Linux

For

BCT DB1

User Guide

Document Reference: BCTDB1 Linux User Guide

Document Issue: 1.0.3

Associated SDK release: 1.00

Authors: D Robinson, M Olejnik

Contents

1. Introduction 4
2. Environment Setup 4
2.1 Embedded Linux Components 4
2.2 Installation of the Embedded Linux build components 4
2.2.1 Git repo setup for the Linux kernel sources 5
2.3 Development Machine Setup 6
3. Building required components for Ubuntu Core 6
3.1 Installing the Ubuntu Core root file system 6
3.2 Compiling the Linux Kernel 7
3.3 Compiling the Little Kernel Bootloader 8
3.4 SD card formatting tool 9
3.5 Firmware binary blobs 10
3.5.1 GPU, VPU, WiFi & BT firmware 10
3.5.2 Hexagon DSP, GPS 10
3.6 Ubuntu Core system components summary 11
4.0 Building embedded Linux with Buildroot 11
4.1 Buildroot introduction 11
4.2.1 Video player demo, with ffplay support 11
4.2.2 QT5 12
4.3 Buildroot outputs 12
5. Updating the firmware / software on DB1 13
6. BCT DB1 Hardware Setup in Linux 13
6.1 Debug Serial Console 13
6.2 BCT DB1 Serial Ports 13
6.2.1 RS-485 Manual Transmit Control 14
6.2.2 RS-485 Automatic Transmit Control 14
6.2.3 RS-422 15
6.3 BCT DB1 GPIO 15
6.4 DB1 Wi-Fi Operation 17
6.5 DB1 Bluetooth Operation 17
6.6 DB1 Audio 18

6.7 DB1 uSD Card 18
6.8 DB1 Watchdog 18
6.9 DB1 Power management 18
6.10 DB1 Class-D amplifier 18
6.11 LCD Backlight 18
6.12 GPS 19
7. Little Kernel bootloader 19
8. QT5 Application development introduction 20
8.2.1 Install QT Creator to the development machine 20
8.2.2 Setup the DB1 environment in QT Creator 20
8.2.3 Setup a simple QT5 “Hello World” application 24
Appendix A - Known Problems 28
Appendix B - Change Log 28

1. Introduction

The content of this document provides information required to start building Linux operating systems for the BCT DB1 platform. It covers:

- The tools and components required for building a Linux operating system
- How to install the build components
- How to compile the Little Kernel boot loaders stand alone
- How to compile the Linux Kernel 4.14.96 stand alone
- How to setup a root file system using Ubuntu 18.04 LXDE
- How to build a root file system including QT5 using build root
- How to boot Linux on the DB1 platform
- How to setup and deploy a simple QT5 application to DB1

2. Environment Setup

2.1 Embedded Linux Components

The components involved in a typical Embedded Linux system targeting the ARM architecture are:

- Bootloader
- Linux Kernel
- Root file system.

Little Kernel bootloader was ported to provide the bootloader functionality for the DB1.

Linux kernel 4.14.96 have been ported to be compatible with the BCT DB1 platform.

Pre-built Ubuntu root file systems are provided for demonstration purposes. As an alternative to Ubuntu, a Buildroot environment is provided to allow bespoke root file systems to be generated for the DB1 platform. Section 3 describes the procedure for building an image with the Ubuntu file system; section 4 describes the procedure for building an image with Buildroot.

The DB1 software components above have all been tested to compile using an Ubuntu 18.04 LTS development machine.

2.2 Installation of the Embedded Linux build components

Create the top level build BSP directory and grant it universal read/write/execute access as follows:

```
cd /
sudo mkdir embedded
sudo chmod 777 embedded
cd embedded
```

Copy the latest DB1 Linux components to the "/embedded" directory. Sources can be distributed in different ways, but usually they can be downloaded from our web site.
https://www.bluechiptechnology.com/product/db1/

Download the Linux source code for DB1 using the command:

```
wget http://dl.bluechiptechnology.com/dl/db1/software/db1linuxv100.tar.bz2
wget http://dl.bluechiptechnology.com/dl/db1/software/db1linuxv100.tar.bz2.md5
```

Check that the integrity of the download is OK by issuing the following command:

```
md5sum -c db1linuxv100.tar.bz2.md5
```

Extract the tar ball by issuing the command:

```
tar xvjf db1linuxv100.tar.bz2
```

Once extracted the build components will be laid out in the following structure on the development machine. The BSP is capable of building images containing kernel 4.14.96. The first directory ("embedded") is the folder created in the root of the file system.

DB1 embedded development directory: **/embedded/projects/db1/**

Directory	Description
bootloader/lk	Source code for the Little Kernel boot loader including configuration for BCT TDB1
bootloader/signlk	Source code for signing tool. The tool is used to sign the bootloader.
kernel_4_14	4.14.96 Linux kernel source code with configuration for BCT DB1
output_kernel_4_14	Staging directory used for holding the build product of Linux kernel 4.14.96. Also contains variants of DB1 kernel images and a kernel module archive after the kernel build is complete.
buildroot	Buildroot top directory, containing buildroot 2020.05 source release with configurations for building root-fs images for DB1. Pre-compiled Linux kernels and root-fs images are also provided.
sd_card_formatter	Linux tool to create a bootable SD card for DB1.
firmware	Binary blobs and tools for peripheral support.

2.2.1 Git repo setup for the Linux kernel sources

In order to keep the Linux kernel source code up-to-date with Blue Chip's patches a git remote repository can be set-up. This step is not required but may save you time installing future kernel modifications manually. To assign a new public git repo to the kernel source code issue the following commands:

```
sudo apt-get install git
cd /embedded/projects/db1/kernel_4_14/
git remote rm origin
```

```
git remote add origin
http://dl.bluechiptechnology.com/dl/dm1/software/linux/L4.14.96/linux-db1.git
git remote update
git branch --set-upstream-to=origin/master master
git pull
```

2.3 Development Machine Setup

Where possible build scripts have been provided for the various components included with the Linux SDK for BCT DB1. These scripts presume the following has been setup on the Ubuntu 18.04 LTS development machine.

The following packages are known to be required on an Ubuntu 18.04 development machine to successfully build the components.

```
sudo apt-get install build-essential
sudo apt-get install mkbootimg
sudo apt-get install flex
sudo apt-get install bison
sudo apt-get install lzop
sudo apt-get install ncurses-dev
sudo apt-get install gcc-aarch64-linux-gnu
sudo apt-get install gcc-arm-none-eabi
sudo apt-get install scons
sudo apt-get install cmake git
```

An ARM 64 bit cross-compiler is required to build the Linux software. We use the default one provided by Ubuntu OS. To check the cross-compiler is available on your machine use the following command:

```
aarch64-linux-gnu-gcc --version
```

The software was built with version 7.5 of the cross-compiler.

3. Building required components for Ubuntu Core

3.1 Installing the Ubuntu Core root file system

There are many Linux distributions available that are compatible with BCT DB1. Ubuntu was chosen as the main distribution for DB1, as it has a large pre-compiled package database, and easy to use configuration tools. Ubuntu is not the ideal choice for all Linux projects, but it will allow a basic operating system to be constructed quickly to allow evaluation of the BCT DB1. For smaller embedded OS requirement consider using Buildroot to generate a root file system from scratch. See section 4 for details.

The process of creating an Ubuntu root file system for BCT TM1/HB5 consists of the following steps.

The process of creating a bootable Linux SD card image for BCT DB1 consists of the following steps.

- Format an SD card via formatting tool located in sd_card_formatter directory.
- Mount the SD card root-fs partition (labelled 'sdrootfs').
- Extract an Ubuntu root-fs image into the SD card root-fs partition.
- Add kernel modules and other specific support to the root file system.
- Copy the Linux kernel image to the 'boot' partition on the SD card.
- Boot the generated Ubuntu root file system on a BCT DB1.
- Configure the Ubuntu root file system using, apt-get, synaptic package manager, or another package manager.

For convenience a pre-built Ubuntu 18.04 root file system can be downloaded from the DB1 product page (https://www.bluechiptechnology.com/products/db1/) . Install the downloaded root-fs image by issuing the following commands:

```
tar xjf rootfsubuntu1804lxde.tar.bz2 -C <SD_card_rootfs_mount_directory>
```

At this point Linux Kernel modules and any other specific support files must be added to the root file system. The following sections describe this process.

3.2 Compiling the Linux Kernel

To compile the kernel we must enter the root of the kernel source tree, optionally make some configuration changes and use **make** to start the compile. Issue the following commands.

```
cd /embedded/projects/db1/kernel_4_14/
./build.sh
```

When the compilation process has completed it will leave the produced files in **/embedded/projects/db1/output_kernel_4_14** directory. Linux kernel files (Android format is used) are named "./boot-*.bin", and kernel modules packaged as "./modules.tgz".

The DB1 kernel implements the device tree model for configuring a hardware platform. The binary form of the device tree (DTB) is bundled inside the kernel file. The DB1 kernels are built for several screen configurations and booting media.

Kernel file definition	Description
boot-07c.bin	DB1 with 7 Inch LCD and capacitive touch, for emmc boot
boot-07c_sd.bin	DB1 with 7 Inch LCD and capacitive touch, for uSD card boot
boot-07r.bin	DB1 with 7 Inch LCD and resistive touch, for emmc boot
boot-07r_sd.bin	DB1 with 7 Inch LCD and resistive touch, for uSD card boot
boot-09c.bin	DB1 with 9 Inch LCD and capacitive touch, for emmc boot
boot-09c_sd.bin	DB1 with 9 Inch LCD and capacitive touch, for uSD card boot
boot-12c.bin	DB1 with 12 Inch LCD and capacitive touch, for emmc boot
boot-12c_sd.bin	DB1 with 12 Inch LCD and capacitive touch, for uSD card boot

build.sh is an example of a script file that simplifies the process of building BCT DB1 Linux components. Please study the script for an understanding of the build steps and built components.

If changes are required to the kernel configuration the command "`make menuconfig`" can be used to present a menu based configuration utility for the Linux kernel. If any changes are made using the menuconfig tool, the "`./build.sh`" command must be re-issued to recompile the kernel and modules.

Once the kernel has been compiled, the kernel file must be written to the 'boot' partition of the booting drive (emmc or uSD card) and the kernel modules must be copied to the DB1 root file system. Ubuntu OS keeps the modules installed in **/lib/modules** directory, Buildroot system keeps them in **/usr/lib/modules directory**. For example the directory structure after installation of kernel modules to DB1 rootfs on Ubuntu OS will look like this:

```
/lib/modules/4.14.96-g46153a175/*
```

The modules version is determined by the modules sub-directory name. In this example the modules version is 4.14.96-g46153a175.

The kernel version installed in the boot partition and the modules version in the root-fs must match. If they don't the Linux kernel will not load the kernel modules which may lead to issues with peripherals. To check the Linux kernel version issue the following command on DB1 terminal:

```
uname -r
```

The kernel installation involves writing the kernel file to 'boot' partition on the booting media. The following command can be used to install the kernel.

```
sudo dd if=boot-07c_sd.bin of=/dev/disk/by-partlabel/boot bs=1M
```

The above example used a kernel binary for 7" screen (07) capacitive touch-screen (c) which boots from uSD card (_sd).

3.3 Compiling the Little Kernel Bootloader

Little Kernel bootloader has been ported to work with BCT DB1. Its purpose is to initialise the hardware, and boot a Linux operating system.

To build U-Boot for BCT DB1 issue the following commands.

```
cd /embedded/projects/db1/bootloader/lk
./build_emmc_linux.sh
./build_sd_linux.sh
```

The compiled boot loader for emmc boot media is "aboot_emmc.mbn" file. The compiled bootloader for uSD boot media is "aboot_sd.mbn" file. Most common installations use the emmc version of the bootloader on emmc boot media and sd card version of the bootloader on sd card media. However it is possible to install sd version of the bootloader to emmc in order to force the booting from uSD card.

The bootloader installation involves writing the bootloader file to 'aboot' partition on the booting media. The following command can be used to install the bootloader.

```
sudo dd if=aboot_sd.mbn of=/dev/disk/by-partlabel/aboot bs=1M
```

3.4 SD card formatting tool

The boot media for BCT DB1 must have particular partition layout and some of the partitions must contain SOC specific binary blobs. To simplify the creation of the bootable uSD card a formatting script is available in /embedded/projects/db1/sd_card_formatter.

IMPORTANT: before using the formatting tool make sure to backup all important files from your uSD card. The tool will overwrite the whole contents of the uSD card!

WARNING: the tool will format the device that is specified by the user as a parameter. If an incorrect device is specified the tool will delete all data from such device.

The tool can print out the list of devices present in your Linux system. To print the list of device issue the following command.

```
cd /embedded/projects/db1/sd_card_formatter
./format_sd_card.sh
```

The list might look like this:

```
NAME    MIN:MAJ RM      SIZE    RO      TYPE    MOUNTPOINT
sdd     8:48    1       7.5GB   0       disk
```

This list shows there is one disk device of 7.5 GB size called 'sdd'. Make sure all partitions on the device are unmounted – that is the uSD card is not in use by the operating system. To do that, right click on device Icon displayed on Desktop and select the Unmount option (do not use Eject option). The Size column displayed in the list is also a good indicator the device is an uSD card because the uSD card capacity should approximately match the value in the column.

To format particular device issue the following command.

```
./format_sd_card.sh /dev/sdX
```

The 'sdX' is the name of the device as seen in the Name column of the list. A confirmation prompt will be displayed before the formatting starts. Type 'y' without quotes and press Enter key to confirm formatting of the specified device. When the formatting process finishes verify the uSD card partition layout by issuing the following command.

```
lsblk
```

The device should now have 9 partitions sdX1 to sdX9. The root-fs (/dev/sdX9) partition will be formatted to ext4 format and will contain no files.

3.5 Firmware binary blobs

The SOC manufacturer provides several binary blobs required to support peripherals which are either not maintained in Linux kernel, or their default driver does not support certain features of the hardware. Specifically these are GPU and VPU accelerators, WiFi & Bluetooth and GPS hardware. The binary blobs are located in /embedded/projects/db1/firmware/rootfs_blobs/rootfs_blobs_<32|64>bit.tgz archive. The archive contains a file structure compatible with Ubuntu root-fs, therefore to install these binary blobs files to the correct place the archive needs to be decompressed on the target (DB1) root-fs.

```
cd <target-root-fs>
sudo tar xzf  /embedded/projects/db1/firmware/rootfs_blobs/rootfs_blobs_64bit.tgz
```

Please note that the command above used the archive for 64 bit Linux OS. For 32bit Linux OS use the 'rootfs_blobs_32bit.tgz' archive.

The Linux kernel searches for the binary blobs on the root-fs during boot time and if it finds them then the hardware feature/peripheral will be enabled.

3.5.1 GPU, VPU, WiFi & BT firmware

The following table describes the location of the binary blobs on the root-fs.

Peripheral	Blob location
GPU	/lib/firmware/a300*.fw
VPU	/lib/firmware/qcom/venus-1.8/*
WiFi & Bluetooth	/lib/firmware/wcnss.*, /lib/firmware/qcom/wlan/prima/*

The binary blobs files listed above are the same for 64 bit and 32 bit Linux OS.

3.5.2 Hexagon DSP, GPS

The GPS hardware employs Hexagon DSP and requires binary blobs and several software components to function. By default the Hexagon DSP and the GPS is disabled. It can be enabled by issuing the following commands on DB1 and rebooting DB1 to activate the changes.

```
cd /lib/firmware
sudo ./set_modem.sh enable
sudo systemctl enable rmtfs
sudo systemctl enable qrtr
```

The binary blob files location is as follows: /lib/firmware/modem.*, /lib/firmware/mba.mbn, /boot/modem_fs*

The Hexagon DSP requires the following user space tools to be present on root-fs. These are 'rmtfs' , 'qrtr-cfg', 'qrtr-ns' located in /usr/local/bin directory and 'libqrtr*' located in /usr/local/lib directory. The user space tools and GPS supporting tools can be recompiled from source. Both 64 bit and 32 bit Linux OS for DB1 are supported. In order to cross-compile for 32 bit Linux OS a 32 bit compiler has to be installed by issuing the following command on the build machine.

```
sudo apt install gcc-arm-linux-gnueabihf
```

To compile 64 bit tools supporting Hexagon DSP and GPS issue the following commands.

```
cd /embedded/projects/db1/firmware
./build.sh arm64
```

Or alternatively for 32 bit target Linux OS issue the following commands.

```
cd /embedded/projects/db1/firmware
./build.sh arm
```

Once the compilation finishes the tool binaries will be located in sub-directories 'rmtfs', 'qrtr' and 'gpsd'.

3.6 Ubuntu Core system components summary

So far this document has described how to set up a build environment and how to build the various components of a Linux Ubuntu Core operating system for BCT DB1. The built components are as follows:

Component	Location
Linux Kernel	/embedded/projects/db1/output_kernel_4_14/boot-*.bin
Linux Kernel modules	/embedded/projects/db1/output_kernel_4_14/modules.tgz
Bootloader for emmc	/embedded/projects/db1/bootloader/lk/aboot_emmc.mbn
Bootloader for uSD card	/embedded/projects/db1/bootloader/lk/aboot_sd.mbn

4.0 Building embedded Linux with Buildroot

4.1 Buildroot introduction

Buildroot is a build system that aids the process of building various components of an Embedded Linux system in a single environment. We think Buildroot is easy to get to grips with, and provides a reasonable amount of package support.

Buildroot 2020.05 is provided in the Linux download for DB1. It contains two sample configurations which build the root file system. Linux kernel files are pre-built to simplify the process.

4.2.1 Video player demo, with ffplay support

This configuration is designed to be small in size and demonstrate a quick boot time. A media player (ffplay) is included in the configuration, and is configured to automatically play videos found in the root of a USB flash drive during boot. AVI, and MP4 video formats are supported and video resolutions don't have to match the target LCD screen resolution.

To build the video player configuration issue the following commands.

```
cd /embedded/projects/db1/buildroot/buildroot-2020.05
./build_video_demo.sh
```

4.2.2 QT5

This configuration will build a root file system containing QT5 libraries and QT5 sample applications. To aid in remote QT5 application deployment, the image is configured with an SSH server, and will print the local IP address to the LCD screen at boot time. The root user is configured with a password of “password”.

To build this configuration issue the following commands.

```
cd /embedded/projects/db1/buildroot/buildroot-2020.05
./build_qt5_demo.sh
```

Please note that the build of this configuration might take several hours depending on the machine you build on. Also, make sure there is enough of free disk space on the build machine. Typically you will need approx. 20 GB of free disk space.

4.3 Buildroot outputs

After the build completion of the example configurations, the built components of the embedded Linux system are as follows:

Video demo

Component	Location
Root file system	/embedded/projects/db1/buildroot/output_video_demo/images/rootfs.tar
Linux Kernel	/embedded/projects/db1/buildroot/kernel/boot-<type>.bin

QT5 demo

Component	Location
Root file system	/embedded/projects/db1/buildroot/output_qt5_demo/images/rootfs.tar
Linux Kernel	/embedded/projects/db1/buildroot/kernel/boot-<type>.bin
Host compiler for Qt5 apps	/embedded/projects/db1/buildroot/output_qt5_demo/host/aarch64-buildroot-linux-gnu

To test the demos on BCT DB1 both Root file system and Linux kernel need to be installed to a bootable media – a uSD card. Check section 3.4 for information about creating bootable uSD card media.

To aid installation of root-fs to a uSD card an installation script can be used. To install the demo to a uSD card’s root-fs ensure a uSD card with the correct partition layout is inserted and is not mounted. Then issue the following command.

Video demo

```
cd /embedded/projects/db1/buildroot
./install_rfs_video.sh
```

QT5 demo

```
cd /embedded/projects/db1/buildroot
./install_rfs_qt5.sh
```

Ensure a matching kernel is installed on the uSD card by running the following command.

```
cd /embedded/projects/db1/buildroot/kernel
sudo dd if=boot-<type>_sd.bin of=/dev/disk/by-partlabel/boot bs=1M
```

The <type> is the BCT DB1 hardware/screen type. Check the kernel file table in section 3.2 for more information.

5. Updating the firmware / software on DB1

Not available at the moment. The installation tool will be developed in due time.

6. BCT DB1 Hardware Setup in Linux

6.1 Debug Serial Console

Linux terminal can be accessed via serial console on BCT DB1. By default the bootloader and Linux are configured to use the RS232 port available on P6 (50 way connector) of the DB1. By default the board is set to communicate at 115200, 8, n, 1. Before turning on the DB1 for the first time it is recommended that this port is connected to a PC with terminal emulator software running. E.g. HyperTerminal, Teraterm or Minicom.

6.2 BCT DB1 Serial Ports

The UARTs on the DB1 are mapped as follows:

DB1 Header	Linux Device Name	Logical name
P6: pin 21- TX, pin 23 – RX, pin 19 - GND	/dev/ttyMSM0 (Linux console port)	COM1
P6: pin 27 - TX, pin 29 – RX, pin 25 - GND	/dev/ttyMSM1	COM2
P6: pin 33 – TXP, pin 35 – TXN, pin 37 – RXP, pin 39 – RXN, pin 31 - GND	/dev/ttySC0	COM3

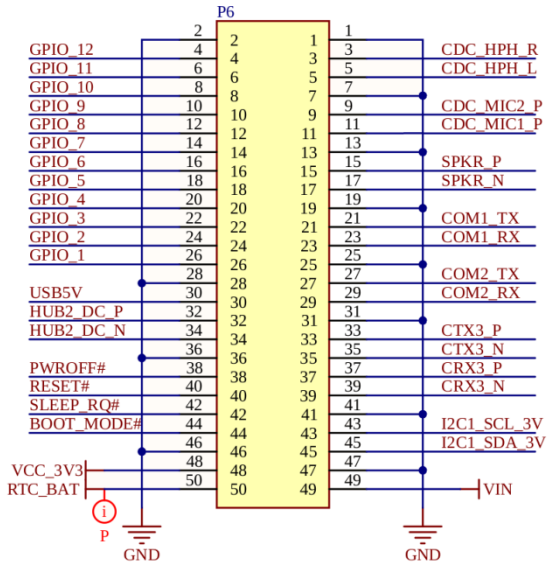


Figure 1: Pinout of the 50 way DB1 expansion connector.

6.2.1 RS-485 Manual Transmit Control

/dev/ttySC0 is an RS485 / RS422 compatible port which has a receive enable signal. The signal is set to logical level 1 in DTS therefore the port will function in RS422 mode by default. If you intend to use the serial port in RS485 mode, you need to configure it in your software – see the next section for more information.

The serial port does not support manual transmit control, the automatic transmit control is supported.

6.2.2 RS-485 Automatic Transmit Control

The DB1 UART hardware of COM3 is designed to use hardware flow control which means the automatic transmit control is always in place. The Linux API for configuring the UART in RS-485 mode can be viewed using the following link.

https://www.kernel.org/doc/Documentation/serial/serial-rs485.txt

The important configuration flag in the linked example is the SER_RS485_RX_DURING_TX. This flag switches the COM3 port to RS422 mode when set and to RS485 mode when unset.

The BCT application note RS485_BETA_APP_NOTE also provides useful information on implementing RS-485 with the DB1 platform.

http://dl.bluechiptechnology.com/dl/tm1/Documentation/RS-485_Application_Note_For_BCT_Beta.pdf

6.2.3 RS-422

The /dev/ttySC0 serial port works by default in RS-422 full duplex mode. If it was switched to RS485 mode and needs to be switched back to RS422 mode the following steps should done in the user space application:

- Enable RS485 mode

```
rs485conf.flags |= SER_RS485_ENABLED;
```

- Enable full duplex mode

```
rs485conf.flags |= SER_RS485_RX_DURING_TX;
```

- Issue the serial port ioctl

```
ioctl (fd, TIOCSRS485, &rs485conf);
```

See the linked documentation in the previous section for more information.

6.3 BCT DB1 GPIO

The recommended way to access the GPIO is using the SYSFS interface. This can be done using the command line (or scripts), or can be done from inside an application.

The Linux GPIO documentation can be found here:
https://www.kernel.org/doc/Documentation/gpio/sysfs.txt

This following page also has some useful examples:
http://falsinsoft.blogspot.co.uk/2012/11/access-gpio-from-linux-user-space.html

By default, the GPIOs on DB1 are setup as inputs with hardware pull ups.

The logical GPIOs on the P6 header of DB1 map to the physical GPIO pins on the SOC as follows:

Logical GPIO	Physical GPIO	Direction GPIO
GPIO 1	69	70
GPIO 2	107	49
GPIO 3	108	50
GPIO 4	109	51
GPIO 5	110	52
GPIO 6	111	53
GPIO 7	112	54
GPIO 8	113	55
GPIO 9	114	56
GPIO 10	115	57
GPIO 11	117	58
GPIO 12	118	59

The DB1 board uses voltage level shifter ICs to raise the GPIO voltages on P6 connector to 3.3V which ensures compatibility with other BCT products. That also means the voltage level shifters need

to be controlled to allow IN/OUT direction of the GPIO signal. Typically, when the logical GPIO needs to be set for Output, then the Physical GPIO has to be set as 'out' and also the output value on the Direction control GPIO has to be set to 1. If the logical GPIO needs to be set for Input, then the Physical GPIO has to be set as 'in' and also the output of the Direction control GPIO has to be set to 0. Please note that the direction of the Direction control GPIO is always 'out' and the output value defines whether the voltage level shifter lets the signal in (0) or out (1).

To setup and control GPIO3 as Output with value 1 the following commands would be used:

```
sudo su
echo 50 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio50/direction
echo 1 > /sys/class/gpio/gpio50/value
echo 108 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio108/direction
echo 1 > /sys/class/gpio/gpio108/value
```

To setup and control GPIO4 as Output with value 0 the following commands would be used:

```
sudo su
echo 51 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio51/direction
echo 1 > /sys/class/gpio/gpio51/value

echo 109 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio109/direction
echo 0 > /sys/class/gpio/gpio109/value
```

To setup and control GPIO5 as Input the following commands would be used:

```
sudo su
echo 110 > /sys/class/gpio/export
echo in > /sys/class/gpio/gpio110/direction

echo 52 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio52/direction
echo 0 > /sys/class/gpio/gpio52/value
```

IMPORTANT: To prevent potential GPIO port damage the set-up of the Physical GPIO and Directional GPIO needs to be performed in certain order:
For In direction: The Physical GPIO needs to be set first, then the Direction GPIO
For Out direction: The Direction GPIO needs to be set first, then the Physical GPIO

To aid setting up logical GPIOs from command line we made few scripts in Lubuntu's OS for DB1. These are as follows: egpiomap, egpioconf, egpioset and egpioget.

```
egpiomap
```

Prints the list of Logical and Physical GPIOs and their corresponding Direction GPIO numbers.

```
sudo egpioconf 3 out 1
```

Configures logical GPIO 3 as output and sets the output value high.

```
sudo egpioconf 4 out 0
```

Configures logical GPIO 4 as output and sets the output value low.

```
sudo egpioconf 5 in
```

Configures logical GPIO 5 as input. The input value depends on the GPIO incoming signal, or is high if the GPIO pin is disconnected.

```
sudo epgioset 3 0
```
Sets the logical GPIO3 output value low

```
sudo epgioset 3 1
```
Sets the logical GPIO3 output value high

```
sudo epgioget 5
```
Gets and prints the logical GPIO5 current value. Works for both input and output GPIOs.

6.4 DB1 Wi-Fi Operation

Presuming that the firmware binary blobs and the kernel modules corresponding to the kernel version have been installed into the root files ystem, the Wi-Fi kernel modules should be automatically loaded upon Ubuntu OS start-up. To verify the modules are loaded issue the following command.

```
lsmod | grep wcn36xx
```

If the modules were successfully loaded the wlan0 network device should be present. This can be checked by issuing the following command.

```
ifconfig –a
```

The following commands can be used to enable the wlan0 interface, and scan for networks.

```
ifconfig wlan0 up
iw wlan0 scan | grep SSID
```

6.5 DB1 Bluetooth Operation

Presuming that the firmware binary blobs and the kernel modules corresponding to the kernel version have been installed into the root file system, the BT kernel modules should be automatically loaded upon Ubuntu OS start-up. To verify the modules are loaded issue the following command.

```
lsmod | btqcomsmd
```

If the module was successfully loaded the hci0 BT device should be present. This can be checked by issuing the following command.

```
hciconfig
```

The following commands can be used to enable the BT interface, and scan for devices.

```
sudo hciconfig hci0 up
sudo hcitool lescan
```

6.6 DB1 Audio

The audio CODEC featured on DB1 implements the standard Linux ALSA API framework. Standard commands like alsamixer, aplay, arecord, speaker-test will work.

6.7 DB1 uSD Card

The uSD card connector featured on DB1 is mapped to mmc1 in the Linux kernel. When an uSD card is inserted the raw sectors of the card can be accessed via /dev/mmcblk1 device.

6.8 DB1 Watchdog

The SOC on the DB1 board includes a watchdog that can reset the system. It is implemented using the standard Linux Watchdog API.

https://www.kernel.org/doc/Documentation/watchdog/watchdog-api.txt

6.9 DB1 Power management

DB1 implements power and thermal management under software control. This can be configured using the DVFS framework in the Linux kernel.

https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

DB1 supports suspend to RAM, which allows the system to enter a low power mode while retaining the contents of RAM. This allows the system to resume to an operational state in a very short period of time. To enter suspend to RAM mode the following command can be issued.

```
sudo su
echo mem > /sys/power/state
```

The SLEEP_RQ# signal on DB1 connector is configured to wake the system up while it is in suspend to RAM mode. See Figure 1 in section 6.2 for the pin assignment of the signal. The signal is asserted when it is tied to ground.

6.10 DB1 Class-D amplifier

The class D amplifier implemented on DB1 can be controlled using physical GPIO 105.

6.11 LCD Backlight

The LCD backlight can be controlled using the standard Linux sysfs backlight class.

https://www.kernel.org/doc/Documentation/ABI/stable/sysfs-class-backlight

The following commands would set the backlight to 0%:

```
sudo su
echo 0 > /sys/class/backlight/backlight/brightness
```

The following commands would set the backlight to 50%:

```
echo 50 > /sys/class/backlight/backlight/brightness
```

The following commands would set the backlight to 100%:

```
echo 100 > /sys/class/backlight/backlight/brightness
```

6.12 GPS

Ensure the GPS antenna is connect to DB1 and is able to receive satellite signal.

Ensure the firmware blobs and GPS tools are installed as described in section 3.5.2.

GPS monitor can be started by using the following commands.

```
sudo gpsdctl add pds://any
gpsmon
```

7. Little Kernel bootloader

The Little Kernel bootloader version was ported to the DB1 platform. At a high level its primary purpose is to load the Linux kernel archive from the booting media, split the archive to separate parts (kernel executable, boot arguments, initial ramdisk etc.) and then pass execution over to the Linux kernel.

The Little Kernel binary can either load the Linux kernel from emmc or from the uSD card. Therefore there are 2 versions of the bootloader available to support each booting media: aboot_emmc.mbn and aboot_sd.mbn respectively. The bootloader can be built from source code, chapter 3.3 describes how to do that. The bootloader binary is installed on partition labelled 'aboot', the Linux kernel archive is installed on partition labelled 'boot'.

The Linux kernel archive compatible with the bootloader has to have an Android format. Android format keeps all parts required for booting a Linux kernel in a single file. This reduces the bootloader complexity as no file system support is needed in order to load the parts from individual files. Linux kernel archive that has the Android format is produced by 'mkbootimg' tool. Its usage can be found in a script that is executed in the final stage of building the Linux kernels for DB1. See the contents of the script located here: /embedded/projects/db1/output_kernel_4_14/pack_kernels.sh

The most important part of the Linux kernel assembly is in the following command.

```
mkbootimg    --kernel Image.gz+dtb \
             --ramdisk initrd.img \
             --output boot.bin \
             --pagesize 2048 \
             --base 0x80000000 \
             --cmdline "<kernel boot arguments>"
```

Little Kernel bootloader expects the Device Tree Binary (.dtb file) to be appended to the Linux kernel binary itself. Such file is passed as the '--kernel' parameter. The '--ramdisk' parameter specifies the kernel's initial ramdisk in gzipped cpio format. The '--output' parameter specifies the file name of

the tool's product – that's the file that can be stored in the 'boot' partition. The '--cmdline' parameter specifies the kernel boot arguments. The rest of the parameters are miscellaneous constants.

Little Kernel bootloader does not support interactive modification of parameters or boot variables via serial console.

8. QT5 Application development introduction

The following section describes how QT creator can be installed and configured to deploy a simple "Hello World" app to the DB1 platform over Ethernet.

8.2.1 Install QT Creator to the development machine

On the same development machine that was used build the QT5 Buildroot root file system issue the following command to install QT creator.

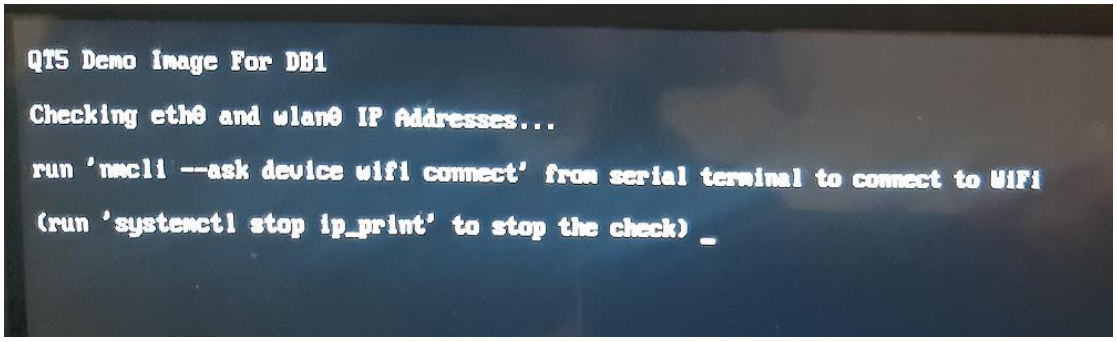
```
sudo apt install qtcreator
```

The recommended Linux OS (Ubuntu 18.04) installs QT Creator version 4.5.2 and is based on QT 5.9.5.

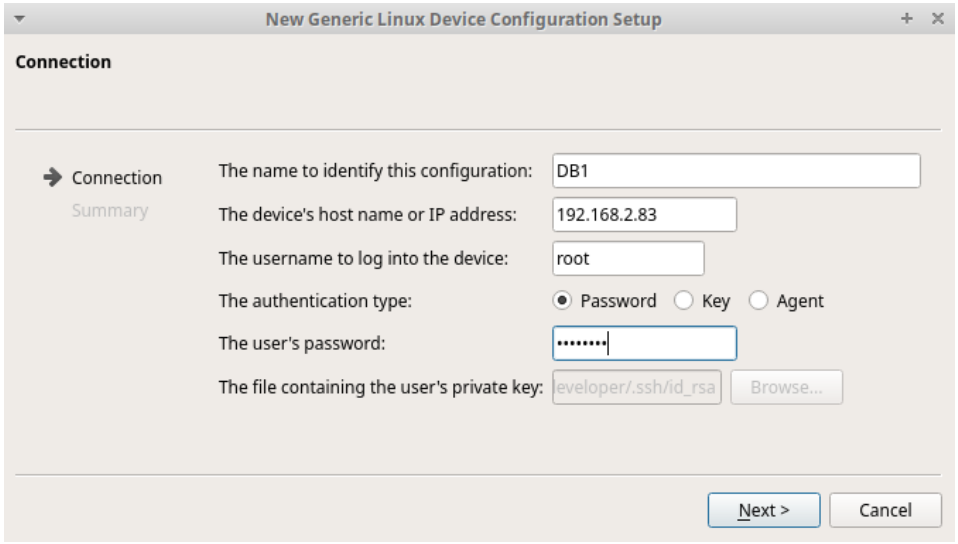
8.2.2 Setup the DB1 environment in QT Creator

QT Creator uses the notion of "Kits" which refer to development environment configurations, targeting specific architectures and devices. By default only a single kit is installed in QT creator that targets applications running in the host environment. This section will focus on the setup of a kit targeting DB1 running the Buildroot generated QT5 root file system created in section 4.2.2.

1. Ensure section 4.2.2 has been followed to create a QT5 based root file system for DB1
2. Prepare a bootable uSD card for DB1 and install the QT5 demo root-fs and Linux kernel to the uSD card as described in section 4.3
3. Boot the DB1 unit from the uSD card with an Ethernet cable attached. The LCD will display the IP address obtained via DHCP. Make a note of this IP Address. Ensure that the Boot priority jumper on the DB1 board is unplugged in order to boot from the uSD card.

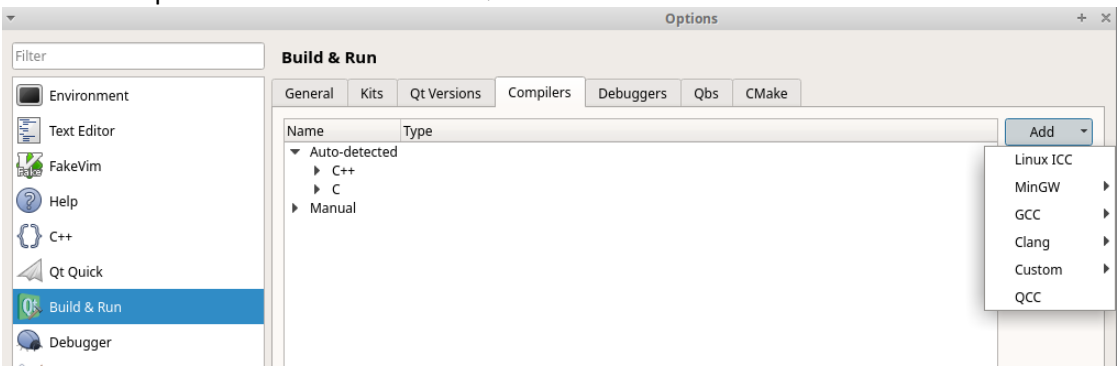


4. Launch QT creator
5. Navigate to Tools -> Options
6. In the left hand pane select “Devices” and then select the Devices tab.
7. Click “Add...” button, select “Generic Linux Device” and click “Start Wizard”.
8. Set the device name to “DB1”
9. Set the host name or IP address to the IP address noted down in step 3.
10. Set the username to “root”
11. Set the authentication type to password.
12. Set the users password to “password” (without quotes)

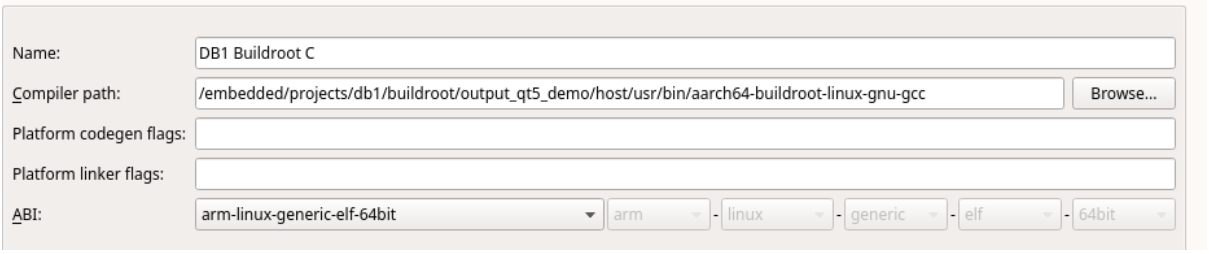


13. Click Next, click Finish and verify that the Device test was successful and click close.

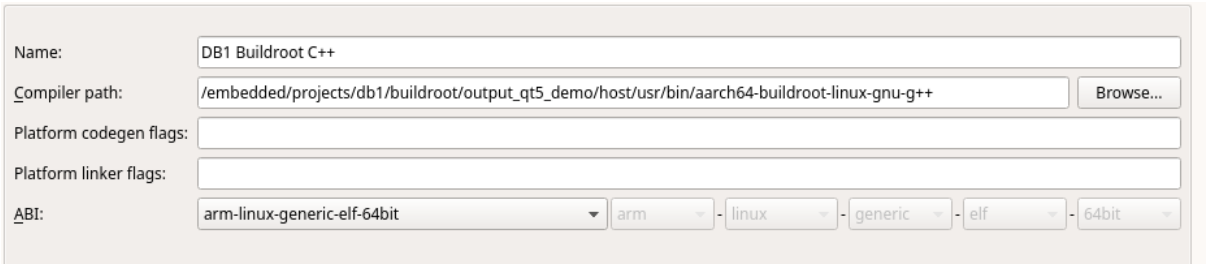
14. Press Apply in the Options Window.
15. In the left hand pane select “Build & Run”, and then select the “Compilers” tab.
16. Add C Compiler. Click “Add” -> “GCC” → “C”.



17. Set Name to “DB1 Buildroot C”.
18. Set Compiler path to "/embedded/projects/db1/buildroot/output_qt5_demo/host/usr/bin/aarch64-buildroot-linux-gnu-gcc”
19. Set ABI to “arm-linux-generic-elf-64bit”
20. Click Apply

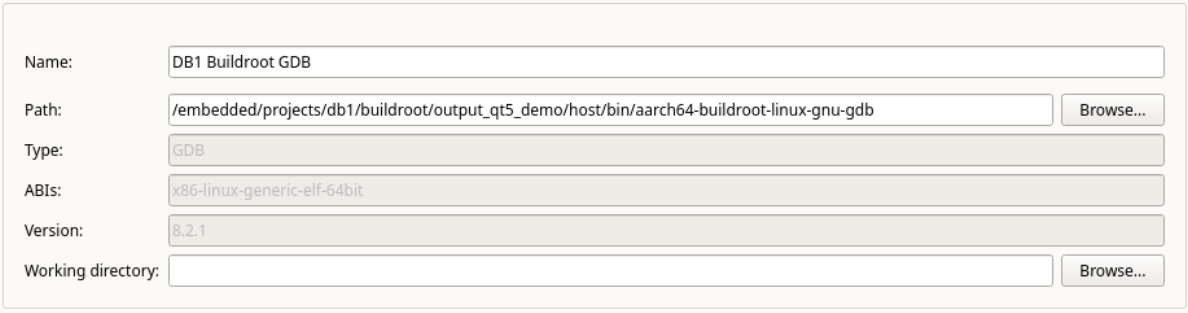


21. Add C++ compiler. Click “Add” → “GCC” → “C++”
22. Set Name to “DB1 Buildroot C++”
23. Set Compiler path to “/embedded/projects/db1/buildroot/output_qt5demo/host/usr/bin/aarch64-buildroot-linux-gnu-g++”
24. Set ABI to “arm-linux-generic-elf-64bit”
25. Click Apply

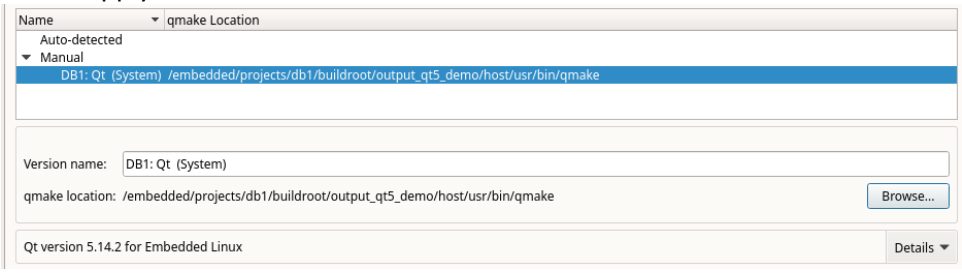


26. In the left hand pane select, “Build & Run” and then select the “Debuggers” tab. Click “add”.

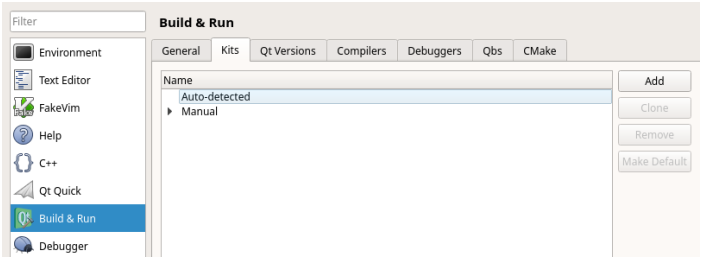
27. Set the name to, “DB1 Buildroot GDB”
28. Set the path to, "/embedded/projects/db1/buildroot/output_qt5_demo/host/bin/aarch64-buildroot-linux-gnu-gdb”
29. Click Apply



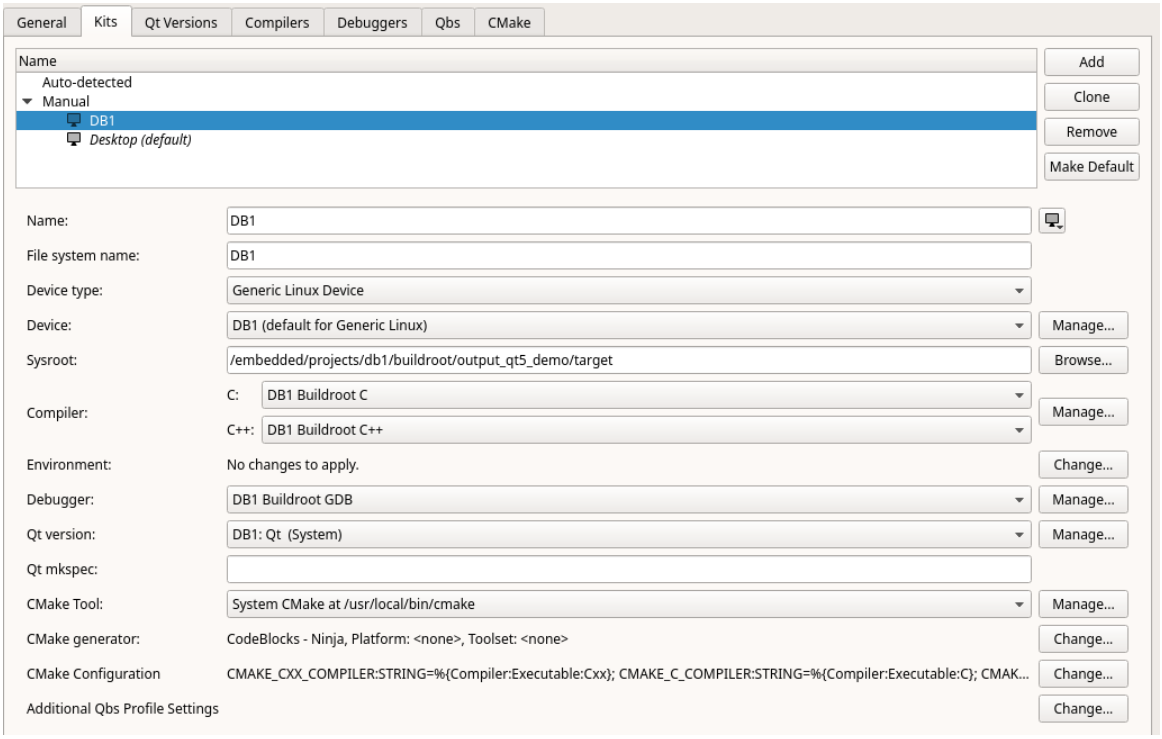
30. In the left hand pane select, “Build & Run” and then select the “QT Versions” tab. Click “Add”.
31. Select the qmake executable, /embedded/projects/db1/buildroot/output_qt5_demo/host/usr/bin/qmake”
32. Set the Version name to “DB1: Qt (System)”
33. Click Apply



34. In the left hand pane select, “Build & Run” and then select the “Kits” tab. Click “Add”.



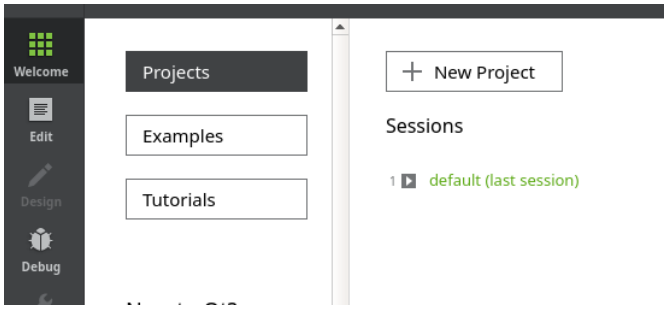
35. Set Name to “DB1”
36. Set File system name to “DB1”
37. Select Device Type to “Generic Linux Device”
38. Select Device to “DB1 (default for Generic Linux)”
39. Set Sysroot to “/embedded/projects/db1/buildroot/output_qt5_demo/target”
40. Set Compiler to “DB1 Buildroot C” and “DB1 Buildroot C++”
41. Set debugger to “DB1 Buildroot GDB”
42. Set Qt Version to “DB1: Qt (System)”
43. Press OK.



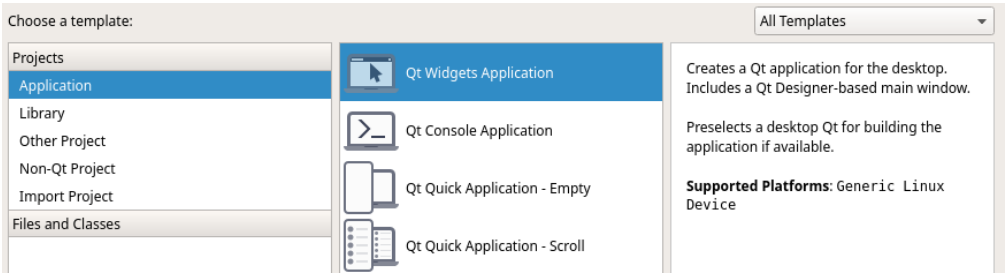
8.2.3 Setup a simple QT5 “Hello World” application

The following section will describe how to setup and deploy a simple “Hello world” application to TM1.

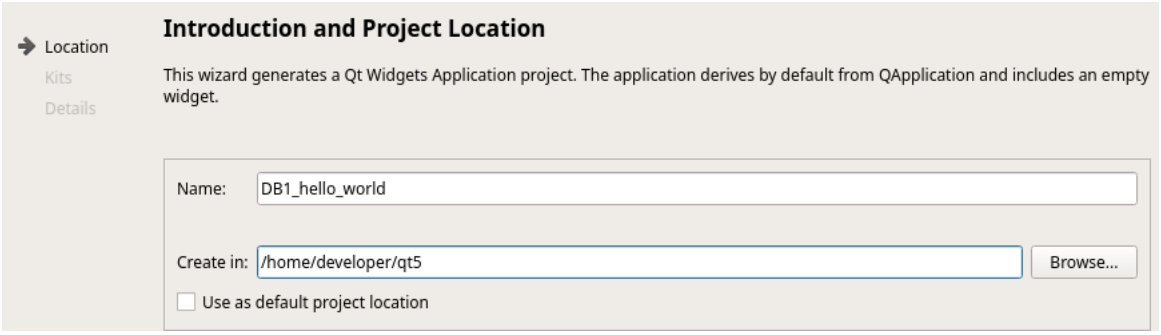
1. Launch QT Creator
2. Select “New Project”



3. Select, “Qt Widgets Application” and click, “Choose”.

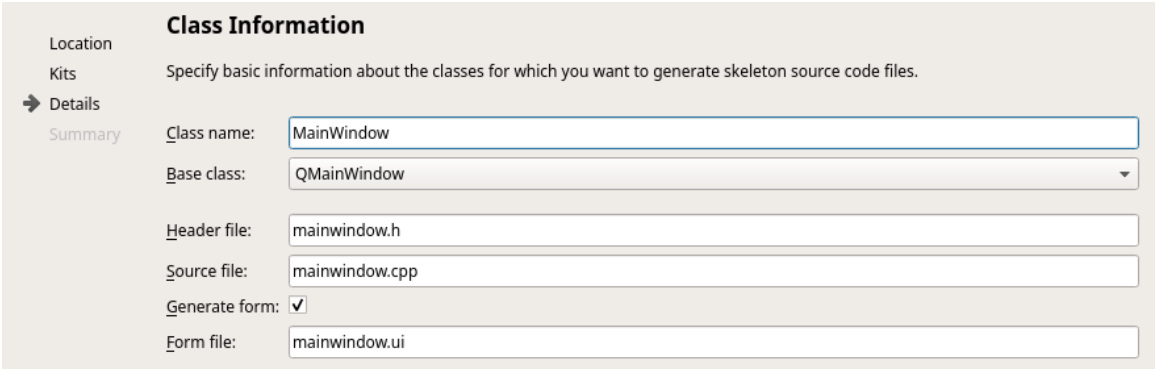


4. Set the name to “DB1_hello_world” and click next



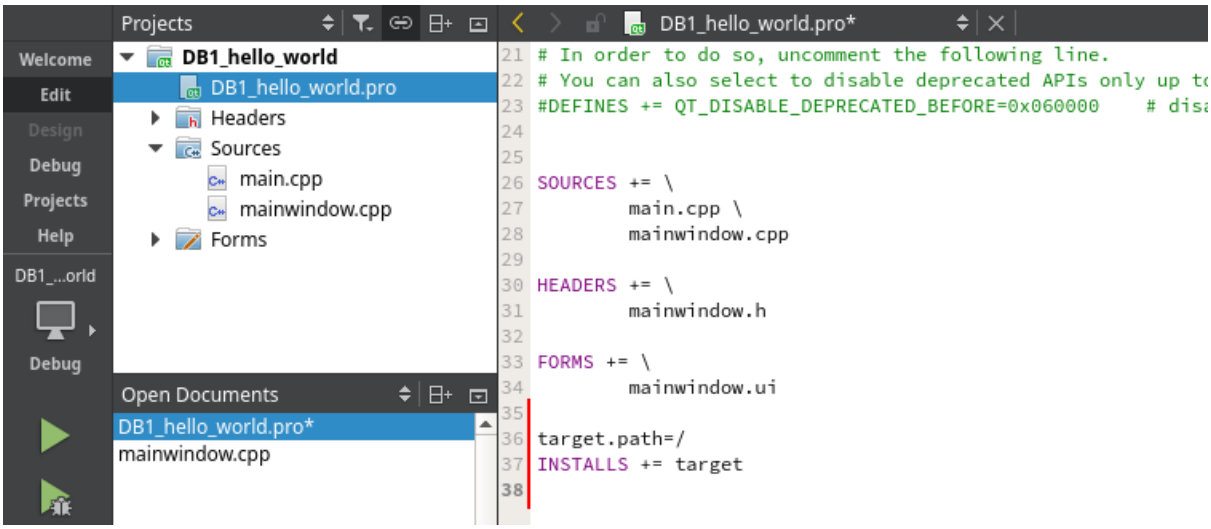
5. Select the “DB1” kit, and click next.

6. Use the default Class Information and click Next

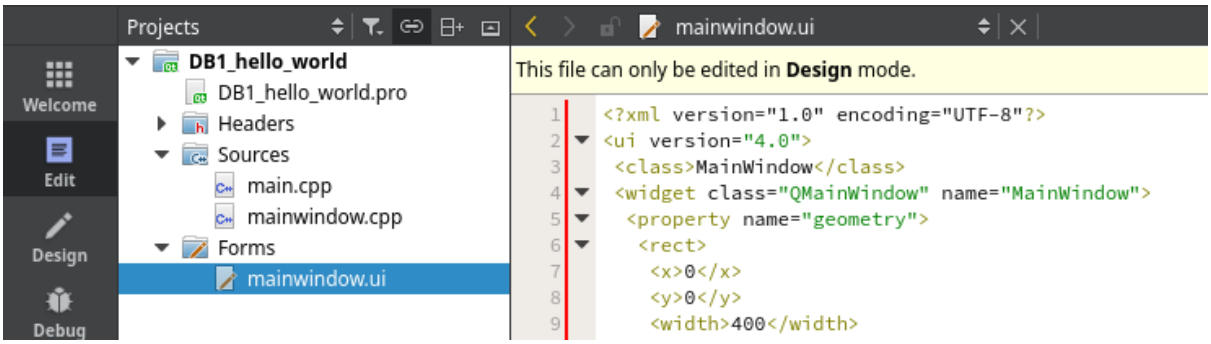


7. Click Finish
8. In the Projects view, double click “DB1_hello_world.pro” file to open the project editor.
9. Append the following to the bottom of the project configuration. This will tell the deployment tool where on the target root file system to put the executable.

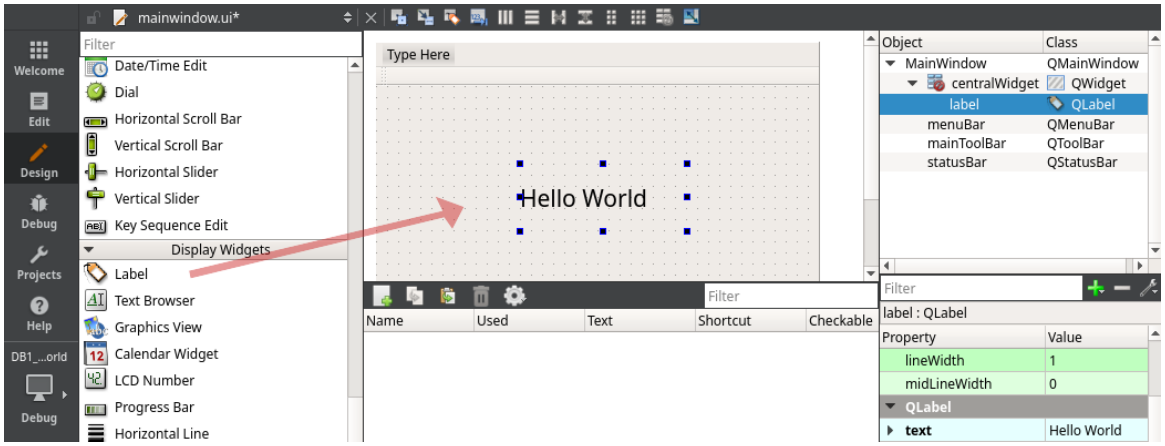
```
target.path=/
INSTALLS += target
```



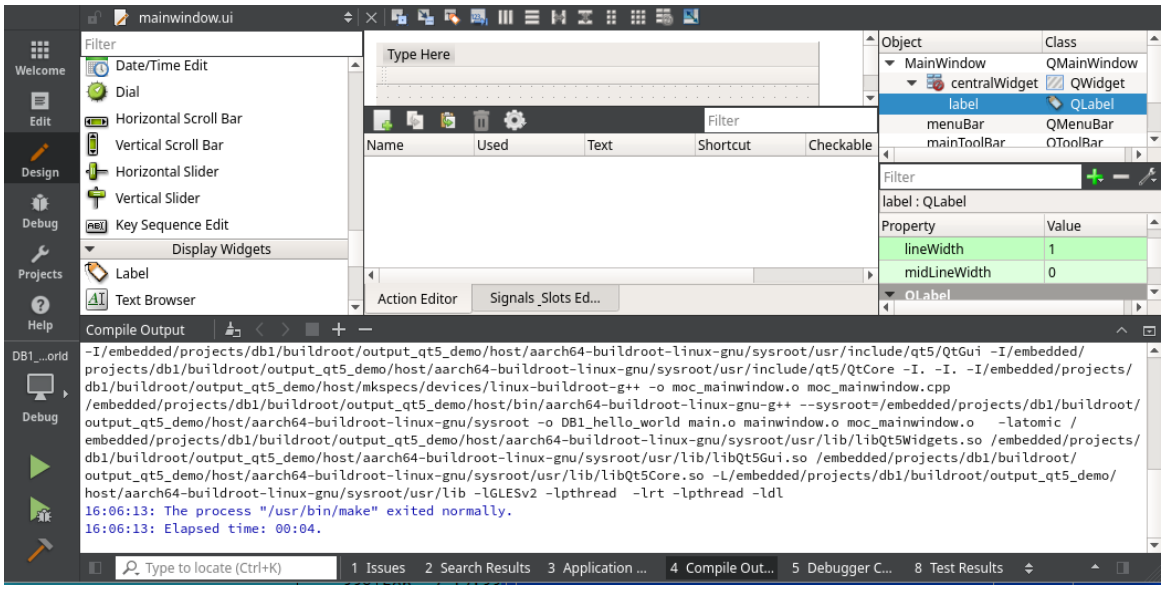
10. In the Projects view, double click “mainwindow.ui” to open the forms designer.



11. Scroll down to Display Widgets, and drag a label widget onto the form.
12. Use the property editor to change the label text to, “Hello World”



13. Select Build -> Build All (ctrl + shift +B), and click, “Save All” changes when prompted.



14. Monitor the “4 Compile Output” window for build completion without errors
15. Select Build → Run (Ctrl + R) to deploy and run the application on the DB1 hardware.

Appendix A - Known Problems

1) When GPU hardware acceleration is used then GTK widgets render texts with artefacts. Known solutions are:

- Disable 'Font Subpixel rendering'. In Lubuntu OS this can be set in Settings→Customise Look and Feel → Font → Sub-pixel geometry: None
- Or alternatively the GPU acceleration can be disabled by removing the /lib/firmware/a300*.fw files from the root-fs.

2) In Lubuntu OS the network widget opened from the bottom panel does not integrate well with touch screens. When the network widget is closed certain icons on LXDE panel no longer respond to touch events. The way how to exit such state is to drag and move any opened window by its title bar.

Appendix B - Change Log

Issue	Date	Author	Changes
1.0	02/02/2021	M Olejnik	Initial draft, based on BCT TM1 Linux document
1.0.1	05/02/2021	M Olejnik	Draft. Updated GPIO pull downs (for REV2 board), added Firmware blob section, Added GPS section. Added font rendering issue to Appendix A.
1.0.2	08/12/2021	M Olejnik	Draft. Updated GPIO pull ups (for REV3 board), updated RS485/422 information. Added Linux kernel git repo set-up. Added Lubuntu panel network issue to Appendix A.
1.0.3	12/01/2022	M Olejnik	Updated 'apt install' packages. Updated info about lubuntu root-fs download. Typos and clarity.