Working with Cortex-M on i.MX8M

Embedded Artists AB
Rundelsgatan 14
SE-211 36 Malmö
Sweden

http://www.EmbeddedArtists.com

Table of Contents

1 Document Revision History

Revision	***Date***	***Description***
A	2021-01-12	First release

2 Introduction

This document provides you with step-by-step instructions for how to work with the Cortex-M microcontroller on the iMX8M (u)COM boards.

2.1 Multi-Core

The i.MX8M processor is utilizing heterogeneous multiprocessing, that is, using different kinds of cores. When developing an application that will utilize both these cores there are a number of things you need to be aware of.

- Both the Cortex-A cores and the Cortex-M core might have access to peripheral blocks in the processor. For your application, you have to decide which core that is responsible for a peripheral. This decision can affect, for example, the device tree file used by Linux when initializing device drivers.
  - In the instructions a specific device tree file will be used that disable some peripherals conflicting with Cortex-M.
- Cortex-A is always the primary core that is the first to boot and responsible for starting Cortex-M. This is done by the u-boot in our examples.
- There are ways to communicate between the cores. Chapter 5 describes how to run an application that utilizes Multi-Core Communication.

2.2 Additional Documentation

Additional recommended documentation:

- *Getting Started with iMX Developer's Kit V2* – shows you how to get started with the hardware.

2.3 Conventions

A number of conventions have been used throughout to help the reader better understand the content of the document.

`Constant width text` – is used for file system paths and command, utility and tool names.

```
$ This field illustrates user input in a terminal running on the
development workstation, i.e., on the workstation where you edit,
configure and build Linux
```

```
# This field illustrates user input on the target hardware, i.e.,
input given to the terminal attached to the COM Board
```

```
This field is used to illustrate example code or excerpt from a
document.
```

This field is used to highlight important information

3 Hardware related

3.1 UART interfaces on COM Carrier board version 2

The COM Carrier board version 2 has a dual channel UART-to-USB bridge, meaning that you will get two UART interfaces via one USB cable connected between the micro-B USB connector (J16) on the carrier board and your PC.

There are jumpers on the carrier board that lets you select which UART interface that is connected to the UART-to-USB bridge, see Figure 1. Jumpers J19/J20 let you select between using UART-A or UART-C as console for the Cortex-A side. By default, these jumpers select the UART-A interface, that is, jumpers are in upper position. This is the position they should have for the iMX8M (u)COM boards.

Jumpers J17/18 lets you select between using UART-B or UART-C as console for the Cortex-M side. By default, these jumpers are not inserted, but they **should be in upper position** for the iMX8M Quad COM board and in the **lower position** for the iMX8M Mini uCOM and iMX8M Nano uCOM boards.

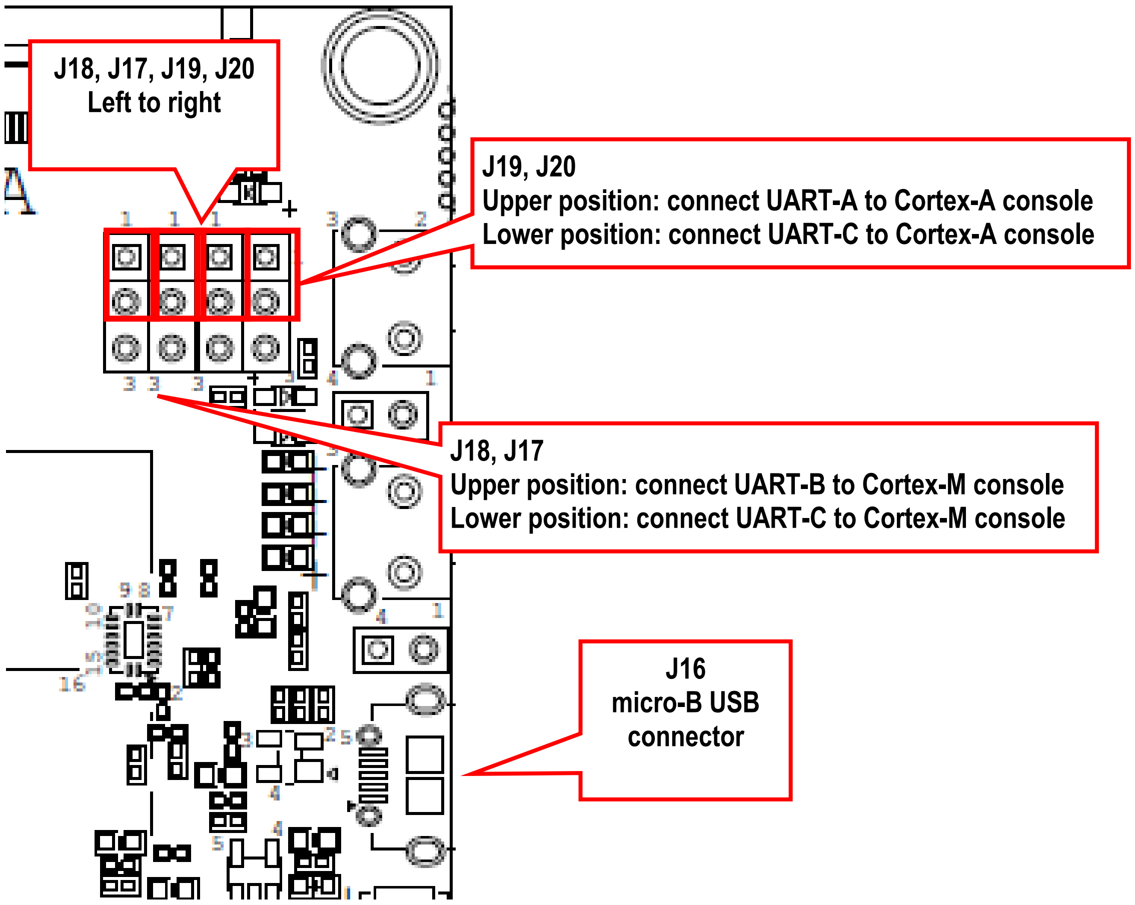

J18, J17, J19, J20
Left to right
J19, J20
Upper position: connect UART-A to Cortex-A console
Lower position: connect UART-C to Cortex-A console
J18, J17
Upper position: connect UART-B to Cortex-M console
Lower position: connect UART-C to Cortex-M console
J16
micro-B USB
connector

Figure 1 - COM Carrier Board V2, UART interface connectors

3.2 Terminal application

You need a terminal application (two instances of it to connect both to the Cortex-A side and the Cortex-M side). We recommend Tera Term, but you can use the terminal application of your choice. Connect to the virtual COM ports using 115200 as baud rate, 8 data bits, 1 stop bit, and no parity.

4 Download and start an application

This section describes how to download and start a pre-compiled application.

4.1 Update boot partition with needed files

The remaining parts of this chapter assumes that the first partition of the eMMC contains the pre-compiled applications. If you have programmed your board using a UUU bundle from 2020-11-04 or later the files will already have been copied to the eMMC flash. If you have programmed using an older version and don't want to update you can follow these instructions.

Note: It is not necessary to have the Cortex-M applications on the eMMC, but for simplicity the following instructions in this chapter assumes they are.

Download pre-compiled applications

Go to http://imx.embeddedartists.com and download the file `compiled_cortex_m_apps.zip` for your (u)COM board.

Copy via USB memory stick

There are several ways to copy these pre-compiled files to the eMMC, but here we will use a USB memory stick.

1. Unpack the file `compiled_cortex_m_apps.zip` file and copy the unpacked files to the USB memory stick. This is something you do on your computer.
2. Boot into Linux and insert the USB memory stick into the USB host port on the carrier board. You will see output like below in the console when inserting the USB memory stick. The most important part is the last line that lists the device name (**`sda1`**).

```
[   23.104504] usb 1-1.2: new high-speed USB device number 4 using ci_hdrc
[   23.165591] usb 1-1.2: New USB device found, idVendor=0781, idProduct=5406,
bcdDevice= 0.10
[   23.173972] usb 1-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[   23.194511] usb 1-1.2: Product: U3 Cruzer Micro
[   23.199055] usb 1-1.2: Manufacturer: SanDisk Corporation
[   23.204371] usb 1-1.2: SerialNumber: 0000185A49619848
[   23.225447] usb-storage 1-1.2:1.0: USB Mass Storage device detected
[   23.264533] scsi host0: usb-storage 1-1.2:1.0
[   24.315418] scsi 0:0:0:0: Direct-Access     SanDisk  U3 Cruzer Micro  2.18 PQ:
0 ANSI: 2
[   24.334542] scsi 0:0:0:1: CD-ROM            SanDisk  U3 Cruzer Micro  2.18 PQ:
0 ANSI: 2
[   24.345768] sd 0:0:0:0: [sda] 8015505 512-byte logical blocks: (4.10 GB/3.82
GiB)
[   24.364543] sd 0:0:0:0: [sda] Write Protect is off
[   24.373248] sd 0:0:0:0: [sda] No Caching mode page found
[   24.378630] sd 0:0:0:0: [sda] Assuming drive cache: write through
[   24.443649]  sda: sda1
```

3. Mount the USB memory stick and eMMC partition. The USB memory stick has in this example the device name **`sda1`** as can be seen in the output in step 2 above. The partition on the eMMC that we will use is available at **`/dev/mmcblk2p1`**, but this can be different for different (u)COM boards.

```
# mkdir /mnt/usb
# mount /dev/sda1 /mnt/usb
# mkdir /mnt/mmcboot
# mount /dev/mmcblk2p1 /mnt/mmcboot
```

4. Copy the bin file(s) from the USB memory stick to the boot partition. In this example we are only copying `cm_TCM_hello_world.bin`.

```
# cp /mnt/usb/cm_TCM_hello_world.bin /mnt/mmcboot/
```

5. Unmount the devices

```
# umount /mnt/usb
# umount /mnt/mmcboot
```

4.2 Change the device tree file

Some of the u-boot environment variables need to be updated.

1. You must have booted into the U-boot console.
2. Change which device tree file to use by Linux. The example below sets the file to use for the iMX8M Mini uCOM Developer's Kit V2. If you are using a different board just use the same name as set by default in the `fdt_file` variable and append '`-m4`'.

```
=> setenv fdt_file imx8mmea-ucom-kit_v2-m4.dtb
=> saveenv
```

4.3 Run from TCM

Make sure you have built an application for TCM or selected a pre-built application for TCM. The application file must have been copied to eMMC as described in section 4.1

Note: All of the U-boot variables created below (starting with `cm_`) might already be available as default variables depending on which U-boot version you are running. You can check if they are available by printing the environment with `printenv`.

1. You must have booted into the U-boot console.
2. Set the file name in the `cm_image` variable.

```
=> setenv cm_image cm_TCM_hello_world.bin
```

3. Set the address where the application will run from (TCM memory in this case).

```
=> setenv cm_addr 0x7e0000
```

4. Create a variable that loads the application from eMMC

```
=> setenv cm_loadimage 'fatload mmc ${mmcdev} ${loadaddr}
${cm_image}'
```

5. Create a variable that will load and boot the application. It loads the image from eMMC to DDR memory, copies from DDR memory to TCM memory and then boots the application. The application is loaded via DDR memory since `fatload` in U-boot can only load to memory regions that aren't marked as reserved and the TCM memory is considered reserved.

```
=> setenv cm_boot 'run cm_loadimage; cp.b ${loadaddr} ${cm_addr}
${filesize}; dcache flush; bootaux ${cm_addr}'
```

6. Save the changes.

```
=> saveenv
```

7. Boot the Cortex-M application.

```
=> run cm_boot
```

4.4 Automatically start the Cortex-M application

If you want the Cortex-M application to start during boot of the board follow the instructions below.

1. You must have booted into the U-boot console.
2. You should also have followed the instructions in any of the sections above depending on which type of application you use.
3. Change the `bootcmd` variable to first run `cm_boot` and then boot Linux.

```
=> setenv bootcmd "run cm_boot; ${bootcmd}"
=> saveenv
```

5 Remote communication applications (RPMsg)

5.1 Ping-pong application

The RPMsg ping-pong application is an example of communication between the Cortex-A core and the Cortex-M core using the RPMsg API.

1. Make sure the `cm_TCM_rpmsg_lite_pingpong_rtos_linux_remote.bin` file is available on eMMC as described in section 4.1 above.
2. Follow the instruction in section 4.3 for how to run an application from TCM memory, but use the file name **`cm_TCM_rpmsg_lite_pingpong_rtos_linux_remote.bin.`**
3. Boot the Cortex-M application

```
=> run cm_boot
```

4. In the console for the Cortex-M you will now see the output below

```
RPMSG Ping-Pong FreeRTOS RTOS API Demo...
RPMSG Share Base Addr is 0xb8000000
```

5. In the console for Cortex-A boot into Linux

```
=> boot
```

6. When Linux has booted you need to load the rpmsg pingpong module.

```
# modprobe imx_rpmsg_pingpong
```

7. You will now see messages in both consoles / terminals.

5.2 TTY application

The RPMsg TTY application is an example of communication between the Cortex-A core and the Cortex-M core using the RPMsg API. A TTY channel will be setup making it possible to send messages from the Cortex-A core to the Cortex-M core from Linux user-space.

1. Make sure the `cm_TCM_rpmsg_lite_str_echo_rtos.bin` file is available on eMMC as described in section 4.1 above.
2. Follow the instruction in section 4.3 for how to run an application from TCM memory, but use the file name **`cm_TCM_rpmsg_lite_str_echo_rtos.bin`**.
3. Boot the Cortex-M application

```
=> run cm_boot
```

4. In the console for the Cortex-M you will now see the output below

```
RPMSG String Echo FreeRTOS RTOS API Demo...
```

5. In the console for Cortex-A boot into Linux

```
=> boot
```

6. When Linux has booted you need to load the rpmsg tty module. You will see a confirmation that a channel has been created (similar as below).

```
# modprobe imx_rpmsg_tty
[   17.860278] imx_rpmsg_tty virtio0.rpmsg-virtual-tty-channel-1.-1.30: new
channel: 0x400 -> 0x1e!
[   17.870207] Install rpmsg tty driver!
```

7. From Linux you can now send a message to the Cortex-M side by using the new TTY channel (`dev/ttyRPMSG0`)

```
# echo hello > /dev/ttyRPMSG30
```

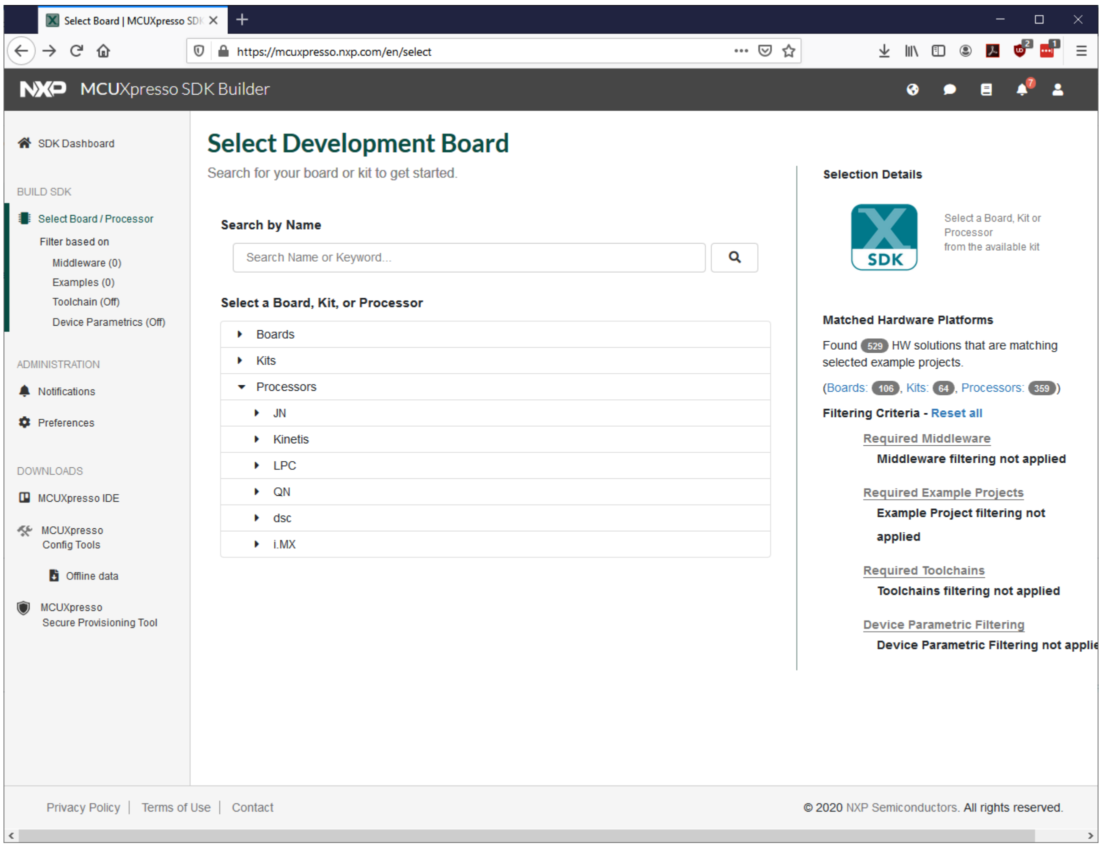
6 Software Development Kit (SDK)

NXP provides SDK's for the iMX8M family of processors. These can be downloaded from NXP's MCUXpresso website.

6.1 Download SDK

Follow these instructions to download an SDK from NXP's website.

1. Go the https://mcuxpresso.nxp.com (you must have an account to create and download an SDK).
2. Click on "Select Development Board" and you will get to the MCUXpresso SDK Builder as shown Figure 2.
3. Choose the i.MX processor you are using. In Figure 3 the i.MX 8M Mini is selected.
4. Click "Build MCUXpresso SDK". This button is shown to the right in Figure 3.
5. When the SDK is ready select "Host OS" (we use Linux in this example) and then click on "Download SDK" as shown in Figure 4.


Select Board | MCUXpresso SDK
https://mcuxpresso.nxp.com/en/select
MCUXpresso SDK Builder

SDK Dashboard
BUILD SDK
Select Board / Processor
Filter based on
Middleware (0)
Examples (0)
Toolchain (Off)
Device Parametrics (Off)
ADMINISTRATION
Notifications
Preferences
DOWNLOADS
MCUXpresso IDE
MCUXpresso Config Tools
Offline data
MCUXpresso Secure Provisioning Tool

Select Development Board
Search for your board or kit to get started.

Search by Name
Search Name or Keyword...

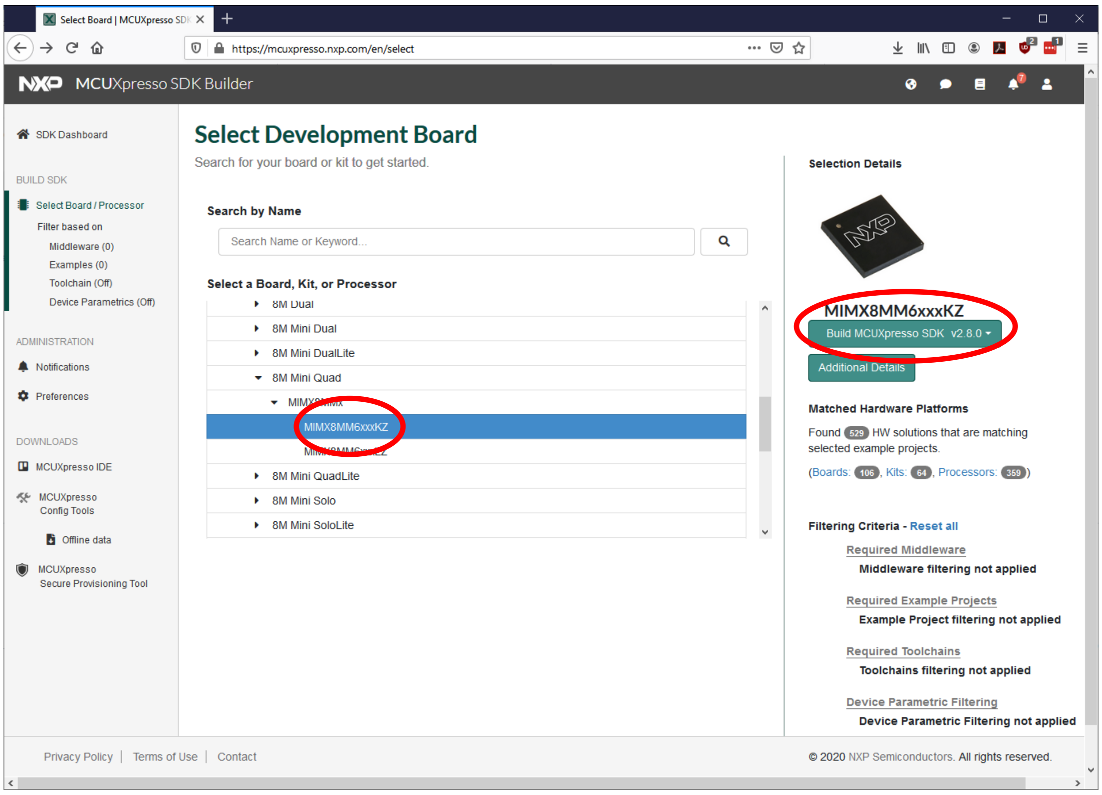
Select a Board, Kit, or Processor
- Boards
- Kits
- Processors
  - JN
  - Kinetis
  - LPC
  - QN
  - dsc
  - i.MX

Selection Details
Select a Board, Kit or Processor from the available kit

Matched Hardware Platforms
Found 529 HW solutions that are matching selected example projects.
(Boards: 106, Kits: 64, Processors: 359)
Filtering Criteria - Reset all
Required Middleware
Middleware filtering not applied
Required Example Projects
Example Project filtering not applied
Required Toolchains
Toolchains filtering not applied
Device Parametric Filtering
Device Parametric Filtering not applie

Privacy Policy | Terms of Use | Contact
© 2020 NXP Semiconductors. All rights reserved.

Figure 2 - MCUXpresso SDK Builder


Select Board | MCUXpresso SDK

https://mcuxpresso.nxp.com/en/select

MCUXpresso SDK Builder

SDK Dashboard
BUILD SDK
Select Board / Processor
Filter based on
Middleware (0)
Examples (0)
Toolchain (Off)
Device Parametrics (Off)
ADMINISTRATION
Notifications
Preferences
DOWNLOADS
MCUXpresso IDE
MCUXpresso Config Tools
Offline data
MCUXpresso Secure Provisioning Tool

Select Development Board

Search for your board or kit to get started.

Search by Name

Search Name or Keyword...

Select a Board, Kit, or Processor

- 8M Dual
- 8M Mini Dual
- 8M Mini DualLite
- 8M Mini Quad
  - MIMX8MM6
    - MIMX8MM6xxxKZ
    - MIMX8MM6xxxLZ
- 8M Mini QuadLite
- 8M Mini Solo
- 8M Mini SoloLite

Selection Details

MIMX8MM6xxxKZ

Build MCUXpresso SDK v2.8.0

Additional Details

Matched Hardware Platforms

Found 529 HW solutions that are matching selected example projects.

(Boards: 106, Kits: 64, Processors: 359)

Filtering Criteria - Reset all

Required Middleware
Middleware filtering not applied

Required Example Projects
Example Project filtering not applied

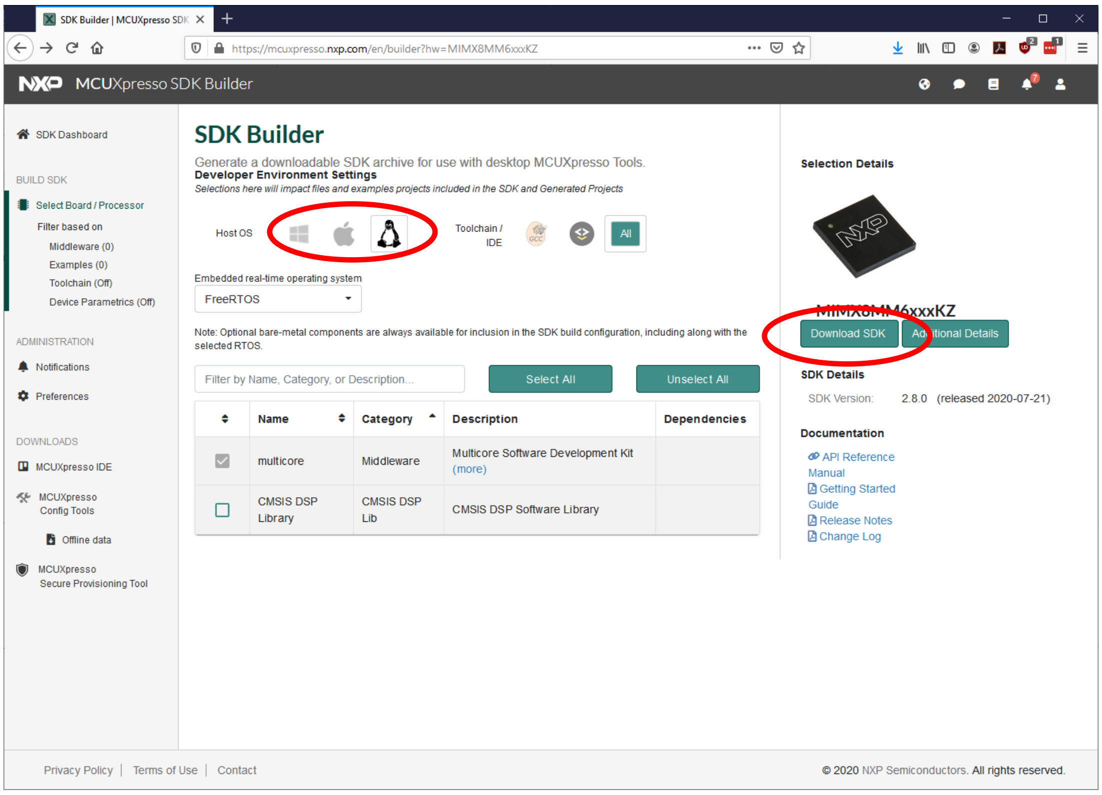
Required Toolchains
Toolchains filtering not applied

Device Parametric Filtering
Device Parametric Filtering not applied

Privacy Policy | Terms of Use | Contact

© 2020 NXP Semiconductors. All rights reserved.

Figure 3 - Select processor (iMX8M Mini in this example)


SDK Builder | MCUXpresso SDK

https://mcuxpresso.nxp.com/en/builder?hw=MIMX8MM6xxxKZ

MCUXpresso SDK Builder

SDK Dashboard
BUILD SDK
Select Board / Processor
Filter based on
Middleware (0)
Examples (0)
Toolchain (Off)
Device Parametrics (Off)
ADMINISTRATION
Notifications
Preferences
DOWNLOADS
MCUXpresso IDE
MCUXpresso Config Tools
Offline data
MCUXpresso Secure Provisioning Tool

SDK Builder

Generate a downloadable SDK archive for use with desktop MCUXpresso Tools.

Developer Environment Settings

Selections here will impact files and examples projects included in the SDK and Generated Projects

Host OS

Toolchain / IDE All

Embedded real-time operating system

FreeRTOS

Note: Optional bare-metal components are always available for inclusion in the SDK build configuration, including along with the selected RTOS.

Filter by Name, Category, or Description...

Select All Unselect All

	Name	Category	Description	Dependencies
☑	multicore	Middleware	Multicore Software Development Kit (more)	
☐	CMSIS DSP Library	CMSIS DSP Lib	CMSIS DSP Software Library	

Selection Details

MIMX8MM6xxxKZ

Download SDK Additional Details

SDK Details

SDK Version: 2.8.0 (released 2020-07-21)

Documentation

API Reference Manual
Getting Started Guide
Release Notes
Change Log

Privacy Policy | Terms of Use | Contact

© 2020 NXP Semiconductors. All rights reserved.

Figure 4 - Download SDK

6.2 Setup a Linux host

These instructions are tested on an Ubuntu 18.04 Linux host. You can also run on other Linux hosts or on a Windows host. The SDK contains a “Getting Started” document with more instructions. This document can be found in the `docs` folder.

1. Install toolchain

```
$ sudo apt-get install gcc-arm-none-eabi binutils-arm-none-eabi
```

2. Install `cmake` and `make`

```
sudo apt-get install cmake make
```

6.3 Build on Linux

Once the toolchain has been installed you can unpack the SDK and build an application.

1. Unpack the downloaded SDK (replace <download dir> with the directory where you downloaded the SDK.

```
$ cd ~
$ mkdir 8mm_sdk
$ cp <download dir>/SDK_2.8.0_MIMX8MM6xxxKZ.tar.gz 8mm_sdk
$ cd 8mm_sdk
$ tar -xzvf SDK_2.8.0_MIMX8MM6xxxKZ.tar.gz$
```

2. The build scripts expect there to be an environment variable called `ARMGCC_DIR` pointing to the toolchain directory.

```
$ export ARMGCC_DIR=/usr
```

3. Go the armgcc directory of the application you want to build.

```
$ cd boards/evkmimx8mm/rtos_examples/freertos_hello/armgcc
```

4. Run the `build_debug.sh` script to build the debug version of the RAM/TCM target.

```
$ ./build_debug.sh
```

5. Now the application will be available in the `debug` directory.

```
$ ls debug
freertos_hello.bin  freertos_hello.elf
```

6. You can now copy and run `freertos_hello.bin` on target as described in chapter 4 above.