Qualcomm Linux Boot Guide - Addendum

80-70018-4A AC

April 10, 2025

Contents

1 Overview

This addendum serves as a supplementary guide, providing more details about the boot architecture, tools, and features. It's intended for developers who have full access to the proprietary software shipped with Qualcomm® Linux®.

You must review the Qualcomm Linux Boot Guide before reading this addendum. .

Note: See Hardware SoCs that are supported on Qualcomm Linux.

2 Develop CDT and UEFI

This section helps you to configure platform IDs for Qualcomm reference hardware boards using the configuration data table (CDT), and to develop UEFI-based applications.

3 Configuration data table

The configuration data table (CDT) is a data block containing platform or device-dependent data, such as the hardware platform ID. Various software modules can use this information to reduce dependencies and perform dynamic initialization. For details about CDT, see CDT on Qualcomm Linux.

4 Unified Extensible Firmware Interface (UEFI)

UEFI development consists of two components: the application and the driver. The EFI loader is responsible for loading and executing both types of extensible firmware interface (EFI) loadable images. All executable files, whether they're drivers or applications, have the .efi file extension.

All executables are .efi files (both driver and application).

For the UEFI specification, see https://github.com/tianocore/tianocore.github.io/wiki/EDK-II/.

4.1 UEFI driver

The UEFI driver consumes and produces protocols that reside in memory and provide services in the form of protocols. The following are the different types of UEFI drivers:

- Driver execution environment (DXE) drivers, where most of the system initialization is done with EFI boot services, which load the UEFI drivers.
- UEFI drivers, where the module-specific functionality is implemented.

For UEFI drivers, it's a best practice to define the UEFI protocols in the UEFI specification. If not, they should be converted to DXE drivers.

The UEFI driver must specify an entry point function that's called upon loading the driver. For example, for the I2C UEFI driver, MODULE_TYPE is DXE_DRIVER and the entry point function is `I2CProtocotlInit`.

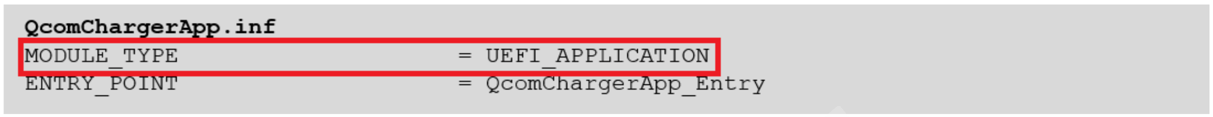
I2CDxe.inf

```
BASE_NAME                        = I2C
MODULE_TYPE                      = DXE_DRIVER
VERSION_STRING                   = 1.0
  ENTRY_POINT                      = I2CProtocolInit
[LibraryClasses]
  I2CLib
```

4.2 UEFI application

The UEFI application consumes but doesn't produce the protocols. The applications are removed from memory after exit.

For the UEFI application, `MODULE_TYPE` is `UEFI_APPLICATION`.



```
QcomChargerApp.inf
MODULE_TYPE                              = UEFI_APPLICATION
ENTRY_POINT                              = QcomChargerApp_Entry
```

The UEFI application provides the following:

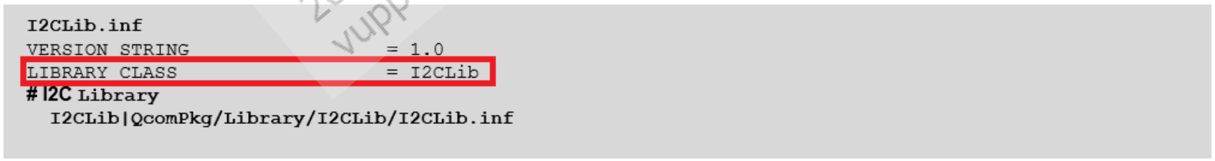
- Platform and factory diagnostics to identify, isolate, and diagnose hardware issues.
- Utilities to perform configurations.
- Driver prototyping to validate the driver as an application before enabling it as a UEFI driver.
- Applications for executing specific use cases defined by the platform.

The application focuses on user interaction and the driver focuses on providing boot and runtime services. The user configures the protocols.

4.3 UEFI library

The UEFI library is designed to link platform or device-specific functionalities (APIs) to one or more UEFI drivers and UEFI applications.

The UEFI driver links the library using `LIBRARY_CLASS`.For example, the `LIBRARY_CLASS` name is `I2CLib`.



```
I2CLib.inf
VERSION_STRING                    = 1.0
LIBRARY_CLASS                     = I2CLib
# I2C Library
  I2CLib|QcomPkg/Library/I2CLib/I2CLib.inf
```

For more information about UEFI development, see https://github.com/tianocore/tianocore.github.io/wiki/UEFI-EDKII-Learning-Dev.

Build a custom protocol

Implement dynamic linking using a protocol. Use libraries for static linking. UEFI drivers access protocol interfaces produced by other modules using protocol services.

Access a UEFI protocol, which is a structure of function pointers associated with a global unique identifier (GUID), using a handle. Each executable image in UEFI has a handle protocol in the handle database.

The UEFI driver adds the UEFI protocol to the system table using the `gBS->InstallProtocol(...)` interface. Different modules retrieve the protocol interface using LocateProtocol().

The UEFI specification defines a set of boot services to handle protocols, including:

- Services to install protocols:
  - InstallProtocolInterface()
  - ReInstallProtocolInterface()
  - InstallMultipleProtocolInterfaces()
- Services to retrieve protocols:
  - LocateProtocol()
  - HandleProtocol()
  - OpenProtocol()

The following example demonstrates how to use `RamPartitionProtocol`, available in the `QcomPkg/Include/Protocol/EFIRamPartition.h` file. This file contains all the declarations and implementations done in the `EnvDXE` driver. The UEFI application uses the protocol to print the RAM partitions.

Include the GUID definition associated with this protocol as follows:

```
/*  Protocol GUID definition */
/** @ingroup efi_ramPartition_protocol */
#define EFI_RAMPARTITION_PROTOCOL_GUID \
{ 0x5172FFB5, 0x4253, 0x7D51, { 0xC6, 0x41, 0xA7, 0x01, 0xF9, 0x73,
0x10, 0x3C } }
/** @cond */
/**
External reference to the RAMPARTITION Protocol GUID defined in the .
dec file.
*/
extern EFI_GUID gEfiRamPartitionProtocolGuid;
/** @endcond */
```

The following code define the functions and structures for this protocol:

```
typedef struct _RamPartition {
UINT64 Base;
UINT64 AvailableLength;
}RamPartitionEntry; typedef
EFI_STATUS
(EFIAPI *EFI_RAMPARTITION_GETRAMPARTITIONVERSION)( IN EFI_
RAMPARTITION_PROTOCOL *This,
OUT UINT32  *MajorVersion,
OUT UINT32  *MinorVersion );
typedef EFI_STATUS
(EFIAPI *EFI_RAMPARTITION_GETHIGHESTBANKBIT)
```

The following `EnvDxe` driver code shows the RAM partition table function:

```
STATIC EFI_RAMPARTITION_PROTOCOL RamPartitionProtocol =
{
  EFI_RAMPARTITION_PROTOCOL_REVISION,
  EFI_GetRamPartitionVersion,
  EFI_GetHighestBankBit,
  EFI_GetRamPartitions,
  EFI_GetMinPasrSize,
  EFI_GetInstalledPhysicalMemory,
  EFI_GetPreLoadedRamPartitions
};


EFI_STATUS
SetupRamPartitionProtocol ()
{
  EFI_STATUS Status;
  STATIC EFI_HANDLE ImageHandle;

  Status = InitRamPartitionTableLib();
  if (Status != EFI_SUCCESS)
    return EFI_UNSUPPORTED;

  Status = gBS->InstallMultipleProtocolInterfaces (&ImageHandle,
                                                    &
gEfiRamPartitionProtocolGuid,
                                                    &
RamPartitionProtocol,
                                                    NULL, NULL);
  return Status;
}
```

Implement UEFI DXE driver

The DXE drivers are responsible for initializing the processor, chip products, and platform components. Additionally, they provide software abstractions for system services, console devices, and boot devices.

For more information about UEFI DXE, see https://uefi.org/specs/PI/1.8/V2_Overview.html.

Before you implement the UEFI DXE driver:

- Create the `<uefidriver_name>.inf` file for the driver at `boot_images/boot/QcomPkg/Drivers/<uefidriver_name>/<uefidriver_name>.inf`.
- Implement the functions produced by this UEFI driver.
- Install the protocols to consume the functions implemented by this UEFI driver.

To implement the UEFI DXE driver, follow these steps:

1. Add the driver in the description (DSC) file of the package and the flash description file (FDF). Go to the `boot_images\boot\QcomPkg\Drivers` build directory to add a UEFI driver that implements the UEFI protocols.

   This example shows the implementation of `RamPartitionProtocol` declared in Build a custom protocol in the `QcomPkg/Drivers/EnvDxe/EnvDxe.c` UEFI driver.

2. Create the `EnvDxe.inf` file for the driver at `QcomPkg/Drivers/EnvDxe/EnvDxe.inf`. Ensure that the `MODULE_TYPE` entry is `DXE_RUNTIME_DRIVER` or `DXE_DRIVER`.

   The following are the contents of the `EnvDXE.inf` file:

   ```
   [Defines]
   INF_VERSION = 0x00010005
   BASE_NAME   = EnvDxe
   FILE_GUID   = 90A49AFD-422F-08AE-9611-E788D3804845
   MODULE_TYPE = DXE_RUNTIME_DRIVER
   VERSION_STRING  = 1.0
   ENTRY_POINT = EnvDxeInitialize
   [Sources]
   EnvDxe.c
   [BuildOptions.AARCH64]
   #GCC:*_*_*_CC_FLAGS = -O0
   #MSFT:*_*_*_CC_FLAGS = -O0
   [Packages] MdePkg/MdePkg.dec ArmPkg/ArmPkg.dec
   MdeModulePkg/MdeModulePkg.dec QcomPkg/QcomPkg.dec
   [LibraryClasses] UefiDriverEntryPoint UefiBootServicesTableLib
   ```

```
UefiRuntimeServicesTableLib QcomBaseLib
DebugLib ProcLib PcdLib FBPTLib
PeCoffGetEntryPointLib BaseLib
UefiLib DxeServicesTableLib RamPartitionTableLib QcomLib
UefiCfgLib SerialPortLib
[GUID]
gEfiEventFBPTOsLoaderLoadImageStartGuid
gEfiEventFBPTOsLoaderStartImageStartGuid
gEfiEventFBPTExitBootServicesEntryGuid
gEfiEventFBPTExitBootServicesExitGuid
gEfiEventVirtualAddressChangeGuid gEfiDebugImageInfoTableGuid
gQcomFwVersionHexGuid
gBlockIoRefreshGuid
[Protocols] gEfiCpuArchProtocolGuid gEfiTimerArchProtocolGuid
gEfiRamPartitionProtocolGuid
[FixedPCD] gQcomTokenSpaceGuid.PcdFBPTPayloadBaseOffset
gArmTokenSpaceGuid.PcdCpuVectorBaseAddress
[Depex] TRUE
```

For information about the INF file, see Table : File definitions (UEFI driver and application INF files).

3. Implement the functions declared in `QcomPkg/Include/Protocol/EFIRamPartition.h`. See Build a custom protocol.

```
EFI_STATUS
EFI_GetRamPartitionVersion (
IN EFI_RAMPARTITION_PROTOCOL *This,
OUT UINT32  *MajorVersion,
OUT UINT32  *MinorVersion
){
if ((MajorVersion == NULL) || (MinorVersion == NULL) ) return
EFI_INVALID_PARAMETER;
*MajorVersion = 2;
*MinorVersion = 0; return EFI_SUCCESS;}
EFI_STATUS
EFI_GetHighestBankBit (
IN EFI_RAMPARTITION_PROTOCOL *This,
OUT UINT32  *HighestBankBit){
if (HighestBankBit == NULL) return EFI_INVALID_PARAMETER;
*HighestBankBit = 0;
#ifdef PRE_SIL
return EFI_UNSUPPORTED;
#else
return EFI_SUCCESS;
```

```
#endif } EFI_STATUS
EFI_GetRamPartitions (
IN EFI_RAMPARTITION_PROTOCOL *This,
OUT RamPartitionEntry   *RamPartitions, IN OUT UINT32
*NumPartition
){
EFI_STATUS  Status;
MemRegionInfo *pRamPartTable = NULL; UINTN  AvailNumParts = 0;
UINTN   i   = 0;
if (NumPartition == NULL) return EFI_INVALID_PARAMETER;
Status = GetRamPartitions(&AvailNumParts, NULL); if (Status ==
EFI_BUFFER_TOO_SMALL){
if (AvailNumParts > (UINTN)(*NumPartition)){
```

4. Install the protocol using the `InstallMultipleProtocolinterfaces` function.

```
STATIC EFI_RAMPARTITION_PROTOCOL RamPartitionProtocol = {
EFI_RAMPARTITION_PROTOCOL_REVISION,
EFI_GetRamPartitionVersion, EFI_GetHighestBankBit, EFI_
GetRamPartitions
};
EFI_STATUS
SetupRamPartitionProtocol ()
{EFI_STATUS Status;
STATIC EFI_HANDLE ImageHandle;
Status = InitRamPartitionTableLib(); if (Status != EFI_SUCCESS)
return EFI_UNSUPPORTED;
Status = gBS->InstallMultipleProtocolInterfaces (&ImageHandle,
&gEfiRamPartitionProtocolGuid,
return Status;
&RamPartitionProtocol, NULL, NULL); }
```

5. Add the driver in the description file of the package as described in Add application to description files.

   In the following example, add `EnvDxe.inf` to the `<chipset>Pkg_Core.dsc` file.

```
QcomPkg/Drivers/EnvDxe/EnvDxe.inf {
<LibraryClasses> DebugLib|QcomPkg/Library/DebugLib/DebugLib.inf
}
```

6. Add the driver to the flash description file. For more information, see Add application to description files.. For example, add `EnvDxe.inf` to the `<chipset>Pkg.fdf` file.

Build and add a UEFI application

Use the UEFI test application in `QcomPkg/QcomTestCommon/EnvTest` as an example.

1. Create the `EnvTest.inf` file for the application.

   The INF file has information about the application as shown in the following sample.

   For the UEFI application, the `MODULE_TYPE` entry should be `UEFI_APPLICATION`. The following is a sample `EnvTest.inf` file:

```
[Defines]
INF_VERSION = 0x00010005
BASE_NAME   = EnvTest
FILE_GUID   = 8CEC0EDF-EF17-425E-8F6A-4B46274FB7C9 MODULE_TYPE
   = UEFI_APPLICATION
VERSION_STRING  = 1.0
ENTRY_POINT = EnvTestMain
#
# The following information is for reference only and not
required by the build tools.
#
#   VALID_ARCHITECTURES = ARM AARCH64
#
[Sources]
EnvTest.c
[BuildOptions.AARCH64]
#GCC:*_*_*_CC_FLAGS = -O0
[Packages] MdePkg/MdePkg.dec
MdeModulePkg/MdeModulePkg.dec QcomPkg/QcomPkg.dec
[Protocols] gEfiRamPartitionProtocolGuid
[LibraryClasses] UefiApplicationEntryPoint UefiLib
TestInterfaceLib MemoryAllocationLib
```

   The following table lists the main section definitions of the INF file.

Table : File definitions (UEFI driver and application INF files)

Item	Description
`INF_VERSION`	Identifies the INF version.
`BASE_NAME`	The name of the driver or application.
`FILE_GUID`	The unique GUID of the driver or application.
`MODULE_TYPE`	The type of module for the driver or application. For example, `UEFI_APPLICATION`
`VERSION_STRING`	The version number of the string to be used, typically `1.0`.
`ENTRY_POINT`	The entry function of the driver or application.

Item	Description
`[Sources]`	The names of the source files in the driver or, application, listed line by line.
`[Packages]`	The different module packages used in the driver or, application, listed line by line.
`[LibraryClasses]`	The libraries that are linked into the code are listed line by line.
`[Guids]`	The list of the global GUID names that are used by the driver or application.
`[Protocols]`	The UEFI protocols consumed or produced by the driver or application.
`[FeaturePcd]`(if necessary)	The feature allows runtime selection of code from the platform configuration database (PCD).
`[Pcd]` (if necessary)	The fixed PCD values used in this component.
`[Depex]` (if necessary)	A list of dependent drivers that must be loaded before this driver or application can load or execute.

In the UEFI application, the source file is at `QcomPkg/ QcomTestPkg/EnvTest/ EnvTest.c`. The following is a simplified explanation of the roles:

- `#include<Uefi.h>`: Required header file for every UEFI application. It includes the types and constructs defined in UEFI.
- `#include<Library/UefiApplicationEntryPoint.h>`: This library contains the functions required to define the entry point of the UEFI application.
- `#include<Library/UefiBootServicesTableLib.h>`: This library allows the UEFI application to access boot services.
- `#include<Library/UefiRuntimeServicesTableLib.h>`: This library allows the UEFI application to access runtime services.

2. Define the entry function for the application.

   The application must have an entry function defined in the source code that indicates where the application begins to run. For example, list the prototype of the entry function in `EnvTest.c`.

   For the UEFI application entry point, there are two parameters:

   - `ImageHandle`: This is the image handle of the UEFI application
   - `SystemTable`: This is the pointer to the EFI system table

```
/**
Entry point for the application
@param[in] ImageHandle  Image handle
@param[in] SystemTable  Pointer to the System Table
@retval EFI_SUCCESS Execution successful
```

```
@retval other   Error occurred
\**/ EFI_STATUS EFIAPI
EnvTestMain (
IN EFI_HANDLE   ImageHandle, IN EFI_SYSTEM_TABLE    *SystemTable
){
EFI_STATUS  Status; TEST_START("EnvTest");
Status = RamPartProtocolTest(); TestStatus("EnvTest", Status);
TEST_STOP("EnvTest");
return EFI_SUCCESS;}
```

The entry function calls `RamPartProtocolTest` to enable the application to display the RAM partitions.

```
STATIC EFI_STATUS
RamPartProtocolTest (VOID){
EFI_STATUS  Status = EFI_NOT_FOUND;
EFI_RAMPARTITION_PROTOCOL   *pRamPartProtocol = NULL;
RamPartitionEntry *pRamPartitions = NULL;
UINT32  NumPartitions = 0;
UINT32  i = 0;
Status = gBS->LocateProtocol(&gEfiRamPartitionProtocolGuid,
NULL, (VOID**)&pRamPartProtocol);
if (EFI_ERROR(Status) || (&pRamPartProtocol == NULL)){
DEBUG((EFI_D_ERROR, "Locate EFI_RAMPARTITION_Protocol failed,
Status
=   (0x%x)\r\n", Status)); return EFI_NOT_FOUND;}
Status = pRamPartProtocol->GetRamPartitions (pRamPartProtocol,
NULL,
&NumPartitions);
if (Status == EFI_BUFFER_TOO_SMALL){
pRamPartitions = AllocatePool (NumPartitions * sizeof
(RamPartitionEntry));
if (pRamPartitions == NULL) return EFI_OUT_OF_RESOURCES;
Status = pRamPartProtocol->GetRamPartitions (pRamPartProtocol,
pRamPartitions, &NumPartitions);
if (EFI_ERROR (Status) || (NumPartitions < 1) )      {
DEBUG((EFI_D_ERROR, "Failed to get RAM partitions")); return
EFI_NOT_FOUND;
}}
DEBUG ((EFI_D_WARN, "\n------------------\n"));
DEBUG ((EFI_D_WARN, "+ RAM Partitions +\n"));
DEBUG ((EFI_D_WARN, "------------------\n"));
DEBUG((EFI_D_WARN, "Base \t\t\t Available Length\n")); for (i =
0; i < NumPartitions; i++) {
```

```
DEBUG((EFI_D_WARN, "0x%016lx \t 0x%016lx\n", pRamPartitions[i].
Base, pRamPartitions[i].AvailableLength));}
return EFI_SUCCESS;}
```

3. Use LocateProtocol() to obtain the `gEfiRamPartitionProtocolGuid` protocol interface.
4. Use `GetRamPartitions` to obtain partition information once LocateProtocol() provides the interface information.

Add application to description files

To build the application with the relevant package, add it to the packager's description (DSC) file. Additionally, to include the application in the flash image, add it to the flash description (FDF) file of the corresponding package.

Add the application in the description (DSC) file of the package

Add the INF file of the application to the DSC file.

Since the Qualcomm UEFI test application is part of QcomTestPkg, add it to the package-specific DSC file.

For example, add `QcomTestPkg` to `QcomTestPkg.dsc`, located at `QcomPkg/QcomTestPkg/QcomTestPkg.dsc`.

In this case, add `EnvTest.inf` to the `QcomTestPkg.dsc` file.

```
## Test Applications
#QcomPkg/QcomTestPkg/EnvTest/EnvTest.inf
```

Add application to flash (FDF) description file

Add the INF file of the application to the DSC file.

For example, since the Qualcomm Linux application is part of `QcomTestPkg`, add it to the package-specific FDF file. In this case, add `QcomTestPkg`, to `QcomTestPkg.fdf`, located at `QcomPkg/QcomTestPkg/ QcomTestPkg.fdf`.

In the following example, add `EnvTest.inf` to the `QcomTestPkg.fdf` file.

```
## Test Applications
#INF QcomPkg/QcomTestPkg/EnvTest/EnvTest.inf
```

Use DeviceTree in UEFI

The XBL DeviceTree feature supports `pre-ddr-<chipset>-1.0.dtb`, `post-ddr-<chipset>-1.0.dtb`, and `platform.dtb` DeviceTree blobs (DTB).

- The `pre-ddr-<chipset>-1.0.dtb` and `post-ddr-<chipset>-1.0.dtb` contains the configurations that are loaded before and after DDR is initialized.
- The `platform.dtb` contains the DeviceTree configuration.

Configure DeviceTree

UEFI uses a DeviceTree format packaged into the XBL configuration to support predefined customizations.

The following steps outline the process to generate a DeviceTree binary:

1. Qualcomm distributes DeviceTree source (DTS) and DTSI files within the boot software.
2. Add your own DeviceTree configurations for UEFI.
3. Build the boot image. For more information, see Build BOOT.
4. The new DeviceTree configuration is available as part of `xbl_config.elf`. Flash the `xbl_config.elf` image.

5 References

Related documents:

Title	**Document number**
Document	
Qualcomm Linux Boot Guide	80-70018-4
CDT usage on Qualcomm Linux Platform	80-70018-9

Acronyms:

Acronym	**Definition**
DeviceTree blobs	DTB
DeviceTree source	DTS
Global unique identifier	GUID
Unified Extensible Firmware Interface	UEFI
Flash description file	FDF

LEGAL INFORMATION

Your access to and use of this material, along with any documents, software, specifications, reference board files, drawings, diagnostics and other information contained herein (collectively this "Material"), is subject to your (including the corporation or other legal entity you represent, collectively "You" or "Your") acceptance of the terms and conditions ("Terms of Use") set forth below. If You do not agree to these Terms of Use, you may not use this Material and shall immediately destroy any copy thereof.

1) Legal Notice.

This Material is being made available to You solely for Your internal use with those products and service offerings of Qualcomm Technologies, Inc. ("**Qualcomm Technologies**"), its affiliates and/or licensors described in this Material, and shall not be used for any other purposes. If this Material is marked as "**Qualcomm Internal Use Only**", no license is granted to You herein, and You must immediately (a) destroy or return this Material to Qualcomm Technologies, and (b) report Your receipt of this Material to qualcomm.support@qti.qualcomm.com. This Material may not be altered, edited, or modified in any way without Qualcomm Technologies' prior written approval, nor may it be used for any machine learning or artificial intelligence development purpose which results, whether directly or indirectly, in the creation or development of an automated device, program, tool, algorithm, process, methodology, product and/or other output. Unauthorized use or disclosure of this Material or the information contained herein is strictly prohibited, and You agree to indemnify Qualcomm Technologies, its affiliates and licensors for any damages or losses suffered by Qualcomm Technologies, its affiliates and/or licensors for any such unauthorized uses or disclosures of this Material, in whole or part.

Qualcomm Technologies, its affiliates and/or licensors retain all rights and ownership in and to this Material. No license to any trademark, patent, copyright, mask work protection right or any other intellectual property right is either granted or implied by this Material or any information disclosed herein, including, but not limited to, any license to make, use, import or sell any product, service or technology offering embodying any of the information in this Material.

THIS MATERIAL IS BEING PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY LAW, QUALCOMM TECHNOLOGIES, ITS AFFILIATES AND/OR LICENSORS SPECIFICALLY DISCLAIM ALL WARRANTIES OF TITLE, MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, COMPLETENESS OR ACCURACY, AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MOREOVER, NEITHER QUALCOMM TECHNOLOGIES, NOR ANY OF ITS AFFILIATES AND/OR LICENSORS, SHALL BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY EXPENSES, LOSSES, USE, OR ACTIONS HOWSOEVER INCURRED OR UNDERTAKEN BY YOU IN RELIANCE ON THIS MATERIAL.

Certain product kits, tools and other items referenced in this Material may require You to accept additional terms and conditions before accessing or using those items.

Technical data specified in this Material may be subject to U.S. and other applicable export control laws. Transmission contrary to U.S. and any other applicable law is strictly prohibited.

Nothing in this Material is an offer to sell any of the components or devices referenced herein.

This Material is subject to change without further notification.

In the event of a conflict between these Terms of Use and the *Website Terms of Use* on www.qualcomm.com, the *Qualcomm Privacy Policy* referenced on www.qualcomm.com, or other legal statements or notices found on prior pages of the Material, these Terms of Use will control. In the event of a conflict between these Terms of Use and any other agreement (written or click-through, including, without limitation any non-disclosure agreement) executed by You and Qualcomm Technologies or a Qualcomm Technologies affiliate and/or licensor with respect to Your access to and use of this Material, the other agreement will control.

These Terms of Use shall be governed by and construed and enforced in accordance with the laws of the State of California, excluding the U.N. Convention on International Sale of Goods, without regard to conflict of laws principles. Any dispute, claim or controversy arising out of or relating to these Terms of Use, or the breach or validity hereof, shall be adjudicated only by a court of competent jurisdiction in the county of San Diego, State of California, and You hereby consent to the personal jurisdiction of such courts for that purpose.

2) Trademark and Product Attribution Statements.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the U.S. and/or elsewhere. The Bluetooth® word mark is a registered trademark owned by Bluetooth SIG, Inc. Other product and brand names referenced in this Material may be trademarks or registered trademarks of their respective owners.

Snapdragon and Qualcomm branded products referenced in this Material are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.