Linux

For

BCT TM3

User Guide

Document Reference: BCTTM3 Linux User Guide

Document Issue: 1.2.0

Associated SDK release: 1.1

Authors: D Robinson, M Olejnik

Contents

1. Introduction

The content of this document provides information required to start building Linux operating systems for the BCT TM3 platform. It covers:

- The tools and components required for building a Linux operating system
- How to install the build components
- How to compile the U-Boot boot loaders stand alone
- How to compile the Linux Kernel 4.9.118 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 TM3 platform
- How to setup and deploy a simple QT5 application to TM3

The BCT TM3 platform consists of TM3 module and several host boards. This document is applicable for HB8 and HB9 host boards including their Lite versions. The HB8 and HB9 host boards are functionally identical; the Lite versions lack the following connectors: HDMI, USB3, M.2. Unless explicitly mentioned throughout the document, when referring to HB8 it also means it is applicable to HB9 host boards.

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.

U-boot 2014.07 was ported to provide the bootloader functionality for the TM3.

Linux kernel 4.9.118 have been ported to be compatible with the BCT TM3 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 TM3 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 TM3 software components above have all been tested to compile using an Ubuntu 18.04 LTS and Ubuntu 22.04 LTS development machines.

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 TM3 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/tm3/

Download the Linux source code for TM3 using the command:

```
cd ~
wget http://dl.bluechiptechnology.com/dl/tm3/software/tm3linuxv110.tar.bz2
wget http://dl.bluechiptechnology.com/dl/tm3/software/tm3linuxv110.tar.bz2.md5
```

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

```
md5sum -c tm3linuxv110.tar.bz2.md5
```

Extract the tar ball by issuing the command:

```
cd /
tar xvjf ~/tm3linuxv110.tar.bz2
```

Setup git to pull the latest code from Blue Chip Technology (requires Internet connection).

```
sudo apt-get install git
```

Kernel 4.9

```
cd /embedded/projects/tm3/lichee/linux-4.9
git status
git pull
```

Bootloader

```
cd /embedded/projects/tm3/lichee/bootloader
git pull
cd /embedded/projects/tm3/lichee/brandy
git pull
```

Buildroot

```
cd /embedded/projects/tm3/buildroot
git pull
```

Packlinux tool

```
cd /embedded/projects/tm3/packlinux
git pull
```

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 4.9.118. The first directory ("embedded") is the folder created in the root of the file system.

TM3 embedded development directory: **/embedded/projects/tm3/**

Directory	Description
lichee/bootloader	Source code for SPL bootloader including configuration for BCT TM3
lichee/brandy	Source code for U-boot bootloader including configuration for BCT TM3
lichee/linux-4.9	4.9.118 Linux kernel source code with configuration for BCT TM3
packlinux	A tool for creating Linux installation images for BCT TM3.
buildroot	Buildroot top directory, containing buildroot 2022.02 source release with configurations for building root file system and installation images for TM3.
rootfsimages	Directory containing prebuilt root file systems, including: Demo Ubuntu 22.04 image distributed with TM3.

2.2.1 Git repo setup for the source trees

In order to keep the source repositories up-to-date with Blue Chip's patches a git remote repositories are set-up. Future source code modifications of the bootloader , kernel and buildroot can be downloaded and applied by issuing `git pull` command. When doing so, ensure the command is issued in the right directory while switched to the master branch. For example, to update Linux kernel source code issue the following commands:

```
cd /embedded/projects/tm3/lichee/linux-4.9
git checkout master
git pull
```

Please note that the Blue Chip's public git repositories are read only and will not let you to push your modifications and branches.

If you plan to do custom modification of the source repositories and want to keep track of your changes you may want to add your own git remote servers. To do that, issue the following command in the target source tree (kernel, bootloader, buildroot etc.):

```
git remote add origin_new <url_to_git_server_or_path_to_bare_git_repository>
```

You can then push new branches to your git repository by the following command:

```
git push origin_new <branch_name>
```

2.3 Development Machine Setup

Where possible, build scripts have been provided for the various components included with the Linux SDK for BCT TM3. These scripts presume the following has been setup on the Ubuntu 18.04 LTS development machine.

- A TFTP server serving files from a tftpboot directory in the root of the filesystem. (/tftpboot)
- An NFS server serving files from /nfs.

The following links provide information on setting up an NFS and TFTP server.

http://www.debianhelp.co.uk/tftp.htm

Setup Required

```
sudo apt-get install tftpd
sudo mkdir /tftpboot
sudo chmod 777 /tftpboot
```

http://www.debianhelp.co.uk/nfs.htm

Setup Required

```
sudo apt-get install nfs-kernel-server nfs-common portmap
cd /
sudo mkdir nfs
```

In the file /etc/exports , add the line

*/nfs/rootfs *(rw,sync,no_root_squash)*

To use the nfs feature, the /nfs directory should be symbolically linked to the root file system directory. Either provide your own rootfs or use the supplied example rootfs stored in rottfsimages directory (see above table). From the /nfs directory issue the following command.

```
sudo ln -s /embedded/projects/tm3/rootfs /nfs/rootfs
```

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
sudo apt-get install mkbootimage
sudo apt-get install android-tools-fsutils

sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386
sudo apt-get install zlib1g:i386
```

If you build on **Ubuntu 22.04** or more recent Ubuntu OS, install the packages above (listed for Ubuntu 18.04) and then issue the following commands. They install extra tools and ensure the gcc version 9 is set as the default compiler:

```
sudo apt install g++-9
sudo apt install gcc-9
sudo ln -sf /usr/bin/g++-9 /usr/bin/g++
sudo ln -sf /usr/bin/gcc-9 /usr/bin/gcc
sudo apt install android-sdk-libsparse-utils
```

Before compiling various standalone Linux components, we must configure our target hardware platform for the 'lichee' build tree. A script is provided to do that, issue the following commands:

```
cd /embedded/projects/tm3/lichee
./build.sh config
```

A series of options will be displayed on the terminal. Select the following options:

- Platform: 0 (android)
- Chip: 19 (tm3)
- Kernel version: 0 (linux-4.9)
- Board: 0 (hb5)

3. Building required components for Ubuntu OS

3.1 Installing the Ubuntu OS root file system

There are many Linux distributions available that are compatible with BCT TM3. Ubuntu was chosen as the main distribution for TM3, 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 TM3. 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 TM3 consists of the following steps.

- Extract an Ubuntu image into the staging directory
- Add kernel modules and other specific support to the root filesystem.
- Boot the generated Ubuntu root filesystem on a BCT TM3
- Configure the Ubuntu root filesystem using, apt-get, synaptic package manager, or another package manager

To support the BCT TM3 hardware and kernel, various files need to be copied to the root filesystem. To simplify this process all build components that need to modify the root filesystem are configured to do so at the nfs staging location, "/nfs/rootfs". We must extract a root filesystem as a starting point to this location.

For convenience a pre-built root filesystem is included in the TM3 download. It can be extracted using the following commands.

```
cd /embedded/projects/tm3/
./extractubunturootfsimage.sh
```

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 Linux 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/tm3/lichee/linux-4.9/
source ./setenv-arch64.sh
make tm3linux_defconfig
./build.sh
```

When the compilation process has completed it will leave a Linux kernel (Image) at, "./arch/arm64/boot/Image", and, "/tftpboot/Image".

The TM3 kernel implements the device tree model for configuring a hardware platform. The TM3 kernel includes several configurations for the various versions of HB8 and HB9 host boards.

Kernel file definition	**Description**
tm3-hb8-43-c.dtb	HB8 with 4.3 Inch LCD and capacitive touch screen
tm3-hb8-43-r.dtb	HB8 with 4.3 Inch LCD and resistive touch screen
tm3-hb8-7-c.dtb	HB8 with 7 Inch LCD and capacitive touch screen
tm3-hb8-7-r.dtb	HB8 with 7 Inch LCD and resistive touch screen
tm3-hb8-9-c.dtb	HB8 and HB9 with 9.7 Inch LCD and capacitive touch screen

tm3-hb8-12-c.dtb	HB8 and HB9 with 12 Inch LCD and capacitive touch screen

build.sh is an example of a script file that simplifies the process of building BCT TM3 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 you change the configuration your new configuration can be saved with

```
make savedefconfig
cp defconfig arch/arm64/configs/tm3linux_defconfig
rm defconfig
```

Once the kernel has been compiled, the kernel modules must be copied to the root filesystem. Issuing the following command performs this task.

```
sudo ./installmodulesrootfs.sh
```

The modules are installed to the rootfs directory where the root filesystem that was previously extracted to in section 3.1.

3.3 Using customised Linux kernel device tree

When new peripherals are used with TM3 HB8 host board or if TM3 module is hosted on a custom board then then extra configuration may be needed. Two common actions are usually required to support a peripheral in Linux kernel:

1. An existing Linux driver has to be added to the kernel configuration (see paragraph related to menuconfig build option in section 3.2 for more information).
2. For peripherals not connected over USB bus (typically SPI or I2C bus) a change in device tree file is often required.

For simple customisation cases the relevant device tree file (name starting with tm3-hb8* and ending with *.dts) located in **`arch/arm64/boot/dts/sunxi/`** directory can be modified.

For complex solutions with customised host board a new device tree file might be preferred. To add a new device tree for customised TM3 board (for example the file name is tm3-custom.dts) do the following:

- Copy over the tm3-custom.dts device tree file (either hand crafted or provided by Blue Chip Technology) to **`arch/arm64/boot/dts/sunxi/`** directory in the Linux kernel source tree.
- Modify **`arch/arm64/boot/dts/sunxi/Makefile`**, locate the line starting with **`dtb-$(CONFIG_ARCH_SUN50IW6P1)`** and add the text "tm3-custom.dtb \" to the list of device trees. For example like that:

```
...
tm3-hb8-12-c.dtb \
tm3-custom.dtb \
tm3-hb5-7-c.dtb \
...
```

Note that the device tree source files have .dts extension, and the same files compiled into binary form have .dtb extension.

When you rebuild the kernel (see chapter 3.2 for more information) your new device tree binary file (tm3-custom.dtb) is produced in **`arch/arm64/boot/dts/sunxi/`** directory, providing there was no syntax error in the device tree source file. Use that device tree binary file to produce Linux installation image via the 'packlinux' tool (see chapter 5.1 for more information). Copy the tm3-custom.dtb into **`packlinux/linuxfiles`** directory and then pass parameter '-d tm3-custom.dtb' to the 'pack' tool to create your installation image with that device tree built in. If you use buildroot build system to produce the Linux installation image then you can pass the custom device tree name to the list of boards as described in paragraph related to '-menuconfig' buildroot option in section 4.2.

3.4 U-Boot Bootloader – Ported (http://www.denx.de/wiki/U-Boot)

U-Boot 2014.07 has been ported to work with TM3. Its purpose is to initialise the hardware, and boot a Linux operating system.

To build U-Boot for BCT TM3 issue the following commands.

```
cd /embedded/projects/tm3/lichee/brandy/
source ./setenv-awtools.sh
export -n ARCH
cd u-boot-2014.07/
make distclean
make tm3linux_config
./buildtm3.sh
```

The compiled boot loader file "uboot.bin" is copied to the /tftpboot directory.

3.5 Ubuntu OS 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 operating system for BCT TM3. The built components are as follows:

Component	Location
Ubuntu Root file system	/embedded/projects/tm3/rootfs
Linux Kernel	/embedded/projects/tm3/lichee/linux-4.9/arch/arm64/boot/Image
Device tree configurations	/embedded/projects/tm3/lichee/linux-4.9/arch/arm64/boot/dts/sunxi/tm3*.dtb
U-Boot	/tftpboot/uboot.bin

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 2022.02 is provided in the Linux download for TM3. It contains a sample configuration which builds the Linux kernel and a root file system. An installation image for TM3 is also produced.

4.2 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/tm3/buildroot
./build_qt5.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. All installation images take around 4 GBytes. You can reduce the number of produced installation images by removing certain boards from the build list. To do that run the build script with '-menuconfig' parameter:

```
./build_qt5.sh -menuconfig
```

A buildroot configuration screen will be presented. Navigate to 'System configuration -->' menu item and scroll down to the very last item that lists various TM3 boards. Trim the list to a single board to reduce the build's disk space requirements. If you use a custom board device tree binary file (see section 3.3), put its name to the list to produce a customised buildroot installation image. Save and exit the configuration screen by pressing Tabulator, Arrows and Enter keys. The build process will continue.

If you'd like to configure the Linux kernel during the buildroot build, then start the buildroot build script with '-kernel-menuconfig' parameter:

```
./build_qt5.sh -kernel-menuconfig
```

The kernel menu configuration will be displayed at the end of the build when the kernel build stage is initiated. The recommended procedure is to produce the default kernel build first (without using the -kernel-menuconfig parameter) and then - once the whole build is finished - to run the build script again, this time passing the -kernel-menuconfig parameter. This ensures that the whole build is not stuck waiting for the user input. Also, the second build will be much quicker, and the kernel menu configuration is presented sooner.

Note that you can run both buildroot menu configuration and kernel menu configuration during the single build like this:

```
./build_qt5.sh -menuconfig -kernel-menuconfig
```

If you've reconfigured buildroot to a configuration that no longer builds (conflicting selection of packages or other errors), you can reset to a clean state by deleting the whole ‘output_qt5’ subdirectory including its contents. You may want to back-up the existing configuration file ‘output_qt5/.config’ to preserve your previous customisations.

4.3 Buildroot outputs

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

QT5

Component	**Location**
Root file system	/embedded/projects/tm3/buildroot/output_qt5/images/rootfs.tar
Linux Kernel	/embedded/projects/tm3/buildroot/output_qt5/kernel/Image
Linux Kernel modules	/embedded/projects/tm3/buildroot/output_qt5/kernel/lib/modules
Device Tree Configurations	/embedded/projects/tm3/buildroot/output_qt5/kernel/*.dtb
Installation images	/embedded/projects/tm3/buildroot/output_qt5/images/tm3_*.img

5. Updating the firmware / software on TM3

WARNING: the installation fully overwrites all contents of the TM3's internal storage. Make sure you've backed all important files up.

WARNING: the installation fully overwrites all contents of the uSD card image. Make sure you've backed all important files up

Once the QT demo is fully built the installation images are produced in the **/embedded/projects/tm3/buildroot/output_qt5/images** directory.

The name of the image is **tm3_linux_<*>.img** where the <*> stands for host board, screen size, touch screen type. For example this file:

tm3_linux_hb8-7-c.dtb.img

is produced for TM3 HB8 host board with a 7" screen and a capacitive touch screen.

5.1 Packlinux tool for producing installation images

A tool for creating custom installation images with Linux OS is available in **/embedded/projects/tm3/packlinux** directory. This tool is used to produce Buildroot QT5 installation images, but can be also used for your customised OS releases.

Note: you can skip this step if you are building Buildroot OS. The build script of the Buildroot does this step automatically as a part of the build process.

To produce your **custom installation image**, copy the following components to the target location as follows.

Component	**Location**
Root file system	/embedded/projects/tm3/packlinux/linuxfiles/rootfs.ext4
Linux Kernel	/embedded/projects/tm3/packlinux/linuxfiles/Image
Device Tree Configurations	/embedded/projects/tm3/packlinux/linuxfiles/*.dtb
U-boot (optional)	/embedded/projects/tm3/packlinux/chips/tm3/bin/u-boot-tm3.bin

Note that the root file system has to be provided as a partition image, in this case ext4 partition.

To create an ex4 partition image from a .tar.gz file issue the following command:

```
sudo virt-make-fs --size=+200M  --type=ext4 rootfs.tar.gz rootfs.ext4
```

virt-make-fs tool is part of the 'libguestfs-tools' package provided by Ubuntu.

The rootfs.ext4 file can be optimised for size by a **tm3simg** tool found in /embedded/projects/tm3/packlinux/tools directory. To optimise the rootfs partition file use the following command:

```
cd /embedded/projects/tm3/packlinux/
sudo tools/tm3simg linuxfiles/rootfs.ext4 linuxfiles/srootfs.ext4
```

To produce a TM3 installation image issue the following commands.

```
cd /embedded/projects/tm3/packlinux/
./pack -b hb5 -d tm3-hb8-9-c.dtb -r srootfs.ext4
```

An installation image file "tm3_linux_hb8-9-c.dtb.img" will be produced in the current directory. Such installation image file can be used with the installation tools described in sections 5.2 and 5.3.

The 'pack' tool's command line arguments are as follows:

-b <board> : defines target board. Use "hb5". All HB8 and HB9 boards use "hb5" board configuration option when used with the 'pack' tool.

-d <dtb_file>: defines the Device tree file to use. The file must exist in packlinux/linuxfiles subdirectory.

-r <part_file> : defines the rootfs partition file (for example rootfs.ext4). The partition file can be either a raw or sparse image. Sparse images may take significantly less space depending on the partition size and its contents. To convert raw image to a sparse image use **tm3simg** command (see above, do not use img2simg which is not compatible). The partition file (rootfs.ext4) file must exist in 'packlinux/linuxfiles' subdirectory.

5.2 PhoenixCard tool

PhoenixCard tool populates a uSD card with an OS installer. Such uSD card – when booted on TM3 HB5 board – installs the full operating system on TM3 internal storage.

To use the tool, transfer the appropriate OS image file (created by Buildroot or by the Packlinux tool) to your Windows PC, along with the installation archive phoenixcard4.1.3.zip downloaded from:

https://downloads.bluechiptechnology.com/tm3/software/tools/phoenixcard4.1.3.zip

Unzip the phoenixcard4.1.3.zip and run the PhoenixCard.exe file. Dismiss the "updatever" dialog presented during start.

The steps to produce the installation uSD card and to install the OS image to TM3 internal storage are as follows:

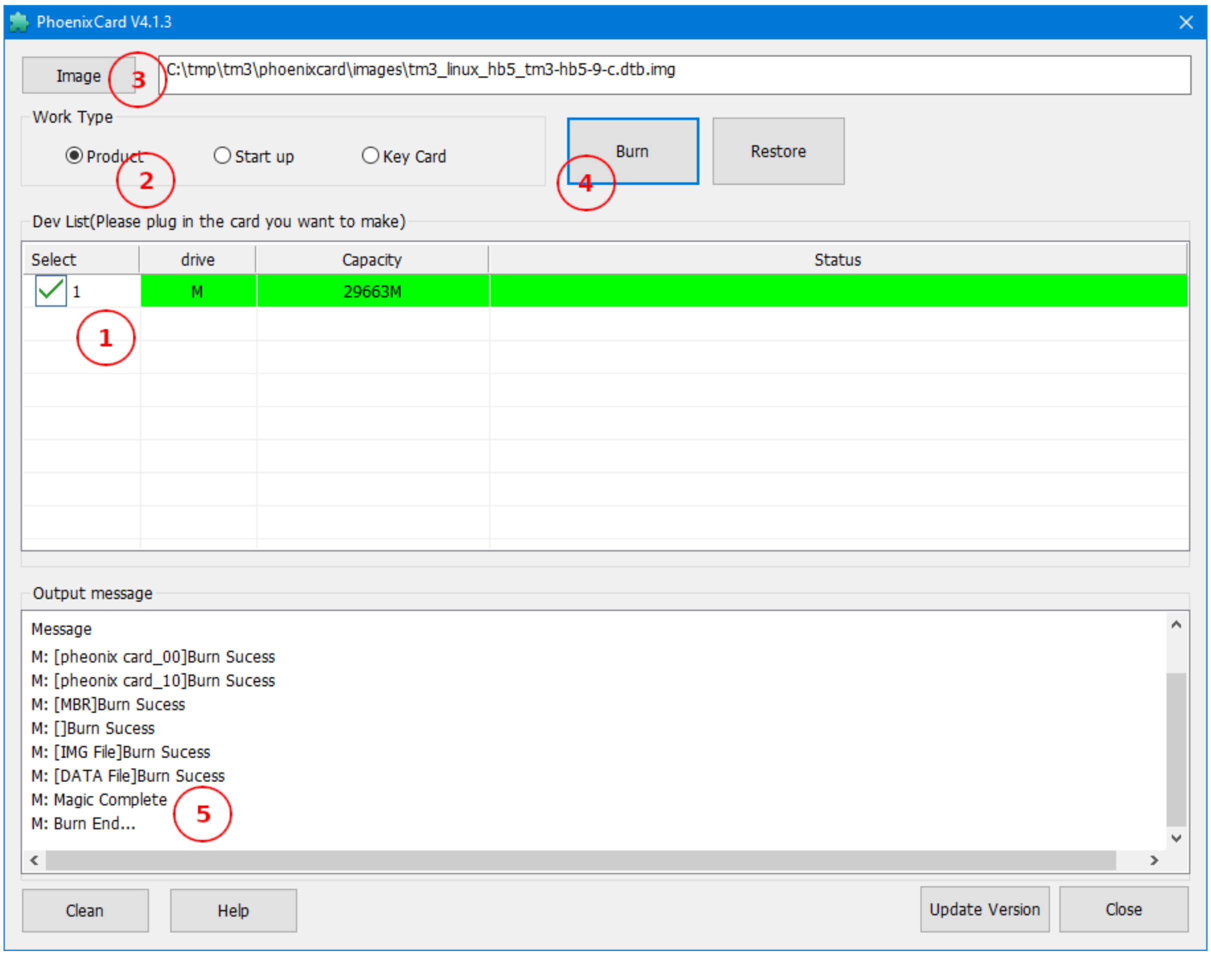
1) Insert a uSD card into your PC (or use USB card reader). The uSD card drive should be displayed in the list in the middle of the tool's window. Ensure the drive item's checkbox is checked.

2) Ensure the Work Type is set to Product.

3) Select the installation image. Use the file produced by the QT5 buildroot build or Packlinux tool.

4) Click the Burn button, the image writing process should start. A progress bar should be displayed next to the uSD drive item in the Status column.

5) After the writing process is finished the uSD drive item turns green and the log window shows "**M: Burn End ...**" message. Close the Phoenixcard tool, eject the uSD card.



PhoenixCard V4.1.3
Image
3
C:\tmp\tm3\phoenixcard\images\tm3_linux_hb5_tm3-hb5-9-c.dtb.img
Work Type
Product
2
Start up
Key Card
Burn
4
Restore
Dev List(Please plug in the card you want to make)
1
Clean
Help
Update Version
Close
5

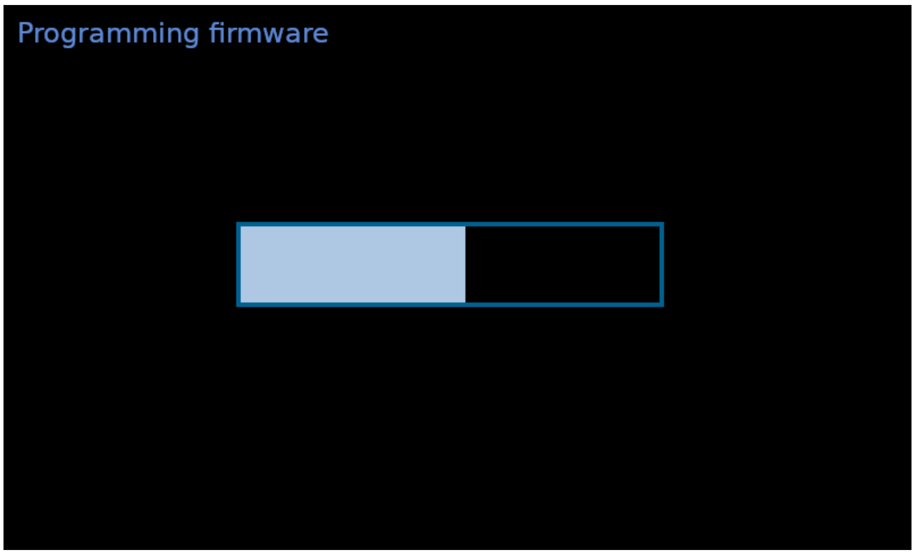
Select	drive	Capacity	Status
✓ 1	M	29663M	

Output message

```
Message
M: [pheonix card_00]Burn Sucess
M: [pheonix card_10]Burn Sucess
M: [MBR]Burn Sucess
M: []Burn Sucess
M: [IMG File]Burn Sucess
M: [DATA File]Burn Sucess
M: Magic Complete
M: Burn End...
```

6) Ensure the TM3 HB8 board is turned off. Plug in the uSD card prepared by the Phoenixcard tool to TM3 HB8 host board uSD card slot.

7) Power on the TM3 HB8 board. An installation progress bar should be displayed.



8) Wait for the installation to finish. A message “Complete. Remove card and reboot.” will be displayed on the screen.

9) Power the TM3 board off and eject the uSD card from the HB8 board.

10) Power the TM3 board on. The board will boot the newly installed OS from the internal storage.

5.3 PhoenixUSB Pro tool

PhoenixUSB Pro is a Windows tool that can install an operating system on TM3 module via connected USB cable. The tool can handle installation of several connected TM3 devices at the same time and requires only minimal user interaction.

To use the tool, transfer the appropriate OS image file (created by Buildroot or by the Packlinux tool) to your Windows PC, along with the installation archive PhoenixUSBPro_v4.0.0.msi downloaded from:

https://downloads.bluechiptechnology.com/tm3/software/tools/PhoenixUSBPro_v4.0.0.msi

Install the msi archive and run the PhoenixUSBPro.exe file. The steps to install an operating system to TM3 HB5 are as follows:

1) Click the Image button and select the installation image file produced by the QT5 buildroot build or Packlinux tool.

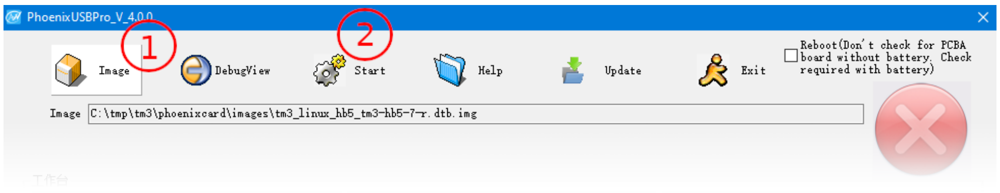
2) Click Start button. The big circular indicator on the top right of the application window should turn green. At this point TM3 devices can be installed.

3) Power the TM3/HB8 board off. Connect the USB cable between the PC and USB Device port on HB8 labelled as P11.

4) Press and hold the BOOT_MODE# button (the button terminals are exposed to 50 way connector, or to a Picoblade connector P1). Power the TM3/HB8 on while keeping the BOOT_MODE# button pressed.

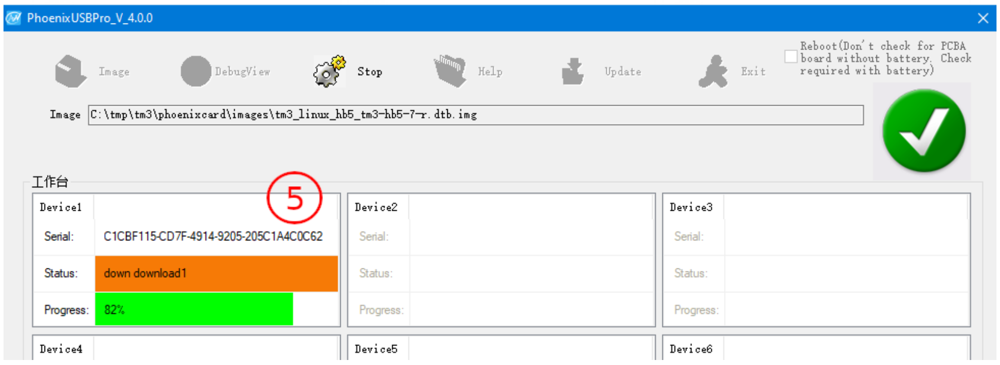
5) When you hear the Windows jingle notifying that a new USB device was plugged-in, you can release the BOOT_MODE# button. The application will discover the connected TM3 board (it may take up to 40 seconds) and start the OS installation.

6) When the installation progress bar gets full the installation is complete. Power the TM3/HB8 off and then unplug the USB cable. You can install another TM3/HB8 device by going back to step 3)



Reboot(Don't check for PCBA board without battery. Check required with battery)

Image C:\tmp\tm3\phoenixcard\images\tm3_linux_hb5_tm3-hb5-7-r.dtb.img



Reboot(Don't check for PCBA board without battery. Check required with battery)

Image C:\tmp\tm3\phoenixcard\images\tm3_linux_hb5_tm3-hb5-7-r.dtb.img

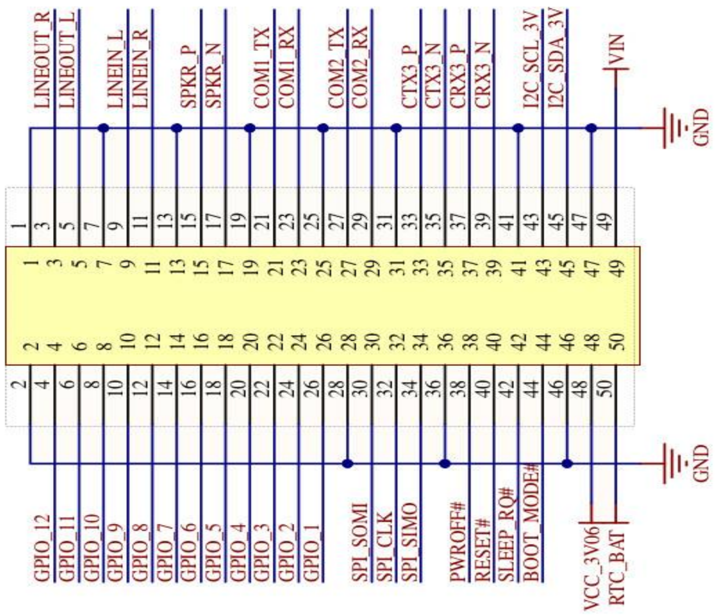
Device1		Device2		Device3	
Serial:	C1CBF115-CD7F-4914-9205-205C1A4C0C62	Serial:		Serial:	
Status:	down download1	Status:		Status:	
Progress:	82%	Progress:		Progress:	
Device4		Device5		Device6	

Note that when the installation mode has been started (step 2) the application prevents accidental closure of its window. To close the application, click the Stop icon first, and then close the window as usual.

6. BCT TM3 Hardware Setup in Linux

6.1 Host board 50way connector

The pin-out of the host board 50way connector is as follows:



6.2 Debug Serial Console

Linux and U-boot for BCT TM3 heavily relies on access to a serial console. By default, U-boot and Linux are configured to use the RS232 port available on pins 21 and 23 of the 50w connector. By default, the board is set to communicate at 115200, 8, n, 1. Before turning the TM3 on for the first time it is recommended that this port is connected to a PC with terminal emulator software running. E.g. HyperTerminal.

Please note that it is **not TTL** logic level UART but RS232, therefore you may need RS232 to USB adapter to use the serial port with a modern PC that does not have a serial COM port connector.

6.3 BCT TM3/HB5 Serial Ports

The serial ports on the host boards are mapped as follows:

Host board 50w connector pins	**Linux Device Name**
COM2_TX, COM2_RX	/dev/ttyS1
CTX3_P, CTX3_N, CRX3_P, CRX3_N	/dev/ttyS3
COM1_TX, COM1_RX	/dev/ttyS0 (Linux console port)

6.3.1 RS-485 Manual Transmit Control

/dev/ttyS3 is an RS485 / RS422 compatible port which has a transmit enable signal. This signal can be controlled using GPIO 386. From the Linux console this signal can be manipulated using the commands:

```
echo 386 >> /sys/class/gpio/export
echo out >> /sys/class/gpio/gpio386/direction
echo 1 >> /sys/class/gpio/gpio386/value
echo 0 >> /sys/class/gpio/gpio386/value
```

6.3.2 RS-485 Automatic Transmit Control

To improve software efficiency when communicating over an RS485 interface it is possible to configure the TM3 Linux kernel to automatically control the transmit enable. 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 BCT application note RS485_BETA_APP_NOTE also provides useful information on implementing RS-485 with the TM3 platform.

https://downloads.bluechiptechnology.com/tm1/Documentation/RS-485_Application_Note_For_BCT_Beta.pdf

6.3.3 UART DMA and FIFO Threshold

The TM3 UART driver in the Linux kernel is designed to be efficient at high throughputs and baud rates. One technology that the driver uses is DMA (direct memory access) to provide efficient transfer of data. A second technique that the driver uses is setting a high FIFO threshold to limit the amount of interrupts requiring software servicing. While these driver optimisations give good performance and efficiency at high throughputs, this is not always the case for low baud rates and small amounts of data which tends to be the case with protocols using RS-485.

To allow the DMA function to be enabled a file called 'dmaenabled' has been added to the sysfs for UARTS. To enable UART DMA for the RS485 UART on TM3 the following command can be issued at a console or through application software.

```
echo 1 > /sys/class/tty/ttyS3/device/dmaenabled
```

Two values are valid: 0 to disable the DMA, 1 to enable it.

Note: when setting the DMA, it must be done before the application software opens the UART device file.

To allow the UART FIFO threshold to be configured a file called 'rxfifothreshold' has been added to the sysfs for UARTS. To modify the UART FIFO threshold to 1 for the RS485 UART on TM3 the following command can be issued at a console or through application software.

```
echo 1 > /sys/class/tty/ttyS3/device/rxfifothreshold
```

The rxfifothreshold can be set to any value between 1 and 256, however due to hardware limitations only 4 effective value ranges are applied. These are as follows:

- Value range 1 – 31: interrupt is raised when 1 character is in RX FIFO
- Value range 32 –95: interrupt is raised when 64 characters are in RX FIFO (¼ of the FIFO buffer size)
- Value range 96-191: interrupt is raised when 128 characters are in RX FIFO (1/2 of the FIFO buffer size)
- Value range 192-256: interrupt is raised when the RX FIFO is full.

Note: When setting the FIFO threshold, it must be done before the application software opens the UART device file.

6.4 BCT TM3 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 TM3 host boards are setup with pull-ups enabled. They are defined in the w50gpio_a_on and w50gpio_b_on structures of the tm3-hb5.dts file (/embedded/projects/tm3/lichee/linux-4.9/arch/arm64/boot/dts/sunxi/tm3-hb5.dts)

The logical GPIOs on the P15 header of HB5 map to the physical GPIO pins on the SOC as follows:

Logical GPIO on 50w connector	**HB8 and HB9 Physical GPIO**
GPIO 1	356
GPIO 2	357
GPIO 3	358
GPIO 4	359
GPIO 5	200
GPIO 6	201
GPIO 7	204
GPIO 8	205
GPIO 9	206
GPIO 10	166
GPIO 11	362
GPIO 12	227

To setup and control GPIO5 as Output with value 1 the following commands would be used:
```
sudo su
echo 200 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio200/direction
echo 1 > /sys/class/gpio/gpio200/value
```

6.5 TM3 Wi-Fi Operation

TM3 modules are produced with and without a WiFi chip.

6.5.1 TM3 Modules without WiFi

TM3 modules without a WiFi capability can use external USB WiFi dongles connected to USB slot on HB8. Please note that the kernel may need to be reconfigured to include specific driver for the WiFi chip used on the USB dongle. See section 3.2 for more information how to change kernel configuration.

6.5.2 TM3 Modules with WiFi

TM3 modules with a WiFi chip use 'ssv6x5x' WiFi driver which is by default built as a Linux kernel module when building the Buildroot QT5 OS image. The Linux kernel module may or may not be loaded during start of the OS depending on the choice of device manager being used. Buildroot QT5 OS and Ubuntu 22.04 OS load the WiFi module automatically.

To check the WiFi Linux kernel module is loaded issue the following command:

```
lsmod | grep ssv6x5x
```

If the module is not loaded by the device manager, then it can be loaded by issuing the following command (presuming the driver was compiled as a module and its module file exists in the root file system):

```
sudo modprobe ssv6x5x
```

The driver requires a configuration file located at the following location on the root file system:

```
/lib/firmware/ssv6x5x-wifi.cfg
```

The configuration file is provided for your reference at the following location (the two lines bellow are part of the same file path):

```
/embedded/projects/tm3/buildroot/buildroot-
2022.02/board/bct/tm3/rfs_overlay/usr/lib/firmware/ssv6x5x-wifi.cfg
```

If the module was 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
```

6.6 TM3 Audio

The audio CODEC featured on TM3 implements the standard Linux ALSA API framework. Standard commands like alsamixer, aplay, arecord, speaker-test will work.

The audio signals on the host board 50w connector can be controlled via ALSA Card 1 (SUNXI-AUDIO acx00-dai-0). Pins for headphone output are marked as LINEOUT_L and LINEOUT_R on the 50w connector. There is a Class D amplifier on the host boards which outputs mono audio signal to SPKR_P and SPKR_N pins when it is turned on. The CLASS D amplifier is by default turned off. To turn the CLASS D amplifier on, set the GPIO 385 as output and value 1. Line-in stereo audio input is present on LINEIN_L and LINEIN_R pins on the 50w connector.

The following example configures the audio mixer for output and plays a sound.

```
amixer -c 1  sset 'LINEOUT' 100%
amixer -c 1  sset 'Left DAC Mixer I2SDACL' on
amixer -c 1  sset 'Left Output Mixer DACL' on
amixer -c 1  sset 'Right DAC Mixer I2SDACR' on
amixer -c 1  sset 'Right Output Mixer DACR' on

aplay -D hw:1,0 /usr/share/sounds/alsa/Front_Left.wav
```

The HDMI audio signals on HB8 can be controlled via ALSA Card 0 (SUNXI-HDMIAUDIO audiohdmi-dai-0).

6.7 HB8 uSD Card

The uSD card connector featured on host boardsis mapped to **/dev/mmcblk0** device in the Linux kernel.

6.8 TM3 Watchdog

The SOC on the TM3 module includes a hardware 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

Please note that the maximum time between watchdog updates can be set up-to 16 seconds on TM3 module due to the hardware limitations.

6.9 TM3 Power management

TM3 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

TM3 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 host board 50w connector is configured to wake the system up when in suspend to RAM mode.

Please see Appendix A for an issue related to suspend to RAM feature.

6.10 TM3 Class-D amplifier

The class D amplifier implemented on host boards can be controlled using physical GPIO 385.

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/pwm-backlight/brightness
```

The following commands would set the backlight to 50%:

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

The following commands would set the backlight to 100%:

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

6.12 HB8 RTC

The HB5 host board has a Real Time Clock IC and a battery connector to keep the current time while the board is powered off.

To check the RTC is present in Linux issue the following command:

```
cat /sys/class/rtc/rtc0/name
```

rtc-pcf8523 should be displayed.

For proper RTC operation the RTC battery must be present. It is either soldered directly on the HB8 board or must be connected to the battery connector.

To read current time stored in RTC issue the following command:

```
hwclock -r -f /dev/rtc0
```

To set current time to RTC issue the following commands:

```
date -s "2022-03-31 15:37"
hwclock -w -f /dev/rtc0
```

Note: if the RTC battery is unplugged from the connector then the current date and time kept by the RTC chip is lost. In such case, you'll need to set the current time to RTC again - as described above.

6.13 HB8 and HB9 vs. Lite versions

The HB8 and HB9 Lite versions do not have certain connectors populated. Specifically, it is HDMI, USB3 and M.2 connectors. Lite versions of the host boards use the same Linux kernel device tree configurations as non-lite host boards, but the missing connectors prohibit use of those peripherals.

6.13.1 HB8 USB3

Linux kernel enables the USB3 bus during boot. Its presence can be verified by the following command:

```
lsusb –t
```

The following device should be displayed in the list:

```
Bus 04.Port 1: Dev 1, Class=root_hub, Driver=xhci-hcd/1p, 5000M
```

No further configuration is required. Devices plugged-in to the USB3 socket should work as expected.

6.13.2 HB8 HDMI out

HDMI screen is by default mapped to the second Linux framebuffer **/dev/fb1**. The primary screen mapped to **/dev/fb0** device is the LCD panel. Xorg (X window system display server) can be configured to display desktop on the HDMI screen by adding the following configuration to the /etc/X11/xorg.conf.d/50-displays.conf file. Create the file if it does not exist and then reboot the board to apply the changes.

```
Section "Device"
  Identifier "FBDEV 1"
  Driver "fbdev"
  Option "fbdev" "/dev/fb1"
EndSection
```

Currently used HDMI screen resolution can be checked by the following command:

```
cat /sys/class/graphics/fb1/modes
```

The capabilities of the connected HDMI screen can be checked by the following command:

```
edid-decode < /sys/devices/virtual/hdmi/hdmi/attr/edid
```

Check the ‘Established timings supported’ information in the list for supported resolutions of the connected HDMI screen.

By default the Full HD screen resolution (1920x1080@60Hz) is configured. The screen resolution can be changed in **/embedded/projects/tm3/lichee/linux-4.9/arch/arm64/boot/dts/sunxi/ tm3-tmxdevboard.dts** device tree file (or **tm3-hb8.dts** for HB8 host board). Change the value of the “screen1_output_mode” to set the new HDMI screen resolution as per the following table:

Resolution	**screen1_output_mode value**
720x480@60	2
720x576@50	3
1280x720@50	4
1280x720@60	5
1920x1080@50	9
1920x1080@60	0xa
3840x2160@30	0x1c
3840x2160@60	0x22

The definition of the video modes can be found in **/embedded/projects/tm3/lichee/linux-4.9/include/video/sunxi_display2.h** file. Search for ‘enum disp_tv_mode’ to get the full video mode list supported by the HB8 boards.

7. UBOOT operation

The u-boot version ported to the TM3 platform is V2014.07. At a high level its primary purpose is to copy the Linux kernel, device tree configuration, and bootargs into memory before passing execution over to the Linux kernel.

7.1 Configuring uboot

Configuration of uboot is performed by issuing commands over the debug serial console available on P2 of HB5. The UART is configured to communicate with 115200,8,n,1 parameters. Connecting a null modem cable between the HB5 and a development PC makes it possible to configure uboot using a terminal emulator. E.g. Putty or HyperTerminal.

To enter configuration mode, uboot must receive a character over the serial port during power on. The bootdelay parameter is set to 1 by default to give a fast boot time, which means that the time window pressing the key is short.

The four most common commands used in uboot for TM3 are.

1. setenv – used to set an environment variable to a value.
2. printenv – used to display the current value of an environment variable.
3. editenv – used to edit an environment variable.
4. saveenv – save the environment

The remainder of this section will focus on the TM3 specific environment variables and how they should be edited. Other commands are available, which can be viewed by issuing the command "`help`". The official uboot website is also a good source of information on uboot.

http://www.denx.de/wiki/U-Boot/

7.2 Uboot environment variables

The following table defines the Uboot variable related to the TM3 platform.

Variable	**Description**
bootdelay	The time window in seconds that Uboot will wait for a key press to enter configuration mode. Default value is 1
mmcargs	The bootargs passed to the Linux kernel when booting from uSD or eMMC storage.
netargs	The bootargs passed to the Linux kernel when booting from NFS storage.
mmc_root	The uSD/emmc partition to mount as the root filesystem.
fdt_file	The device tree blob file to load
serverip	The IP address of tftpserver, and NFS server. Used when booting over a network.
nfsroot	The nfs root directory to mount on the host PC when booting over NFS.

7.3 Uboot configuration examples

7.3.1 Changing the Uboot boot delay

```
setenv bootdelay 3
saveenv
```

7.3.2 Booting the Linux kernel over tftp and mounting a rootfs over NFS

```
setenv serverip <IP address of development machine>
setenv nfsroot /nfs/rootfs
setenv bootcmd run netboot
saveenv
```

7.3.3 Boot a root filesystem from the HB8 uSD

```
set mmc_root /dev/mmcblk0p2 rootwait rw
saveenv
```

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 TM3 platform over a network connection.

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
```

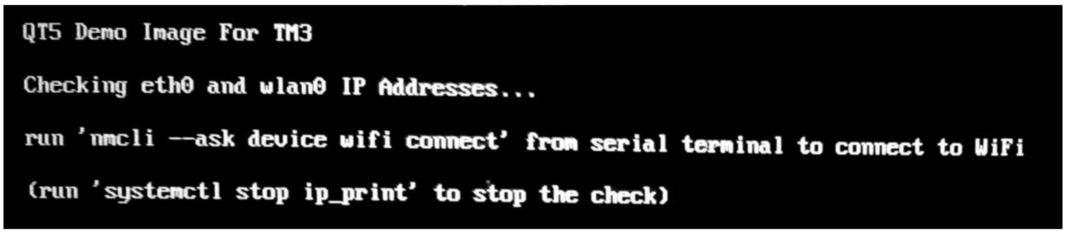
The recommended Linux OS (Ubuntu 22.04) installs QT Creator version 6.0.2 and is based on QT 5.15.3.

8.2.2 Setup the TM3 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 TM3 running the Buildroot generated QT5 root file system created in section 4.2.2.

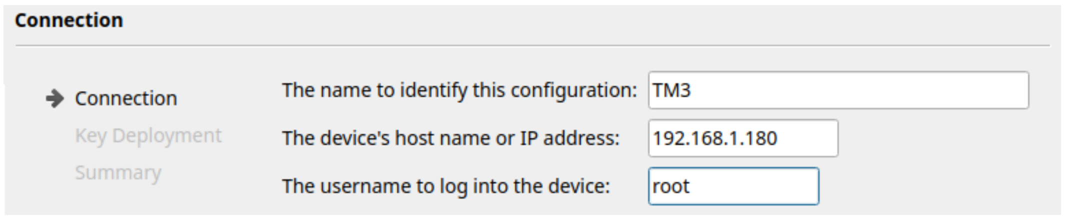
1. Ensure section 4.2.2 has been followed to create a QT5 based root file system for TM3.

2. Prepare an installation uSD card for TM3 as described in section 5.2 and install the QT5 demo image to TM3's internal storage.
3. Boot the TM3 unit with an Ethernet cable attached. The LCD will display the IP address obtained via DHCP. Make a note of this IP Address.



```
QT5 Demo Image For TM3

Checking eth0 and wlan0 IP Addresses...

run 'nmcli --ask device wifi connect' from serial terminal to connect to WiFi

(run 'systemctl stop ip_print' to stop the check)
```

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 "TM3"
9. Set the host name or IP address to the IP address noted down in step 3.
10. Set the username to "root"



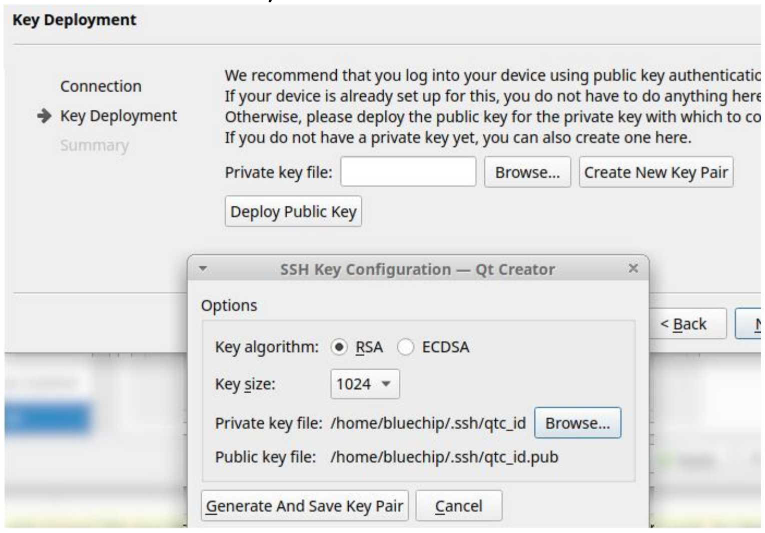
Connection

The name to identify this configuration: TM3

The device's host name or IP address: 192.168.1.180

The username to log into the device: root

11. Click Next to display the Key Deployment page.
12. Click Create New Key Pair button and confirm the default values (RSA, 1024 bit) by clicking Generate And Save Key Pair button.



Key Deployment

We recommend that you log into your device using public key authenticatic
If your device is already set up for this, you do not have to do anything here
Otherwise, please deploy the public key for the private key with which to co
If you do not have a private key yet, you can also create one here.

Private key file:

SSH Key Configuration — Qt Creator

Options

Key algorithm: ● RSA ○ ECDSA

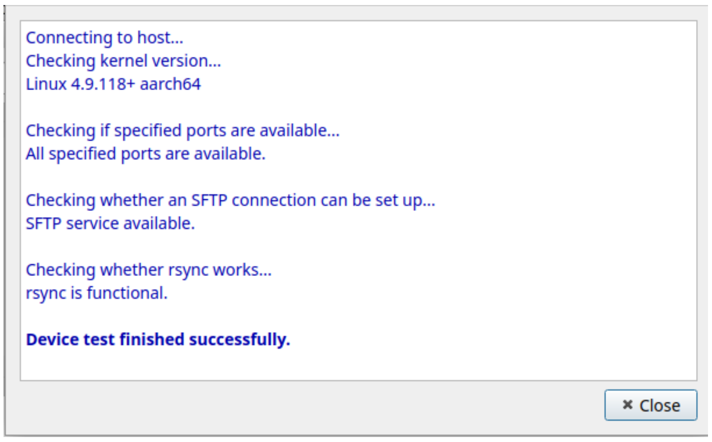
Key size: 1024

Private key file: /home/bluechip/.ssh/qtc_id

Public key file: /home/bluechip/.ssh/qtc_id.pub

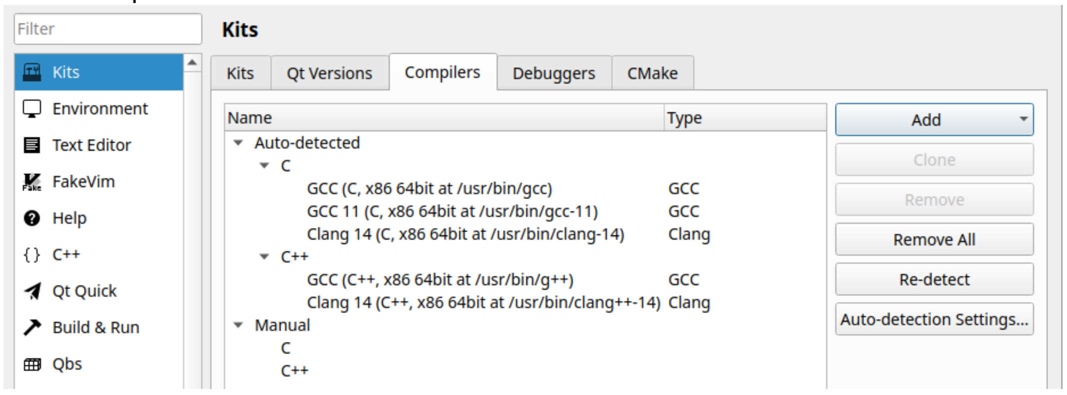
13. On Key Deployment page click Deploy Public Key button. When password is requested enter 'password' without quotes. A confirmation pop-up 'Deployment finished successfully' should be displayed. Close it.

14. Click Next then click Finish. Verify that the Device test was successful, and then click Close.



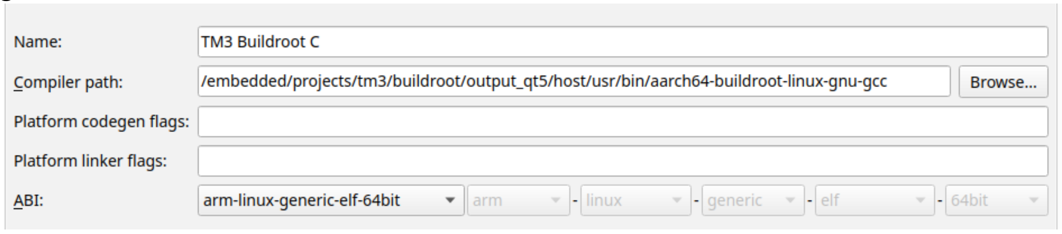
```
Connecting to host...
Checking kernel version...
Linux 4.9.118+ aarch64

Checking if specified ports are available...
All specified ports are available.

Checking whether an SFTP connection can be set up...
SFTP service available.

Checking whether rsync works...
rsync is functional.

Device test finished successfully.
```

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



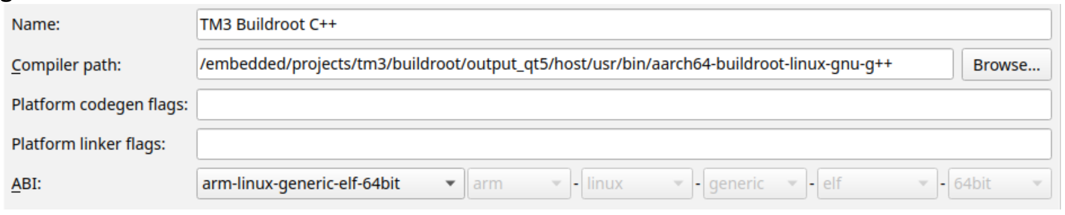
Name	Type
Auto-detected	
C	
GCC (C, x86 64bit at /usr/bin/gcc)	GCC
GCC 11 (C, x86 64bit at /usr/bin/gcc-11)	GCC
Clang 14 (C, x86 64bit at /usr/bin/clang-14)	Clang
C++	
GCC (C++, x86 64bit at /usr/bin/g++)	GCC
Clang 14 (C++, x86 64bit at /usr/bin/clang++-14)	Clang
Manual	
C	
C++	

18. Set Name to “TM3 Buildroot C”.
19. Set Compiler path to "/embedded/projects/tm3/buildroot/output_qt5/host/usr/bin/aarch64-buildroot-linux-gnu-gcc”



Name: TM3 Buildroot C
Compiler path: /embedded/projects/tm3/buildroot/output_qt5/host/usr/bin/aarch64-buildroot-linux-gnu-gcc
Platform codegen flags:
Platform linker flags:
ABI: arm-linux-generic-elf-64bit - arm - linux - generic - elf - 64bit

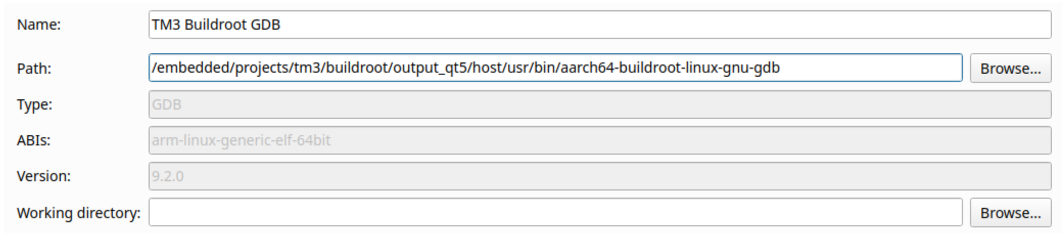
20. Ensure ABI is set to “arm-linux-generic-elf-64bit”
21. Click Apply
22. Add C++ compiler. Click “Add” → “GCC” → “C++”
23. Set Name to “TM3 Buildroot C++”
24. Set Compiler path to “/embedded/projects/tm3/buildroot/output_qt5/host/usr/bin/aarch64-buildroot-linux-gnu-g++”



Name: TM3 Buildroot C++
Compiler path: /embedded/projects/tm3/buildroot/output_qt5/host/usr/bin/aarch64-buildroot-linux-gnu-g++
Platform codegen flags:
Platform linker flags:
ABI: arm-linux-generic-elf-64bit - arm - linux - generic - elf - 64bit

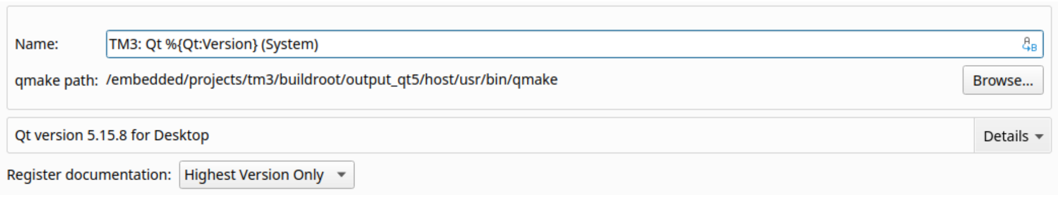
25. Ensure ABI is set to “arm-linux-generic-elf-64bit”

26. Click Apply
27. In the left hand pane select, “Kits” and then select the “Debuggers” tab. Click “add” button.
28. Set the name to, “TM3 Buildroot GDB”
29. Set the path to, "/embedded/projects/tm3/buildroot/output_qt5/host/usr/bin/aarch64-buildroot-linux-gnu-gdb”



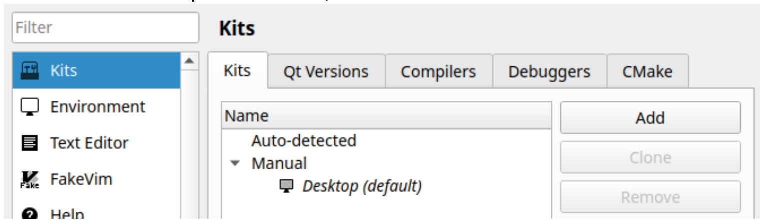
Name: TM3 Buildroot GDB
Path: /embedded/projects/tm3/buildroot/output_qt5/host/usr/bin/aarch64-buildroot-linux-gnu-gdb
Type: GDB
ABIs: arm-linux-generic-elf-64bit
Version: 9.2.0
Working directory:

30. Click Apply
31. In the left hand pane select, “Kits” and then select the “QT Versions” tab. Click “Add...” button.
32. Select the qmake executable, “/embedded/projects/tm3/buildroot/output_qt5/host/usr/bin/qmake”
33. Set the Version name to “TM3: Qt %{Qt:Version} (System)”

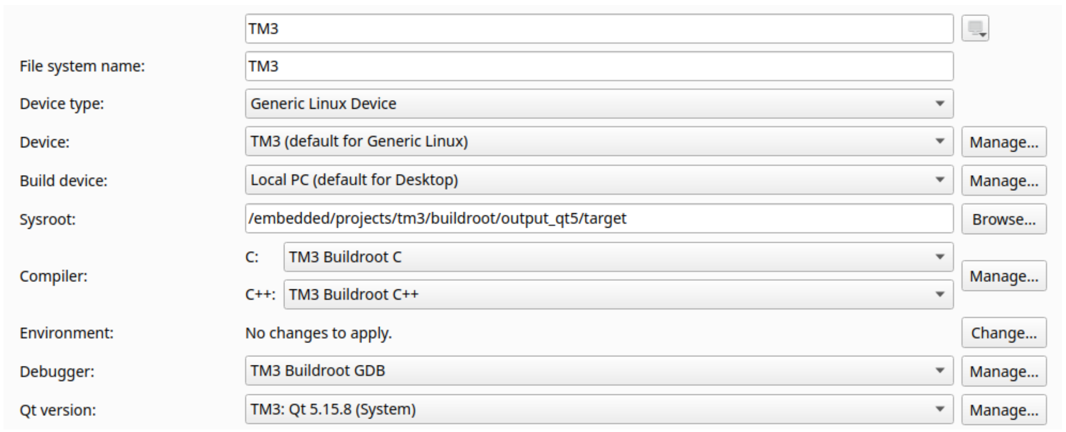


Name: TM3: Qt %{Qt:Version} (System)
qmake path: /embedded/projects/tm3/buildroot/output_qt5/host/usr/bin/qmake
Qt version 5.15.8 for Desktop
Register documentation: Highest Version Only

34. Click Apply
35. In the left hand pane select, “Kits” and then select the “Kits” tab. Click “Add”.



36. Set Name to “TM3”
37. Set File system name to “TM3”
38. Select Device Type to “Generic Linux Device”
39. Select Device to “TM3 (default for Generic Linux)”
40. Set Sysroot to “/embedded/projects/tm3/buildroot/output_qt5/target”
41. Set Compilers to “TM3 Buildroot C” and “TM3 Buildroot C++”
42. Set debugger to “TM3 Buildroot GDB”
43. Set Qt Version to “TM3: Qt 5.15.8 (System)”



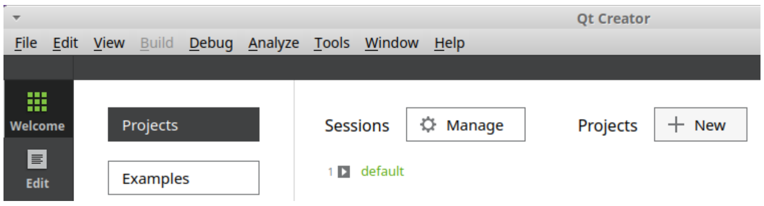
	TM3	
File system name:	TM3	
Device type:	Generic Linux Device	
Device:	TM3 (default for Generic Linux)	Manage...
Build device:	Local PC (default for Desktop)	Manage...
Sysroot:	/embedded/projects/tm3/buildroot/output_qt5/target	Browse...
Compiler:	C: TM3 Buildroot C C++: TM3 Buildroot C++	Manage...
Environment:	No changes to apply.	Change...
Debugger:	TM3 Buildroot GDB	Manage...
Qt version:	TM3: Qt 5.15.8 (System)	Manage...

44. Press Apply button and then OK button.

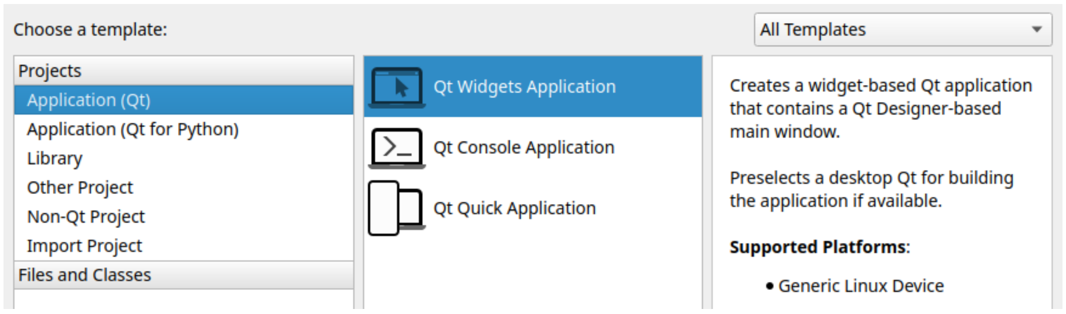
8.2.3 Setup a simple QT5 “Hello World” application

The following section will describe how to setup and deploy a simple “Hello world” application to TM3.

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



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



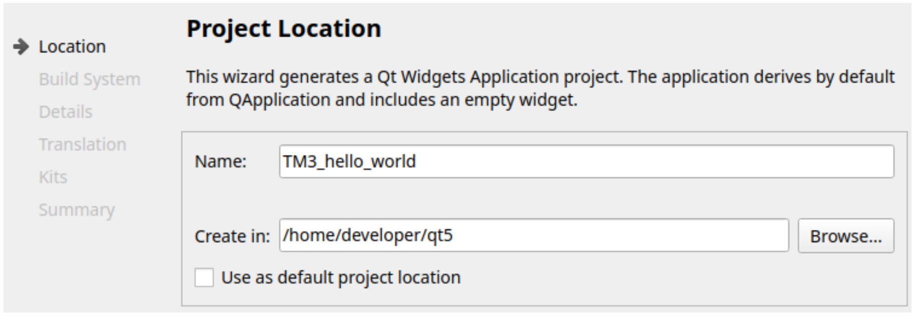
Creates a widget-based Qt application that contains a Qt Designer-based main window.

Preselects a desktop Qt for building the application if available.

Supported Platforms:

- Generic Linux Device

4. Set the name to “TM3_hello_world”
5. Set the project directory (Create in) to /home/developer/qt5 and click Next button



Project Location

This wizard generates a Qt Widgets Application project. The application derives by default from QApplication and includes an empty widget.

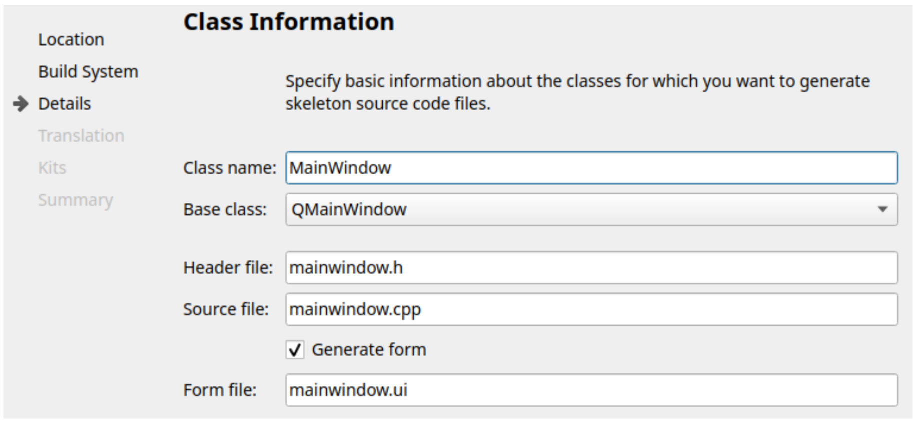
Name: TM3_hello_world

Create in: /home/developer/qt5

☐ Use as default project location

6. Set the Build system to ‘qmake’ and click Next button.

7. Leave the default options intact on the Class Information page and click Next button.

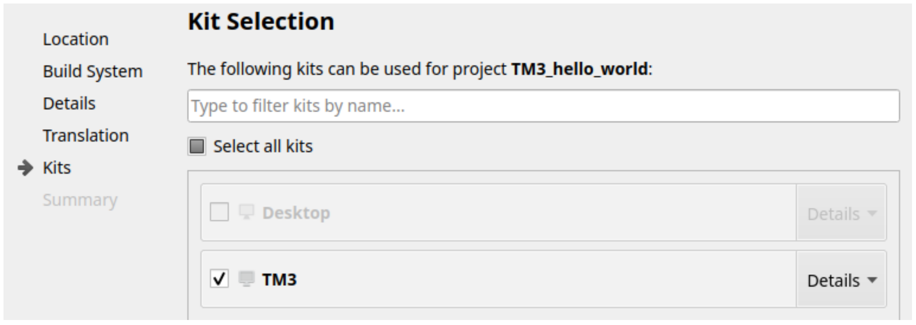


Class Information

Specify basic information about the classes for which you want to generate skeleton source code files.

Class name:	MainWindow
Base class:	QMainWindow
Header file:	mainwindow.h
Source file:	mainwindow.cpp
	✓ Generate form
Form file:	mainwindow.ui

8. On the Translation page click Next button (no translation file required).
9. On the Kits Selection page select the “TM3” kit, and click Next button.



Kit Selection

The following kits can be used for project **TM3_hello_world**:

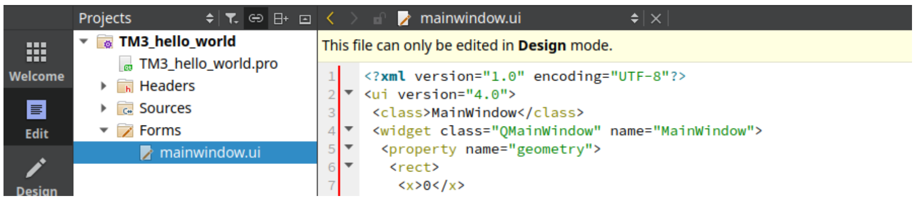
Type to filter kits by name...

☐ Select all kits

☐ Desktop — Details

✓ TM3 — Details

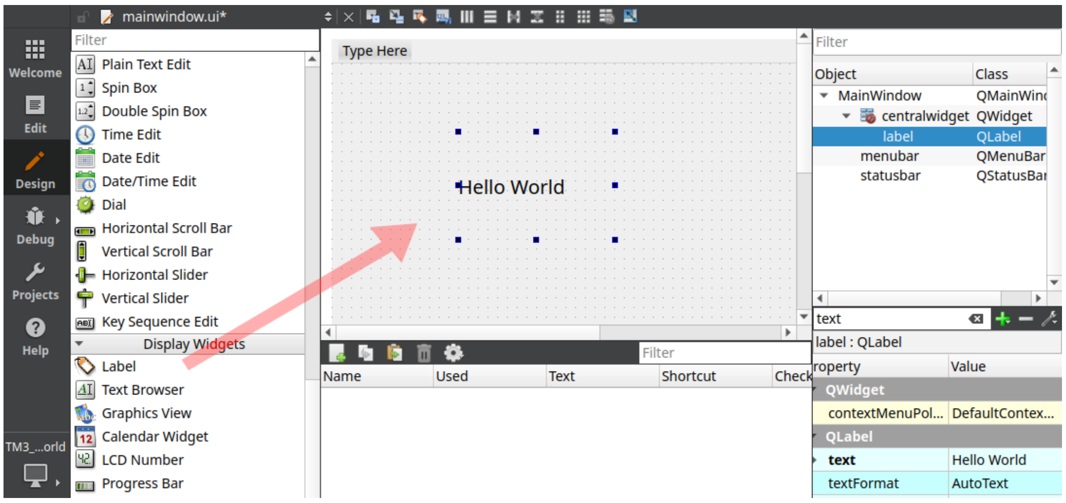
10. On the Summary page click Finish button.
11. In the Projects view, double click “mainwindow.ui” to open the forms designer.



This file can only be edited in **Design** mode.

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
```

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



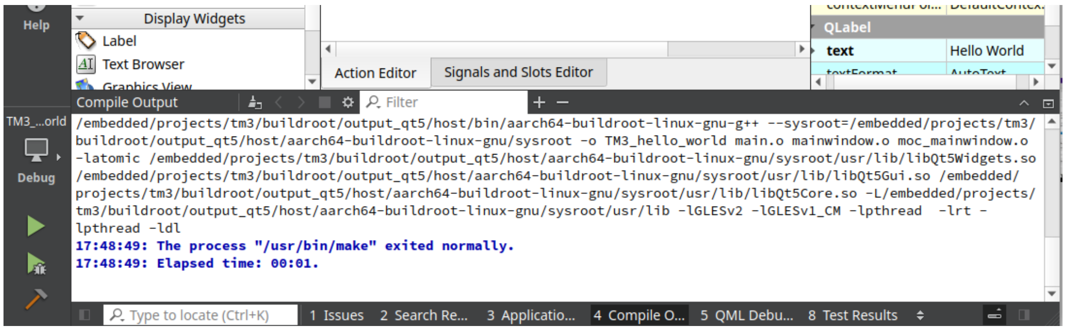
Object	Class
MainWindow	QMainWin
centralwidget	QWidget
label	QLabel
menubar	QMenuBar
statusbar	QStatusBar

text

label : QLabel

roperty	Value
QWidget	
contextMenuPol...	DefaultContex...
QLabel	
text	Hello World
textFormat	AutoText

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



```
/embedded/projects/tm3/buildroot/output_qt5/host/bin/aarch64-buildroot-linux-gnu-g++ --sysroot=/embedded/projects/tm3/
buildroot/output_qt5/host/aarch64-buildroot-linux-gnu/sysroot -o TM3_hello_world main.o mainwindow.o moc_mainwindow.o
-latomic /embedded/projects/tm3/buildroot/output_qt5/host/aarch64-buildroot-linux-gnu/sysroot/usr/lib/libQt5Widgets.so
/embedded/projects/tm3/buildroot/output_qt5/host/aarch64-buildroot-linux-gnu/sysroot/usr/lib/libQt5Gui.so /embedded/
projects/tm3/buildroot/output_qt5/host/aarch64-buildroot-linux-gnu/sysroot/usr/lib/libQt5Core.so -L/embedded/projects/
tm3/buildroot/output_qt5/host/aarch64-buildroot-linux-gnu/sysroot/usr/lib -lGLESv2 -lGLESv1_CM -lpthread  -lrt -
lpthread -ldl
17:48:49: The process "/usr/bin/make" exited normally.
17:48:49: Elapsed time: 00:01.
```

15. Monitor the “4 Compile Output” window for build completion without errors

16. Select Build → Run (Ctrl + R) to deploy and run the application on the TM3 hardware.

Appendix A - Known Problems

1) The suspend to RAM feature causes intermittent issues with the power management chip firmware in the SoC. The issue is identified by the following message in the kernel system log:

[SCP Error] :hard syn message error

There is no known solution to fix the issue.

2) Ubuntu 22.04 OS reboot may occasionally take extra time due to a lock in xscreensaver application. This happens only when the board reboot is issued in less than 50 seconds after the desktop appears on the screen. The issue may be fixed in future Ubuntu package upgrades.

Solution: uninstall xscreensaver by running the following command:

```
sudo apt purge xscreensaver
```

Appendix B - Change Log

Issue	Date	Author	Changes
1.0	08/06/2022	M. Olejnik	Initial draft, based on BCT TM1 Linux document
1.1.0	17/03/2023	M. Olejnik	Section 1, added information about host boards Section 3, reworded Ubuntu Core to Ubuntu OS (not to be confused with Ubuntu's IOT product) Section 3.2, updated the dtb table Section 4, 5 and 6, replaced references of HB5 to more generic 'host board' Section 6.3: fixed serial port names Section 6.6: Updated pinout, CLASS D amp and HDMI audio Section 7.2 and 7.3.3: Fixed mmc_root name
1.1.1	24/03/2023	M. Olejnik	Section 5.1, improved clarity, removed reference to img2simg tool as it is not compatible.
1.1.2	12/06/2023	M. Olejnik	Section 3.2, added .dtb file for 12” HB8 board.
1.1.3	26/06/2023	M. Olejnik	Added section 3.3 Using customised Linux kernel device tree. Added note to section 6.8 TM3 Watchdog

1.2.0	11/10/2023	M. Olejnik	Removed HB5, HB6 and Tmxdevboard references from the whole document. These boards were superseded by HB8 and HB9 boards. Added Wifi information (Sections 6.5.1 and 6.5.2). Removed CB3 information (CAN, Accelerometer) as this option it is not compatible with HB8 and HB9 boards Fixed section 6.13.2 Hdmi - Xorg configuration path to reflect location on Ubuntu 22.04 root fs. U-boot bootdelay information changed to 1 Updated Appendix A: known problems (removed Ubuntu 18.04 issue, added Suspend to RAM issue, added xscreensaver issue). Updated version of the TM3 bsp linux archive to v110 (now includes Ubuntu 22.04 root fs). RTC battery: added note about disconnected battery.