QNX® Software Development Platform 8.0

BSP User's Guide
Raspberry Pi 4 Board

Contents

Copyright and patent notice

BlackBerry Limited
1001 Farrar Road
Ottawa, Ontario
K2K 0B3
Canada

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

November 27, 2023

About This Guide

This guide contains installation and start-up instructions for the QNX Board Support Package (BSP) for the Raspberry Pi 4 Board (Model B). This BSP supports Raspberry Pi 4 Board (Model B) with 2GB, 4GB, and 8GB RAM. Only 64-bit support is provided by QNX Software Systems.

For convenience, this document refers to the Raspberry Pi 4 Board (Model B) as the Raspberry Pi 4 board. This guide covers the following information:

- an overview of the BSP for the Raspberry Pi 4 board
- structure and contents of the BSP
- how to build the BSP using commandline or the QNX Momentics IDE
- how to prepare a bootable microSD card
- how to transfer the image to a bootable microSD card
- how to boot your board

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, and the supported host OSs and boards	About This BSP
Booting the reference board with the provided reference image	Installation Notes
Building this BSP	Build the BSP
Running the driver commands	Driver Commands

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.

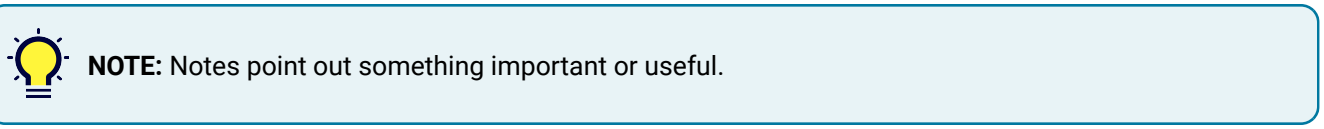
The following table summarizes our conventions:

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**

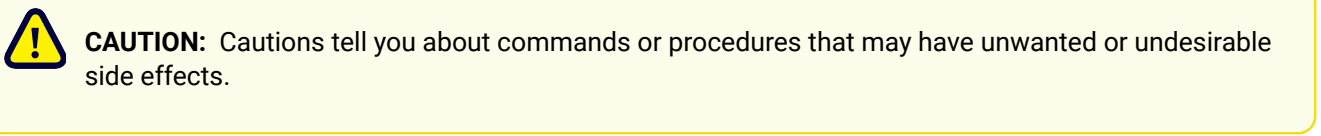
We use an arrow in directions for accessing menu items, like this:

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

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



NOTE: Notes point out something important or useful.



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

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

ALERT: Alerts tell you about commands or procedures that could expose private information or otherwise compromise security.

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.

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 (https://blackberry.qnx.com/en/support). You'll find a wide range of support options.

Before You Begin

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

Essential information

Before you install your BSP, review the following documentation:

- Information about your board's hardware and firmware provided by the vendor of the board. Documentation for the Raspberry Pi 4 Board (Model B) can be found on the Raspberry PI website at https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications.
- *Building Embedded Systems*. This guide contains general information about working with BSPs from QNX Software Systems that's common to all BSPs and is available from the QNX Software Development Platform 8.0 documentation.

Required software

To work with this BSP, you require the following:

- BSP ZIP file for the board.
- QNX Software Development Platform 8.0 installed on a Linux or Windows host system.
- Virtual COM Port (VCP) Driver. This driver must be installed on your host system that connects to the board. 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 (Qtalk, Hyperterminal, PuTTy, etc.) to connect to the board for debugging. Alternatively, you can use `ssh` session from the QNX Momentics IDE.

About This BSP

These notes list what's included in the BSP, and identify the host OSs and boards it supports.

What's in this BSP

This BSP contains the following components: For the status of a driver, see the release notes for this BSP.

Component	Format	Comments
I2C driver	Binary and source	
Serial driver	Binary and source	
Network driver	Binary and source	io-sock driver for QNX High-Performance Networking Stack
PCI driver	Binary only	
SD/MMC driver	Binary and source	
SPI driver	Binary and source	
Startup	Binary and source	
USB Host Controller driver	Binary only	
Watchdog	Binary and source	

Supported OSs

To install and use this BSP, you must have installed the QNX SDP 8.0 on a Linux or Windows host. This BSP supports the following target OS:

- QNX OS 8.0

Supported boards

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**.

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.

Installation Notes

These installation notes describe how to build, work with, and run this BSP.

Overview

Here's the process to build and run your BSP:

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 to set DIP switches on your board to further configure it. For more information about how to connect to your board, see "Set up the hardware."
3. Create a bootable microSD card. After you create this card, you can either transfer prebuilt images that come with the BSP or change the buildfile to build your own, customized image. For more information about how to create a bootable microSD card, see "Prepare a bootable microSD card."
4. If this is the first time you're running the BSP, you must transfer the provided IFS image file to the microSD card to boot your board.
5. After you've put your image on the microSD card, you can boot the board. At this point, your board is running QNX OS.

After your board boots, you can use that image to understand how it works. Depending on the other devices that 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 buildfile and build a custom image. For specific information about building the image for this board, see the"Build the BSP" chapter in this guide. For generic information about modifying and building an image, see the *Building Embedded Systems* guide in the QNX SDP 8.0 documentation.

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 8.0 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 commandline to a working directory that's outside of your QNX SDP 8.0 installation.

  NOTE: The `unzip` utility is available on all supported host platforms and is included with your installation of QNX SDP 8.0.

  The QNX-packaged BSP is independent of the installation and build paths, if the QNX Software Development Platform 8.0 is installed. To determine the base directory, open a Command Prompt window (Windows) or a terminal (Linux), and then type `qconfig`.

Extract from the commandline

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), run **qnxsdp-env.bat** (Windows) or `source` **qnxsdp-env.sh** (Linux) from your QNX SDP 8.0 installation to set up your environment. You must run the shell or batch file to use the tools provided with QNX SDP 8.0.

   NOTE: 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/raspberrypi-bcm2711-rpi4**). 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) 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_raspberrypi-bcm2711-rpi4_br-hw-rel_be-800_build_ID.zip
   ```

   where *build_ID* is the BSP build number (e.g., `JBN11`).

You should now be ready to build your BSP. See the "Build the BSP (commandline)" chapter for more information.

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** from the list and then click **Next**.
5. In the **Select the package** dialog box, click **Browse for ZIP Archive...**.
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-raspberrypi-bcm2711-rpi4**). 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.

Structure of a BSP

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

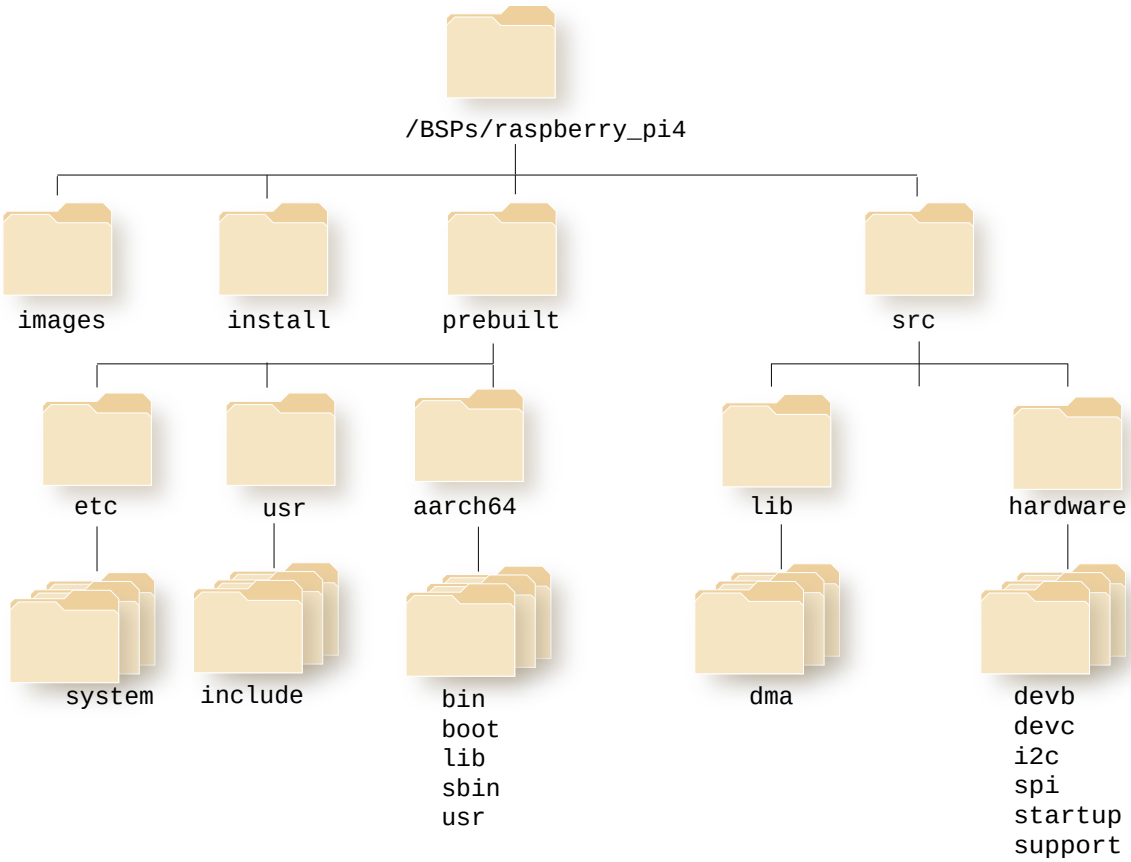


Figure 1: Structure of a BSP

For more information about the contents of the directory structure, see the *Building Embedded Systems* guide in the QNX SDP 8.0 documentation.

Set up the hardware

Connect the hardware

In addition to the board, you require these cables:

- USB to TTL serial cable
- an Ethernet cable
- 5V DC power cable

CAUTION:

Use only the power supplies provided with the board. Use of any other power supply may permanently damage the board. When power cycling the board, turn the power on or off from the AC adapter side, rather than removing and inserting the DC power plug on the board, to prevent damaging the board's power-regulation circuitry.

To connect the Raspberry Pi 4 board:

1. Insert the microSD card with your QNX IFS image file into the selected boot slot on the board.
2. Connect a USB to TTL serial cable from the board's serial pins and the USB port on your host machine.

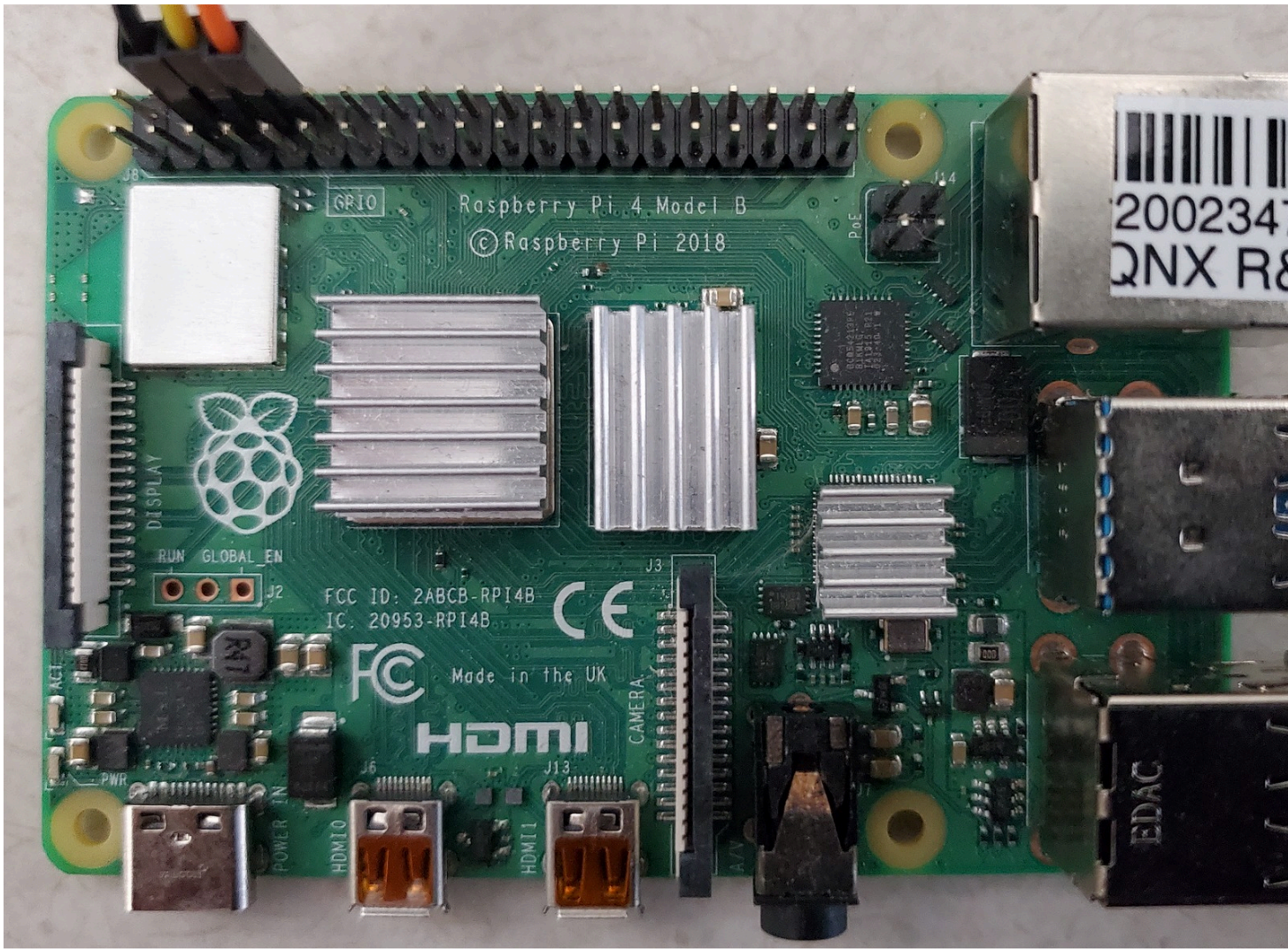


Figure 2: Serial pins on the 40-pin GPIO expander header (J8)

- J8 pin6: **GND**
- J8 pin8: **GPIO14 (UART_TXD0)**
- J8 pin10: **GPIO15 (UART_RXD0)**

3. Connect the Ethernet cable to the Ethernet jack on the board.
4. Connect the board's 5V DC power supply to the **POWER IN** (USB-C) connector on the board.
5. Identify the host serial port on your host system:

   On a Linux host, you can check which port is the host serial port by looking at what port appears when the cable is inserted. To do this, type the command:

   ```
   $ ls /dev/ttyUSB*
   ```

On a Windows system, open the **Device Manager** and expand the “Ports (COMand LPT)” section. Then, look for a COM port named Gadget Serial or USB Serial Port.

6. Connect to the COM port using your favorite terminal program with these settings:
   - Baud rate: **115200**
   - Data: **8 bit**
   - Parity: **none**
   - Stop: **1 bit**
   - Flow control: **none**

System Layout

The following is the memory layout required for the Raspberry Pi 4 board.

The table below shows the memory layout for the BSP image:

Item	Start	End
OS image location	`0x80000`	

Prepare a bootable microSD card

You can prepare a bootable microSD card on a Linux, or Windows host system. We recommend that you use Class 10 (or UHS-1) microSD cards.

Depending on your host OS, follow the steps in either the “Prepare a bootable microSD card on a Linux host” or “Partition and format the microSD card on a Windows host” section to prepare your microSD card. After you've prepared the card, you can copy the image to it so that you can boot the board.

CAUTION: Ensure that your microSD card isn't write-protected.

Typically there's a lock switch on the side of an microSD or the SD case (for microSD cards). When this switch is in the lock position, you can't modify or delete the contents of the card. Make sure that this switch is in the unlock position so that you can format and partition the card.

Prepare a bootable microSD card on a Linux host

The following is a quick, step-by-step method for formatting the microSD card from a terminal:

1. Run the following command, once with the microSD card out of the reader on your host system:

```
$ mount
. . .
```

   Note the mounted devices listed. Now insert the card and run the same command a second time.

```
$ mount
. . .
/dev/sda1
```

   When the command is run with the card inserted, an additional device should appear (for example, **sda1** or **mmcblk0p1**; what you see may differ on your computer). This additional device is the target device.

   For the remainder of these instructions, we'll use **sda1** for the device used. Substitute your own device that you noted when you ran the `mount` command for the second time.

2. If the device is mounted, unmount it. For example:

```
$ umount /dev/sda1
```

   After you run the command, your microSD card is now ready to be partitioned.

3. Using administrator privileges, run the following commands, substituting your device for **sda1**, and responding to the prompts as indicated. For example:

```
$ sudo fdisk /dev/sda
```

NOTE: Since you are formatting the whole microSD card and not just a partition, you may need to strip the identifier at the end of the mountpoint when you run the `fdisk` command. For example, for **sda1**, the identifier is 1 or for **mmcblk0p1**, the identifer is p1. For more information about the identifier, see the user guide for the variant of Linux that you're running.

4. Remove the existing partitions. Keep typing the d command until no partitions are left. For example:

```
Command (m for help): d
No partition is defined yet!
```

5. Type u and then o to create an empty DOS partition table:

```
...

Command (m for help): u
Changing display/entry units to cylinders

Command (m for help): o

Building a new DOS disklabel with disk identifier 0xcdd1b702.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by write.

WARNING: cylinders as display units are deprecated. Use command 'u' to change units to sectors.
```

6. Type n followed by p to create a new primary partition. For example:

```
Command (m for help): n

Partition type:
p   primary (0 primary, 0 extended, 4 free)
e   extended
Select (default p): p
```

7. Type 1 to set the partition as the primary partition and specify the defaults for your card. For example, enter the specified defaults for *start cylinder* and *end cylinder*, which in this example are 1 and 240. Your defaults may be different depending on the size of your microSD card.

```
Partition number (1-4, default 1): 1

First cylinder (1-240, default 1): start cylinder

Last cylinder, (100-240, default 240): end cylinder
```

```
Using default value 240
```

8. Type a to set the active partition. Generally, this should be partition 1.

```
Command (m for help): a Partition number (1-4): 1
```

9. Type `t` and then c to set the partition type (1 is default).

```
Command (m for help): t
Selected partition 1

Hex code (type L to list codes): c
Changed system type of partition 1 to c (W95 FAT32 (LBA))
```

10. Type w to write the changes.

```
Command (m for help): w

The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

11. Format the microSD card. Note that the identifier, such as “1” or “p1” at the end of the mountpoint must be included:

```
$ sudo mkfs.vfat /dev/sda1
mkfs.msdos 3.0.12 (16 March 2021)
```

Partition and format the microSD card on a Windows host

The default Windows formatting tool that appears when you insert a blank (or unrecognized) microSD card into a Windows host isn't sufficient to format the microSD card with a bootable partition. In addition, some microSD cards come with pre-created partitions that aren't suitable to boot the board. Instead, use the diskpart command-line tool to format your microSD card.

1. Start a Command Prompt window (e.g., on Windows 10, run **Start Menu** > **All Programs** > **Accessories** > **Command Prompt**). Ensure that you start the Command Prompt window using Administrator privileges.
2. Run the `diskpart` utility from your Command Prompt window:

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\windows\system32>diskpart

Microsoft DiskPart version 6.3.9600

Copyright (C) 1999-2013 Microsoft Corporation.
```

```
On computer: CI0700000001064

DISKPART>
```

3. Run the `list disk` command to get a list of available drives, then identify your microSD card on this list (in this example, Disk 1):

```
DISKPART> list disk

Disk ###     Status       Size       Free      Gyn      Gpt
--------     ------       ----       ----      ---      ---
Disk 0       Online       465 GB     1024 KB
Disk 1       Online      7580 MB     4401 MB

DISKPART>
```

When the command is run with the card inserted, an additional disk should appear (in this case Disk 1). This additona is the target device. For the steps that follow, we'll be using Disk 1. Ensure you use the disk that matches your microSD card.

4. Run the `select disk` command to perform operations on the microSD card:

```
DISKPART > select disk 1

Disk 1 is now the selected disk.

DISKPART >
```

5. Show the partitions on the microSD card. In this case, there are two partitions:

```
DISKPART> list part
Partition ### Type Size Offset
------------ ---------------- ------- -------
Partition 1 Primary  512 MB 1024 KB
Partition 0 Primary 2048 MB 1125 MB

DISKPART>_
```

6. Delete all available partitions on the microSD card:

```
DISKPART> select part 1

Partition 1 is now the selected partition.

DISKPART> delete part

DiskPart successfully deleted the selected partition.
```

```
DISKPART> select part 1

Partition 1 is now the selected partition.

DISKPART> delete part

DiskPart successfully deleted the selected partition.

DISKPART> list part

There are no partitions on this disk to show.

DISKPART>_
```

7. Create a 256 MB partition with a 1 MB offset, by running the following command:

```
DISKPART> create partition primary size=256 offset=1024

DiskPart succeeded in creating the specified partition.

DISKPART>
```

NOTE: You can choose a larger size for the partition. The offset you use may vary based on the hardware platform. For more information regarding the offset to use, see the documentation for your hardware platform.

8. Run the `list partition` command, to see the partition you created:

```
DISKPART> list partition

  Partition ###  Type              Size     Offset
  -------------  ----------------  -------  -------
* Partition 1    Primary            256 MB  1024 KB
```

9. Run the `select partition` command to perform operations on the partition that you created in the previous step:

```
DISKPART> select partition 1

Partition 1 is now the selected partition.

DISKPART>
```

10. Run the `active` command to make the partition active:

```
DISKPART> active
```

```
DiskPart marked the current partition as active.
```

11. Run the `format` command to format the partition:

```
DISKPART> format fs=fat32 quick

  100 percent completed

DiskPart successfully formatted the volume.

DISKPART>
```

WARNING: Ensure you specify that drive where microSD card is located. If you incorrectly specify the wrong disk, it could result in data loss.

12. Run the `list partition` command to verify that the partition is active and that you have the correct partition size and offset:

```
DISKPART> list partition

  Partition ###  Type              Size     Offset
  -------------  ----------------  -------  -------
* Partition 1    Primary            256 MB  1024 KB

DISKPART>
```

The "*" (asterisk) beside the partition name indicates that the partition is active. You can now copy the QNX IFS file to the partition.

13. Exit the `diskpart` utility:

```
DISKPART> exit

Leaving DiskPart...

C:\windows\system32>
```

Prepare a bootable microSD card using Raspberry Pi Imager

You can use “Raspberry Pi Imager” to install Pi OS to a bootable microSD card on a Linux or Windows host system. We recommend that you use Class 10 (or UHS-1) microSD cards.

Depending on your host OS, download and install “Raspberry Pi Imager” and follow the instruction described in https://www.raspberrypi.com/software/.

For example, if you choose to install “Raspberry Pi OS (64-bit)”, after installation, you will find many the software components in the “bootfs/” directory on microSD card, including firmware files, like “start*.elf”, “fixup*.dat” and “bootcode.bin”, system config file “config.txt”, OS image and device tree files.

You can use this bootable microSD card to boot QNX IFS, please see the “Transfer the image” section in this chapter.

Transfer the image

After you prepare a bootable microSD card, you're ready to transfer the image to it and change the system configuration file “config.txt” to boot QNX IFS image.

Provided with your BSP are prebuilt files that you can use. Alternatively, if you make modifications to the buildfile, you can rebuild the IFS. For information about building your BSP, see the “Build the BSP” section in this chapter.

Copy the necessary binaries and IFS to microSD card

Obtain boot firmware files

The Raspberry Pi 4 board needs the following firmware files to boot:

```
bcm2711-rpi-4-b.dtb, start4.elf, start4x.elf, start4db.elf, start4cd.elf,
fixup4.dat, fixup4x.dat, fixup4cd.dat, fixup4db.dat
bcm2711-rpi-4-b.dtb, overlays/
```

If you already have a bootable microSD card which has files listed above, you can skip downloading them, otherwise these files can be downloaded from Raspberry wite site, such as: https://github.com/raspberrypi/firmware/tree/stable/boot (see the BSP release notes for the specific revision recommended) and should be saved to the DOS partition on the prepared bootable microSD card.

Prepare boot config file

The Raspberry Pi 4 early-stage boot firmware reads config.txt file. If your bootable microSD card already has this file, please make sure that it include:

```
[rpi4]
arm_64bit=1
force_turbo=1
enable_uart=1
gpu_mem=16
max_framebuffers=2
kernel=ifs-rpi4.bin
```

If you don't have a config file, you can create one and store it along side with other firmware files.

For Linux hosts

Copy all firmware and config files and QNX IFS image to an microSD card, by using the following steps:

1. Insert the microSD card into your host system and mount it if necessary.

   The microSD card should appear in the list of mounted devices. If it doesn't appear, remove and re-insert the card into your host system. You may need to run the `mount` command before and after inserting the microSD card to determine its mountpoint. For example, the microSD card mountpoint may appear like:

   ```
   $ mount
   ...
   /dev/sda1 on /media/074B-DAC7
   ...
   ```

2. In a terminal on your host system, copy all firmware and config files to mount point `/media/074B-DAC7`.

3. Copy the QNX IFS image to the microSD card as ifs-rpi4.bin. The filename must be ifs-rpi4.bin which is the boot image name defined in the **config.txt** file.

```
$ cd $BSP_ROOT_DIR/images
$ cp ifs-rpi4.bin /media/074B-DAC7/
```

The microSD card should now be ready to boot your Raspberry Pi 4 board.

For Windows hosts

1. In a Command Prompt window, run `bash`. You might need to run **qnxsdp-env.bat** from your QNX SDP 8.0 installation to set up your environment to use `bash`.

```
$ bash
```

2. Copy all firmware and config files to the `G:/` directory mounted on the microSD card.
3. Copy the QNX IFS image to the microSD card as ifs-rpi4.bin. The filename must be ifs-rpi4.bin which is the name defined in the **config.txt** file. In the example below, the microSD card appears `G:/` drive.

```
$ cd $BSP_ROOT_DIR/images
$ cp ifs-rpi4.bin G:/
```

Boot the board

After you transfer the image to an microSD, you can use it to boot your board.

After you boot your board, you should see the board printing to the console. Here's an example of what you should see in the console connection for the Raspberry Pi 4 board:

```
Hypervisor support disabled
MMU: 16-bit ASID 44-bit PA TCR_EL1=b5183519
GICv2: 256 interrupts
GICv2: routing SPIs to gic cpu 0
cpu0: MPIDR=80000000
cpu0: MIDR=410fd083 Cortex-A72 r0p3
cpu0: CWG=4 ERG=4 Dminline=4 Iminline=4 PIPT
cpu0: CLIDR=a200023 LoUU=1 LoC=2 LoUIS=1
cpu0: L1 Icache 48K linesz=64 set/way=256/3
cpu0: L1 Dcache 32K linesz=64 set/way=256/2
cpu0: L2 Unified 1024K linesz=64 set/way=1024/16
cpu0: GICv2 cpu interface 0
Loading IFS...done
cpu1: MPIDR=80000001
cpu1: MIDR=410fd083 Cortex-A72 r0p3
cpu1: CWG=4 ERG=4 Dminline=4 Iminline=4 PIPT
cpu1: CLIDR=a200023 LoUU=1 LoC=2 LoUIS=1
cpu1: L1 Icache 48K linesz=64 set/way=256/3
cpu1: L1 Dcache 32K linesz=64 set/way=256/2
cpu1: L2 Unified 1024K linesz=64 set/way=1024/16
cpu1: GICv2 cpu interface 1
cpu2: MPIDR=80000002
cpu2: MIDR=410fd083 Cortex-A72 r0p3
cpu2: CWG=4 ERG=4 Dminline=4 Iminline=4 PIPT
cpu2: CLIDR=a200023 LoUU=1 LoC=2 LoUIS=1
cpu2: L1 Icache 48K linesz=64 set/way=256/3
cpu2: L1 Dcache 32K linesz=64 set/way=256/2
cpu2: L2 Unified 1024K linesz=64 set/way=1024/16
cpu2: GICv2 cpu interface 2
cpu3: MPIDR=80000003
cpu3: MIDR=410fd083 Cortex-A72 r0p3
cpu3: CWG=4 ERG=4 Dminline=4 Iminline=4 PIPT
cpu3: CLIDR=a200023 LoUU=1 LoC=2 LoUIS=1
cpu3: L1 Icache 48K linesz=64 set/way=256/3
cpu3: L1 Dcache 32K linesz=64 set/way=256/2
cpu3: L2 Unified 1024K linesz=64 set/way=1024/16
cpu3: GICv2 cpu interface 3

System page at phys:0000000000012000 user:ffffff8040206000 kern:ffffff8040204000
Starting next program at vffffff8060059ae0
```

```
syspage::hypinfo::flags=0x00000000

Welcome to QNX 8.0.0 on RaspberryPi4B !

Starting wdtkick ...
Starting I2C driver ...
Starting PCI Server ...
Starting serial driver (/dev/ser1)
Starting SPI master driver ...
Starting SDMMC driver (/dev/sd0)
Path=0 - bcm2711
 target=0 lun=0     Direct-Access(0) - SDMMC: SD32G Rev: 8.5
Inform vc to load vl805 firmware
Starting USB xHCI controller in the host mode (/dev/usb/*)...
Starting devf-ram filesystem ...
Starting networking ...
Starting DHCP client ...
Starting SSH daemon ...
Starting devc-pty manager ...
Starting qconn daemon ...
Starting shell ...
#
```

QNX OS should now be running on your target. You can test it by executing any shell command, or any command residing within the OS image (`ls`, `pidin`, etc.). You can also type `uname` and it should return `QNX`.

Build the BSP

You can use the QNX Momentics IDE or the commandline on Linux or Windows to build an image.

Generic instructions to modify your BSP (add contents and modify the buildfile) can be found in the *Building Embedded Systems* guide, which is available as part of the QNX SDP 8.0 documentation.

For instructions on how to build this BSP using the IDE, see "Build the BSP (IDE)" section in this chapter. For detailed information on how to build using the IDE, see the *IDE User's Guide*, which is available as part of the QNX SDP 8.0 documentation.

If you plan to work with multiple BSPs, we recommend that you create a top-level BSP directory and then subdirectories for each different BSP. The directory you create for a specific 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 a convenience to quickly get QNX OS 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 own environment.

Prebuilt image

After you've unzipped the BSP, a prebuilt QNX IFS image is available in the BSP's **/images** directory. The prebuilt IFS image (**ifs-rpi4.bin**) can be regenerated with the BSP `make` file, and is configured for the BSP device drivers already available for your board.

If you modify and build the IFS, the original IFS files are overwritten. If you need to revert to this prebuilt image, simply run the `make` command from the BSP's root directory using the original buildfiles. To determine the location of the buildfile to modify, open the ***$BSP_ROOT_DIR*/source.xml** file.

NOTE: If you forget to make a backup copy of the IFS file, you can still recover the original prebuilt IFS; simply extract the BSP from the **.zip** archive into a new directory and get the prebuilt image from that directory.

Build the BSP (commandline)

You can use the commandline on Linux or Windows to work with this BSP.

Makefile targets in the BSP

The **Makefile** is located under the ***$BSP_ROOT_DIR*/images** directory. It defines more than one target:

`all`

Builds the QNX IFS for the target.

`ifs-rpi4.bin`

Builds the IFS file used by the Raspberry bootloader and specified in the config.txt file.

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

Build from the commandline

To build the BSP on your host system, in a Command Prompt window (Windows) or a terminal (Linux), you must first build at the root directory of the BSP (*BSP_ROOT_DIR*), then you can build using the Makefile targets described previously in the ***$BSP_ROOT_DIR*/images** directory. To build your BSP, 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."
2. In the BSP's **images** directory (***$BSP_ROOT_DIR*/images**), type `make clean` to remove the default image files from the ***$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`. Running `make` does the following:
   - builds the BSP and creates its directories
   - places any images required for the board into the ***$BSP_ROOT_DIR*/images** directory

```
cd $BSP_ROOT_DIR
make
```

You should now have an IFS file called ifs-rpi4.bin.

NOTE: After you build, there might be multiple IFS files in the ***$BSP_ROOT_DIR*/images** directory if your **Makefile** builds multiple IFS files.

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

Build the BSP (IDE)

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

Build in the IDE

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

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 (see "Import to the QNX Momentics IDE" for details).
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-raspberrypi-bcm2711-rpi4**) 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**.

Build changes to the buildfile

You can make changes to the buildfile(s) 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 the previous section. 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.

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, **ifs-rpi4.bin**; it maps to the `ifs-rpi4.bin` target in the Makefile, you would type `ifs-rpi4.bin` as the target. For more information on the targets available, see the "Build the BSP (commandline)" section.

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

After your image builds, you should see it appear under the **images** folder in the IDE.

Build the IFS image used by U-boot (commandline)

You can use the commandline on Linux to build IFS images which can be used by U-boot.

Makefile targets in the BSP

The **Makefile** is located under the ***$BSP_ROOT_DIR*/images** directory. It defines additional targets which are not built by default:

`ifs-rpi4.raw`

Builds the QNX IFS image which is used by the U-boot "go" command to boot.

`ifs-rpi4.ui`

Builds the QNX IFS image which is used by the U-boot "bootm" command to boot.

Boot using U-boot "go" command

Below is the example how to used U-boot go command to boot QNX IFS image:

```
U-Boot> fatload mmc 0:1 0x80000 ifs-rpi4.raw
33190196 bytes read in 1485 ms (21.3 MiB/s)
U-Boot> env print fdtcontroladdr
fdtcontroladdr=3df3fa00
U-Boot> fdt addr -c
Control fdt: 3df3fa00
U-Boot> go 0x80000 0x3df3fa00
## Starting application at 0x00080000 ...
Hypervisor support disabled
MMU: 16-bit ASID 44-bit PA TCR_EL1=00000014b5183519
GICv2: 256 interrupts
GICv2: routing SPIs to gic cpu 0
cpu0: MPIDR=0000000080000000
cpu0: MIDR=410fd083 Cortex-A72 r0p3
cpu0: CWG=4 ERG=4 Dminline=4 Iminline=4 PIPT
cpu0: CLIDR=a200023 LoUU=1 LoC=2 LoUIS=1
cpu0: L1 Icache 48K linesz=64 set/way=256/3
cpu0: L1 Dcache 32K linesz=64 set/way=256/2
cpu0: L2 Unified 1024K linesz=64 set/way=1024/16
cpu0: GICv2 cpu interface 0
Loading IFS...done
..

Welcome to QNX 8.0.0 on RaspberryPi4B !

Starting wdtkick ...
Starting I2C driver ...
```

```
Starting PCI Server ...
Starting serial driver (/dev/ser1)
Starting SPI master driver ...
Starting SDMMC driver (/dev/sd0)
Path=0 - bcm2711
 target=0 lun=0     Direct-Access(0) - SDMMC: SD32G Rev: 8.5
Inform vc to load vl805 firmware
Starting USB xHCI controller in the host mode (/dev/usb/*)...
Starting devf-ram filesystem ...
Starting networking ...
Starting DHCP client ...
Starting SSH daemon ...
Starting devc-pty manager ...
Starting qconn daemon ...
Starting shell ...

# ifconfig genet0 | grep -e ether -e inet -e status
ether dc:a6:32:e7:95:d9
inet6 fe80::986a:3b5e:ee00:ba37%genet0 prefixlen 64 scopeid 0x5
inet 10.122.24.139 netmask 0xfffffe00 broadcast 10.122.25.255
status: active
```

Boot using U-boot "bootm" command

Below is the example how to used U-boot `bootm` command to boot QNX IFS image:

```
U-Boot> fatload mmc 0:1 0x80000 ifs-rpi4.ui
33190260 bytes read in 1478 ms (21.4 MiB/s)
U-Boot> fdt addr -c
Control fdt: 3df3fa00
U-Boot> bootm 0x80000 - 0x3df3fa00
## Booting kernel from Legacy Image at 00080000 ...
   Image Name:
   Image Type:   AArch64 Linux Kernel Image (uncompressed)
   Data Size:    33190196 Bytes = 31.7 MiB
   Load Address: 00080000
   Entry Point:  00080000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 3df3fa00
   Booting using the fdt blob at 0x3df3fa00
   Loading Kernel Image
   Using Device Tree in place at 000000003df3fa00, end 000000003df4fd49
size=58, ptr=9a8, limit=2000: 7ffe770
Starting kernel ...
Hypervisor support disabled
```

```
MMU: 16-bit ASID 44-bit PA TCR_EL1=00000014b5183519
GICv2: 256 interrupts
GICv2: routing SPIs to gic cpu 0
..
Welcome to QNX 8.0.0 on RaspberryPi4B !

Starting wdtkick ...
..
# ifconfig genet0 | grep -e ether -e inet -e status
ether dc:a6:32:e7:95:d9
inet6 fe80::9442:aa31:6bb5:bebb%genet0 prefixlen 64 scopeid 0x5
inet 10.122.24.158 netmask 0xfffffe00 broadcast 10.122.25.255
status: active
```

Driver Commands

The tables below provide a summary of driver commands for the Raspberry Pi 4 board.

Some of the drivers are commented out in the buildfile. To use these drivers on the target hardware, you may need to uncomment them in the buildfile, rebuild the image, and then 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 your system requirements. 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.

NOTE: Some drivers depend on other drivers, so it's sometimes necessary to start them in a specific order.

- 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 8.0 documentation.
- For more information about the drivers and commands listed here, see the QNX OS *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*. In some cases, the driver or command is specific to this BSP, in which case, you'll find the information in the "BSP-specific Drivers and Utilities" chapter.

Here's the list of drivers available for this board:

- Inter-integrated Circuit (I2C)
- Network
- PCI Server
- SD/MMC
- Serial
- SPI
- Startup
- USB OTG host controller (io-usb-otg stack)
- Watchdog

Inter-integrated Circuit (I2C)

Device	I2C
Command	`i2c-bcm2711 -p0xfe804000`
Required binaries	**i2c-bcm2711**

Required libraries	**libsecpol.so**, **libc.so**, **libgcc_s.so**
Source location	***$BSP_ROOT_DIR*/src/hardware/i2c/bcm2711**

For information about this driver, see the "i2c-bcm2711" section in the "BSP-specific Drivers and Utilities" chapter of this guide.

Network

Device	Ethernet
Command	`io-sock -m phy -m fdt -m phy_fdt -d genet -m pci -d em -d ix -d re -m usb -d axe -d axge -d cdce -d smsc`
Required binaries	`io-sock`, `dhcpcd`, `dhcpcd-run-hooks`, `ifconfig`, `if_up` ,`pfctl`, `ping`, `netstat`
Required libraries	**devs-genet.so**, **devs-em.so**, **devs-ix.so**, **devs-axge.so**, **devs-axe.so**, **mods-pci.so**, **mods-phy**, **mods-usb.so**, **mods-fdt.so**, **mods-phy_fdt.so**, **libsocket.so**, **libsecpol.so** and so on
Source location	Binary only

For more information about **devs-genet.so**, see "devs-genet.so" in the "BSP-specific Drivers and Utilities" chapter of this guide.

PCI Server

Device	PCI
Command	`pci-server -c --bus-scan-limit=1`
Required binaries	`pci-server`
Required libraries	**pci_hw-bcm2711-rpi4.so**, **pci_server-buscfg-generic.so**, **pci_debug2.so**, **pci_slog2.so**, **pci_cap-0x01.so**, **pci_cap-0x05.so**, **pci_cap-0x10.so**, **pci_cap-0x11.so**
Source location	Binary only

SD/MMC

Device	SD/MMC
Command (microSD driver on CPU board)	`devb-sdmmc-bcm2711 mem name=below1G sdio addr=0xfe340000,irq=158,bs=bmstr_base=0xc0000000 disk name=sd`
Required binaries	**devb-sdmmc-bcm2711**
Required libraries	**libcam.so**, **io-blk.so**, **cam-disk.so**, **libsecpol.so**, **libc.so**, **libgcc_s.so,**
Source location	***$BSP_ROOT_DIR*/src/hardware/devb/sdmmc**

Example of mounting a DOS partition on the microSD card to the **/fs/sd** directory:

```
devb-sdmmc-bcm2711 sdio addr=0xfe340000,irq=158,bs=bmstr_base=0xc0000000 disk name=sd
```

```
waitfor /dev/sd0 3
mount -t dos /dev/sd0t12 /fs/sd
```

Serial

Device	Serial
Command for MINI-UART (USB connected to RS-232 bridge for console)	`devc-serminiuart -b115200 -c500000000 -e -F -u1 0xfe215000,125`
Required binaries	`devc-serminiuart`
Required libraries	**libsecpol.so**, **libc.so**, **libgcc_s.so**
Source location	***$BSP_ROOT_DIR*/src/hardware/devc**

For more information about the serial drivers, see the "devc-serminiuart" section in the "BSP-specific Drivers and Utilities" chapter of this guide.

SPI

Device	SPI
Command	`spi-bcm2711`
Required binaries	`spi-bcm2711`
Required binaries	**libsecpol.so**, **libc.so**, **libgcc_s.so.1**
Required config file	**/etc/system/config/spi/spi.conf**
Source location	***$BSP_ROOT_DIR*/src/hardware/spi/bcm2711**

For more information about this driver, see the "spi-bcm2711" section in the "BSP-specific Drivers and Utilities" chapter of this guide.

Startup

Device	Startup
Command	`startup-bcm2711-rpi4 -v -D miniuart -W 2500 -u reg`
Required binaries	`startup-bcm2711-rpi4`
Required libraries	N/A
Source location	***$BSP_ROOT_DIR*/src/hardware/startup/boards/bcm2711** (common) ***$BSP_ROOT_DIR*/src/hardware/startup/boards/bcm2711/rpi4** (Raspberry PI 4 board)

In addition to the "`startup-*`" options described in the *Utilities Reference*, you can also use the following option with this command:

-u [reg | arg]

Add a flattened device tree (FDT) to the *asinfo* (Address Space Information) section of the system page.

Specify `reg` or `arg`, depending on how the FDT address is passed:

- `reg` — Populate *asinfo* using an FDT blob. The address of the FDT blob is in the x0 register.
- `arg` — The address of the FDT blob is passed by arguments in memory.

-W

Enable watchdog timer support. Ensure that you start the watchdog driver after when you use this option.

USB OTG host controller (`io-usb-otg` stack)

CAUTION: It is mandatory to run the `pci-server` before starting this driver since the USB driver on this board is dependent on PCI bus being functional.

USB host controller and OTG controller; host mode uses `io-usb-otg` stack.

Device	USB host controller
Command	XHCI: `io-usb-otg -t memory=below1G -d bcm2711-xhci pindex=0,memory=below1G`
Required binaries	`io-usb-otg`, `usb`, `devb-umass`
Required libraries	**devu-hcd-bcm2711-xhci.so**, **libsecpol.so**, **libc.so**, **libgcc_s.so.1**
Source location	Prebuilt only

Example of xHCI controller:

```
io-usb-otg -t memory=below1G -d bcm2711-xhci pindex=0,memory=below1G
waitfor /dev/usb/io-usb-otg 4
waitfor /dev/usb/devu-hcd-bcm2711-xhci.so 4 4
devb-umass cam pnp mem name=below1G disk name=umass
```

When a mass storage device is plugged in, `devb-umass` creates a resource manager under **/dev/hd*XtY*** where *X* represents the partition number and *Y* represents the partition type. For example, if you use a mass storage with one FAT32 partition (MBR), it creates two resource manager paths, which are **/dev/umass0** and **/dev/umass0t12**. You would mount the second resource manager, where in this example, "12" specifies a FAT32 filesystem (i.e., `mount -t dos /dev/umass0t12 /fs/usb0`).

For more information about the **devu-hcd-bcm2711-xhci.so** driver, see the "devu-hcd-bcm2711-xhci.so" section in the "BSP-specific Drivers and Utilities" chapter.

Watchdog

To enable the watchdog:

1. Launch `startup` with the `-W` option, with an optional parameter for timeout:

   ```
   startup-bcm2711-rpi4 -v -D miniuart -W 2500
   ```

2. Launch the watchdog timer utility early on in the boot script:

   ```
   wdtkick -W0x24:0x5A028E4C -W0x1c:0x5A000020
   ```

3. To adjust the watchdog timeout value of the PM_WDOG register, use the formula:

   ```
   register_val = ((timeout_in_ms * 67) | 0x5a000000)
   ```

   For example, use 0x5A028E4C for 2500 ms timeout value:

   ```
   ((2500 * 67) | 0x5a000000) = 0x28E4C | 0x5a000000 = 0x5A028E4C
   ```

Device	Watchdog timer
Commands	`wdtkick -W0x24:0x5A01000A -W0x1c:0x5A000020`
Required binaries	**wdtkick**
Required libraries	**libc.so**, **libgcc_s.so**
Source location	***$BSP_ROOT_DIR*/src/hardware/support/wdtkick**

BSP-specific Drivers and Utilities

These are the specific drivers and utilities for this BSP.

- devc-serminiuart
- devs-genet.so
- devu-hcd-bcm2711-xhci.so
- i2c-bcm2711
- spi-bcm2711
- gpio-bcm2711
- mbox-bcm2711

devc-serminiuart

Serial driver for the mini UART on the Raspberry Pi 4 board.

Syntax:

```
devc-serminiuart [-b number]
            [-c clock[/div]]
            [-C size]
            [-e] [-E]
            [-f] [-F]
            [-I size]
            [-o opt[,opt...]
            [-O number]
            [-s] [-S]
            [-u serial_unit_num]
            [-v]
```

Runs on:

QNX OS

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 Software Development Platform 8.0 documentation.

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

`-c` *clk[/div]*

Set the input clock rate and an optional divisor.

`-u` *serial_unit_num*

Set serial unit number. The default is 1.

Examples:

Disable the hardware and software flow control for a UART device with the base address of 0xFE215000 and the IRQ 125.

```
devc-serminiuart -b115200 -c500000000 -e -F -u1 0xfe215000,125
```

devs-genet.so

Driver for the Raspberry Pi GENET ETHERNET

Syntax:

```
io-soci -m fdt -m phy -d genet [option[,option ...]]

mount -T io-sock [-o option[,option...]] fdt
mount -T io-sock [-o option[,option...]] phy
mount -T io-sock [-o option[,option...]] genet
```

Runs on:

QNX OS

Options:

Use commas, not spaces, to separate the options.

`prefix=`*prefix*

Specifies the instance of `io-sock` that mounts this driver. You specify the prefix that creates an alternative stack when you start `io-sock`. Used only when you load a driver with `mount` after starting `io-sock`.

`qnxdev=`*device*

When Plug & Play is disabled, specifies the device instance to use. Use `devinfo -v` to determine the location value to use (e.g., `qnxdev=pci0:0:25:0`).

Description:

The **devs-genet.so** driver provides support for Raspberry Pi 4 Gigabit Ethernet devices.

This driver requires the **mods-phy.so** and **mods-fdt.so** drivers. For information on starting `io-sock` with a driver or loading a driver later using `mount`, see “Starting `io-sock` and Driver Management” in the *High-Performance Networking Stack (io-sock) User's Guide*.

devu-hcd-bcm2711-xhci.so

Driver for Extensible Host Controller Interface (XHCI) USB controllers on Raspberry Pi 4 board variants

Syntax:

```
io-usb-otg -d hcd-bcm2711-xhci [option[,option...]]
```

```
io-usb-otg -d bcm2711-xhci [option[,option...]]
```

Runs on:

QNX OS

Targets:

aarchle64

Options:

`cerr=`*retries*

Set the number of hardware retries. You can set *retries* to a value of 0 to 3, where 0 indicates unlimited retries. The default is 3 retries.

`debug_listener=`*retries*

Starts the PPS debug command handler at **/pps/usb/debug**.

`error_threshold=`*num*

The number (*num*) of cumulative errors before triggering the driver to reinitialize. The error-based triggering is disabled by default.

`ioport=`*addr*

USB controller register base address. The default is to scan the PCI bus.

`iosize=`*size*

The register space size. The default is 64 kilobytes.

`irq=`*num*

The interrupt request number.

`max_reinit_attempts=`*num*

The maximum number of reinitializations allowed. No limit is the default.

`memory=`*`name`*

Set the typed-memory name for descriptors. The default is **/memory/below4G**.

`nousb3`

Do not enable USB3 mode.

`pindex=` ***num***

Instance of controller on PCI bus to apply argument.

`prio=` ***num***

Set the pulse-handler thread's priority. The default is 21.

`soft_retries=` ***num_retries***

Set the number of software retries for a failed transfer. Supported only on control, bulk, and interrupt endpoints. Software retries are disabled by default.

`soft_retries_mask` ***=num***

Specifies the endpoints to use when software retries (`soft_retries`) are set. The endpoint to use is specified using a mask. The bit masks include:

- 0x1 – Control endpoint, which is set (ON) by default.
- 0x2 – Reserved endpoint, which isn't set (OFF). This bit should never be set.
- 0x4 – Bulk endpoint, which is set (ON) by default.
- 0x8 – interrupt endpoint, which is set (ON) by default.

The default mask is `0xD`, which indicates that the control, bulk, and interrupt endpoint bits are set (ON).

`verbose=`*`level`*

Set the verbosity level (0-7).

Description:

The `devu-hcd-bcm2711-xhci.so` server provides support for computers that have Extensible Host Controller Interface (XHCI) USB controllers. This server creates the **/dev/usb/devu-hcd-bcm2711-xhci.so device** device.

NOTE: The `devu-hcd-bcm2711-xhci` controller supports SuperSpeed, high, full, and low speed devices.

Examples:

Attach to the controller after scanning the PCI bus.

```
io-usb-otg -d hcd-bcm2711-xhci
```

i2c-bcm2711

Driver for inter-integrated circuits on the Raspberry Pi 4 Board (Model B).

Syntax:

```
i2c-bcm2711 [board_specific_option [board_specific_option]...]
         [generic_option [generic_option]...]]
```

Runs on:

QNX OS

Options:

There are Raspberry Pi 4 board-specific (single-dash) and generic (double-dash) options available.

CAUTION: The Raspberry Pi 4 board-specific options must always appear before generic options, otherwise the driver fails to initialize. For example, the correct sequence would as follows:

```
i2c-bcm2711 -v
```

However, if you incorrectly issue the command in the wrong sequence as shown here, it will fail:

```
i2c-bcm2711 -v
```

Raspberry Pi 4 board-specific options:

These are the board-specific options available for this driver.

-c *clockspeed*

Input clock. If this option isn't specified, the default is set as -1.

-i *irq*

The I2C interrupt event number. The default is 149.

-p address

The I2C physical base address.

-s slave_address

The slave address. The default is set as 0.

-v

Set to be verbose.

Generic options:

These are generic options for the driver.

--b

The default bus speed. If this option isn't specified, the default is 100000.

--u*unit*

The unit number to spend to the device name. The default is 0.

Example:

Specify a port of address 0xFE804000 with an IRQ of 149 to the first unit:

```
i2c-bcm2711  -p0xfe804000 --b100000
```

spi-bcm2711

Enhanced Serial Peripheral Interface (SPI)

Syntax:

```
spi-bcm2711 [-c config_file] [-v]
```

Runs on:

QNX OS

Options:

These are the options available for this driver.

-c config_file

Path to the SPI configuration file (default: **/etc/system/config/spi/spi.conf**)

Note: SPI config file template can be found at **"lib/io-spi/config_files/spi-template.conf"**

-v

Increase verbosity. This will override the "verbose" value in the config file

Examples:

Start SPI driver:

```
spi-bcm2711 -c /etc/custom/spi.conf
```

gpio-bcm2711

GPIO utility for BCM2711 SOC

Syntax:

```
gpio-bcm2711 [get|set|funcs|raw] [gpio number] [options]
```

Runs on:

QNX OS

Options:

GPIO is a comma-separated list of pin numbers or ranges (without spaces), e.g. 4 or 18-21 or 7,9-11. Valid [options] for gpio-bcm2711 set are:

```
ip      set GPIO as input
op      set GPIO as output
a0-a5   set GPIO to alternate function alt0-alt5
pu      set GPIO in-pad pull up
pd      set GPIO pin-pad pull down
pn      set GPIO pull none (no pull)
dh      set GPIO to drive to high (1) level (only valid if set to be an output)
dl      set GPIO to drive low (0) level (only valid if set to be an output)
```

Examples:

```
gpio-bcm2711 get              Prints state of all GPIOs one per line
gpio-bcm2711 get 20           Prints state of GPIO20
gpio-bcm2711 get 20,21        Prints state of GPIO20 and GPIO21
gpio-bcm2711 set 20 a5        Set GPIO20 to ALT5 function (GPCLK0)
gpio-bcm2711 set 20 pu        Enable GPIO20 ~50k in-pad pull up
gpio-bcm2711 set 20 pd        Enable GPIO20 ~50k in-pad pull down
gpio-bcm2711 set 20 op        Set GPIO20 to be an output
gpio-bcm2711 set 20 dl        Set GPIO20 to output low/zero (must already be set as an output)
gpio-bcm2711 set 20 ip pd     Set GPIO20 to input with pull down
gpio-bcm2711 set 35 a0 pu     Set GPIO35 to ALT0 function (SPI_CE1_N) with pull up
gpio-bcm2711 set 20 op pn dh  Set GPIO20 to ouput with no pull and driving high
```

mbox-bcm2711

Mailbox utility for BCM2711 SOC

Syntax:

```
mbox-bcm2711 [commandstring|command_id] [:|=] [parameters]
```

Runs on:

QNX OS

Options:

commandstring:

```
    clockrate
    clocks
    clockstate
    cmdline
    dmachan
    firmwarerev
    firmwarevar
    firmwarehash
    gpioconfig
    gpiostate
    macaddress
    maxclockrate
    maxtemperature
    maxvoltage
    memory
    minclockrate
    minvoltage
    model
    notifyxhcireset
    powerstate
    powertiming
    revision
    serial
    temperature
    turbo
    vcmemory
    voltage
```

Examples:

```
mbox-bcm2711 temperature
mbox-bcm2711 0x30006 [same as above]
mbox-bcm2711 clockrate:10
mbox-bcm2711 clockrate=10:100000000
```