BSP User's Guide

Generic x86 and x86_64

BlackBerry | QNX

©2014–2020, QNX Software Systems Limited, a subsidiary of BlackBerry Limited. All rights reserved.

QNX Software Systems Limited
1001 Farrar Road
Ottawa, Ontario
K2K 0B3
Canada

Voice: +1 613 591-0931
Fax: +1 613 591-3579
Email: info@qnx.com
Web: http://www.qnx.com/

Trademarks, including but not limited to BLACKBERRY, EMBLEM Design, QNX, MOMENTICS, NEUTRINO, and QNX CAR, are the trademarks or registered trademarks of BlackBerry Limited, its subsidiaries and/or affiliates, used under license, and the exclusive rights to such trademarks are expressly reserved. All other trademarks are the property of their respective owners.

Patents per 35 U.S.C. § 287(a) and in other jurisdictions, where allowed: http://www.blackberry.com/patents

Electronic edition published: January 31, 2020

Contents

About This Guide

This guide contains installation and start-up instructions for the QNX Board Support Package (BSP) for x86_64 platforms. This guide can be used for both the x86_64 (64-bit) and x86 (32-bit) variants of the BSP. Throughout this guide, *x86_64* also refers to *x86* unless explicitly stated. It's important to note that SMMU Manager support has been added for the x86_64 (64-bit) platform of the BSP, however SMMU Manager support isn't available for the x86 (32-bit) platform.

At times, there's specific information for variants of the x86_64 platforms based on a particular processor family or SoC. You should refer to information in those chapters if your board uses that processor family or SoC.

This documentation includes:

- an overview of QNX BSP
- links to documentation about how to build QNX embedded systems, which you should read before you begin working with this BSP
- what's new in the BSPs for this release
- structure and contents of a BSP
- how to prepare a bootable USB device (e.g., USB memory stick)
- how to install the BSP and boot it on a board
- how to build your BSP

To find out about:	See:
The resources available to you, and what you should know before starting to work with this BSP	*Before You Begin*
What's included in the BSP, supported host operating systems, and boards	*About This BSP*
Installing this BSP	*Installation Notes*
Using the Screen Graphics Subsystem with this BSP	*Using the Screen Graphics Subsystem*
Building this BSP	*Build the BSP*
Driver commands for both 32-bit and 64-bit variants of the BSP	*Generic x86_64 (64-bit) Driver Commands* and *Generic x86 (32-bit) Driver Commands*
Working with ADI RCC C2000 (Rangeley) Platforms	*Working with ADI RCC C2000 Platforms*
Working with Denverton Platforms	*Working with Denverton SoC-based platforms*

© 2020, QNX Software Systems Limited

Typographical conventions

Throughout this manual, we use certain typographical conventions to distinguish technical terms. In general, the conventions we use conform to those found in IEEE POSIX publications.

The following table summarizes our conventions:

Reference	Example
Code examples	`if( stream == NULL)`
Command options	`-lR`
Commands	`make`
Constants	NULL
Data types	`unsigned short`
Environment variables	*PATH*
File and pathnames	**/dev/null**
Function names	*exit()*
Keyboard chords	**Ctrl–Alt–Delete**
Keyboard input	`Username`
Keyboard keys	**Enter**
Program output	`login:`
Variable names	*stdin*
Parameters	*parm1*
User-interface components	**Navigator**
Window title	**Options**

We use an arrow in directions for accessing menu items, like this:

You'll find the Other... menu item under **Perspective → Show View**.

We use notes, cautions, and warnings to highlight important messages:

Notes point out something important or useful.

© 2020, QNX Software Systems Limited

CAUTION: Cautions tell you about commands or procedures that may have unwanted or undesirable side effects.

DANGER: Warnings tell you about commands or procedures that could be dangerous to your files, your hardware, or even yourself.

Note to Windows users

In our documentation, we typically use a forward slash (/) as a delimiter in pathnames, including those pointing to Windows files. We also generally follow POSIX/UNIX filesystem conventions.

© 2020, QNX Software Systems Limited

Technical support

Technical assistance is available for all supported products.

To obtain technical support for any QNX product, visit the Support area on our website (*www.qnx.com*). You'll find a wide range of support options, including community forums.

© 2020, QNX Software Systems Limited

Chapter 1
Before You Begin

Before you begin working with this BSP, you should become familiar with the resources available to you to work with QNX embedded systems.

Essential information

Before you begin to install or build your BSP, review the following documentation:

- Information about your board's hardware and firmware provided by the board vendor. This differs for each board.
- *Building Embedded Systems*. This guide contains general information about working with BSPs from QNX Software Systems that's common to all BSPs and is included in QNX Software Development Platform documentation.

Required software

To work with the BSP for the x86_64 board, you require the following:

- the ZIP file for the BSP
- QNX Software Development Platform installed on a Linux, macOS, or Windows host system
- Virtual COM Port (VCP) Driver. This driver is required on your host system if your board has a COM or USB-Serial port for debugging. To get the driver and read more information about it, see the Future Technology Devices International Ltd. (FTDI) website at *http://www.ftdichip.com/FTDrivers.htm*
- a terminal emulation program (e.g., Qtalk, Hyperterminal, Tera Term, PuTTy) to connect to the board for debugging. Alternatively, you can use a `telnet` or `ssh` session from the QNX Momentics IDE.

© 2020, QNX Software Systems Limited

Chapter 2
About This BSP

These notes list what's included in the Board Support Package (BSP), and identify the host operating systems and the boards the BSP supports.

Embedding QNX SDP 7.0 on targets

QNX SDP 7.0 (like QNX SDP 6.6) doesn't support a self-hosted option. If you've worked with previous QNX SDP versions, such as QNX SDP 6.5.x, which supported a self-hosted development environment, the self-hosted option isn't available.

QNX Software Systems provides a virtual machine image (**.iso**) that allows you to run on VMware software. This image provides a collection of the standard x86_64-based tools and utilities to explore the QNX Neutrino RTOS. The tools provided with this virtual machine can be used to put QNX Neutrino on an actual target. For more information about using the virtual machine, see "*Create a bootable USB device for BIOS boot mode (interactive method using VMware)*" in the "*Installation Notes*" chapter.

You can still install QNX SDP 7.0 on a PC, but an installation CD-ROM to configure a self-host, GUI-hosted development is no longer provided. QNX Neutrino RTOS 7.0 on x86- and x86-64-based hardware is treated as target and no different from an embedded target.

Why a generic x86 or x86_64 BSP?

Unlike other QNX SDP 7.0 BSPs, which are targeted to a specific board, the BSP isn't intended to be an exact match to any particular x86_64 (64-bit) platform. That being said, we do in some cases provide steps for some specific boards.

There are many different models of x86_64-based PCs and targets, each with different chipsets, and each with its own collection of peripheral devices, such as audio interfaces, network devices, and graphics controllers. QNX SDP 7.0 includes startup modules, device drivers, and general support for a wide variety of different x86_64 chipsets and peripherals. In most cases, these components can be run as-is on x86_64 platforms; it's simply a matter of selecting the correct device drivers and utilities for the specific hardware, and creating a custom bootable image for that board. In case there is modification required, this BSP provides the source code for startup modules and device drivers, which you can modify as needed.

If you want to include the Screen Graphics Subsystem (Screen) with your image, you must know the variant of the x86_64 platform that you're using. In many cases, you must modify the configuration files to run Screen. For more information, see the "*Using the Screen Graphics Subsystem*" chapter in this guide.

This BSP user guide provides instructions for:

- generating a QNX Image File System (IFS) bootable image for your target
- preparing a bootable drive (USB memory stick or USB hard drive) you can use to boot your target
- transferring your IFS image to your target

© 2020, QNX Software Systems Limited

What's in this BSP

This BSP contains the following components:

Component	Format	Comments
Startup	Source	
Serial or USB-serial driver	Binary	
PCI	Binary	PCI server
Network driver	Binary	Intel Gigabit or Realtek RTL8169 Ethernet driver
USB Host Controller	Binary	USB (host mode) drivers are provided in binary format as part of QNX SDP.
SATA	Binary	
Fast IDE (EIDE)	Binary	
Screen Graphics Subsystem sample runtime environment	Configuration and buildfiles	Systems based on Ivy Bridge, Sandy Bridge, Haswell, Broadwell, and Skylake processors
SMMU Manager	Binary	Makes use of he DMA containment and memory management support available for he ARM architecture. The libraries and header files are part of QNX SDP (QNX Neutrino 7.0.4).
Audio driver	Binary	Driver for Intel HD audio interface and mixer (codec support) or Ensoniq or Creative Labs audio controllers

Binaries for the serial driver, PCI server, network driver, configuration files for the Screen Graphics Subsystem, and audio driver are provided as part of the QNX SDP 7.0 installation.

Supported OSs

The x86 variant of the BSP supports QNX Neutrino RTOS 7.0 on x86 (32-bit) and x86_64 variant of the BSP supports x86_64 (64-bit) platforms.

Before you install and use this BSP, you must install the QNX Software Development Platform (QNX SDP) on your host system. You can install the QNX SDP on a Windows, macOS, or Linux host system.

Supported boards and chipsets

In the QNX Software Center, see the release notes for this BSP for the list of supported boards using these steps:

1. On the **Available** or **Installed** tab, navigate to Board Support Packages BSP, right-click *Name_of_your_BSP*, and then select **Properties**.
2. In the **Properties** window, on the **General Information** pane, click the link beside **Release Notes**.

© 2020, QNX Software Systems Limited

You should be redirected to the release notes on the QNX website as long as you're logged in with your myQNX account.

Known issues

For the list of known issues, see the release notes for this BSP.

© 2020, QNX Software Systems Limited

Chapter 3
Installation Notes

These installation notes describe how to install and boot a board using this BSP.

About installing a BSP

Here's the process to install the BSP and run it on your board:

1. Download the BSP, set up your environment, and extract the BSP. For more information, see "*Download and set up the BSP*."
2. Connect your target board. After you connect the hardware, it may be necessary set DIP switches on your board to further configure it. For more information about how to connect to your board, see "*Working with the hardware*."
3. Create and transfer an image to a bootable USB device. If this is the first time you're bringing up the BSP, you can transfer the provided IFS image file to the USB device to boot your board. You can either transfer a bootable USB image (offline method) or create a bootable USB using VMware.

   To prepare a bootable USB device, you must first connect the USB device (such as a USB memory stick or a USB hard drive) to the host computer where you installed QNX Software Development Platform. After the USB device has been prepared with a bootable QNX partition and IFS image, you can insert it into a USB port on the x86_64 target, and use it to boot that target to run the QNX Neutrino RTOS. At this point, if you want to configure a different boot device on the x86_64 target as the primary boot device, you can use the tools and utilities included in your bootable image to do so.

   Depending on the method you use to build the BSP, follow the instructions in either "*Create a bootable USB device for BIOS boot mode (interactive method using VMware)*" or "*Create and transfer a bootable USB image (offline method)*." The two methods differ in how the bootable USB device is created, and how the IFS image is transferred to the USB device.
4. After you have your image on the card, you can boot it. For more information about booting your board, see "*Boot the target*."

After your board boots, you can use that image to understand how it works. Depending on the other devices you've connected to the board, you may need to modify configuration settings on the image.

At some point, you may want to customize the image, rebuild it, and then put the image onto a bootable USB stick. For information about how to create a bootable USB stick and how to customize the image, see the *Building Embedded Systems* guide, which is available in the QNX SDP documentation. For specific information about building the image for this board, see the "*Build the BSP*" chapter in this guide.

© 2020, QNX Software Systems Limited

Download and set up the BSP

You can download Board Support Packages (BSPs) using the QNX Software Center.

BSPs are provided in a ZIP archive file. You must use the QNX Software Center to install the BSP, which downloads the BSP archive file for you into the **bsp** directory located in your QNX SDP installation. To set up your BSP, you can do one of the following steps:

- Import the BSP archive file into the QNX Momentics IDE (extracting the BSP isn't required)
- Navigate to where the BSP archive file has been downloaded, and then extract it using the `unzip` utility on the command line to a working directory that's outside your QNX SDP installation.

The `unzip` utility is available on all supported host platforms and is included with your installation of QNX SDP.

The QNX-packaged BSP is independent of the installation and build paths, if the QNX Software Development Platform is installed. To determine the base directory, open a Command Prompt window (Windows) or a terminal (Linux and macOS), and then type `qconfig`.

Extract from the command line

We recommend that you create a directory named after your BSP and extract the archive from there:

1. In a Command Prompt window (Windows) or a terminal (Linux and macOS), run **qnxsdp-env.bat** (Windows) or `source` **qnxsdp-env.sh** (Linux, macOS) from your QNX SDP installation to set up your environment. You must run the shell or batch file to use the tools provided with QNX SDP.

   

   If you type `env`, you should see environment variables that are set, such as *QNX_TARGET*.

2. Navigate to the directory where you want to extract the BSP (e.g., `cd` **/BSPs/bsp_x86_64-generic**). The directory where you extracted the BSP is referred to throughout this document as *bsp_working_dir*. The archive is extracted to the current directory, so you should create a directory specifically for your BSP. For example:

   `mkdir` *bsp_working_dir*

   You should also set the *BSP_ROOT_DIR* environment variable to point to your *bsp_working_dir*. You can use the `export BSP_ROOT_DIR=`*bsp_working_dir* (Linux and macOS) or `set BSP_ROOT_DIR=`*bsp_working_dir* (Windows) command to set the environment variable.

3. In the directory you created in the previous step, use the `unzip` command to extract the BSP. For example:

   `unzip -d` *bsp_working_dir* **BSP_x86_64-generic-700_SVN***build_ID***.zip**

   where *build_ID* is the BSP build number (e.g., `810350_JBN17`).

You should now be ready to build your BSP (see "*Build the BSP*" chapter).

© 2020, QNX Software Systems Limited

Import to the QNX Momentics IDE

If you use the QNX Momentics IDE to build your BSP, you don't need to extract the ZIP file. To import the BSP code into the IDE:

1. Open the QNX Momentics IDE.
2. From the C/C++ Perspective, select **File → Import**.
3. Expand the **QNX** folder.
4. Select **QNX Source Package and BSP (archive)** from the list and then click **Next**.
5. In the **Select the archive file** dialog, click **Browse...**.
6. In the Open dialog box, navigate to and select the BSP archive file that you downloaded earlier, and then click **Open** and then click **Next**.
7. Confirm the BSP package you want is shown in the **Selected Package** list and then click **Next** to proceed importing the BSP archive file.
8. Optionally, set the **Project Name** and **Location** to import the BSP archive file. Defaults are provided for the project name and location, but you can override them if you choose.
9. Click **Finish**. The project is created, and the source brought from the archive.

You should see the project in the **Project Explorer** view (e.g., **bsp-x86_64-generic**). You can now build your BSP. For more information, see the "*Build the BSP (IDE)*" section in the "*Build the BSP*" chapter of this guide.

© 2020, QNX Software Systems Limited

Structure of a BSP

After you unzip a BSP archive using the command line, or if you look at the source that's extracted by the IDE in your workspace, the resulting directory structure looks something like this:

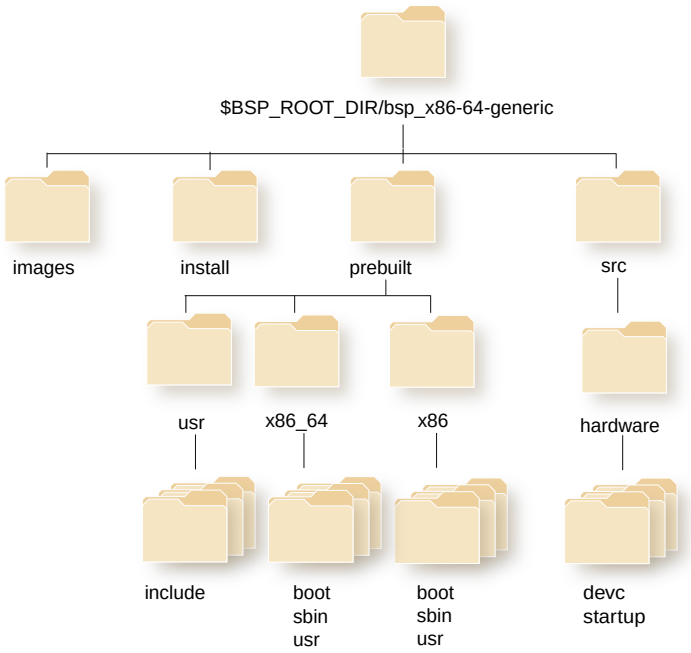

Figure 1: Structure of a BSP

For more information about the contents of the directory structure, see the *Building Embedded Systems* guide in the QNX SDP documentation.

© 2020, QNX Software Systems Limited

Working with the hardware

You must attach the appropriate connectors to your hardware.

Since this is a generic BSP guide, you must refer to the hardware guide provided by your target board provider for more information. Here are things that you should consider:

- How to connect to power and the proper power supply to use.
- The location of the USB ports, console, USB serial ports, Ethernet port, or other connectors such as a parallel port for a camera.
- Whether any DIP switches on the board need to be set. Some boards require certain DIP switches to boot from USB or even other storage devices, such as Flash memory on the board.
- How to configure the BIOS setting to boot from the proper device. For example, if you're booting from a USB device, you may need to modify your BIOS settings so that it boots from the USB device.

Connect serial device driver

Most modern targets provide a USB serial device driver that you connect to the board. To use this driver, you require a USB serial device driver on your host development machine to establish a terminal session with the target. If your board supports an embedded serial-USB conversion chip, you need to install an FTDI driver on your host system to use it.

You can download the appropriate device driver for your host system from the Future Technology Devices International Ltd. (FTDI) website at *http://www.ftdichip.com/FTDrivers.htm*.

Here are some typical settings for the target board. You should refer to the target board documentation the appropriate information:

On your host machine, using a terminal software and the appropriate COM port, these are typical settings that can be used with recent x86_64-based boards:

- Baud rate: 115200
- Data: 8 bit
- Stop: 1 bit
- Parity: no
- Flow control: none

For information about your particular board, see the platform or user guide for your board. These guides are usually available on the vendor's website.

© 2020, QNX Software Systems Limited

Create and transfer an image to a bootable USB device

You can transfer the provided prebuilt IFS image to your USB device.

Alternatively, you can build the BSP to create an IFS with changes you make to the buildfile. For more information about building this BSP, see the "*Build the BSP*" chapter in this guide.

To create and transfer a bootable image on a USB device, use one of the following methods:

Offline method

This method allows you to use host tools to build a USB disk image (which is bootable). You can later flash that USB image to a USB device. This option has the advantage that you can do everything using development tools provided with your SDP installation. For information about how to create a USB image and copy it to your USB device, see "*Create and transfer a bootable USB image (offline method)*."

Interactive method

This method requires that you use a virtual machine running the QNX Neutrino RTOS. Users who are familiar with using the self-hosted option can use this method. This method has the advantage that you can quickly copy a new IFS image onto the bootable device. For information about creating a bootable USB and then copying the IFS image to the USB device, see "*Create a bootable USB device for BIOS boot mode (interactive method using VMware)*."

This mechanism requires that you use a virtual machine running the QNX Neutrino RTOS. To download a virtual machine, use the QNX Software Center.

Create and transfer a bootable USB image (offline method)

To boot QNX Neutrino on your x86_64 target, you can create a bootable USB disk image that contains the IFS file. The USB image can be copied over to a USB device.

The offline method is the simplest approach to use the IFS image from your BSP because it requires that you use the host tools provided with your SDP 7.0 installation. This method requires that you do these tasks:

1. Create a USB disk image, which builds a BIOS or a UEFI disk image file (e.g., **bios_usb.img**, **uefi_usb.img**). For more information, see "*Create a USB disk image*."
2. Transfer the USB image to a USB device, such as a USB memory stick. For more information, depending on your host environment, see "*Transfer the USB image using host tools on Linux*", "*Transfer the USB image using host tools on macOS*", or "*Transfer the USB image using host tools on Windows*."

Create a USB disk image

You can either create a USB disk image for BIOS or UEFI boot mode because this BSP supports both boot modes. For more information, see " Transferring an OS image to an x86 target platform" in the "Working with BSPs" chapter of the *Building Embedded Systems* guide.

© 2020, QNX Software Systems Limited

Create a USB disk image for BIOS boot mode

To build a USB disk image for BIOS boot mode, in a terminal or Command Prompt on your host machine, navigate to the ***$BSP_ROOT_DIR*/images** directory and run the `make bios_disk_image` command. The `bios_mkusbimage` script does the following:

- calls the `mkqnx6fsimg` utility and a buildfile named **bios_root.build** to create a Power-safe filesystem image named **bios_root.img**.
- calls the `diskimage` utility using the configuration file called **bios_disk.cfg** and the IPL binary called `ipl-diskpcl` to create a bootable disk image named **bios_disk.img**

The buildfile **bios_root.build** configs the QNX6 filesystem and specifies the QNX IFS file under the **./boot** directory. By default, the **x86_64-generic.bin** file (QNX IFS) is used, which is provided with this BSP. If you want to use a different QNX IFS file to build your disk image, modify the **bios_root.build** buildfile and specify the name of your BIN file.

The name of QNX IFS you choose must be less than 27 characters in length and the QNX IFS file you specify can't be more than 16 MB in size.

Create a USB disk image for UEFI boot mode

To build a USB disk image for UEFI boot mode, in a terminal or Command Prompt on your host machine, navigate to the ***$BSP_ROOT_DIR*/images** directory and run the `make uefi_disk_image` command. The `make uefi_disk_image` calls the `uefi_mkusbimage` script to create the **uefi_disk.img** file, which is a disk image file that includes a Power-safe (**fs-qnx6.so**) filesystem and a bootable DOS filesystem. The `uefi_mkusbimage` script does the following:

- calls the `mkqnx6fsimg` utility and a buildfile named **uefi_root.build** to create a Power-safe filesystem image named **uefi_root.img**
- calls the `mkfatfsimg` utility with a buildfile called **uefi_boot.build** to create a bootable DOS file system image called **uefi_boot.img**
- calls the `diskimage` utility using the configuration file called **uefi_disk.cfg** to create a bootable disk image named **uefi_disk.img**

The buildfile **uefi_boot.build** configures the DOS filesystem and specifies the bootfile as **EFI/BOOT/BOOTX64.EFI**. By default, the **x86_64-uefi.efi** file (QNX IFS) is used, which is provided with this BSP. If you want to use a different QNX IFS file to build your disk image, modify the **uefi_boot.build** buildfile and specify the name of your EFI file.

The steps to create a bootable USB device using host tools (offline method) are as follows:

1. Navigate to the *$BSP_ROOT_DIR*`/images` directory. If you're in the BSP root directory (*BSP_ROOT_DIR*):

   ```
   cd images
   ```

2. Create the USB disk image (**bios_disk.img** or **uefi_disk.img**) using the following `make` commands:

   ```
   make bios_disk_image
   ```

   or

   ```
   make uefi_disk_image
   ```

© 2020, QNX Software Systems Limited

After you build your bootable USB images, transfer the image to a USB device. For more information, depending on your host environment, see "*Transfer the USB image using host tools on Linux* ", "*Transfer the USB image using host tools on macOS*", or "*Transfer the USB image using host tools on Windows*."

Transfer the USB image using host tools on Linux

On a Linux system, you transfer the bootable image to your USB device from the command line. Use these steps, where *path* is the path to your USB device, such as **/dev/usb0**:

- If your version of Linux supports the `sudo` command, enter:

  ```
  sudo dd if=bios_disk.img of=path
  ```

  or

  ```
  sudo dd if=uefi_disk.img of=path
  ```

- If your version of Linux does *not* support the `sudo` command, enter:

  ```
  su -c dd if=bios_disk.img of=path
  ```

  or

  ```
  su -c dd if=uefi_disk.img of=path
  ```

When the copy operation is complete, your USB device should be ready to *boot QNX Neutrino* on your x86-based target.

Transfer the USB image using host tools on macOS

On a macOS system, you transfer the bootable image to your USB device from the command line. Use these steps, where *path* is the path to your USB device, such as **/dev/usb0**:

- `sudo dd if=bios_disk.img of=path bs=1024k`

  or:

  ```
  sudo dd if=uefi_disk.img of=path bs=1024k
  ```

Transfer the USB image using host tools on Windows

On a Windows system, to transfer the bootabe image to your USB device, use these steps:

1. Download and install Win32 Disk Imager from *http://sourceforge.net/projects/win32diskimager/*
2. Insert your USB device, such as a USB stick into your host computer.
3. Run Win32 Disk Imager.
4. In the Win32 Disk Imager, in the Image File box, browse to the location where your image file (**bios_disk.img** or **bios_disk.img**) is located and click **Open**. The selected image file appears in the box.
5. Select the letter representing your USB device.
6. Click **Write** and click **Yes** to overwrite the contents on your USB device.

When the copy operation is complete, your USB device should be ready to *boot QNX Neutrino* on your x86-based target.

© 2020, QNX Software Systems Limited

Create a bootable USB device for BIOS boot mode (interactive method using VMware)

To boot QNX Neutrino RTOS on your x86_64 target, you can set up a USB device separately and then copy the bootable image (.bin) that you built earlier.

For UEFI boot mode, you simply format a USB stick as a DOS-formatted partition and copy the **x86_64-uefi.efi** to the DOS filesystem as the specific UEFI boot file **EFI/BOOT/BOOTX64.EFI**.

You can use these steps to partition a USB device, such as a USB storage key, format the partition, and set up the boot loader to make the USB device bootable. After your USB device is ready, copy the bootable image (**.bin**) file to the USB device. To use this method, you should follow these steps:

1. Create a bootable USB device. Typically, this mechanism needs to be done once. For more information, see " *Create a bootable USB device using a virtual machine running QNX Neutrino RTOS*."
2. Copy the IFS file to the bootable USB device. For more information, see "*Copy the IFS image to the bootable USB device*."

CAUTION: It's important to mention that if you have any data on this USB device (even in separate partitions), it'll be lost after completing this procedure.

Create a bootable USB device using a virtual machine running QNX Neutrino RTOS

To complete the steps, you must download and run the VMware image running QNX Neutrino RTOS (referred to as simply VMware here).You can use the virtual image of the QNX Neutrino RTOS. You also require a USB device, such as a USB memory stick or USB storage key. The size of the storage should be at least 4 GB.

The steps for creating a bootable USB device using the interactive method using a virtual machine are as follows:

1. Download the QNX SDP 7.0 runtime for VMware, install and run the ISO in VMware on your host environment. For more information, see the installation note for the *QNX SDP 7.0 x86-64 virtual machine for VMware*.

   After you boot the virtual machine, you're running a QNX Neutrino environment.
2. Insert the USB device into an unused USB port on your host PC. Keep in mind, this is the USB device you intend to use for booting your x86_64 target.
3. Your virtual machine software, such as VMware Workstation, should detect the drive, and prompt you to connect it either to your host computer or to the virtual machine. In the prompt, select **Connect to Virtual Machine**.
4. In QNX Neutrino, start the **devb-umass** driver. This driver supports generic USB mass storage devices, such as USB memory sticks and hard drives. When you run the driver, you should see output on your VMWare image that's similar to the following:

   ```
   # devb-umass
   # Path=0 - QNX USB Storage
   ```

© 2020, QNX Software Systems Limited

```
target = 0 lun = 0 Direct Access(0) - Kingston DataTraveler G3
#
```

For information about the devb-umass, see the QNX Neutrino *Utilities Reference*.

5. Run the following command to determine the drive number of the USB device. By default, the virtual drive containing the QNX SDP 7.0 runtime for VMware (or virtual machine) is **/dev/hd0t179**.

   Your USB drive is likely **/dev/hd1**. If the drive already has an existing filesystem partition on it, this partition will be represented as **/dev/hd1t***nn*, where "nn" represents the type of partition (e.g. 11 or 12 is a DOS FAT32; 177, 178, or 179 is the QNX6 filesystem, etc.).

   

   Don't worry if there's no partition there. Any existing partitions are deleted when the new partition is created in the next step.

   ```
   # ls /dev/hd*
   /dev/hd0 /dev/hd1
   /dev/hd0t179 /dev/hd1t11
   ```

6. Run the following commands to delete any existing partitions on the USB drive, add a type 177 parition into slot 1, and make partition slot 1 bootable using the `fdisk` command:

   

   CAUTION: The first command deletes all partitions on your USB drive and the second command creates a new partition!

   ```
   # fdisk /dev/hd1 delete -a
   ... [output from running command]...
   # fdisk /dev/hd1 add -s1 -t177
   ...[output from running commmand]...
   # fdisk /dev/hd1 boot -s1
   ```

7. Slay and restart the **devb-umass** driver, to enumerate the type 177 partition that was created in the previous step:

   ```
   # slay devb-umass
   # devb-umass
   # Path=0 - QNX USB Storage
   target = 0 lun = 0 Direct Access(0) - Kingston DataTraveler G3
   # ls /dev/hd*
   /dev/hd0 /dev/hd1
   /dev/hd0t179 /dev/hd1t177
   ```

8. Format the type 177 partition with a Power-Safe filesystem, using the **mkqnx6fs** utility, as follows:

   ```
   # mkqnx6fs /dev/hd1t177
   All files on /dev/hd0t177 will be lost!
   Confirm filesystem re-format (y) or (n): y
   Format fs-qnx6: 958712 blocks, 119840 inodes, 4 groups
   #
   ```


© 2020, QNX Software Systems Limited

9. Mount the drive, so that you'll be able to copy the bootable image to it:

```
# mount -t qnx6 -o sync=optional /dev/hd1t177 /usb
```

You now have a bootable USB image that you can transfer to a USB device.

Copy the IFS image to the bootable USB device

After you've created a bootable USB device, you need to transfer the QNX IFS bootable image that you built using this BSP.

To transfer the QNX IFS image to your bootable USB device, you must transfer the bootable image from your QNX SDP 7.0 host environment to the VMware runtime environment. To do this, use a second DOS-formatted USB device:

1. Insert the DOS-formatted USB stick into a USB port on your Linux, macOS, or Windows host, and copy the file (**x86_64-generic.bin**) from the ***$BSP_ROOT_DIR*/images** directory to the second USB stick.

WARNING: Ensure that the USB stick is connected to the host machine instead of the virtual machine running QNX Neutrino.

2. On your host, reconnect the same USB stick to the virtual machine running by doing one of the following steps:
   - remove and re-insert the USB stick. When the prompt from your VMware software appears, connect the USB stick to the virtual machine instead of the host machine
   - In VMware, select **VM → Removable Devices → [*name of your USB device*] → Connect (Disconnect from Host)** menu to connect the USB stick to the virtual machine running the QNX Neutrino, where *name of your USB device* is a string name matching your device. For example, Kingston USB device.
3. After the USB stick containing the QNX IFS image is connected to the VMware image running QNX Neutrino, you can mount the DOS filesystem (Type 12) using these steps (this example assumes that the USB stick has appeared as **/dev/hd2t12**):

```
# mount -t dos /dev/hd2t12 /stick
# cd /stick
# ls
```

4. You should see the **x86_64-generic.bin** file. Copy this file to the bootable USB drive with a Power-Safe (**fs-qnx6.so**) filesystem (using this command:

```
cp -V /stick/x86_64-generic.bin /usb/.boot/x86_64-generic.bin
```

When the copy operation completes, your USB drive is ready to *boot QNX Neutrino* on your x86_64 target.

© 2020, QNX Software Systems Limited

Boot the target

After you've transferred your image to a bootable USB device, you can use it to boot the QNX Neutrino RTOS on the system.

To boot QNX Neutrino on your x86_64-based target:

1. Remove the USB device from your host machine, and insert it into a USB port on your x86_64 target.
2. Turn on the power for your x86_64 board, and enter the BIOS, to configure the machine to boot from the USB device you're using.
3. After you configure the BIOS to use the USB device as the primary boot device, reboot the target.

You should see your target boot the IFS image. At this point, you can edit the default buildfile in the BSP to customize the boot image to meet the specific hardware and peripherals on your target, rebuild your customized image (IFS file), and put the new image on your USB device.

© 2020, QNX Software Systems Limited

Chapter 4
Using the Screen Graphics Subsystem

The Screen Graphics Subsystem (Screen) environment includes the libraries, services, and tools required to run graphics applications on x86 and x86_64 target boards with supported graphics processors, and to manage graphics shown on the display.

Since this generic BSP can support a variety of chipsets and GPUs, Screen might not initially work on your board when you use the image provided with this BSP. If this occurs, you'll need to modify the settings in the graphics configuration file to match your board to get Screen working. If you're using a board with Intel-based GPUs, the buildfile we provide usually works with minimal changes. To confirm whether Screen is running properly on your target, we provide demo applications with this BSP that you can run on the target. After you've confirmed that the graphics configuration file changes work on your board, you can rebuild the IFS to include your changes.

The **x86_64-generic.build** file (or **x86-generic.build** file for x86 targets) that's provided with this BSP doesn't start Screen on the QNX IFS image. To enable graphics, use the **x86_64-nuc-graphics.build** file (or **x86-nuc-graphics.build** file for x86 targets) to start Screen.

To get the correct binaries and graphics configuration file for this board, you must install the Screen Board Support package for this board from the QNX Software Center. For information about how to determine the Screen Board Support package to install for this board, see the Screen release notes. After you install the Screen Board Support package, the ***$QNX_TARGET*/x86_64/usr/lib/graphics/intel-drm** directory in your host installation will contain the drivers and graphics configuration you require. It's important to note that these binaries are necessary for you to build this BSP with graphics.

For more information about the graphics configuration file, see the *Screen Developer's Guide*.

Supported boards

Screen requires x86 or x86_64-based targets, such as Sandy Bridge, Ivy Bridge, Haswell, Broadwell, or Skylake chipsets, which we have validated using Intel NUC Kits. For more information about the kits, see the Intel NUC Kits at *http://ark.intel.com/products/series/70406/Intel-NUC-Boards-and-Kits*. For information about the supported chipsets and GPUs for Screen, see the Screen release notes in the QNX Software Center.

Graphics configuration

The specifics of a board's graphics configuration, such as the target output display port or the output resolution, are defined in the graphics configuration file (**graphics-*platform_name*.conf**), which is located at ***$QNX_TARGET*/x86_64/usr/lib/graphics/intel-drm** in your SDP installation and gets renamed to **/usr/lib/graphics/intel-drm/graphics.conf** on your target. We provide separate buildfiles (e.g., **x86_64-graphics-nuc.build**) that include the graphics-related components in the build. The graphics versions of the buildfiles include multiple chipset-specific versions of the graphics configuration file, identified by the name convention, **graphics-*platform name*.conf**. For example, **graphics-nuc-NUC6i5SYK.conf** is for the Skylake chipset.

© 2020, QNX Software Systems Limited

Before you use these buildfiles, you must uncomment the **graphics-*platform_name*.conf** file that matches the chipset on your target and ensure that the other entries are commented. For example, if you wanted to build an IFS for a target with the Skylake chipset, you would comment out all the entries in the buildfile except for the entry for the Skylake graphics configuration file:

```
...
...
#-------------------------------------------------------------------------------
# Graphics Conf - uncomment the graphics.conf file for the specific Intel NUC
#-------------------------------------------------------------------------------

# Intel - Skylake - Model # NUC6i5SYK
/usr/lib/graphics/intel-drm/graphics.conf=${QNX_TARGET}/x86_64/usr/lib/graphics/
intel-drm/graphics-nuc-6i5SYK.conf

# Intel - Haswell - Model # D34010WYK
#/usr/lib/graphics/intel-drm/graphics.conf=${QNX_TARGET}/x86_64/usr/lib/graphics/
intel-drm/graphics-nuc-D34010WYK.conf

# Intel - Broadwell - Model # NUC5i5MYHE
#/usr/lib/graphics/intel-drm/graphics.conf=${QNX_TARGET}/x86_64/usr/lib/graphics/
intel-drm/graphics-nuc-5i5MYHE.conf
...
...
```

In addition, included with this BSP is the *drm-probe-displays* utility that you can run on a target to determine the necessary settings to use in the **graphics.conf** file. You can also use the `drm-probe-displays` utility to validate that configuration settings are correct. For more information about the `drm-probe-displays` utility, see the "*drm-probe-displays*" section in this chapter. The **graphics.conf** file can be modified at runtime, but Screen needs to be restarted. For example:

```
# slay screen
# screen
```

For more information about using Screen, see the *Screen Developer's Guide* in the QNX Software Development Platform documentation.

Configure the display for Screen

In the **graphics.conf** file, the *video-mode* setting must match the screen resolution and frame rate supported by the display that's connected to your board. Screen reads these parameters from the **graphics.conf** file to initialize the display drivers. These are the resolution and frame rates that the board supports, and these can be found in the specific graphics configuration file that matches your board.

For example, for the Skylake chipset (**graphics-nuc-6i5SYH.conf** on your SDP installation), the default is for a 1920 x 1080 at 60 Hz display and has `video-mode` in the configuration file set as follows:

```
...
...
begin display 1
```


© 2020, QNX Software Systems Limited

```
    video-mode = 1920 x 1080 @ 60
    # Adjust the stack size of Screen's composition thread; required when the
    # display's framebuffer uses Mesa (e.g., "usage = gles2"), as noted above.
    stack-size = 65536  # in units of bytes
  end display
...
...
```

For more information about configuring the `display` subsection in the graphics configuration file, see the "Configure display subsection" section in the *Screen Developer's Guide*.

Display drivers

The video display configuration library is located on the target at **/usr/lib/graphics/intel-drm/libWFDintel-drm.so**.

Running Screen applications

This BSP archive comes with built-in demo applications that show how to use Screen. It's a good idea to run these demos on your target to confirm that Screen is configured properly and all the necessary drivers are available for Screen to run. If the following demo applications run successfully, it's a good indication that Screen is properly configured on your target.

`sw-vsync`

Shows `vsync` that uses software rendering, but doesn't use GLES. This application is useful to test that basic Screen functionality works on your display.

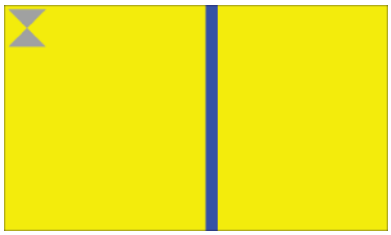

Figure 2: sw-vsync

`gles2-gears`

Shows gears that use OpenGL ES 2.X for the rendering API; if `gles2-gears` runs, then it confirms that `screen` and drivers necessary for OpenGL ES 2.x have started successfully.

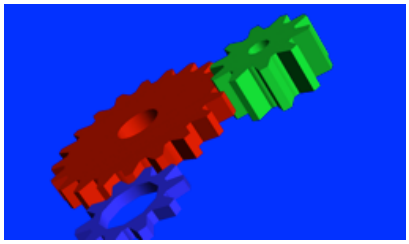

Figure 3: gles2-gears

For more information about using the demo applications or debugging them, see the "Utilities and Binaries" and the "Debugging" chapters in the *Screen Developer's Guide*, respectively.

© 2020, QNX Software Systems Limited

Create Screen applications

When you create an application that supports Screen, it must link to the Screen library (**libscreen.so**). This library provides the interfaces for applications to use the Screen API. Depending on the graphical requirements of your application, you may need to link in additional libraries. Below are some of the libraries commonly used when building applications that use Screen:

libEGL

The interface to access vendor-specific EGL functionality. Refer to the EGL standard from *www.khronos.org*.

libGLESv2

The interface to access vendor-specific OpenGL ES functionality. Refer to the OpenGL ES standard from *www.khronos.org*.

libimg

The interface to load and decode images. For more information, see the *Image Library Reference* guide in the QNX Software Development Platform documentation.

For more information about application development using Screen, see the *Screen Developer's Guide*. For information about linking libraries, see "QNX C/C++ Project properties" in the "Reference" chapter of the *QNX Momentics IDE User's Guide* or the "Compiling and Debugging" chapter of the *QNX Neutrino Programmer's Guide*.

© 2020, QNX Software Systems Limited

drm-probe-displays

Detect ports and pipelines for Intel graphics hardware (available for Intel platforms only)

Syntax:

```
drm-probe-displays
```

Runs on:

QNX Neutrino

Options:

None.

Description:

The `drm-probe-displays` utility is a command-line tool that detects ports and pipelines for Intel graphics hardware. This utility lists the available displays and pipelines.

You must have `drm-intel` running to use `drm-probe-displays`.

The information retrieved from this utility can be used to modify **graphics.conf** for a specific target. For more information about configuring **graphics.conf** for Intel graphics hardware, refer to the usage message of **libWFDintel-drm.so**, and to the *Screen Developer's Guide*.

To invoke `drm-probe-displays` on your target:

1. Start `drm-intel`.
2. Run `drm-probe-displays` .

Examples:

Query displays and pipelines on a x86-based platform for the NUC-based board:

```
# drm-probe-displays
count_displays : 3
count_pipelines: 2

display 1: VGA, connected
Mode: "1280x1024" 1280x1024 60
Mode: "1280x1024" 1280x1024 75
Mode: "1152x864" 1152x864 75
Mode: "1024x768" 1024x768 75
Mode: "1024x768" 1024x768 60
Mode: "800x600" 800x600 75
```

© 2020, QNX Software Systems Limited

```
Mode: "800x600" 800x600 60
Mode: "640x480" 640x480 75
Mode: "640x480" 640x480 60
Mode: "720x400" 720x400 70
display 2: HDMI-A, disconnected
display 3: displayport, connected
Mode: "1280x720" 1280x720 60
Mode: "1920x1080" 1920x1080 60
Mode: "1920x1080" 1920x1080 60
Mode: "1920x1080i" 1920x1080 60
Mode: "1920x1080i" 1920x1080 60
Mode: "1920x1080" 1920x1080 50
Mode: "1920x1080i" 1920x1080 50
Mode: "1280x1024" 1280x1024 75
Mode: "1440x900" 1440x900 85
Mode: "1440x900" 1440x900 75
Mode: "1440x900" 1440x900 60
Mode: "1280x720" 1280x720 60
Mode: "1280x720" 1280x720 50
Mode: "1024x768" 1024x768 75
Mode: "1024x768" 1024x768 70
Mode: "1024x768" 1024x768 60
Mode: "800x600" 800x600 75
Mode: "800x600" 800x600 72
Mode: "800x600" 800x600 60
Mode: "800x600" 800x600 56
Mode: "720x576" 720x576 50
Mode: "720x480" 720x480 60
Mode: "720x480" 720x480 60
Mode: "640x480" 640x480 75
Mode: "640x480" 640x480 73
Mode: "640x480" 640x480 60
Mode: "640x480" 640x480 60
Mode: "720x400" 720x400 70

pipeline 1
pipeline 2
```


© 2020, QNX Software Systems Limited

Chapter 5
Build the BSP

You can use the QNX Momentics IDE or the command line on Linux, Windows, or macOS to build an image.

Generic instructions on how to extract and build a BSP can be found in the *Building Embedded Systems* guide, available as part of the QNX Software Development Platform 7.0 documentation.

For instructions on how to build using the IDE, see "*Build the BSP (IDE)*." For generic information on how to build using the IDE, see the *IDE User's Guide* as part of the QNX SDP documentation.

If you plan to work with multiple, different BSPs, we recommend that you create a top-level BSP directory, then create subdirectories for each different BSP. The directory you create for each BSP will be that BSP's root directory (*BSP_ROOT_DIR*). This allows you to conveniently switch between different BSPs by simply setting the *BSP_ROOT_DIR* environment variable.

This BSP includes prebuilt IFS images that are provided as convenience for you to quickly get QNX Neutrino running on your board, however these prebuilt images might not have the same components as your development environment.

For this reason, we recommend that you rebuild the IFS image on your host system to ensure that you pick up the same components from your development environment.

To build your BSP, complete the following tasks:

1. Build the BSP, either using the command line or the QNX Momentics IDE.
2. If you aren't using the default IFS file provided with your board (e.g. **x86_64-generic.bin**), you can create a bootable USB image using your host or using a VMWare of the QNX Neutrino RTOS.

Once you've unzipped the BSP, you may notice prebuilt IFS files, such as **x86_64-generic.bin**, **x86_64-adi-rcc-c2000.bin**, **x86_64-nuc-graphics.bin**, **x86_64-uefi.efi**, **x86_64-ccm-denverton.efi**, **x86_64-ccm-denverton-smmu.efi**, **x86_64-sm-denverton.efi**, and **x86_64-sm-denverton-smmu.efi** in the ***$BSP_ROOT_DIR*/images** directory. You need to run `make clean` to remove these files before you build the BSP on the command line. You don't need to remove them if you use the IDE to build the BSP because the IDE does a `make clean` each time it builds.

Support for systems with or without Screen

Support for Screen is included in separate buildfiles found in the **/images** directory of the BSP. The default buildfile generates a target image without Screen; only the **x86_64-nuc-graphics.build** file generates an image with Screen. Before you can use the buildfile to generate the image files to boot your board with Screen, you may need to modify the buildfile. For information about how to modify the buildfile for the target, see "Graphics configuration" in the "*Using the Screen Graphics Subsystem*" chapter of this guide.

© 2020, QNX Software Systems Limited

Build the BSP (command line)

You can use the command line on Linux, macOS, or Windows to work with this BSP.

The **Makefile** found in the ***$BSP_ROOT_DIR*/images** directory defines the following targets:

Available for the 64-bit variant of the BSP

`all`

Builds all QNX IFS images (**x86_64-generic.bin**, **x86_64-adi-rcc-c2000.bin**, **x86_64-nuc-graphics.bin**, **x86_64-uefi.efi**, **x86_64-ccm-denverton.efi**, **x86_64-ccm-denverton-smmu.efi**, **x86_64-sm-denverton-smmu.efi**, and **x86_64-sm-denverton.efi**)

`x86_64-generic.bin`

Builds a QNX IFS image for a generic x86_64 platforms that's loaded with BIOS. The buildfile used to build this image is **x86_64-generic.build**.

`x86_64-uefi.efi`

Builds a QNX IFS image for a generic x86_64 platforms that loaded using UEFI. The buildfile used to build this image is **x86_64-uefi.build**.

`x86_64-nuc-graphics.bin`

Builds a QNX IFS image for an x86_64 platform and builds the necessary files required for the Screen Graphics Subsystem to run. This image is loaded using BIOS. The buildfile used to build this image is **x86_64-nuc-graphics.build**.

`x86_64-adi-rcc-c2000.bin`

Builds a QNX IFS image for x86_64 ADI RCC C2000 (Rangeley) platform that's loaded with BIOS. The buildfile used to build this image is **x86_64-adi-rcc-c2000.build**.

`x86_64-ccm-denverton.efi`

Builds a QNX IFS image for the x86_64 Intel Car Creek Module (Denverton SoC-based platform) that's loaded using UEFI. The buildfile used to build this image is **x86_64-ccm-denverton.build**.

`x86_64-ccm-denverton-smmu.efi`

Builds a QNX IFS image for the x86_64 Intel Car Creek Module (Denverton SoC-based platform) that's loaded using UEFI. This IFS image enables SMMU Manager on drivers that support SMMU Manager. The buildfile used to build this image is **x86_64-ccm-denverton-smmu.build**.

`x86_64-sm-denverton.efi`

Builds a QNX IFS image for the Intel Atom C3000 Supermicro platforms (Denverton SoC-based platform) that's loaded using UEFI. The buildfile used to build this image is **x86_64-sm-denverton.build**.

© 2020, QNX Software Systems Limited

x86_64-sm-denverton-smmu.efi

Builds a QNX IFS image for the Intel Atom C3000 Supermicro platforms (Denverton SoC-based platform) that's loaded using UEFI. This IFS image enables SMMU Manager on drivers that support SMMU Manager. The buildfile used to build this image is **x86_64-sm-denverton-smmu.build**.

bios_disk_image

Creates a BIOS bootable USB image using the `diskimage` utility. The QNX image file that's used to build the USB image is specified in **bios_root.build** file, which by default uses the **x86_64-generic.bin** QNX IFS image. If you want to use a different QNX IFS image, you must modify the file to specify the BIOS (***.bin**) QNX IFS image to use.

uefi_disk_image

Creates a UEFI bootable USB image using the `diskimage` utility. The QNX image file that's used to build the USB image is specified in the **uefi_boot.build**, which by default uses the **x86_64-uefi.efi** QNX IFS image. If you want to use a different QNX IFS image, you must modify the file to specify the UEFI (***.efi**) QNX IFS image to use.

Available for the 32-bit variant of the BSP

bios_disk_image

Creates a BIOS bootable USB image using the `diskimage` utility. The QNX image file that's used to build the USB image is specified in **bios_root.build** file, which by default uses the **x86-generic.bin** QNX IFS image. If you want to use a different QNX IFS image, you must modify the file to specify the BIOS (***.bin**) QNX IFS image to use.

x86-generic.bin

Builds a QNX IFS image for a generic x86 (32-bit) platforms that's loaded with BIOS. The buildfile used to build this image is **x86-generic.build**.

x86-adi-rcc-c2000.bin

Builds a QNX IFS image for x86 (32-bit) ADI RCC C2000 (Rangeley) platform that's loaded with BIOS. The buildfile used to build this image is **x86_64-adi-rcc-c2000.build**.

x86_nuc-graphics.bin

Builds a QNX IFS image for an x86 (32-bit) platform and builds the necessary files required for the Screen Graphics Subsystem to run. This image is loaded using BIOS. The buildfile used to build this image is **x86-nuc-graphics.build**.

If you don't specify a target, `make` invokes the `all` target.

Build from the command line

To build the BSP on your host system, in a Command Prompt window (Windows) or a terminal (Linux, macOS), perform the following steps:

1. Download the BSP archive, unzip it, and set up your environment to build. For more information, see "*Download and set up the BSP*" in the "*Installation Notes* chapter."

© 2020, QNX Software Systems Limited

2. In the BSP's root directory (*BSP_ROOT_DIR*), type `make clean` to remove the default image files from the ***$BSP_ROOT_DIR*/images** directory.

   ```
   > cd $BSP_ROOT_DIR
   > cd images
   > make clean
   ```

   This action removes all existing image files.

3. Navigate back to the *BSP_ROOT_DIR* and type `make` to build the BSP. When you run `make`, it does the following:

   - creates the **install** directory and builds the BSP
   - places any images required for the board into the ***$BSP_ROOT_DIR*/images** directory

   

   No buildfile is generated. The buildfiles are preloaded in the ***$BSP_ROOT_DIR*/images** directory

```
> cd $BSP_ROOT_DIR
> make
```

For the 64-bit variant of the BSP, you should now have IFS files called **x86_64-generic.bin**, **x86_64-adi-rcc-c2000.bin**, **x86_64-nuc-graphics.bin**, **x86_64-uefi.efi**, **x86_64-ccm-denverton.efi**, **x86_64-ccm-denverton-smmu.efi**, **x86_64-sm-denverton.efi**, and **x86_64-sm-denverton-smmu.efi**.

For the 32-bit variant of the BSP, you should now have IFS files called **x86-adi-rcc-c2000.bin**, **x86-nuc-graphics.bin**, and **x86-generic.bin**.

We recommend that you use the `make` command to build your IFS image. If you use the `mkifs` utility directly, you must ensure that you use the `-r` option to specify the location of the binaries.

© 2020, QNX Software Systems Limited

Build the BSP (IDE)

You can use the QNX Momentics IDE on Linux, macOS, or Windows, to work with this BSP.

Build in the QNX Momentics IDE

You can build the entire BSP in the QNX Momentics IDE, which includes building the source and associated images.

1. If you haven't done so, launch the IDE, and then import the BSP archive (which was downloaded for you using the QNX Software Center) into the IDE. For more information see "*Import to the QNX Momentics IDE*" in the "*Installation Notes*" chapter.

   After you've imported your BSP source, you're ready to build.

2. If you haven't already, switch to the C/C++ Perspective, then in the Project Explorer view, right-click the BSP source project (such as **bsp-x86_64-generic**) and select **Build Project**. This step builds the entire BSP, which includes the images and the binaries required for the images. You can check the progress and outcome of your build in the Console view.
3. If you want to modify the buildfile, open the **.build** file under the **System Builder Files** folder. After you've made your changes, right-click the BSP source project and select **Build Project**.

You can make changes to the buildfile in the **images** folder, see "*Building the **images** folder in the IDE*" section in this chapter.

Building the images folder in the IDE

You can make changes to the buildfiles in the **images** folder.

CAUTION: Changes you make to the buildfile in the **images** folder are overwritten if you build the entire BSP Project as described in "*Build the IDE*." Ensure that you save a backup copy of the buildfile elsewhere.

To build the changes you've made to a buildfile, create a *make target* in the **images** folder and then map it to an existing make target in the **Makefile** located in the **images** folder, which is equivalent to running the `make` command from the **images** directory on the command line.

For more information about creating make targets, see **Tasks > Building Projects > Creating a make target** in the *C/C++ development User Guide* in the IDE Help.

1. In the Project Explorer view, expand the **images** folder, right-click the IFS file you want recreated, and select **Delete**. The IFS must be deleted or it won't be rebuilt.
2. In the example view, right click the **images** folder and select **Build Targets → Create...**.
3. In the Create Build Target dialog box, type the name of an existing target in the **Makefile**, and then click **OK**.

   For example, if you wanted to build the default IFS file, **x86_64-generic.bin**, it maps to the x86_64-generic.bin target in the Makefile, you would type `x86_64-generic.bin` as the target. For more information on the targets available, see "*Build the BSP (command line)*."

© 2020, QNX Software Systems Limited

It's a good idea to create `clean` and `all` build targets, which run the `make clean` (to remove all created files in the **images** folder) and `make all` (to rebuild all the IFS files) commands, respectively.

4. In the Project Explorer view, under the **images** folder, you should see **Build Targets** created. Right-click **Build Targets**, select the build target that you created in the previous step, and click **Build**.

The IDE starts to build the image. You can look for any issues that may occur in the **Console** and **Problems** views. After your image is built, you should see your rebuilt **.bin** file beneath the **images** folder as shown in the following illustration:

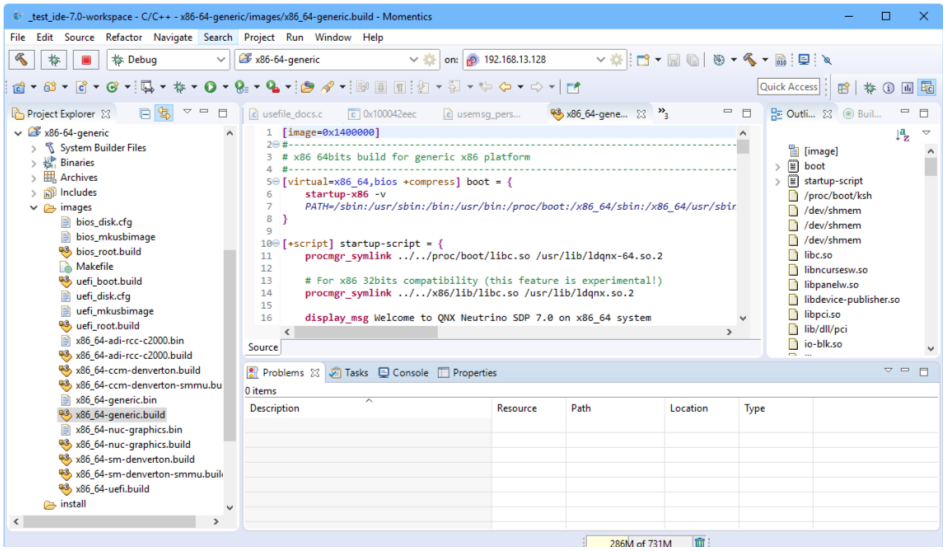

Figure 4: Build image in the IDE

You should now have an IFS file, such as **x86_64-generic.bin**. You can copy the file to a USB device that has been made bootable or you can build a USB image and flash a USB device with that image; see "*Create and transfer a bootable USB image (offline method)*." If your target can boot from a USB device, see "*Create a bootable USB device for BIOS boot mode (interactive method using VMware)*" to create a USB image. Both of these sections are in the "*Installation Notes*" chapter.

© 2020, QNX Software Systems Limited

Chapter 6
Generic x86_64 (64-bit) Driver Commands

The tables below provide a summary of driver commands.

Some of the drivers are commented out in the buildfile provided in the **images** directory of this BSP. To use these drivers on the target hardware, you may need to uncomment them in the buildfile, rebuild the image, and load the image onto the board.

The order that the drivers are started in the provided buildfile is one way to start them. The order that you start the drivers is completely customizable and is often specific to the requirements of your system. For example, if you need an audible sound to play immediately, you must start the audio driver earlier than other drivers. It's important to recognize that the order you choose impacts the boot time.

Some drivers depend on other drivers, so it's sometimes necessary to start them in a specific order because of these dependencies. For example, in the provided buildfile, the USB driver is started before the mass storage drivers because they rely on it.

- For more information on best practices for building an embedded system and optimizing boot times for your system, see the *Building Embedded Systems* and the *Boot Optimization* guides in the QNX Software Development Platform documentation.
- For more information about the drivers and commands listed here, see the QNX Neutrino *Utilities Reference*. This chapter provides information about BSP-specific options for any of the commands and drivers that aren't described in the *Utilities Reference*.

Here's the list of drivers available for this board and the order they appear in the provided buildfile:

Audio

Device	Intel HD audio controller
Command	`io-audio -d intel_hda`
Required binaries	`io-audio`, `wave`, `waverec`, `mix_ctl`
Required libraries	**libasound.so**, **deva-ctrl-intel_hda.so**, **deva-mixer-hda.so**, **libsecpol.so**,

© 2020, QNX Software Systems Limited

Source location	Prebuilt only

Device	Ensoniq or Creative Labs audio controller
Command	`io-audio -d audiopci`
Required binaries	`io-audio`, `wave`, `waverec`, **mix_ctl**
Required libraries	**libasound.so**, **deva-ctrl-audiopci.so**, **deva-mixer-ac97.so**
Source location	Prebuilt only

For more information about using the Intel HD audio driver, refer to the QNX Neutrino *Utilities Reference* entries for:

- **io-audio**
- **deva-ctrl-intel_hda.so**
- **deva-mixer-hda.so**

EIDE

Device	Fast IDE (EIDE)
Command	`devb-eide cam quiet blk automount=hd0t179:/:qnx6`
Required binaries	`devb-eide`
Required libraries	**io-blk.so**, **libcam.so**, **fs-qnx6.so**, **fs-dos.so**, **cam-disk.so**
Source location	Prebuilt only

Graphics

Device	GRAPHICS
Command	`screen`
Required binaries	`screen`, `gles1-gears`, `gles2-gears`, `sw-vsync`, `drm-probe-displays`, `drm-intel`, `screencmd`, `font-freetype`
Required libraries	For more information, see the *Screen Graphics Subsystem Developer's Guide*.
Required configuration files	**graphics.conf** and for its location in your QNX SDP installation, see the "*Using the Screen Graphics Subsystem*" chapter in this guide.
Source location	**N/A**

© 2020, QNX Software Systems Limited

Before starting Screen, you must set *LD_LIBRARY_PATH* environment variable as follows:

```
LD_LIBRARY_PATH=/usr/lib:/lib:/lib/dll:$LD_LIBRARY_PATH
```

Network

Device	NETWORK (Intel Gigabit E1000 Ethernet driver)

Commands	`io-pkt-v6-hc -d e1000`
	`if_up -r 10 -p wm0`
	`ifconfig wm0 up`
	`dhclient -m -lf /dev/shmem/dhclient.leases -pf /dev/shmem/dhclient.pid -nw wm0`

Required binaries	`io-pkt-v6-hc`, `ifconfig`, `dhclient`
Required libraries	**devnp-e1000.so**, **libsocket.so**, **libcrypto.so**
Source location	Prebuilt only

Device	NETWORK (Realtek RTL8169 Ethernet driver)

Command	`io-pkt-v6-hc -d rtl8169`
	`if_up -r 10 -p rt0`
	`ifconfig rt0 up`
	`dhclient -m -lf /dev/shmem/dhclient.leases -pf /dev/shmem/dhclient.pid -nw rt0`

Required binaries	`io-pkt-v6-hc`, `ifconfig`, `dhclient`
Required libraries	**devnp-rtl8169.so**, **libsocket.so**, **libcrypto.so**
Source location	Prebuilt only

PCI Server

Use `pci-server`. For more information, see the *PCI Server User's Guide* in the QNX SDP 7.0 documentation.

Device	Intel Host/PCI bridge
Environment variables	*PCI_HW_MODULE*, *PCI_SLOG_MODULE*, *PCI_DEBUG_MODULE*, *PCI_BASE_VERBOSITY*,

© 2020, QNX Software Systems Limited

	PCI_HW_CONFIG_FILE, For more information about the environment, see the "Environment variables" section in the Overview chapter of the *PCI Server User's Guide*.
Command	`pci-server -aspace-enable`
Required libraries	***$QNX_TARGET*/x86_64/lib/dll/pci**. Use **pc_hw_Intel_x86.so** for Intel-based platforms or **pci_hw_AMD_x86.so** for AMD-based platforms.
Source location	Prebuilt only

SATA

Device	SATA
Command	`devb-ahci cam quiet ahci nports=4 blk automount=hd0t179:/:qnx6`
Required binaries	`devb-ahci`
Required libraries	**io-blk.so**, **libcam.so**, **fs-qnx6.so**, **fs-dos.so**, **cam-disk.so**
Source location	Prebuilt only

Serial

CAUTION: You must have the *USB (Host Controller)* driver started before you start the USB serial driver.

Device	SERIAL (UART2, UART3)
Command	`devc-ser8250 -e -b115200` (Serial driver) `devc-serusb -e -b115200` (USB serial)
Required binaries	`devc-ser8250`, `devc-serusb`
Required libraries	Only Serial-USB: **libusbdi.so**, **libdevice-publisher.so**
Source location	Prebuilt only

Startup

Device	STARTUP
Command	`startup-x86`
Required binaries	`startup-x86`

© 2020, QNX Software Systems Limited

Required libraries	**libstartup.a**
Source location	***$BSP_ROOT_DIR*/src/hardware/startup/boards/x86**

USB (Host Controller)

Device	USB (Host driver)
Command	`io-usb-otg -d xhci -d ehci -d ohci -d uhci`
Required binaries	`io-usb-otg`, `usb`, `devb-umass`
Required libraries	**libusbdi.so**, **libhiddi.so**, **devu-hcd-ehci.so**, **devu-hcd-ohci.so**, **devu-hcd-uhci.so**, **devu-hcd-xhci.so**
Source location	Prebuilt only

© 2020, QNX Software Systems Limited

Chapter 7
Generic x86 (32-bit) Driver Commands

The tables below provide a summary of driver commands.

Some of the drivers are commented out in the buildfile provided in the **images** directory of this BSP. To use these drivers on the target hardware, you may need to uncomment them in the buildfile, rebuild the image, and load the image onto the board.

The order that the drivers are started in the provided buildfile is one way to start them. The order that you start the drivers is completely customizable and is often specific to the requirements of your system. For example, if you need an audible sound to play immediately, you must start the audio driver earlier than other drivers. It's important to recognize that the order you choose impacts the boot time.

Some drivers depend on other drivers, so it's sometimes necessary to start them in a specific order because of these dependencies. For example, in the provided buildfile, the USB driver is started before the mass storage drivers because they rely on it.

- For more information on best practices for building an embedded system and optimizing boot times for your system, see the *Building Embedded Systems* and the *Boot Optimization* guides in the QNX Software Development Platform documentation.
- For more information about the drivers and commands listed here, see the QNX Neutrino *Utilities Reference*. This chapter provides information about BSP-specific options for any of the commands and drivers that aren't described in the *Utilities Reference*.

Here's the list of drivers available for this board and the order they appear in the provided buildfile:

Audio

Device	Intel HD audio controller
Command	`io-audio -d intel_hda`
Required binaries	`io-audio`, `wave`, `waverec`, `mix_ctl`
Required libraries	**libasound.so**, **deva-ctrl-intel_hda.so**, **deva-mixer-hda.so**, **libsecpol.so**

© 2020, QNX Software Systems Limited

Source location	Prebuilt only

Device	Ensoniq or Creative Labs audio controller
Command	`io-audio -d audiopci`
Required binaries	`io-audio`, `wave`, `waverec`, **mix_ctl**
Required libraries	**libasound.so**, **deva-ctrl-audiopci.so**, **deva-mixer-ac97.so**
Source location	Prebuilt only

For more information about using the Intel HD audio driver, refer to the QNX Neutrino *Utilities Reference* entries for:

- **io-audio**
- **deva-ctrl-intel_hda.so**
- **deva-mixer-hda.so**

EIDE

Device	Fast IDE (EIDE)
Command	`devb-eide cam quiet blk automount=hd0t179:/:qnx6`
Required binaries	`devb-eide`
Required libraries	**io-blk.so**, **libcam.so**, **fs-qnx6.so**, **fs-dos.so**, **cam-disk.so**, cd
Source location	Prebuilt only

Graphics

Device	GRAPHICS
Command	`screen`
Required binaries	`screen`, `gles1-gears`, `gles2-gears`, `sw-vsync`, `drm-probe-displays`, `drm-intel`, `screencmd`, `font-freetype`
Required libraries	For more information, see the *Screen Graphics Subsystem Developer's Guide*.
Required configuration files	**graphics.conf** and for its location in your QNX SDP installation, see the "*Using the Screen Graphics Subsystem*" chapter in this guide.
Source location	**N/A**

© 2020, QNX Software Systems Limited

Before starting Screen, you must set *LD_LIBRARY_PATH* environment variable as follows:

```
LD_LIBRARY_PATH=/usr/lib:/lib:/lib/dll:$LD_LIBRARY_PATH
```

Network

Device	NETWORK (Intel Gigabit E1000 Ethernet driver)
Command	`io-pkt-v6-hc -d e1000`
Required binaries	`io-pkt-v6-hc`, `ifconfig`, `dhclient`
Required libraries	**devnp-e1000.so**, **libsocket.so**, **libcrypto.so**
Source location	Prebuilt only

Device	NETWORK (Realtek RTL8169 Ethernet driver)
Command	`io-pkt-v6-hc -d rtl8169`
Required binaries	`io-pkt-v6-hc`, `ifconfig`, `dhclient`
Required libraries	**devnp-rtl8169.so**, **libsocket.so**, **libcrypto.so**
Source location	Prebuilt only

PCI Server

Use `pci-server`. For more information, see the *PCI Server User's Guide* in the QNX SDP 7.0 documentation.

Device	Intel Host/PCI bridge
Environment variables	*PCI_HW_MODULE*, *PCI_SLOG_MODULE*, *PCI_DEBUG_MODULE*, *PCI_BASE_VERBOSITY*, For more information about the environment, see the "Environment variables" section in the Overview chapter of the *PCI Server User's Guide*.
Command	`pci-server -aspace-enable`
Required libraries	***$QNX_TARGET*/x86_64/lib/dll/pci**. Use **pc_hw_Intel_x86.so** for Intel-based platforms, or **pci_hw_AMD_x86.so** for AMD-based platforms.
Source location	Prebuilt only

© 2020, QNX Software Systems Limited

SATA

Device	SATA
Command	`devb-ahci cam quiet ahci nports=4 blk automount=hd0t179:/:qnx6`
Required binaries	`devb-ahci`
Required libraries	**io-blk.so**, **libcam.so**, **fs-qnx6.so**, **fs-dos.so**, **cam-disk.so**
Source location	Prebuilt only

SD/MMC

Device	SD/MMC
Command	SD card `devb-sdmmc cam quiet,pnp sdio idx=0,vid=0x8086,did=0x5acc`
Required binaries	`devb-sdmmc`
Required libraries	N/A
Source location	**src/hardware/devb/sdmmc**

Serial

CAUTION: You must have the *USB (Host Controller)* driver started before you start the USB serial driver.

Device	SERIAL (UART2, UART3)
Command	`devc-ser8250 -e -b115200` (Serial driver) `devc-serusb -e -b115200 -d path=/dev/usb/io-usb-otg` (USB serial)
Required binaries	`devc-ser8250`, `devc-serusb`
Required libraries	Only Serial-USB: **libusbdi.so**, **libdevice-publisher.so**
Source location	Prebuilt only

Startup

Device	STARTUP
Command	`startup-x86`

© 2020, QNX Software Systems Limited

Required binaries	`startup-x86`
Required libraries	**libstartup.a**
Source location	***$BSP_ROOT_DIR*/src/hardware/startup/boards/x86**

USB (Host Controller)

Device	USB (Host driver)
Command	`io-usb-otg -d xhci -d ehci -d ohci -d uhci`
Required binaries	`io-usb-otg`, `usb`, `devb-umass`
Required libraries	**libusbdi.so**, **libhiddi.so**, **devu-hcd-ehci.so**, **devu-hcd-ohci.so**, **devu-hcd-uhci.so**, **devu-hcd-xhci.so**
Source location	Prebuilt only

© 2020, QNX Software Systems Limited

Chapter 8
Working with ADI RCC C2000 Platforms

The ADI RCC C2000 platform (code name: Rangeley) uses many of the same source and binaries provided the generic BSP, however, there are a few differences.

- Connecting the power, the serial connection, and DIP switch settings are different from common x86 boards. For information, see "*Using the C2000 platform*."
- There's a separate buildfile, and a separate image is built for the C2000 Platform. For more information, see "*Modifying the IFS for C2000 platforms*."
- You must create a USB image as described in "*Create and transfer a bootable USB image (offline method)*" in this guide. For more information, see "*Creating a USB bootable image for the C2000 platforms*."

Using the C2000 platforms

For information about the hardware setup, jumper, and DIP switch settings, peripheral connections, etc., refer to the board's reference manual, which can be found at the following location: *www.adiengineering.com/products/rcc/*. The ADI RCC doesn't ship with an on-board graphics controller. You won't be able to render graphics using this platform, but you can still run the Screen Graphics Subsystem on the platform to use *streams*. For more information about streams, see the *Screen Developer's Guide*.

Modifying the IFS for C2000 platforms

This BSP includes an IFS buildfile specifically for the ADI RCC platform (**x86_64-adi-rcc-c2000.build**), as well as the buildfile for the generic IFS (x86_64-generic.build). This platform-specific buildfile is required because the ADI RCC C2000 doesn't ship with an on-board graphics controller.

Creating a USB bootable image for C2000 platforms

Modify the ***BSP_ROOT_DIR*/images/bios_root.build** to use the x86_64-adi-rcc-c2000.bin file instead of the x86_64-generic.bin file. Change the **bios_root.build** file from:

```
# Include the boot IFS in the image, in /.boot
 # In this example, x86_64-generic.bin is the image that you
 # build from the BSP and primary_boot_image.bin is its name in the image
 # You can choose any name less than 27 characters for the boot IFS in /.boot
 /.boot/primary_boot_image.bin=x86_64-generic.bin
```

to:

```
# Include the boot IFS in the image, in /.boot
 # In this example, x86_64-generic.bin is the image that you
 # build from the BSP and primary_boot_image.bin is its name in the image
 # You can choose any name less than 27 characters for the boot IFS in /.boot
 /.boot/x86_64-adi-rcc-c2000.bin=x86_64-adi-rcc-c2000.bin
```

The steps to create the USB image after you build the BSP are the same as described in the "*Create and transfer a bootable USB image (offline method)*" section of this guide.

© 2020, QNX Software Systems Limited

ADI RCC C2000 (Rangeley) driver commands

The tables below provide a summary of driver commands that are specific for the ADI RCC C2000 platform.

Any other drivers used by the buildfile for the ADI RCC C2000 platform are available as part of the generic driver commands. For more information about the generic commands, see the "*Generic x86_64 (64-bit) Driver Commands*" chapter in this guide.

For more information about these and other commands, see the QNX Neutrino *Utilities Reference*.

Here are the Rangeley-specific drivers:

- *Startup*
- *PCI Server*
- *USB Controller*
- *Serial*
- *Network*
- *SATA*

Network

Device	NETWORK (Intel E1000 Gigabit Ethernet controller)
Command	`io-pkt-v6-hc -d e1000`
Required binaries	`io-pkt-v6-hc`, `ifconfig`, `dhclient`
Required libraries	**devnp-e1000.so**, **libsocket.so**, **libcrypto.so**
Source location	Prebuilt only

PCI Server

Use `pci-server`. For more information, see the *PCI Server User's Guide* for details in the QNX SDP 7.0 documentation.

Device	Intel Host/PCI bridge
Environment variables	*PCI_HW_MODULE* *PCI_SLOG_MODULE* *PCI_DEBUG_MODULE* *PCI_BASE_VERBOSITY* *PCI_BKWD_COMPAT_MODULE*

© 2020, QNX Software Systems Limited

	For more information about the environment, see the "Environment variables" section in the Overview chapter of the *PCI Server User's Guide*.
Command	`pci-server -aspace-enable`
Required libraries	***$QNX_TARGET*/x86_64/lib/dll/pci**
Source location	Prebuilt only

SATA

Device	SATA
Command	`devb-ahci cam quiet ahci nports=4 blk automount=hd0t179:/:qnx6`
Required binaries	`devb-ahci`
Required libraries	**io-blk.so**, **libcam.so**, **fs-qnx6.so**, **fs-dos.so**, **cam-disk.so**
Source location	Prebuilt only

Serial

Device	SERIAL (UART2, UART3)
Command	`devc-ser8250 -e -b115200` (Serial driver)
Required binaries	`devc-ser8250`
Required libraries	N/A
Source location	Prebuilt only

Startup

Device	STARTUP
Command	`startup-x86 -D8250.2f8.115200 -v`
Required binaries	`startup-x86`
Required libraries	**libstartup.a**
Source location	***$BSP_ROOT_DIR*/src/hardware/startup/boards/x86**

USB (Host Controller)

Device	USB (Host driver)

© 2020, QNX Software Systems Limited

Command (EHCI is interrupt 23)	`io-usb-otg -d ehci pindex=0,irq=23` (when the USB Serial Bus Controller [EHCI] interrupt is 23)
Command	`io-usb-otg -d ehci`
Required binaries	`io-usb-otg`, `usb`, `devb-umass`
Required libraries	**libusbdi.so**, **libhiddi.so**, **devu-hcd-ehci.so**
Source location	Prebuilt only

The default location for USB is **/dev/usb/io-usb-otg**.

© 2020, QNX Software Systems Limited

Chapter 9
Working with Denverton SoC-based platforms

The Denverton SoC-based platform uses many of the same source and binaries provided the generic BSP, however, there are a few differences. The boards we provide buildfiles for are the Intel Car Creek Modules (CCM) and Intel Atom C3000 series Supermicro boards

- Connecting the power and the serial connection are different from common boards. For information, see "*Connecting power to the Denverton platform*" and "*USB UART Connection on the Denverton*."
- There's a separate buildfile, and separate image is built for the Denverton platform. For more information, see "*Modifying the IFS for Denverton platforms*"
- Configure board to boot with UEFI if it was previously configured to boot with Automotive Boot Loader (ABL). For more information, see "*Configure board to boot with UEFI*."
- Disable the APIC extended (x2APIC) feature. For more information, see "*Set EDKII BIOS to disable the APIC extended feature*."
- After you have your board setup to boot using UEFI, you can boot the board with an image from this BSP. For more information, see "*Boot the Denverton target*."

Connecting power to the Denverton platform

If you use the Car Creek Module carrier board, you must install a jumper between the green and black pin on the 2x10 pin connector. The power supply must be a 4-pin connector from the Standard Power supply to the J9 on the carrier board. For more information, see section 2.2 Board Setup of the *Car Creek Module User Guide* on the Intel website at *http://www.intel.com*.

USB UART Connection on the Denverton

You'll require a USB serial device driver on your host development machine in order to establish a terminal session with the target over USB. You must install an FTDI driver on your host system before you connect the USB UART on the Car Creek Module. After you install the FTDI driver on host system, connect the J2A1 connector with a USB cable to any USB port on the host machine.

You can download the appropriate FTDI device driver for your host system from the Future Technology Devices International Ltd. (FTDI) website at *http://www.ftdichip.com/FTDrivers.htm*.

On your host machine, using a terminal software and the appropriate COM port, connect with these settings to the Denverton:

- Baud rate: 115200
- Data: 8 bit
- Stop: 1 bit
- Parity: no
- Flow control: none

Configure board to boot with UEFI

You may need to configure your board to boot using UEFI if it was previously configured to boot with Automotive Boot Loader (ABL). This section describes the steps necessary to do so. Note that the menu

© 2020, QNX Software Systems Limited

may appear different based on the serial program you use to connect to the board. For example, the Miniport program available on Linux may show selected items that don't correspond to the the selected item you see in the console. For example, **Processor Configuration** shows as **Processor Test Settings** in the Miniport.

The steps below use PuTTy, which is a console program ran on Windows.

1. Put the USB image on a USB stick as described in "*Create and transfer an image to a bootable USB device*" and insert the USB stick into the board.
2. Reboot the board or cycle the power.
3. Wait a few minutes and then when you see the `Shell` prompt, type `exit` in your console and press the **Enter** key:

   ```
   Shell>
   ```

4. In the main menu that appears, use your arrow keys to navigate to and press **Enter** after each of the following menu items:
   - **EDKII Menu**
   - **Advanced**
   - **Processor Configuration** (Depending on your serial connection program, this might show as **Processor Test Settings**)
   - **Extended APIC**
5. After you press the **Enter** key for the **Extended APIC**, select **Disable**, press the **F10** key, press **Escape** key, and if prompted, press `Y` to save the changes (ensure that it's a capital Y (**Shift–y**)).
6. Press the **Escape** key and press `Y` to save the changes until you return to the main menu.
7. In the main menu where you started, use your arrow keys to navigate to and then press the press the **Enter** key following menu items:
   - **Boot Manager Menu**
   - **UEFI Internal Shell**

   You should see the following prompt in your console in a box appear, and if it doesn't skip ahead to step 9:

   ```
   Configuration changed. Reset to apply it now.
   Press ENTER to reset.
   ```

8. Press **Enter**, which reboots the board.

On some versions of the board, when you disable the APIC setting, it doesn't persist, therefore if you cycle the power (either push the power button on the front or the black switch on the back of the module), you must repeat steps 1-8 before you try to boot the reference image.

EDKII BIOS to disable the APIC extended feature (x2APIC)

If you want to use `startup-x86`, you must disable the APIC extended (x2APIC) feature to ensure that `startup-x86` runs properly. The feature, x2APIC isn't supported. To do this, complete these steps:

© 2020, QNX Software Systems Limited

1. In the EDKII BIOS (a text-based tool that runs on your target), navigate to "Extended APIC" and navigate through the menus in the following sequence: **EDKII Menu** > **Advanced** > **Processor Configuration** > **Extended APIC**.
2. Change the configuration value from `enabled` to `disabled`.

Modifying the IFS for Denverton platforms

By default, the **x86_64-ccm-denverton.build** and **x86_64-ccm-denverton-smmu.build** buildfiles generate IFS images that can be used to boot Car Creek Modules, while the **x86_64-sm-denverton.build** and **x86_64-sm-denverton-smmu.build** generate IFS images that are used to boot Supermicro boards. The **x86_64-ccm-denverton.efi**, **x86_64-ccm-denverton-smmu.efi**, **x86_64-sm-denverton.efi**, and **x86_64-sm-denverton-smmu.efi** IFS files are found in the ***$BSP_ROOT_DIR*/images** directory. To see what's included in the default IFS images, you can view each of the buildfiles.

Boot the Denverton target

Booting a Denverton-based target uses these steps:

- Copy one of the QNX IFS images for the Denverton platform to a DOS-formatted USB stick.
  - **x86_64-ccm-denverton.efi**
  - **x86_64-ccm-denverton-smmu.efi**
  - **x86_64-sm-denverton.efi**
  - **x86_64-sm-denverton-smmu.efi**
- Insert the USB stick to the J4 or J5 port on the CCM carrier board.
- Turn on the power. You should see the the following shell in your terminal.

```
EFI Interactive Shell v2.187477C2-69C7-11D2-8E39-00A0C969723B 5516D020
  EDK IIlProtocolInterface: 752F3136-4E16-4FDC-A22A-E5F46812F4CA 55171118
  UEFI v2.40 (EDK II, 0x00010000)008-7F9B-4F30-87AC-60C9FEF5DA4E 54357140
  Mapping table
      FS0: Alias(s):HD9h0b:;BLK1:
           PciRoot(0x0)/Pci(0x15,0x0)/USB(0x7,0x0)/HD(1,MBR,0x00000000,0x800,0x80000)
     BLK0: Alias(s):
           PciRoot(0x0)/Pci(0x15,0x0)/USB(0x7,0x0)
     BLK2: Alias(s):
           PciRoot(0x0)/Pci(0x1C,0x0)/Ctrl(0x0)

  Press ESC in 1 seconds to skip startup.nsh or any other key to continue.
  Shell>
```

- Type `fs0:` to specify the USB stick that includes the QNX IFS image for Denverton SoC-based platform that you want to use:

  ```
  Shell> fs0:
  ```

- Type the QNX IFS file name to boot the QNX images. For example, type the following:

  ```
  FS0:\> x86_64-ccm-denverton.efi
  ```

© 2020, QNX Software Systems Limited

Denverton driver commands

The tables below provide a summary of driver commands for the Denverton platform.

Any other drivers used by the buildfile for the Denverton platform are available as part of the generic driver commands. For more information about the generic commands, see the "*Generic x86_64 (64-bit) Driver Commands*" chapter in this guide. For more information about these and other commands, see the QNX Neutrino *Utilities Reference*.

Here are the Denverton-specific drivers:

- *Startup*
- *PCI Server*
- *SMMU Manager*
- *USB Controller*
- *Serial*
- *Network*
- *SATA*

The commands summarized below represent the commands that are used on the Denverton SoC-based platform we support that include the Intel Car Creek Module (CCM) and Intel Atom C3000 Supermicro boards (Supermicro). We've indicated which commands are board-specific. For more information, see the buildfiles.

Network

Device	NETWORK (Intel 10 Gigabit Ethernet driver)
Commands	`io-pkt-v6-hc -d ixgbe` (onboard PCIe IXGBE driver) `io-pkt-v6-hc -d asix` (ASIX USB-Ethernet dongle driver) `io-pkt-v6-hc -d e1000` (e1000 PCIe Ethernet)
Commands when SMMUMAN is running	`io-pkt-v6-hc -d ixgbe -p tcpip smmu=on (onboard PCIe IXGBE driver)` `io-pkt-v6-hc -d asix -p tcpip smmu=on` (ASIX USB-Ethernet dongle driver) `io-pkt-v6-hc -d e1000 -p tcpip smmu=on` (e1000 PCIe Ethernet)
Required binaries	`io-pkt-v6-hc`, `ifconfig`, `dhclient`, `smmuman` (if SMMUMAN is running)
Required libraries	**devnp-ixgbe.so**, **libsocket.so**, **libcrypto.so**, **smmu-vtd.so** (if SMMUMAN is running)
Source location	Prebuilt only

© 2020, QNX Software Systems Limited

To support additional Ethernet ports:

In your buildfile, you can add additional ports. To do so, find where the networking is started and then add the following lines after the existing port:

- `if_up -r 10 -p ix`*n*, where *n* represents instances based on the number of interfaces available
- `ifconfig ix`*n* `up`, where *n* is the same value as the previous line you added and represents the network interface

---

WARNING: : You must ensure that the IFWI is configured to use the LEK6 Ethernet firmware so that the `devnp-ixgbe` driver is not impacted by a "No carrier" error. If you don't configure the IFWI to use the LEK6 driver, after you reboot the board, the `ix0` won't be initialized correctly and a No Carrier error occurs.

In addition to what's described in the SDP 7.0 documentation in the QNX Neutrino *Utilities Reference* for the **devnp-ixgbe.so** driver, the **devnp-ixgbe.so** supports has these new options:

`typed_mem=name`

The name of the typed memory to use.

`ign_cksum`

Ignore the EEPROM checksum check. The network interface chip firmware is contained in the EEPROM. The checksum check ensures that the EEPROM is valid.

`event`

Use *InterruptAttachEvent()* rather than *InterruptAttach()*function. This option is a mandatory option and must be specified when you start the driver for use in this BSP.

`pause_rx_enable`

Enable Rx pause frames with respect to full duplex flow control.

`pause_tx_enable`

Enable Tx pause frames with respect to full duplex flow control.

`ptp`

Enable Precision Time Protocol (PTP).

`unsupported_sfp`

Allow unsupported small form-factor pluggable (SFP) modules (fiber only).

The following options are no longer available for use with the **devnp-ixgbe.so** driver:

- `force_link`
- `pauseignore`
- `pausesuppress`
- `promiscuous`

© 2020, QNX Software Systems Limited

PCI Server

Use `pci-server`. For more information, see the *PCI Server User's Guide* in the QNX SDP documentation.

Device	Intel Host/PCI bridge
Environment variables	*PCI_HW_MODULE* *PCI_SLOG_MODULE* *PCI_DEBUG_MODULE* *PCI_BASE_VERBOSITY* *PCI_BKWD_COMPAT_MODULE* For more information about the environment, see the "Environment variables" section in the "Overview" chapter of the *PCI Server User's Guide*.
Command	`pci-server --bus-scan-limit=16 -aspace-enable`
Required libraries	`pci-server`
Required libraries	***$QNX_TARGET*/x86_64/lib/dll/pci**
Source location	Prebuilt only

SATA

Device	SATA
Command	`devb-ahci cam quiet ahci nports=8 blk`
Command (when SMMUMAN is running)	`devb-ahci disk name=sata cam smmu=on`
Required binaries	`devb-ahci`, `smmuman` (if SMMUMAN is running)
Required libraries	**io-blk.so**, **libcam.so**, **fs-qnx6.so**, **fs-dos.so**, **cam-disk.so**, **smmu-vtd.so** (if SMMUMAN is running)
Source location	Prebuilt only

Serial

Driver for the serial/UART/HSUART interface.

Device	SERIAL (UART2, UART3)
Commands (CCM)	`devc-serpci vid=0x8086,did=0x19d8,pci=0 -e -b115200` `devc-serhsu vid=0x8086,did=0x19d8,pci=0 -e -b115200`

© 2020, QNX Software Systems Limited

Command when SMMUMAN is running (CCM)	`devc-serhsu -o smmu=on vid=0x8086,did=0x19d8,pci=0,dma -e -b115200`
Command (Supermicro)	`devc-ser8250 -e -b115200`
Required binaries	`devc-serhsu`, `devc-ser8250`, `devc-serpci`, `smmuman` (if SMMUMAN is running)
Required libraries	**libpci.so**, **libc.so**, **smmu-vtd.so** (if SMMUMAN is running)
Source location	***$BSP_ROOT_DIR*/src/hardware/devc/serhsu**

For information about the `devc-serhsu` driver, see "*devc-serhsu*" in the "*BSP-specific Drivers and Utilities*" chapter of this guide.

SMMU Manager

Device	Memory Management
Commands (CCM)	`smmuman smmu vtd`
Required binaries	`smmuman`
Required libraries	**smmu-vtd.so**
Source location	N/A

It's recommended that you start SMMU Manager (SMMUMAN) before you start any drivers that use SMMU. Some drivers may not start until you SMMU Manager has been started. You can enable SMMU Manager support for a driver using an option. SMMUMAN refers to providing DMA containment and memory-management support. For most drivers, you use the `smmu` option to enable SMMUMAN. In other cases, the `smmu` is specified with the `-o` option. To see whether your driver provides SMMU Manager support, see the documentation for the driver either in this guide or the QNX SDP 7.0 documentation.

For more information about the SMMU Manager, see the *SMMUMAN User's guide* in the QNX SDP 7.0 documentation.

Startup

Device	STARTUP
Command (CCM)	`startup-x86 -D8250_pci_io.0:26:0:0^0.115200.1843200.16 -vv`
Command (Supermicro)	`startup-x86 -v -D8250..115200`
Required binaries	`startup-x86`

© 2020, QNX Software Systems Limited

Required libraries	**libstartup.a**
Source location	***$BSP_ROOT_DIR*/src/hardware/startup/boards/x86**

USB (Host Controller)

Device	USB (Host driver)
Command	`io-usb-otg -d xhci`
Command when SMMUMAN is running	`io-usb-otg -s smmu=on -d xhci`
Required binaries	`io-usb-otg`, `usb`, `devb-umass`, `smmuman` (if SMMUMAN is running)
Required libraries	**libusbdi.so**, **libhiddi.so**, **devu-hcd-xhci.so**, **smmu-vtd.so** (if SMMUMAN is running)
Source location	Prebuilt only

© 2020, QNX Software Systems Limited

Chapter 10
BSP-specific Drivers and Utilities

Here's the specific driver included for this BSP.

- *devc-serhsu*

devc-serhsu

The Serial/UART/HSUART interface for the Intel Car Creek Module.

This driver doesn't work on Intel Atom C3000 Supermicro boards.

Syntax:

```
devc-serhsu devc-serhsu [vid=x,did=x[,pci=x]]
                        [dma[=typed_mem_name]
                        [isr]
                        [options]
```

Runs on:

QNX Neutrino

Options:

This driver supports generic `devc-ser*` options. For information about those options, see "`devc-ser*`" in the QNX Neutrino *Utilities Reference* guide in the QNX SDP 7.0 documentation.

For this driver, the generic `-o` option supports the `smmu` option.

`smmu=0|1|off|on`

(QNX Neutrino 7.0.4 or later) Disable or enable the SMMU Manager support using one of these values:

- 0/`off`: (Default) Disable SMMU Manager support.
- 1/`on`: Enable SMMU Manager support.

If the SMMU Manager service (`smmuman`) service isn't running, the driver won't start and an error is logged to the `slogger`.

In addition to the generic `devc-ser*` options, this driver supports the following options:

`did=0x`*address*

PCI Device ID (Intel). The *address* is the form of XXXX.

`dma[=`*typed_mem_name*`]`

The DMA typed name to use for the driver. If you don't specify a name, below4G is used as the typed memory. If you don't use this option for high-speed communication and you're not using hardware flow control, characters are lost (data is dropped).

`isr`

Use ISR, which causes the driver to use *InterruptAttach()* instead of *InterruptAttachEvent()*. You **must not** use this option for safety products.

© 2020, QNX Software Systems Limited

If you specify both the `dma` and `isr` options, the `isr` option is ignored and *InterruptAttachEvent()* is used.

`loopback`

This option is only for testing purposes and shouldn't be used in a production environment. When the `loopback` option is enabled, data written to the output buffer is placed into the receive buffer by the UART hardware. By default, this option is disabled.

`pci=`*`pci_index`*

The PCI index.

`vid=0x`*`address`*

PCI Vendor ID (Intel). The *address* is the form of XXXX.

`-c` ***`clk[/div]`***

Set the input clock rate and an optional divisor.

-C number

Size of the canonical input buffer (default: 256).

`-t` ***`level`***

Set receive FIFO trigger level. The SCIF default is 14.

`-u` ***`serial_unit_num`***

Set serial unit number. The default is 1.

Description:

For this driver, if you specify the VID, PCI, PCI index, and a device with that combination exists, it'll be treated as a UART device regardless of its PCI class code. If either one of these parameters isn't set, the class/subclass code 7, 0 (i.e., PCI_CCODE_COMM_UART_ANY as defined in **pci/pci_ccode.h**) will be used as part of the search criteria. This allows devices which don't report a correct class code to be found.

Examples:

- To start the Serial driver with specific Vendor Device IDs, with baud rate=115200:

  ```
  devc-serhsu vid=0x8086,did=0x19d8,pci=0 -e -b115200
  ```

- To start the Serial driver with specific Vendor Device IDs, with baud rate=115200 with SMMU Manager:

  ```
  devc-serhsu vid=0x8086,did=0x19d8,pci=0,dma,smmu -e -b115200
  ```

The commands above uses the `-e` option, which runs the driver in edited mode. Edited mode is recommended when you are using the serial connection as a debug console or terminal. When you are using the serial connection to transfer files or log data, you should not use the

© 2020, QNX Software Systems Limited

`-e` option; instead, you should run the driver in raw mode, which is the default mode when you start the driver without the `-e` option. You also explicitly specify the `-E` option to run the driver in raw mode. For more information about raw and edited mode, see *Input modes* in the QNX Neutrino *System Architecture Guide* in the SDP 7.0 documentation.

© 2020, QNX Software Systems Limited