AN13706

Partitioning the Storage Medium on the i.MX 8M Family

Rev. 0 — 23 August 2022 Application note

Document information

Information	Content
Keywords	AN13706, i.MX 8M, Storage partitions, UUU, U-Boot
Abstract	This application note provides details about how to partition and resize the storage medium on the i.MX 8M family at the image creation and deployment stages.

1 Introduction

This application note provides details about how to partition and resize the storage medium on the i.MX 8M family at the image creation and deployment stages.

This document explains the following two objectives:

- How to partition the storage space using pre-built images (downloaded from the NXP website)
- How to partition the storage space using images resulted from the yocto project build environment

1.1 Software environment

Linux BSP release 5.15.32_2.0.0, Embedded Linux for i.MX Applications Processors (document IMXLINUX) is used throughout the document.

Note: *The same BSP release is used for the yocto project build.*

The Universal Update Utility (UUU) version 1.4.193 is used to deploy the images to the board.

1.2 Hardware environment

The information described in this document applies to the i.MX 8M family development boards. The supplied scripts are for the i.MX 8MQ development boards, but they can be easily ported to other boards from the i.MX 8M family.

2 Overview of image deployment

To begin partitioning the storage space on the board, the necessary files must be obtained. There are two main methods of getting a deployable image as explained below:

- The first method involves downloading the Linux BSP pre-built binaries, which offer a starting point for testing the features supported on the evaluation boards. However, since they are pre-built, there is no easy way to add/remove functionalities included in the supplied images without a building process for that specific component.
- This limitation brings us to the second method: building the image itself. This method offers more flexibility and control over the generated image as features can be added or removed as desired via the yocto project build system.

Since pre-built binaries can avoid the building stage, partitioning methodologies differ primarily at the deployment stage, depending on the intermediary used to handle commands (Linux/U-Boot). The standard deployment image for the i.MX 8M Linux BSP includes the following major components:

- **Bootloader image**
  - It does not represent a partition in itself
  - It has a specific start address in relation to the board, as well as a boot mode (normal/fast). For more information, see the reference manuals of the chip listed in Section 6.
- **Boot partition**
  - FAT32 partition containing the kernel image, dtbs, and Cortex-M4 demos

  - It is a bootable partition
- **Rootfs partition**
  - EXT type partition containing the root filesystem.

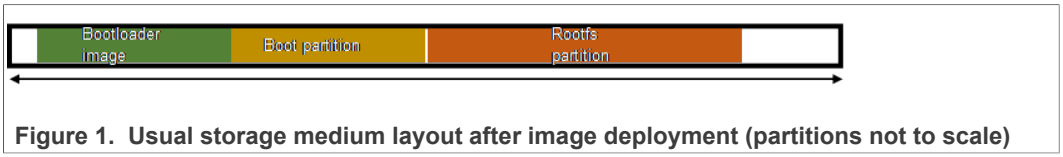

Figure 1. Usual storage medium layout after image deployment (partitions not to scale)

Note: *Boot and Rootfs partitions can be resized and their start address can be changed. However, care must be taken when doing this change as they cannot overlap with the bootloader location. Also, ensure that their sizes must not be lower than their respective images. The standard partition table type is MBR for Linux BSPs.*

3 Partitioning for pre-built binaries

The release package for the Linux BSP contains the following components:

- SD/EMMC prebuilt image for the release target SoC
- Kernel and device tree binaries
- Boot images
- Applicable Arm Cortex-M4 demos if applicable to target SoC
- UUU default and example scripts.

For the exact contents, refer to the associated *i.MX Linux Release Notes* (document IMXLXRN).

The UUU tool is used for deployment. This tool provides an easy way of writing images to the i.MX 8M family boards using either the built-in scripts or a user provided command file. The tool can be run on both Windows and Linux.

UUU uses the fast-boot protocol to issue commands and transfer files to the board either in the bootloader or Linux environment. The FB tag precedes U-Boot commands, while the FBK tag precedes Linux commands. For more information about the tool, refer to the *UUU readme* (document UUU (Universal Update Utility)).

The `example_kernel_emmc` file represents the starting point for the current script, available in the samples folder in the release package for the specific version of Linux BSP. The placeholder names for the specific components must be replaced with the desired ones. The script adapted for writing on the EMMC on the i.MX 8MQ development board is as follows, with a few additions which are explained later:

```
uuu_version 1.2.39
# This command runs when i.MX6/7 i.MX8MM, i.MX8MQ are used
SDP: boot -f imx-boot-imx8mqevk-sd.bin-flash_evk
# This command runs when ROM support stream mode
# i.MX8QXP, i.MX8QM
SDPS: boot -f imx-boot-imx8mqevk-sd.bin-flash_evk
# These commands run when SPL is used and is skipped if there is no SPL
# SDPU is deprecated, therefore use SDPV instead of SDPU
# {
SDPU: delay 1000
SDPU: write -f imx-boot-imx8mqevk-sd.bin-flash_evk -offset 0x57c00
SDPU: jump
# }
# These commands run when SPL is used and is skipped if there is no SPL
# if (SPL support SDPV)
# {
SDPV: delay 1000
SDPV: write -f imx-boot-imx8mqevk-sd.bin-flash_evk -skipspl
SDPV: jump
# }
FB: ucmd setenv fastboot_buffer ${loadaddr}
FB: download -f Image-imx8mqevk.bin
FB: ucmd setenv fastboot_buffer ${fdt_addr}
FB: download -f imx8mq-evk.dtb
FB: ucmd setenv fastboot_buffer ${initrd_addr}
FB: download -f fsl-image-mfgtool-initramfs-imx_mfgtools.cpio.zst.u-boot
#FB: ucmd setenv bootargs console=${console},${baudrate} earlycon=${earlycon},${baudrate}
FB: acmd ${kboot} ${loadaddr} ${initrd_addr} ${fdt_addr}
# Get mmc dev number from kernel command line
# Wait for emmc
FBK: ucmd while [ ! -e /dev/mmcblk*boot0 ]; do sleep 1; echo "wait for /dev/mmcblk*boot*
 appear"; done;
# Search emmc device number, if your platform have more than two emmc chip, echo dev number
 >/tmp/mmcdev
FBK: ucmd dev=`ls /dev/mmcblk*boot*`; dev=($dev); dev=${dev[0]}; dev=${dev#/dev/mmcblk};
 dev=${dev%boot*}; echo $dev > /tmp/mmcdev;
# dd to clear the possible MBR
FBK: ucmd mmc=`cat /tmp/mmcdev`; dd if=/dev/zero of=/dev/mmcblk${mmc} bs=512 count=1
# Create partition
FBK: ucmd mmc=`cat /tmp/mmcdev`; PARTSTR=$'10M,500M,0c\n600M,,83\n'; echo "$PARTSTR" |
 sfdisk --force /dev/mmcblk${mmc}
FBK: ucmd mmc=`cat /tmp/mmcdev`; dd if=/dev/zero of=/dev/mmcblk${mmc} bs=1k seek=4096
 count=1
FBK: ucmd sync
# Enable below command to write boot partition but offset is different at difference
 platform
FBK: ucmd mmc=`cat /tmp/mmcdev`; echo 0 > /sys/block/mmcblk${mmc}boot0/force_ro
FBK: ucp imx-boot-imx8mqevk-sd.bin-flash_evk t:/tmp
FBK: ucmd mmc=`cat /tmp/mmcdev`; dd if=/tmp/imx-boot-imx8mqevk-sd.bin-flash_evk of=/dev/mmc
${mmc}boot0 bs=1K seek=33
FBK: ucmd mmc=`cat /tmp/mmcdev`; echo 1 > /sys/block/mmcblk${mmc}boot0/force_ro
FBK: ucmd mmc=`cat /tmp/mmcdev`; while [ ! -e /dev/mmcblk${mmc}p1 ]; do sleep 1; done
FBK: ucmd mmc=`cat /tmp/mmcdev`; mkfs.vfat /dev/mmcblk${mmc}p1
FBK: ucmd mmc=`cat /tmp/mmcdev`; mkdir -p /mnt/fat
FBK: ucmd mmc=`cat /tmp/mmcdev`; mount -t vfat /dev/mmcblk${mmc}p1 /mnt/fat
FBK: ucp Image-imx8mqevk.bin t:/mnt/fat
FBK: ucmd mmc=`cat /tmp/mmcdev`; mv /mnt/fat/Image-imx8mqevk.bin /mnt/fat/Image
FBK: ucp imx8mq-evk.dtb t:/mnt/fat
FBK: ucmd umount /mnt/fat
FBK: ucmd mmc=`cat /tmp/mmcdev`; mkfs.ext3 -F -E nodiscard /dev/mmcblk${mmc}p2
FBK: ucmd mkdir -p /mnt/ext3
FBK: ucmd mmc=`cat /tmp/mmcdev`; mount /dev/mmcblk${mmc}p2 /mnt/ext3
FBK: acmd export EXTRACT_UNSAFE_SYMLINKS=1; tar -jx -C /mnt/ext3
FBK: ucp imx-image-full-imx8mqevk.tar.bz2 t:-
FBK: Sync
FBK: ucmd umount /mnt/ext3
FBK: DONE
```

This script loads the bootloader, followed by Linux, into the RAM of the board. Using Linux to apply the modifications is easier since the commands are documented and have larger community support.

The partition table layout is specified in the `PARTSTR` variable, which is used by the `sfdisk` tool in order to create the partitions. In the supplied script two partitions are generated as follows:

- First (Boot partition): Start offset - 10 MB, Size - 500 MB, Type 0x0c, bootable - FAT32 filesystem

- Second (Rootfs partition): Start offset - 600 MB, Size - remaining disk space from start address onward, Type 0x83 - Linux filesystem

The next step is to write the bootloader at the address determined by the storage medium, boot type, chip type, and revision. In this example, the bootloader is written on the EMMC, with the chip following a normal boot procedure, leading to the 33 kB starting address. Refer to the associated *i.MX 8MQ Applications Processor Reference Manual* (document IMX8MDQLQRM).

Note: *The starting offset for the bootloader can differ based on platform and revision – check the used chip associated reference manual listed in Section 6.*

After the bootloader is written, there is a sequence of straight-forward operations left: the mounting of the partitions, creation of necessary filesystems, downloading of the associated data and unmounting.

Note: *When downloading the kernel image and device tree binary, take care that their names match the ones expected by the bootloader. The solution is either renaming them before/after writing or changing the associated U-Boot environment variables to match the names of the files at the first boot. If the names differ, the board most likely gets stuck at the bootloader stage, because it cannot find the necessary components to continue booting.*

4 Partitioning for binaries obtained from a yocto build

Additional features are easily added when building the image with the yocto project environment, resulting in final packages containing the modifications according to the supplied instructions. The yocto build enables additional methods of configuring the partitioning scheme when building and deploying the image.

4.1 Partitioning at the image creation stage

The yocto build system partitioning consists of two methods for modifying the sizing during the image creation stage, depending on the partition involved:

- Sizing the Rootfs:
  The build system determines the required size for the generated Rootfs image, and then adds additional space based on that size and some other parameters. For more detailed information on this algorithm, refer to the The Yocto Project version 4.0.999 reference manual. As stated in the manual, the following variables are in play:
  - `IMAGE_ROOTFS_SIZE` defines the size in kB for the generated image.
  - `IMAGE_OVERHEAD_FACTOR` defines a multiplier that the build system applies to the initial image size.
  - `IMAGE_ROOTFS_EXTRA_SPACE` defines additional free disk space created in the image in kB.

  By adding these variables in the `local.conf`, the Rootfs partition can be extended accordingly.
- Placing the three elements and sizing the Boot partition:
  When creating a Wic image, the yocto build system must know what goes into the final image and where. This information is structured in the form of partitioning commands located in an `Openembedded` kickstart file (*.wks). The build system determines which *.wks file to use by inspecting the `WKS_FILE` variable. For more information regarding the *.wks files, refer to the The Yocto Project version 4.0.999 reference manual.

For example, the *.wks for the i.MX 8M family is defined in `meta-freescale/conf/machine/include/imx-base.inc` as follows:

```
SOC_DEFAULT_WKS_FILE ?= "imx-uboot-bootpart.wks.in"
SOC_DEFAULT_WKS_FILE_mx8m ?= "imx-imx-boot-bootpart.wks.in"
SOC_DEFAULT_WKS_FILE_mx8 ?= "imx-imx-boot-bootpart.wks.in"
SOC_DEFAULT_WKS_FILE_mxs ?= "imx-uboot-mxs-bootpart.wks.in"
<…>
WKS_FILE ?= "${SOC_DEFAULT_WKS_FILE}"
```

The `imx-imx-boot-bootpart.wks.in` file contains the following partitioning relevant information:

```
part u-boot --source rawcopy --sourceparams="file=imx-boot" --ondisk mmcblk --no-table --
align ${IMX_BOOT_SEEK}
part /boot --source bootimg-partition --ondisk mmcblk --fstype=vfat --label boot --active
 --align 8192 --size 64
part / --source rootfs --ondisk mmcblk --fstype=ext4 --label root --align 8192
bootloader --ptable msdos
```

- The size for the `/boot` partition can be set using the `--size` parameter, specified in MB.
- Other parameters like filesystem and labels can also be set here.
- For Rootfs, specify the size using `IMAGE_ROOTFS_SIZE`, `IMAGE_OVERHEAD_FACTOR`, and `IMAGE_ROOTFS_EXTRA_SPACE`.

The individual partitions and full Wic image can be found in the work directory for the `imx-image-(imagetype)` package, where `imagetype` corresponds to the chosen image type (multimedia/full/core). The Wic can then be written normally using UUU, with no additional script required.

Note: *UUU version 1.4.165 or later can be used to write entire Wic images.*

4.2 Partitioning at the image deployment stage

This method is similar to the Section 3, but requires usage of the building process in order to activate the required features.

The partitions can be listed using the `MMC part` or `GPT read` command in U-Boot for the specified storage medium. However, in order to write the MBR type partition table, the MBR tool in U-Boot must be included by enabling the MBR feature. Use `make menuconfig` in the build folder, after issuing the following command:

```
bitbake -f -c configure u-boot-imx
```

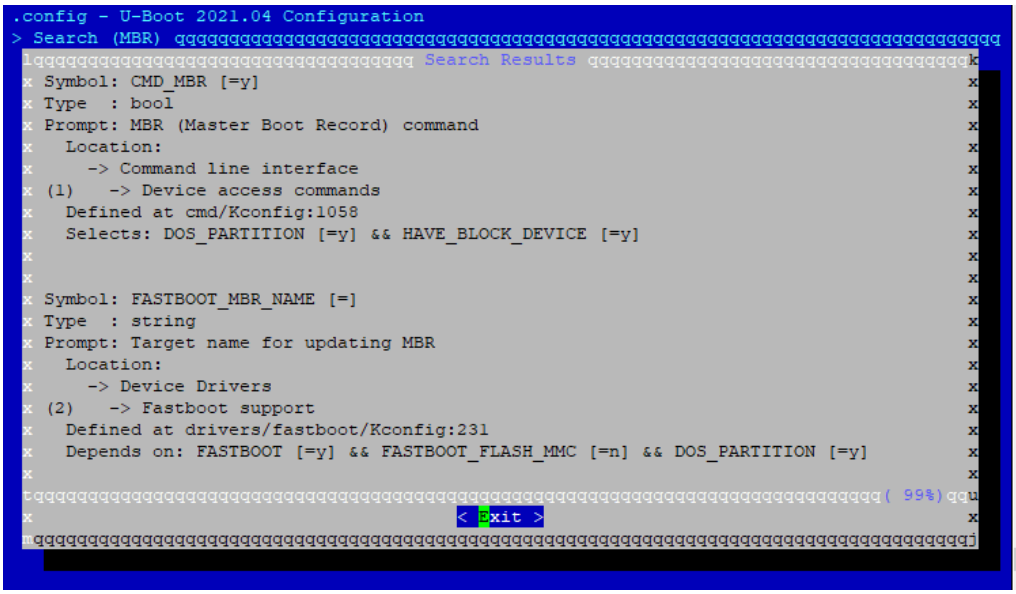

Figure 2. MBR symbols in U-Boot menuconfig tool

After enabling the MBR support, the bootloader image is generated. It is used to write directly from U-Boot the MBR partition table, followed by the chosen `imx-image` type build process.

The images for the Boot and Rootfs partitions are extracted from the resulting `imx-image-(imagetype).wic` file found in the `deploy` folder where `imagetype` is the chosen image type (multimedia/full/core).

The partition images are labeled as follows:

- `0.fat` for Boot partition
- `1.img` for Rootfs partition

The partitions are also found in the `imx-image-(imagetype)` build folder, under the following name:

```
imx-imx-boot-bootpart-(build_date)-mmcblk.direct.(partition_id)
```

The `partition_id` identifies the partitions according to the list described in the previous chapters (`imx-imx-boot-bootpart.wks.in`).

The partitioning is done from U-Boot rather than Linux, therefore a separate script must be used for this method. Similar to the pre-built image, the names of the binary must be replaced with the desired ones. The script adapted for the i.MX 8MQ platform is as follows:

```
uuu_version 1.2.39
# This command runs when i.MX6/7 i.MX8MM, i.MX8MQ
SDP: boot -f imx-boot-imx8mqevk-sd.bin-flash_dp_evk

# This command runs when ROM support stream mode
# i.MX8QXP, i.MX8QM
SDPS: boot -f imx-boot-imx8mqevk-sd.bin-flash_dp_evk
# These commands run when SPL is used and is skipped if no SPL is used
# SDPU is deprecated, use SDPV instead of SDPU
# {
SDPU: delay 1000
SDPU: write -f imx-boot-imx8mqevk-sd.bin-flash_evk -offset 0x57c00
SDPU: jump
# }
# These commands run when SPL is used and is skipped if no SPL is used
# if (SPL support SDPV)
# {
SDPV: delay 1000
SDPV: write -f imx-boot-imx8mqevk-sd.bin-flash_evk -skipspl
SDPV: jump
# }
FB: ucmd setenv fastboot_dev mmc
FB: ucmd setenv mmcdev ${emmc_dev}
FB: ucmd mmc dev ${emmc_dev}
FB: ucmd setenv mbr_parts
'name=boot,start=8M,size=128M,bootable,id=0x0e;name=rootfs,start=140M,size=4096M,id=0x83'
FB: ucmd mbr write mmc ${emmc_dev}
FB: flash -raw2sparse mmcsda1 0.fat
FB: flash -raw2sparse mmcsda2 1.img
FB: flash bootloader imx-boot-imx8mqevk-sd.bin-flash_evk
FB: ucmd if env exists emmc_ack; then ; else setenv emmc_ack 0; fi;
FB: ucmd mmc partconf ${emmc_dev} ${emmc_ack} 1 0
FB: done
```

The script uses only the U-Boot environment to do most of the operations required. It loads the bootloader, and generates the partition table using the `mbr write` command. It populates the Boot and Rootfs partitions, ending with the writing of the bootloader.

The partition table layout is specified in the `mbr_parts` variable, which is used by the `mbr write` command in order to create the partitions. In the supplied script two partitions are generated as follows:

- First (Boot partition): Start offset - 8 MB, Size - 128 MB, Type 0x0c, bootable - FAT32 filesystem
- Second (Rootfs partition): Start offset - 140 MB, Size - 4096 MB, Type 0x83 - Linux filesystem

Note: *For the flash `-raw2sparse` instructions the partition handles should be checked, as they differ based on the storage medium and platform used. This information is found out after writing a pre-built image by issuing the `gpt read mmc <dev>` command in U-Boot, where `<dev>` is the device to which the image is written. In the current example, `mmcsda1` and `mmcsda2` are used for the eMMC on the i.MX 8MQ EVK.*

If the partition space is checked after the script is finished, it reports the correct dimension, but the filesystem is unaware of the partition expanding. As a result, the `resize2fs` command must be used on the first boot to Linux to allow the filesystem to include the new available space. Otherwise, the space cannot be used.

5 Conclusion

This application note describes how to partition and resize the storage medium on the i.MX 8M family at the image creation and deployment stages. These objectives are achieved either through the UUU deployment tool or through the yocto build process.

6 References

The following references are available to supplement this document. Some of the documents listed below may be available only under a non-disclosure agreement (NDA). To request access to these documents, contact your local NXP field applications engineer (FAE) or sales representative:

- *i.MX 8MP Applications Processor Reference Manual* (document IMX8MPRM)
- *i.MX 8MQ Applications Processor Reference Manual* (document IMX8MDQLQRM)
- *i.MX 8MM Applications Processor Reference Manual* (document IMX8MMRM)
- *i.MX 8MN Applications Processor Reference Manual* (document IMX8MNRM)
- *UUU readme* (document UUU (Universal Update Utility))
- *Welcome to the Yocto Project Documentation* (document The Yocto Project version 4.0.999)
- *Linux 5.15.32_2.0.0, Embedded Linux for i.MX Applications Processors* (document IMXLINUX)

7 Revision history

The Table 1 lists the substantive changes done to this document since the initial release.

Table 1. Revision history

Revision number	Date	Substantive changes
Rev 0	23 August 2022	Initial release

8 Legal information

8.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

8.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

8.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 23 August 2022
Document identifier: AN13706