G3 MANUAL

Version 1.0 | January 23, 2024

Reach Technology | reachtech.com

Sales | 503-675-6464 x2 | sales@reachtech.com

Technical Support | 503-675-6464 x1 | techsupport@reachtech.com

Note: Software included is subject to a license agreement as described in this manual.

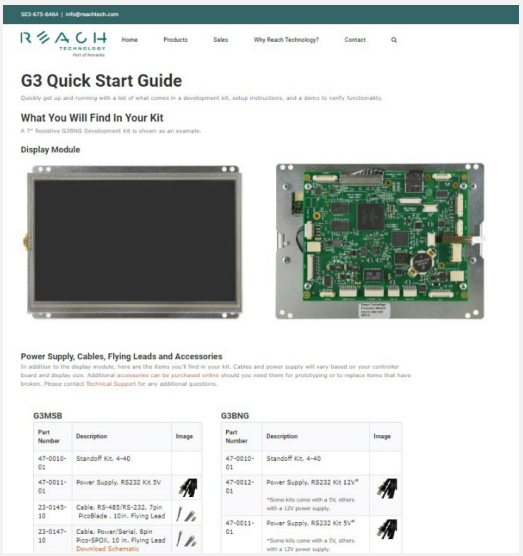
Introduction

This manual is for people wanting to create applications for G3 modules. Find more information in the G3 Development Process, G3 Software and G3 Hardware sections.

Additional Resources

We are here to help. Do not hesitate to contact our technical support team (503-675-6464, techsupport@reachtech.com).

G3 Quick Start Guide



If you have not already, visit the G3 Quick Start Guide to quickly get up and running with a list of what comes in a development kit, setup instructions, and a demo to verify functionality.

Table of Contents

Development Process

We recommend using a desktop development environment that supports cross-compilation to the ARM instruction set Application Programming Interface (API) and the embedded Linux Application Binary Interface (ABI) to develop G3 applications. There are several reasons for this:

- There are better, often GUI-based, tools on the desktop.
- Increase your productivity by creating and testing most applications on the desktop at higher speeds.
- The target is space-constrained, so it has no compiler suite installed[1].
- Compiling applications on the target can be very slow[2].
- The desktop environment also supports on-target debugging.

Development Environment Options

Choose from one of four development environment setup options listed in order of recommendation.

Option	Description
Linux Desktop	Use the Yocto Software Development Kit (SDK) and Graphical User Interface (GUI) toolkit on a Linux desktop.
Virtual Machine (VM)	Use the G3 Developer VM[3].

[1] A compiler exists on the SD card image (intended for developing products), not on the eMMC image (intended for shipping products). Due to space limitations, the toolchain provided on the SD card image does not have a complete set of include files. If needed, see this workaround.

[2] This can be worked around using a distcc-based build farm. However, that is beyond the scope of this manual.

[3] The G3 Developer VM can run on any host that supports VirtualBox and has the requisite physical resources. If you want to rapidly start development on a Linux host, the G3 Developer VM may be a faster path to get started than installing all the tools natively.

Linux Desktop Development

Using a Linux Desktop is the top recommendation for many reasons:

- It is easier to set up and maintain.
- It performs better using lower hardware requirements.
- It has no potential source code file line ending or POSIX permission issues.
- It has no potential case-blind filesystem issues.
- It uses host compilers like those on the target, so it is easy to debug many executables on your desktop if they do not use target-specific hardware[4].
- It is much closer to the target environment, which makes testing things like required shell scripts or configuration files on your desktop easy.
- There are almost no issues with a non-deterministic enumeration of USB->serial converters[5].

However, not everyone has the option, or preference, to set up a Linux machine to develop the application for the G3 module. If this is the option you would like to choose, see Linux Desktop Installation and Linux Desktop Configuration.

Linux Development Environment Host Requirements

Item	**Requirement**
CPU	Two Core, Hyper-threaded i5 or Better (Four Hardware Threads)
RAM	8 GB or Better
DISK	250 GB or Better (Solid State Drives preferred)
NIC	100Mbit or Gbit Ethernet
Operating System (OS)	Linux Mint 20 XFCE, Gentoo AMD64 17.1 Multi-lib XFCE, Ubuntu 18.046[6]

[4] You can work around this issue using mock routines to simulate hardware responses during early development.

[5] As long as USB->serial convertors stay in the same USB port (or hub port), they will always enumerate in the same order. This is not always true with Windows 10/11.

[6] The G3-specific Yocto SDK has been tested and is known to work on these Linux distributions. No others are supported.

VirtualBox Virtual Machine (VM) Development

VirtualBox[7] works well for all but the smallest projects. There are some issues to avoid. See Windows 10/11 desktop installation and configuration directions for more information. It works well on Windows, Mac, or Linux platforms with all the benefits and drawbacks of a VM.

Benefits:

- The VM operating system runs independently of the host operating system.
- The VM is preconfigured with all necessary toolchain installations and sample configurations to enable the target hardware.

Drawbacks:

- The required resources are higher than running a Linux host.
- USB enumeration can be problematic.
- Introduces the possibility of line-ending character problems.
- Opens the chance of POSIX permission incompatibility problems in the host filesystem.
- It makes it possible to have case-blind filesystem problems.

The G3 Developer VM runs Linux Mint 20 XFCE (a Ubuntu derivative we recommend over the Unity desktop) internally.

Windows 10/11 Development Environment Host Requirements

Item	**Requirement**
CPU	Four Core, Hyper-threaded i5 or Better (Eight Hardware Threads)
RAM	16 GB or Better
DISK	500 GB or Better (Solid State Drives Preferred)
NIC	100Mbit or Gbit Ethernet
VirtualBox Version	6.1.16 or Newer (Matching Extension Pack)

Next Steps

Once you have selected and set up your preferred development environment, view the usage section for details on each available application/GUI toolkit option.

[7] Some customers have asked if the G3 Developer VM can use VMware. It is technically possible. However, the scope of building a VMware virtual machine using the virtual disk from the G3 Developer is not in our development plan.

Tools

G3 Developer Software Development Kit (SDK)

You can install the G3 Developer Yocto SDK, built alongside the actual embedded image for the G3 module, on almost any Linux distribution. Sometimes, you may need to install it inside a Docker container for compatibility reasons.

> IMPORTANT: Do not install the G3 Developer SDK directly on Windows 10 or any other version of Windows, it will not work.

G3 Developer Virtual Machine (VM)

Qt (pronounced "cute") is a free and open-source toolkit for creating graphical user interfaces and cross-platform applications that run on the G3 module. This tool is only available on the Qt versions of the G3 Developer VM.

The Developer VM versions that support Qt development come with Qt 5.15.2 or greater pre-installed, the current LTS release.

Cross Compiler

The G3 Developer VM has an ARM cross-compiler toolchain (GCC 9.3.0 or later) pre-installed as part of the G3 Developer SDK.

Qt Creator

Qt Creator is a cross-platform Integrated Development Environment (IDE). Qt Creator runs on Windows, Linux, and macOS X desktop operating systems, allowing developers to create applications across desktop, mobile, and embedded platforms. This tool is only available on the Qt versions of the G3 Developer VM. The G3 Developer VM comes pre-installed with Qt Creator 4.11.0 or greater.

Other Pre-installed Tools

The G3 Developer VM includes several pre-installed applications that may be helpful for development. They may need to be updated periodically. Refer to each tool's online documentation for more information.

VSCodium

VSCodium is an open-source code editor similar to Microsoft Visual Studio Code (VS Code). One significant difference is that it does not include the proprietary telemetry and tracking features in the official VS Code distribution.

Gvim

The Gvim editor is Vim inside a GUI window, with a menu and toolbar system supplementing the keystroke commands found in the shell version of Vim.

FileZilla

FileZilla is a file transfer application consisting of FileZilla Client and FileZilla Server, both of which support FTP and FTPS (FTP over SSL/TLS).

Use FileZilla to transfer files from the G3 Developer VM to the target and back.

Meld

Meld is a visual differencing and merge tool targeted at developers. It compares two or three files side-by-side, color-coding the different lines and supports directory tree comparison, highlighting directories and files that do not match.

Meld supports many version control systems, including Git and Subversion.

Installation

As discussed earlier, several options exist for the G3 module.

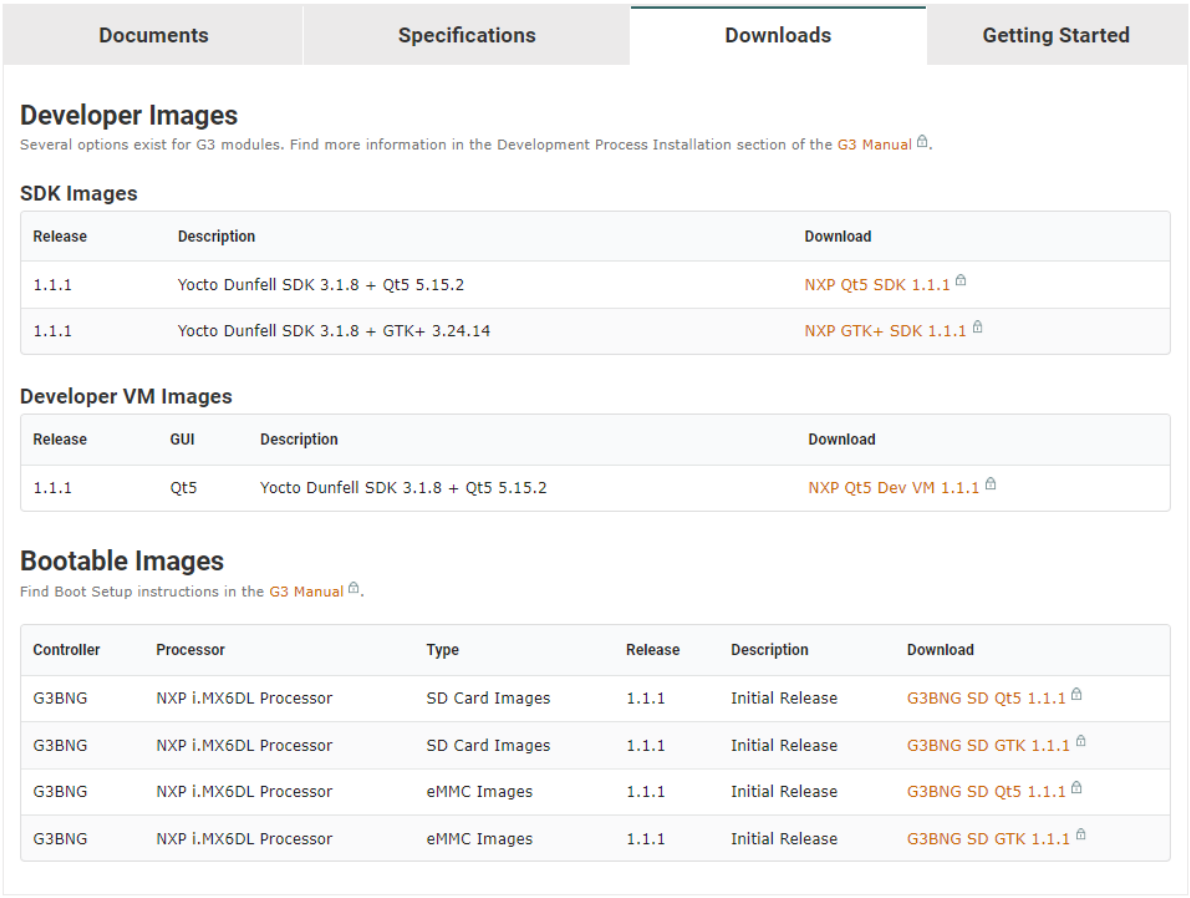
1. Find instructions for the basic installation of your preferred G3 development environment below.
2. Then, visit the post-installation configuration setup.

Developer Images

WARNING: SDK images can only be installed on a Linux desktop or a VM running a Linux guest OS.

IMPORTANT: Match the correct SDK/Developer VM image to the embedded Linux image on your G3 target: Processor, GUI toolkit and Release (e.g., 1.1.1).

To download the image you need, find your product at Compare Modules and go to its page. Scroll down to the **Downloads** tab (see an example below), and choose the appropriate image for your setup.



Documents | Specifications | Downloads | Getting Started

Developer Images

Several options exist for G3 modules. Find more information in the Development Process Installation section of the G3 Manual.

SDK Images

Release	Description	Download
1.1.1	Yocto Dunfell SDK 3.1.8 + Qt5 5.15.2	NXP Qt5 SDK 1.1.1
1.1.1	Yocto Dunfell SDK 3.1.8 + GTK+ 3.24.14	NXP GTK+ SDK 1.1.1

Developer VM Images

Release	GUI	Description	Download
1.1.1	Qt5	Yocto Dunfell SDK 3.1.8 + Qt5 5.15.2	NXP Qt5 Dev VM 1.1.1

Bootable Images

Find Boot Setup instructions in the G3 Manual.

Controller	Processor	Type	Release	Description	Download
G3BNG	NXP i.MX6DL Processor	SD Card Images	1.1.1	Initial Release	G3BNG SD Qt5 1.1.1
G3BNG	NXP i.MX6DL Processor	SD Card Images	1.1.1	Initial Release	G3BNG SD GTK 1.1.1
G3BNG	NXP i.MX6DL Processor	eMMC Images	1.1.1	Initial Release	G3BNG SD Qt5 1.1.1
G3BNG	NXP i.MX6DL Processor	eMMC Images	1.1.1	Initial Release	G3BNG SD GTK 1.1.1

Host Installation

Here are the options for setting up a Reach Technology supported development environment:

1. Linux (Debian) Desktop Installation
2. Virtual Box-based Installation (works for Windows, Mac, or Linux, including non-Debian distros)

> NOTE: On-target installations are not advised nor supported.

Linux Desktop Installation

Installing a Linux[8] development host consists of installing three components:

- Yocto SDK
- Qt
- Qt Creator

Yocto SDK Installation

Your Linux desktop system must have standard build tools installed and tested. These will allow you to develop as much of the application as practical on the host to improve productivity.

1. Download the G3 Developer SDK that matches your G3 module CPU and embedded image from the Developer Images section. The SDK comes as a tarball.
2. Unpack the tarball.
3. In the new SDK sub-directory, execute the script with a .sh extension. Accept the defaults. This will install the SDK under /opt/reach/sdk. You will need to run the script as root.
4. Remove the SDK sub-directory, as it is no longer needed.

[8] See how to set up Linux Mint 20 XFCE using the Synaptic Package Manager. This distribution is used to build the G3 Developer VM and is similar to Ubuntu with fewer issues. Although the package names will probably differ, a similar process can be applied to a RPM-based system.

Qt Installation

IMPORTANT: This task is only necessary if you are developing a Qt-based end-product application.

Your Linux desktop system must have the required set of Linux development libraries to install Qt libraries. See the Qt website for further information.

WARNING: If you use a binary Qt build, either from desktop Linux distribution packages or the official Qt download site, be sure:

- The Qt libraries version on the development host is greater than or equal to the G3 module version.
- Due to space limitations, the G3 Qt installation does not support every Qt module. Ensure your build only uses G3 supported modules.
- This see all the steps needed to install Qt on a Linux machine at download and install Qt.

NOTE: Select version 5.15.2, the supported version on G3, when loading Qt.

The basic installation is complete. Proceed to the detailed post-installation configuration setup.

VirtualBox Installation

The G3 Developer VM runs in VirtualBox.

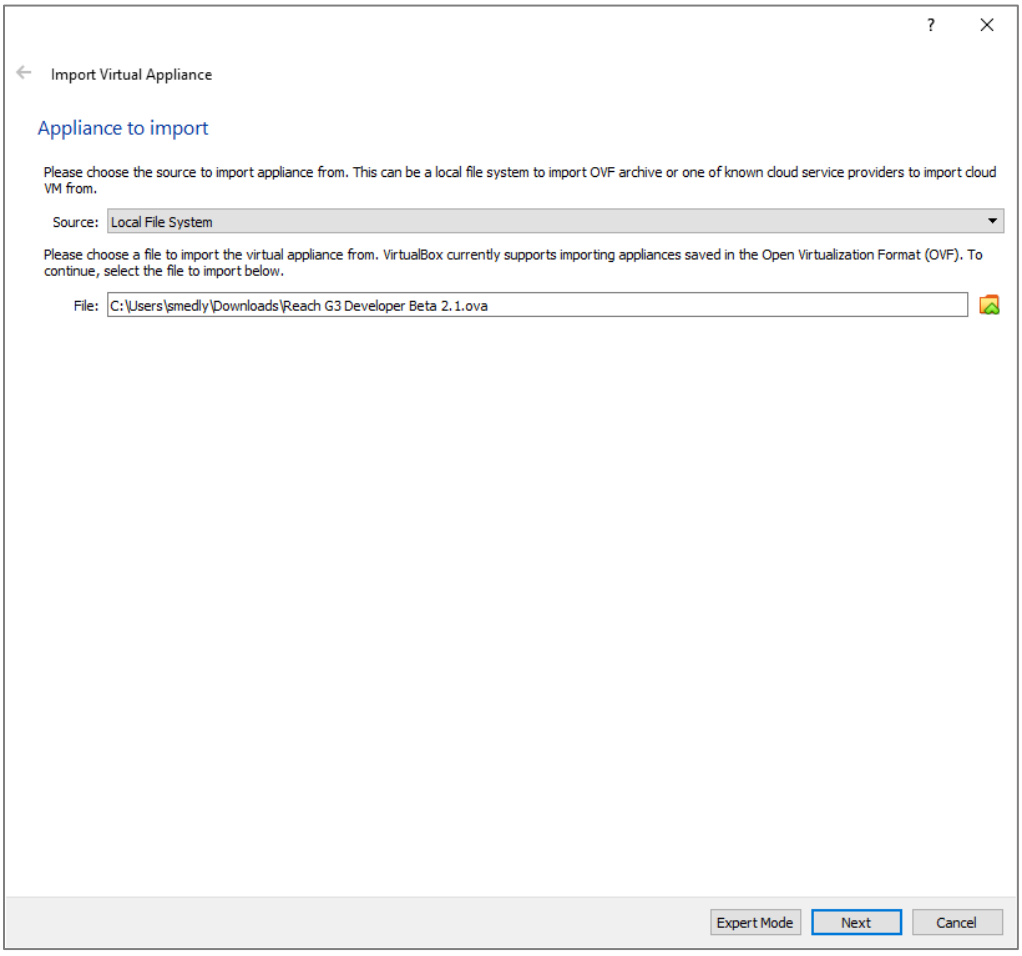
> TIP: When using a Windows host, ensure it meets or exceeds the requirements. See the requirements table for *minimum* requirements. If your Windows machine is already running many tasks, you'll need more CPU and/or more RAM. If you already have a lot of data on your disk, ensure you have at least 50 GB free.

Installing VirtualBox

We recommend following these VirtualBox installation instructions.

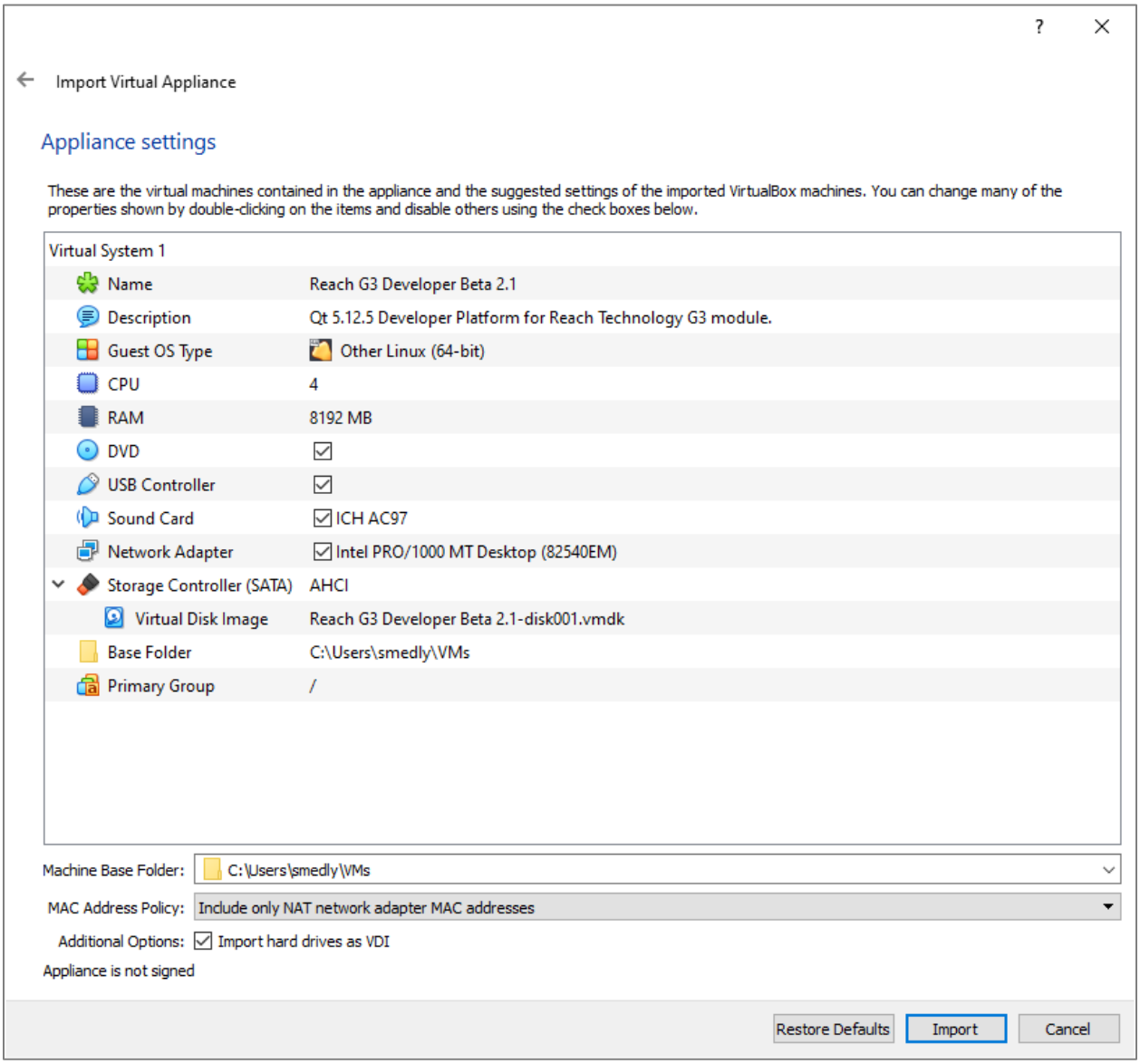
Installing the G3 Developer VM

1. Download the correct developer OVA from the Developer Images section.
2. Open VirtualBox Manager.
3. Launch the virtual appliance import dialog shown in the figure below:

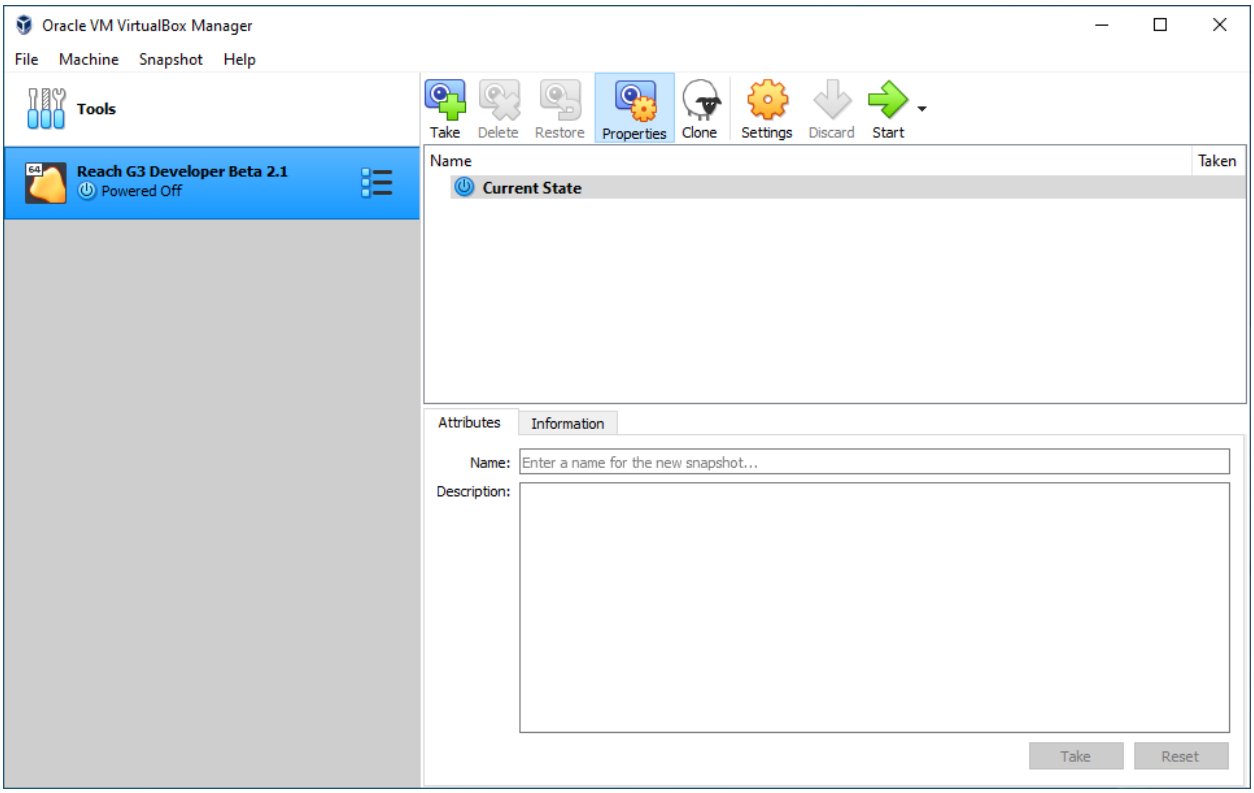


4. The file textbox will be empty when the virtual appliance import dialog launches. Click the **folder icon** to the right of the file text box to launch the virtual appliance file selector and select the **G3 Developer OVA file** you downloaded previously.

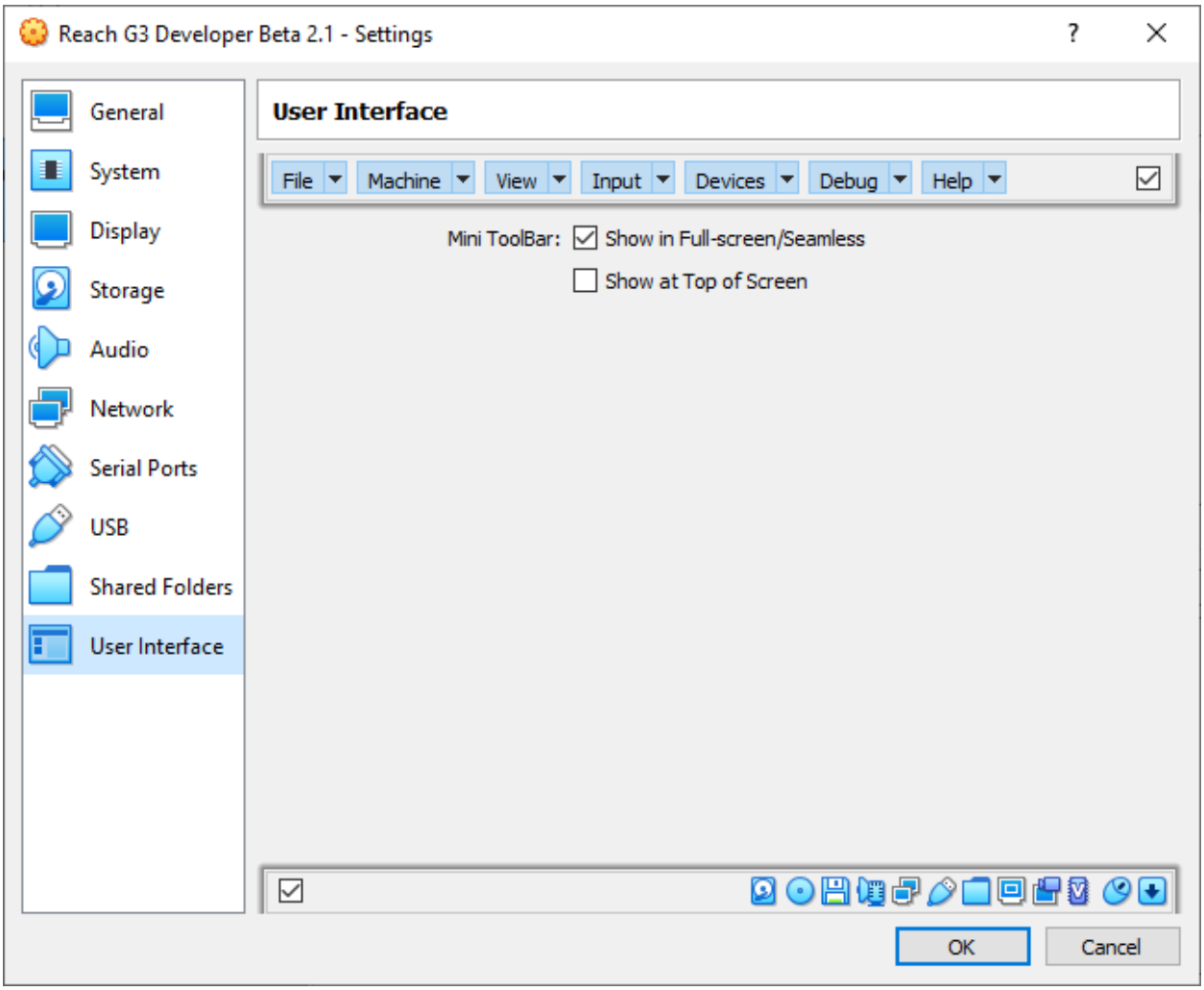
5. Click **Next** to launch the virtual appliance settings dialog, which should look like the figure below:



6. Click **Import** to import the G3 Developer OVA. The VirtualBox Manager should now look something like this:

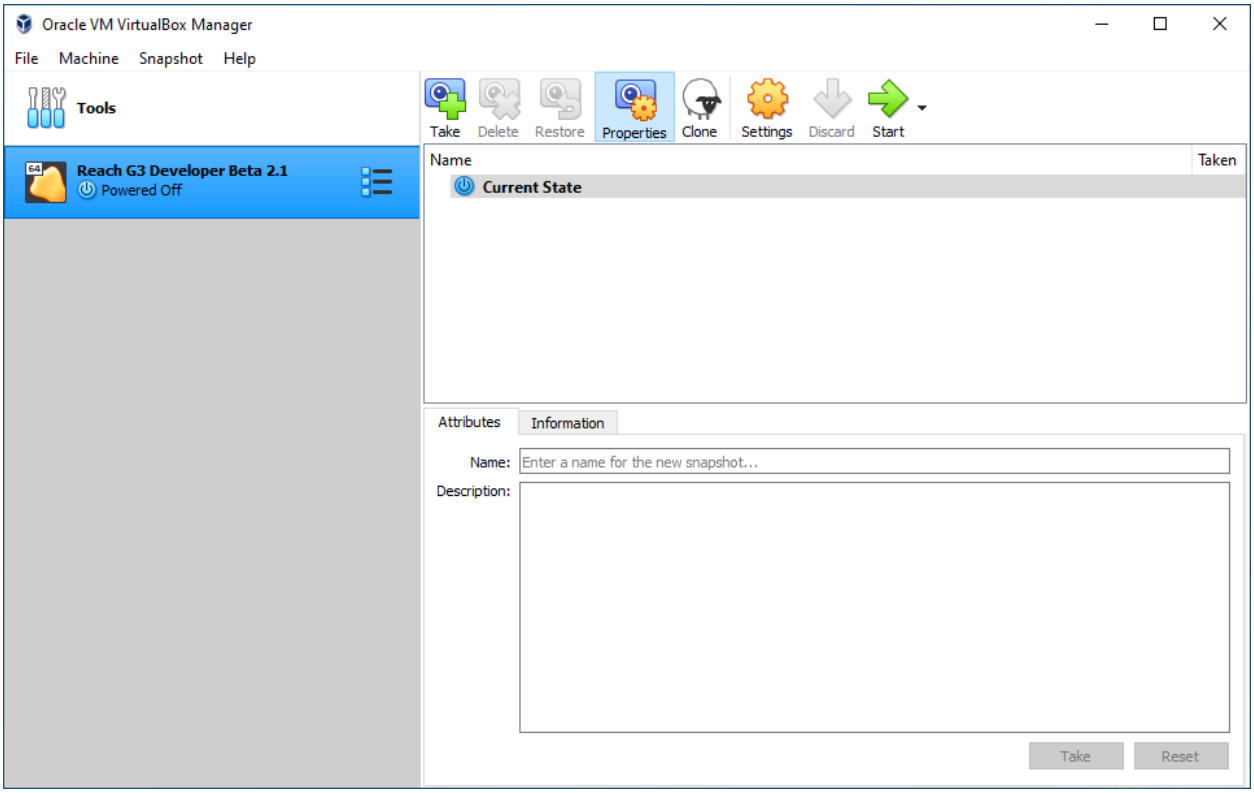


7. Click the **Settings** button to verify that your appliance is imported correctly by ensuring the various settings match the figures below. There may be slight differences due to underlying host hardware differences (e.g., the number of CPU cores)—only update settings after you have read the configuration section.

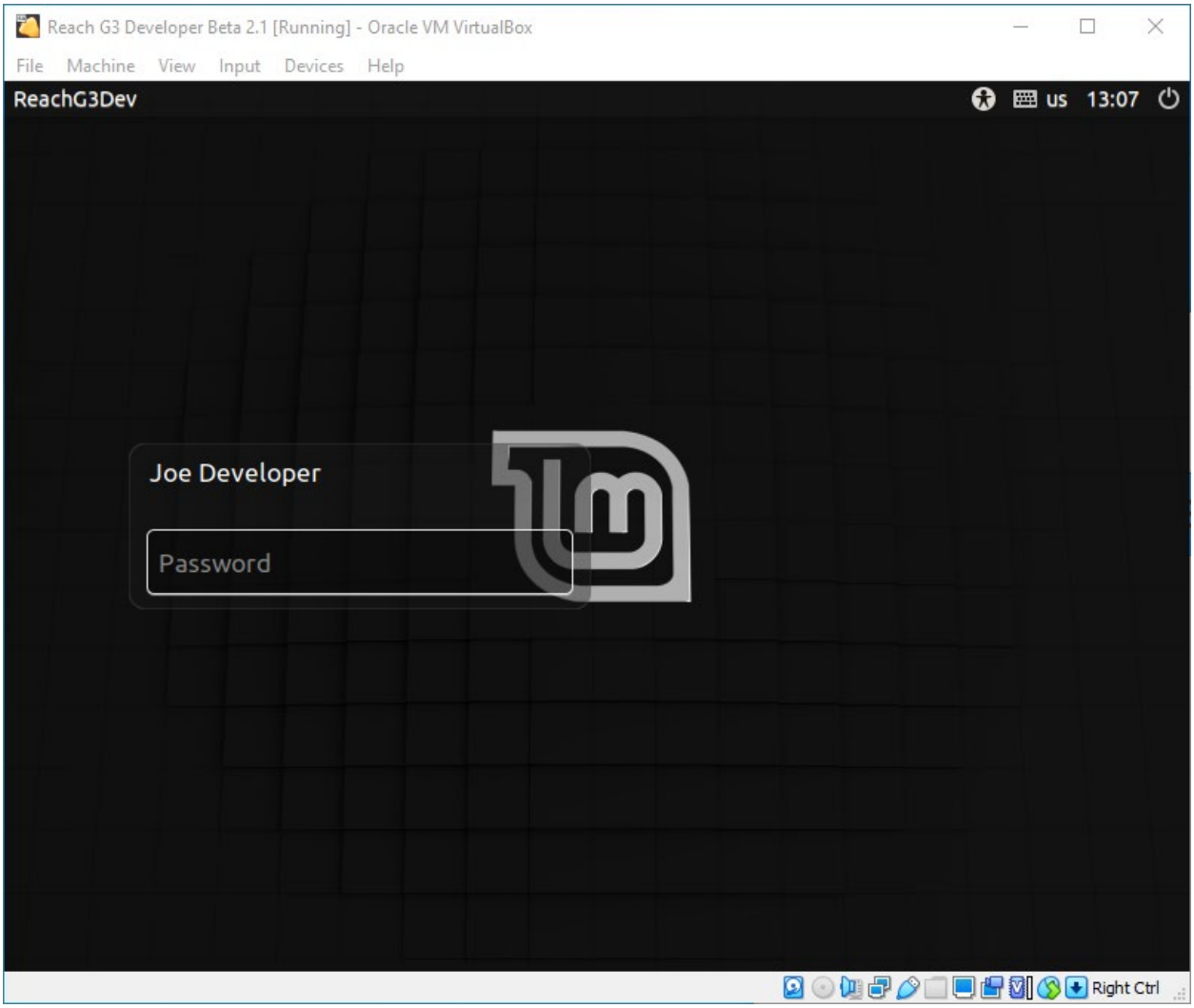


TIP: If you installed version 6.1.16 or greater, you're done with the basic installation and can move on to configuration, if not, continue through the remaining steps to update the guest additions inside the virtual machine.

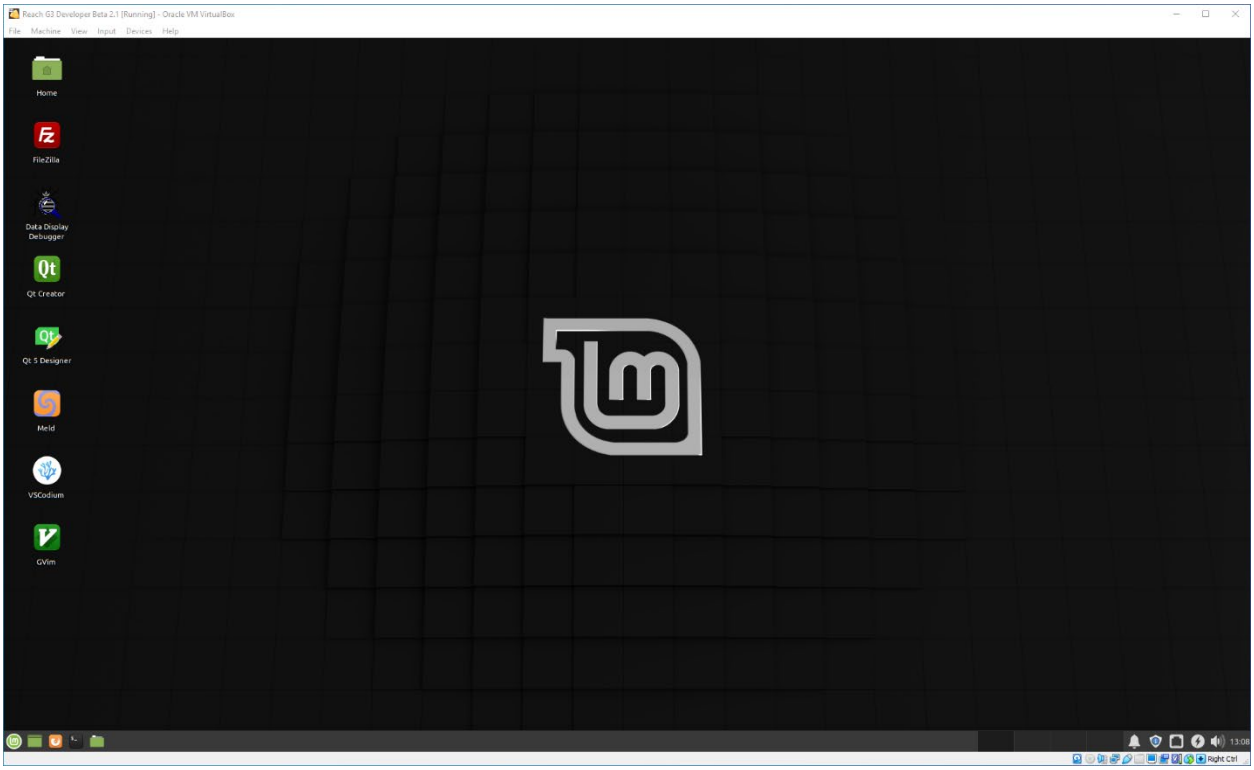
8. Select the G3 Developer VM in the left column as in the figure below:



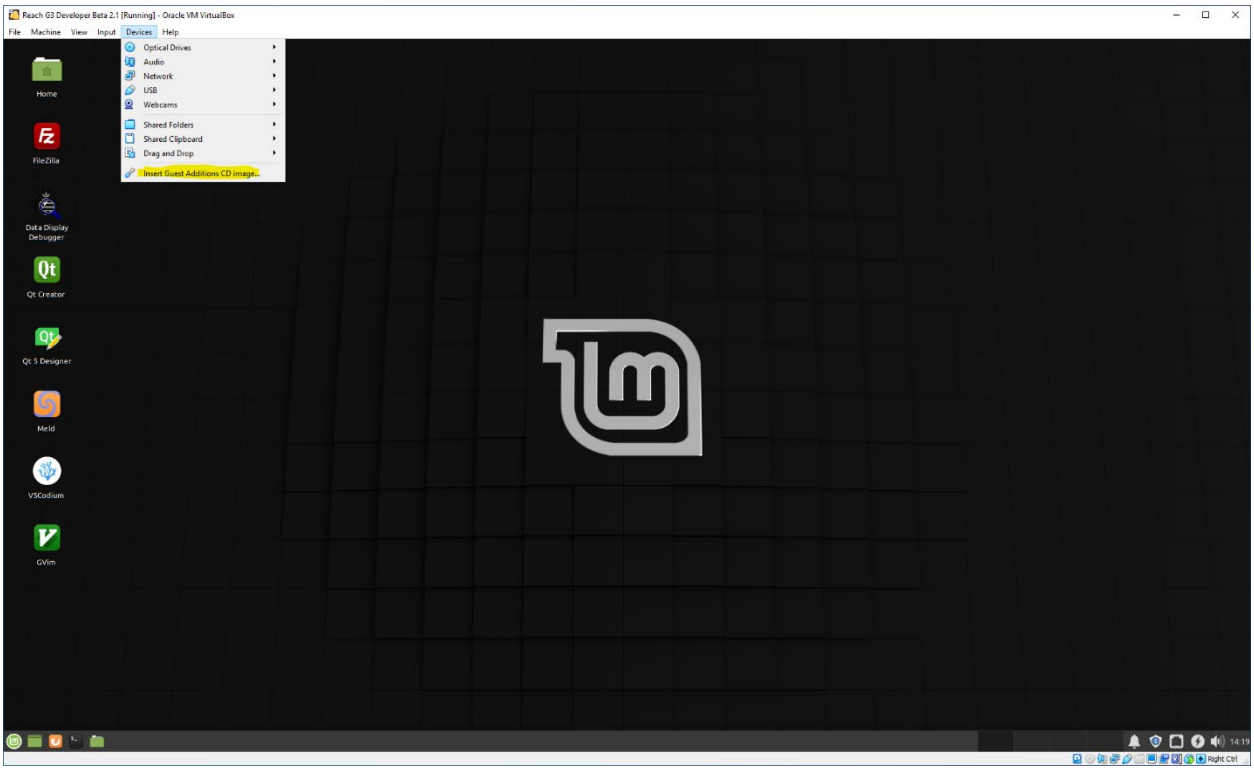
9. Click the **Start** button to launch the virtual machine. You should see a window that looks like this:



10. Login with the password ***developer***. Your window should now look like this:

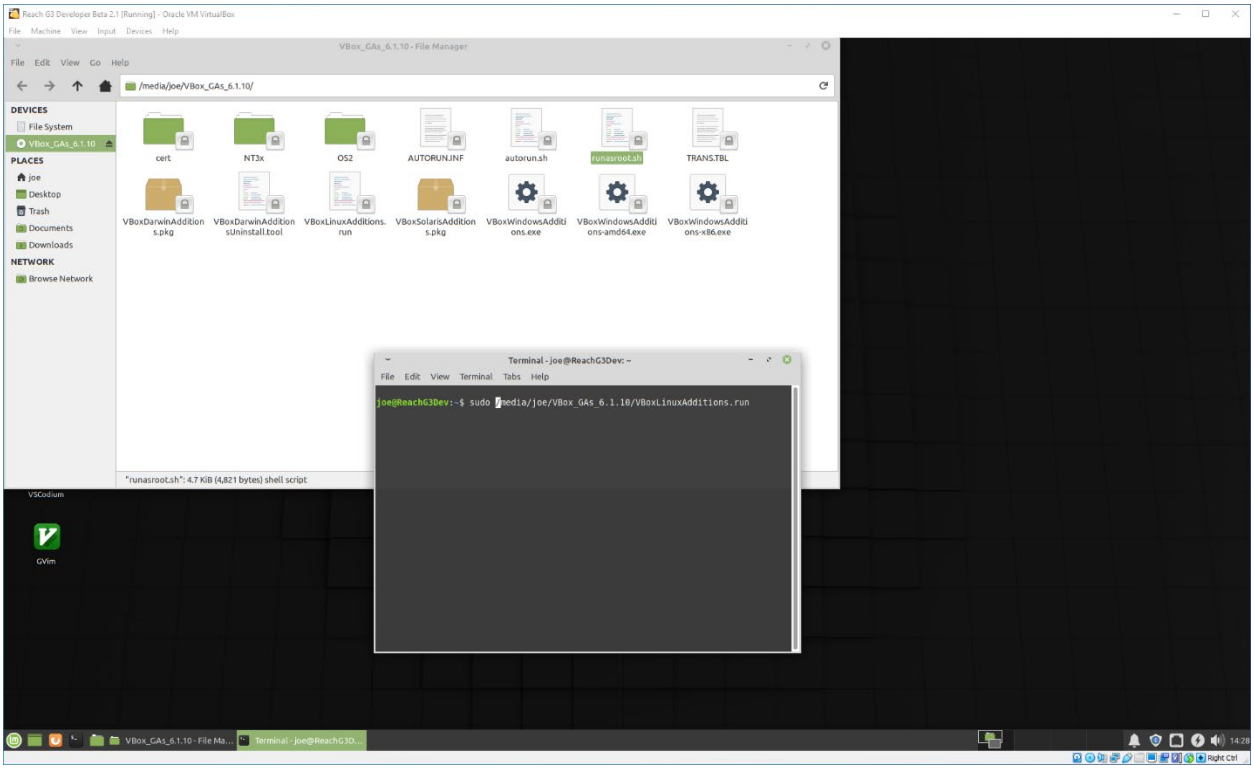


11. Click **Devices** in the menu bar and select **Insert Guest Additions CD Image**. See the figure below:



After a few seconds, Linux Mint will automount the CD image and open a File Manager window.

12. Click the **terminal launcher** button at the left end of the Linux Mint toolbar at the bottom of the desktop. It is the black one with the $ in it. Other than the command string in the terminal window, your Linux Mint desktop should look like this:



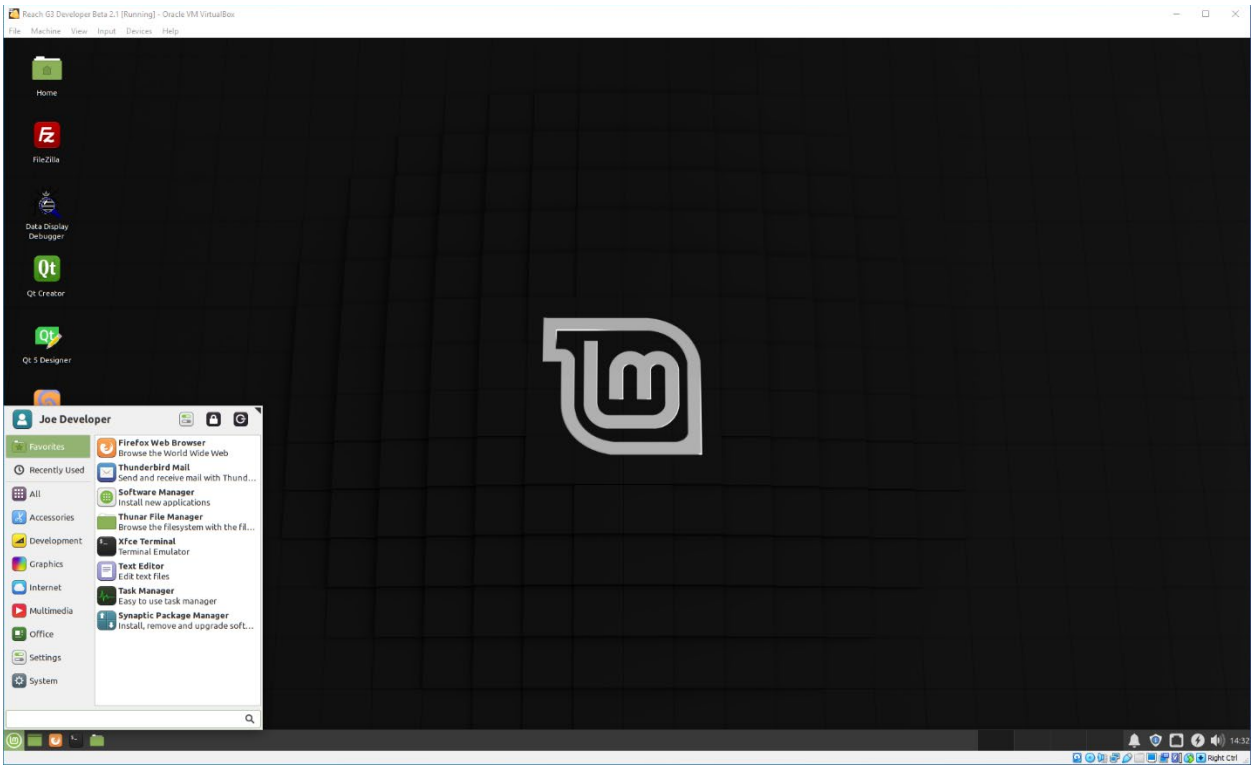
13. Type in the command in the snippet below and press <ENTER>. This step will take **several** minutes. Wait until the green "joe@ReachG3Dev:~$" shell prompt returns.

```
joe@ReachG3Dev:~$ sudo /media/joe/VBox_GAs_6.1.10/VBoxLinuxAdditions.run
```

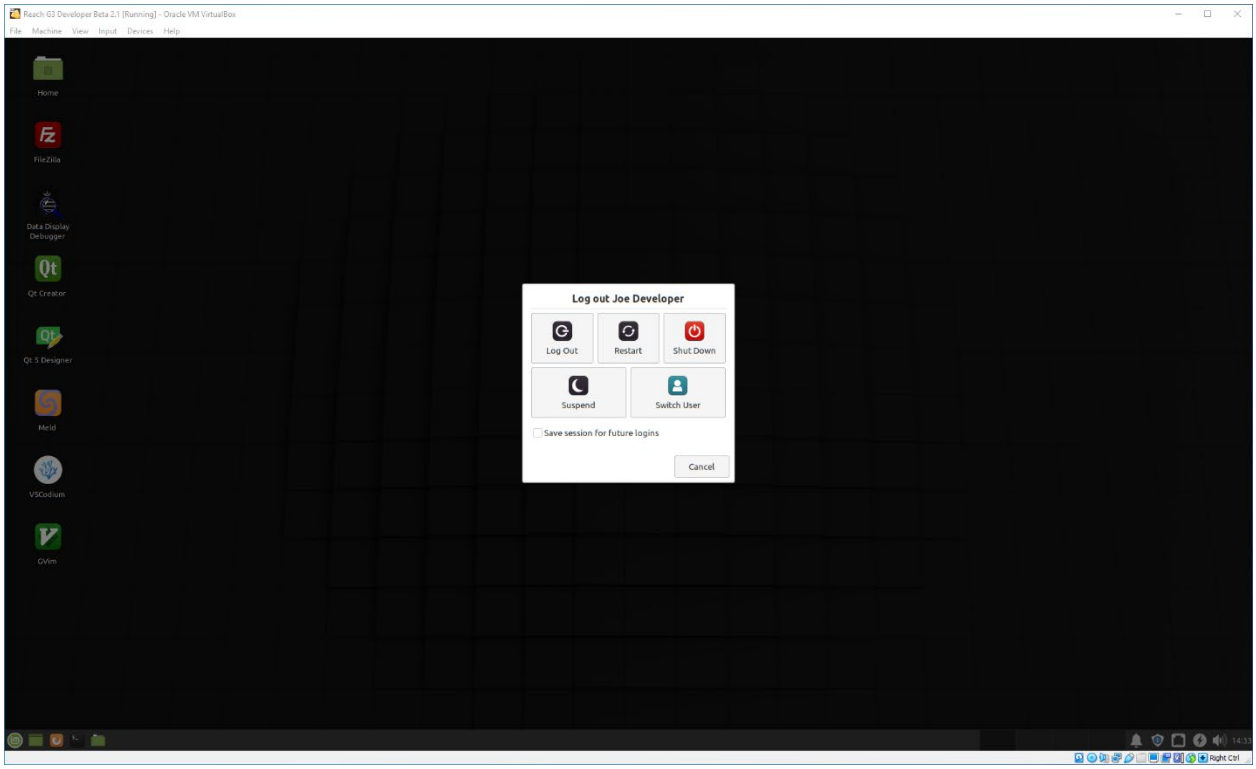
> IMPORTANT: Remember to change the "6.1.10" number sequence to match your version of VirtualBox. Look at the path in the File Manager window to confirm the proper substitution value.

14. Once the shell prompt is back, close both windows on the Linux Mint desktop by clicking the green X at the upper right corner of the window. Finish by shutting down your new Linux Mint 20 XFCE virtual machine.

15. Click the **Linux Mint Start Panel** button, the green one at the toolbar's left. Your virtual machine window should now look like this:



16. Click the **power** button at the very upper right of the Start Panel to popup the shutdown dialog, as shown below:



17. Click the red **Shutdown** button to shut down the Linux Mint virtual machine.
18. You can close that window if the VirtualBox Manager is still running.

The basic installation is now complete. Proceed to post-installation configuration setup.

Configuration

As discussed in more detail in Development Environment Options, three alternatives exist for the G3 module. The sections below contain instructions for detailed configuration for each.

In addition to setting up a development environment, you must, at a minimum, configure two things to develop applications for the G3 module:

- Set up a serial debug interface.
- Set up a wired Ethernet connection.

While your final application may not need these connections, finding a wired Ethernet connection within the product's architecture is common.

Linux Desktop Configuration

Configuring a Linux development host consists of up to four procedures:

1. Yocto SDK Configuration
2. Qt Creator Configuration
3. Serial Port Configuration
4. Network Configuration

Yocto SDK Configuration

> NOTE: Your Linux desktop system must have standard build tools installed and tested. These will allow you to develop as much of the application as practical on the host to improve productivity.

Use the Makefile script below to compile code for the Linux host or the target. We recommend you name this script /usr/local/bin/sdk_make and make it executable. Once complete, you use normal make to build for the host and sdk_make to build for the target.

```
1 | #! /bin/sh
2 | SDK_HOME="/opt/reach/sdk/imx6dl-g3-sd/2.7.4"
3 | ( . ${SDK_HOME}/environment-setup-cortexa9t2hf-neon-reach-linux-gnueabi ; make "$@" )
```

You will need to adjust the value of SDK_HOME and the name of the environment setup script to match your installation.

Qt Creator Configuration

Setting up Qt Creator on a Linux Mint box and configuring it for cross-compilation involves four main steps:

1. Installing Qt Creator on Linux Mint.
2. Cross-compilation setup.
3. Configuring Qt Creator.
4. Building and deploying.

IMPORTANT: This task is only necessary if you are developing a Qt-based end-product application. Reach Technology does not recommend Qt Creator to create general C/C++ code or projects using other GUI technologies. There are better IDEs available.

NOTE: Instructions assume your Linux desktop system has required Qt libraries (and their dependencies) and Qt Creator installed. See the Qt website for further information.

WARNING: If you use a binary Qt build, either from desktop Linux distribution packages or the official Qt download site, be sure:

- The version of Qt libraries you use on the development host is equal to or greater than the version on the G3 module.
- You don't use features available in the desktop build that may not be in the G3 module Qt build.

1. Installing Qt Creator on Linux Mint

- Update your system:
  sudo apt update && sudo apt upgrade –y

- Install Qt Creator:
  sudo apt install qtcreator

2. Cross-compilation Setup Example

IMPORTANT: The example below is using the SDK image for the G3BNG module, which uses the i.MX6 processor. If you are using the G3MSB module, with the STM32 processor, the filenames in the SDK will be slightly different.

- Obtain the SDK: Download the appropriate SDK. In this example, we will be using reach-eglfs-glibc-x86_64-reach-image-qt5-armv7at2hf-neon-imx6dl-g3-sd-toolchain-FSL-QT5-1.1.1.

- Install the SDK: Once you have obtained the SDK installer, give it execute permissions and run it.

  chmod +x reach-eglfs-glibc-x86_64-reach-image-qt5-armv7at2hf-neon-imx6dl-g3-sd-toolchain-FSL-QT5-1.1.1.sh

  reach-eglfs-glibc-x86_64-reach-image-qt5-armv7at2hf-neon-imx6dl-g3-sd-toolchain-FSL-QT5-1.1.1.sh

Follow the instructions and note the installation directory. We recommend using the default values.

3. Configuring Qt Creator

- Open Qt Creator and go to **Tools > Options.**
- Add a new compiler.
  - Go to the **Build & Run** section, then the **Compilers** tab.
  - Click the **Add** button and choose the compiler type (typically GCC for i.MX6).
  - For the compiler path, point it to the **GCC** binary in the SDK directory:
    - /opt/reach/sdk/imx6dl-g3-sd/FSL-QT5-1.1.1/sysroots/x86_64-reachsdk-linux/usr/bin/arm-reach-linux/ arm-reach-linux-gcc
    - Do the same for g++/opt/reach/sdk/imx6dl-g3-sd/FSL-QT5-1.1.1/sysroots/x86_64-reachsdk-linux/usr/bin/arm-reach-linux/ arm-reach-linux-g++
- Add a new Qt version.
  - In the **Build & Run** section, switch to the **Qt Versions** tab.
  - Click on **Add** and point it to the **qmake** binary in the SDK.
    - /opt/reach/sdk/imx6dl-g3-sd/FSL-QT5-1.1.1/sysroots/x86_64-reachsdk-linux/usr/bin/qmake
- Set up a new kit.
  - Switch to the **Kits** tab.
  - Click **Add** to create a new kit.
  - Name it appropriately (e.g., iMX6).
  - For the device type, select **Generic Linux Device**.
  - Choose the GCC and G++ compilers you added in the previous steps.
  - Choose the version you added during the last step for the Qt version.
- Configure the device.
  - Go to the **Devices** section.
  - Click on **Add** and choose **Generic Linux Device**.
  - Provide the IP address and SSH login details for the i.MX6 device or board.

4. Building and Deploying

When you create or open a Qt project, choose the appropriate G3 kit for building. Once built, deploy the application to the G3 device using the deployment options in Qt Creator.

Serial Port Configuration

To set up the communication (comm) port on a Linux system, you need to identify the port, set its parameters, and use a tool or write a program to ensure it communicates appropriately.

1. Identify the Port

Serial ports on Linux are represented as devices in the /dev directory.

Common Names:

- For standard onboard serial ports, use /dev/ttyS0, /dev/ttyS1
- For USB-to-serial adapters use /dev/ttyUSB0, /dev/ttyUSB1

Use the script dmesg | grep tty or ls /dev/tty* to list serial ports.

OR

If you plug in a USB-to-serial adapter, you can also monitor dmesg output to see which device it is assigned: using dmesg | tail.

2. Set Port Parameters

To configure the serial port, you can use the stty command. For instance, to set /dev/ttyUSB0 to 115200 baud, use stty -F /dev/ttyUSB0 115200.

Permission Issues: Your user account might need to be part of your system's dialout group (or equivalent group) to access a comm port. You can add your user account to the dialout group with sudo usermod -a -G dialout $USER.

Remember to log out and log back in for group changes to occur.

3. Communicate with the Port

Using a Terminal Program

There are several terminal programs you can use to communicate with serial ports. Examples include Screen, Minicom, Picocom, and Putty.

- To use Screen:

  sudo apt-get install screen

  screen /dev/ttyUSB0 115200

- To exit Screen:

  Press **Ctrl + A** followed by **Ctrl + **.

Using a Script/Program

You can also write scripts or programs in various languages like Python, C, and others to communicate via the serial port. For Python, the pyserial library is a popular choice.

Monitoring Serial Data

If you want to monitor data from a serial port without sending anything, you can use cat:

cat /dev/ttyUSB0

Or use dd for a more controlled read:

dd if=/dev/ttyUSB0 bs=1 count=100

The example above will read 100 bytes from the serial port.

TIPS:

- **Lock Files:** If you face issues accessing the serial port, check for lock files in /var/lock. Sometimes, programs lock the serial port and do not unlock properly after exit.
- **Hardware Flow Control:** If your device uses hardware flow control, you might need to enable it using the -crtscts option with stty.

Network Configuration

IP Address

To determine the IP address of the G3 module:

1. Connect Ethernet to the G3 module.
2. Power it up.
3. Open a terminal screen.
4. Type ifconfig.
5. Record the information. Dynamic Host Configuration Protocol (DHCP) is enabled by default.

Command Line Configuration

Use IP and Route Commands

1. To bring up an interface: ip link set dev eth0 up
2. To assign an IP address: ip addr add 192.168.1.10/24 dev eth0
3. To set the default gateway: ip route add default via 192.168.1.1

Use Ifconfig and Route

1. To bring up an interface: ifconfig eth0 up
2. To assign an IP address: ifconfig eth0 192.168.1.10 netmask 255.255.255.0
3. To set the default gateway: route add default gw 192.168.1.1

NOTE: These configurations are not persistent across reboots.

Persistent Configuration

Use /etc/network/interfaces

Add sudo nano /etc/network/interfaces and

```
auto eth0
iface eth0 inet static
  address 192.168.1.10
  netmask 255.255.255.0
  gateway 192.168.1.1
  dns-nameservers 8.8.8.8 8.8.4.4
```

DNS Configuration

The resolver configuration is usually set in /etc/resolv.conf:

```
nameserver 8.8.8.8
nameserver 8.8.4.4
```

> NOTE: If you use NetworkManager or another network management service, you should not edit this file directly, as it will get overwritten.

Restart the Network Service

To restart the network service or the entire networking sub-system:

```
sudo systemctl restart networking
```

Test Your Connection

Once configured, you can test the network connection using:

```
ping 8.8.8.8
```

```
ping www.google.com
```

If you can ping by IP but not by domain name, there is likely a DNS issue. If neither works, check your IP settings and routing.

If using Uncomplicated Firewall (UFW) as a firewall, these additional commands might be helpful:

```
sudo ufw allow ssh
```

Add any other protocols you wish, such as http or https.

Check the Status

Use this command to check the status.

```
sudo ufw status
```

Windows Desktop Configuration Using VirtualBox

IMPORTANT: Considerations when connecting a G3 module to a network.

Addressing security concerns when connecting any device to a network is crucial. Many IT departments forbid connecting devices not directly in their control. There are two options for dealing with this:

- Add a second network interface to the host PC (a USB dongle).
- Remove the PC from the main network and set up an isolated network (i.e., air-gapped from the main corporate network) for end-product application development.

No matter which option you choose, the development host and the G3 module must effectively have static IP addresses to allow communications for on-target debugging and migrating applications and associated data between the host and the G3 module without reconfiguring each time. You can do this by:

- Assigning the IP addresses manually.
- Configuring a DHCP server to use host reservations for the development host and the G3 module, which is equivalent to manually setting static IP addresses.

G3 Developer VM Settings for Windows

Adjusting some virtual machine settings allows you to use your development host hardware.

TIP: You can only adjust VM settings when the VM is stopped.

Before making any changes, complete these physical connections:

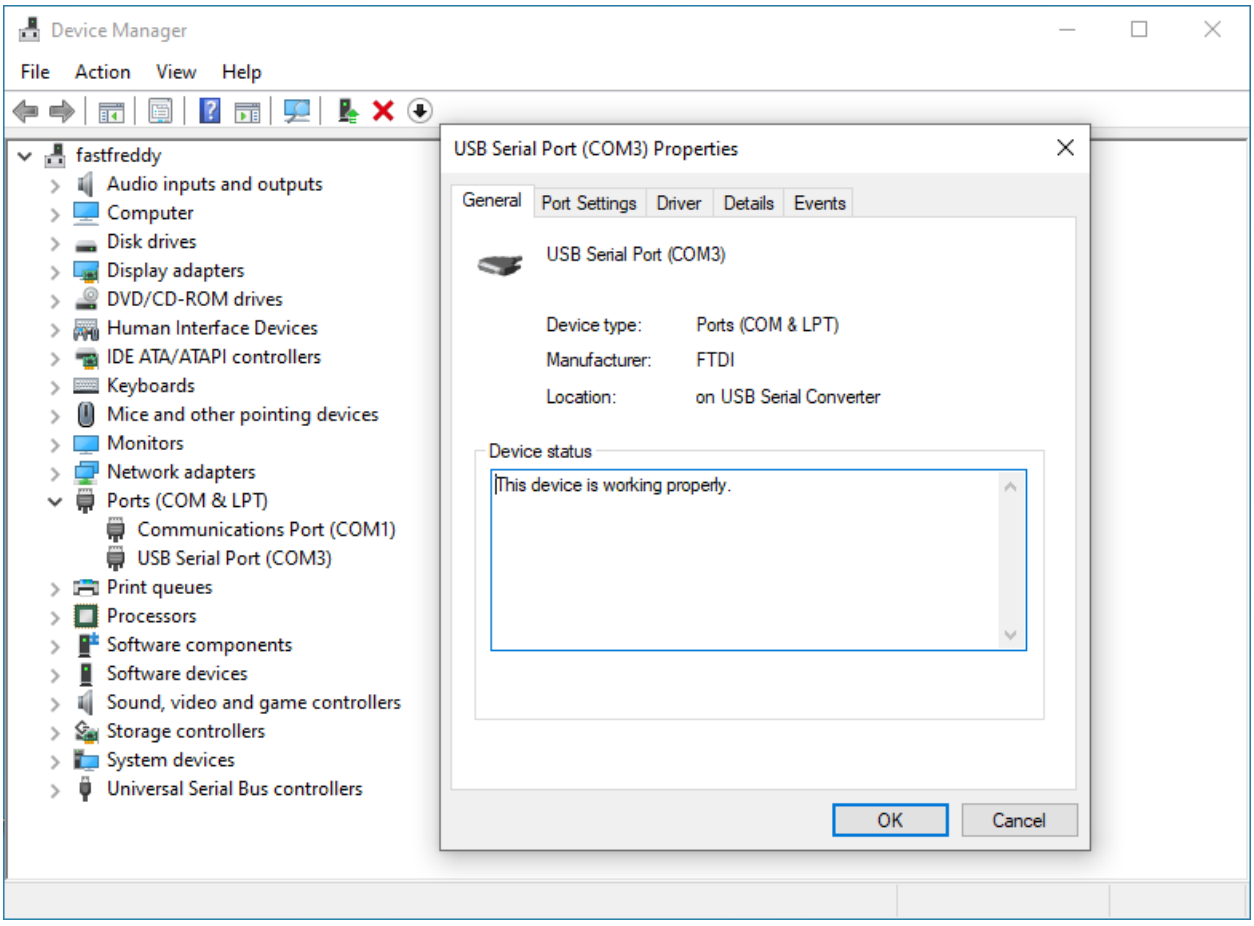
- Debug Interface Port.
- Ethernet Port (J3).
- “Y” power/RS-232 cable (J2), but **do not power up** the G3 module yet.

Settings You Must Change

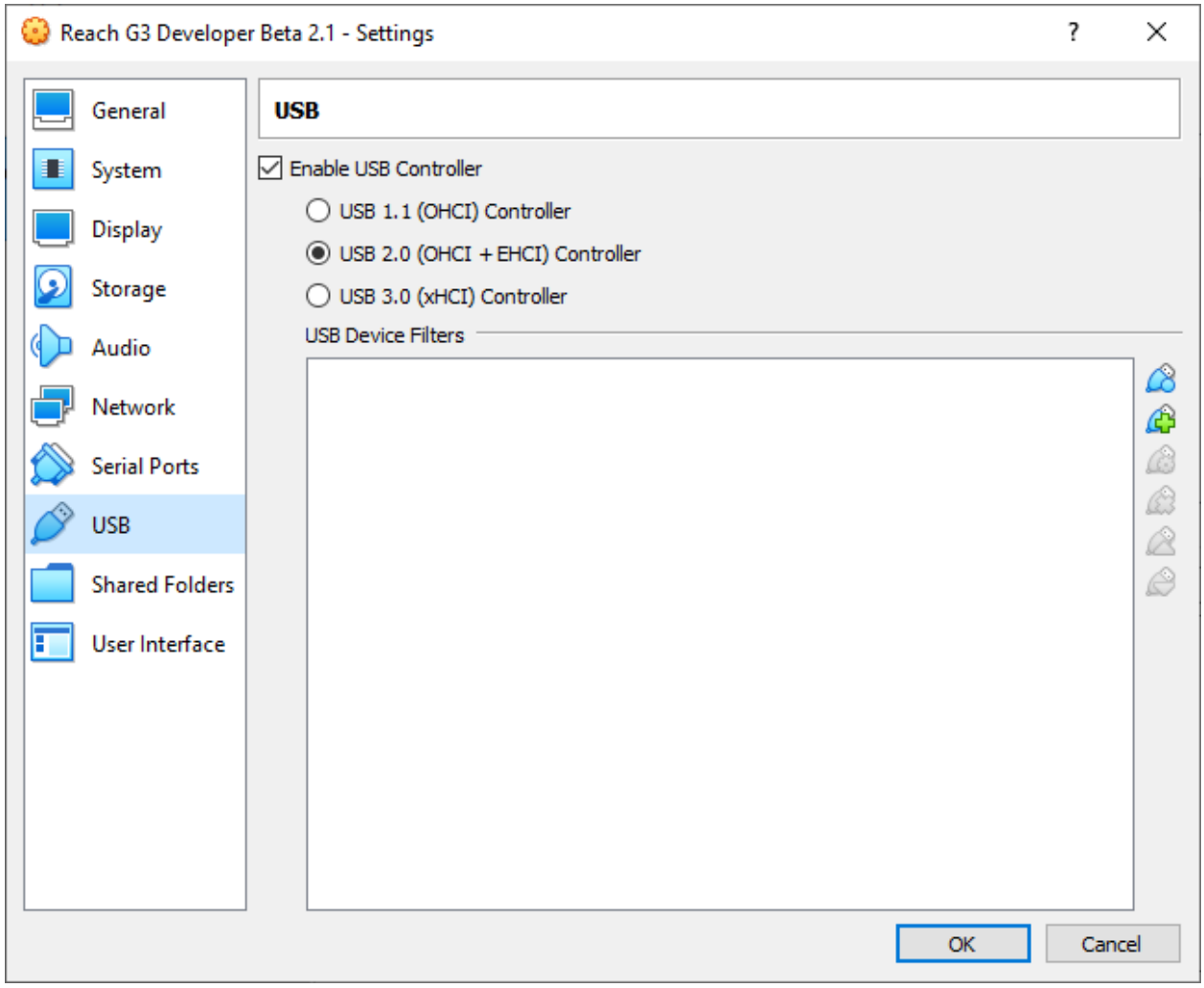
You **must** change these configuration settings, or the G3 Developer VM cannot communicate with and control the G3 module target device.

Serial Port Configuration

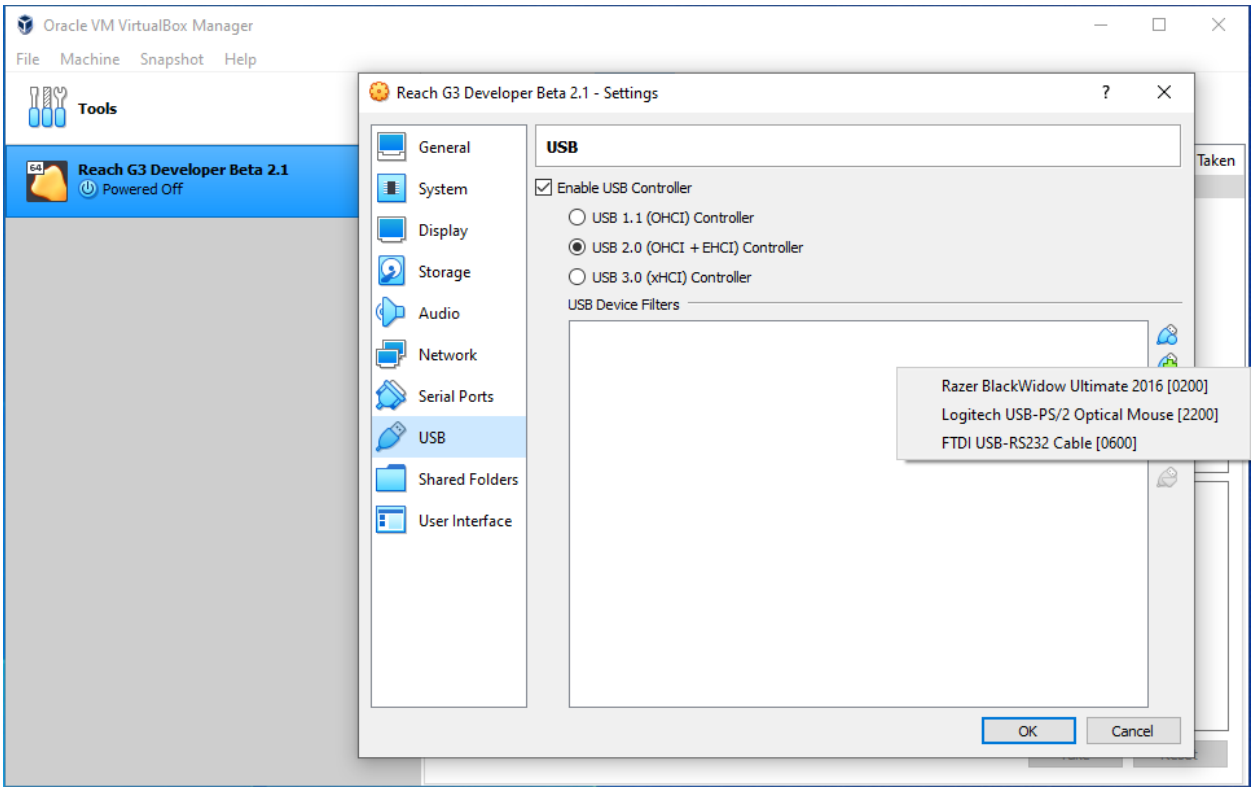
When you plug the USB connector from the Debug Interface Cable into your Windows 10/11 host, Windows will set up a new COM port for it. You may have to use the Device Manager to install the required driver the first time you do this. Search the network for the driver and pull the required one directly from Microsoft. When the driver is installed, your Device Manager should look something like this:



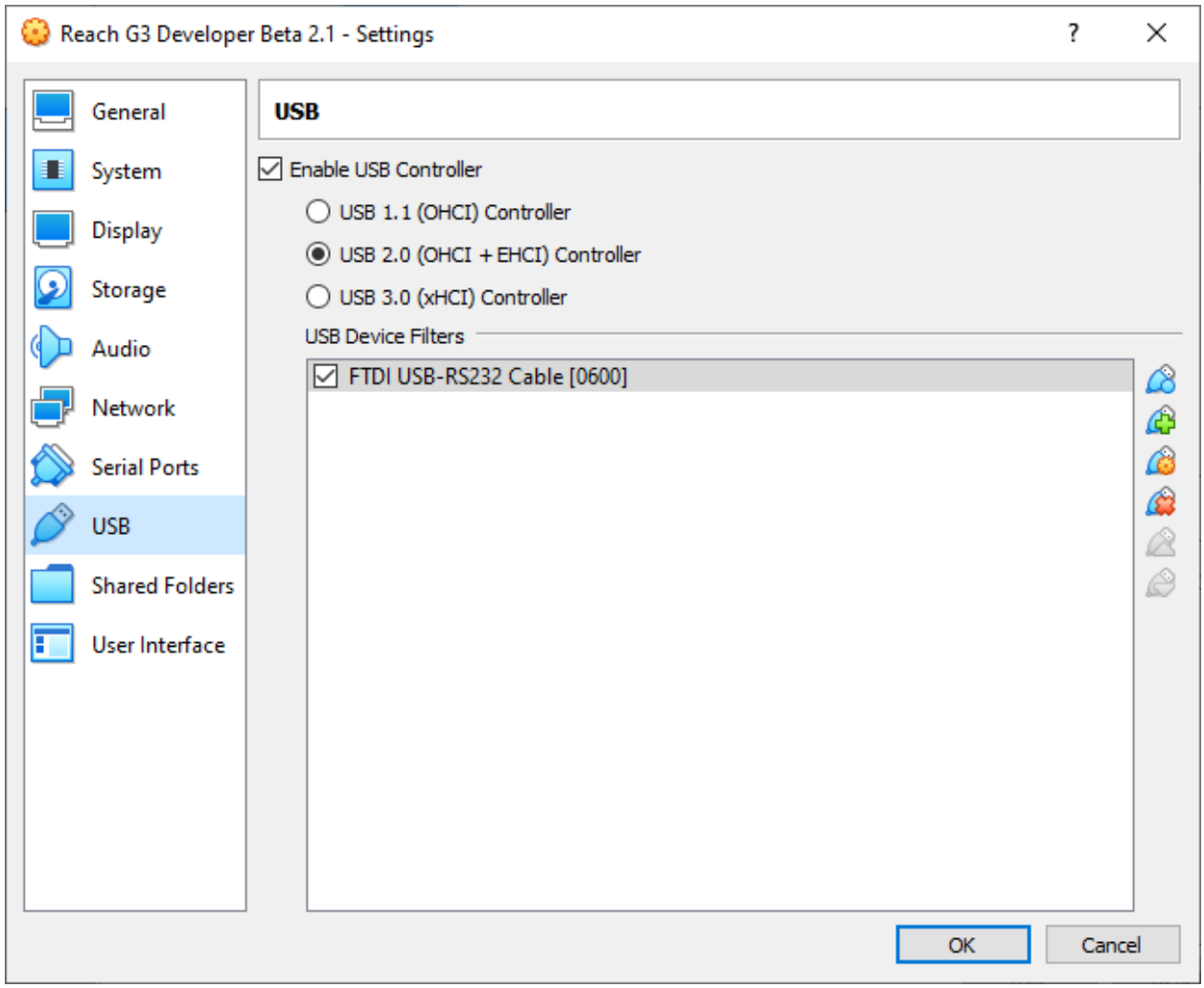
- Once this process is complete, open the VM settings dialog and switch to the USB panel. It should look something like this:



- Click the blue USB connector button with the **green +** on it to list the available USB devices as shown below:

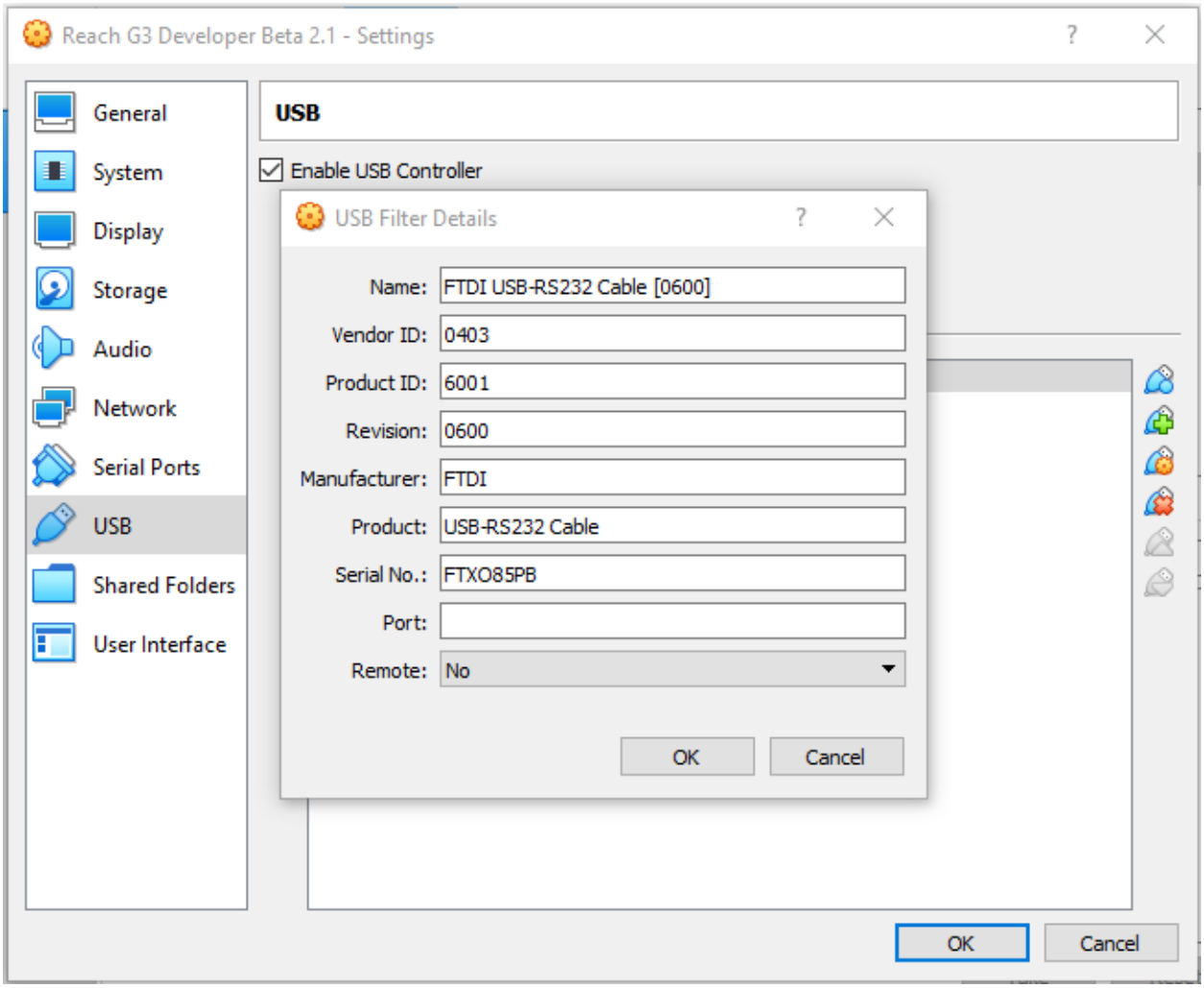


- Click the **FTDI USB-RS232 Cable [0600]** entry to add the filter for the Debug Interface Cable. The VM USB settings panel should now look like this:



TIP: Remove all USB to serial converters and insert only the Debug Cable to make adding the filter easy. You cannot see the converter serial number until you add the filter.

- With the filter entry selected in the list, you can click the blue USB connector button with the orange gear on it to popup the device details dialog as shown below:



TIP: The main USB settings panel does not display the device serial numbers. They show up in the details dialog. So, if you have several similar USB devices (e.g., multiple USB to serial converters), the entries in the filter list will all look the same. Thus, you cannot tell which is which on quick inspection.

When first adding the filter, change the filter's name on the details dialog for better identification in the main USB filter panel.

Testing

Serial Port Connection to the Target

Linux Desktop

Open a terminal window, then:

```
# list the available USB serial convertor ports
user@host:~$ ls -l /dev/ttyUSB*

# so, assuming the debug interface is on /dev/ttyUSB0...
user@host:~$ picocom -b 115200 /dev/ttyUSB0
```

You can now see any messages and log in to the target in your terminal window.

Power on the target. You should see several lines of output as the target boots. Then, you will see a login prompt. Enter *root*; there is no password.

If you do not see any output, try any other available /dev/ttyUSB* ports listed by the above ls command. If there is no response, see troubleshooting or seek assistance from a system administrator.

From the G3 Developer VM

Once you have configured the serial ports, launch the VM and log in as "Joe Developer." The password is *developer*. Now, open a terminal window in the VM, then:

```
# list the available USB serial convertor ports
joe@g3b1dev:~$ ls -l /dev/ttyUSB*

# list the available command aliases
joe@g3b1dev:~$ alias

# other aliases removed for clarity...
# these are the serial port access aliases
alias com0='picocom -b 115200 /dev/ttyUSB0'
alias com1='picocom -b 115200 /dev/ttyUSB1'
alias com2='picocom -b 115200 /dev/ttyUSB2'
alias com3='picocom -b 115200 /dev/ttyUSB3'

# so, assuming the debug interface is on /dev/ttyUSB0...
joe@g3b1dev:~$ com0
```

You can now see any messages and log in to the target in your terminal window.

Power on the target. You should see several lines of output as the target boots. Then, you will see a login prompt. Enter root; there is no password.

If you do not see any output, try any other available /dev/ttyUSB* ports listed by the above ls command. If there is no response, see troubleshooting or seek assistance from a system administrator.

Ethernet Connection to the Target

This procedure applies to either the Linux desktop or within the G3 Developer VM. Start by opening a terminal window, then:

```
# test network reachability of the target at IP address <n.n.n.n>
joe@g3b1dev:~$ ping <n.n.n.n>

# test network reachability of the target assuming /etc/fstab
# has been setup correctly
joe@g3b1dev:~$ ping g3b1
```

If there is no response, see troubleshooting or seek assistance from a system administrator.

Usage

Qt Application Development

If you are unfamiliar with Qt Creator or want a quick overview, here is an official link from Qt and a video on installing and getting started. There are many third-party sites and videos available to help you. We recommend KDAB, a video series that is informative on a wide variety of Qt topics.

TIP: The main USB settings panel does not display the device serial numbers. They show up in the details dialog. So, if you have several similar USB devices (e.g., multiple USB to serial converters), the entries in the filter list will all look the same. Thus, you cannot tell which is which on quick inspection.

When first adding the filter, change the filter's name on the details dialog for better identification in the main USB filter panel.

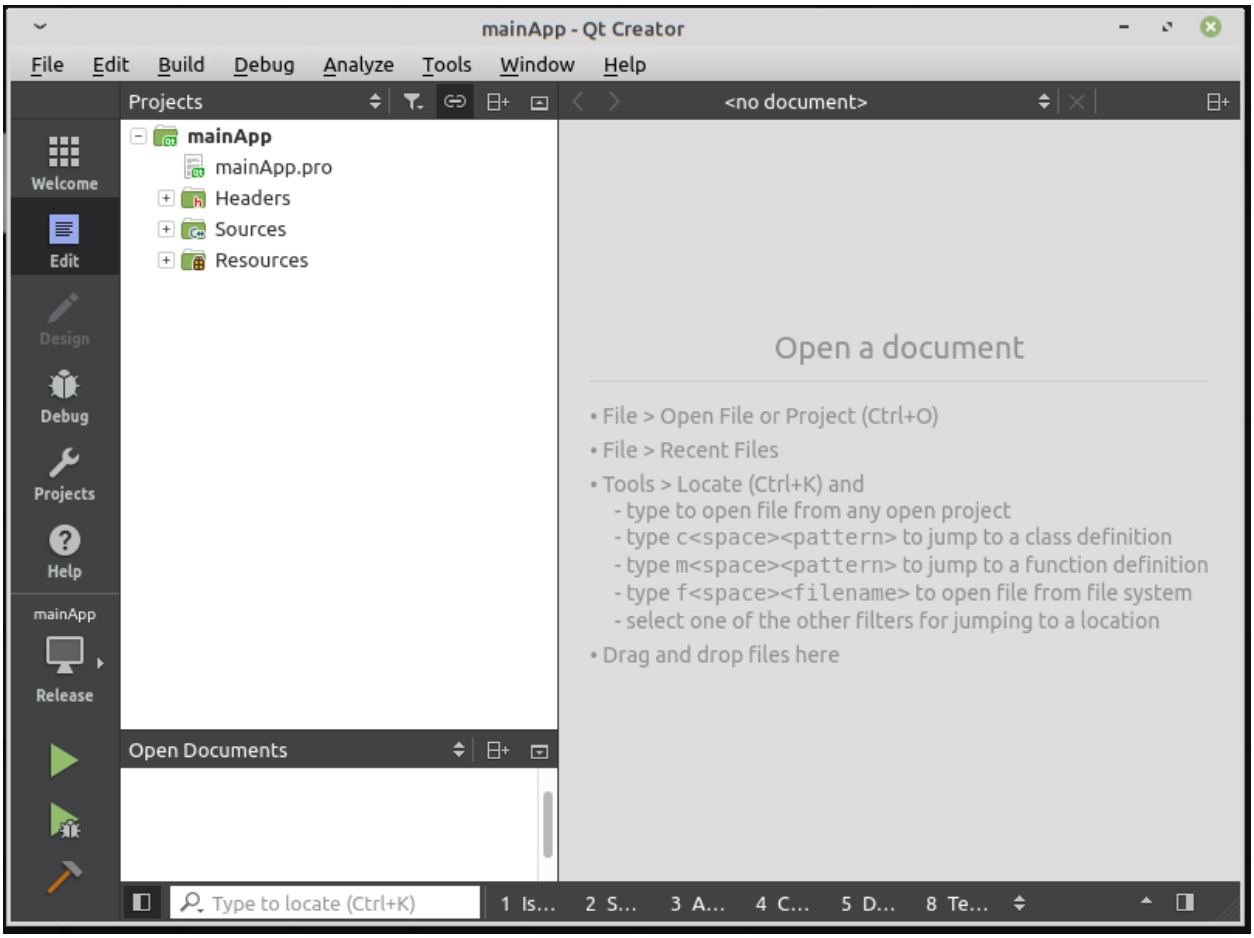
> NOTE: You will need your development environment set up and functioning to continue:
>
> G3 Developer VM or the G3 Developer SDK has been installed and tested.
> Qt Creator is configured to develop and test applications. If not, see configuration instructions for Qt Creator on a Linux host, or configuration instructions for Qt Creator in the G3 Developer VM.
> Serial and network connections from host to target are functioning correctly.

Fire up Qt Creator to get started. First, download the code to the target.

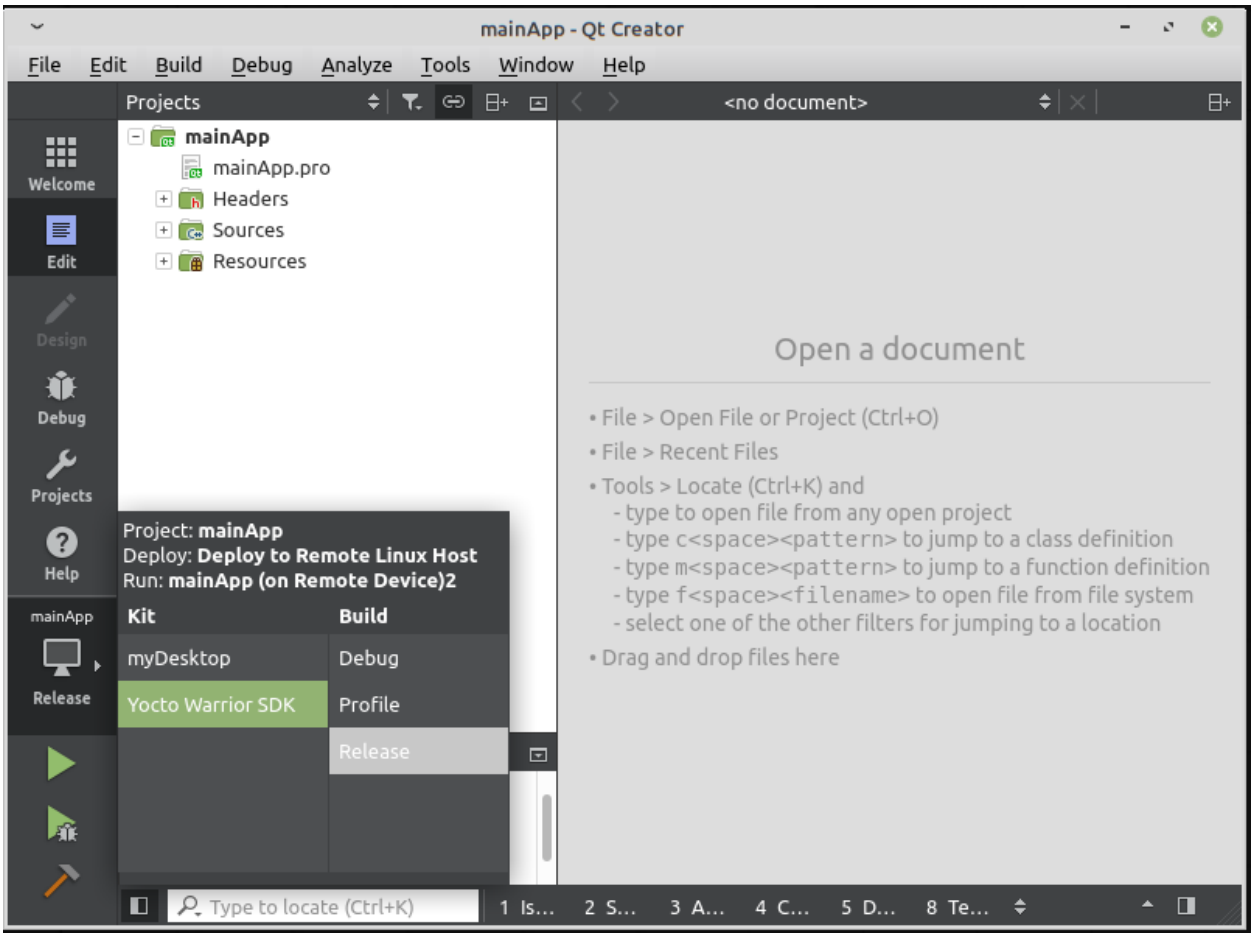
As the Quick Start Guide noted, a few sample applications are already in the VM. Most of them are available in the Reach Technology GitHub Repository for download.

```
# to clone our sample applications repo in your VM
joe@reach-qt5-nxp-dev-vm:~$ git clone https://github.com/jmore-reachtech/reach-g3-qt5-sample-apps.git
```

- Open **mainApp** from the sample projects provided. Start with **File** -> "Open File or Project," then find the sample projects in /data/app on the VM. This application displays system resources and should look something like this.



- Select the mainApp deploy/run box as shown below. Ensure the configured Yocto SDK kit[9] and **Release** are selected.



- To run the application on the target, click the **green play arrow** (the play arrow with a bug just below it will run the debugger).

> TIP: Ensure no other applications are running by running the command /etc/init.d/user_app stop from a shell on the target.
>
> NOTE: If you see the warning, "The build directory needs to be at the same level as the source directory," go to **Projects** -> **Build and Run** -> **Yocto Warrior SDK** -> **Build** (left pane) and look at the **Build directory** (under General). Qt Creator wants “Source” and “Build” directories in the same parent directory.

[9] In this example, the Yocto SDK configured in Qt Creator is called, “Yocto Warrior SDK, “ yours will probably be different.

- After clicking the **play arrow**, Qt Creator will compile and download the code to the target device.
- You should see the **mainApp** running on the target. On Qt Creator, click on **Edit** on the left side panel.

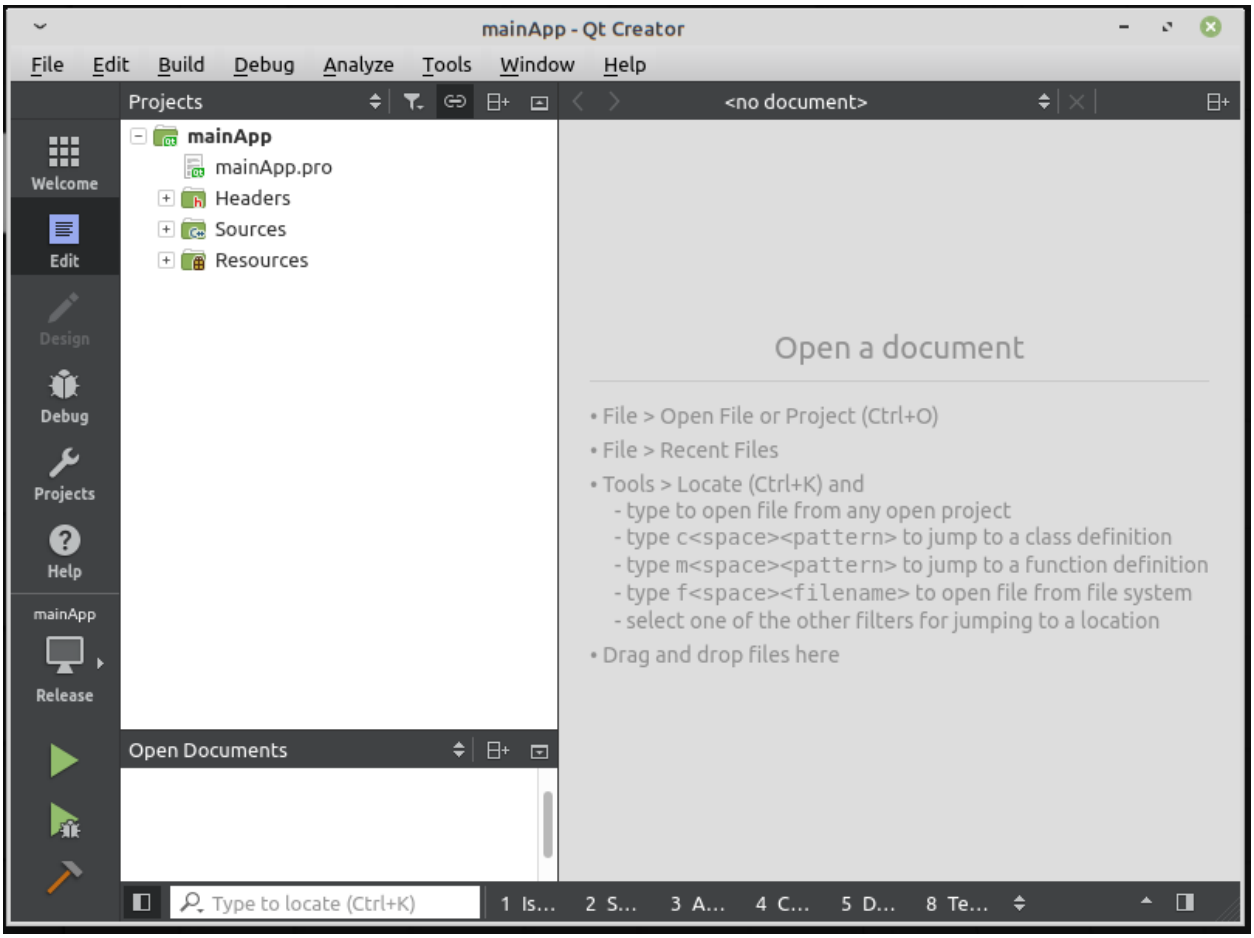
Now, you should be up and running.

Sample Qt Application – mainApp

NOTE: Download updated developer images.

Load The Sample Project- mainApp

The screen should look like this.



Let us look at the project structure. First, the Headers and Sources directories contain sub-directories with /data/ app/common roots. These sub-directories contain common *.cpp* and *.h* files that contain hardware-related code and definitions, drivers of a sort for the G3 module hardware.

There are two files of note inside the Sources directory:

- myGlobal.cpp
- myStyle.cpp

These two files each create a QQmlPropertyMap.

For an overview of property maps, see "A Quick Description of Qt Property Maps" in the appendix. The sample code uses the property map functionality to change the values you will display in the UI.

To send serial data from the UI or C++ code out over the console port, call the submitTextField() function in QML or use the serial.write() command in C++ code.

As such, they are global (spanning C++ and QML) and hold all system variables needed for external access. These property maps store key/value pairs displayed (i.e., temperature, RPM) or affect the display (i.e., display color for a text box, text size, background color).

Each key is a unique, user-defined name. Values can be sent over a communications interface, triggering automatic display updates.

For example, using the serial interface, if you send:

themeDark=true

Then, the background and text colors change to the dark theme color set.

To test this approach from a console-only perspective[10]:

- In the G3 Developer VM runs mainApp from Qt Creator.
- Open a terminal and run picocom for the Debug Interface. Using the shell alias, the command line is com0.
- Open another terminal. We will use this terminal to pipe test stimulus data to the RS-232 device.
- In the second terminal window (in the home directory), type cat main.txt > /dev/ttyUSB1.
- Now type cat main2.txt > /dev/ttyUSB1. Watch the screen colors change.
- Repeat the cat main.txt > /dev/ttyUSB1 command. Repeat as you wish. From here, explore the capabilities of *myGlobal* and *myStyle*.

All Reach Technology sample Qt applications use this Property Map paradigm.

[10] In this example, we assume:

- The Debug Interface is accessed from the G3 Developer VM via /dev/ttyUSB0; on target this port is /dev/ttymxc0.
- The RS-232 port on the Y-type cable is accessed from the G3 Developer VM via /dev/ttyUSB1; on target, this port is /dev/ttymxc1.

Software Manual

Theory of Operation

WARNING: Your product application should never run as the root user. Instead, create a specific user and group and grant necessary access to the required system resources.

The Init System

G3 modules use the traditional System V Init. Although time-tested and reliable, it uses a serial process, so boot times are not instant, and it does not handle service dependencies well. To ensure services start in the proper order, system administrators need to be vigilant when adding them.

The System V Init system employs multiple run levels, in contrast to Windows, which offers only a choice between maintenance/repair mode or a comprehensive alternative. Linux breaks the system state into several functional levels, significantly simplifying finding the root causes of service failures. The System V Init system also controls how system components or services are launched[11] via a series of "rc" directories in /etc. Each of these directories contains a series of symbolic links that start with either **S** (start service) or **K** (stop service), followed by a two-digit number and the name of the init script it links to. Find init scripts in /etc/init.d. The table below lists the various run levels, the "rc" directory and their purposes[12].

Directory	**Description**
rcS.d	**Sysinit** run level for initial boot items.
rc0.d	**Halt** run level for shutdown services and halt the processor[13].
rc1.d	**Single User** run level allows root only to log in on the console.
rc2.d	**Multi-User** run level allows users to log in on local serial connections.
rc3.d	**Multi-User with Networking** run level allows multi-users to log in with networking turned on.

[11] Use care when changing the order in which services are started or stopped, as this will likely cause errors and/or service/application failures. If you change the order of start/stop of optional services controlled by chkconfig, you need to update the comment block at the top of the init script to reflect your changes.
[12] Several “modern” Linux distros have deprecated the multiple run level approach as most desktop users are more familiar with the Windows approach. Reach Technology believes significant value is added to embedded systems with the flexibility offered by traditional init system run levels.
[13] For systems capable of software power control, the system will also power down.

Directory (continued)	Description (continued)
rc4.d	**Multi-User Networking**, **Services** run level, allows you to add services with a remote login over the network.
rc5.d	**Multi-User, Networking, Services, and Applications** run level allows you to add applications.
rc6.d	**Reboot** run level for shutdown services and warm processor reboots.

As discussed in more detail in the optional services configuration section, you can easily control many optional services using the chkconfig utility. The tool manipulates the service's **S** and **K** symlinks using information in a specially formatted comment block near the top of the associated init script.

Read-only Root Filesystem

While a few desktop Linux systems and larger servers may mount the root filesystem (commonly called the "rootfs") read-only for security reasons, most of these systems mount it writeable as a matter of convenience. These systems share the ability of qualified system administration resources to regularly access, monitor, and update the rootfs and the various software packages on the system.

System reliability and tampering resistance become overriding considerations since you typically deploy the G3 module as one component of your end project in an office environment. As with any Linux system, there are scenarios in which the rootfs could be irrecoverably damaged or tampered with if you deploy a G3 module with writeable rootfs. Reach Technology has engineered the production of embedded software images with read-only rootfs to help prevent these scenarios. See the implications discussed below.

There are four areas of the rootfs which commonly require regular write access:

- Configuration files in */etc*
- Runtime state files in */var*
- Temporary files in */tmp*
- Items in the root user home directory (commonly */root or /home/root*)

To support writing to areas where rootfs are mounted read-only[14], you can use various techniques:

- /etc is mounted using the OverlayFS filesystem. The /data/etc directory is the "upperdir" while the /etc in the rootfs partition is the "lowerdir". Access this stack via the standard /etc path. The result is /etc appears to be writeable; however, any changes write to /data/etc.

[14] Refer to the next section on the layout of the /data filesystem for additional information.

This step is critical to support over-the-air (OTA) updates to the rootfs! It prevents "soft" bricking of your product, which can occur when you perform a rootfs update that loses configuration data for either your end product application or the system itself. A typical example would be losing network configuration data so your product cannot access needed resources or offer services (e.g., a web UI or RESTful API).

The other advantage is you can use Mender to update your application in /data without disturbing the rootfs. This setup makes it possible to recover from a bad end product application update without resorting to A/B copies of the /data filesystem, making more of the fixed size eMMC available to your application. A/B copies of the rootfs are required (and are in place) to facilitate fully recoverable updates to the rootfs.

- /var is bind mounted to /data/var. Refer to the mount manpage for more information on bind mounts, a more straightforward and faster technique than using the OverlayFS filesystem. The net effect is essentially the same; anything written to /var shows up in /data/var. It is not an appropriate solution for /etc for several reasons. Most importantly, the init system must access some configuration files before the OverlayFS filesystem mount takes place.

- On desktop systems, /tmp is usually kept separate from /var/tmp, and they have different semantics. Traditionally, /tmp is strictly for non-persistent data, while /var/tmp is for temporary files that should survive a crash or reboot until explicitly deleted. Most Linux-embedded systems have merged the two locations by making /tmp a symbolic link to /var/tmp. Further, several sub-directories of /var, including /var/tmp, are relocated at boot time to a tmpfs (in memory) filesystem mounted at /var/volatile. Because of this, neither the contents of /tmp nor the contents of /var/tmp will survive a crash or reboot.

The other important ramification is that it is possible to cause an out-of-memory (OOM) kernel issue by writing temporary files without controlling the expansion of the tempfs in system RAM, as you have several things competing for available RAM:

  - Kernel
  - Buffer cache
  - /var/volatile tempfs
  - Your application
  - Other system services and applications

- We relocated the /etc/root to take advantage of the OverlayFS filesystem. Any writes to that directory will show up in /data/etc/root.

IMPORTANT: The product application architect must install all components in /data. It is possible to revert to a read-write rootfs, whoever Reach Technology *strongly* discourages you from doing this. If you absolutely must do this, system updates with Mender are not supported.

The /data File System Layout

IMPORTANT: All files related to your product application must be in /data.

Production Layout

The Big Picture

/data is the only persistent writeable filesystem on the G3 module. G3 modules use the Linux Filesystem Standard as a model to provide a recognizable structure to /data. Some directories in the table below (notably /data/etc) may contain several service-specific sub-directories.

/data Production Sub-directories

Directory	Contents
bin	Product application executables and scripts.
etc	System and product application configuration data.
lib	Product application shared libraries.
lost+found	Find recovered bits of corrupted files here.
mender	Mender safely applies updates to the system.
sbin	Product application management, configuration, and diagnostic tools.
share	Product application data files.
share/audio	Product application sound files.
share/fonts	Product application font files.
share/pixmaps[15]	Product application image files.
share/video	Product application video files.
u-boot	Boot loader configuration. It also contains the tw_env.config file.
var	Access various system state files via */var*.
var/tmp	Product application temporary files and Unix domain socket endpoints.

[15] The Linux boot splash image, splash.bmp is located in /data/share/pixmaps.

System Configuration Files

Linux desktop systems and servers place most configuration files in or under /etc. These must be writeable during product development. However, they do not need modification when the product deploys. Because G3 modules use a field-deployed embedded Linux system, the configuration files must not be vulnerable to accidental or malicious modifications that could leave the deployed product non-functional.

G3 modules are field upgradeable via over-the-air or local updates deployed on a USB flash drive. As mentioned above, the root filesystem is mounted read-only to prevent accidental updates of system configuration files that could render the product unusable. See the embedded image upgrade section for a more detailed discussion of the upgrade capability.

The OverlayFS filesystem mount offers the best of both worlds. There is an immutable copy of the original configuration file in the "lowerdir" (i.e., the actual /etc in the rootfs partition). Changes to the file show up as a new file of the same name in /data/etc, taking precedence if it exists.

So, the original configuration file cannot be deleted unless:

1. The overlay is unmounted (non-trivial but doable under controlled circumstances).
2. The rootfs is remounted read-write.

It is still possible to maliciously obscure a configuration file by writing something new that ends up in /data/etc and would require root-level access.

> IMPORTANT: This is one reason your product application should **never run as root,** and you should set a non-trivial root password.

For additional system security, you can implement a script hat periodically examines /data/etc and compares the contents (e.g., SHA256 sums) against a master list and deletes anything that does not match.

Pre-Production Layout

The /data directory on the G3 module stores the product application and associated data.

> NOTE: The equivalent /data directory in the developer VM also exists. It is exported by the VM using NFS so it can be mounted on the development host to enable easy copying of data and executable code between the host and the VM.

The G3 module /data directory currently contains four sub-directories, described in the table below.

Directory	Contents
app	Directory for applications and data.
lost+found	Find recovered bits of corrupted files here.
mender	Mender safely applies updates to the system.
u-boot	Boot loader configuration. It also contains the fw_env.config file.

The file splash.bmp is displayed while the Linux init system starts up services. Change this to whatever you wish for end-product branding. The image must be in 24-bit BMP format.

Product Application Launch

The last action of the G3 module Init system is to launch the product application. The /etc/init.d/user_app script first attempts to start any installed Reach Technology demos.

It will look for and run the application launcher script if the demo launch scripts are missing. In other words, the run_demos script takes precedence over the run_application script, so you should remove run_demos when setting up the G3 module to run your application. Find the launch scripts in /data/sbin.

Script	Description
run_demos	Launch Reach Technology demos (supplied by Reach Technology)
run_application	Launch your product application (supplied by you)

IMPORTANT: Since the rootfs is read-only, you **should avoid modifying** /etc/init.d/user_app. Instead, create your own /data/sbin/run_application script.

Installation

This section covers installing an embedded Linux image on the G3 module.

Console (Debug) Port

Access the controller Debug Interface via the connector table below using the Reach Technology debug cable (Part Number: 23-0161-72).

Debug Interface Connectors

Controller Board	Debug Interface Connector
G3BNG	J1
G3MSB	J3
G3MSG	J3

This RS-232 level serial port runs at 115.2k Baud, 8-bit, no parity, and 1 stop bit. During product development, this port **always** needs to be connected. Use a terminal emulator program or shell that understands ANSI color escape sequences. Examples:

- For Windows, Tera Term or PuTTY[16]
- For Linux, Picocom in a bash shell.

IMPORTANT: If you are using Windows 10 as your development host (with or without the G3 Developer VM) the most common problem you will run into is non-deterministic enumeration of USB->serial converters (like the Reach Technology Debug Interface Cable). You will see the Windows COM port number changes, and the path to the device special file in the VM (/dev/ttyUSB<n>) will also change, making it challenging to use a single config file for your terminal emulation program to access the console.

This is especially problematic when flashing and booting a new SD card or eMMC image for the first time. The display will not illuminate when the u-boot Device Tree Blob (DTB) variable is not set.

If you do not see messages scrolling on the Debug Interface during boot, it is most likely due to one of the following:

- You are connected to the wrong COM port, or the COM port parameters are incorrect.
- The eMMC boot jumper is not in, and no SD card is in the socket.
- The eMMC boot jumper is in, but the eMMC is not flashed with a valid image, and no SD card is inserted.

[16] We recommend PuTTY because it supports both serial and network connections and runs on Windows 10/11 and Linux.

Boot Setup

Boot the G3 module from one of three sources:

- SD card
- eMMC (flash)
- USB OTC[17]

NOTE: Select version 5.15.2, the supported version on G3, when loading Qt. The first time an eMMC or SD card image boots, the system automatically extends the data partition to use all available space. In eMMC, this will result in the /data filesystem being approximately 4.5GB. On SD cards, the final size depends on the SD card's size.

The expansion is a two-step process:

- The data partition[1] expands on the first boot. Once complete, the G3 module automatically reboots to ensure the kernel recognizes the larger partition.
- The /data filesystem resizes after the reboot to fill the expanded partition. Once complete, the G3 module continues a normal boot sequence until it displays the login prompt.

IMPORTANT: The G3 module includes several optional services enabled by default, including the sshd secure shell server. The first time a new eMMC or SD card image fully boots, the init script for sshd generates a unique set of host keys in several formats, which takes some time. More importantly, you must ensure the new key files have been flushed to the SD card before halting or rebooting the G3 module. The simplest way to ensure this is to log in as root at the prompt and issue the sync command.

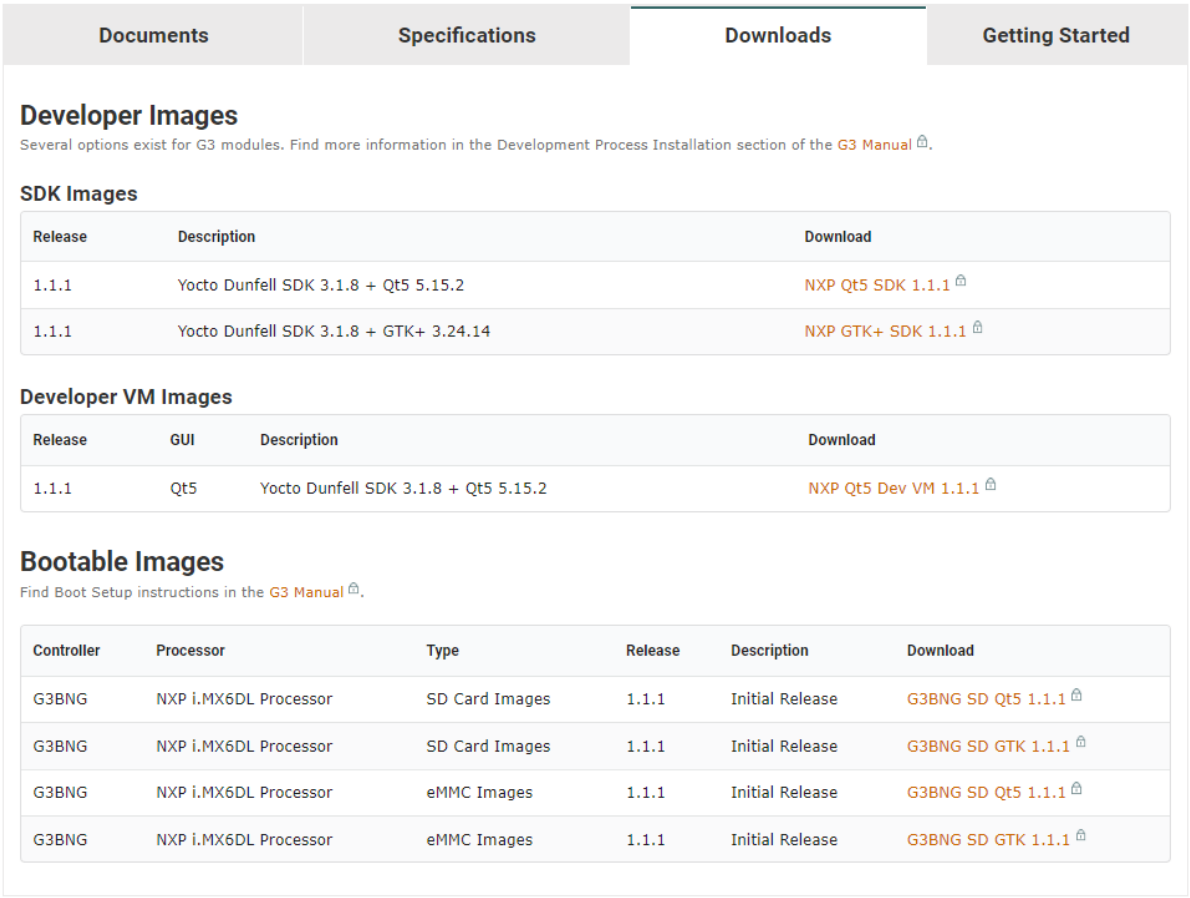
From a strict system reliability perspective, we recommend using the eMMC as the boot device for field-deployed systems. However, using the SD card will be simpler to manage during product development.

In some cases, a hybrid solution may make sense. For this type of system, you would use eMMC for the OS and the SD card for the product application.

[17] Booting over USB is not available on all G3 module variants. Booting over USB is impractical for most field applications so it is not documented here. Contact Reach Technology Technical Support for more information.

Bootable Images

To download the image you need, find your product at Compare Modules, and go to its page. Scroll dow to the **Downloads** tab (see an example below), and choose the appropriate image for your setup.



Documents | Specifications | Downloads | Getting Started

Developer Images

Several options exist for G3 modules. Find more information in the Development Process Installation section of the G3 Manual.

SDK Images

Release	Description	Download
1.1.1	Yocto Dunfell SDK 3.1.8 + Qt5 5.15.2	NXP Qt5 SDK 1.1.1
1.1.1	Yocto Dunfell SDK 3.1.8 + GTK+ 3.24.14	NXP GTK+ SDK 1.1.1

Developer VM Images

Release	GUI	Description	Download
1.1.1	Qt5	Yocto Dunfell SDK 3.1.8 + Qt5 5.15.2	NXP Qt5 Dev VM 1.1.1

Bootable Images

Find Boot Setup instructions in the G3 Manual.

Controller	Processor	Type	Release	Description	Download
G3BNG	NXP i.MX6DL Processor	SD Card Images	1.1.1	Initial Release	G3BNG SD Qt5 1.1.1
G3BNG	NXP i.MX6DL Processor	SD Card Images	1.1.1	Initial Release	G3BNG SD GTK 1.1.1
G3BNG	NXP i.MX6DL Processor	eMMC Images	1.1.1	Initial Release	G3BNG SD Qt5 1.1.1
G3BNG	NXP i.MX6DL Processor	eMMC Images	1.1.1	Initial Release	G3BNG SD GTK 1.1.1

SD Card Boot

Using an SD card is the default setup. Using a Reach Technology Development Kit, you will not need to flash an SD card unless Reach Technology provides a link for an updated image.

> IMPORTANT: Be sure to use a quality SD card that is 8 GB or larger.
>
> TIP: Various SD card images are available for download here.

Once you have downloaded the compressed image from the website, follow these steps[18] to flash the SD card for the G3 module:

```
# to generate the hash of the downloaded file to compare to the value on the website
> md5sum <image-file-name.bz>

# to decompress the image file
> bunzip2 <image-file-name.bz>

# to flash the image on the SD card
> dd if=<image-file-name.sdimg> of=<sd-card-dev> bs=4M conv=fdatasync status=progress
```

> WARNING: Pay close attention when typing in the value of <sd-card-dev>, your SD card writer's root device special file (e.g., /dev/sdh). Note there should be no partition number. If you accidentally enter the device special file for your host hard drive, you will damage the host OS and need to reinstall it.

Once you have a flashed SD card, insert it into the holder on the G3 module, make sure the boot jumper block has no jumpers installed, and power up the board. Assuming you have the Debug Interface Cable hooked up and are connected via a terminal application such as Picocom (on Linux) or Tera Term (on Windows), you should be able to watch the G3 module boot.

Alternative: Flashing an SD card on Windows 10/11

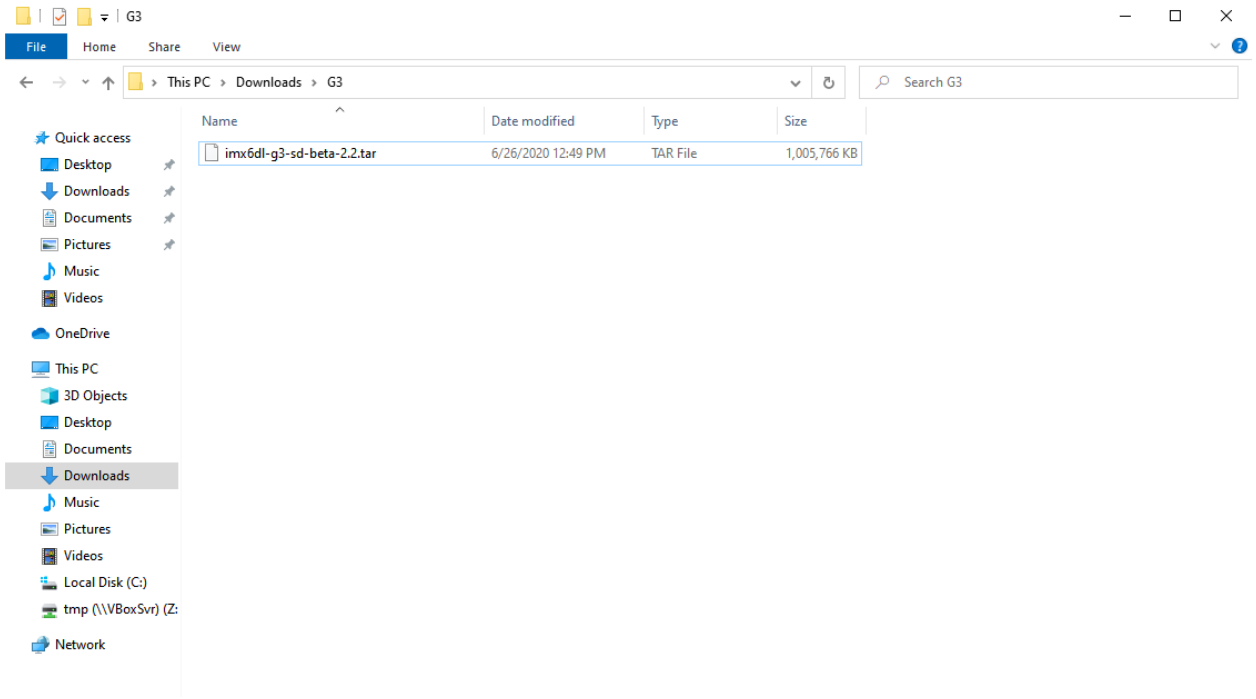
We do not recommend this approach as it is easy to set up incorrectly[19] on Windows 10/11. Here are the steps to burn an embedded image file to an SD card on Windows 10/11. These steps require two Windows utility programs to be downloaded and installed on your Windows 10/11 system:

- 7 Zip
- Win32 Disk Imager

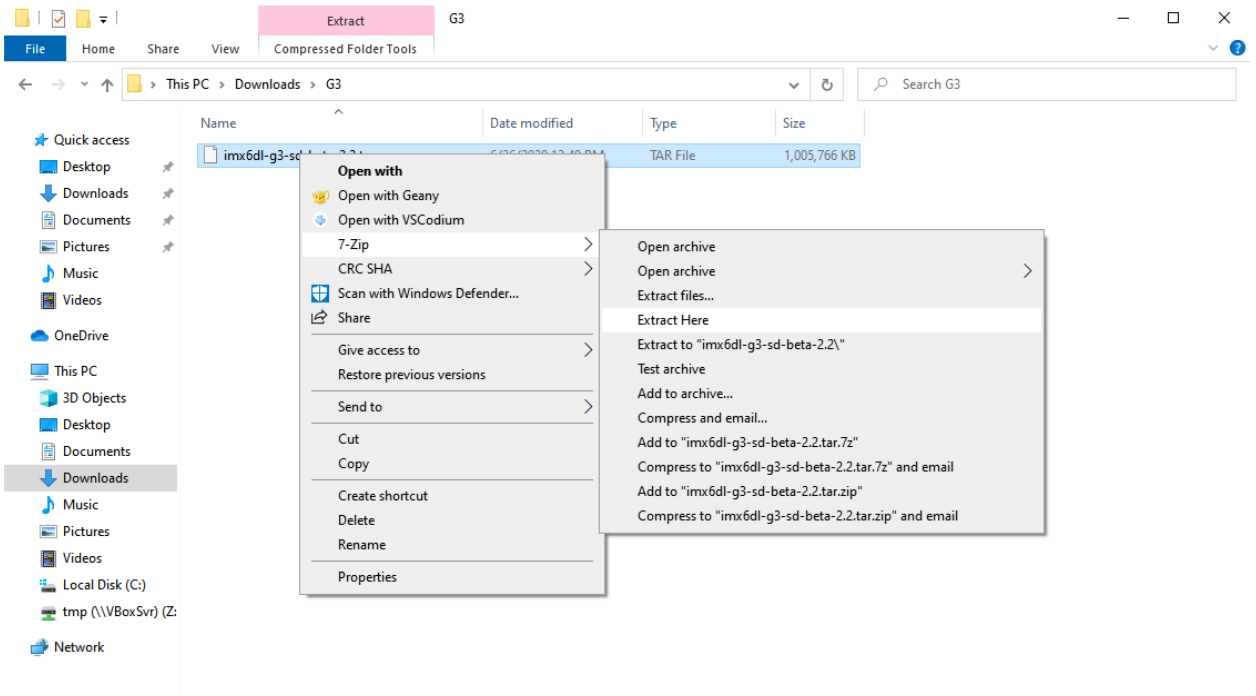
[18] These steps assume you are using a Linux host (e.g., Ubuntu or Linux Mint 20 XFCE) to flash the SD card. If you are using a Windowns host computer, you'll need an appropriate application to flash the card.
[19] The most common error is to select the wrong target device. This often occurs when the SD card has multiple partitions with filesystems readable by Windows 10/11.

Once you install the required utilities, proceed as follows.

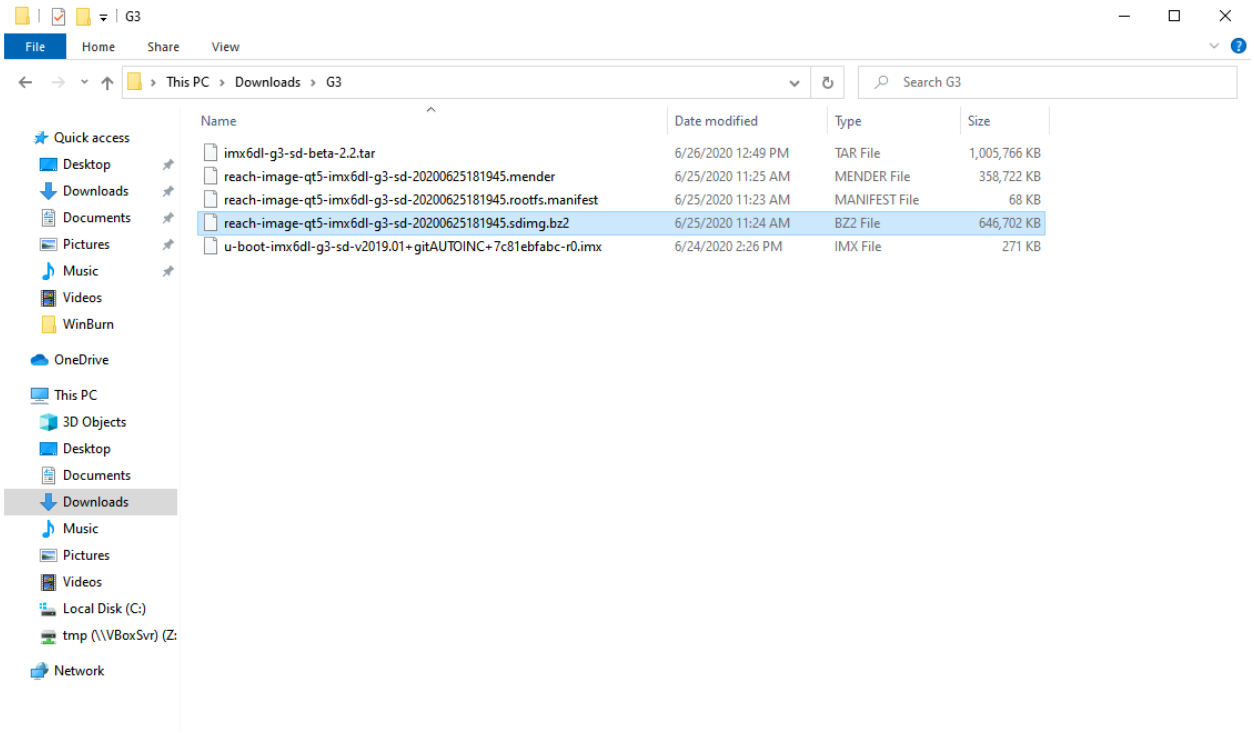
- In Windows Explorer, select the tarball containing the G3 module release, as shown in the figure below:



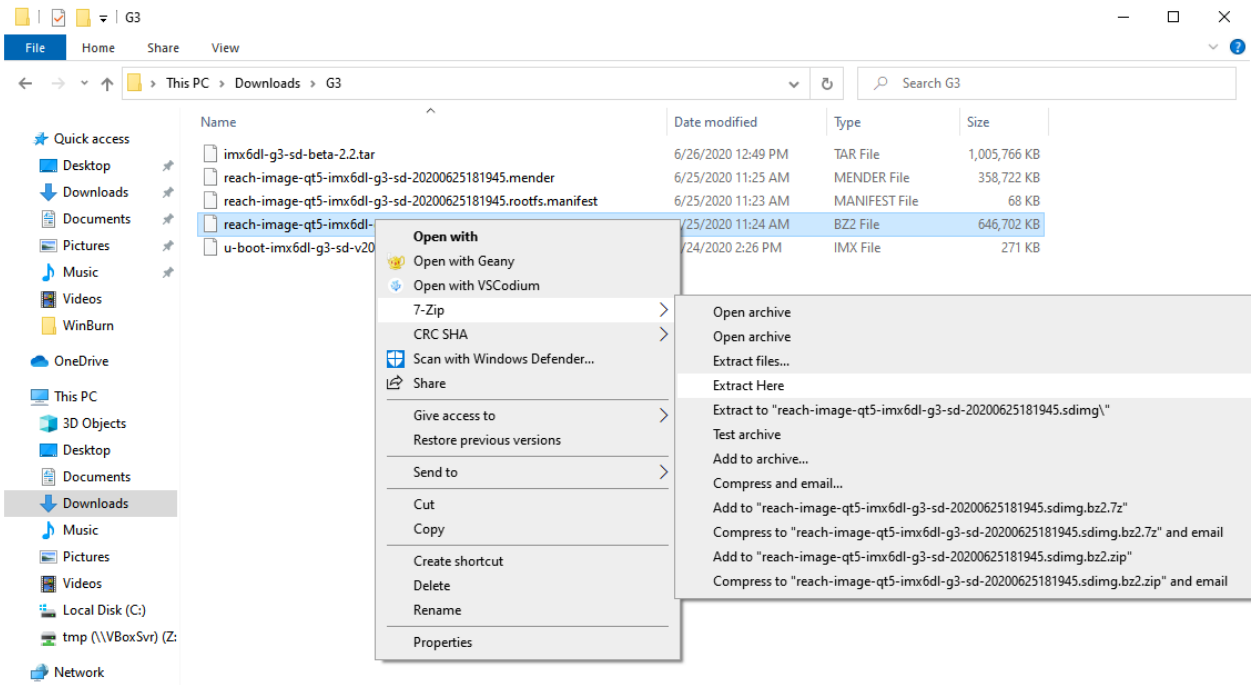
- Right-click on the tarball file, then select **Extract Here** under the **7-Zip** section of the context menu as in the figure below:



This step will extract four files from the tarball, including the bzip2 compressed embedded image. Select the compressed image, as shown in the figure below:



- Right-click on the bz2 file, then select **Extract Here** under the **7-Zip** section of the context menu as in the figure below:

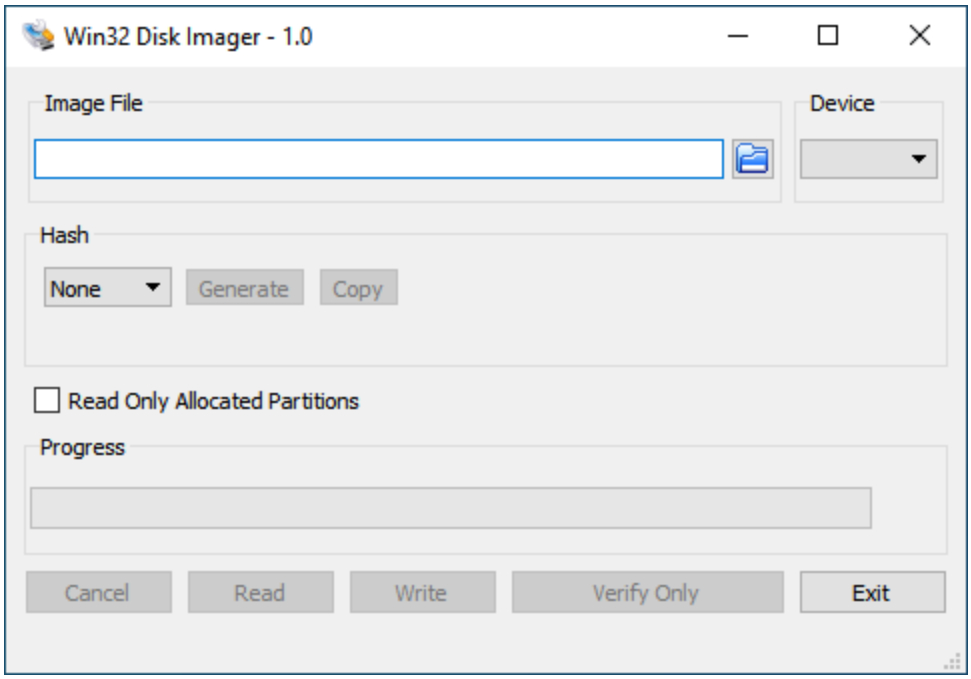


This will add a new and much larger file to the directory with the same name as the compressed image file minus the ".bz2" extension.

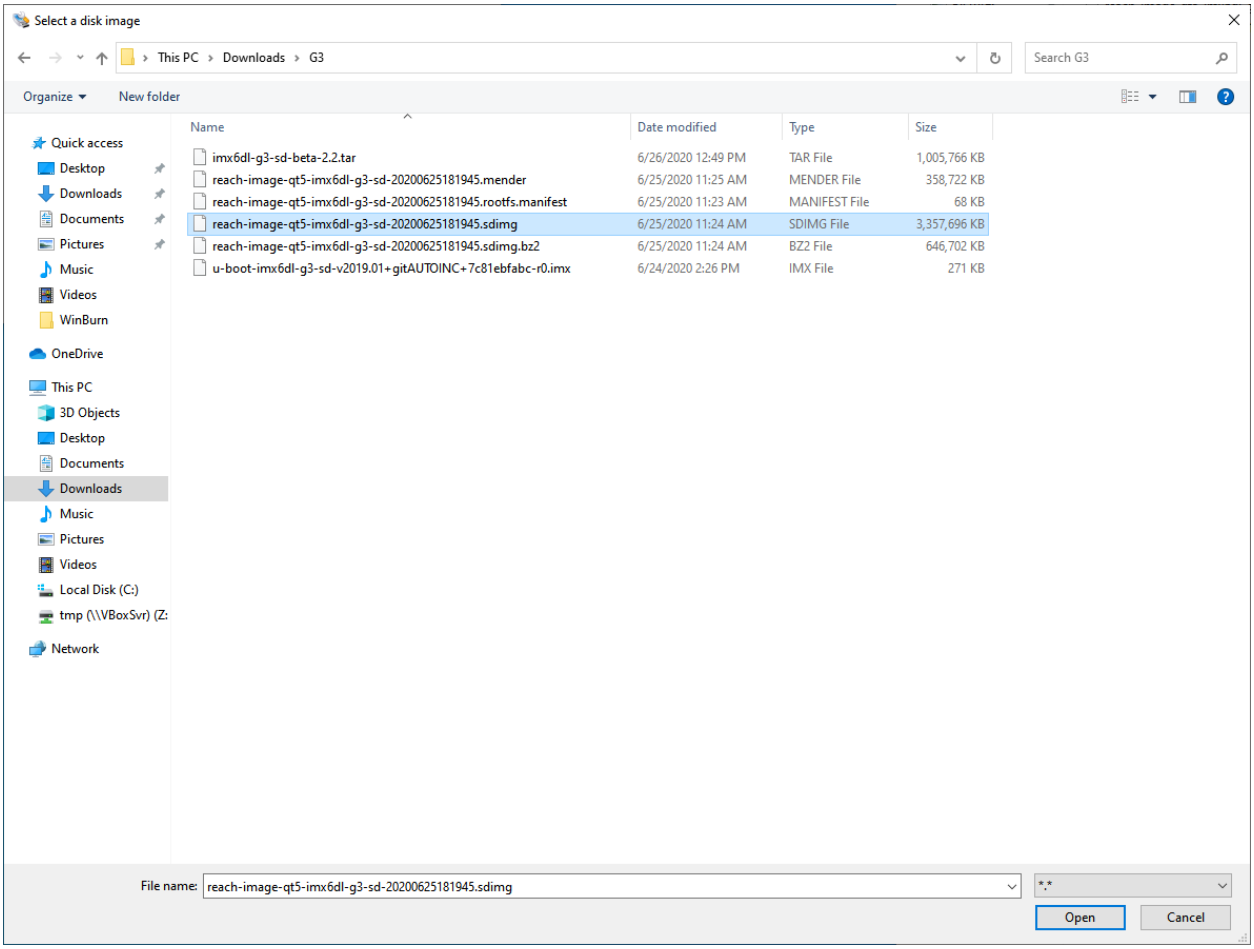
- Insert an SD card into your reader/writer.

> WARNING: Testing has shown inserting an SD card with multiple partitions after launching the win32 disk imager will cause it to potentially list the incorrect Windows 10 drive letter that represents the whole SD card.

- Launch the win32 disk imager application, which will bring up a window like the one in the figure below:

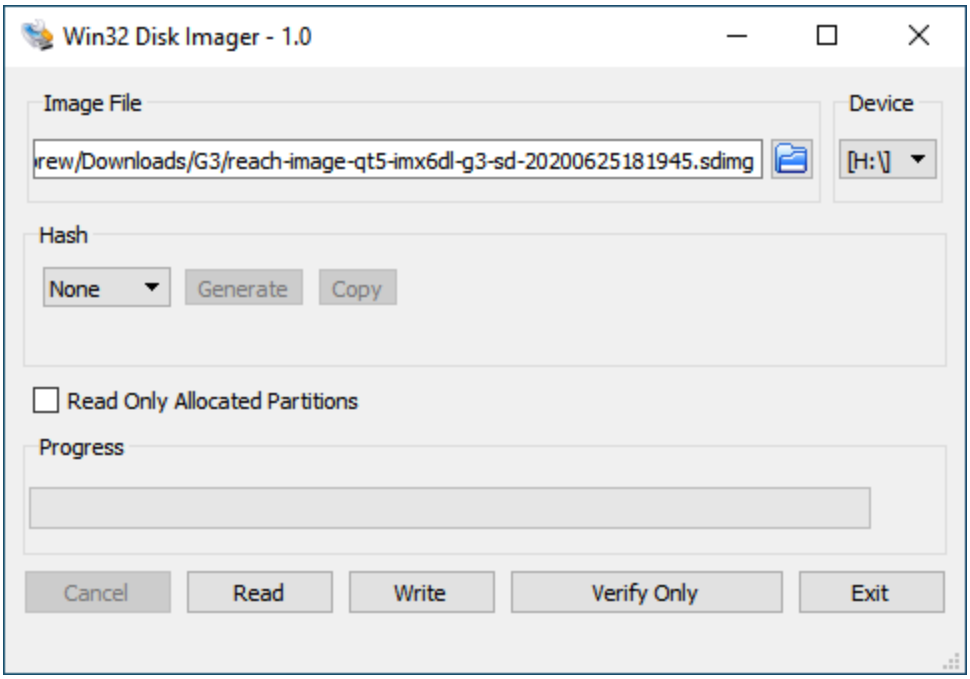


- Click the blue folder button in the **Image File** group box to open the image file selection dialog. Change the filter at the lower right corner to "*.*" so the ".sdimg" file will be listed. Select the **.sdimg** file as in the figure below:



Unless you have inserted more than just a single SD card, you have one available choice in the **Device** group box pull-down list. If there is more than one, use extreme care to select the Windows 10 drive letter representing the entire SD card.

- Select the appropriate target device as in the figure below:

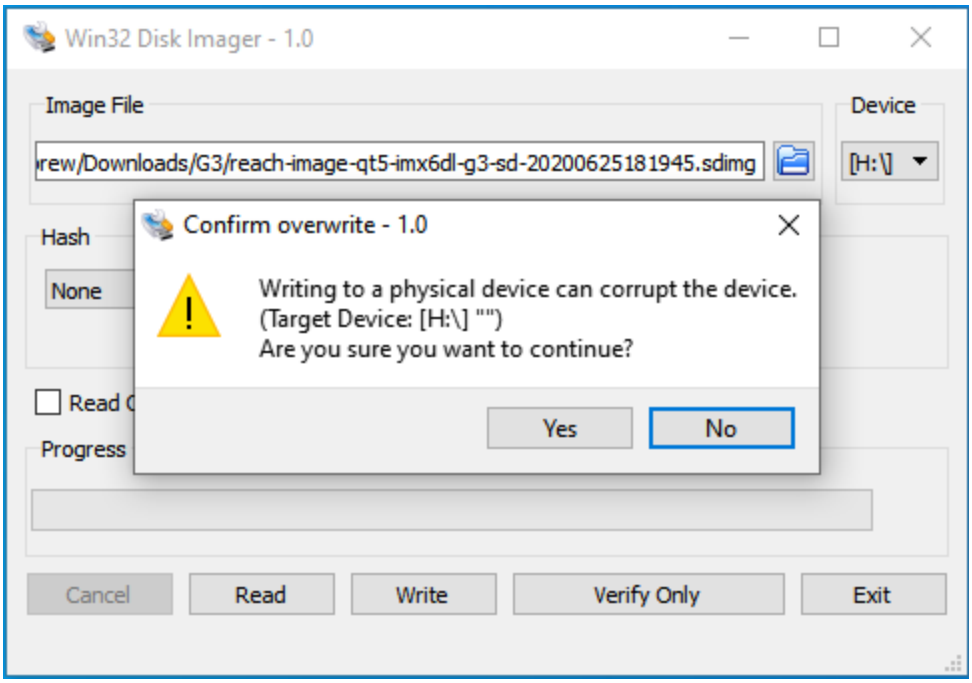


WARNING: If using a new SD card, you should only see a single option in the pull-down list. However, if the SD card has multiple partitions with filesystems that Windows 10 can read, there will be numerous options because Windows 10 will automatically mount those filesystems and assign drive letters. Be careful to select the option that represents the entire SD card.

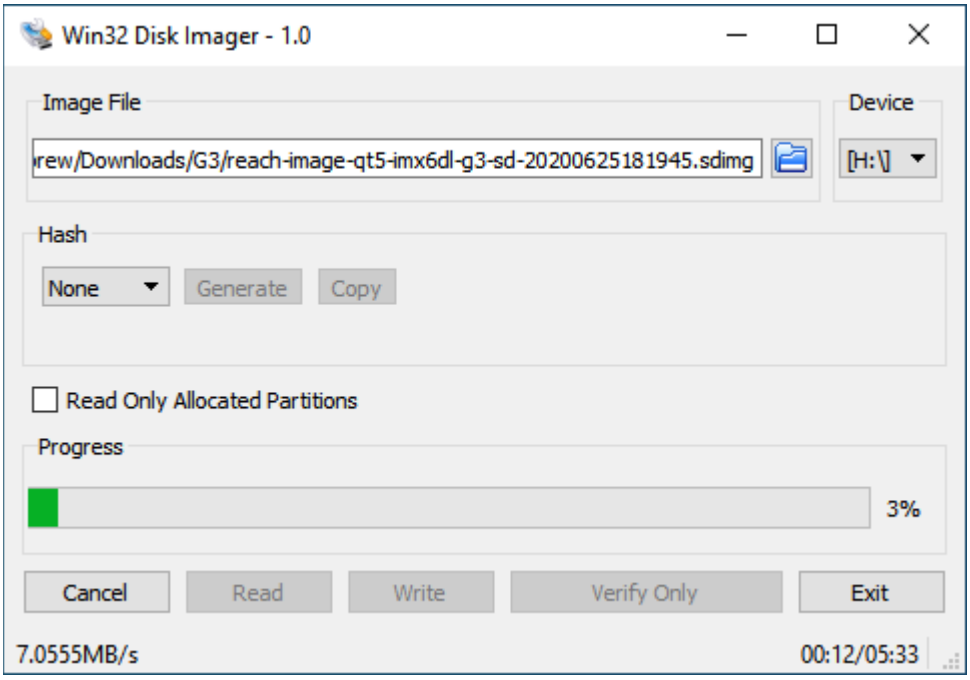
TIP: When in doubt, replace any existing partition table on the SD card with an empty DOS partition table, easily achieved on Linux with the **o command** in fdisk. On Windows 10, you can do it with Disk Management, found in the Control **Panel** -> **Administrative Tools** -> **Computer Management** -> **Disk Management**.

When you insert an SD card with an empty partition table, Windows 10 must format the disk before you use it. Dismiss the dialog box and launch the win32 disk imager to burn it. If you try the format, it will fail because the partition table is empty.

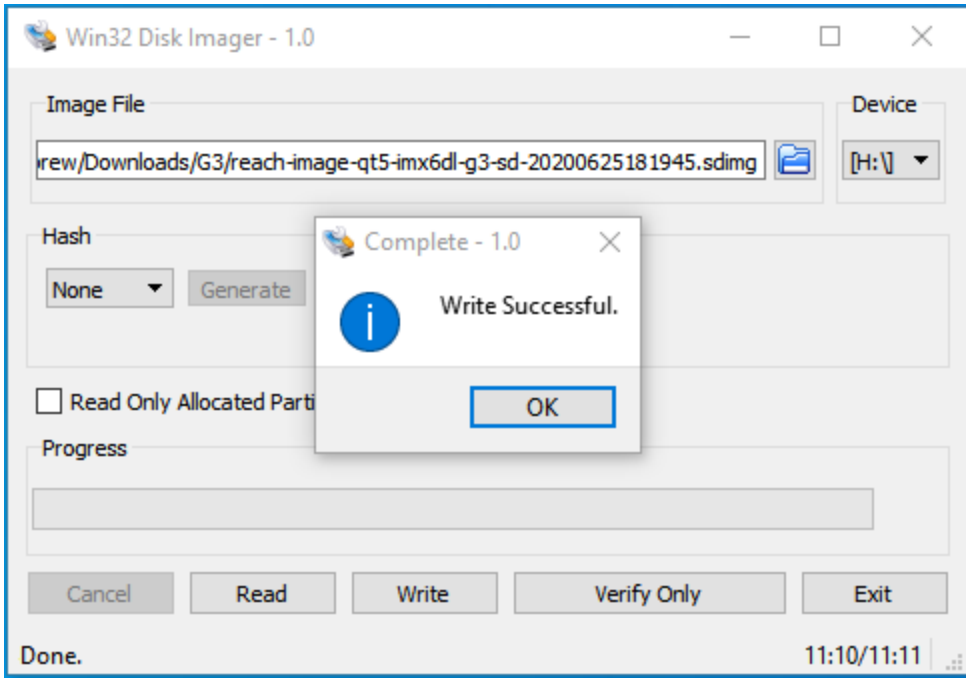
- Click the "Write" button, which will pop up the confirmation dialog below:



Click **Yes** to start the flashing process. The progress bar will show the status of the burn:



When the image burn is complete, you will get the **Write Successful** dialog:



eMMC Boot

Choose from the eMMC images available for download here.

Setting up the G3 module to boot from eMMC requires a different image. It comes as a tarball with a Mender[20] artifact, manifest file, bziped image, and u-boot image. The following bash sessions show how to:

- Unpack the tarball.
- Decompress the image.
- Use the mkemmc.sh script to flash the u-boot and Linux images to eMMC.

> NOTE: In most cases, before you flash the Linux image, the existing eMMC partitions need to be zeroed. Then use the mkemmc.sh script to reboot the system to guarantee the kernel recognizes that all old partitions are gone. Then enter the mkemmc.sh command with the same arguments to perform the actual flash.

The first two steps, unpacking the tarball and decompressing the image, should be done on a Linux desktop as it will be much slower on the target device. If you are using a Windows 10 development host, you can perform these steps inside the G3 Developer VM. Make a copy of the extracted u-boot and decompressed Linux image to a USB stick to transfer to the G3 module.

> TIP: The following examples use the Beta 2.2 eMMC tarball. Be sure to substitute filenames consistent with the release you download.

1. Unpack the tarball.

```
# to get the size of the downloaded file to compare to the value on the website.
> ls -l
total 1005768
-rw-r--r-- 1 reach reach 1029903360 Jul 30 13:46 imx6dl-g3-emmc-beta-2.2.tar

# to generate the hash of the downloaded file to compare to the value on the website.
> md5sum imx6dl-g3-emmc-beta-2.2.tar
6e38388f053338c585c2bbf73ecb1d41  imx6dl-g3-emmc-beta-2.2.tar

# to unpack the tarball
> tar xvf imx6dl-g3-emmc-beta-2.2.tar
reach-image-qt5-imx6dl-g3-emmc-20200625182854.rootfs.manifest
reach-image-qt5-imx6dl-g3-emmc-20200625182854.mender
reach-image-qt5-imx6dl-g3-emmc-20200625182854.sdimg.bz2
u-boot-imx6dl-g3-emmc-v2019.01+gitAUTOINC+7c81ebfabc-r0.imx
```

2. Decompress the image[21].

[20] Mender is used for system updates. See Appendix B: Updating the Embedded Linux OS for more information.
[21] Be sure to substitute the correct path to your USB stick.

```
# to decompress the image file
> bunzip2 reach-image-qt5-imx6dl-g3-emmc-20200625182854.sdimg.bz2

# to copy the u-boot and Linux images to the USB stick
> cp u-boot-imx6dl-g3-emmc-v2019.01+gitAUTOINC+7c81ebfabc-r0.imx reach-image-qt5-imx6dl-g3-emmc-20200625182854.sdimg /run/media/usbstick
```

3. The USB stick will automount at /run/media/sda1 when inserted[22] on the G3 module.

```
# change to the USB stick on the G3BNG
root@imx6dl-g3-sd:~# cd /run/media/sda1/B_2.2
root@imx6dl-g3-sd:/run/media/sda1/B_2.2# ls -l
total 3716768
-rwxrwx--- 1 root disk  367331328 Jun 25 11:34 reach-image-qt5-imx6dl-g3-emmc-20200625182854.mender*
-rwxrwx--- 1 root disk      69870 Jun 25 11:32 reach-image-qt5-imx6dl-g3-emmc-20200625182854.rootfs.manifest*
-rwxrwx--- 1 root disk 3438280704 Jun 25 11:32 reach-image-qt5-imx6dl-g3-emmc-20200625182854.sdimg*
-rwxrwx--- 1 root disk     277504 Jun 25 11:30 u-boot-imx6dl-g3-emmc-v2019.01+gitAUTOINC+7c81ebfabc-r0.imx*
```

4. Here is the actual eMMC flash. The first argument is the name of the u-boot image. The second argument is the Linux image you just decompressed. There will be an automatic reboot after zeroing eMMC. It will take 4+ minutes to zero eMMC.

```
# to flash the u-boot and Linux images to eMMC
root@imx6dl-g3-sd:/run/media/sda1/B_2.2# mkemmc.sh u-boot-imx6dl-g3-emmc-v2019.01+gitAUTOINC+7c81ebfabc-r0.imx reach-image-qt5-imx6dl-g3-emmc-20200625182854.sdimg

checking for eMMC partition at "/run/media/mmcblk3p1"

checking for eMMC partition at "/run/media/mmcblk3p2"

checking for eMMC partition at "/run/media/mmcblk3p3"

checking for eMMC partition at "/run/media/mmcblk3p4"

found existing eMMC partitions...
zeroing out eMMC
7818182656 bytes (7.8 GB, 7.3 GiB) copied, 279 s, 28.0 MB/s
932+0 records in
932+0 records out
7818182656 bytes (7.8 GB, 7.3 GiB) copied, 290.648 s, 26.9 MB/s

rebooting...
rerun mkemmc.sh after the reboot


The system is going down for reboot NOW! (ttymxc0) (Thu Jul 30 22:34:33 2020)
INIT: Sending processes the TERM signal

# ... a bunch of output snipped here for readability ...

Rebooting... reboot: Restarting system

```

[22] The USB stick used in this example has Beta 2.1 and Beta 2.2, hence the subdir under /run/media/sda1.

```
U-Boot 2019.01 (Jun 24 2020 - 21:26:46 +0000)

CPU:   Freescale i.MX6DL rev1.3 at 792 MHz
Reset cause: WDOG
Board: imx6dl-g3
I2C:   ready
DRAM:  2 GiB
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
Loading Environment from MMC... OK
Could not find supported display, disabling splash
In:    serial
Out:   serial
Err:   serial
Hit any key to stop autoboot:  0
Boot splash disabled
41368 bytes read in 125 ms (322.3 KiB/s)
6508000 bytes read in 428 ms (14.5 MiB/s)
Kernel image @ 0x12000000 [ 0x000000 - 0x634de0 ]
## Flattened Device Tree blob at 18000000
Booting using the fdt blob at 0x18000000
Using Device Tree in place at 18000000, end 1800d197

Starting kernel ...

Uncompressing Linux... done, booting the kernel.

# ... a bunch of output snipped here for readability ...

Starting User Application: OK

Reach Technology FrameBuffer 2.7.4 imx6dl-g3-sd ttymxc0

imx6dl-g3-sd login:
```

5. Following the system restart, navigate back to the directory on the USB stick and reissue the mkemmc.sh command with the same arguments.

```
# to actually flash the u-boot and Linux images to eMMC
Reach Technology FrameBuffer 2.7.4 imx6dl-g3-sd ttymxc0

imx6dl-g3-sd login: root
root@imx6dl-g3-sd:~# cd /run/media/sda1/B_2.2
root@imx6dl-g3-sd:/run/media/sda1/B_2.2# mkemmc.sh u-boot-imx6dl-g3-emmc-v2019.01+gitAUTOINC+7c81ebfabc-r0.imx reach-image-qt5-imx6dl-g3-emmc-20200625182854.sdimg

checking for eMMC partition at "/run/media/mmcblk3p1"

checking for eMMC partition at "/run/media/mmcblk3p2"

checking for eMMC partition at "/run/media/mmcblk3p3"

checking for eMMC partition at "/run/media/mmcblk3p4"

flashing eMMC SD image "reach-image-qt5-imx6dl-g3-emmc-20200625182854.sdimg" to "/dev/mmcblk3"
3422552064 bytes (3.4 GB, 3.2 GiB) copied, 174 s, 19.7 MB/s
409+1 records in
409+1 records out
3438280704 bytes (3.4 GB, 3.2 GiB) copied, 181.88 s, 18.9 MB/s
542+0 records in
542+0 records out
277504 bytes (278 kB, 271 KiB) copied, 0.197568 s, 1.4 MB/s

setting up eMMC boot
Changing ext_csd[BOOT_BUS_CONDITIONS] from 0x01 to 0x01

The image has been flashed to eMMC.
1) Run "halt"
2) Power down the board
3) Insert the eMMC boot jumper
4) Power up the board

root@imx6dl-g3-sd:/run/media/sda1/B_2.2#
```

DTB Selection

TIP: All production G3 controllers now include an EEPROM factory programmed with the proper DTB for your module. The DTB setting will survive a complete firmware image reflash or update.

The G3 module controller card uses the same embedded Linux image regardless of which display or touch controller a specific module uses. So that the system can access the correct drivers for display and touch hardware, it needs to know which binary device tree blob (DTB) to use. This action is only required the first time a new image is flashed to an SD card or eMMC[23].

Log in to the G3 module as root, then:

```
# to select the DTB for your panel and touch controller
root@imx6dl-g3-sd:~# fw_setenv mender_dtb_name <dtb-file-name.dtb>

# to activate the selected DTB
root@imx6dl-g3-sd:~# sync
root@imx6dl-g3-sd:~# reboot
```

The following tables list the available DTBs as of the time of writing this document.

G3BNG Available Device Tree Blobs (DTBs)

Panel Size	Touch Sensor	DTB File
5.7"	Resistive (Eversion)	imx6dl-g3-5p7-vga-lcd-tsc2046.dtb
5.7"	Resistive (KOE)	imx6dl-g3-5p7-vga-ldb-tsc2046.dtb
7"	Projected Capacitive	imx6dl-g3-7-wvga-lcd-focaltech.dtb
7"	Resistive	imx6dl-g3-7-wvga-lcd-tsc2046.dtb
7"	Resistive (KOE)	imx6dl-g3-7-wvga-ldb-tsc2046-inv.dtb
10.1"	Projected Capacitive	imx6dl-g3-10p1-wxga-ldb-focaltech.dtb
10.1"	Resistive	imx6dl-g3-10p1-wxga-ldb-tsc2046.dtb
10.4"	Projected Capacitive/HID	imx6dl-g3-10p4-xga-ldb-egalax.dtb
12.1"	Projected Capacitive/HID	imx6dl-g3-12p1-wxga-ldb-amtouch.dtb

[23] In general, update releases are very infrequent so you may never need to flash a new image and set the DTB variable. Also, most G3 modules are shipped pre-assembled with a display and touch controller. Unless you order bare controller cards, the SD card shipped with the unit will already have the DTB selected and the touch sensor calibrated.

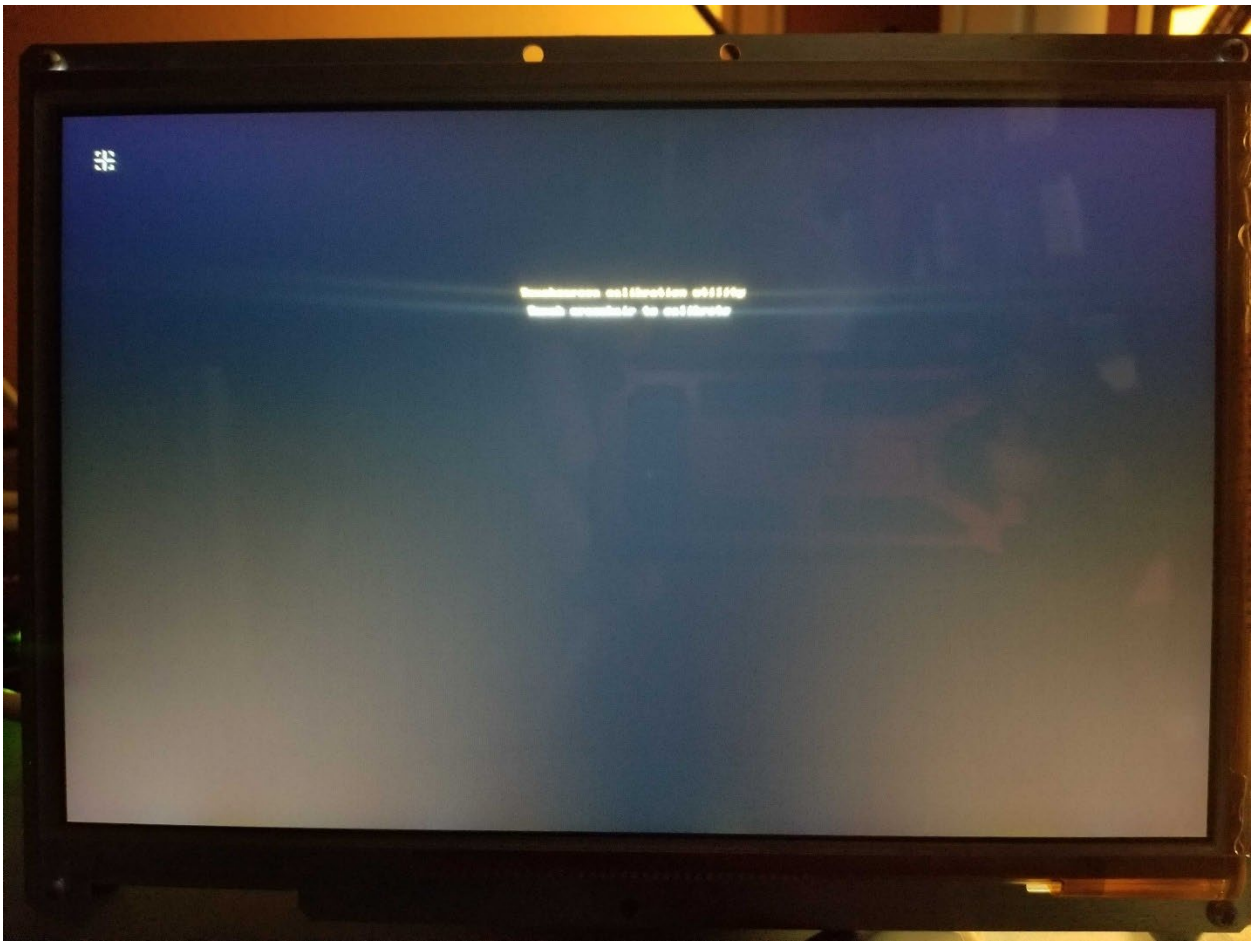
G3MSB Available Device Tree Blobs (DTBs)

Panel Size	Touch Sensor	DTB File
4.3" WQVGA	Resistive (Innolux)	stm32mp1-g3msb-4p3-wqvga-tsc2046.dtb
4.3" WQVGA	Projected Capacitive (Microtips)	stm32mp1-g3msb-4p3-wqvga-focaltech.dtb
4.3" WVGA	Projected Capacitive (Microtips)	stm32mp1-g3msb-4p3-wvga-focaltech.dtb
5" WVGA	Resistive (Microtips)	stm32mp1-g3msb-5-wvga-tsc2046.dtb

For example, to select the proper DTB for a G3BNG with a 7" display and a projected capacitive touch sensor:

```
1 | # to select the DTB for 7" PCAP
2 | root@imx6dl-g3-sd:~# fw_setenv mender_dtb_name imx6dl-g3-7-wvga-lcd-focaltech.dtb
3 |
4 | # to activate the selected DTB
5 | root@imx6dl-g3-sd:~# sync
6 | root@imx6dl-g3-sd:~# reboot
```

Once your G3 module reboots after selecting the appropriate DTB, you must calibrate the touch sensor. This process starts automatically at the first reboot. The calibration program draws a series of five boxes with internal crosshairs, one at a time. Touch the panel at each crosshair as it is displayed. Once the calibration is complete, the G3 module will complete its normal boot process.



End Product Application

WARNING: Your product application should never run as the root user for many reasons. A specific user and group “apps” have been created and granted necessary access to use the required system resources.

IMPORTANT: ALL files related to your end product application must be in /data. See more information on the /data layout.

The data filesystem paths table notes several directories under /data for installing the product application executables, configuration files, and multimedia resources. Your challenge will be to create a proper /data/sbin/run_application script to install your product application. The user_app init script with the "apps" effective user id will launch this script.

IMPORTANT: Do not modify the /etc/init.d/user_app script. All end product application-specific startup and shutdown code must be in /data/sbin/run_application.

The run_application Script

As explained in the Product Application Launch, you must create a run_application script to launch your end product application and disable or remove the existing run_demos script in /data/sbin.

```
# change to /data/sbin
root@imx6dl-g3-sd:~# cd /data/sbin

# to disable the run_demos script
root@imx6dl-g3-sd:/data/sbin# chmod -x run_demos

# or, to remove the run_demos script
root@imx6dl-g3-sd:/data/sbin# rm -f run_demos
```

The following code block shows a sample run_application script. *Remember to make it executable.*

```
#!/bin/sh

# sample run_application script

APP="foo-viewer"

# DO NOT change XDG_RUNTIME_DIR; it is set up by /etc/init.d/user_app
# before it uses su to execute this script as the "apps" user
export XDG_RUNTIME_DIR=/var/run/user_app
PID_FILE=${XDG_RUNTIME_DIR}/${APP}.pid
APP_EXE="/data/bin/${APP}"

# use this var to pass args to your app
# DO NOT put "-platform eglfs -plugin tslib" here!!!
# It **will** break TS input on some screens
```

```
APP_ARGS=""

source /etc/profile.d/tslib.sh
# NOTE: the following script has **all** env vars needed for proper Qt operation
#       DO NOT add Qt env vars here!!!
source /etc/profile.d/qt-eglfs.sh

start() {
   start-stop-daemon -b -S -q -m -p ${PID_FILE} --exec $APP_EXE -- $APP_ARGS
}

stop() {
    start-stop-daemon -s QUIT -K -q -p ${PID_FILE}
}

```

```
case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    *)
        echo "Usage: $0 {start|stop}"
        exit 1
esac

exit $?
```

Configuration

This section covers the configuration of the G3 module embedded Linux image.

Splash Screens

> NOTE: Splash screen image files must be in 24-bit BMP format.

The G3 module supports up to two boot splash screens. The u-boot boot loader displays the first and is only visible for a second or two. The second is displayed once the Linux kernel runs and will remain visible until the end product application takes over the screen. There is a brief period in which the screen is blank while the kernel initializes. By default, the same image is used for both splash screens.

U-Boot Splash

> IMPORTANT: **The u-boot splash screen is turned off by default.** The blanking period between the u-boot and Linux splash screens causes flashing on many panels. The information to enable the u-boot splash is retained here so that you can try it and see if it works for you.

To control the u-boot splash screen, use the variables in the table below. Set these variables using the fw_setenv command from the Linux prompt on the G3 module[24].

Variable Name	Description	Default
Splashen	Turn uboot splash screen on or off	"n"[25]
Splash	Splash screen image file	"splash.bmp"[26]
splashpos	Splash image position	"m,m"[27]
splashpart	Splash screen image file partition	"4"[28]

[24] This can also be accomplished by breaking into u-boot during power-up using the setenv command to the u-boot environment variable(s). This should be followed with the saveenv command to persist the setting into the u-boot backing store file.
[25] Must be “y” to enable. Anything else will disable. Disabled by default due to excessive flashing with several panels.
[26] Effectively /data/share/pixmaps/splash.bmp. We recommend using “bootsplash.bmp” for the file name if you change it. Then copy your u-boot splash image to /data/share/pixmaps/bootsplash.bmp.
[27] The default value causes the image to be centered if smaller than the screen. It can also be set to an x,y offset relative to the upper left corner of the screen.
[28] By default, Partition 4 on i.MX6-based modules and Partition 7 on STM32MP157-based modules are mounted at /data. We recommend you do not change this.

Linux Splash

After fully initializing the kernel, the image stored in /data/share/pixmaps/splash.bmp will display until the product application takes over the screen. You can not turn off this splash screen, but you can achieve the same effect by storing a pure black 24-bit BMP at /data/share/pixmaps/splash.bmp. If this image is smaller than the screen size, it will be centered.

/data Filesystem

The /data filesystem is the only persistent writeable filesystem in the production-embedded software images. Transient data such as log files are typically written to files in a small memory resident filesystem. The root filesystem is intentionally mounted read-only to avoid breaking the system.

/data Filesystem Resize

As of the RC1 release, the /data partition and filesystem automatically expand to fill the available space, either on SD card or in eMMC, the first time the image is booted. For more information, see the Note under Boot Setup.

System Configuration Files

As discussed in detail in the Read-only Root Filesystem and Systems Configuration Files, the rootfs is mounted read-only. Occasionally, you may need to change the fixed configuration files in /etc. First, the system must be in an inactive state. To do this, shut down your end product application and all optional services possible.

> TIP: Any process that holds an open file descriptor on /etc will prevent unmounting the overlay. The udev system service is an excellent example of this. If you have trouble umounting /etc, you can use lsof | grep /etc to find the processes with open file descriptors on /etc.

Second, you must unmount the overlay by:

```
# to stop udev (you must be root to do this)
root@imx6dl-g3-sd:~# /etc/init.d/udev stop

# to unmount the overlay (you must be root to do this)
root@imx6dl-g3-sd:~# umount /etc
```

Finally, remount the rootfs so it is temporarily writeable. Here is how:

```
# to remount rootfs read-write (you must be root to do this)
root@imx6dl-g3-sd:~# mount -n -o remount,rw /

# now you are ready to edit configuration files in the real /etc
```

When your changes are complete and tested, you should immediately reboot the G3 module so the rootfs return to their default read-only state.

Optional Services

The G3 module includes several Linux services that may or may not be of value to your application use case. Reach Technology has engineered the system so these services can be enabled or disabled using the chkconfig command.

You should turn off services your product application does not require to reduce memory and CPU cycle consumption and reduce your boot time.

> TIP: To manage optional services, log in to the G3 module as root, then:

```
# to list available services.
root@imx6dl-g3-sd:~# chkconfig

# to enable a service.
root@imx6dl-g3-sd:~# chkconfig <service-name> on

# to disable a service.
root@imx6dl-g3-sd:~# chkconfig <service-name> off

# to activate the selected set of services.
root@imx6dl-g3-sd:~# sync
root@imx6dl-g3-sd:~# reboot
```

Here are the optional services.

Service	**Description**
bluetooth	Setup Bluetooth devices
cgroups-init	Setup kernel control groups
dbus-1	System software communication BUS
docker.init	Application service containerization[29]
firehol	Strong firewall with minimal rules required
lighttpd	Lightweight web server
mosquitto	MQTT broker
netmount	Mount NFS filesystems listed in /etc/fstab[30]

[29] Docker containers are available. They have not been extensively tested.
[30] The G3 module can mount shares from NFS servers. It cannot act as an NFS server.

Service (continued)	Description (continued)
networking	Control loopback[31], Ethernet and wireless interfaces[32]
nfscommon	Provides rpc.statd for NFS
ntpdate	`ntpdate` sets system time with NTP server synchronization.
ntpd	Network Time Protocol daemon synchronizes computer clocks over the internet.
nginx	Reverse proxy
ppp	Control dialup interfaces[33]
pulseaudio.sh	Audio mixing server
redis-server	Fast memory resident KVP database
rng-tools	Random number generator tools[34]
rpcbind	rpc mapper for NFS
sshd	secure shell server[35]
user_app	controls end product (user) application[36]

[31] The networking service should *always* be enabled if the end product application requires the loopback (localhost) interface, which is almost always the case.
[32] Wireless devices such as WiFi or BLE dongles are controlled by the networking service. Most CDMA/GSM/LTE modems are controlled be the ppp service even though they are actually wireless devices.
[33] Wireless devices such as WiFi or BLE dongles are controlled by the networking service. Most CDMA/GSM/LTE modems are controlled be the ppp service even though they are actually wireless devices.
[34] The random number generator service should be left enabled unless you can guarantee that your application does not employ networking or encryption. The only advantage to turning it off is that boot time will decrease by a few seconds.
[35] You should only turn sshd off if there are no network connections into the end product the G3 module is integrated. See cautions regarding sshd in the Important Note in Boot Setup.
[36] See End Product Application for information on installing the end product application and how it is launched at system boot. Note: The node.js package is installed but does not have a dedicated init script because it requires application-specific code. Reach Technology assumes your end product application init script, run_application, will handle starting and stopping of node.js-based application components.

Boot Time Optimization

A splash screen lets a user know the system is alive as early as possible while it finishes booting. System boot times can vary widely depending on the complexity of your product application and the services it requires. Despite the desire to have the system "online" as quickly as possible, one must always ensure the system is safe. Adding internet connectivity brings critical considerations, which inevitably affect boot times adversely. Here are some things to think about when optimizing your product boot time.

- **Firehol:** This is the biggest time offender used to set up an iptables-based firewall[37]. You can safely turn this off if you do not have an active Ethernet, WiFi or PPP network connection or only have a local Ethernet inside your product. This service is turned off by default, so any network connections to the target are open until you enable it.

- **Networking:** This is the second biggest offender. The eth0 interface is set in "auto" mode by default. If a wire is plugged in, the interface will be brought up (takes 4 to 5 seconds) and then dhcpcd will attempt to get a dynamically assigned Internet Protocol (IP) address, DNS, and routing configuration data. On the other hand, if the wire is not plugged in, it can take quite a while (up to 15 seconds) for the init code to stop attempts at bringing up the interface. Since turning off the networking service is generally a bad idea, you should comment out the "auto eth0" line in /etc/network/interfaces file if you are not using the Ethernet port.

- **NTPDate:** The ntp_date client makes significant corrections to the system timestamp (much more extensive than what ntpd allows) at the system boot. If you do not have a network connection, it is not needed. It is also unnecessary if your product does not sit with the power off for extended periods. The RTC is temperature-compensated and should remain within the limits of ntpd for quite some time.

- **NTPD:** Like ntpdate, ntpd is unnecessary if you do not have an external network connection. If you utilize NFS, it is crucial to employ ntpd to prevent system freezes caused by clock misalignment on both the G3 module and the server.

- **lighttpd, Nginx, Node.js:** Several services related to HTML content and web applications. Node.js will not start unless your run_application script launches it. If your product UI does not use HTML5 (i.e., web UI or a RESTful API) you do not need lighttpd. The nginx proxy is a bit more complicated. Generally, if you have an external network and are offering web services via lighttpd or a node.js-based app, or if you are using either the redis-server database or the mosquitto MQTT server and allow external access, then you should proxy those things behind nginx. This enables you to do your web security and access logging config in nginx rather than doing it multiple times for all those other components.

[37] Setting up firewall rules is slow because the kernel drivers that implement the actual filters are compiled as modules. So each time a rule is created that uses a filter not already loaded, the kernel module loading infrastructure has to pull another one in from SD or eMMC. While it is possible to build common rules into the kernel and make it faster, Reach Technology chose not to make more RAM available to end-product applications that do not need a firewall.

- **Audio:** If your product does not use audio, you can safely disable pulseaudio.sh and dbus-1.
- **Containers:** If your product does not use containers, you can safely disable docker.init and cgroups-init.

WARNING: There are dependencies between some of the optional services. See the table below. You can meet the "net" dependency by using any of the following three services: Bluetooth, networking, or ppp. You should *always* enable the networking service if your end product application requires the loopback (localhost) interface, which is almost always the case.

In some cases, if the service will only be used internally on the G3 module by your end product application, the networking script is sufficient because it will start the loopback interface.

Service	Dependent On
docker.init	cgroups-init
lighttpd	net
mosquitto	net
netmount	net, nfscommon, rpcbind, ntpd
Nginx	net
ntpdate	net
Ntpd	net
pulseaudio.sh	dbus-1
redis-server	Net

Internet of Things (IoT) and Industrial Internet of Things (IIoT)

The G3 module embedded software image includes IoT/IIoT application components controlled by the standard chkconfig command, disabled by default.

Note: IoT/IIoT applications generally require installing Trusted Platform Module (TPM) hardware in the G3 module. All modules support the addition of a TPM on an application-specific daughter card via the I2C bus. The following sections discuss enabling and configuring the IoT/IIoT-related components.

Hardware Manual

This section is for people who need to understand the sub-systems available on touchscreen display modules. There is a sub-manual for each board type within the G3 family. See the G3 Software and the G3 Developer sections for information on the embedded software environment, development options, and tools.

External References

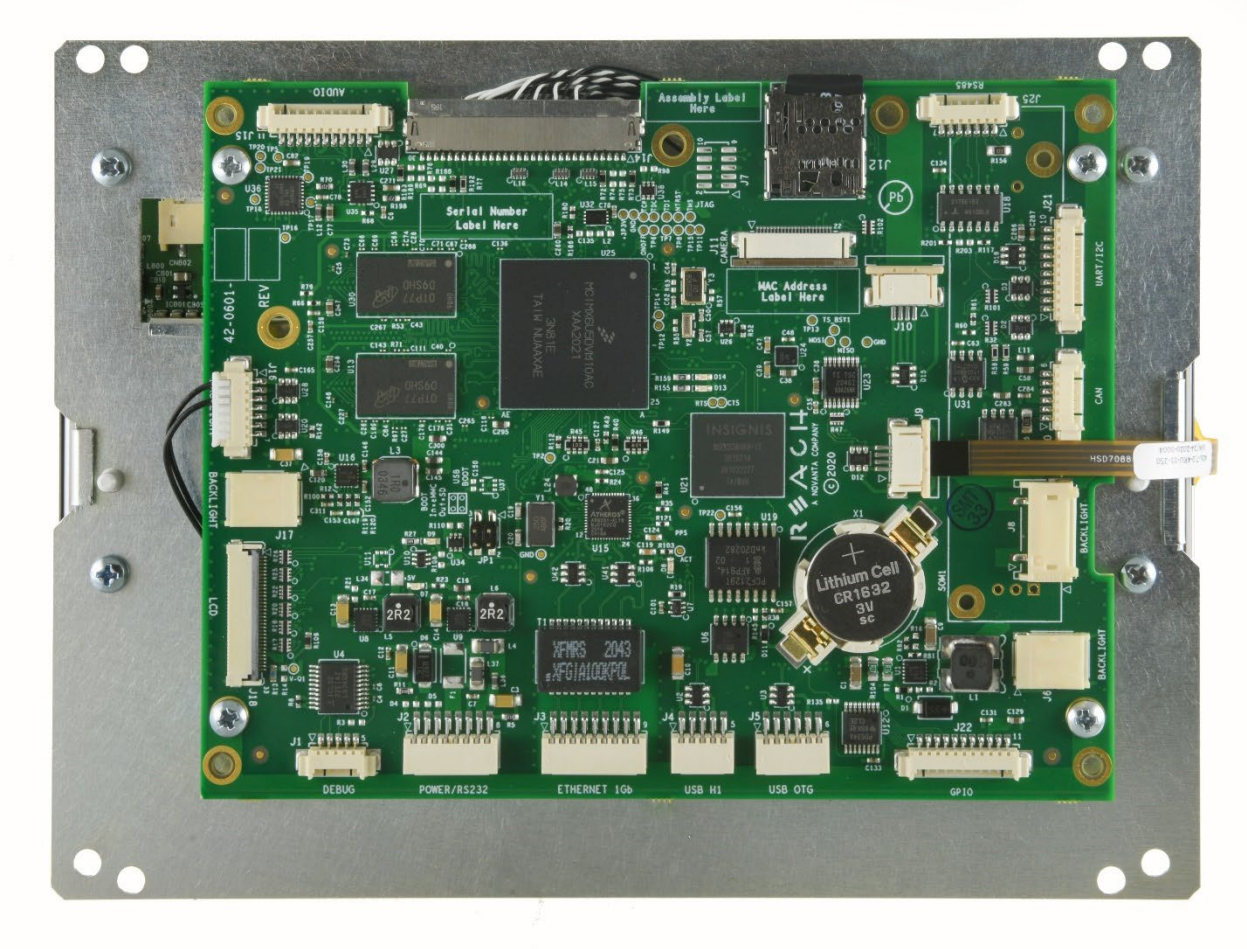
Please see the following sources for additional information:

Reference	Subject
i.MX6DL	NXP i.MX6 DualLite Product Documentation
STM32MP157C	STM32MP157C Arm Dual Cortex-A7
ICL3222E	Renesas ICL3222E Product Documentation
USB2	USB2 Specification
802.3AB	IEEE 802.3ab (1000baseT) Specification

G3BNG Controller Hardware Manual

Reach Technology G3BNG embedded controllers support LCD screen resolutions up to 1366 x 786 pixels. They allow you to build a custom user interface and utilize many built-in options to communicate with your product components. G3BNG is based on a high-speed, dual-core ARM processor and can support a variety of displays and touch sensors.

It uses the NXP i.MX 6DualLite processor, which is a dual-core implementation of the Arm Cortex-A9. Take advantage of various I/O interfaces running low-cost connectors. Modules consist of single board controllers integrated with touchscreen panels. Development Kits include everything you need to create a prototype quickly. When you are ready, move smoothly into production with off-the-shelf display modules that offer 10+ years of availability as a minimum.

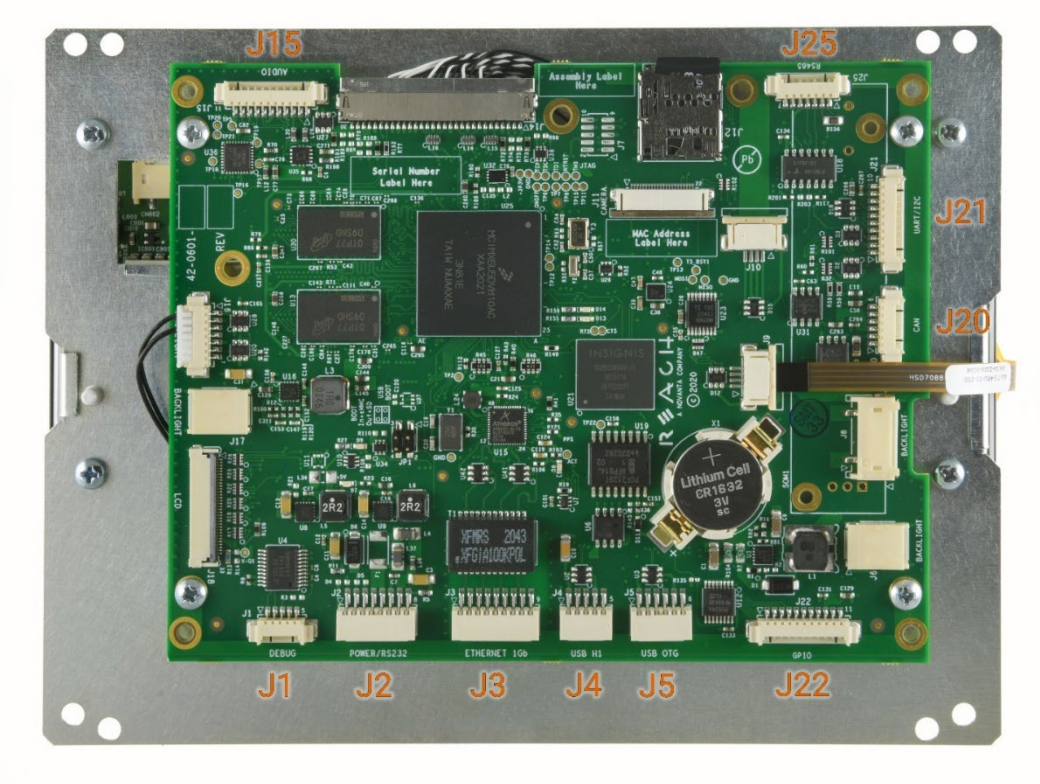


G3BNG Controller Interfacing

G3BNG contains a rich set of I/O interfaces, discussed in detail in the following sections. The table below lists the onboard connectors and pins, ports, and busses available on those connectors.

Connector	Port(s)
J1	Console (Debug) Port
J2	Power, RS-232 Port
J3	Gigabit Ethernet
J4	USB 2.0 Host Port
J5	USB 2.0 OTG Port
J15	Audio
J20	CAN Bus
J21	UART, I2C Bus, GPIO
J22	GPIO
J25	RS-485 Port

See location annotations in the image showing I/O connectors in the previous table.



Power

Connect DC power for the G3BNG controller and the touch panel via J2[38]. Although exact power consumption depends on peripherals and the touch panel, the module will consume 7W to 15W when the backlight is active.

G3 modules with 7" and smaller displays use a 5V, 3A DC power source. Larger displays, for example, 10.1" and greater, use a 12V, 1.5A DC power source.

Console (Debug) Port

Access the controller Debug Interface via J1[39]. This RS-232 level serial port runs at 115.2k Baud, 8-bit, no parity, and 1 stop bit. During product development, you **always** need to have this port connected. You will need a terminal emulator program or shell compatible with ANSI color escape sequences.

Examples:

- For Windows, Tera Term or PuTTY[40]
- For Linux, Picocom in a bash shell

RS-232 Port

A standard RS-232 port[41] is also available on J2[42].

Gigabit Ethernet

A standard 1000baseT[43] network port is available on J3[44].

SB 2.0 Host Port

Connect standard USB 1.0 and USB 2.0 peripherals[45] to J4[46].

---

[38] Use Reach Technology "Y" cable, PN 23-0160-18 for 5 VDC or PN 23-0173-18 for 12 VDC, if you do not wish to build custom cables.
[39] Use Reach Technology debug cable, PN 23-0161-72, if you do not wish to build custom cables.
[40] We recommend PuTTY because it supports both serial and network connections, as well as both Windows 10/11 and Linux.
[41] Standard RS-232 voltage levels.
[42] Use Reach Technology "Y" cable, PN 23-0160-18 for 5 VDC or PN 23-0173-18 for 12 VDC, if you do not wish to build custom cables.
[43] Due to limitations in the i.MX6 architecture, the ethernet port maximum throughput is limited to approximately 430 Mbit/S.
[44] Use Reach Technology Ethernet cable, PN 23-0148-12, if you do not wish to build custom cables.
[45] Some touch panels with HID touch controllers use this port.

[46] Use Reach Technology USB cable, PN 23-0166-12, if you do not wish to build custom cables.

USB 2.0 OTG Port

If J4 is already in use[47] or you want to use the G3BNG as a USB device[48] rather than as a host, you can use the USB 2.0 OTG port on J5. This port can be either a host port[49] or a device port[50].

Audio

Connect stereo headphones, a mono microphone and/or a mono speaker to J15[51].

CAN Bus

A CAN bus connection is available on J20[52].

UART

A logic-level[53] serial port is available on J21[54].

I²C Bus

The I2C bus is also available on J21[55].

GPIO

GPIO pins are available on J21[56] and J22[57].

RS-485 Port

Connect full duplex RS-485[58] peripherals to J25[59].

Camera port

Connect MIPI-CSI peripherals to J11. This port is not currently available and should not be used.

---

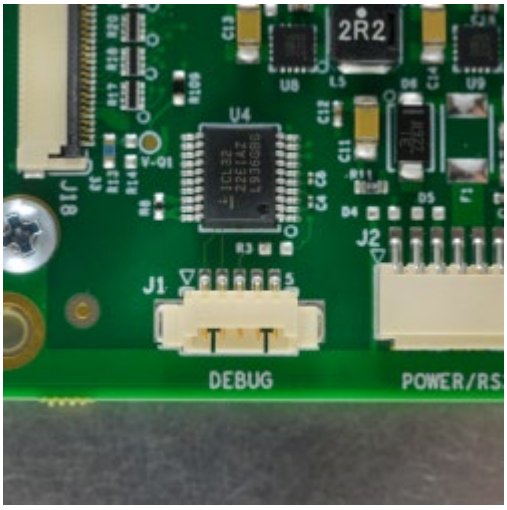
[47] For example, by an HID touch controller.
[48] For example, as a Linux Network Gadget for peer-to-peer USB networking.
[49] Use Reach Technology USB Host cable, PN 23-0167-12, if you do not wish to build custom cables.
[50] Use Reach Technology USB Device cable, PN 23-0168-12, if you do not wish to build custom cables.
[51] Use Reach Technology audio flying leads cable, PN 23-0144-10, for prototyping.
[52] Use Reach Technology CAN flying leads cable, PN 23-0146-10, for prototyping.
[53] CMOS voltage levels.
[54] Use Reach Technology I²C/UART/GPIO flying leads cable, PN 23-0149-10, for prototyping.
[55] Use Reach Technology I²C /UART/GPIO flying leads cable, PN 23-0149-10, for prototyping.
[56] Use Reach Technology I²C /UART/GPIO flying leads cable, PN 23-0149-10, for prototyping.
[57] Use Reach Technology GPIO flying leads cable, PN 23-0144-10, for prototyping.
[58] This port supports full duplex **only**.
[59] Use Reach Technology RS-485 flying leads cable, PN 23-0145-10, for prototyping.

Connector Pinouts and Part Numbers

The following sections provide the pinout for each connector and part number for the required mating plug.

J1: Debug Interface Connection

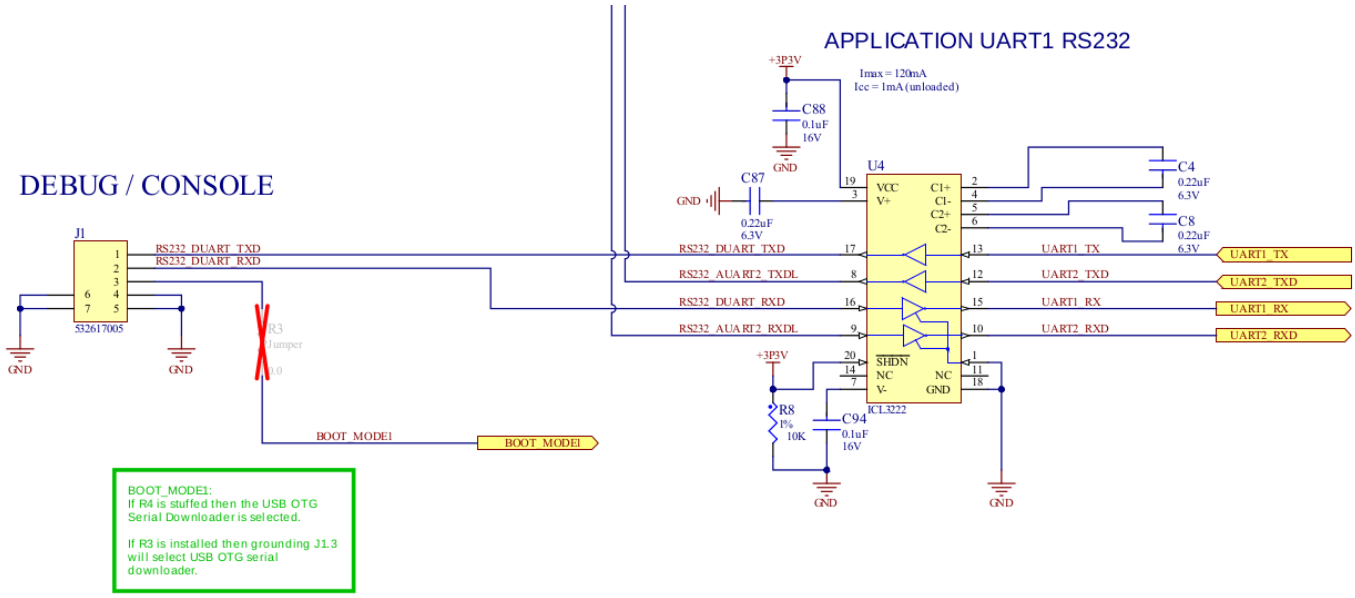
5 pin PicoBlade, MOLEX Part Number: [51021-0500]



J1 Pinout

Pin	Signal
1	TxD Output ESD-protected[60]
2	RxD Input ESD-protected[61]
3	Boot_Mode_1[62]
4	GND
5	GND

G3BNG Debug Interface Schematic



[60] Internal ESD protection as specified for the ICL3222E device, +/- 16 KV.
[61] Internal ESD protection as specified for the ICL3222E device, +/- 16 KV.
[62] If R3 is stuffed, short to GND to enable USB OTG serial bootloader.

J2: Power, RS-232 Connection

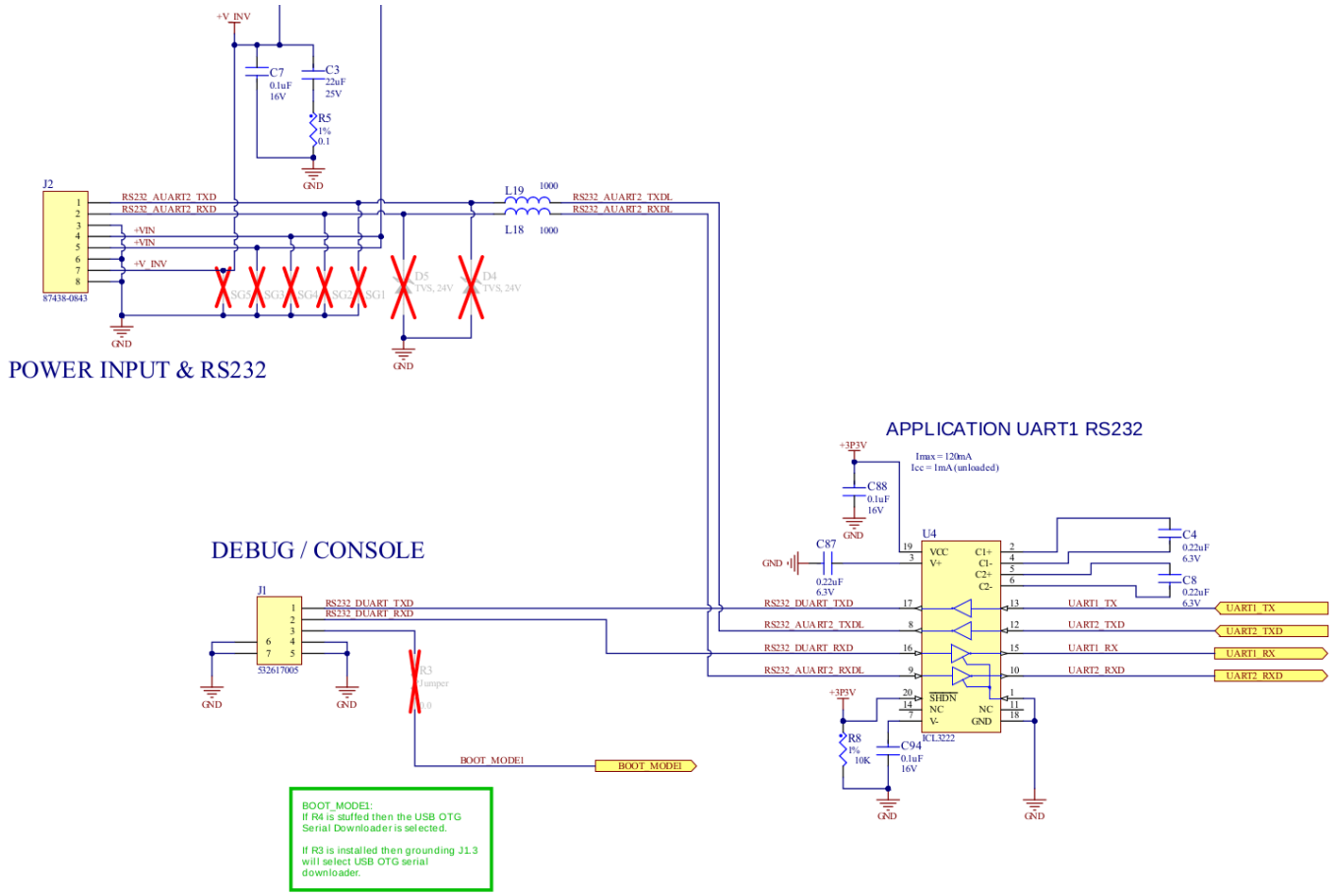
8 pin Pico-SPOX, MOLEX Part Number: [87439-0800]

J2 Pinout

Pin	Signal
1	TxD Output ESD-protected[63]
2	RxD Input ESD-protected[64]
3	GND (Power and Communications)
4	VIN 5V up to 12V DC Input[65]
5	VIN 5V up to 12V DC Input[66]
6	GND (Power and Communications)
7	V INV Optional Backlight Voltage Source[67]
8	GND (Power and Communications)

[63] For ESD protection details, refer to the specifications of the ICL3222E transceiver chip.
[64] For ESD protection details, refer to the specifications of the ICL3222E transceiver chip.
[65] Depends on LCD panel requirements.
[66] Depends on LCD panel requirements.
[67] Contact Reach Technology for more information.

Power, RS-232 Schematic



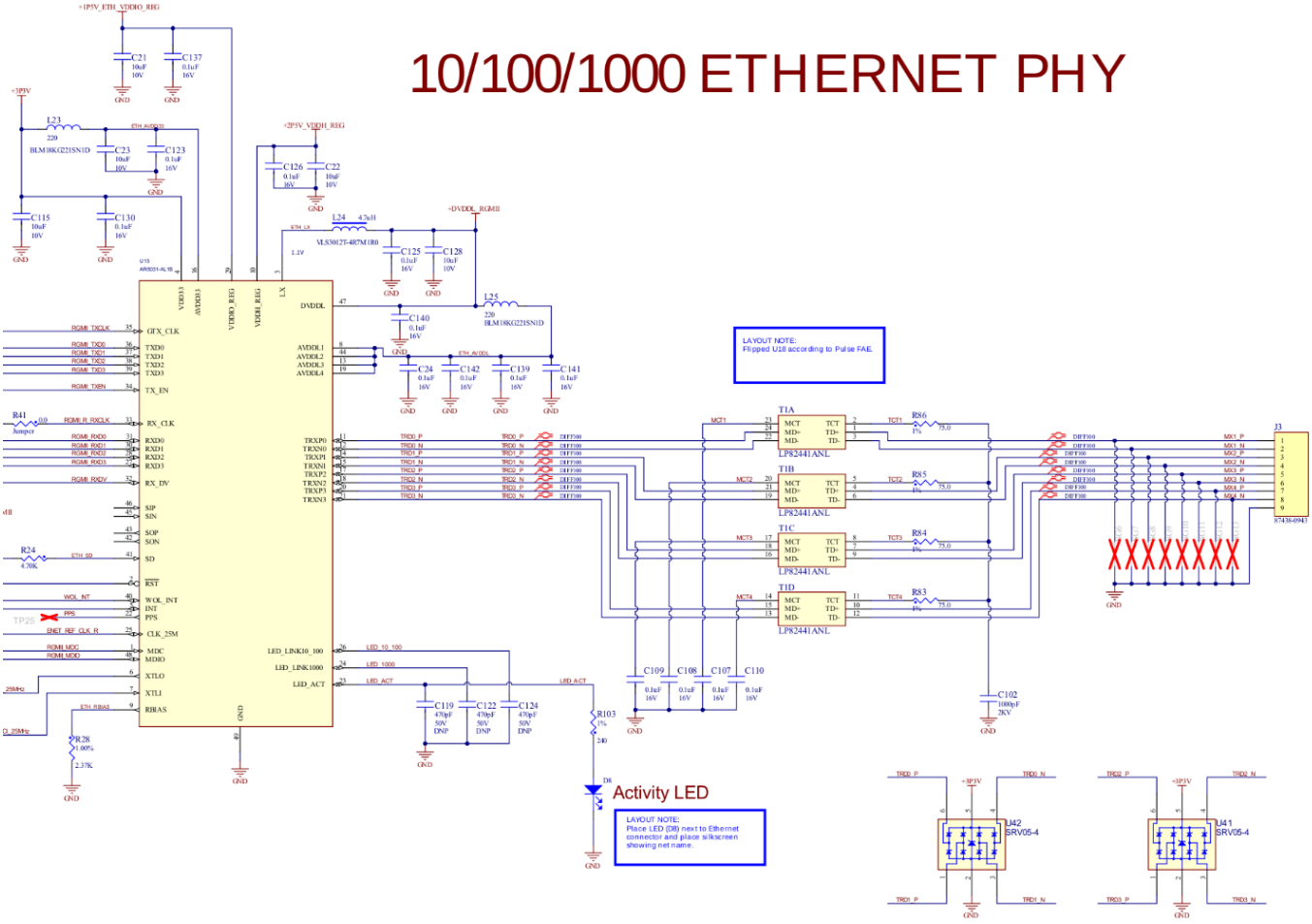
J3: Gigabit Ethernet Connection

9 pin Pico-SPOX, MOLEX Part Number: [87439-0900]

J3 Pinout

Pin	Signal
1	Bi-Directional Pair D1+
2	Bi-Directional Pair D1-
3	Bi-Directional Pair D2+
4	Bi-Directional Pair D2-
5	Bi-Directional Pair D3+
6	Bi-Directional Pair D3-
7	Bi-Directional Pair D4+
8	Bi-Directional Pair D4-
9	Ethernet Shield/GND

Gigabit Ethernet Schematic



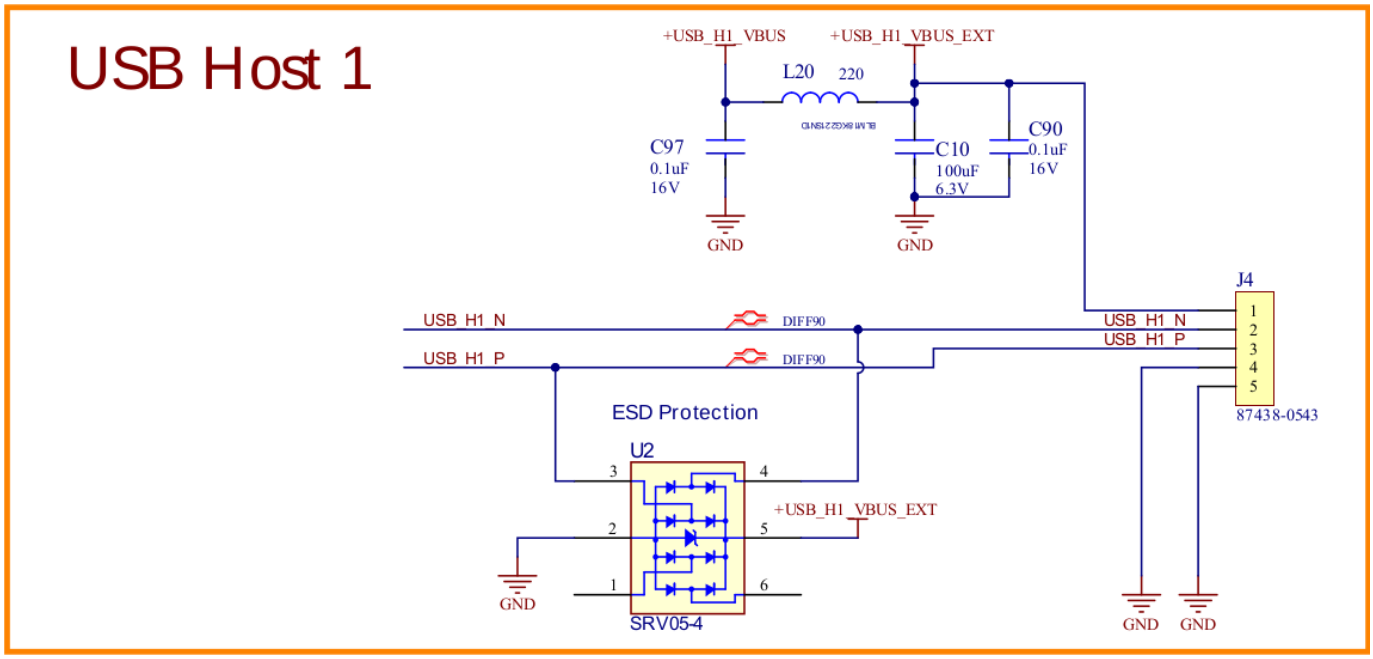
J4: USB 2.0 Host Port Connection

5 pin Pico-SPOX, MOLEX Part Number: [87439-0500]

J4 Pinout

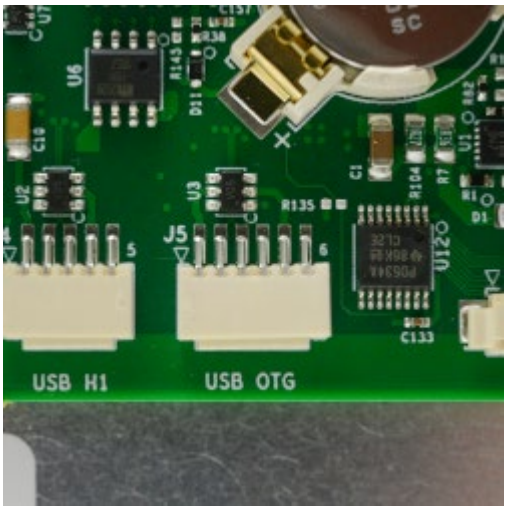
Pin	Signal
1	+5V
2	Data -
3	Data +
4	GND
5	GND shield

USB 2.0 Host Port Schematic



J5: USB 2.0 OTG Port Connection

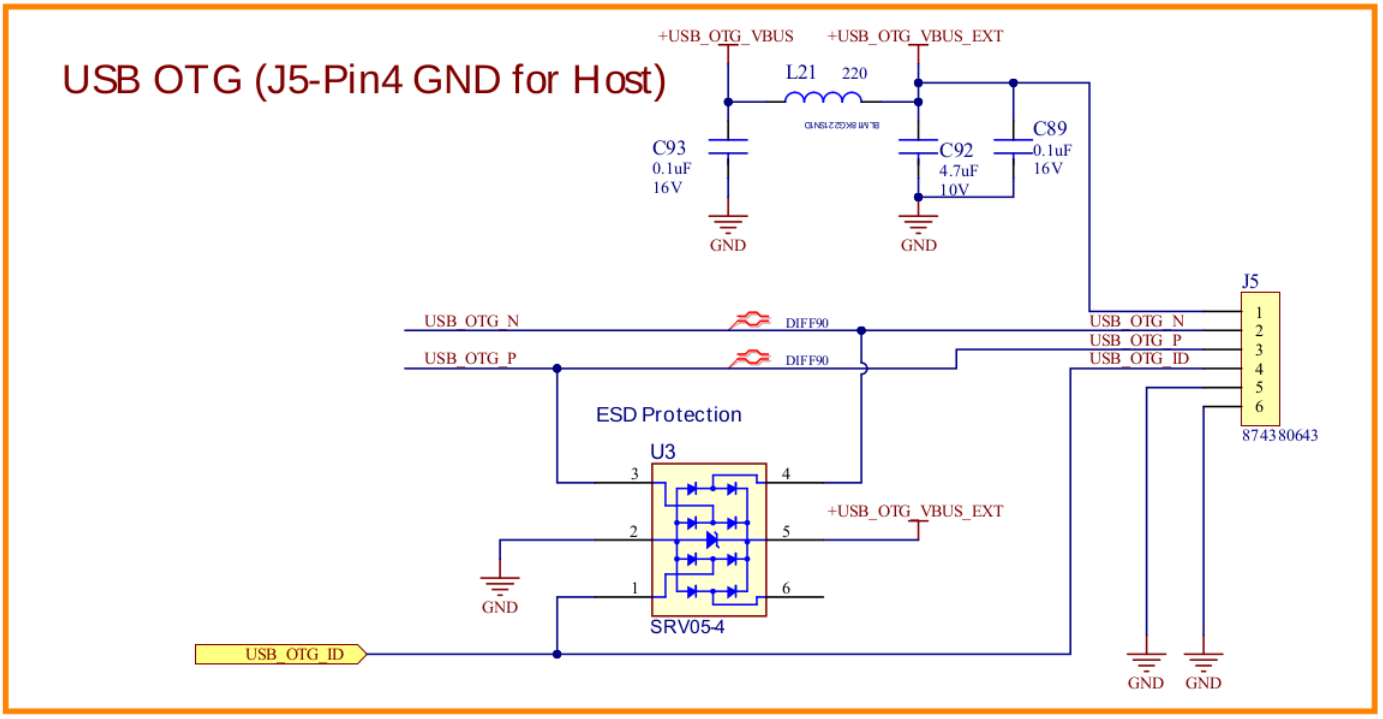
6 pin Pico-SPOX, MOLEX Part Number: [87439-0600]



J5 Pinout

Pin	Signal
1	+5V
2	Data -
3	Data +
4	USB_OTG_ID[68]
5	GND
6	GND Shield

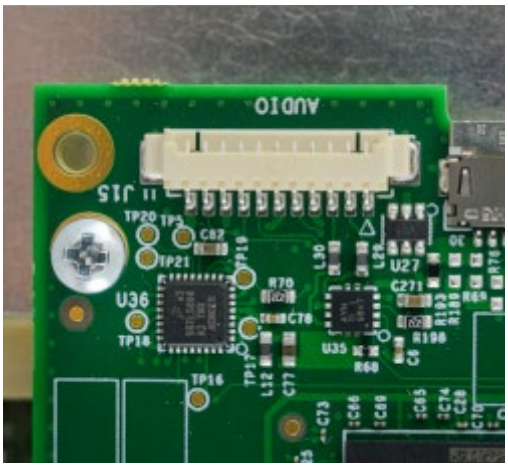
USB 2.0 OTG Port *Schematic*



[68] Ground to make J5 a USB Host port.

J15: Audio Connection

11 pin PicoBlade, MOLEX Part Number: [51021-1100]



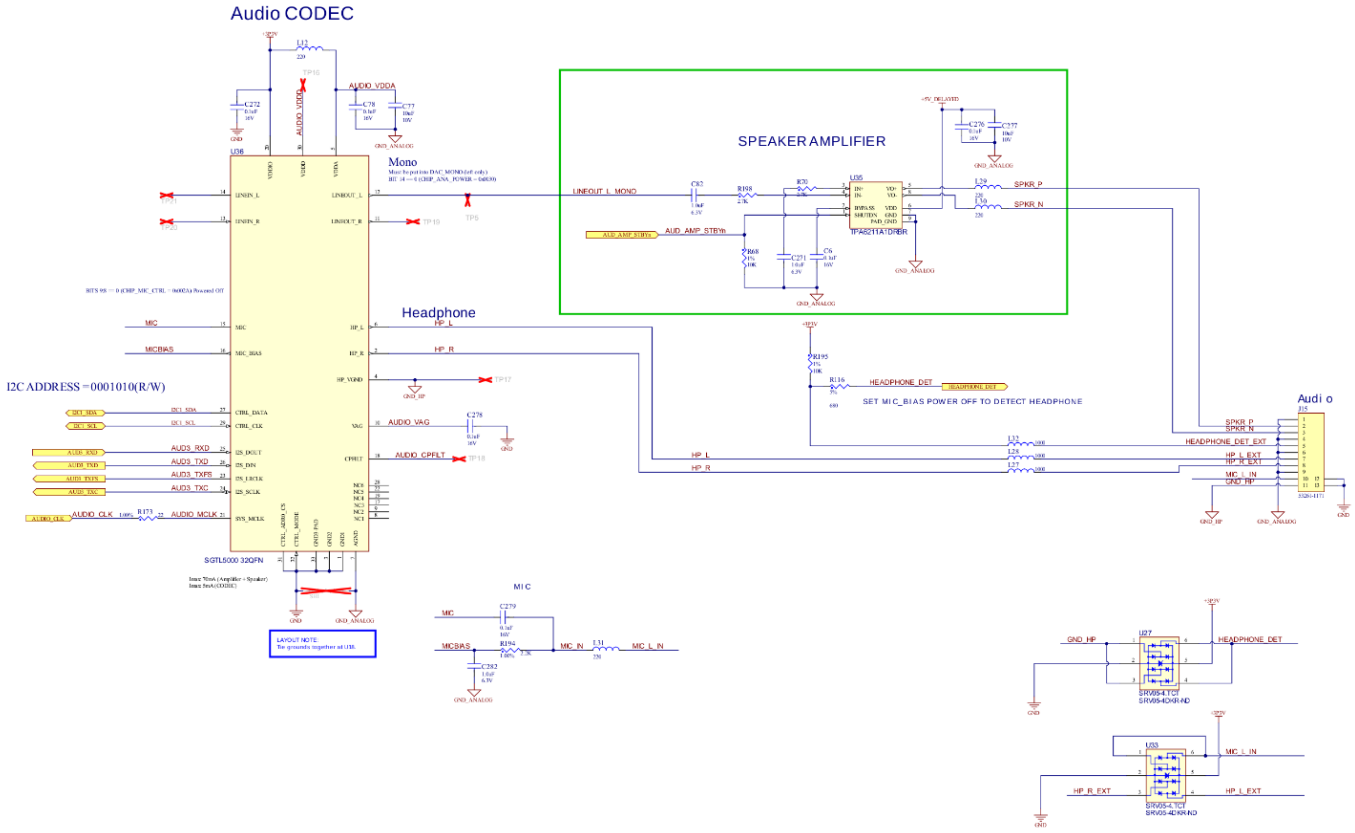
J15 Pinout

Pin	Signal
1	Analog GND
2	Speaker Connection +[69]
3	Speaker Connection –
4	Analog GND
5	External Headphone Detect
6	Analog GND
7	Headphone Output Left
8	Headphone Output Right
9	Analog GND
10	Microphone Input Left[70]
11	Headphone Virtual GND

[69] 3.1W into 3ohm load max, mono only.

[70] Mono only.

Audio Schematic



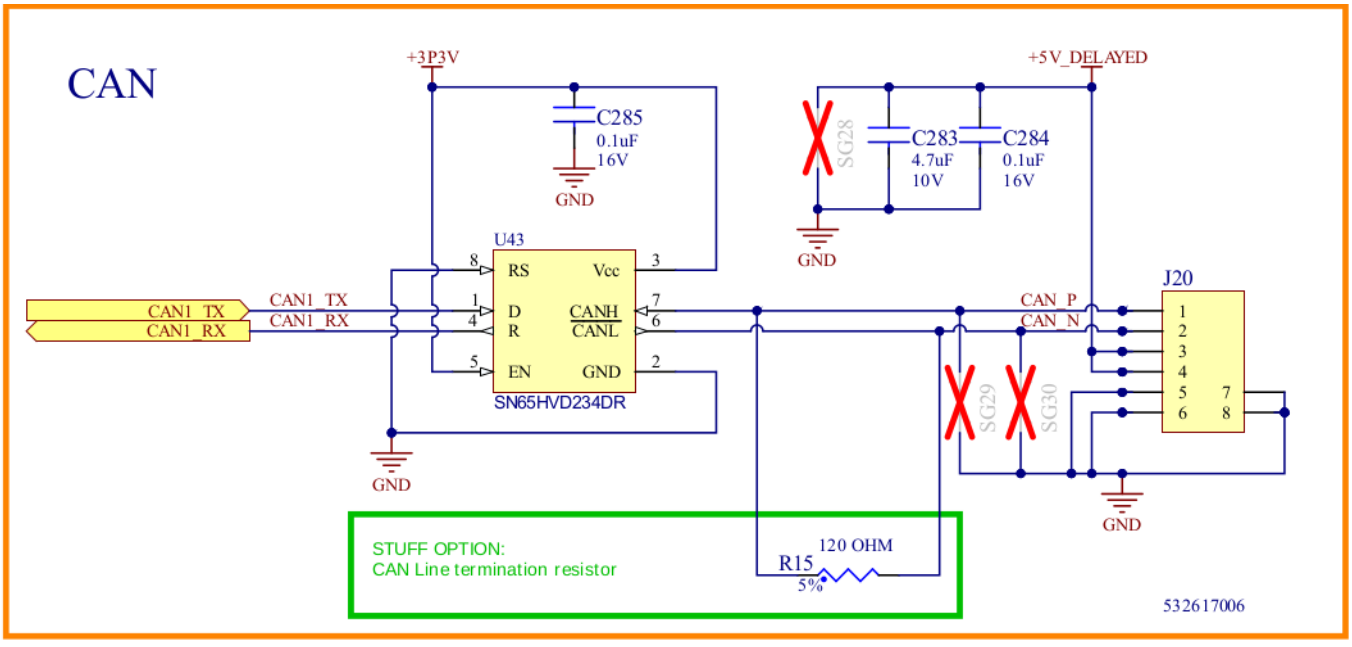
J20: CAN Bus

6 pin PicoBlade, MOLEX Part Number: [51021-0600]

J20 Pinout

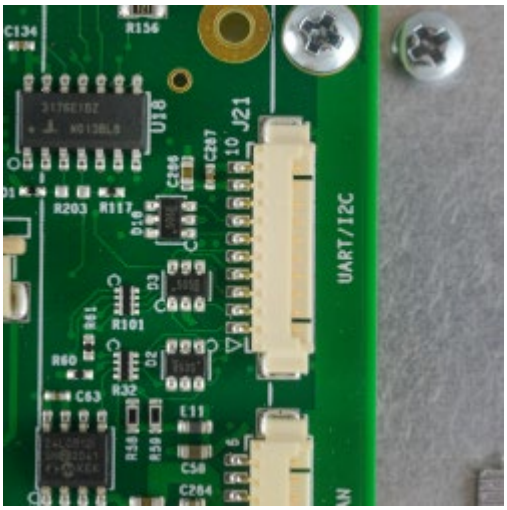
Pin	Signal
1	CAN +
2	CAN -
3	+5V
4	+5V
5	GND
6	GND

CAN Bus Schematic



J21: UART / I²C Bus / GPIO Connection

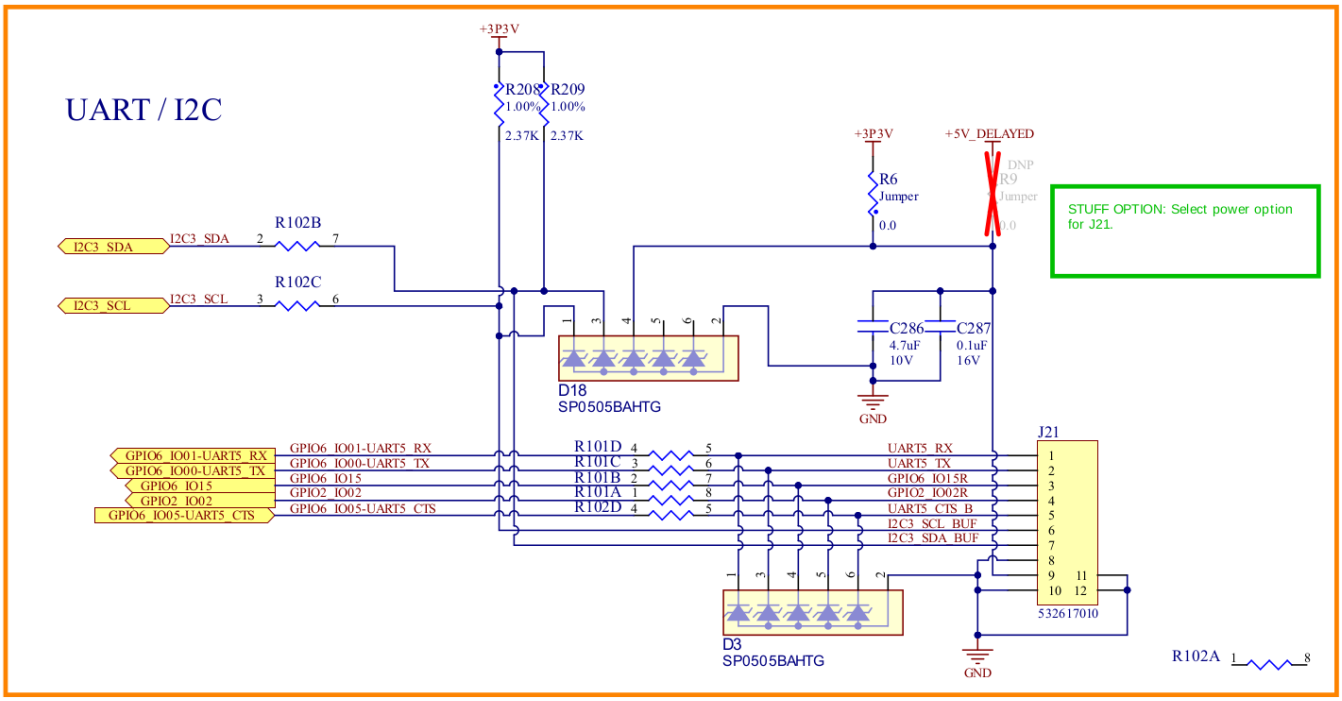
10 pin PicoBlade, MOLEX Part Number: [51021-1000]



J21 Pinout

Pin	Signal
1	UART5_RX
2	UART5_TX
3	GPIO6_IO15R
4	GPIO2_IO2R
5	UART5_CTS_B
6	I2C3_SCL_BUF
7	I2C3_SDA_BUF
8	GND
9	+3.3V
10	GND

UART / I²C Bus / GPIO Schematic



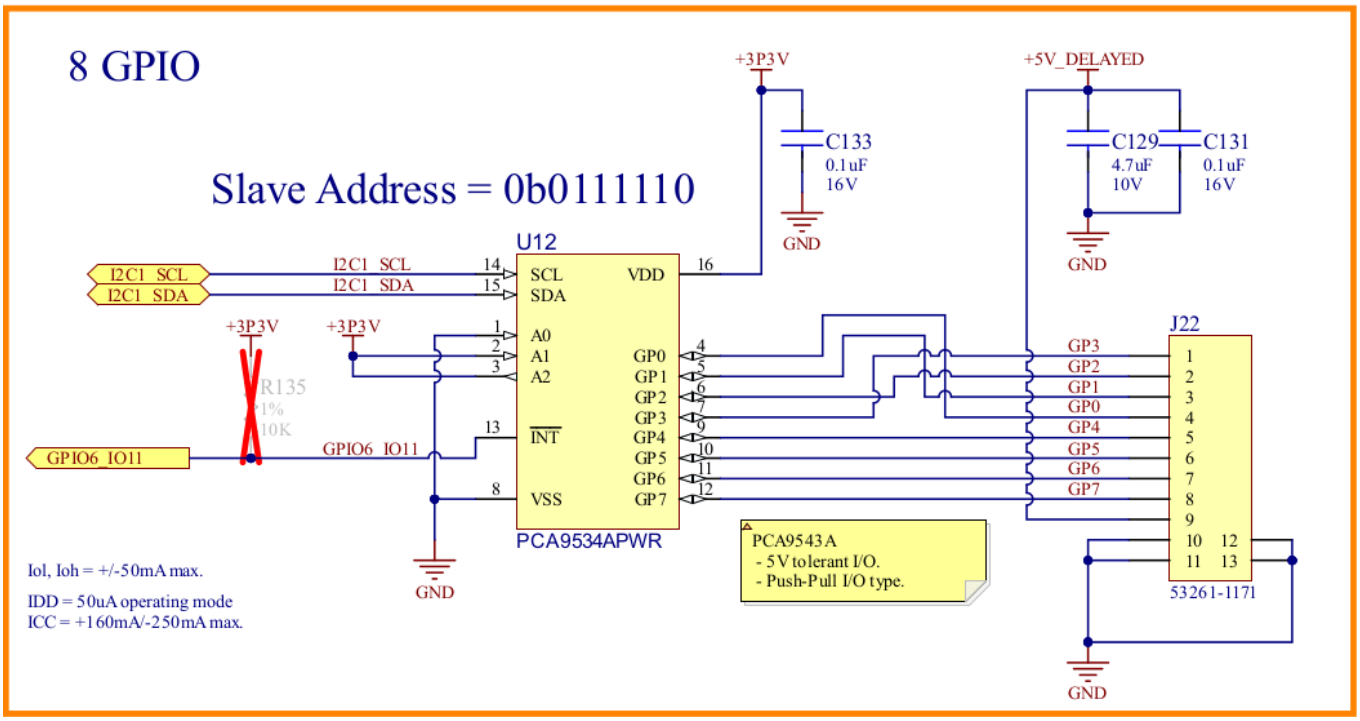
J22: GPIO Connection

11 pin PicoBlade, MOLEX Part Number: [51021-1100]

J22 Pinout

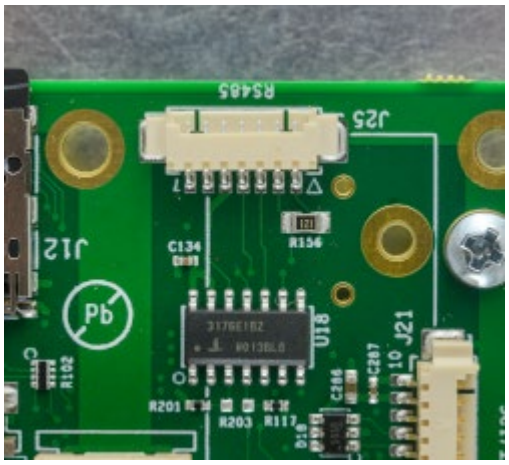
Pin	Signal
1	GP3
2	GP2
3	GP1
4	GP0
5	GP4
6	GP5
7	GP6
8	GP7
9	+5V
10	GND
11	GND

GPIO Schematic



J25: RS-485 Port Connection

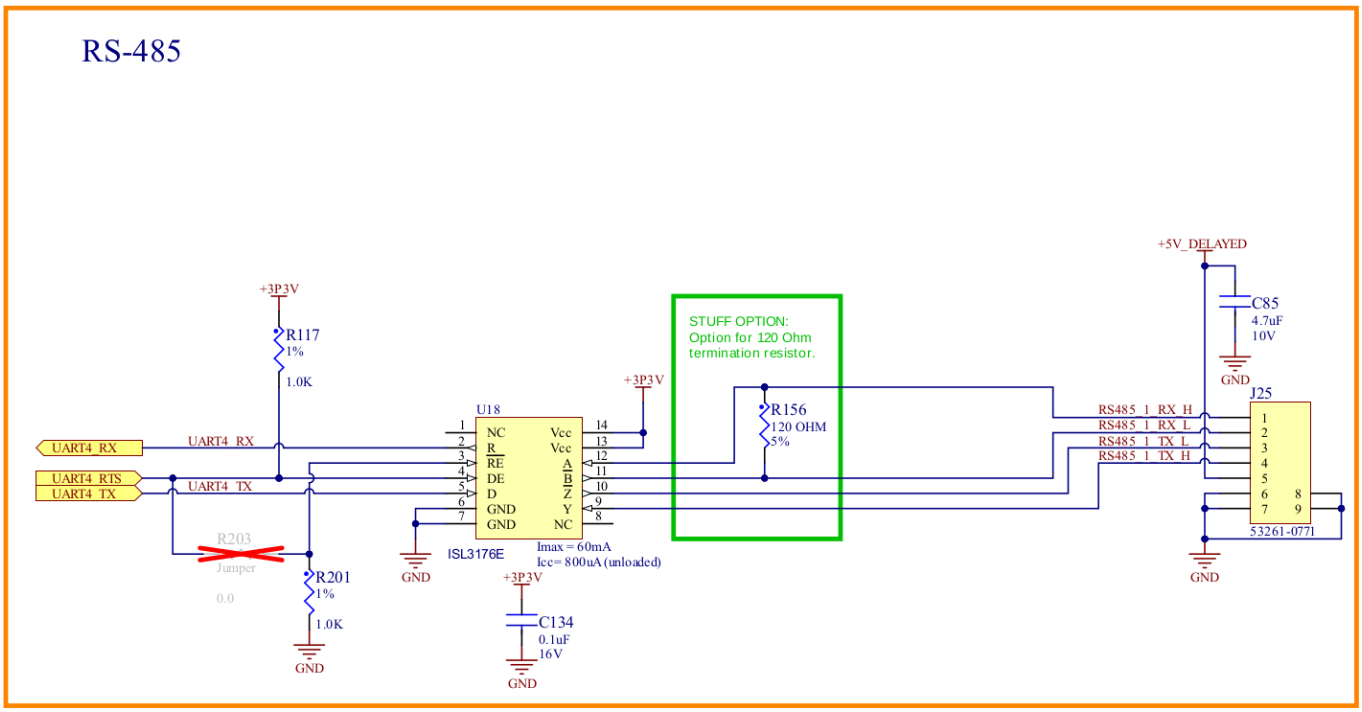
7 pin PicoBlade, MOLEX Part Number: [51021-0700]



J25 Pinout

Pin	Signal
1	Rx H
2	Rx L
3	Tx L
4	Tx H
5	+5V
6	GND
7	GND

RS-485 Port Schematic



G3BNG Controller Appendix

Specifications

Specification	Value
Processor	i.MX6DL
RAM	2GB DDR3
Flash	8GB eMMC
RS-232 Ports	1
RS-485 Ports	1 (Full Duplex only)
UART	1 CMOS Logic Level
Ethernet	1 Gigabit Port
CAN	1 Port at up to 1Mbps
USB 2.0	1 Host Port
USB 2.0 OTG	1 Host/Device Port
GPIOs	10
Audio	…
Temperature Range	…

Thermal Considerations

Reach Technology display modules have varying temperature ratings, predominantly determined by the display they use. Please refer to the panel specifications of the display module you use in the documents tab on your module page. While modules fall into varying temperature grades, ratings are not absolute and must be considered and tested at a system level.

For models with extended temperature ranges, the following considerations apply:

- To use SD card storage, ensure the card is high-quality from a name-brand manufacturer and is industrial or automotive temperature range qualified. Unpredictable system behavior or data loss can occur with low-quality or insufficiently rated storage. This consideration does not apply to controllers exclusively using onboard eMMC storage.
- If the controller is exposed to elevated temperatures, the CPU or GPU may throttle its internal clock if the core temperature (not ambient temperature) exceeds set thresholds. This may cause unwanted performance loss in designs that utilize GPU-intensive GUIs or extended periods of heavy CPU usage. Please get in touch with Reach Technology engineering if this is a concern.

- Additional thermal management measures may be needed for use in high-temperature environments, especially when exposed to hot, direct sunlight. Don't hesitate to get in touch with Reach Technology engineering for assistance.

Accessories

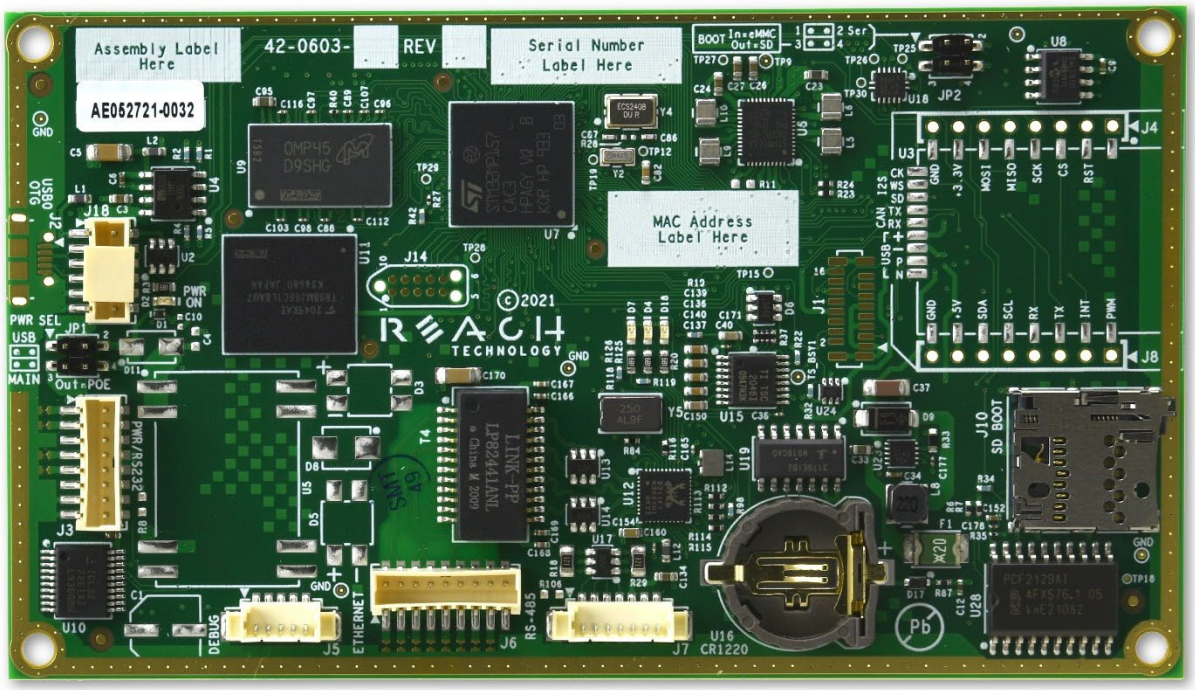
Part Number	Description
23-0160-18	Power - RS-232 "Y" Cable (5VDC)
23-0173-18	Power - RS-232 "Y" Cable (12VDC)
23-0161-72	Debug Interface Cable
23-0148-12	Ethernet Adapter Cable
23-0166-12	USB Host Cable
23-0167-12	USB OTG Host Cable
23-0168-12	USB OTG Device Cable
23-0144-10	CAN/GPIO Flying Leads
23-0145-10	RS-485 Flying Leads
23-0146-10	Audio Flying Leads
23-0149-10	I^2C/UART/GPIO Flying Leads
23-0172-12	Audio - Headphone Jack Adapter Cable

Find additional information on pricing and how to order accessories.

G3MSB Controller Hardware Manual

Reach Technology G3MSB embedded controllers support an LCD screen resolution of up to 1280 x 800 pixels. They allow you to build a custom user interface and utilize many built-in options to communicate with your product components. G3MSB is based on a high-speed, multi-core Arm processor and can support a variety of displays and touch sensors. You can use it in low-cost applications, without a display, or where unique I/O or sensor requirements require a product-specific daughter card.

It uses the STM32MP157 processor, a dual-core implementation of the Arm Cortex-A7. It features an auxiliary Cortex-M4 core for real-time monitoring and control tasks that can communicate directly with embedded Linux software on the primary Cortex-A7 cores. Take advantage of various I/O interfaces running low-cost connectors. Modules consist of single board controllers; most come with integrated touchscreen panels. Development Kits include everything you need to create a prototype quickly. When you are ready, move smoothly into production with off-the-shelf display modules that offer 10+ years of availability as a minimum.



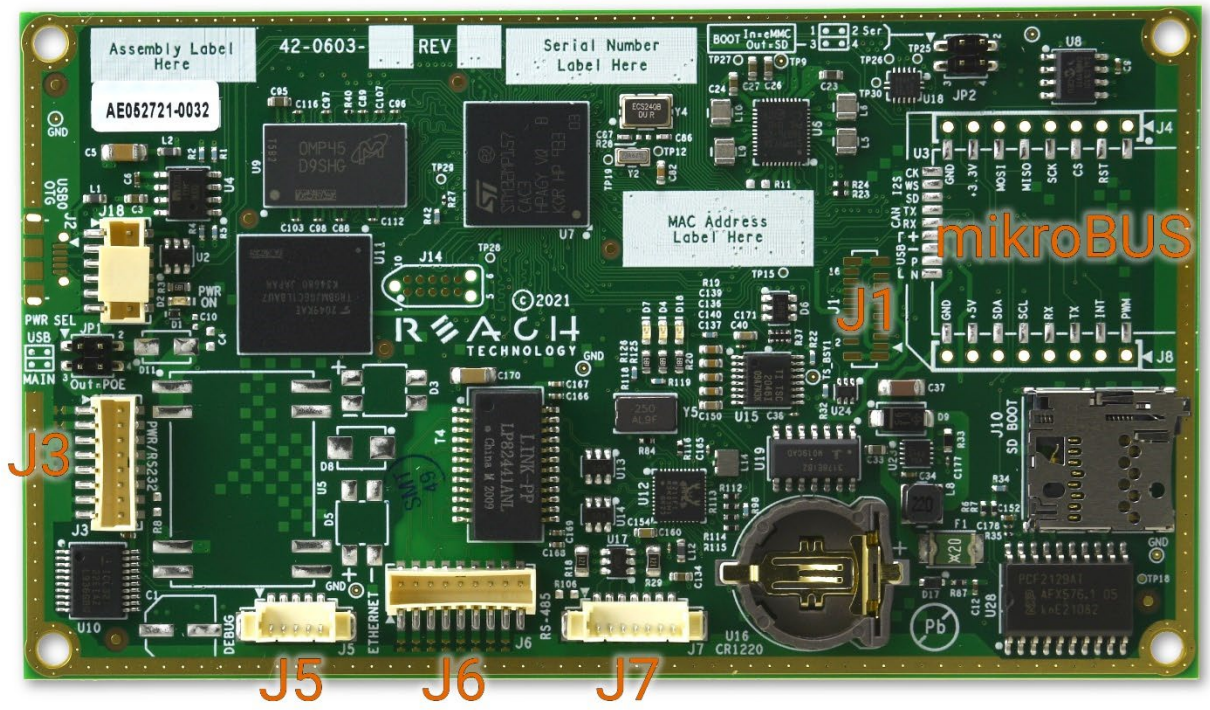
Controller Interfacing

The G3MSB controller contains a reduced set of I/O interfaces, for example, compared to the G3BNG controller board, discussed in detail in the following sections. The table below lists onboard connectors and pins, ports, and buses available on those connectors.

I/O Connectors

Connector	Port(s)
J3	Power, RS-232 Port
J5	Console (Debug) Port
J6	Gigabit Ethernet
J7	RS-485 Port
J4, J8	mikroBUS Pads
J18	USB 2.0 OTG Port

See location annotations in the image below showing the I/O connectors in the previous table.



Power

Connect DC power to the G3MSB controller and the touch panel via J3[71]. Although exact power consumption depends on peripherals and the touch panel, the package will consume 7W to 15W when the backlight is active.

Use a 5V, 3A DC power source to drive G3MSB controllers.

> IMPORTANT: The G3MSB controller has an option for Power over Ethernet (PoE), unstuffed by default. Use J3 to power the module. If you want to use PoE, contact Technical Support.

Console (Debug) Port

Access the controller Debug Interface via J5[72]. This RS-232 level serial port runs at 115.2k Baud, 8-bit, no parity, and 1 stop bit. During product development, you **always** need to have this port connected. You will need a terminal emulator program or shell that is compatible with ANSI color escape sequences. Examples:

- For Windows, Tera Term or PuTTY[73]
- For Linux, Picocom in a bash shell

RS-232 Port

A standard RS-232[74] port is also available on J3[75]

Gigabit Ethernet

A standard 1000baseT network port is open on J6[76].

USB 2.0 OTG Port

The OTG port is supported in host mode. Contact Technical Support if you need to use it in device mode.

Connect standard USB 1.0 and USB 2.0 peripherals[77] to the USB host port (J18[78]). If you want to use the G3MSB as a USB device[79] rather than as a host, you can use this port as it can be either a host[78] or a device port[78].

RS-485 Port

Connect RS-485 peripherals to J7[80]. The default mode is full duplex.

---

[71] Use Reach Technology "Y" cable, PN 23-0160-18 for 5 VDC.
[72] Use Reach Technology debug cable, PN 23-0161-72, if you do not wish to build custom cables.
[73] We recommend PuTTY. It supports both serial and network connections and Windows 10/11 and Linux.
[74] Standard RS-232 voltage levels.
[75] Use Reach Technology "Y" cable, PN 23-0173-18, if you do not wish to build custom cables.
[76] Use Reach Technology Ethernet cable, PN 23-0148-12, if you do not wish to build custom cables.
[77] Some touch panels with HID touch controllers use the USB host port. If you need additional USB 2.0 ports, put a hub on your I/O card and connect it here.

[78] Use Reach Technology USB Host cable, PN 23-0167-12, if you do not wish to build custom cables.
[79] For example, as a Linux Network Gadget for peer-to-peer USB networking.

Connector Pinouts and Part Numbers

The following sections provide the pinout for each connector and part number for the required mating plug[81].

J3: Power/RS-232 Connection

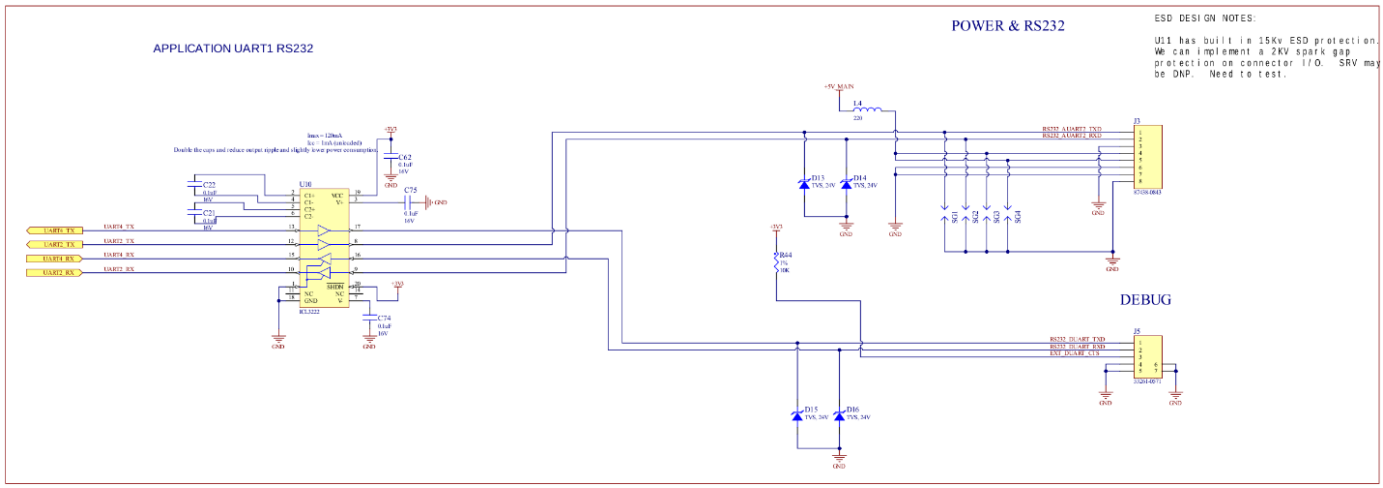
8 pin Pico-SPOX, MOLEX Part Number: 87439-0800

J3 Pinout

Pin	Signal
1	TxD Output ESD Protected[82]
2	RxD Input ESD Protected[83]
3	GND (Power and Communications)
4	VIN 5VDC Input Powers the Board
5	VIN 5VDC Input Powers the Board
6	GND (Power and Communications)
7	GND (Power and Communications)
8	GND (Power and Communications)

[81] The mikroBUS pads are standard 0.1" pitch through-hole pads. If using a standard, off the shelf, Mikroe Click board, you can either solder it directly to the G3MSB controller or use standard 8 pin, single row socket headers.
[82] For ESD protection details, refer to the specifications of the ICL3222E transceiver chip.
[83] For ESD protection details, refer to the specifications of the ICL3222E transceiver chip.

Debug Interface, Power, RS-232 Schematic

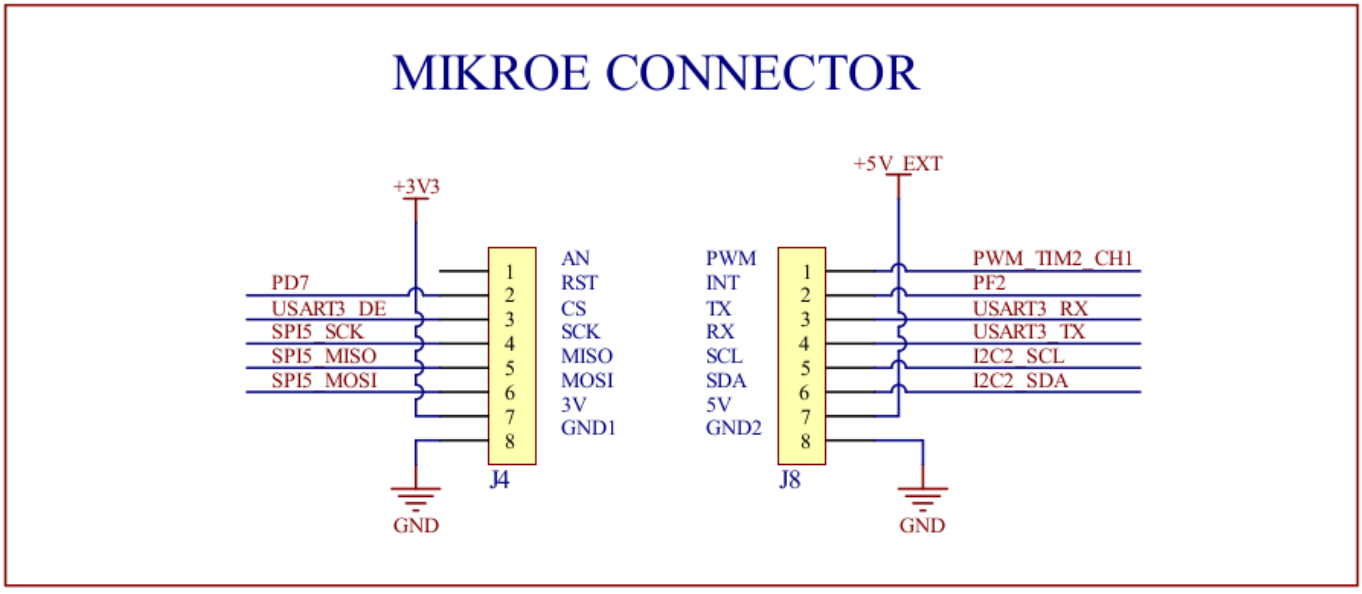


J4: mikroBUS Connection (SPI, RST)

J4 Pinout

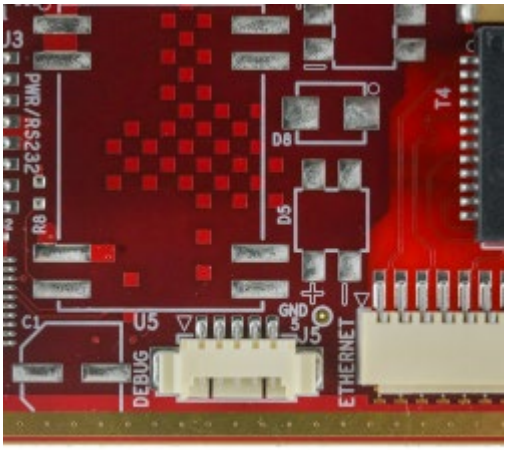
Pin	Signal
1	NC
2	RST (Reset)
3	CS (Chip Select)
4	SCK (SPI Clock)
5	MISO (SPI Master in Slave Out)
6	MOSI (SPI Master Out Slave In)
7	+3.3V
8	GND

mikroBUS Connection (SPI, RST) Schematic



J5: Debug Interface Connection

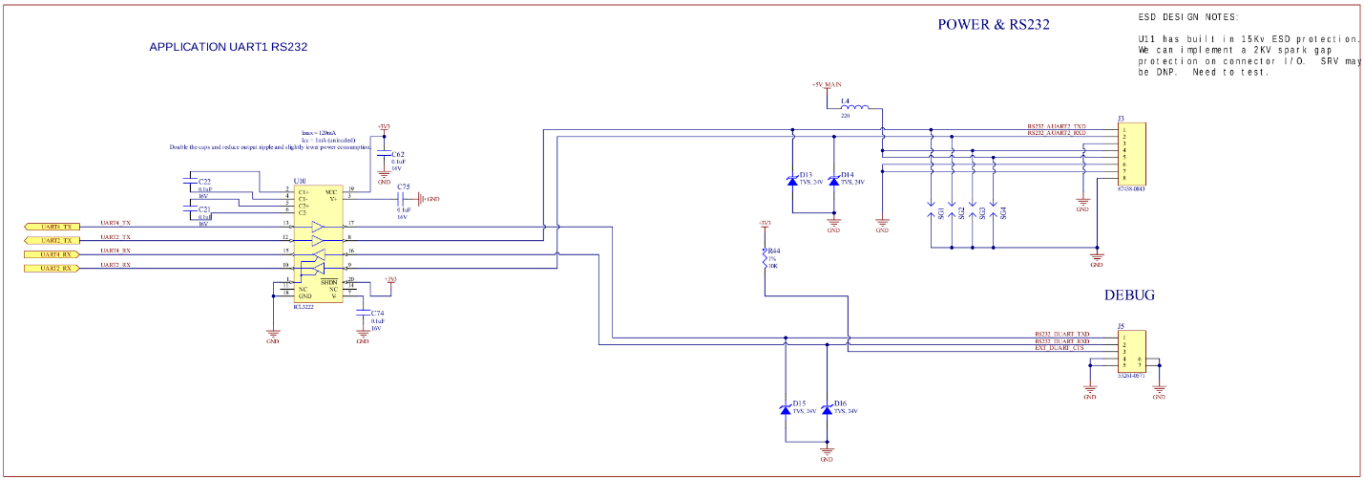
5 pin PicoBlade, MOLEX Part Number: [51021-0500]



J5 Pinout

Pin	Signal
1	TxD Output ESD Protected[84]
2	RxD Input ESD Protected[85]
3	CTS
4	GND
5	GND

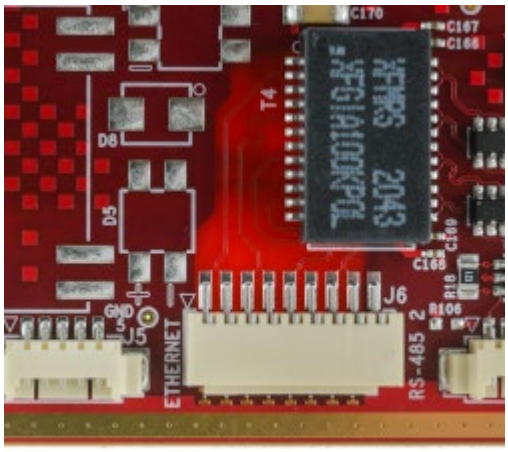
Debug Interface, Power, RS-232 Schematic



[84] Internal ESD protection as specified for the ICL3222E device, +/- 16 KV.
[85] Internal ESD protection as specified for the ICL3222E device, +/- 16 KV.

J6: Gigabit Ethernet Connection

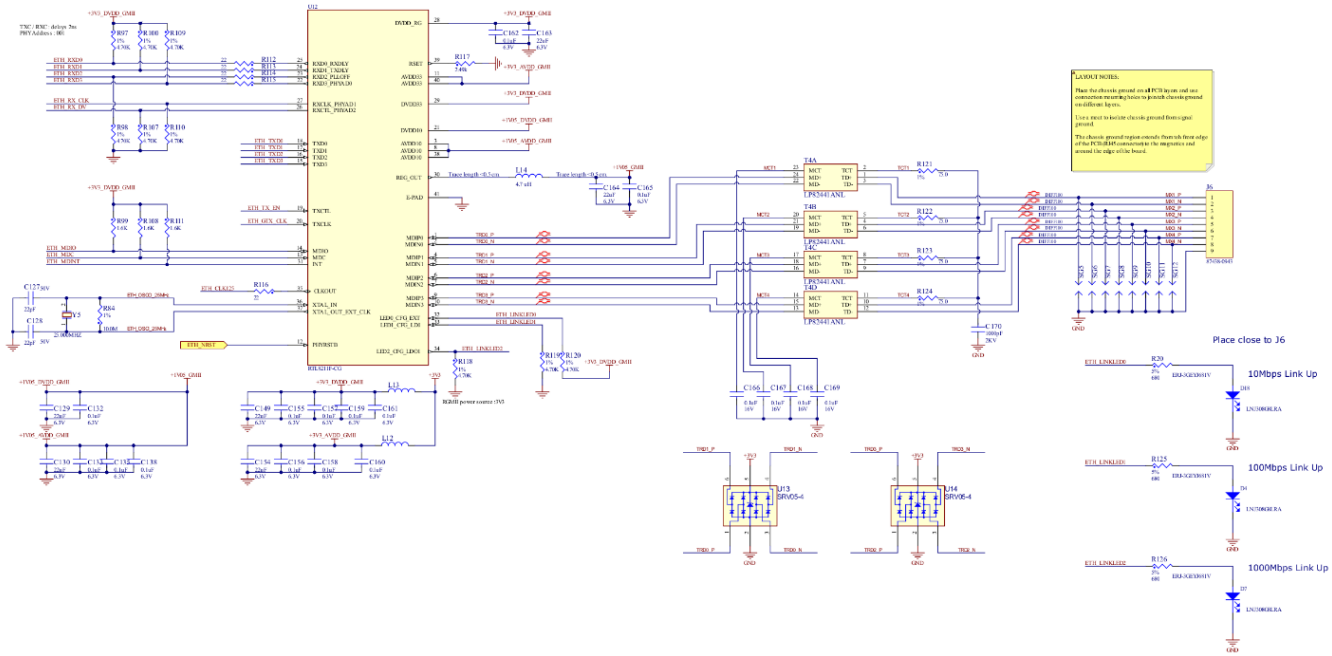
9 pin Pico-SPOX, MOLEX Part Number: [87439-0900]



J6 Pinout

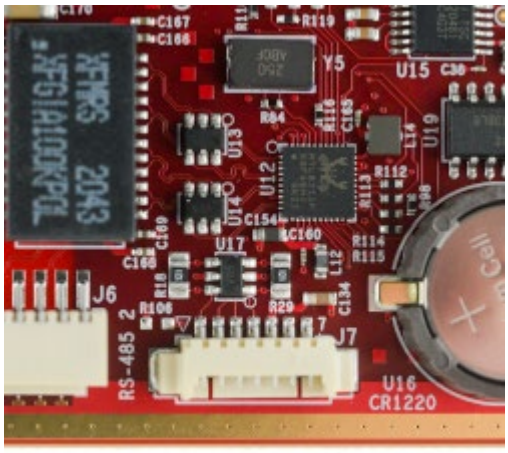
Pin	Signal
1	Bi-Directional Pair D1+
2	Bi-Directional Pair D1-
3	Bi-Directional Pair D2+
4	Bi-Directional Pair D2-
5	Bi-Directional Pair D3+
6	Bi-Directional Pair D3-
7	Bi-Directional Pair D4+
8	Bi-Directional Pair D4-
9	Ethernet Shield/GND

G3MSB Gigabit Ethernet Schematic



J7: RS-485 Port Connection

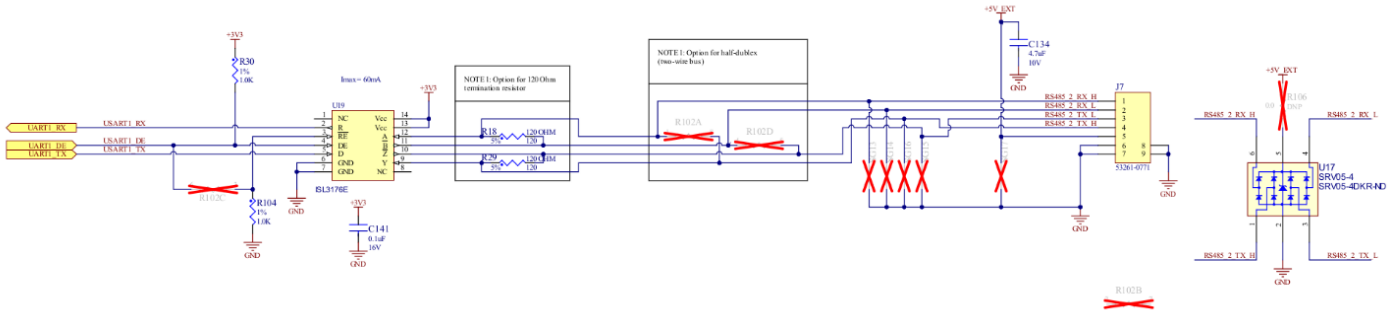
7 pin PicoBlade, MOLEX Part Number: [51021-0700]



J7 Pinout

Pin	Signal
1	Rx H
2	Rx L
3	Tx L
4	Tx H
5	+5V
6	GND
7	GND

RS-485 Port Schematic



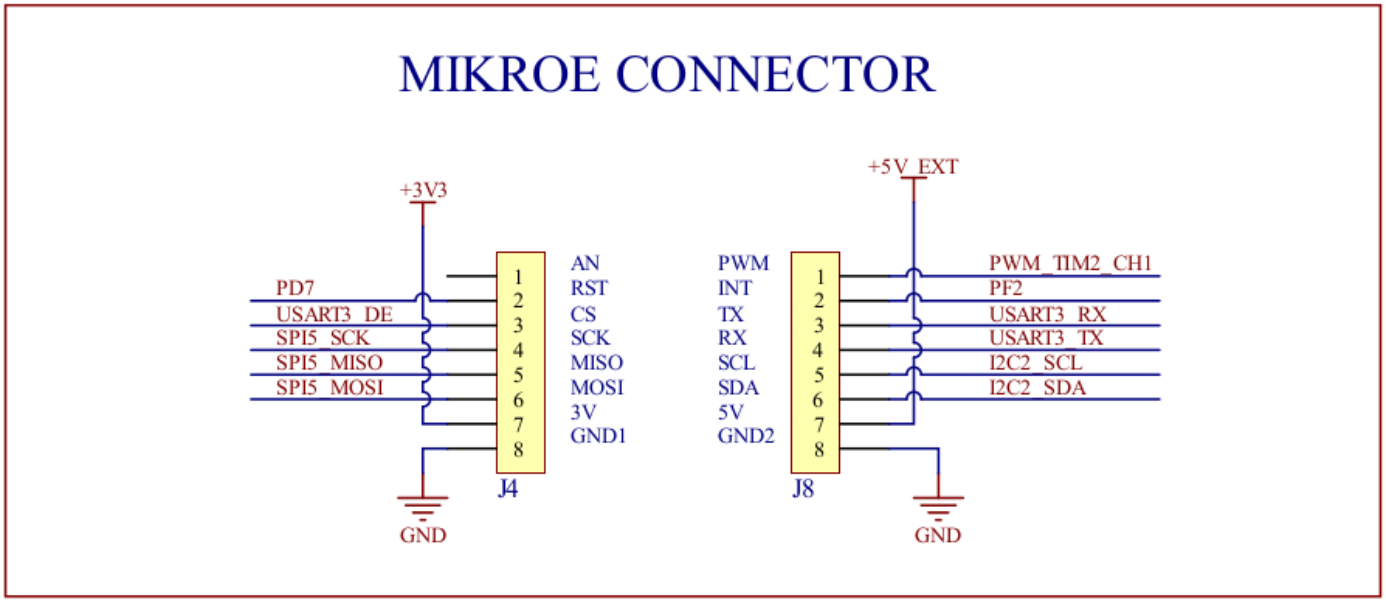
J8: mikroBUS Connection (UART, I2C, PWM, INT)

J8 Pinout

Pin	Signal
1	PWM (Pulse Width Modulation)
2	INT (Interrupt)
3	TX (UART Transmit)
4	RX (UART Receive)

Pin	Signal
5	SCL (I^2C Clock)
6	SDA (I^2C Data)
7	+5V
8	GND

mikroBUS Connection (UART, I2C, PWM, INT) Schematic



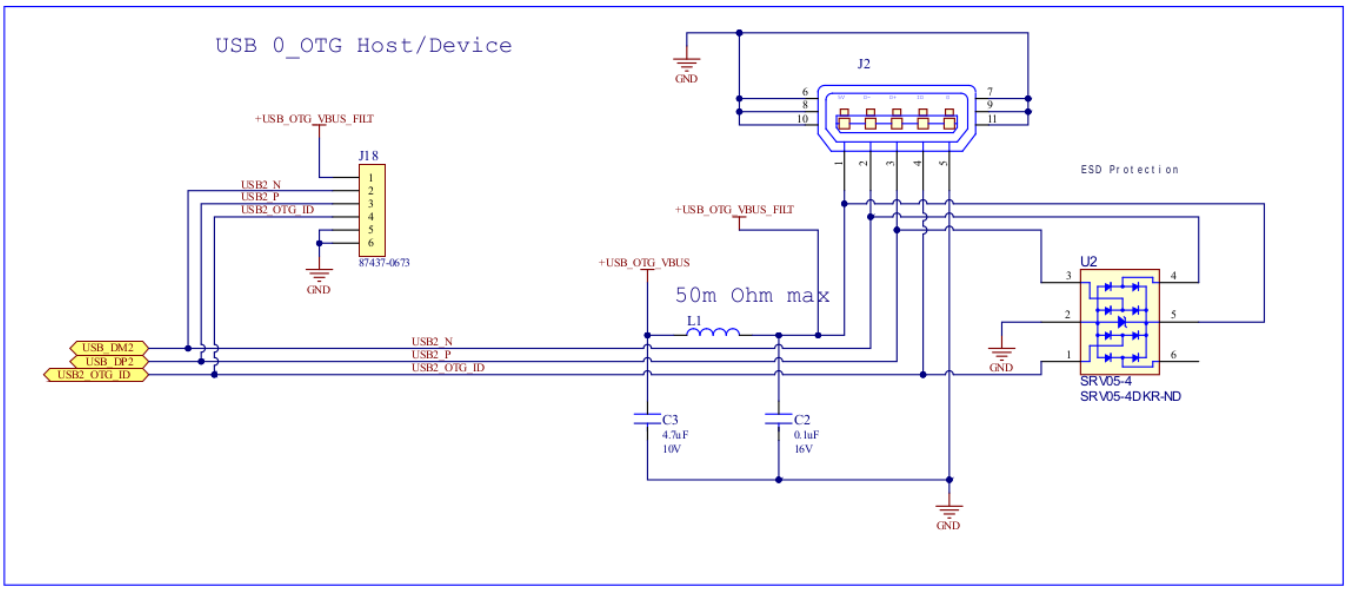
J18: 2.0 OTG Port Connection

6 pin Pico-SPOX, MOLEX Part Number: [87439-0600]

J18 Pinout

Pin	Signal
1	+5V
2	Data -
3	Data +
4	USB_OTG_ID[86]
5	GND
6	GND shield

USB 2.0 OTG Port Schematic



[86] Ground to make J18 a USB Host port.

Controller Specifications

Specifications

Specification	Value
Processor	STMP32MP157
RAM	512MB DDR3
Flash	8GB eMMC
RS-232 Ports	1
RS-485 Ports	2 (Full Duplex Only)
UART	1 CMOS Logic Level
Ethernet	1 Gigabit Port
CAN	1 Port at up to 1Mbps
USB 2.0	2 Host Ports
USB 2.0 OTG	1 Host/Device Port
GPIOs	10
Temperature Range	...

Thermal Considerations

Reach Technology display modules have varying temperature ratings, predominantly determined by the display they use. Please refer to the panel specifications of the display module you use in the documents tab on your module page. While modules fall into varying temperature grades, ratings are not absolute and must be considered and tested at a system level.

For models with extended temperature ranges, the following considerations apply:

- To use SD card storage, ensure the card is high-quality from a name-brand manufacturer and is industrial or automotive temperature range qualified. Unpredictable system behavior or data loss can occur with low-quality or insufficiently rated storage. This consideration does not apply to controllers exclusively using onboard eMMC storage.
- If the controller is exposed to elevated temperatures, the CPU or GPU may throttle its internal clock if the core temperature (not ambient temperature) exceeds set thresholds. This may cause unwanted performance loss in designs that utilize GPU-intensive GUIs or extended periods of heavy CPU usage. Please get in touch with Reach Technology engineering if this is a concern.

- Additional thermal management measures may be needed for use in high-temperature environments, especially when exposed to hot, direct sunlight. Don't hesitate to get in touch with Reach Technology engineering for assistance.

Accessories

Accessories come with Development Kits. Some customers need additional items for prototyping. In that case, you can purchase items by display module controller type from our website.

Warranty and Software License Agreement

Warranty

See our One Year Limited Hardware Warranty for more information.

Software License Agreement

See the Software License Agreement for more details.

Troubleshooting

When problems arise, we are here to help. Do not hesitate to contact our technical support team (503-675-6464, techsupport@reachtech.com).

Appendix A: A Quick Discussion of Qt Property Maps

You can use Qt's property system, introduced with the **`Q_PROPERTY`** macro, to define class properties that can be read, written, and observed, much like regular member variables. They come with additional features useful in a GUI environment and for other tasks. This system is deeply integrated with the Qt Meta-Object System.

Let us break down the aspects of this sub-system:

- **Definition with `Q_PROPERTY`**: Use the **`Q_PROPERTY`** macro to define property class declarations.

```
class MyClass : public QObject {
        Q_OBJECT
        Q_PROPERTY(int myProperty READ getMyProperty WRITE setMyProperty NOTIFY
myPropertyChanged)
    public:
      int getMyProperty() const;
      void setMyProperty(int value);
    signals:
       void myPropertyChanged();
};

    int index = meta->indexOfProperty("myProperty");
    if (index != -1) {
       QMetaProperty prop = meta->property(index);
       QVariant value = prop.read(object);
    }
```

- **Binding with UI:** You can easily bind properties to UI elements in Qt's QML language, allowing you to synchronize your UI with underlying data models without much boilerplate code.
- **Animations:** Qt's animation framework can animate properties to allow for smooth transitions between property values over time.
- **Advantages:**
    - **Standardization:** Properties offer a standardized way to define object attributes to help create uniform APIs, especially in UI components.
    - **Data Binding:** In QML, properties can be easily bound, ensuring the UI reflects underlying data changes without much manual intervention.
    - **Dynamic Property Access:** You can dynamically get and set properties using **`property()`** and **`setProperty()`** methods on **`QObject`** without knowing the actual type of the object.
- **Limitations:**
    - **Performance**: Direct member access is faster than property access due to the overhead of the meta-object system.
    - **C++ Only**: The property system is specific to Qt's C++ libraries.

The Qt property system is a robust mechanism to enhance C++ classes, providing a rich set of features that are especially relevant in GUI development and dynamic object handling. While there is a slight overhead compared to plain C++ member variables, the benefits of flexibility, introspection, and ease of binding usually outweigh the costs.

Appendix B: Updating the Embedded Linux OS

This section covers upgrading the embedded Linux image on the G3 in the field.

Over-the-Air (OTA) Updates

> NOTE: The OTA process is only available over USB. Connecting to a mender server - either remote or in the cloud is NOT supported.

OTA updates simplify and minimize the risk of rendering a device irretrievably inoperable following a device update. A popular OTA update strategy, and the one that Mender uses, is using A/B root filesystem partitions. The example below shows that the A/B rootfs partitions are `/dev/mmcblk0p5` and `/dev/mmcblk0p6`.

```
root@stm32mp1-g3-sd:~# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
mmcblk0     179:0    0  7.4G  0 disk
 -mmcblk0p1 179:1    0  256K  0 part
 -mmcblk0p2 179:2    0  256K  0 part
 -mmcblk0p3 179:3    0    2M  0 part
 -mmcblk0p4 179:4    0   16M  0 part /uboot
 -mmcblk0p5 179:5    0  1.6G  0 part /
 -mmcblk0p6 179:6    0  1.6G  0 part
 `-mmcblk0p7 179:7    0  4.2G  0 part /data
```

The device stores two copies of the OS rootfs. During a rootfs OTA update, the new rootfs is loaded to the inactive partition and then booted into after the device is rebooted to ensure that the device always has at least one working copy of the OS. Data that needs to persist between OTA updates is strored in the writable `/data` partition, including any custom application updates.

At a high level, all OTA update client software does the following:

- Downloads or streams a zip/tar/bzip file that contains the necessary files to perform an update. The OTA artifact could include bootloaders, firmware binaries for attached devices, entire operating system root filesystems, and more.
- Unpacks the updated artifact to the correct location. If it's a rootfs update, it unpacks the artifact to the rootfs partition not currently in use. Application updates should be installed to the R/W `/data` partition to persist between rootfs OTA updates.
- Runs a script (usually Bash) to perform other necessary operations like copying files from the old rootfs to the new rootfs partition.
- If necessary, it updates the bootloader environment. During a rootfs update, the bootloader environment is always updated to boot into the newly updated rootfs.
- Reboots the device or restarts the application.

To install an OTA update artifact:

```
mender install </path/to/artifact.mender>

reboot
```

After rebooting, if the update is successful, run “Mender commit” to make the update persistent. If you do not commit the update, Mender will roll back to the previous update artifact after a power cycle. If the update is unsuccessful but the OS still booted, run “Mender rollback” to force a rootfs partition roll back. If the update is a rootfs artifact, the device will boot into the newly updated rootfs partition. To verify by hand that the device switched rootfs partitions, run “lsblk” or “cat /proc/cmdline” before and after rebooting and make a note of which partition the “/” root directory is mounted.

We recommend installing local OTA update artifacts with a USB thumb drive instead of copying the artifact to “/data” directory.

For more in-depth information about Mender, visit https://docs.mender.io/3.0.

Creating and Updating the G3 in the Field

Signing Mender Artifacts

1. Create your public and private keys:

   openssl ecparam -genkey -name prime256v1 -out private-and-params.key

   openssl ec -in private-and-params.key -out private.key

   openssl ec -in private-and-params.key -pubout -out public.key

2. Copy the public key to the device:

   scp public.key root@<IP_ADDRESS>:/data/mender/public.key

3. Configure Mender to verify artifacts with the public key by adding the “ArtifactVerifyKey” to the mender configuration file “/data/mender/mender.conf`”:

   ```
   {
      "RootfsPartA": "/dev/mmcblk0p5",
      "RootfsPartB": "/dev/mmcblk0p6",
      "ArtifactVerifyKey": "/data/mender/public.key"
   }
   ```

4. Sign and verify your Mender artifact using the private key:

   mender-artifact sign artifact.mender -k private.key -o artifact-signed.mender

   mender-artifact validate artifact-signed.mender -k public.key

5. To test if Mender is indeed verifying the artifact's signature, try installing your signed Mender artifact. You should see this message in the output log:

   INFO[0000] Installer: authenticated digital signature of artifact

To verify Mender will not install non-signed Mender artifacts, try installing a non-signed Mender artifact after configuring the “ArtifactVerifyKey”. You should see the error message below:

ERRO[0000] Reading headers failed: installer: failed to read Artifact: reader: expecting signed artifact, but no signature file found

ERRO[0000] installer: failed to read Artifact: reader: expecting signed artifact, but no signature file found

For more information, please read the official Mender documentation visit docs.mender.io/artifact-creation/sign-and-verify.