Automotive USB Stack User Manual

Document Number: UMAUSBSTACK
Rev. 1.3

User Manual

TABLE OF CONTENTS

1. Revision History

Table 1-1. Revision History

Revision	Date	Author	Description
1.0	29.11.2017	Mihail Ocneanu	First version (BETA milestone)
1.1	08.02.2018	Mihail Ocneanu	Second version (BETA milestone)
1.2	01.08.2018	Mihail Ocneanu	RTM revision
1.3	14.09.2018	Mihail Ocneanu	BETA 1.9.0 revision

2. Names and definitions

Name	Definition	References
USB stack	USB – Universal Serial Bus is a standard that defines communication between a host and a device	http://www.usb.org/developers/docs/devclass_docs/
S32 SDK	A software development kit product targeted and including all the software layers [drivers, stacks, OS, applications] for 32-bit automotive device	It is distributed inside the S32 DS for PA and ARM products.
FATFS	FAT file system – open source FATFS targeted for embedded	https://en.wikipedia.org/wiki/File_Allocation_Table
ECM	Ethernet Control Model	https://en.wikipedia.org/wiki/Ethernet_over_USB
USB - CDC	USB communications device class	https://en.wikipedia.org/wiki/USB_communications_device_class

3. Introduction

This User Manual describes NXP Semiconductors USB stack for automotive products.

3.1 Overview

The USB stack is an implementation of the USB standard. It is a port of the Kinetis USB stack on NXP's automotive processors.

The current implementation supports:

- mass storage devices on both the host controller and the OTG interface. The file system is an open source project, supported by Elm Chan and for the latest news and releases please check: http://elm-chan.org/fsw/ff/00index_e.html.
- USB CDC-ECM communication model running on both the host controller and OTG interface

3.2 Features

The stack includes the following features:

- Host controller and OTG support
- Support for mass storage devices
- Support for UART messages or console messages using semihosting (please check S32 SDK settings)
- Support for CDC-ECM communication model

The FAT file system includes the following features:

- DOS/Windows compatible FAT filesystem (FAT12, FAT16, FAT32).
- exFAT filesystem
- Very small footprint for program code and work area.
- Multiple volumes (physical drives and partitions).
- Multiple code pages including DBCS.
- Long file name in OEM code or Unicode.
- Supports multiple sector size.
- Supports file size of up to 4 GB (exFAT variant is virtually unlimited).
- Supports volume size of up to 2 TB (exFAT variant is virtually unlimited).
- Supports cluster size of up to 64 KB (exFAT variant is virtually unlimited).
- Supports sector size of up to 4 KB (exFAT variant is virtually unlimited).

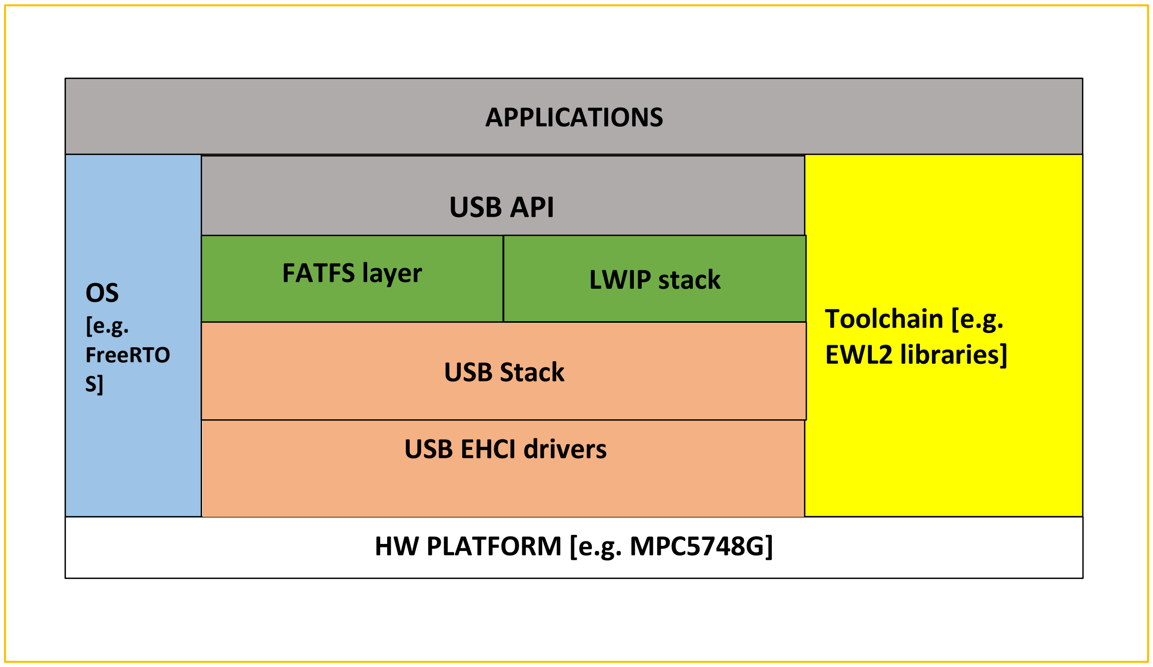
- exFAT: Improved performance of cluster allocation by use of allocation bitmap.
- exFAT: Improved performance of data transfer by increasing cluster size (up to 32 MB).
- Supports UTC timestamp.
- Thread safe.

The TCP/IP stack includes the following features:

- IP (Internet Protocol)
- ICMP (Internet Control Message Protocol) for network maintenance and debugging
- UDP (User Datagram Protocol)
- TCP (Transmission Control Protocol) with congestion control, RTT estimation and fast recovery/fast retransmit
- ARP (Address Resolution Protocol) for Ethernet

The USB stack API consists of three parts:

- USB API: implementing USB specific instructions
- FATFS API: implementing file system functionality. This layer is independent of the USB stack and it uses disk I/O operations to communicate with the mass storage device.
- LWIP API: implementing ENET read/write functionality. This layer is dependent on the USB stack, as it reads and writes packets on the USB port.



4. USB stack mass storage demo application

The USB stack features a set of applications for each device family, which demonstrates key functionalities of this software product and help users to quickly ramp-up into the software product.

4.1 Features

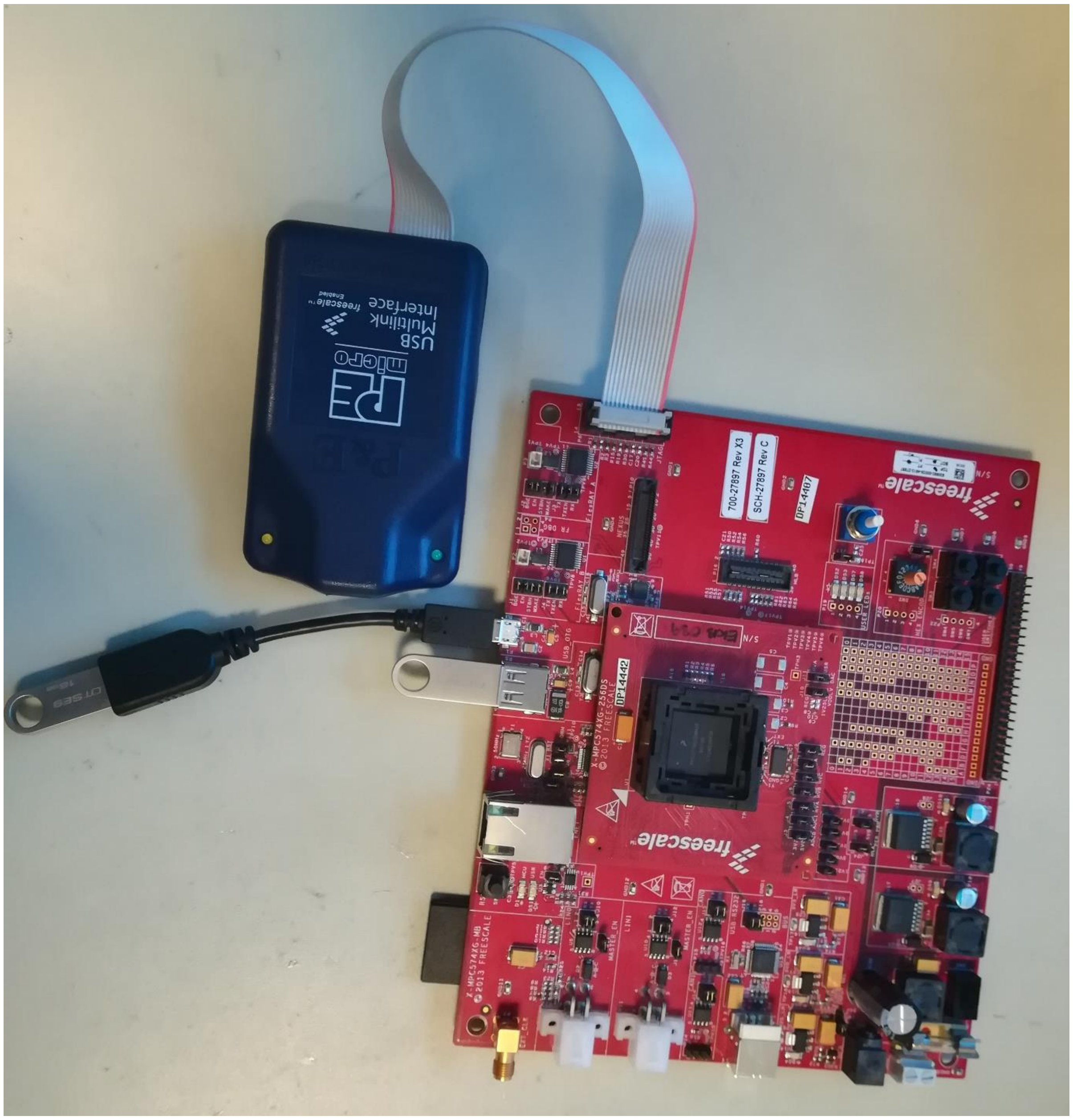
The current release of the stack contains a demo application which includes the following functionalities (applications):

1. **Mass storage device over Host controller interface**. Simple application, using the host controller interface, that enumerates an USB flash drive and performs basic FATFS operations.

2. **Mass storage device over OTG interface.** Simple application, using the on-the-go controller interface, that enumerates an USB flash drive and performs basic FATFS operations.

4.2 Hardware prerequisites

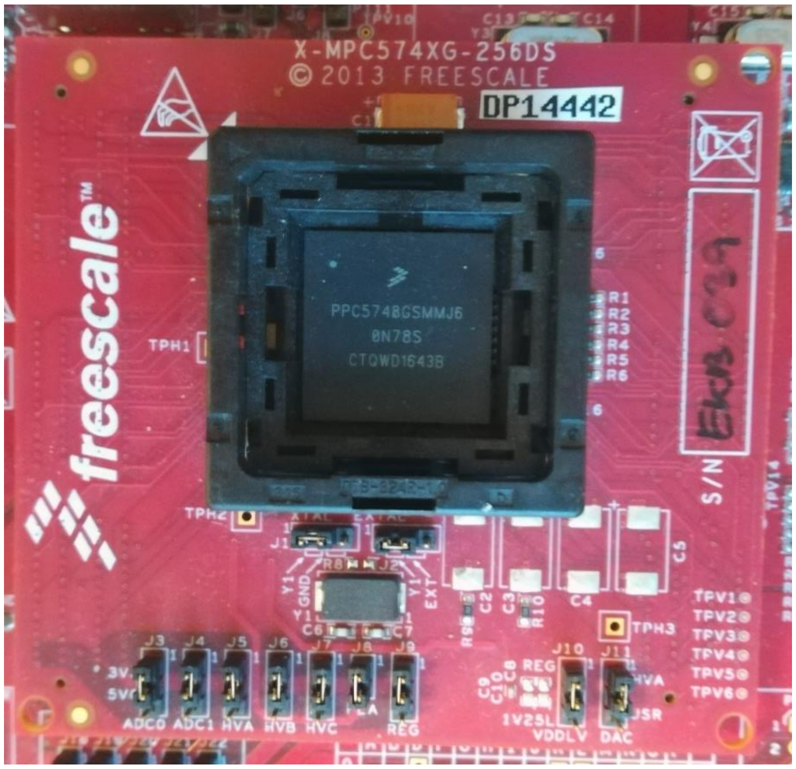
The application is designed to run on **MPC5748G main board***. The board features two USB connectors: an OTG and a Host Controller. In order to use the OTG, one would need an OTG cable.



To load the software on the mainboard, one would need an USB Multilink Interface or a Lauterbach debugger.

***Additional mentions:**

- **The processor's revision must to be 0N78S or higher. (See picture below)**
- **The application should run on the MPC5748G devkit revision C.**



4.3 Software prerequisites

- **S32 Design Studio for Power Architecture** v2017.R1
  Includes toolchain GCC E200 VLE GNU compiler 4.9.4- 20160726 and EWL2 libraries
- Drivers for the debugging interface. They should be delivered with the debug interface.

4.4 Obtaining the components needed for the stack

The USB stack relies on the following software components:

- SDK drivers (e.g. USB stack and FATFS implementation)
- (optional) FreeRTOS (SW574XG-FREERTOS-EAR b9.0.0 version)

All those software components are being delivered by the SDK for Power Architecture (with support for MPC574XG devices).

SDK for Power Architecture is delivered via S32 Design Studio IDE for Power Architecture based MCUs located at:

https://www.nxp.com/support/developer-resources/run-time-software/s32-design-studio-ide/s32-design-studio-ide-for-power-architecture-based-mcus:S32DS-PA

4.5 Setting the USB interface and the print output:

- By setting the ***CONTROLLER_ID*** one can choose between the USB Host controller or USB OTG interface.
- By setting the ***SDK_DEBUGCONSOLE*** one can choose the UART or the IDE's console to print the messages. In order to use the semihosting option, additional settings need to be made:
    - Right click on the project -> Properties -> C/C++ Build -> Settings -> Target processor -> Libraries Support -> choose an option with ***-specs=semihost.specs*** (i.e. newLib Debugger Console).
    - Debug Configurations-> choose the debug instance -> Startup -> Enable Semihosting needs to be checked.
    - Clean build and rebuild

4.6 Building the demo application for MPC5748G device from SDK

1. Run SDK EAR installer. It will install on known path.
2. Start S32 DS
3. File -> New -> S32DS Project from Example -> usb_msd_fatfs
4. Build.

4.7 Expected results:

The sample app is doing the following:

- It mounts the device
- Reads the size of the device
- Reads the sectors size of the device
- Writes a single aligned/unaligned sector, multiple sectors and erases a block.
- Tests the device on the 4GB barrier problem.
- Formats the device, creating the FATFS
- Creates text file "*newFile0.bin*"
- *Writes to "newFile0.bin", syncs it and closes it.*
- Creates binary file "*newFile1.bin*"
- *Writes to "newFile1.bin", syncs it, closes it.*
- Creates text file "*newFile2.bin*"
- *Writes to "newFile2.bin", syncs it, closes it.*

- Copies file “newFile0.bin to “newFile0copy.bin”
- Copies file “newFile1.bin” to “newFile1copy.bin”
- Reads data from “newFile0.bin”, “newFile1.bin” and “newFile2.bin”
- Lists files

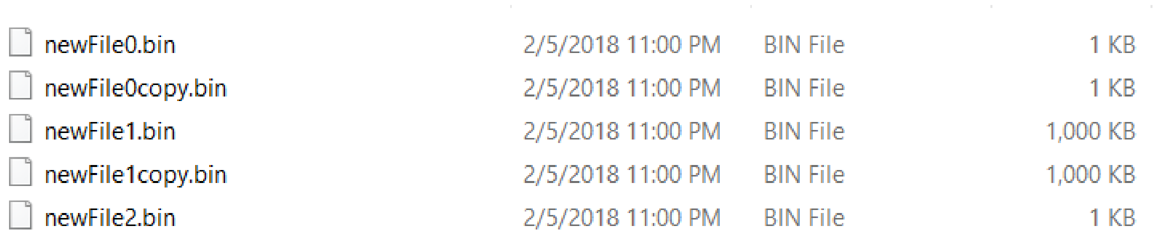
These messages are printed in the console when testing the USB stack with a mass storage device:

```
host init done, the host stack version is 1.0.0.
mass storage device attached:pid=0x6544
                             vid=0x930
                             address=1
FATFS diskio tests
 task_diskio_all(0, 5, 0x40012814, 0x0000C800)
 disk_initialize(0) - ok.
 disk_ioctl(0, GET_SECTOR_COUNT, 0x40012740) - ok.
 Number of sectors on the drive 0 is 30274559.
 disk_ioctl(0, GET_SECTOR_SIZE, 0x0) - ok.
 Size of sector is 512 bytes.
 disk_ioctl(0, GET_BLOCK_SIZE, 0x101A27C) - ok.
 Size of the erase block is 0 sectors.
Single sector write
 disk_write(0, 0x40012814, 0, 1) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012814, 0, 1) - ok.
 Test data matched.
Multiple sector write
 disk_write(0, 0x40012814, 1, 4) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012814, 1, 4) - ok.
 Test data matched.
Single sector write unaligned
disk_write(0, 0x40012817, 5, 1) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012819, 5, 1) - ok.
 Test data matched.
4GB barrier
disk_write(0, 0x40012814, 6, 1) - ok.
 disk_write(0, 0x40012A14, 8388614, 1) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012814, 6, 1) - ok.
 disk_read(0, 0x40012A14, 8388614, 1) - ok.
Test data matched.
Test cycle 1 of 5 completed
 disk_initialize(0) - ok.
 disk_ioctl(0, GET_SECTOR_COUNT, 0x01CDF3FF) - ok.
 Number of sectors on the drive 0 is 30274559.
 disk_ioctl(0, GET_SECTOR_SIZE, 0x200) - ok.
 Size of sector is 512 bytes.
 disk_ioctl(0, GET_BLOCK_SIZE, 0x0) - ok.
 Size of the erase block is 0 sectors.
Single sector write
 disk_write(0, 0x40012814, 0, 1) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012814, 0, 1) - ok.
 Test data matched.
Multiple sector write
 disk_write(0, 0x40012814, 1, 4) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012814, 1, 4) - ok.
 Test data matched.
Single sector write unaligned
disk_write(0, 0x40012817, 5, 1) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012819, 5, 1) - ok.
 Test data matched.
4GB barrier
disk_write(0, 0x40012814, 6, 1) - ok.
 disk_write(0, 0x40012A14, 8388614, 1) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012814, 6, 1) - ok.
 disk_read(0, 0x40012A14, 8388614, 1) - ok.
Test data matched.
Test cycle 2 of 5 completed
 disk_initialize(0) - ok.
 disk_ioctl(0, GET_SECTOR_COUNT, 0x01CDF3FF) - ok.
 Number of sectors on the drive 0 is 30274559.
 disk_ioctl(0, GET_SECTOR_SIZE, 0x200) - ok.
 Size of sector is 512 bytes.
 disk_ioctl(0, GET_BLOCK_SIZE, 0x0) - ok.
 Size of the erase block is 0 sectors.
Single sector write
 disk_write(0, 0x40012814, 0, 1) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012814, 0, 1) - ok.
 Test data matched.
Multiple sector write
 disk_write(0, 0x40012814, 1, 4) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012814, 1, 4) - ok.
 Test data matched.
Single sector write unaligned
disk_write(0, 0x40012817, 5, 1) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012819, 5, 1) - ok.
 Test data matched.
4GB barrier
disk_write(0, 0x40012814, 6, 1) - ok.
 disk_write(0, 0x40012A14, 8388614, 1) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012814, 6, 1) - ok.
 disk_read(0, 0x40012A14, 8388614, 1) - ok.
Test data matched.
Test cycle 3 of 5 completed
 disk_initialize(0) - ok.
 disk_ioctl(0, GET_SECTOR_COUNT, 0x01CDF3FF) - ok.
 Number of sectors on the drive 0 is 30274559.
 disk_ioctl(0, GET_SECTOR_SIZE, 0x200) - ok.
 Size of sector is 512 bytes.
 disk_ioctl(0, GET_BLOCK_SIZE, 0x0) - ok.
 Size of the erase block is 0 sectors.
Single sector write
 disk_write(0, 0x40012814, 0, 1) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012814, 0, 1) - ok.
 Test data matched.
Multiple sector write
 disk_write(0, 0x40012814, 1, 4) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012814, 1, 4) - ok.
 Test data matched.
Single sector write unaligned
disk_write(0, 0x40012817, 5, 1) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012819, 5, 1) - ok.
 Test data matched.
4GB barrier
disk_write(0, 0x40012814, 6, 1) - ok.
 disk_write(0, 0x40012A14, 8388614, 1) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012814, 6, 1) - ok.
 disk_read(0, 0x40012A14, 8388614, 1) - ok.
Test data matched.
Test cycle 4 of 5 completed
 disk_initialize(0) - ok.
 disk_ioctl(0, GET_SECTOR_COUNT, 0x01CDF3FF) - ok.
 Number of sectors on the drive 0 is 30274559.
 disk_ioctl(0, GET_SECTOR_SIZE, 0x200) - ok.
 Size of sector is 512 bytes.
 disk_ioctl(0, GET_BLOCK_SIZE, 0x0) - ok.
 Size of the erase block is 0 sectors.
Single sector write
 disk_write(0, 0x40012814, 0, 1) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012814, 0, 1) - ok.
 Test data matched.
Multiple sector write
 disk_write(0, 0x40012814, 1, 4) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012814, 1, 4) - ok.
 Test data matched.
Single sector write unaligned
```

```
disk_write(0, 0x40012817, 5, 1) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012819, 5, 1) - ok.
 Test data matched.
4GB barrier
disk_write(0, 0x40012814, 6, 1) - ok.
 disk_write(0, 0x40012A14, 8388614, 1) - ok.
 disk_ioctl(0, CTRL_SYNC, NULL) - ok.
 disk_read(0, 0x40012814, 6, 1) - ok.
 disk_read(0, 0x40012A14, 8388614, 1) - ok.
Test data matched.
Test cycle 5 of 5 completed
diskio tests result: 0
PASS
FATFS f_fdisk/f_mkfs test
This test may take a few minutes...
f_fdisk status: 0
f_mkfs status: 0
f_mount status: 0
PASS
FATFS f_open(FA_CREATE_ALWAYS) test
f_mount status: 0
createTextFile
Written byte(s): 21
f rw noh  reg  21 octets  2018-02-06 00:00:00 0:newFile0.bin
createBinaryFile
f rw noh  reg  1024000 octets  2018-02-06 00:00:00 0:newFile1.bin
createTextFile
Written byte(s): 21
f rw noh  reg  21 octets  2018-02-06 00:00:00 0:newFile2.bin
                                                            PASS
FATFS task_fatfs_copy_file test
f_mount status: 0
copyFile
fdate: 19526 ftime: 0
copyFile
fdate: 19526 ftime: 0
PASS
FATFS f_open(FA_READ) test
f_mount status: 0
Read byte(s): 21
Read byte(s): 1024000
Read byte(s): 21
PASS
FATFS f_opendir()/f_readdir() test
f_mount status: 0
f rw noh  reg  21 octets  2018-02-06 00:00:00 /
f rw noh  reg  1024000 octets  2018-02-06 00:00:00 /
f rw noh  reg  21 octets  2018-02-06 00:00:00 /
f rw noh  reg  21 octets  2018-02-06 00:00:00 /
f rw noh  reg  1024000 octets  2018-02-06 00:00:00 /
```

```
     File(s): 6, Space: 0 octets
PASS
Number of logical units supported by the device is 0.
Mass Storage Reset - PASS
```

This is a snapshot of the file structure on the USB mass storage device after running the application:



newFile0.bin	2/5/2018 11:00 PM	BIN File	1 KB
newFile0copy.bin	2/5/2018 11:00 PM	BIN File	1 KB
newFile1.bin	2/5/2018 11:00 PM	BIN File	1,000 KB
newFile1copy.bin	2/5/2018 11:00 PM	BIN File	1,000 KB
newFile2.bin	2/5/2018 11:00 PM	BIN File	1 KB

5. USB CDC-ECM demo application:

5.1 Features

The current release of the stack contains a demo application which includes the following functionalities (applications):

1. **Ethernet communication over Host controller interface**. Simple application, using the host controller interface, that enumerates an USB CDC-ECM compatible device, initializes the TCP/IP stack, responds to ping and receives TCP/UDP packets.
2. **Ethernet communication over OTG interface.** It's the same application, using the USB OTG interface.

5.2 Hardware prerequisites

The application is designed to run on **MPC5748G main board***. The board features two USB connectors: an OTG and a Host Controller. To use the OTG, one would need an OTG cable.

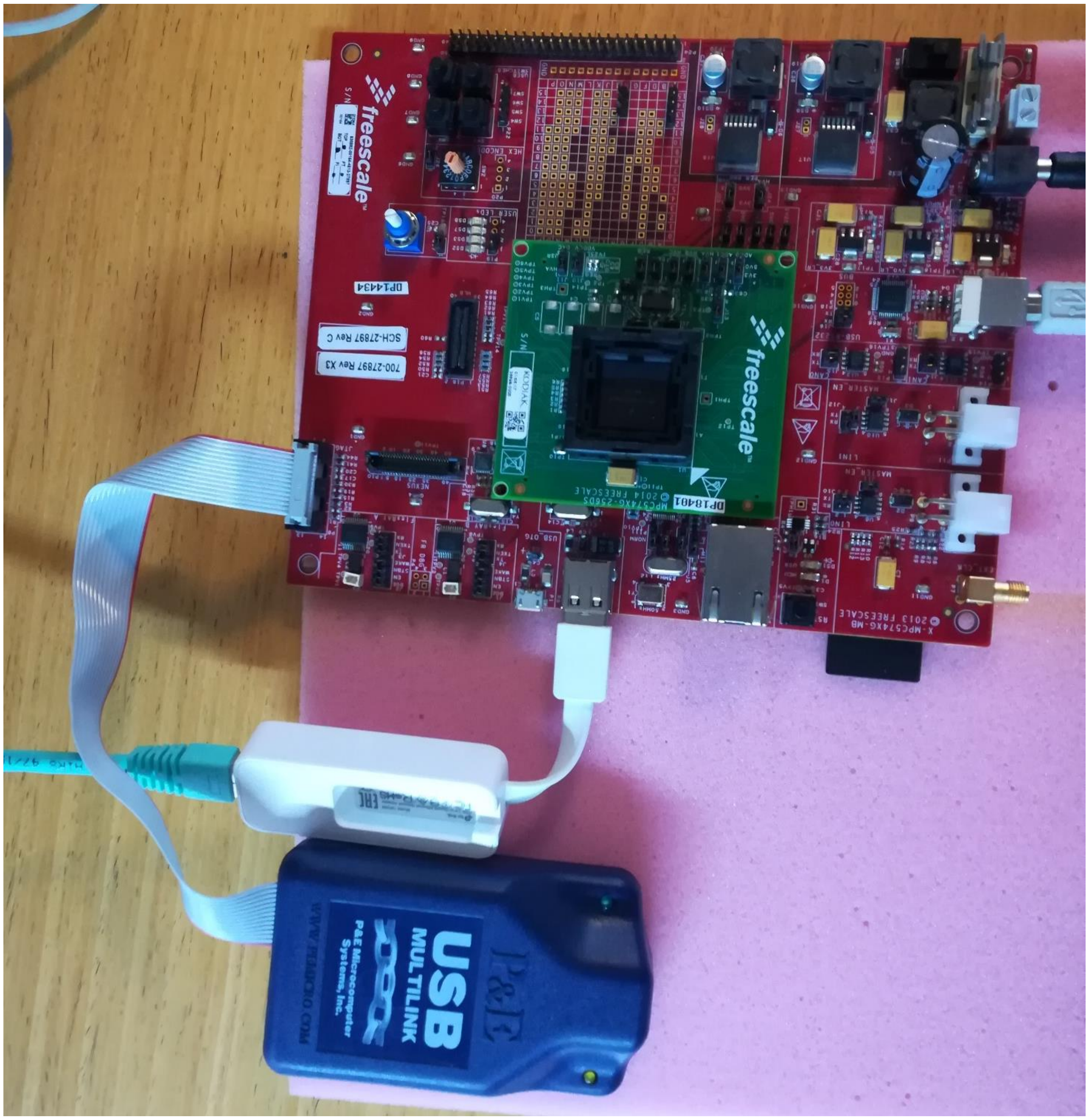
To load the software on the mainboard, one would need an USB Multilink Interface or a Lauterbach debugger.

***Additional mentions:**

- **The processor's revision must to be 0N78S or higher. (See picture below)**
- **The application should run on the MPC5748G devkit revision C.**

To run this application, a **USB CDC-ECM** to **ENET** convertor is required. This convertor should be considered a generic device and its purpose is to receive the USB messages and transmit them over an ethernet cable. **It must be USB CDC-ECM compatible as the stack only supports this communication model**. Below you can find some compatible devices with this example:

- TP-Link UE300
- I.MX6 board with Linux and the USB driver running as USB device.
- Raspberry PI, running the USB driver as USB device.
- A generic Linux distribution running on a PC with USB ports.



To load the software on the mainboard, one would need an USB Multilink Interface or a Lauterbach debugger.

5.3 Software prerequisites

- **S32 Design Studio for Power Architecture v2017.R1**

Includes toolchain GCC E200 VLE GNU compiler 4.9.4-20160726 and EWL2 libraries

- Drivers for the debugging interface. They should be delivered with the debug interface.

5.4 Obtaining the components needed for the stack

The USB stack relies on the following software components:

- SDK drivers (e.g. USB stack and LWIP stack)

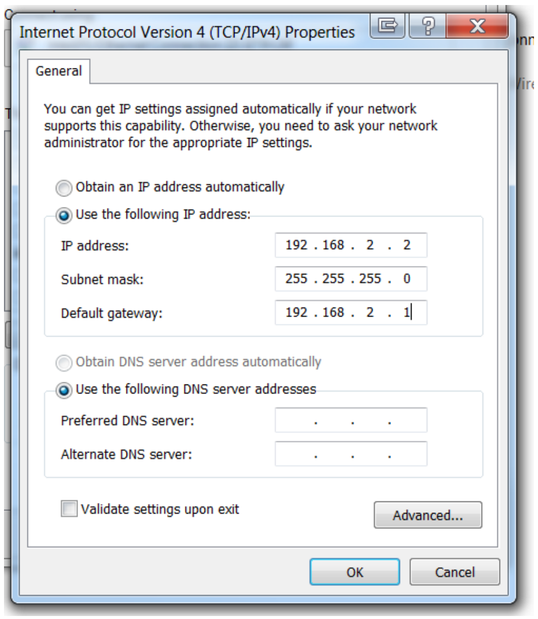
All those software components are being delivered by the SDK for Power Architecture (with support for MPC574XG devices).

SDK for Power Architecture is delivered via S32 Design Studio IDE for Power Architecture based MCUs located at:

https://www.nxp.com/support/developer-resources/run-time-software/s32-design-studio-ide/s32-design-studio-ide-for-power-architecture-based-mcus:S32DS-PA

5.5 Setting the USB, IP and MAC address:

- By setting the ***CONTROLLER_ID*** one can choose between the USB Host controller or USB OTG interface.
- Set the IP address on your device to an IP in the same network. E.g.:



You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

Obtain an IP address automatically [✓ Use the following IP address:]

IP address: 192 . 168 . 2 . 2

Subnet mask: 255 . 255 . 255 . 0

Default gateway: 192 . 168 . 2 . 1

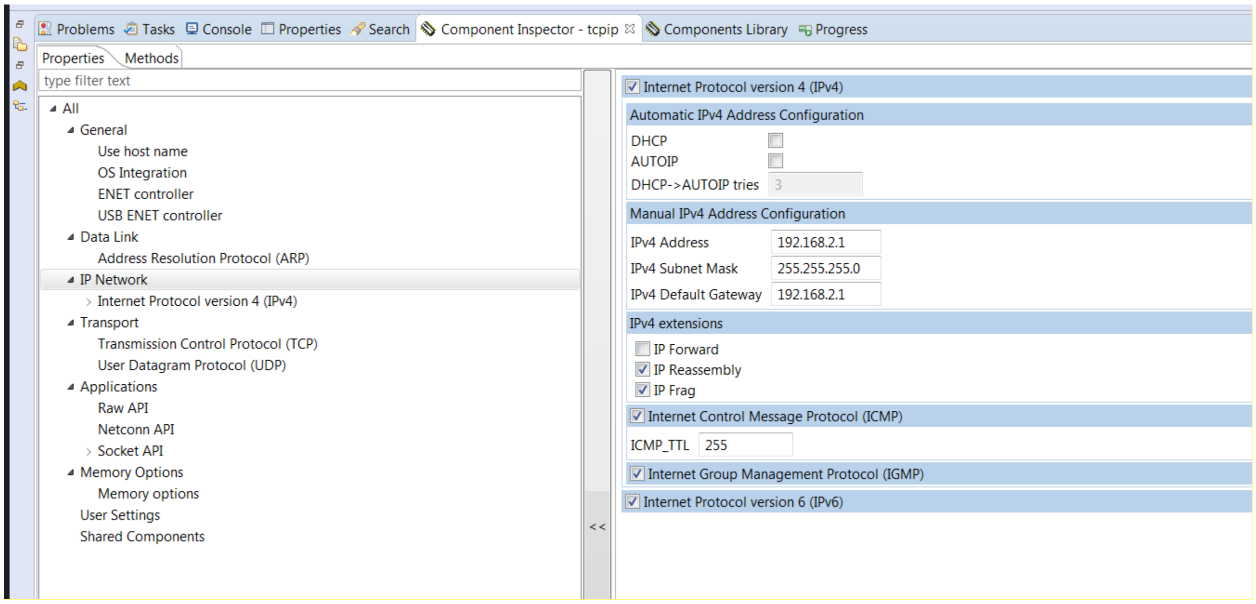
Obtain DNS server address automatically [✓ Use the following DNS server addresses]

Preferred DNS server: . . .

Alternate DNS server: . . .

Validate settings upon exit

- By default, the **IP set in the example is "192.168.2.1", and the MAC address is 0x223344556677**. To change these, please check `LWIP_MAC_ADDR_BASE`, `LWIP_PORT_INIT_IPADDR` and `LWIP_PORT_INIT_GW`.
- The board IP can also be changed from the tcpip PEx component:



☑ Internet Protocol version 4 (IPv4)

Automatic IPv4 Address Configuration

DHCP	☐
AUTOIP	☐
DHCP->AUTOIP tries	3

Manual IPv4 Address Configuration

IPv4 Address	192.168.2.1
IPv4 Subnet Mask	255.255.255.0
IPv4 Default Gateway	192.168.2.1

IPv4 extensions

☐ IP Forward
☑ IP Reassembly
☑ IP Frag

☑ Internet Control Message Protocol (ICMP)

ICMP_TTL 255

☑ Internet Group Management Protocol (IGMP)

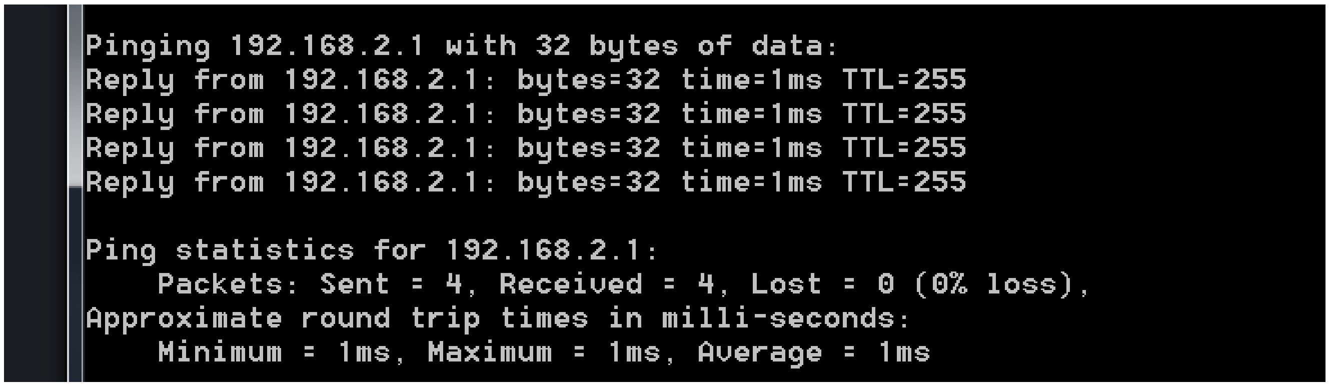
☑ Internet Protocol version 6 (IPv6)

5.6 Building the demo application for MPC5748G device from SDK

1. Run SDK EAR installer. It will install on known path.
2. Start S32 DS
3. File -> New -> S32DS Project from Example -> usb_cdc_lwip
4. Build.

5.7 Expected results:

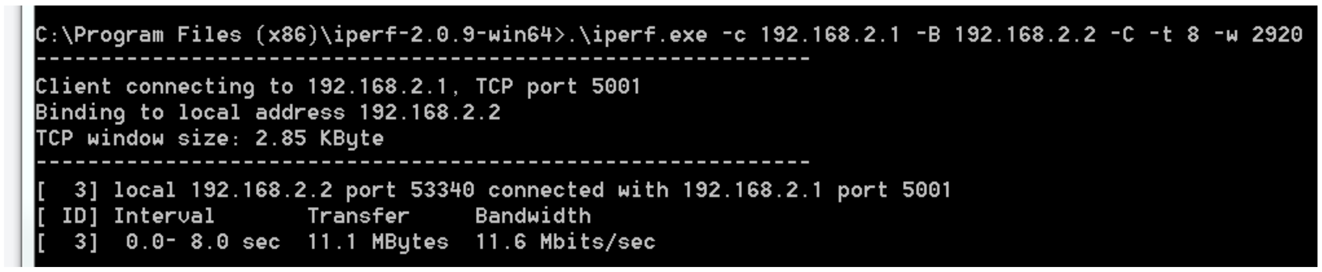
Ping the device:



```
Pinging 192.168.2.1 with 32 bytes of data:
Reply from 192.168.2.1: bytes=32 time=1ms TTL=255
Reply from 192.168.2.1: bytes=32 time=1ms TTL=255
Reply from 192.168.2.1: bytes=32 time=1ms TTL=255
Reply from 192.168.2.1: bytes=32 time=1ms TTL=255

Ping statistics for 192.168.2.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 1ms, Average = 1ms
```

Test the throughput using iperf. Please note that for best performances, the TCP windows should be equal to the one defined in the TCP/ip stack(see *TCP_SND_BUF*).



```
C:\Program Files (x86)\iperf-2.0.9-win64>.\iperf.exe -c 192.168.2.1 -B 192.168.2.2 -C -t 8 -w 2920
------------------------------------------------------------
Client connecting to 192.168.2.1, TCP port 5001
Binding to local address 192.168.2.2
TCP window size: 2.85 KByte
------------------------------------------------------------
[  3] local 192.168.2.2 port 53340 connected with 192.168.2.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 8.0 sec  11.1 MBytes  11.6 Mbits/sec
```

6. How to Reach Us:

Home Page: nxp.com

Web Support: nxp.com/support