AN13799

Using Lightweight TCP/IP on Cortex-M core of i.MX 8MM Processor

Rev. 1 — 25 April 2023 Application note

Document information

Information	Content
Keywords	AN13799, lwIP, i.MX 8MM, Cortex-M
Abstract	This document describes how to use the lightweight TCP/IP (lwIP) stack on the Arm Cortex-M core of the i.MX 8M Mini processor, running without an operating system (bare-metal) or with FreeRTOS.

1 Introduction

This document describes how to use the lightweight TCP/IP (lwIP) stack on the Arm Cortex-M core of the i.MX 8M Mini processor, running without an operating system (bare-metal) or with FreeRTOS.

Lightweight TCP/IP (lwIP) is an open source TCP/IP stack. The main purpose of this stack implementation is to reduce resource usage, while still having a full-scale TCP. This makes lwIP suitable for use in embedded systems, which have limited memory size.

1.1 Software environment

A host PC running a recent version of Ubuntu is assumed.

- Install the Real Time Edge Software 2.4.0 environment.
- Build the Real-time Edge Image (using Yocto environment). For more details on how to do that, see *Section 5.5* from the *Real-Time Edge Software User Guide* (document REALTIMEEDGEUG).
- Write the resulting `nxp-image-real-time-edge-imx8mm-lpddr4-evk.wic.bz2` complete image (this can be found in the `<yocto_build_directory>/tmp/deploy/images/imx8mm-lpddr4-evk/` directory) on an SD card.

Note: *Check your card reader partition and replace `sd<x>` with your corresponding partition.*

```
$ bzcat nxp-image-real-time-edge-imx8mm-lpddr4-evk.wic.bz2 | sudo dd of=/dev/
sd<x> bs=1M
```

1.2 Hardware setup and equipment

- Development kit: NXP i.MX 8MM EVK LPDDR4
- Micro SD card: SanDisk Ultra 32 GB Micro SDHC I Class 10 is used for the current experiment
- Micro-USB cable for the debug port
- Ethernet cable

2 Prerequisites

- Install CMake:

```
$ sudo apt-get install cmake
$ # Check the version >= 3.0.x
$ cmake --version
```

- Install the GCC Arm embedded toolchain:
  Note: *Install version 10.3, because the newest 12.2 version does not work properly at the moment of writing this document.*

```
$ mkdir ~/gcc_compiler
$ cd ~/gcc_compiler
$ wget -v https://developer.arm.com/-/media/Files/downloads/gnu-
rm/10.3-2021.10/gcc-arm-none-eabi-10.3-2021.10-x86_64-linux.tar.bz2
$ tar -xf gcc-arm-none-eabi-10.3-2021.10-x86_64-linux.tar.bz2
```

  Create a new system environment variable and name it `ARMGCC_DIR`. The value of this variable should point to the Arm GCC embedded toolchain installation path. For this example, the path is `~/gcc_compiler/gcc-arm-none-eabi-10.3-2021.10`. Add the below line to `~/.bashrc` file.

```
export ARMGCC_DIR=~/gcc_compiler/gcc-arm-none-eabi-10.3-2021.10
```

- Download MCUXpresso SDK
  On the Linux host machine, download the MCUXpresso SDK - a package designed to simplify and accelerate application development with Arm Cortex-M-based devices.
  Note: *Both Git and West must be installed to download the MCUXpresso SDK.*
  After the installation of Git and West, execute the following commands to achieve the whole SDK delivery at revision `MCUX_2.12.0` and place it in the `mcuxsdk-2.12.0` folder.

```
$ west init -m https://github.com/NXPmicro/mcux-sdk --mr MCUX_2.12.0
 mcuxsdk-2.12.0
$ cd mcuxsdk-2.12.0
$ west update
```

3 Disable Ethernet driver from U-Boot and Linux Kernel

To use the Ethernet on Cortex-M core, the Cortex-M core must have exclusive access to the peripheral. The Ethernet access should be disabled from U-Boot and Linux kernel.

3.1 Disable Ethernet driver from U-Boot

To disable the Ethernet driver from U-Boot, follow the steps below:

1. Add the following lines at the end of the U-Boot device tree file:
   Location: `<yocto_build_directory>tmp/work/imx8mm_lpddr4_evk-poky-linux/u-boot-imx/<specified_git_folder>/git/arch/arm/dts/imx8mm-evk.dts`

```
&fec1
{
  status = “disabled”;
};
```

2. Recompile the U-Boot.

```
$ bitbake -f -c compile u-boot-imx
$ bitbake u-boot-imx imx-boot
```

3. Copy the new U-Boot image on the SD card.

```
$ dd if=imx-boot-imx8mm-lpddr4-evk-sd.bin-flash_evk of=/dev/sd<x> bs=1k
 seek=33 conv=fsync
```

   Note: *The storage location may vary. Adjust the `sd<x>` parameter to point to the SD card location.*

3.2 Disable Ethernet driver from Linux Kernel

To disable the Ethernet driver from Linux kernel, follow the steps below:

1. Add the following lines at the end of the kernel device tree:
   Location: `<yocto_build_directory>/tmp/work-shared/imx8mm-lpddr4-evk/kernel-source/arch/arm64/boot/dts/freescale/imx8mm-evk-rpmsg.dts`

```
&fec1
{
  status = “disabled”;
};
```

2. Recompile the kernel:

```
$ bitbake -f -c compile virtual/kernel
$ bitbake virtual/kernel
```

3. Copy the new device tree and the kernel image to the SD boot partition (first (FAT) partition):
   Note: *The built image is located in the* `<yocto_build_directory>/tmp/deploy/images/imx8mm-lpddr4-evk` *folder.*

   ```
   $ sudo mount /dev/sd<x>1 /mnt
   $ cp imx8mm-evk-rpmsg.dtb /mnt
   $ cp Image /mnt
   $ umount /mnt
   ```

   Note: *The storage location may vary. Adjust the mounted partition accordingly.*
4. Now, you can again check the new image by booting the board. The Ethernet interface should not be available in Linux.

   ```
   $ ip addr
   ```

4 lwIP integration and usage

For convenience, the patches are prepared. The patches are located here.

- Download the lwIP stack and place it into the `~/mcuxsdk-2.12.0/middleware` folder:

```
$ cd ~/mcuxsdk-2.12.0/middleware
$ git clone https://github.com/lwip-tcpip/lwip.git
$ git checkout 239918ccc173cb2c2a62f41a40fd893f57faf1d6
```

Note: *The checkout is optional. It brings the exact version on which the patch was developed, but it should work on the latest master.*

- Download the `imx8m_lwip_port.patch` patch and apply it to the lwip directory. This fetches the port support for i.MX 8M (bare-metal lwIP and with FreeRTOS):

```
$ cd lwip
$ wget https://raw.githubusercontent.com/nxp-imx-support/lwip_demo/master/
imx8m_lwip_port.patch
$ git apply --whitespace=nowarn imx8m_lwip_port.patch
```

- Download the `imx8mm_lwip_examples.patch` patch and apply it to the example folder. This fetches the usage examples for i.MX 8MM:

```
$ cd ~/mcuxsdk-2.12.0/examples
$ wget https://raw.githubusercontent.com/nxp-imx-support/lwip_demo/master/
imx8mm_lwip_examples.patch
$ git apply --whitespace=nowarn imx8mm_lwip_examples.patch
```

- The four examples are now in the `~/mcuxsdk-2.12.0/examples/evkmimx8mm/lwip_examples` folder.

4.1 Application structure

The following four examples of lwIP usage are in the `lwip_examples` folder:

- The `lwip_ping` is an example of a ping sender that can be used as a start point to maintain an opened network connection.
- The `lwip_tcpecho` example is a TCP echo server.
- The `lwip_udpecho` example is a UDP echo server.
- The `lwip_mqtt` example is an MQTT client subscriber.

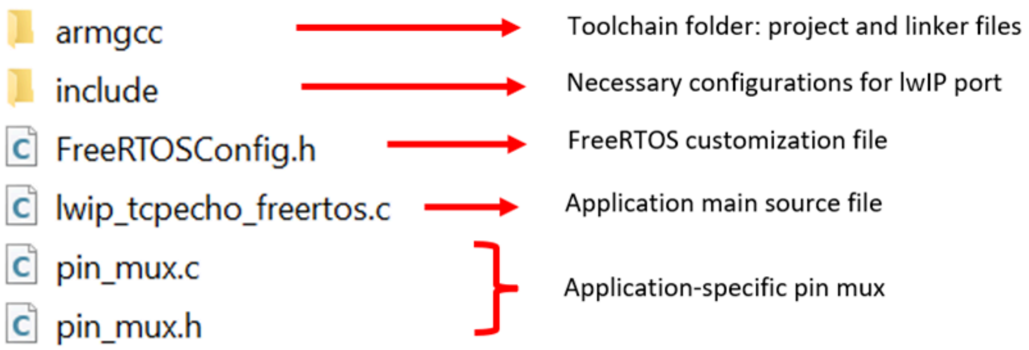

Figure 1. Application folder structure

The generic structure of a FreeRTOS application is in Figure 1. For bare-metal, the `FreeRTOSConfig.h` file is not included.

- The `include` folder contains the `lwipopts.h` file. It is used to overwrite the default configuration of the lwIP, located in the `lwip/src/include/lwip/opt.h` folder. The application-specific options are defined here.
- The `lwip_tcpecho_freertos.c` file contains the main application. The TCP/IP stack uses the Ethernet driver implementation in `~/mcuxsdk-2.12.0/core/drivers/enet`. The implementation of the PHY driver is in `~/mcuxsdk-2.12.0/core/components/phy`.
- The `armgcc` folder is the build directory that contains the project and the linker file (`MIMX8MM6xxxxx_cm4_ram.ld` and `MIMX8MM6xxxxx_cm4_ddr_ram.ld`):
  – The `CMakeLists.txt` file is used by `cmake` to automatically generate the `Makefile`.

The files used for lwIP porting are in `middleware/lwip/port`.

4.2 Building the examples

The four examples described above are built in the same way. The examples are built in order to run either in TCM (Tightly Coupled Memory), or in DDR. For building the bare-metal application, go to `bm/armgcc` directory. For building the FreeRTOS applications, go to `freertos/armgcc` directory. In the `armgcc` directory, four building scripts (two of them for TCM, and two of them for DDR) can be found.

1. Change the folder to the example application project folder, which has a path similar to the following:

```
<install_dir>/examples/evkmimx8mm/lwip_examples/<application_name>/<op_sys>/
armgcc
```

2. To perform the build, run the build script in the command line. The output is as follows:

```
$ ./build_release.sh
-- TOOLCHAIN_DIR: /home/user/gcc_compiler/gcc-arm-none-eabi-10.3-2021.10
-- BUILD_TYPE: release
-- TOOLCHAIN_DIR: /home/user/gcc_compiler/gcc-arm-none-eabi-10.3-2021.10
-- BUILD_TYPE: release
-- The ASM compiler identification is GNU
-- Found assembler: /home/user/gcc_compiler/gcc-arm-none-eabi-10.3-2021.10/
bin/arm-none-eabi-gcc
-- Configuring done
-- Generating done
-- Build files have been written to: /home/user/mcuxsdk-2.12.0/examples/
evkmimx8mm/lwip_examples/lwip_tcp_udp_responder/bm/armgcc
Scanning dependencies of target lwipcore
```

```
[  1%] Building C object CmakeFiles/lwipcore.dir/home/user/mcuxsdk-2.12.0/
middleware/lwip/src/core/init.c.obj
[  1%] Building C object CmakeFiles/lwipcore.dir/home/user/mcuxsdk-2.12.0/
middleware/lwip/src/core/inet_chksum.c.obj
[  2%] Building C object CmakeFiles/lwipcore.dir/home/user/mcuxsdk-2.12.0/
middleware/lwip/src/core/dns.c.obj
< -- skipping lines -- >
[100%] Building C object CmakeFiles/lwip_ping_bm.elf.dir/home/user/
mcuxsdk-2.12.0/core/drivers/gpt/fsl_gpt.c.obj
[100%] Linking C executable release/lwip_ping_bm.elf
Memory region         Used Size  Region Size  %age Used
    m_interrupts:          576 B        576 B    100.00%
          m_text:        78040 B     130496 B     59.80%
          m_data:        10096 B       128 KB      7.70%
         m_data2:        12416 B        16 MB      0.07%
[100%] Built target lwip_ping_bm.elf
```

This script compiles the project and creates the `release` folder, which contains the `*.bin` and `*.elf` files.

Note: *To print the additional debug messages, use the `build_debug.sh` script. This script creates the `debug` folder, which contains the resulting binary file.*

3. Copy the binary file to the first (FAT) partition of the SD card (similar to the image copying in Section 3.2).

4.3 Run the applications using U-Boot

Connect the i.MX 8MM platform to the host Ubuntu PC via USB cable between the DEBUG USB-UART connector and the PC USB connector. The Ubuntu OS finds the serial devices automatically.

To determine your debug port, find the TTY device with name /dev/ttyUSB*. One port is for the debug messages from the Cortex-A53, and the other is for the Cortex-M4. The port number is allocated randomly, so opening both is beneficial for development.

Open the serial device in your preferred serial terminal emulator (ex. PuTTY). Set the speed to 115200 bps, 8 data bits, 1 stop bit (115200, 8N1), and no parity.

- Before starting the Cortex-M core, connect the board to a PC via an Ethernet cable. Set the static IP address of the PC:

```
$ ip addr add 192.168.11.2/24 dev eno1
```

Note: *Replace the Ethernet device name according to your case.*

4.3.1 Run the ping server application

- Boot the board and stop the execution in U-Boot: You can then write the image and run it from TCM or DRAM with the following commands:
  1. If the `lwip_ping_bm.bin` file is made from `release` target, which means the binary runs from TCM, use the following commands to boot:

     ```
     u-boot=> fatload mmc 1:1 0x48000000 lwip_ping_bm.bin
     u-boot=> cp.b 0x48000000 0x7e0000 0x20000
     u-boot=> bootaux 0x7e0000
     ```

  2. If the `lwip_ping_bm.bin` file is made from `ddr_release` target, which means the binary file runs from DRAM, use the following commands:

     ```
     u-boot=> fatload mmc 1:1 0x80000000 lwip_ping_bm.bin
     u-boot=> dcache flush
     u-boot=> bootaux 0x80000000
     ```

For convenience, a U-Boot variable can be created, that stores the above commands for the subsequent boot ups (example below for bare-metal):

```
u-boot=> setenv lwip "fatload mmc 1:1 0x48000000
lwip_ping_bm.bin; cp.b 0x48000000 0x7e0000 0x20000; bootaux 0x7e0000"
```

The binary can be loaded into the TCM/DDR and the Cortex-M core can be started using the following command:

```
u-boot=> run lwip
```

Note: *If the Linux OS kernel runs together with the M4 core, make sure that the correct `dtb` file is used. This `dtb` file reserves the resources used by the M4 core and avoids the Linux kernel from configuring them. Use the following command before running the kernel:*

```
u-boot=> setenv fdtfile imx8mp-evk-rpmsg.dtb
```

- Test the application.
  The `ping` application starts to send Ethernet packets immediately after the ENET initialization. To view these packets on the connected PC, type the following command:

  ```
  $ sudo tcpdump -v -i eno1
  ```

  Note: *Change the Ethernet interface name according to your case.*

  Figure 2 shows the output on the Cortex-M core, while running the application.

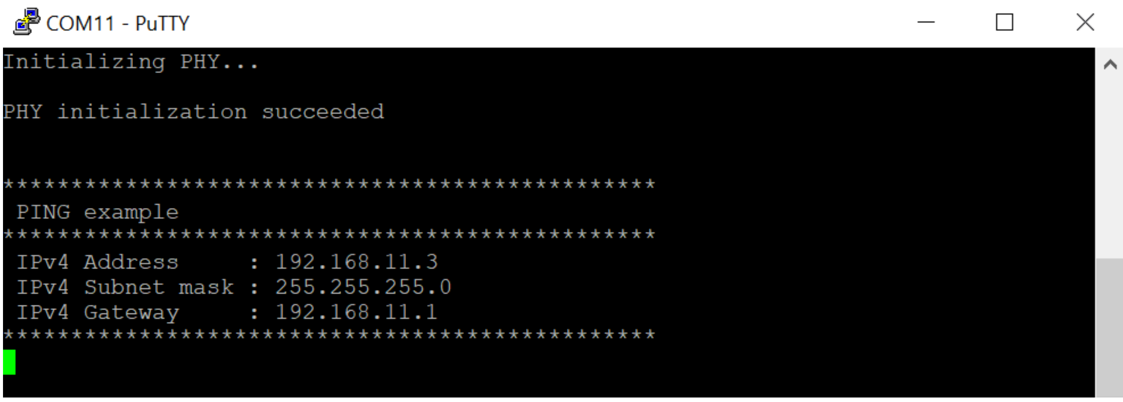

Figure 2. Text display of the ping_server demo

4.3.2 Run the TCP responder application

This application implements a TCP client that replies at each request from the server.

- Boot the board and stop the execution in U-Boot.
  You can then write the image and run it from TCM or DRAM with the following commands:
  1. If the `lwip_tcpecho_bm.bin` file is made from `release` target, which means the binary runs from TCM, use the following commands to boot:

     ```
     u-boot=> fatload mmc 1:1 0x48000000 lwip_tcpecho_bm.bin
     u-boot=> cp.b 0x48000000 0x7e0000 0x20000
     u-boot=> bootaux 0x7e0000
     ```

  2. If the `lwip_tcpecho_bm.bin` file is made from `ddr_release` target, which means the binary file runs from DRAM, use the following commands:

     ```
     u-boot=> fatload mmc 1:1 0x80000000 lwip_tcpecho_bm.bin
     u-boot=> dcache flush
     u-boot=> bootaux 0x80000000
     ```

You can create a U-Boot variable that stores the above commands for the subsequent bootups:

```
u-boot=> setenv lwip "fatload mmc 1:1 0x48000000 lwip_tcpecho_bm.bin; cp.b
 0x48000000 0x7e0000 0x20000; bootaux 0x7e0000"
u-boot=> saveenv
```

The binary can be loaded into the TCM/DDR and the Cortex-M core can be started using the following command:

```
u-boot=> run lwip
```

Note: *If the Linux OS kernel runs together with the M4 core, make sure that the correct* `dtb` *file is used. This* `dtb` *file reserves the resources used by the M4 core and avoids the Linux kernel from configuring them. Use the following command before running the kernel:*

```
u-boot=> setenv fdtfile imx8mp-evk-rpmsg.dtb
```

- Test the application:
  Type the following commands on the PC:
  - `ping`

    ```
    $ ping 192.168.11.3
    ```

  - `nc` (`netcat`)
    Send TCP packets:

    ```
    $ nc 192.168.11.3 7
    ```

    Anything you type is echoed by the board.
    Note: *Do not change the number of the port. The TCP application is configured to listen and send on port 7.*
    Figure 3 shows the output on Cortex-M.

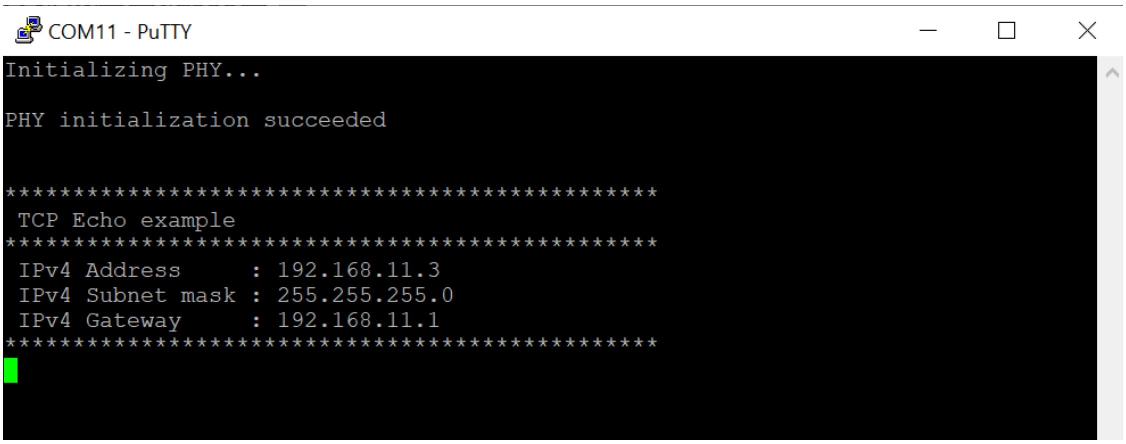

Figure 3. Text display of the tcp echo demo

4.3.3 Run the UDP responder application

This application implements a UDP client that replies at each request from the server.

- Boot the board and stop the execution in U-Boot.
  You can then write the image and run it from TCM or DRAM with the following commands:
  1. If the `lwip_udpecho_bm.bin` file is made from `release` target, which means the binary runs from TCM, use the following commands to boot:

     ```
     u-boot=> fatload mmc 1:1 0x48000000 lwip_udpecho_bm.bin
     u-boot=> cp.b 0x48000000 0x7e0000 0x20000
     ```

```
u-boot=> bootaux 0x7e000
```

2. If the `lwip_udpecho_bm.bin` file is made from `ddr_release` target, which means the binary file runs from DRAM, use the following commands:

```
u-boot=> fatload mmc 1:1 0x80000000 lwip_udpecho_bm.bin
u-boot=> dcache flush
u-boot=> bootaux 0x80000000
```

You can create a U-Boot variable that stores the above commands for the subsequent bootups:

```
u-boot=> setenv lwip "fatload mmc 1:1 0x48000000 lwip_udpecho_bm.bin; cp.b
 0x48000000 0x7e0000 0x20000; bootaux 0x7e0000"
u-boot=> saveenv
```

The binary can be loaded into the TCM/DDR and the Cortex-M core can be started using the following command:

```
u-boot=> run lwip
```

Note: *If the Linux OS kernel runs together with the M4 core, make sure that the correct `dtb` file is used. This `dtb` file reserves the resources used by the M4 core and avoids the Linux kernel from configuring them. Use the following command before running the kernel:*

```
u-boot=> setenv fdtfile imx8mp-evk-rpmsg.dtb
```

- Test the application:
  Type the following commands on the PC:
  - `ping`

    ```
    $ ping 192.168.11.3
    ```

  - `nc` (`netcat`)
    Send UDP packets:

    ```
    $ nc -u 192.168.11.3 7
    ```

    Anything you type is echoed by the board.
    Note: *Do not change the number of the port. The UDP application is configured to listen and send on port 7. Figure 4 shows the output on Cortex-M.*

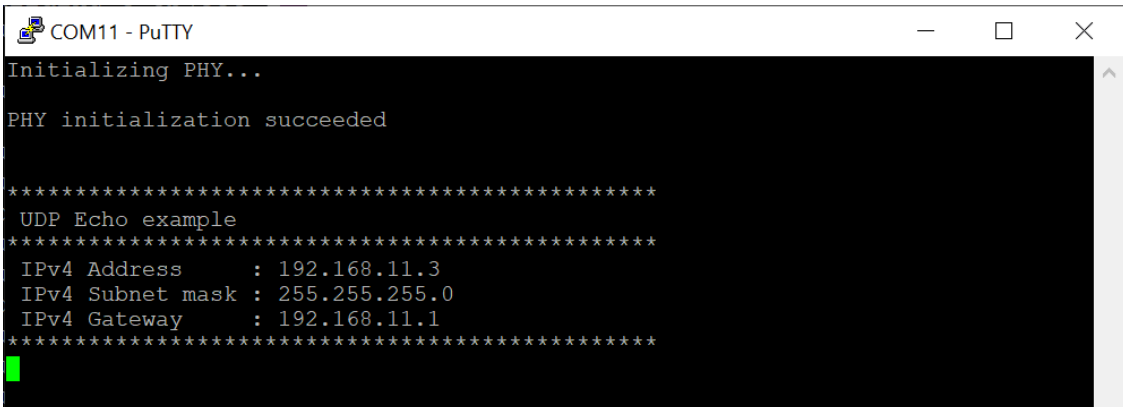

Figure 4. Text display of the udpecho demo

4.3.4 Run the MQTT client application

The MQTT (Message Queueing Telemetry Transport) is a protocol used for communication between IoT devices. MQTT communication works as a publish and subscribe system. Some devices publish messages on a specific topic and all devices that are subscribed to that topic receive the message. The MQTT broker is an intermediary entity that receives messages published by clients, filters the messages by topic, and distributes them to subscribers.

In this example, i.MX 8MM runs an MQTT client that is subscribed to two topics. On another machine, a second client publishes the messages on these two topics.

4.3.4.1 MQTT broker

To get and start the MQTT broker on your PC, follow the steps below:

1. Before starting the MQTT example, install an MQTT broker on the connected PC. For this example, the Mosquitto broker is used:

```
$ sudo apt-get update
$ sudo apt-get install mosquitto
$ sudo apt-get install mosquitto-clients
```

2. Create the `mosquitto.config` configuration file and include the following lines:

```
listener 1883
allow_anonymous true
```

3. Start the MQTT broker:

```
$ mosquitto -c mosquitto.config
```

4.3.4.2 MQTT client

- Boot the board and stop the execution in U-Boot:
  You can then write the image and run it from TCM or DRAM with the following commands:
  1. If the `lwip_mqtt_bm.bin` file is made from `release` target, which means the binary runs from TCM, use the following commands to boot:

```
u-boot=> fatload mmc 1:1 0x48000000 lwip_mqtt_bm.bin
u-boot=> cp.b 0x48000000 0x7e0000 0x20000
u-boot=> bootaux 0x7e0000
```

  2. If the `lwip_mqtt_bm.bin` file is made from `ddr_release` target, which means the binary file runs from DRAM, use the following commands:

```
u-boot=> fatload mmc 1:1 0x80000000 lwip_mqtt_bm.bin
u-boot=> dcache flush
u-boot=> bootaux 0x80000000
```

You can create a U-Boot variable that stores the above commands for the subsequent bootups:

```
u-boot=> setenv lwip "fatload mmc 1:1 0x48000000 lwip_mqtt_bm.bin; cp.b
 0x48000000 0x7e0000 0x20000; bootaux 0x7e0000"
u-boot=> saveenv
```

The binary can be loaded into the TCM/DDR and the Cortex-M core can be started using the following command:

```
u-boot=> run lwip
```

Note: *If the Linux OS kernel runs together with the M4 core, make sure that the correct `dtb` file is used. This `dtb` file reserves the resources used by the M4 core and avoids the Linux kernel from configuring them. Use the following command before running the kernel:*

```
u-boot=> setenv fdtfile imx8mp-evk-rpmsg.dtb
```

- Test the application:
  In this example, the client is subscribed to the `topic_qos1` and `topic_qos0` topics. The client receives all messages published with one of these two topics. To publish messages, use the following command on a PC, but in a new terminal:

```
$ mosquitto_pub -t 'topic_qos0' -m 'Test topic_qos0'
$ mosquitto_pub -t 'topic_qos1' -m 'Test topic_qos1'
```

  Figure 5 shows the output on the Cortex-M core:

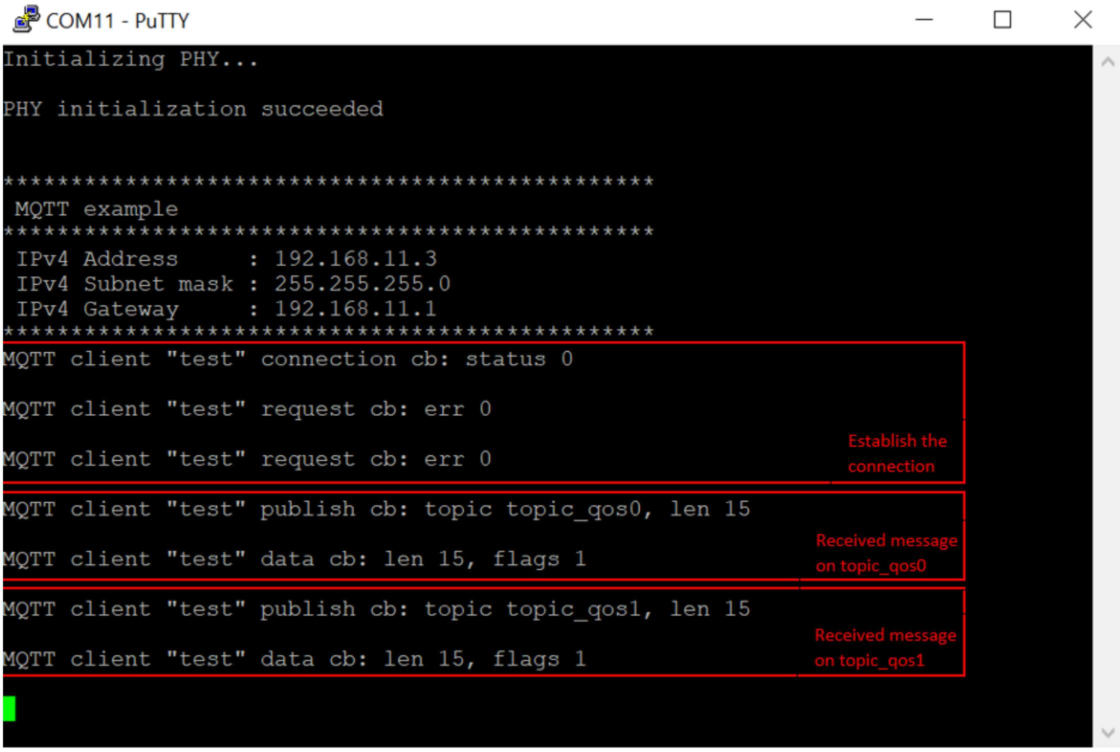

Figure 5. Text display of the mqtt_client demo

5 Revision history

Table 1 summarizes the changes done to this document since the initial release.

Table 1. Revision history

Revision number	Date	Substantive changes
1	25 April 2022	• Updated Section 1.1, Section 2, Section 3.2, Section 4, Section 4.1, Section 4.2, Section 4.3.1, Section 4.3.2, and Section 4.3.4.2 • Added Section 4.3.3 • Made few editorial changes
0	06 December 2022	Initial release

6 Legal information

6.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

6.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. - NXP B.V. is not an operating company and it does not distribute or sell products.

6.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

i.MX — is a trademark of NXP B.V.

Contents

1	**Introduction**	**2**
1.1	Software environment	2
1.2	Hardware setup and equipment	2
2	**Prerequisites**	**2**
3	**Disable Ethernet driver from U-Boot and Linux Kernel**	**3**
3.1	Disable Ethernet driver from U-Boot	3
3.2	Disable Ethernet driver from Linux Kernel	3
4	**lwIP integration and usage**	**4**
4.1	Application structure	4
4.2	Building the examples	5
4.3	Run the applications using U-Boot	6
4.3.1	Run the ping server application	6
4.3.2	Run the TCP responder application	7
4.3.3	Run the UDP responder application	8
4.3.4	Run the MQTT client application	9
4.3.4.1	MQTT broker	10
4.3.4.2	MQTT client	10
5	**Revision history**	**11**
6	**Legal information**	**12**

Date of release: 25 April 2023
Document identifier: AN13799