AN13138

Xen Deployment on Layerscape Platforms

Rev. 0 — 03/2021 Application Note

Contents

1 Introduction

The intent of this document is to provide the necessary information to hardware, software, and system engineers to run the Xen hypervisor on NXP Layerscape platforms. In this action, we consider basic scenarios, such as basic domain management, dom0less, and device passthrough.

2 Hands-on

The Xen project hypervisor can be run directly on hardware or under QEMU. In this application note, we consider only the first scenario. The following instructions provide you the location of the images and the steps that are required to boot the hypervisor and one or more domains.

The below build instructions have been tested on a x86_64 machine running Ubuntu 18.04 as a non-root user with sudo privileges.

2.1 Build

We use the Yocto Project to build the Linux kernel and Xen images, including the hypervisor binary and Xen Project toolstack and filesystems.

2.1.1 Requirements

Make sure that your build machine meets these requirements:

- 50 GB or more of free disk space
- Git version 1.8.3.1 or later
- tar version 1.27 or later
- Python version 3.4.0 or later

2.1.2 Dependencies

Install the following packages on the build host:

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential chrpath
socat cpio python python3 python3-pip python3-pexpect xz-utils debianutils iputils-ping libsqlite3-dev
```

NOTE

Packages may differ depending on the build system.

2.1.3 Download and install the Yocto Project

1. Install repo tool:

```
$ mkdir ~/bin
$ curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
```

```
$ chmod a+x ~/bin/repo
$ PATH=${PATH}:~/bin
```

2. Download Yocto metadata:

```
$ mkdir yocto-sdk
$ cd yocto-sdk
$ repo init -u https://source.codeaurora.org/external/qoriq/qoriq-components/yocto-sdk -b
<branch> #zeus and dunfell branches have been tested
$ repo sync --force-sync
```

3. Use the poky script to set the environment for the target platform:

```
$ . ./setup-env -m <target_board>
```

4. Add the following lines to the `conf/local.conf` file:

```
IMAGE_FSTYPES_append = " ext2"
DISTRO_FEATURES_append = " xen"
IMAGE_INSTALL_append = " zlib-dev xen-base"  #zeus branch
IMAGE_INSTALL_append = " zlib-dev xen-tools" #dunfell branch
```

5. Make sure that the following kernel options are enabled:

```
CONFIG_XEN_DOM0=y
CONFIG_XEN=y
CONFIG_XEN_BLKDEV_FRONTEND=y
CONFIG_XEN_NETDEV_FRONTEND=y
CONFIG_INPUT_XEN_KBDDEV_FRONTEND=y
CONFIG_HVC_XEN=y
CONFIG_HVC_XEN_FRONTEND=y
CONFIG_XEN_FBDEV_FRONTEND=y
CONFIG_XEN_BALLOON=y
CONFIG_XEN_SCRUB_PAGES_DEFAULT=y
CONFIG_XEN_DEV_EVTCHN=y
CONFIG_XEN_BACKEND=y
CONFIG_XENFS=y
CONFIG_XEN_COMPAT_XENFS=y
CONFIG_XEN_SYS_HYPERVISOR=y
CONFIG_XEN_XENBUS_FRONTEND=y
CONFIG_XEN_GNTDEV=y
CONFIG_XEN_GRANT_DEV_ALLOC=y
CONFIG_SWIOTLB_XEN=y
CONFIG_XEN_PRIVCMD=y
CONFIG_XEN_EFI=y
CONFIG_XEN_AUTO_XLATE=y
```

6. Use the following command, if you need to enable certain kernel options:

```
$ bitbake -c menuconfig virtual/kernel
```

7. Launch build:

```
$ bitbake fsl-image-networking
```

8. Find the resulted images in the following directory:

/<workdir>/yocto-sdk/build_<target_board>/tmp/deploy/images/<target_board>

```
fsl-image-networking-<target_board>.rootfs.cpio.gz        #used as dom0-rfs and domu-rfs
fsl-image-networking-<target_board>.rootfs.ext2           #used as domu-rfs in section 2.2.2
fsl-<target_board>.dtb                                    #device tree blob used for Dom0
Image                                                     #used as dom0-image and domu-image
xen-<target_board>
```

9. Image represents the kernel image used for Dom0. It can be used for unprivileged domains as well. In case you need a different kernel configuration, run the following commands to rebuild the kernel image with the desired options:

```
$ bitbake -c menuconfig virtual/kernel
$ bitbake -c compile -f virtual/kernel
```

NOTE

In case you use a newer version of GCC, such as GCC 9, you will encounter a warning of type address-of-packed-member, which leads to a build failure. Use this patch to fix it: https://patchwork.kernel.org/patch/11265041/. The Xen sources are in the following directory: /<workdir>/yocto-sdk/build_<target_board>/tmp/work/aarch64-fsl-linux/xen/<version>.

2.2 Deployment

Before performing the below steps, make sure that a TFTP server is up and running on the machine where images are built.

2.2.1 Run Xen and Dom0

Run the following instructions in U-Boot on the proper flash bank. In case you are not familiar with Layerscape platforms, see https://lsdk.github.io/document.html to determine the proper bank. The addresses used in the below instructions must be chosen so that they fit the memory layout of the platform.

1. Load the Xen image to $xen_addr address:

```
u-boot> tftp $xen_addr $path/xen-<target_board>
```

2. Load the Dom0 kernel image to $kernel_addr address:

```
u-boot> tftp $kernel_addr $path/Image
```

3. Load the Dom0 device tree blob to $fdt_addr address:

```
u-boot> tftp $fdt_addr $path/fsl-<target_board>.dtb
```

In this document, we provide instructions to use a RAM disk for Dom0 or use a filesystem stored on a mass storage device (for example, SD card). However, it is possible to use other devices for storage.

4. If you choose to use a RAM disk, run the following command to load it to $rootfs_addr address:

```
u-boot> tftp $rootfs_addr $path/dom0-rfs
```

You may also write the rootfs to an SD card.

5. Add a boot module containing the dom0 kernel:

```
u-boot> fdt addr $fdt_addr
u-boot> fdt resize 1024
u-boot> fdt set /chosen \#address-cells <0x2>
u-boot> fdt set /chosen \#size-cells <0x2>
u-boot> fdt set /chosen xen,xen-bootargs "console=dtuart dtuart=serial0 dom0_mem=2G
dom0_max_vcpus=1 bootscrub=0 vwfi=native sched=null earlycon=xen loglvl=all guest_loglvl=all"
```

```
#Xen command line
u-boot> fdt mknod /chosen dom0
u-boot> fdt set /chosen/dom0 compatible "xen,linux-zimage" "xen,multiboot-module"
u-boot> fdt set /chosen/dom0 reg <0x0 <$kernel_addr> 0x0 <$kernel_size>>
```

6. Add a boot module containing the dom0 ramdisk, if needed:

```
u-boot> fdt set /chosen xen,dom0-bootargs "console=hvc0 earlycon=xen earlyprintk=xen
clk_ignore_unused root=/dev/ram0" #Dom0 command line
u-boot> fdt mknod /chosen dom0-ramdisk
u-boot> fdt set /chosen/dom0-ramdisk compatible "xen,linux-initrd" "xen,multiboot-module"
u-boot> fdt set /chosen/dom0-ramdisk reg <0x0 <$rootfs_addr> 0x0 <$rootfs_size>>
```

7. If the filesystem is stored on the SD card, use the following dom0 command line:

```
u-boot> fdt set /chosen xen,dom0-bootargs "console=hvc0 earlycon=xen earlyprintk=xen
clk_ignore_unused root=/dev/mmcblk0p1 rw rootwait"
```

NOTE

This information can be directly added in the device tree source file.

8. Boot:

```
u-boot> booti $xen_addr - $fdt_addr
```

9. Print Xen host information:

```
# xl info
host                   : ls1046ardb
release                : 4.19.46-g4ddb5d78dd74-dirty
version                : #8 SMP PREEMPT Thu Jul 9 11:18:53 EEST 2020
machine                : aarch64
nr_cpus                : 4
max_cpu_id             : 3
nr_nodes               : 1
cores_per_socket       : 1
threads_per_core       : 1
cpu_mhz                : 25.000
hw_caps                : 00000000:00000000:00000000:00000000:00000000:00000000:00000000:00000000
virt_caps              :
total_memory           : 8126
free_memory            : 186
sharing_freed_memory   : 0
sharing_used_memory    : 0
outstanding_claims     : 0
free_cpus              : 0
xen_major              : 4
xen_minor              : 12
xen_extra              : .0
xen_version            : 4.12.0
xen_caps               : xen-3.0-aarch64 xen-3.0-armv7l
xen_scheduler          : credit2
xen_pagesize           : 4096
platform_params        : virt_start=0x200000
xen_changeset          :
xen_commandline        : console=dtuart dtuart=serial0 dom0_mem=7500M loglvl=all
guest_loglvl=all sync_console
cc_compiler            : aarch64-fsl-linux-gcc (GCC) 9.2.0
cc_compile_by          : nxf41040
```

```
cc_compile_domain      :
cc_compile_date        : Thu Jul  9 08:49:59 UTC 2020
build_id               : 6f101bbbf7a8f3f35404b08fe936718a0af2adda
xend_config_format     : 4
```

10. Print the information about existent domains:

```
# xl list
Name                                ID   Mem VCPUs State Time(s)
Domain-0                             0  2048     1    r-----   28.7
```

2.2.2 Run DomU

After Dom0 is booted, you can use the Xen management toolstack `xl` to create and access unprivileged domains.

1. Mount the guest filesystem:

```
# losetup /dev/loop0 $path/domu-rfs
```

   Where: `domu-rfs` is `fsl-image-networking-<target_board>.rootfs.ext2`

2. Create config file `domu.cfg` and add the following lines:

```
name = "domu"
vcpus = 1
memory = 1024
kernel = "$path/domu-image"
disk = [ 'phy:/dev/loop0,xvda,w' ]
extra = "root=/dev/xvda rw earlyprintk=xenboot console=hvc0"
```

   For more details, see https://xenbits.xen.org/docs/unstable/man/xl.cfg.5.html.

> **NOTE**
>
> You can use the same kernel for Dom0 and DomU, however it is not mandatory. If you want to use another kernel configuration for the unprivileged domain, follow the instructions listed at the end of section Download and install the Yocto Project to build a different kernel image.

3. Create domain:

```
# xl create domu.cfg
```

4. Verify that the domain has been created:

```
$: xl list
Name                                ID   Mem VCPUs  State     Time(s)
Domain-0                             0  4096     4  r-----      28.7
domu                                 1  1024     1  r-----       2.7
```

5. Boot domain:

```
# xl console domu
```

6. Login as `root`.

   The following command is analogous to running `xl create domu.cfg && xl console domu`:

```
# xl create -c domu.cfg
```

7. Detach from the domain console using `<CTRL> ']'`. Be aware that this escape character is also used by `telnet`. You may connect to the board console using `-E` option to stop any character from being used as an escape character:

```
# telnet -E <ip> <port>
```

8. Destroy domain:

```
# xl destroy domu
```

NOTE

Unprivileged domains cannot access the system hardware or device drivers. Pass through hardware to such domains or use available paravirtualization device drivers if needed.

2.2.3 Dom0less

There is an alternative solution to deploy Xen domains, namely dom0less. It does not rely on the control domain, instead it allows the user to create domains at boot time by passing required information to Xen via device tree modules.

1. After loading the Xen and Dom0 images, load DomU kernel and filesystem:

```
u-boot> tftp $domu_kernel_addr $path/domu-image
u-boot> tftp $domu_rootfs_addr $path/domu-rfs
```

NOTE

These instructions assume that you are using the commands described in the section Download and install the Yocto Project to obtain the kernel image and filesystem for the unprivileged domain.

2. Edit the device tree blob:

```
u-boot> fdt set /chosen/domU0 compatible "xen,domain"
u-boot> fdt set /chosen/domU0 \#address-cells <0x2>
u-boot> fdt set /chosen/domU0 \#size-cells <0x2>
u-boot> fdt set /chosen/domU0 memory <0x0 0x100000>
u-boot> fdt mknod /chosen domU0
u-boot> fdt set /chosen/domU0 cpus <0x1>
u-boot> fdt set /chosen/domU0 vpl011 <0x1>
u-boot> fdt mknod /chosen/domU0 module$domu_kernel_addr
u-boot> fdt set /chosen/domU0/module$domu_kernel_addr compatible "multiboot,kernel"
"multiboot,module"
u-boot> fdt set /chosen/domU0/module$domu_kernel_addr reg <0x0 $domu_kernel_addr 0x0
$domu_kernel_size>
u-boot> fdt set /chosen/domU0/module$domu_kernel_addr bootargs "console=ttyAMA0"
u-boot> fdt mknod /chosen/domU0 module$domu_rootfs_addr
u-boot> fdt set /chosen/domU0/module$domu_rootfs_addr compatible "multiboot,ramdisk"
"multiboot,module"
u-boot> fdt set /chosen/domU0/module$domu_roootfs_addr reg <0x0 $domu_rootfs_addr 0x0
$domu_rootfs_size>
```

NOTE

This information can be directly added in the device tree source file.

For more details, see https://xenbits.xenproject.org/docs/unstable/features/dom0less.html.

2.2.4 ImageBuilder

You can also use `ImageBuilder`, a collection of scripts used for build automation, to deploy Xen on Layerscape platforms.

The following config file has been tested on LS2088ARDB:

```
MEMORY_START="0x80000000"
MEMORY_END="0xFBE00000"

DEVICE_TREE="$path/fsl-ls2088a-rdb.dtb"
XEN="$path/xen-ls2088ardb"
DOM0_KERNEL="$path/dom0-img"
DOM0_RAMDISK="$path/dom0-rfs"

NUM_DOMUS=1
DOMU_KERNEL[0]="$path/domu-img"
DOMU_RAMDISK[0]="$path/domu-rfs"

UBOOT_SOURCE="boot.source"
UBOOT_SCRIPT="boot.scr"
```

Use the `bdinfo` U-Boot command to get the memory start address and size.

1. Invoke uboot-script-gen:

   ```
   $ bash ./scripts/uboot-script-gen -c config -d . – tftp
   Generated uboot script boot.scr, to be loaded at address 0x80200000:
   tftpb 0x80200000 boot.scr; source 0x80200000
   ```

   NOTE

   The `$path` is relative to the directory passed as an argument to the `-d` option.

2. Load and source the script in U-Boot using the instructions printed by `ImageBuilder`.

   There are some default options in `ImageBuilder`, such as `dom0-bootargs`. If needed, use the following instructions to perform changes:

   a. Edit `boot.source`

   b. Regenerate `boot.scr`

   ```
   $ mkimage -A arm64 -T script -C none -a 0x80200000 -d boot.source boot.scr
   ```

2.2.5 Enable SMMU support

In the Xen `arm-smmu` driver, the start level of the page table walk is hardcoded to 1, which causes errors in scenarios where the level has a different value, as is the case with Layerscape platforms.

If you want to run the Xen hypervisor on Layerscape platforms with SMMU enabled:

1. Apply the following patch which was sent upstream:

   ```
   From xen-devel  Fri Oct 02 10:33:44 2020
   From: laurentiu.tudor () nxp ! com
   Date: Fri, 02 Oct 2020 10:33:44 +0000
   To: xen-devel
   Subject: [PATCH v3] arm,smmu: match start level of page table walk with P2M
   Message-Id: <20201002103344.13015-1-laurentiu.tudor () nxp ! com>
   X-MARC-Message: https://marc.info/?l=xen-devel&m=160163484031524

   From: Laurentiu Tudor <laurentiu.tudor@nxp.com>

   Don't hardcode the lookup start level of the page table walk to 1
   and instead match the one used in P2M. This should fix scenarios
   ```

```
involving SMMU where the start level is different than 1.
In order for the SMMU driver to also compile on arm32 move the
P2M_ROOT_LEVEL in the p2m header file (while at it, for
consistency also P2M_ROOT_ORDER) and use the macro in the smmu
driver.

Signed-off-by: Laurentiu Tudor <laurentiu.tudor@nxp.com>
---
Changes in v3:
 - also export 'p2m_root_order'
 - moved variables in their rightful #ifdef block

Changes in v2:
 - made smmu driver compile on arm32

 xen/arch/arm/p2m.c                 |  9 ++-------
 xen/drivers/passthrough/arm/smmu.c |  2 +-
 xen/include/asm-arm/p2m.h          | 11 ++++++++++
 3 files changed, 14 insertions(+), 8 deletions(-)

diff --git a/xen/arch/arm/p2m.c b/xen/arch/arm/p2m.c
index ce59f2b503..4eeb867ca1 100644
--- a/xen/arch/arm/p2m.c
+++ b/xen/arch/arm/p2m.c
@@ -17,17 +17,12 @@
 #define INVALID_VMID 0 /* VMID 0 is reserved */

 #ifdef CONFIG_ARM_64
-static unsigned int __read_mostly p2m_root_order;
-static unsigned int __read_mostly p2m_root_level;
-#define P2M_ROOT_ORDER    p2m_root_order
-#define P2M_ROOT_LEVEL p2m_root_level
+unsigned int __read_mostly p2m_root_order;
+unsigned int __read_mostly p2m_root_level;
 static unsigned int __read_mostly max_vmid = MAX_VMID_8_BIT;
 /* VMID is by default 8 bit width on AArch64 */
 #define MAX_VMID       max_vmid
 #else
-/* First level P2M is always 2 consecutive pages */
-#define P2M_ROOT_LEVEL 1
-#define P2M_ROOT_ORDER    1
 /* VMID is always 8 bit width on AArch32 */
 #define MAX_VMID        MAX_VMID_8_BIT
 #endif
diff --git a/xen/drivers/passthrough/arm/smmu.c b/xen/drivers/passthrough/arm/smmu.c
index 94662a8501..4ba6d3ab94 100644
--- a/xen/drivers/passthrough/arm/smmu.c
+++ b/xen/drivers/passthrough/arm/smmu.c
@@ -1152,7 +1152,7 @@ static void arm_smmu_init_context_bank(struct arm_smmu_domain *smmu_domain)
           (TTBCR_RGN_WBWA << TTBCR_IRGN0_SHIFT);

     if (!stage1)
-        reg |= (TTBCR_SL0_LVL_1 << TTBCR_SL0_SHIFT);
+        reg |= (2 - P2M_ROOT_LEVEL) << TTBCR_SL0_SHIFT;

     writel_relaxed(reg, cb_base + ARM_SMMU_CB_TTBCR);

diff --git a/xen/include/asm-arm/p2m.h b/xen/include/asm-arm/p2m.h
index 5fdb6e8183..28ca9a838e 100644
--- a/xen/include/asm-arm/p2m.h
```

```
+++ b/xen/include/asm-arm/p2m.h
@@ -13,6 +13,17 @@
 /* Holds the bit size of IPAs in p2m tables.  */
 extern unsigned int p2m_ipa_bits;

+#ifdef CONFIG_ARM_64
+extern unsigned int p2m_root_order;
+extern unsigned int p2m_root_level;
+#define P2M_ROOT_ORDER    p2m_root_order
+#define P2M_ROOT_LEVEL p2m_root_level
+#else
+/* First level P2M is always 2 consecutive pages */
+#define P2M_ROOT_ORDER    1
+#define P2M_ROOT_LEVEL 1
+#endif
+
 struct domain;

 extern void memory_type_changed(struct domain *);
--
2.17.1
```

NOTE

For more details regarding the two-level table access, see *https://developer.arm.com/documentation/100940/0101*.

The Xen `arm-smmu` driver uses the legacy device tree bindings, therefore additional changes to the device tree used by Dom0 are required. For instance, the `mmu-masters` property is deprecated, therefore it is not included in the native device tree source.

2. Add the `mmu-masters` property to the `smmu` node and specify the `StreamID` for each master:

```
smmu: iommu@9000000 {
    /* Other properties */
    mmu-masters = <&eshdc 0xc04>, <&sata 0xc05> …
};
```

3. Add `#stream-id-cells` to each node listed in `mmu-masters`:

```
esdhc: esdhc@1560000 {
    /* Other properties */
    #stream-id-cells = <1>;
};

sata: sata@3200000 {
    /* Other properties */
    #stream-id-cells = <1>;
};
```

In case of misconfiguration, a global fault is triggered and the peripheral will not work. For instance, in the following example, the `StreamID` 0x420 was received by SMMU, but could not be identified.

```
(XEN) smmu: /soc/iommu@5000000: Unexpected global fault, this could be serious
(XEN) smmu: /soc/iommu@5000000:  GFSR 0x80000002, GFSYNR0 0x00000000, GFSYNR1 0x00000420, GFSYNR2
0x00000000
```

2.2.6 Device passthrough

You can assign resources to unprivileged guests via device passthrough. The following instructions provide guidance on how to passthrough a USB device. Same steps can be performed to passthrough a PCIe controller. For more details, see https://xenbits.xen.org/docs/4.9-testing/misc/arm/passthrough.txt.

USB passthrough

1. Create a device tree blob for DomU:

```
/dts-v1/;

/ {
    #address-cells = <2>;
    #size-cells = <2>;

    passthrough {
        compatible = "simple-bus";
        ranges;
        #address-cells = <2>;
        #size-cells = <2>;
        usb0: usb3@3100000 {
                  status = "okay";
                  #stream-id-cells = <0x1>;
                  compatible = "snps,dwc3";
                  reg = <0x0 0x3100000 0x0 0x10000>;
                  interrupts = <0 80 0x4>;
                  dr_mode = "host";
                  snps,quirk-frame-length-adjustment = <0x20>;
                  snps,dis_rxdet_inp3_quirk;
                  snps,incr-burst-type-adjustment = <1>, <4>, <8>, <16>;
                  snps,host-vbus-glitches;
              };
    };
};
```

2. Run the following command in U-Boot to mark the device, so that Xen is aware that it will be used for passthrough:

```
u-boot> fdt set /soc/usb3@3100000 "xen,passthrough"
```

3. Add the following lines to the DomU `domu.cfg` config file:

```
device_tree = "$path/domu.dtb"
irqs = [ 112 ]
iomem = [ "0x03100,0x10" ]
```

PCIe controller passthrough (LS1046ARDB)

This scenario is like the previous one, except it requires the passthrough of MSI controllers as well since they are used by the controller.

NOTE

On this platform, the MSI controller is NXP custom and it is seen by Xen as a regular device, being assigned by default to Dom0. Dom0 receives regular interrupts, which are converted to MSIs by the device driver. Hence, it is possible to use PCIe with MSIs, even though MSI support is missing in Xen.

3 Known limitations

DPAA2 features are not supported because management complex bus and drivers support is missing in Xen.

Device passthrough of devices which require a clocksource is not supported.

MSI support is missing in Xen.

4 Troubleshooting

4.1 Shared interrupts

On Layerscape platforms, serial devices share the same interrupt as it can be noticed in the below device tree fragment:

```
duart0: serial@21c0500 {
            compatible = "fsl,ns16550", "ns16550a";
            reg = <0x00 0x21c0500 0x0 0x100>;
            interrupts = <GIC_SPI 54 IRQ_TYPE_LEVEL_HIGH>;
            clocks = <&clockgen 4 1>;
};

duart1: serial@21c0600 {
            compatible = "fsl,ns16550", "ns16550a";
            reg = <0x00 0x21c0600 0x0 0x100>;
            interrupts = <GIC_SPI 54 IRQ_TYPE_LEVEL_HIGH>;
            clocks = <&clockgen 4 1>;
};
```

Xen does not allow shared interrupts between it and the other domains. In a basic scenario, running Xen and Dom0, one UART is used by the hypervisor and the second is assigned to the domain when it is created. One workaround is to remove the `duart1` node from the Dom0 device tree.

4.2 SPIs assignment

Because Xen is assigning only SPIs to its domains, devices that use other types of interrupts must be removed from the dts (for example, on LS1028ARDB the PMU uses a PPI type interrupt). Otherwise, domain 0 cannot be properly set up and a panic event is triggered.

4.3 LS1028ARDB boot issues

On LS1028ARDB platforms, the GPU node has the entire low memory bank reserved. This triggers a panic event in Xen scenarios. One possible workaround is to remove the `gpu` node from the dts or reserve the second memory bank instead of the first one.

4.4 Duplicate stream IDs

In the current SMMU driver implementation, the scenario in which multiple master devices share the same stream ID is not supported. To overcome this issue, it is enough to list only one of the master devices in `mmu-masters`.

5 References

- https://wiki.xenproject.org
- https://xenbits.xen.org/docs/

6 Revision history

Table 1 summarizes the changes made to this document since the initial release.

Table 1. Revision history

Revision number	Date	Substantive changes
0	03/2021	Initial release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. 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.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 03/2021

Document identifier: AN13138

arm