select Automatically

Enable auto build | Disable auto build

[✓ Use workspace settings]

Figure 11. Debug configuration window in the “Main” tab.


Debug Configurations

Create, manage, and run configurations

Name: hexfive-kern-arm_IMX8QM

GDB Client Setup

[✓] Start GDB session

Executable name: /home/sandro/multizone-arm/toolchain/gcc-arm-none-eabi-8

Actual executable: /home/sandro/multizone-arm/toolchain/gcc-arm-none-eabi-8-2019-q3-update/bin/arm-n

Other options:

Commands:
```
set mem inaccessible-by-default off
add-symbol-file /home/sandro/multizone-sdk-imx8/zone1/zone1.elf
add-symbol-file /home/sandro/multizone-sdk-imx8/zone2/zone2.elf
```

Remote Target

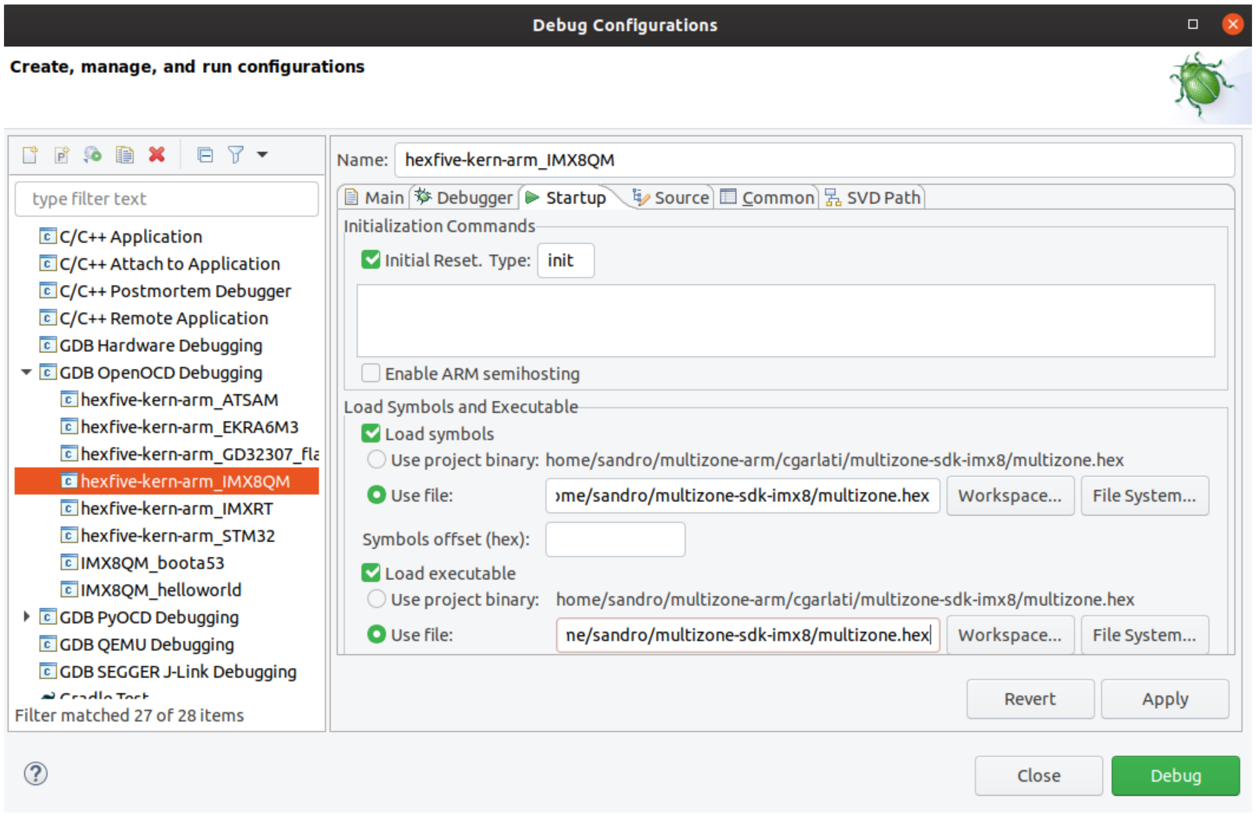
Host name or IP address: localhost

Port number: 3334

[ ] Force thread list update on suspend

Filter matched 27 of 28 items

Figure 12. Debug configuration window in the “Debugger” tab.


Debug Configurations

Create, manage, and run configurations

Name: hexfive-kern-arm_IMX8QM

Initialization Commands

[✓] Initial Reset. Type: init

[ ] Enable ARM semihosting

Load Symbols and Executable

[✓] Load symbols

○ Use project binary: home/sandro/multizone-arm/cgarlati/multizone-sdk-imx8/multizone.hex

● Use file: ›me/sandro/multizone-sdk-imx8/multizone.hex

Symbols offset (hex):

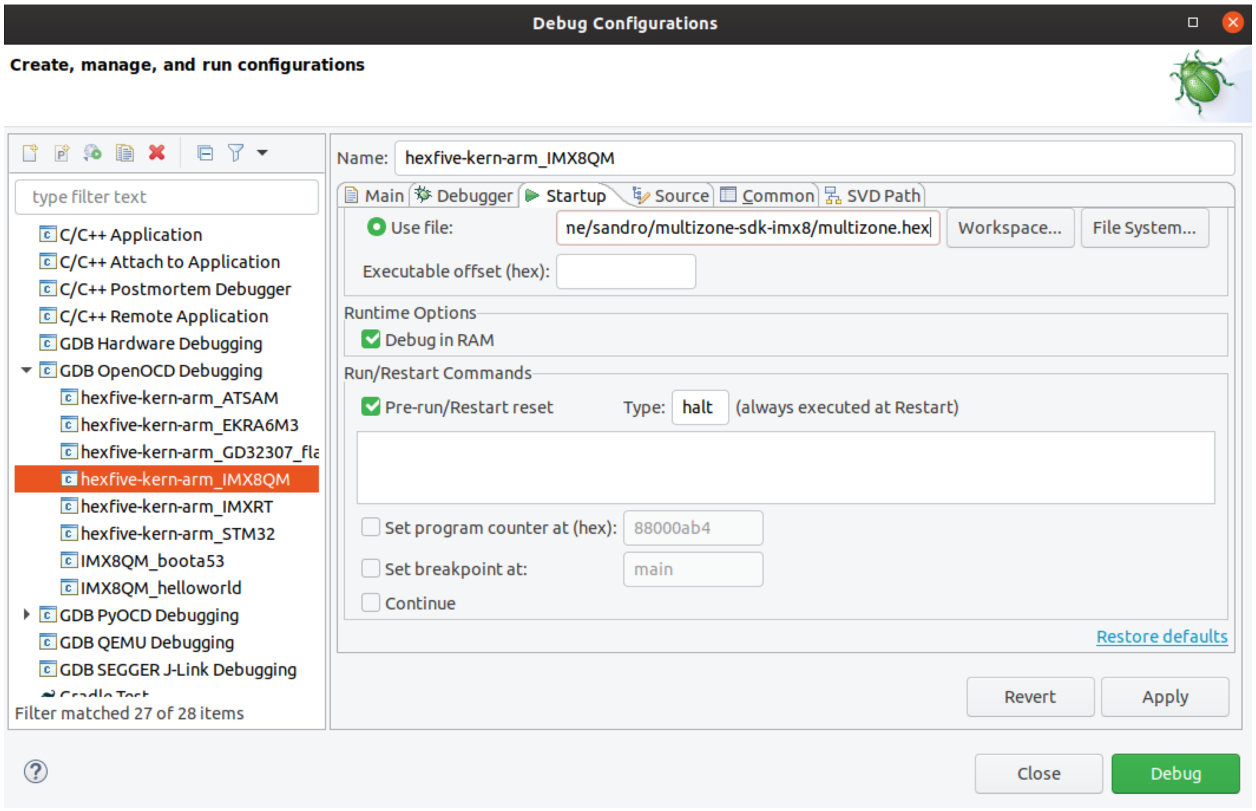
[✓] Load executable

○ Use project binary: home/sandro/multizone-arm/cgarlati/multizone-sdk-imx8/multizone.hex

● Use file: ne/sandro/multizone-sdk-imx8/multizone.hex

Filter matched 27 of 28 items

Figure 13. Debug configuration window in the “Startup” tab.



C/C++ Application
C/C++ Attach to Application
C/C++ Postmortem Debugger
C/C++ Remote Application
GDB Hardware Debugging
GDB OpenOCD Debugging
hexfive-kern-arm_ATSAM
hexfive-kern-arm_EKRA6M3
hexfive-kern-arm_GD32307_fla
hexfive-kern-arm_IMX8QM
hexfive-kern-arm_IMXRT
hexfive-kern-arm_STM32
IMX8QM_boota53
IMX8QM_helloworld
GDB PyOCD Debugging
GDB QEMU Debugging
GDB SEGGER J-Link Debugging
Filter matched 27 of 28 items

Name: hexfive-kern-arm_IMX8QM

Main | Debugger | Startup | Source | Common | SVD Path

● Use file: ne/sandro/multizone-sdk-imx8/multizone.hex [Workspace...] [File System...]
Executable offset (hex):

Runtime Options
✓ Debug in RAM

Run/Restart Commands
✓ Pre-run/Restart reset Type: halt (always executed at Restart)

☐ Set program counter at (hex): 88000ab4
☐ Set breakpoint at: main
☐ Continue

Figure 14. Debug configuration window in the "Startup" tab.

Security and Performance Assessment

This section focuses on the security and performance system's assessment. It is split in two parts: the first part focuses on explaining the security provided by MultiZone on the Cortex-M4 Platform 0 as well as assessing the performance impact; the second part focuses on the full system security.

Trusted Applications

For the first part of the security and performance assessment, please take as reference for the multizone.cfg file the configuration depicted in Listing 2.

```
# Copyright(C) 2020 Hex Five Security, Inc. - All Rights Reserved

# MultiZone reserved memory: 16K @0x1FFE0000, 4K @0x20000000

tick = 10 # ms

zone = 1
       dev  = M4_0, M4_0_UART
       base = 0x1FFE8000; size = 32K; rwx = rx # PROGRAM
       base = 0x20004000; size =  4K; rwx = rw # DATA

zone = 2
       dev  = M4_0, M4_0_SEMA42
       base = 0x1FFF0000; size =  8K; rwx = rx # PROGRAM
       base = 0x20005000; size =  4K; rwx = rw # DATA

zone = 3
       dev  = M4_0, M4_0_RGPIO
       base = 0x1FFF2000; size =  8K; rwx = rx # PROGRAM
       base = 0x20006000; size =  4K; rwx = rw # DATA

zone = 4
       dev  = M4_0, M4_0_TPM
       base = 0x1FFF4000; size =  8K; rwx = rx # PROGRAM
       base = 0x20007000; size =  4K; rwx = rw # DATA
```

Listing 2. MultiZone policy definition file

At any time, it is possible to enter an empty line to show the list of commands available:

```
Z1 >
Commands: yield send recv mpu load store exec stats timer restart
```

- "***mpu***": shows the zone isolation policies.
- "***load***" and "***store***": reads and writes data from/to an arbitrary memory location.
- "***exec***": jumps the zone execution to an arbitrary memory location.
- "***send***" and "***recv***": sends and receives messages to/from any zone.

- "***yield***": yields the CPU to the next zone showing the time taken to loop through all zones .
- "***stats***": repeats the yield command ten times and prints detailed kernel statistics.
- "***timer***": sets the zone timer to current time plus a time delay expressed in milliseconds.
- "***restart***": jumps the zone execution to its first address restarting the zone.

At any time, it is possible to type the command "restart" to restart the individual zone and refresh the splash screen – this will not affect in any way the other zones. At any time, it is also possible to use the UP arrow to recall the previously entered command.

Secure execution

All zones code runs securely in Unprivileged Thread mode including user-mode interrupts handlers. You can see this from the splash screen that shows the content of privileged registers otherwise not available in Unprivileged Thread mode – CPUID which specifies the Implementer (0x41, Arm), Variant (0x1, Revision 1), PartNo (0xC24, Cortex-M4), and Revision (0x1, Patch 1).

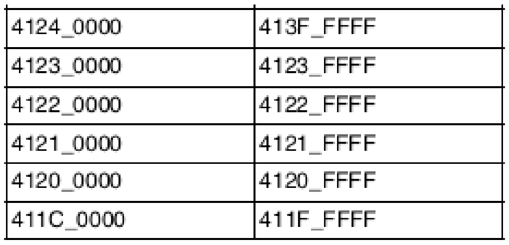
```
Implementer    : 0x41, Arm.
Variant        : 0x0, Revision 0.
PartNo         : 0xC24, Cortex-M4.
Revision       : 0x1, Patch 1.
```

Memory Protection

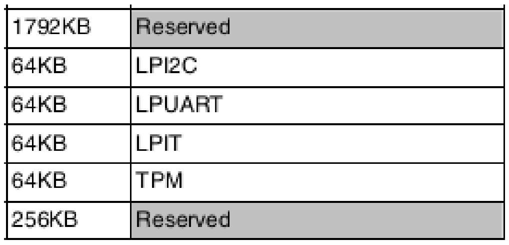
From any terminal session type "***mpu***" to show the zone isolation policies defined in the MultiZone configuration file (Listing 2).

```
Z1 > mpu

0x1FFE8000 0x1FFEFFFF r-x
0x20004000 0x20004FFF rw-
0x41220000 0x4122FFFF rw-
```

Note: *The last memory entry corresponds to the memory-map area of the M4_0_UART. The well-defined memory map of the platform defines that the memory area reserved for the M4_0_UART has a size of 64KB.*



4124_0000	413F_FFFF
4123_0000	4123_FFFF
4122_0000	4122_FFFF
4121_0000	4121_FFFF
4120_0000	4120_FFFF
411C_0000	411F_FFFF



1792KB	Reserved
64KB	LPI2C
64KB	LPUART
64KB	LPIT
64KB	TPM
256KB	Reserved

Only access consistent with these Memory Protection Unit settings will take place. Any attempt to violate the security boundaries will get a hardware exception that is rapped back and displayed on the screen.

1) Read from a memory address assigned to zone 1:

```
Z1 > load 0x1FFE8001
0x1ffe8001 : 0x50
```

2) Read from a memory address not assigned to zone 1:

```
Z1 > load 0x1FFF0000
Memory protection fault : 0x1ffe9706
Press any key to restart ...
```

3) Write to a read-write memory address assigned to zone 1:

```
Z1 > store 0x20004FFF aa
0x20004fff : 0xaa
```

Note: *Depending on your target read-write memory address you may corrupt the heap or stack of the zone, leading the zone to "crash".*

4) Read from the same memory address to verify the value in memory was modified:

```
Z1 > load 0x20004FFF
0x20004fff : 0xaa
```

Note: *Depending on your target address you may have changed a memory address which can be modified by the Zone at runtime, leading a consecutive load to the same address to not display the value that you have stored. The same rationale applies to memory-mapped peripherals that may have read-only or write-only registers.*

5) Write to a read and execute memory address assigned to zone 1:

```
Z1 > store 0x1FFE8000 cc
Memory protection fault : 0x1ffe976a
Press any key to restart ...
```

6) Read from a memory address belonging to a memory-map peripheral (M4_0_UART):

```
Z1 > load 0x41220000
0x41220000 : 0x03
```

Note: *Write operations to* memory-mapped peripheral registers may change the configuration of the device and lead to unexpected and unintended behaviours. *Write operation to read-only registers may have no effect.*

7) Read from another memory address belonging to a memory-map peripheral (M4_0_UART), but without a mapped register:

```
Z1 > load 0x41220030
Bus fault : 0x1ffe9706
Press any key to restart ...
```

Note: *Although the reserved memory-map area for the M4_0_UART is 64KB long, the peripheral just exposes twelve 32-bit registers.*

Offset	Register	Width (In bits)	Access	Reset value
0h	Version ID Register (VERID)	32	RO	0401_0003h
4h	Parameter Register (PARAM)	32	RO	0000_0505h
8h	LPUART Global Register (GLOBAL)	32	RW	0000_0000h
Ch	LPUART Pin Configuration Register (PINCFG)	32	RW	0000_0000h
10h	LPUART Baud Rate Register (BAUD)	32	RW	0F00_0004h
14h	LPUART Status Register (STAT)	32	RW	00C0_0000h
18h	LPUART Control Register (CTRL)	32	RW	0000_0000h
1Ch	LPUART Data Register (DATA)	32	RW	0000_1000h
20h	LPUART Match Address Register (MATCH)	32	RW	0000_0000h
24h	LPUART Modem IrDA Register (MODIR)	32	RW	0000_0000h
28h	LPUART FIFO Register (FIFO)	32	RW	00C0_0044h
2Ch	LPUART Watermark Register (WATER)	32	RW	0000_0000h

Secure Messaging

From any terminal session type "***send***" and "***rscv***" to display the messaging-related commands syntax:

```
Z1 > send
Syntax: send {1|2|3|4} message

Z1 > recv
Syntax: recv {1|2|3|4}
```

1) Send a message to the local zone and observe the local reply:

```
Z1 > send 1 mustang

Z1 > recv 1
msg : mustang
```

2) Send a “***ping***” message to each zone and observe the local reply:

```
Z1 > send 2 ping

Z2 > pong

Z1 > send 3 ping

Z3 > pong

Z1 > send 4 ping

Z4 > pong
```

3) Block a zone by sending a “***block***” message:

```
Z1 > send 2 block
```

4) Send a “***ping***” message to the blocked zone. Observe that no reply has come back.

```
Z1 > send 2 ping
```

5) Send a second ping. Observe that the inbox is full, because zone 2 is blocked and cannot process incoming messages:

```
Z1 > send 2 ping
Error: Inbox full.
```

Safety Critical Applications

The MultiZone runtime implements a dual policy scheduler - preemptive / cooperative. This guarantees that no zone can stop the system in safety critical applications while simultaneously allowing for responsive real-time applications with minimal interrupt latency. Zones normally yield context when waiting for external events. If the zone doesn’t release the CPU in the configured time (tick = 10ms in this system configuration), the runtime kicks in and execution continues with the next zone.

1) Enter “***yield***” to measure the current round robin time:

```
Z1 > yield
yield : elapsed time 1us
```

2) Block a zone by sending a “***block***” message:

```
Z1 > send 2 block
```

3) Enter “***yield***” again. Observe that yield time has increase to the order of magnitude of the Tick time:

```
Z1 > yield
yield : elapsed time 10077us
```

4) Block another zone:

```
Z1 > send 3 block
```

5) Enter “***yield***” again. Observe that yield time has increase to two time the order of magnitude of the Tick time:

```
Z1 > yield
yield : elapsed time 20152us
```

6) Change the Tick parameter to 1ms. Recompile your system. Block one zone and enter “***yield***” again. Observe the change on the yield time:

```
Z1 > send 2 block

Z1 > yield
yield : elapsed time 1008us
```

Performance Statistics

Zone 1 makes available two specific commands to test the performance of the system: “***yield***” and “***stats***”. The “***yield***” measures the round robin trip time for all zones while the “***stats***” measures and summarizes ten rounds of the yield and provides additional runtime statistics (kernel context switch time and interrupt latency).

1. Enter “***yield***” to measure the current round robin trip time for the whole 4 zones:

```
Z1 > yield
yield : elapsed time 1us
```

Note: *All zones are in the WFI state.*

2. Enter “***stats***” to measure and summarize ten rounds of yield – min/med/max:

```
Z1 > stats
187 instr 374 cycles  1 us
187 instr 374 cycles  1 us
187 instr 374 cycles  1 us
187 instr 374 cycles  1 us
187 instr 374 cycles  1 us
187 instr 374 cycles  1 us
187 instr 374 cycles  1 us
187 instr 374 cycles  1 us
187 instr 374 cycles  1 us
187 instr 374 cycles  1 us
187 instr 374 cycles  1 us
-----------------------------------------
instrs  min/med/max = 187/187/187
cycles  min/med/max = 374/374/374
time    min/med/max = 1/1/1 us
```

3. Observe MultiZone runtime context switch and interrupt latency metrics at the very bottom (Kernel) – i.e. ***cycles*** = 201 and ***irq lat cycles*** = 62.

```
Kernel
-----------------------------------------
instrs = 80
cycles = 201
time   = 0 us
-----------------------------------------
irq lat instrs = 35
irq lat cycles = 62
time           = 0 us
```

Note: *MultiZone context switch is too fast to be measured in microseconds (time = 0) – considering an operation frequency of 264MHz, the exact context switch time is 761 nanoseconds and the interrupt latency is 235 nanoseconds.*

4) Block all the three other zones:

```
Z1 > send 2 block

Z1 > send 3 block

Z1 > send 4 block
```

5) Run the "***stats***" command again. Observe MultiZone runtime context switch time (Kernel) has dropped to 132 cycles. The reason behind the drop is related to the fact zones are now not going through the WFI (because are blocked and so looping indefinitely), which imposes less pressure on the scheduler.

```
Kernel
```

```
----------------------------------------
instrs = 41
cycles = 132
time   = 0 us
----------------------------------------
irq lat instrs = 35
irq lat cycles = 62
time           = 0 us
```

Note: *Considering an operation frequency of 264MHz, the exact context switch time is 500 nanoseconds.*

To exercise a bit the impact of MultiZone runtime in the overall performance of the system let's consider a different system tick values. Considering the CPU is operating at 264MHz, for a Tick time of 10ms, the expected performance overhead is between 0.00761% and 0.00500%. For a Tick time of 1ms, the expected performance overhead is between 0.0761% and 0.0500%. Everyone can easily conclude these values are completely negligible and that MultIZone has almost no performance impact in the system.

System policies enforcement

Before delving into the details of the full system security assessment let's start by explaining why there is a need to enforce security policies in the full system. To assess the security posture of the full system, Hex Five has developed a QNX root application that performs operations over arbitrary physical memory addresses. In the QNX system, the "***multizone-qnx-test***" application can be run by executing the following command:

```
# ./multizone-qnx-test
```

At any time, it is possible to enter an empty line to show the list of commands available:

```
QNX >
Commands: load store dump erase
```

- "***load***" and "***store***": reads and writes data from/to an arbitrary memory location.
- "***dump***": reads a consecutive number of bytes starting from a specific address.
- "***erase***": writes a consecutive number of zeros to memory, starting from a specific address.

This table shows a simplified view of the system memory map.

Start Address	End Address	Region	Size	Allocation
8_8000_0000	B_FFFF_FFFF	DDR	14GB	DDR Main Memory (high)
8000_0000	FFFF_FFFF	DDR	2GB	DDR Main Memory (low)
7000_0000	7FFF_FFFF	HSIO	256MB	PCIe1

6000_0000	6FFF_FFFF	HSIO	256MB	PCIe0
3800_0000	3BFF_FFFF	CM4_1	64MB	Cortex-M4 Platform 1
3400_0000	37FF_FFFF	CM4_0	64MB	Cortex-M4 Platform 0
0800_0000	1BFF_FFFF	LSIO	320MB	FlexSPI0 + Other
0010_0000	0013_FFFF	LSIO	256KB	OCRAM
0000_0000	0001_7FFF	LSIO	96KB	OCRAM alias (lower 96KB)

1) Read from a memory address belonging to the OCM alias memory:

```
QNX > load 0x0
0x0000000000000000: 0x6e
```

2) Write to a memory address belonging to the OCM memory:

```
QNX > store 0x100000 0x100
0x0000000000100000: 0x100
```

Note: *Store operations (writes) are 1, 2 or 4 Bytes depending on the size of the value. Load operations (reads) are always 1 Byte.*

3) Read from the memory alias of the previous address. Observe the value was in fact written:

```
QNX > load 0x0
0x0000000000000000: 0x00

QNX > load 0x1
0x0000000000000001: 0x01
```

4) Dump 128 bytes from the OCM alias memory:

```
QNX > dump 0x0 128
0x0000000000000000: 6e a6 a6 50 48 8b 60 f8 04 53 aa 81 |n..PH....S..|
0x000000000000000c: 80 29 51 26 49 99 25 65 2a 02 40 07 |..Q.I..e....|
0x0000000000000018: 00 4a 85 2c 28 d8 5b 80 df f8 ef f6 |.J..........|
0x0000000000000024: bf ed e7 54 be 7f ed b3 ee db ef 7f |...T........|
0x0000000000000030: f5 e9 3e 7f db da ce fb df dd 77 87 |..........w.|
0x000000000000003c: ff 05 de 2f 8d 18 56 9b 2b 20 25 0e |......V.....|
0x0000000000000048: 05 01 23 61 0c 04 20 0a 06 85 10 bc |...a........|
0x0000000000000054: 30 00 44 3a 9a 67 6e 12 a1 60 30 cf |0.D..gn...0.|
0x0000000000000060: b5 9a f3 7f ff 6c 69 60 75 fa 65 f3 |.....li.u.e.|
0x000000000000006c: 5d 81 f7 ad f3 f4 7c 95 0f 7c ec 7f |............|
0x0000000000000078: f8 6e 11 a8 b5 e8 9f f2             |.n......    |
```

5) Read from a memory address belonging to the PCIe0:

```
QNX > load 0x6fffffff
0x000000006fffffff: 0xff
```

6) Write and read to/from DDR memory:

```
QNX > store 0x80000000 0xbb
0x0000000080000000: 0xbb

QNX > load 0x80000000
0x0000000080000000: 0xbb
```

7) Write and read to/from the TCM of the Cortex-M4 Platform 1:

```
QNX > store 0x38FE0001 0xcc
0x0000000038fe0001: 0xcc

QNX > load 0x38FE0001
0x0000000038fe0001: 0xcc
```

8) Read from a memory address belonging to the Cortex-M4 Platform 0:

```
QNX > load 0x34FE8001
0x0000000034fe8001: 0x50
```

Note: *The TCM memory of the Cortex-M4 CPU 0 is addressable in the range 0x34FE0000-0x3501FFFF (system view) / 0x1FFE0000-0x2001FFFF (local view).*

9) Repeat the same operation on the MultiZone terminal side, taking into account the local memory view. Observe the same value from two different memory map views:

```
Z1 > load 0x1FFE8001
0x1ffe8001 : 0x50
```

10) Write to the same memory address belonging to the Cortex-M4 Platform 0. Repeat the same load operation on the MultiZone side:

```
QNX > store 0x34FE8001 0xff
0x0000000034fe8001: 0xff
```

```
Z1 > load 0x1FFE8001
0x1ffe8001 : 0xff
```

At this point, since no isolation (zone) has been enforced over QNX (please refer to Listing 2), the QNX cluster has access to the full system, including the Cortex-M4 platform 0 where MultiZone runs. This highlights that no matter how strong and robust the isolation implemented on the Cortex-M4 platform 0 is, if the separation it's not approached from a complete system perspective, this may have high damaging consequences for the system.

11) For example, run the "***erase***" command to kill the MultiZone system completely. Observe that you cannot interact with the MultiZone terminal anymore and you need to reboot the system.

```
QNX > erase 0x34FE4000 8192
```

Verify system policies enforcement

For the remainder of this section, please take as reference Listing 3 for the system configuration.

There are two additional zones. Zone 5 is assigned to the Cortex-M4 CPU 1 and has been mapped to the memory region of the Cortex-M4 Platform 1. Zone 6 is assigned to the QNX cluster, mapped to the remaining masters 4xCortex-A53 and 2xCortex-A72, and mapped to 4 memory regions: low and high DDR and PCIe 0 and 1. At this stage the OCM-related memory range is commented out.

```
# MultiZone reserved memory: 16K @0x1FFE0000, 4K @0x20000000

tick = 10 # ms

zone = 1
       dev  = M4_0, M4_0_UART
       base = 0x1FFE8000; size = 32K; rwx = rx # PROGRAM
       base = 0x20004000; size =  4K; rwx = rw # DATA

zone = 2
       dev  = M4_0, M4_0_SEMA42
       base = 0x1FFF0000; size =  8K; rwx = rx # PROGRAM
       base = 0x20005000; size =  4K; rwx = rw # DATA

zone = 3
       dev  = M4_0, M4_0_RGPIO
       base = 0x1FFF2000; size =  8K; rwx = rx # PROGRAM
       base = 0x20006000; size =  4K; rwx = rw # DATA

zone = 4
       dev  = M4_0, M4_0_TPM
       base = 0x1FFF4000; size =  8K; rwx = rx # PROGRAM
       base = 0x20007000; size =  4K; rwx = rw # DATA

zone = 5
       dev  = M4_1, M4_1_RGPIO, M4_1_UART
       base = 0x38000000; size = 64M; rwx = rwx # M4_1_MEM
```

```
zone = 6
       dev  = A53, A72
       #base = 0x000000000; size = 448M; rwx = rwx; # OCM
       base = 0x060000000; size = 256M; rwx = rwx; # PCIe0
       base = 0x070000000; size = 256M; rwx = rwx; # PCIe1
       base = 0x080000000; size =   2G; rwx = rwx; # DDR (low)
       base = 0x880000000; size =  30G; rwx = rwx; # DDR (high)
```

Listing 3. MultiZone policy definition file

Rebuild the MultiZone TEE to apply the new policies and reboot the target. Connect over ssh, login as root/root, and run the "./***multizone-qnx-test***" application.

1) Read an address in the OCM memory region. Observe the returned value 0x0. The QNX cluster in Zone 6 has no longer read access to this region.

```
QNX > load 0x0
0x0000000000000000: 0x00
```

2) Read a consecutive block of 128 bytes from the OCM alias memory. Observe that all reads return value 0x0, indicating that QNX has no longer read access to this region.

```
QNX > dump 0x0 128
0x0000000000000000: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x000000000000000c: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x0000000000000018: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x0000000000000024: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x0000000000000030: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x000000000000003c: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x0000000000000048: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x0000000000000054: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x0000000000000060: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x000000000000006c: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x0000000000000078: 00 00 00 00 00 00 00 00             |........    |
```

3) Read a memory address in the PCIe0 region. Observe you can read the same value.

```
QNX > load 0x6fffffff
0x000000006fffffff: 0xff
```

4) Write and read to/from DDR memory:

```
QNX > store 0x80000000 0xbb
0x0000000080000000: 0xbb

QNX > load 0x80000000
0x0000000080000000: 0xbb
```

5) Read from a memory location in the Cortex-M4 Platform 0 region. Observe the read value is now 0x00.

```
QNX > load 0x34FE8001
0x0000000034fe8001: 0x00
```

6) Read a consecutive block of 128 bytes from the TCM memory space of the Cortex-M4 Platform 0. Observe all values are now 0x0, indicating that the memory is not accessible to QNX.

```
QNX > dump 0x34FE4000 128
0x0000000034fe4000: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x0000000034fe400c: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x0000000034fe4018: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x0000000034fe4024: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x0000000034fe4030: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x0000000034fe403c: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x0000000034fe4048: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x0000000034fe4054: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x0000000034fe4060: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x0000000034fe406c: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x0000000034fe4078: 00 00 00 00 00 00 00 00             |........    |
```

7) Repeat the "***erase***" command that caused complete TEE failure. Observe that the new system separation policies shield the TEE from this attack and instead stop the QNX application, which becomes unresponsive. Observe that you can still interact with the MultiZone TEE via the serial terminal connected to zone 1.

```
QNX > erase 0x34FE4000 8192
```

Note: alternatively, run this step 7 via UART, instead of ssh, to see the QNX fault message.

8) Reboot the target and restart the "***multizone-qnx-test***" application. This time, read 128 bytes from the TCM of the Cortex-M4 Platform 1. Observe all values are now 0x0, indicating that the memory is not also accessible to QNX. The same isolation exists due to zone 5.

```
QNX > dump 0x3BFE4000 128
0x000000003bfe4000: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x000000003bfe400c: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x000000003bfe4018: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x000000003bfe4024: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x000000003bfe4030: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x000000003bfe403c: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x000000003bfe4048: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x000000003bfe4054: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x000000003bfe4060: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x000000003bfe406c: 00 00 00 00 00 00 00 00 00 00 00 00 |............|
0x000000003bfe4078: 00 00 00 00 00 00 00 00             |........    |
```

9) In the reference multizone.cfg (Listing 3), uncomment the OCM-related memory region.

```
zone = 6
       dev  = A53, A72
       base = 0x000000000; size = 448M; rwx = rwx; # OCM
       base = 0x060000000; size = 256M; rwx = rwx; # PCIe0
       base = 0x070000000; size = 256M; rwx = rwx; # PCIe1
       base = 0x080000000; size =   2G; rwx = rwx; # DDR (low)
       base = 0x880000000; size =  30G; rwx = rwx; # DDR (high)
```

10) Rebuild the MultiZone TEE and reboot the target. Login as root/root and run the "./***multizone-qnx-test***" application. Read 128 bytes from the OCM alias memory. Observe QNX access to the OCM memory.

```
QNX > dump 0x0 128
0x0000000000000000: 6e a7 ae 50 48 8b 60 e9 04 53 28 91 |n..PH....S..|
0x000000000000000c: 80 29 11 26 49 99 25 25 23 02 40 06 |....I.......|
0x0000000000000018: 00 4a 45 2e a8 d8 53 80 5f f8 ef f6 |.JE...S.....|
0x0000000000000024: bf ed e7 5d bf 7f fd b3 ee d3 cf 7f |............|
0x0000000000000030: f5 f9 3e 6f df fa ce fb cf db f7 07 |...o........|
0x000000000000003c: 7f 05 7f 2f 8d 58 16 9b 2b 60 21 0e |.....X......|
0x0000000000000048: 04 00 05 63 4c 84 30 0a 06 05 48 ae |...cL.0...H.|
0x0000000000000054: 30 00 44 3b 9a 67 6e 13 a0 20 30 cf |0.D..gn...0.|
0x0000000000000060: a5 9e f7 7f ff 6c cf 60 75 ba 25 f7 |.....l..u...|
0x000000000000006c: 5d 31 d5 ee f9 f4 7c 95 0f 78 cc 3d |.1.......x..|
0x0000000000000078: fb 7e 15 a8 b5 e8 9f f2             |........    |
```

Note: The contents of the OCM memory change across power cycles as this memory region is not used and therefore not initialized.

11) Repeat the "***erase***" attack. Observe that QNX is stopped while the TEE is unaffected as you can verify by connecting to zone 1 via serial terminal.

```
QNX > erase 0x34FE4000 8192
```

Appendix I - MultiZone Policies

This section explains syntax and semantics of the MultiZone policies. The MultiZone SDK stores these policies in the bsp/IMX8QM/multizone.cfg file. This is a plain text format file that can be modified with any text editor of choice. The content of this file is case insensitive. White space characters are ignored. A typical policy configuration file includes comments and definitions for tick, zones, devices, and memory regions.

```
# Copyright(C) 2020 Hex Five Security, Inc. - All Rights Reserved

# MultiZone reserved memory: 16K @0x1FFE0000, 4K @0x20000000

tick = 10 # ms

zone = 1
       dev  = M4_0, M4_0_UART
       base = 0x1FFE8000; size = 32K; rwx = rx # PROGRAM
       base = 0x20004000; size =  4K; rwx = rw # DATA

zone = 2
       dev  = M4_0, M4_0_SEMA42
       base = 0x1FFF0000; size =  8K; rwx = rx # PROGRAM
       base = 0x20005000; size =  4K; rwx = rw # DATA

zone = 3
       dev  = M4_0, M4_0_RGPIO
       base = 0x1FFF2000; size =  8K; rwx = rx # PROGRAM
       base = 0x20006000; size =  4K; rwx = rw # DATA

zone = 4
       dev  = M4_0, M4_0_TPM
       base = 0x1FFF4000; size =  8K; rwx = rx # PROGRAM
       base = 0x20007000; size =  4K; rwx = rw # DATA

zone = 5
       dev  = M4_1, M4_1_RGPIO, M4_1_UART
       base = 0x38000000; size = 64M; rwx = rwx # M4_1_MEM

zone = 6
       dev  = A53, A72
       base = 0x000000000; size = 448M; rwx = rwx; # OCM
       base = 0x060000000; size = 256M; rwx = rwx; # PCIe0
       base = 0x070000000; size = 256M; rwx = rwx; # PCIe1
       base = 0x080000000; size =   2G; rwx = rwx; # DDR (low)
       base = 0x880000000; size =  30G; rwx = rwx; # DDR (high)
```

Listing 4. MultiZone policy definition file

Comments

The MultiZone Configurator syntax specifies that comments are marked by the '#' symbol. The first comment line after the copyright notice is a reminder of the memory regions reserved to the MultiZone runtime. These memory regions cannot be assigned to zones and will raise a configuration error if used:

```
Error  : zone 1 range 3 - kernel reserved [0x1FFE0000 - 0x1FFE4000].
```

Tick

The *tick* parameter drives the preemptive scheduler. It specifies the maximum amount of time in milliseconds that each zone can hold the CPU. If a zone exceeds this limit, the preemptive scheduler suspends the zone execution and moves to the next according to a fair round robin schema. If the *tick* value is set to 0, the scheduler switches to fully cooperative: zones must explicitly yield execution via the *MZONE_YIELD* api for the other zones to run. The range allowed for the tick value is between 0 to 1000; any attempt to specify a tick value outside of this range will trigger a configuration error:

```
Error  : bsp/IMX8QM/multizone.cfg (5) - Invalid tick value 10000, range 0 to 1000.
```

Zones

Zones are equivalent to hardware threads mapped to specific hardware resources. Zones define a logical partitioning of the whole system including devices, core masters, memory-mapped peripherals, and contiguous memory regions.

Zones are numbered consecutively starting from zone 1. Gaps are not allowed. Invalid zone numbers trigger a configuration error:

```
Error  : bsp/IMX8QM/multizone.cfg (13) - Invalid zone number 3, range 1 to 8.
```

The number of zones is limited by the resources of the given system. In the case of the i.MX8QM the maximum number of zones is 8 of which up to 4 are assigned to the Cortex-M4 CPU 0 (M4_0). Exceeding these ranges triggers a configuration error:

```
Error  : bsp/IMX8QM/multizone.cfg (46) - Invalid zone number 9, range 1 to 8.
```

```
Error  : invalid zone count for device M4_0 (5) - must be 1 to 4.
```

Zones assigned to the M4_0 go first and must be numbered 1 to 4:

```
Error  : invalid zone order - M4_0 zones go first.
```

Each zone must be assigned at least to one master core. Zones assigned to Cortex-M4 CPU 0 (M4_0) or

Cortex-M4 CPU 1 (M4_1) cannot include other cores (i.e., Cortex-A53 cluster, Cortex-A53 CPU 0, Cortex-A72 cluster,etc):

```
Error  : zone 5 - no masters assigned to this zone.
```

```
Error  : zone 4 - too many masters assigned to this zone.
```

Note: The last Zone defined receives by default all unmapped devices. This keeps the configuration file readable in systems like the i.MX8QM that have several hundreds of devices. This applies only to devices: memory regions must be explicitly assigned (whitelisted) to all zones including the last zone.

Note: MultiZone policies are enforced only to masters specifically assigned to zones. Masters not assigned to any zones are unconstrained. This configuration is intended for development and test and should not be deployed in production environments.

Devices

Devices include core masters and memory-mapped peripherals. The list of supported devices is included in Appendix. This is a subset of the i.MX8QM resources[6]. This version of the MultiZone SDK includes support for 106 resources.

The assignment of a device to a zone implicitly grants read / write access to its memory region. The maximum number of devices that can be assigned to a M4 master is a function of the number of memory regions allocated. The maximum total number of devices and memory regions is 8.

```
Error  : zone 1 - Too many regions/devices (max 8).
```

There is no limit to the number of devices that can be assigned to the other master cores.

Devices cannot be mapped to multiple zones.

```
Error  : zone 2 - device M4_0_UART already assigned to zone 1.
```

The assignment of a device to a M4_0 zone implicitly maps the device interrupt sources to the zone. Note that the M4_0 local devices have IRQ number < 128 - i.e M4_0_UART IRQ number is 41.

Memory Regions

Memory regions represent contiguous blocks of memory and must be explicitly mapped to zones on a white-list basis. Memory region attributes include: start address, size, and read / write / execute flags.

M4_0 zones can be mapped to a maximum of 8 memory regions each. Note that each device assigned to a M4_0 zone consumes one memory region, thus reducing the maximum .

```
Error  : zone 1 - Too many regions/devices (max 8).
```

Zones assigned to the other masters can be mapped to a maximum of 10 memory regions each.

```
Error  : too many RDC memory regions (max=10).
```

Regions assigned to M4_0 zones can specify any combination of RWX access. The first region is for the program segment and should have RX access. Otherwise a warning is triggered:.

```
Warning: zone 1 range 1 - should include 'rx' policy.
```

Regions assigned to the other masters must have RWX access.

```
Error  : zone 6 range 1 - invalid RDC policy (should be 'rwx').
```

M4_0 memory region definitions can overlap across M4_0 zones. This might be necessary in some particular situation although in general it is considered not secure and will trigger a warning. It is not possible to define overlapping regions for the other masters.

```
Warning: zone 3 range 2 overlaps zone 2 range 3.
```

```
Error  : zone 6 range 4 overlaps zone 6 range 3 (xRDC).
```

Note: Memory regions definitions for the M4_0 use the M4_0 local memory map view. Memory regions definitions for the other masters use the system memory map view. For example, the M4_0 local view address 0x1FFF4000 is equivalent to the system view address 0x34FF4000 used by the other masters.

Appendix II - i.MX8QM Resources List

MultiZone	SCFW Resource Name	ID	Resource Description
A53	SC_R_A53	0	Cortex-A53 cluster
A53_0	SC_R_A53_0	1	Cortex-A53 CPU 0
A53_1	SC_R_A53_1	2	Cortex-A53 CPU 1
A53_2	SC_R_A53_2	3	Cortex-A53 CPU 2
A53_3	SC_R_A53_3	4	Cortex-A53 CPU 3
A72	SC_R_A72	5	Cortex-A72 cluster
A72_0	SC_R_A72_0	6	Cortex-A72 CPU 0
A72_1	SC_R_A72_1	7	Cortex-A72 CPU 1
SPI_0	SC_R_SPI_0	53	SPI 0 from DMA subsystem
SPI_1	SC_R_SPI_1	54	SPI 1 from DMA subsystem
SPI_2	SC_R_SPI_2	55	SPI 2 from DMA subsystem
SPI_3	SC_R_SPI_3	56	SPI 3 from DMA subsystem
UART_0	SC_R_UART_0	57	UART 0 from DMA subsystem
UART_1	SC_R_UART_1	58	UART 1 from DMA subsystem
UART_2	SC_R_UART_2	59	UART 2 from DMA subsystem
UART_3	SC_R_UART_3	60	UART 3 from DMA subsystem
UART_4	SC_R_UART_4	61	UART 4 from DMA subsystem
EMVSIM_0	SC_R_EMVSIM_0	62	EMV_SIM 0 from DMA subsystem
EMVSIM_1	SC_R_EMVSIM_1	63	EMV_SIM 1 from DMA subsystem
DMA_0_CH0	SC_R_DMA_0_CH0	64	DMA 0 channel 0 from DMA subsystem
I2C_0	SC_R_I2C_0	96	I2C 0 from DMA subsystem
I2C_1	SC_R_I2C_1	97	I2C 1 from DMA subsystem
I2C_2	SC_R_I2C_2	98	I2C 2 from DMA subsystem
I2C_3	SC_R_I2C_3	99	I2C 3 from DMA subsystem

I2C_4	SC_R_I2C_4	100	I2C 4 from DMA subsystem
ADC_0	SC_R_ADC_0	101	ADC 0 from DMA subsystem
ADC_1	SC_R_ADC_1	102	ADC 1 from DMA subsystem
FTM_0	SC_R_FTM_0	103	FTM 0 from DMA subsystem
FTM_1	SC_R_FTM_1	104	FTM 1 from DMA subsystem
CAN_0	SC_R_CAN_0	105	CAN 0 from DMA subsystem
CAN_1	SC_R_CAN_1	106	CAN 1 from DMA subsystem
CAN_2	SC_R_CAN_2	107	CAN 2 from DMA subsystem
DMA_1_CH0	SC_R_DMA_1_CH0	108	DMA 1 channel 0 from DMA subsystem
PWM_0	SC_R_PWM_0	191	PWM 0 from LSIO subsystem
PWM_1	SC_R_PWM_1	192	PWM 1 from LSIO subsystem
PWM_2	SC_R_PWM_2	193	PWM 2 from LSIO subsystem
PWM_3	SC_R_PWM_3	194	PWM 3 from LSIO subsystem
PWM_4	SC_R_PWM_4	195	PWM 4 from LSIO subsystem
PWM_5	SC_R_PWM_5	196	PWM 5 from LSIO subsystem
PWM_6	SC_R_PWM_6	197	PWM 6 from LSIO subsystem
PWM_7	SC_R_PWM_7	198	PWM 7 from LSIO subsystem
GPIO_0	SC_R_GPIO_0	199	GPIO 0 from LSIO subsystem
GPIO_1	SC_R_GPIO_1	200	GPIO 1 from LSIO subsystem
GPIO_2	SC_R_GPIO_2	201	GPIO 2 from LSIO subsystem
GPIO_3	SC_R_GPIO_3	202	GPIO 3 from LSIO subsystem
GPIO_4	SC_R_GPIO_4	203	GPIO 4 from LSIO subsystem
GPIO_5	SC_R_GPIO_5	204	GPIO 5 from LSIO subsystem
GPIO_6	SC_R_GPIO_6	205	GPIO 6 from LSIO subsystem
GPIO_7	SC_R_GPIO_7	206	GPIO 7 from LSIO subsystem
GPT_0	SC_R_GPT_0	207	GPT 0 from LSIO subsystem
GPT_1	SC_R_GPT_1	208	GPT 1 from LSIO subsystem
GPT_2	SC_R_GPT_2	209	GPT 2 from LSIO subsystem
GPT_3	SC_R_GPT_3	210	GPT 3 from LSIO subsystem

GPT_4	SC_R_GPT_4	211	GPT 4 from LSIO subsystem
KPP	SC_R_KPP	212	KPP from LSIO subsystem
MU_0A	SC_R_MU_0A	213	Message Unit 0A from LSIO subsystem
MU_1A	SC_R_MU_1A	214	Message Unit 1A from LSIO subsystem
MU_2A	SC_R_MU_2A	215	Message Unit 2A from LSIO subsystem
MU_3A	SC_R_MU_3A	216	Message Unit 3A from LSIO subsystem
MU_4A	SC_R_MU_4A	217	Message Unit 4A from LSIO subsystem
MU_5A	SC_R_MU_5A	218	Message Unit 5A from LSIO subsystem
MU_6A	SC_R_MU_6A	219	Message Unit 6A from LSIO subsystem
MU_7A	SC_R_MU_7A	220	Message Unit 7A from LSIO subsystem
MU_8A	SC_R_MU_8A	221	Message Unit 8A from LSIO subsystem
MU_9A	SC_R_MU_9A	222	Message Unit 9A from LSIO subsystem
MU_10A	SC_R_MU_10A	223	Message Unit 10A from LSIO subsystem
MU_11A	SC_R_MU_11A	224	Message Unit 11A from LSIO subsystem
MU_12A	SC_R_MU_12A	225	Message Unit 12A from LSIO subsystem
MU_13A	SC_R_MU_13A	226	Message Unit 13A from LSIO subsystem
MU_5B	SC_R_MU_5B	227	Message Unit 5B from LSIO subsystem
MU_6B	SC_R_MU_6B	228	Message Unit 6B from LSIO subsystem
MU_7B	SC_R_MU_7B	229	Message Unit 7B from LSIO subsystem
MU_8B	SC_R_MU_8B	230	Message Unit 8B from LSIO subsystem
MU_9B	SC_R_MU_9B	231	Message Unit 9B from LSIO subsystem
MU_10B	SC_R_MU_10B	232	Message Unit 10B from LSIO subsystem
MU_11B	SC_R_MU_11B	233	Message Unit 11B from LSIO subsystem
MU_12B	SC_R_MU_12B	234	Message Unit 12B from LSIO subsystem
MU_13B	SC_R_MU_13B	235	Message Unit 13B from LSIO subsystem
FSPI_0	SC_R_FSPI_0	237	FSPI 0 from LSIO subsystem
FSPI_1	SC_R_FSPI_1	238	FSPI 1 from LSIO subsystem
M4_0	SC_R_M4_0_PID0	278	Cortex-M4 CPU 0
M4_0_RGPIO	SC_R_M4_0_RGPIO	283	GPIO from Cortex-M4 CPU 0

M4_0_SEMA42	SC_R_M4_0_SEMA42	284	SEMA42 from Cortex-M4 CPU 0
M4_0_TPM	SC_R_M4_0_TPM	285	TPM from Cortex-M4 CPU 0
M4_0_PIT	SC_R_M4_0_PIT	286	PIT from Cortex-M4 CPU 0
M4_0_UART	SC_R_M4_0_UART	287	UART from Cortex-M4 CPU 0
M4_0_I2C	SC_R_M4_0_I2C	288	I2C from Cortex-M4 CPU 0
M4_0_INTMUX	SC_R_M4_0_INTMUX	289	INTMUX from Cortex-M4 CPU 0
M4_0_MU_0B	SC_R_M4_0_MU_0B	292	MU 0B from Cortex-M4 CPU 0
M4_0_MU_0A0	SC_R_M4_0_MU_0A0	293	MU 0A0 from Cortex-M4 CPU 0
M4_0_MU_0A1	SC_R_M4_0_MU_0A1	294	MU 0A1 from Cortex-M4 CPU 0
M4_0_MU_0A2	SC_R_M4_0_MU_0A2	295	MU 0A2 from Cortex-M4 CPU 0
M4_0_MU_0A3	SC_R_M4_0_MU_0A3	296	MU 0A3 from Cortex-M4 CPU 0
M4_1	SC_R_M4_1_PID0	298	Cortex-M4 CPU 1
M4_1_RGPIO	SC_R_M4_1_RGPIO	303	GPIO from Cortex-M4 CPU 1
M4_1_SEMA42	SC_R_M4_1_SEMA42	304	SEMA42 from Cortex-M4 CPU 1
M4_1_TPM	SC_R_M4_1_TPM	305	TPM from Cortex-M4 CPU 1
M4_1_PIT	SC_R_M4_1_PIT	306	PIT from Cortex-M4 CPU 1
M4_1_UART	SC_R_M4_1_UART	307	UART from Cortex-M4 CPU 1
M4_1_I2C	SC_R_M4_1_I2C	308	I2C from Cortex-M4 CPU 1
M4_1_INTMUX	SC_R_M4_1_INTMUX	309	INTMUX from Cortex-M4 CPU 1
M4_1_MU_0B	SC_R_M4_1_MU_0B	312	MU 0B from Cortex-M4 CPU 1
M4_1_MU_0A0	SC_R_M4_1_MU_0A0	313	MU 0A0 from Cortex-M4 CPU 1
M4_1_MU_0A1	SC_R_M4_1_MU_0A1	314	MU 0A1 from Cortex-M4 CPU 1
M4_1_MU_0A2	SC_R_M4_1_MU_0A2	315	MU 0A2 from Cortex-M4 CPU 1
M4_1_MU_0A3	SC_R_M4_1_MU_0A3	316	MU 0A3 from Cortex-M4 CPU 1

Appendix III - MultiZone Security API

This section covers the MultiZone Security API. Consistently with MultiZone zero trust design philosophy, this API is not implemented in the form of a traditional static or dynamic library. Instead, only a C header file is provided containing macro expansions into assembly code. To guarantee complete separation no cross-references or linking are required - and in fact even allowed.

The API is logically organized in three groups: thread scheduling and communications, timer management, and high-performance emulated access to privileged registers.

```
/* MultiZone Security API */

#define MZONE_YIELD();
#define MZONE_WFI();
#define MZONE_SEND(zone, msg);
#define MZONE_RECV(zone, msg);

#define MZONE_RDTIME();
#define MZONE_RDTIMECMP();
#define MZONE_WRTIMECMP(cycles);
#define MZONE_ADTIMECMP(cycles);

#define LOAD_SCB(scb_reg);
#define STORE_NVICISER(irq);
#define STORE_NVICICER(irq);
#define LOAD_MPURBAR(val);
#define LOAD_MPURASR(val);
```

Listing 5. MultiZone Security API

Scheduling and communications

API	Syntax and Function	Example
MZONE_YIELD	***void MZONE_YIELD();*** Indicates to the MultiZone scheduler that the zone has nothing to do and causes the scheduler to switch execution to the next zone.	***MZONE_YIELD();*** In the case of a three zone implementation with a tick time of 10ms, the maximum time to come back to context is 20ms, faster if the other zones yield as well.

MZONE_WFI	***void MZONE_WFI();***
	
	Unprivileged implementation of the Wait for Interrupt instruction - wfi. The Wait for Interrupt instruction provides a hint to the scheduler that the current zone can be paused until an interrupt or a message
		int main (void){
		... setup interrupt handlers
		... while(1) {
		...... do something
		...... MZONE_WFI();
		... }
		}

	might need servicing. If all zones are waiting for interrupt, the scheduler puts the core in a suspended low power state.	Typical implementation of the main loop of an event-driven zone.
MZONE_SEND	***int MZONE_SEND(zone num, *char);*** Sends a 16-byte fixed length message from the current zone to zone num. The return value is 1 if the message has been delivered or 0 if the receiving mailbox is full. Delivery is synchronous and non-blocking. Upon delivery, the recipient zone is resumed if it was paused waiting for interrupt.	***int state = MZONE_SEND(2, "ping");*** Sends the "ping" string to Zone 2 - {'p', 'i', 'n', 'g'}; state = 1 if the message is delivered.

MZONE_RECV	***int MZONE_RECV(zone num, *char);***
	
	Checks the inbox for a new message from zone num. If a new message is present, it is copied to the local memory pointed by *char and the inbox made available for a new message. Otherwise returns 0.
		char msg[16];
		int state = MZONE_RECV(1, msg);
		
		If there is a new message from zone number 1 it is copied to the local array.

Timer management

API	Syntax and Function	Example
MZONE_RDTIME	***uint64_t MZONE_RDTIME();*** The Armv7-M architecture defines one 24-bit System Tick Timer. MultiZone provides a separate 64-bit copy of the timer to each zone.	***const uint64_t T0 = MZONE_RDTIME();*** Read the number of cycles passed since system reset.
MZONE_RDTIMECMP	***uint64_t MZONE_RDTIMECMP();*** Read the timer comparator.	***const uint64_t T = MZONE_RDTIMECMP();*** Read the time compare value.

MZONE_WRTIMECMP	***void MZONE_WRTIMECMP(uint64_t cycles);***
	
	Sets the timer comparator.
		const uint64_t T0 = MZONE_RDTIME();
		const uint64_t T1 = T0 + (RTC_FREQ*(ms/1000.0));
		MZONE_WRTIMECMP(T1);
		
		Write the time compare value.

MZONE_ADTIMECMP	***void MZONE_ADTIMECMP(uint64_t cycles);*** Read the timer value, increment by cycles, and set the comparator.	***MZONE_ADTIMECMP(RTC_FREQ*(ms/1000.0));*** Offsets the time compare value by a specific number of cycles.

Access to privileged registers

API	Syntax and Function	Example
LOAD_SCB	***uint32_t LOAD_SCB(uint32_t scb_reg);*** Secure read-only operation of specific privileged system registers: e.g., CPUID (SCS_SCB_CPUID) and VTOR (SCS_SCB_VTOR).	***uint32_t cpuid = LOAD_SCB(SCS_SCB_CPUID);***
STORE_NVICISER	***STORE_NVICISER(uint32_t irq_num);*** Secure access to the privileged interrupt controller (NVIC) kernel driver to **enable** an interrupt registered with the zone. In case the interrupt number is not registered with the zone (by assigning the device to the zone via Multizone policy file) the operation is ineffective.	***STORE_NVICISER(UART_IRQn);***
STORE_NVICICER	***STORE_NVICISER(uint32_t irq_num);*** Secure access to the privileged interrupt controller (NVIC) kernel driver to **disable** an interrupt registered with the zone. In case the interrupt number is not registered with the zone (by assigning the device to the zone via Multizone policy file) the operation is ineffective.	***STORE_NVICISER(UART_IRQn);***

LOAD_MPURBAR	***uint32_t LOAD_MPURBAR(uint32_t mpu_reg);***
	
	Returns MPU_RBAR to a variable in a zone. MPU_RBAR is a privileged system register normally only available in privileged thread/handler mode.
		for(int i=0; i<mpu_regions; i++)
		...{
		......mpu_rbar[i] = LOAD_MPURBAR(i);
		...}

LOAD_MPURASR	***uint32_t LOAD_MPURASR(uint32_t mpu_reg);***
	
	Returns MPU_RBAR to a variable in a zone. MPU_RBAR is a privileged system register normally only available in privileged thread/handler mode.
		for(int i=0; i<mpu_regions; i++)
		...{
		......mpu_rsar[i] = LOAD_MPURBAR(i);
		...}

Appendix IV - MultiZone Toolchain Extension

The MultiZone Toolchain Extension utility (multizone.jar) is invoked by the make script and provided as a portable jar file - Java 1.8 or newer required.

```
Usage: java -jar multizone.jar [OPTION...] file.elf/hex ... [-o file.hex]
Hex Five MultiZone(TM) Configurator

 -c, --config=file.cfg      Config file. Default: multizone.cfg
 -o, --output=file.hex      Output file. Default: multizone.hex
 -a, --arch={IMX8QM|(...)}  Architecture. Default: IMX8QM
 -q, --quiet                Don't produce any output
 -?, --help                 Give this help list
 -V, --version              Print version info

Example: java -jar multizone.jar zone1.elf zone2.elf zone3.elf -o multizone.hex
```

References

[1] NXP, “Quick Start Guide Multisensory Enablement Kit i.MX 8QuadMax MEK CPU Board”, IMX8QUADMAXQSG, Rev. 3, 2019.

[2] BlackBerry QNX, “QNX Technical Articles: QNX Software Center 1.6 Installation Note”, March, 2020. [Online]. Available: http://www.qnx.com/developers/articles/inst_6738_3.html [Accessed Apr. 22, 2020].

[3] BlackBerry QNX, “myQNX License Manager and QNX Software Center User's Guide”, Feb. 24, 2020.

[4] BlackBerry QNX, “BSP User's Guide NXP i.MX 8 QuadMax CPU Card Board and NXP i.MX 8 QuadMax Multisensory Enablement Kit”, Nov. 12, 2019.

[5] NXP Community, “i.MX8 Boot process and creating a bootable image”, Apr. 18, 2019. [Online]. Available: https://community.nxp.com/docs/DOC-343178 [Accessed Apr. 23, 2020].

[6] NXP. “System Controller Firmware API Reference Guide i.MX8 QM Die”, version 1.5, Oct, 2019.

[7] https://www.segger.com/products/debug-probes/j-link/models/j-link-edu-mini/