# AN14253

## USB to CAN-FD Adapter based on MCXN Microcontroller

Rev. 1.0 — 16 April 2024 Application note

**Document information**

| Information | Content |
|---|---|
| Keywords | AN14253, MCXN, MCXA, MCX_N9XX_EVK boards, MCX_N9XX_FDRM boards, Software Development Kit (SDK) |
| Abstract | This document describes two demo examples to build a USB to CAN-FD adapter using MCX_N9XX_EVK and MCX_N9XX_FDRM boards. |

# 1 Introduction

This application note provides two demo examples to build a USB to CAN-FD adapter where the USB retransmits data to the CAN-bus and vice versa. It uses MCX_N9XX_EVK and MCX_N9XX_FDRM boards for the demo. NXP MCXN devices have a high-speed (HS) USB port and CAN-FD controllers. HS USB can reach up to 480 Mbit/s transmission speed, which is enough for transmitting CAN-FD frame at highest CAN baud rate on MCXN 8 Mbit/s.

To make the system easy to use and compatible with other devices, the examples use USB CDC virtual COM port for communication. A Python GUI is used to display the CAN-FD information in ASCII format.

# 2 CAN-FD

CAN-FD is defined in the international standard ISO 11898-1:2015. This section introduces the key features of CAN-FD for the users who are familiar with using CAN. For more information about using CAN, visit the URL: community.nxp.com/CAN.

## 2.1 Differences between CAN and CAN-FD

There are two key differences between classical CAN and CAN-FD. The first is that CAN-FD can use higher bit rates than classical CAN. Classical CAN is limited to 1 Mbit/s. CAN-FD does not have a theoretical limit, but in practice it is limited by the transceivers. The second key difference is the increased amount of data per CAN message. Classical CAN is limited to 8 data bytes. CAN-FD limit is 64 data bytes per message, which is an eight-fold increase from the CAN limit. With the increased amount of data per CAN message, CAN-FD frames need higher bit rate to decrease the delay time in the communication and increase real-time performance. The CAN-FD frames can reach higher bit rates by enabling the bit rate switch feature. On the other hand, though the bit rate is higher, the bit time is shorter. To enable a data phase bit time that is even shorter than the transmitter delay, delay compensation is introduced. Without transmitter delay compensation, the bit rate in the data phase of a CAN-FD frame is limited by the transmitter delay.

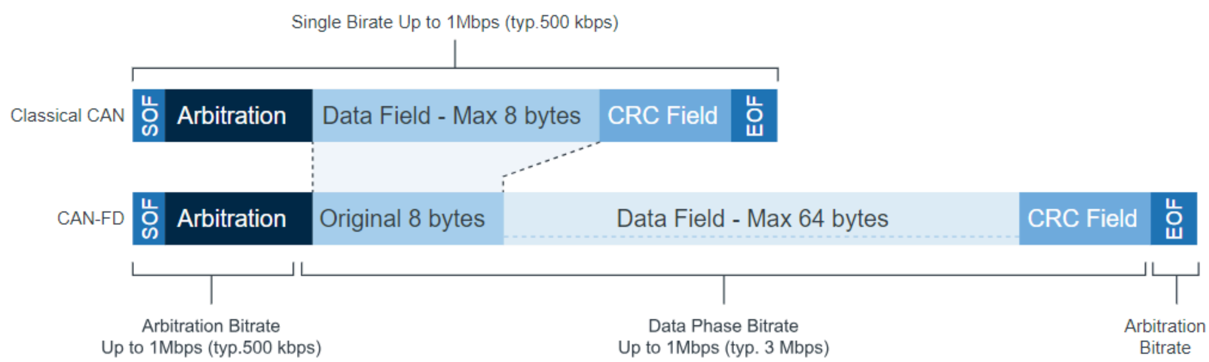



**Figure 1. CAN-FD versus CAN frames**

# 3 USB CDC class driver

The USB Communications Device Class (or USB CDC) is a composite Universal Serial Bus device class. The class includes several interfaces, such as custom control interfaces, data interfaces, audio, or mass storage-related interfaces. In such cases, a USB interface can be used to implement the function of the Virtual COM Port (VCOM). The VCOM port on the PC helps perform communication between the PC and the embedded system. More information about USB can be obtained via the URL: USB basic training.

# 4 Demo implementation

## 4.1 Overview

USB CDC uses two USB physical buck endpoints to transfer data between PC and MCU. Each endpoint is responsible for uni-directional data transfer.

The example uses two buffers for each pipe. One is used for communication between USB to CAN-FD bus and the other for CAN-FD bus to USB. Once data is on MCU, it is responsible to use the information obtained to create the CAN-FD frame and send it and in the opposite direction. The MCU receives the CAN-FD frame and then extracts the data from the frame to send it using the USB CDC to the PC.

## 4.2 Related SDK examples

To implement the steps listed in this application note, users must have the preliminary knowledge of USB CDC and CAN-FD usage. Both the below examples are available in the MCXN SDK:

- **`mcxn9xxevk_flexcan_interrupt_transfer`** example:

The FlexCAN interrupt example shows how to use the FlexCAN driver in a non-blocking interrupt way.

In this example, 2 boards are connected through a CAN bus. Endpoint A (board A) sends a CAN message to Endpoint B (board B) when the user presses any key in the terminal. Endpoint B receives the message, prints the message content to the terminal, and echoes back the message. Endpoint A increases the received message and waits for the next transmission of the user to be initiated.

- **`mcxn9xxevk_dev_cdc_vcom_bm`** example:

The Virtual COM project is a simple demonstration program based on the SDK. It is enumerated as a COM port, which the users can open using terminal tools, such as Teraterm. The demo echoes back any character that that it receives. The purpose of this demo is to show how to build a device of USB CDC class and to provide a simple project for further development.

Both examples can be imported from the MCXN SDK available at the URL: Welcome | MCUXpresso SDK Builder (nxp.com). Users must be familiar with the above two examples before further reading. Those two examples are the building blocks for the USB-CAN adapter design.

## 4.3 Hardware

The examples described in this Application Note use the MCX_N9XX_EVK and MCX_N9XX_FDRM boards. These boards have the USB PHY and the CAN transceiver available for use without any hardware rework required in the boards. The appropriate hardware to use must be selected in the `board.h` file using the below macros:

```
/*! @brief the board name */
#define MCX_N9XX_EVK    (1U)
#define MCX_N9XX_FRDM   (2U)

#define BOARD_NAME MCX_N9XX_EVK
```

### 4.3.1 MCX-N9XX-EVK board

Table 1 shows the GPIO pin functions used for the USB-CAN adapter example on MCX-N9XX-EVK board.

Table 1. GPIO pins used in USB-CAN adapter on MCX-N9XX-EVK board

| Function | GPIO | Description |
|---|---|---|
| CAN0_TX | P1_18 | CAN bus transmission signal |
| CAN0_RX | P1_19 | CAN bus reception signal |
| USB1_DM | USB1_DM | HS USB DM |
| USB1_DP | USB1_DP | HS USB DP |
| UART_RXD | P1_8 | Debug UART RXD |
| UART_TXD | P1_9 | Debug UART TXD |

### 4.3.2 MCX-N9XX-FRDM board

Table 2 shows the GPIO pin functions for the USB-CAN adapter example on MCX-N9XX-FRDM board.

Table 2. GPIO pins used in USB-CAN adapter on MCX-N9XX-FRDM board

| Function | GPIO | Description |
|---|---|---|
| CAN0_TX | P1_10 | CAN bus transmission signal |
| CAN0_RX | P1_11 | CAN bus reception signal |
| USB1_DM | USB1_DM | HS USB DM |
| USB1_DP | USB1_DP | HS USB DP |
| UART_RXD | P1_8 | Debug UART RXD |
| UART_TXD | P1_9 | Debug UART TXD |

## 4.4 Software

The software is based on two BareMetal SDK examples: USB Device CDC VCOM and FlexCAN interrupt. Once both of them are integrated, a simple serial protocol is adapted in the application on top of them. This protocol enables conversion of CAN messages into ASCII serial messages that are sent over the USB Device CDC. In the case of this example, the messages are sent to the python interface and vice versa.

Follow the steps listed below to create the USB to CAN project example:

1. Use `mcxn9xxevk_dev_cdc_vcom_bm` as baseline.
2. Integrate `mcxn9xxevk_flexcan_interrupt_transfer` demo.
3. Copy CAN TxD and RxD pin configurations into the `pin_mux.c` file.
4. Integrate `fls_flexcan` driver to the project in the `drivers` folder.
5. Integrate the functions in `flexcan_interrupt_transfer.c` file.
6. Create the adaptation layer were the CAN message is converted into serial message and vice versa
7. In USB callbacks (***USB_DeviceCdcVcomCallback***), identify where to process the received message to convert into CAN message and send it.
8. In CAN callback, identify the reception complete message to know when to convert CAN frame into serial message and send it using USB CDC.

The MCXN software example is available at: https://github.com/nxp-appcodehub/an-usb-to-can-adaptor-mcxn947.

Figure 2 shows the high level block diagram design for this example.

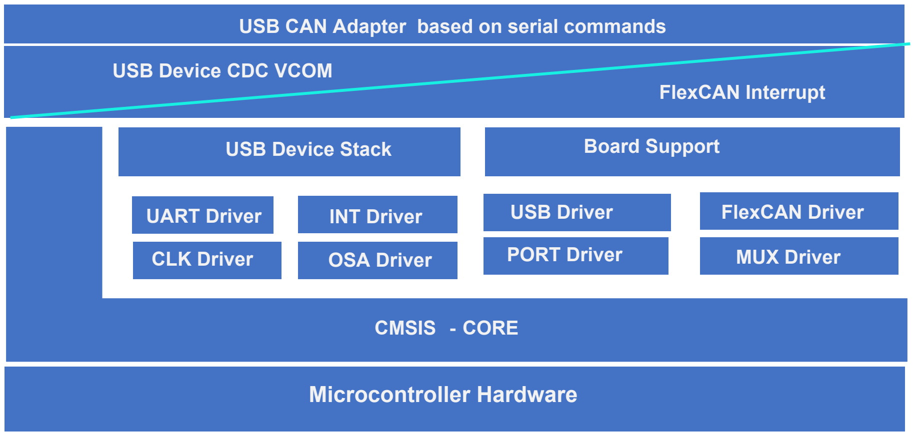



**Figure 2. Software block diagram**

The main functions for the application are located in the files described in Table 3.

**Table 3. CAN files**

| S.No | File name | Description |
|---|---|---|
| 1 | *can_interface.c* | File with all CAN related functions such as CAN send, CAN receive, and FlexCAN initialization functions. |
| 2 | *usb_cdc_vcom.c* | File with all USB related functions. Contains USB CDC send, USB CDC receive, and USB initialization functions. |
| 3 | *usb_to_can.c* | File that supports the serial protocol with reception inputs functions to parse the messages. |
| 4 | *usb_can_adapter.c* | File with the main function to call the initializations. |

## 4.5 Serial command frames

The USB-CAN adapter registers as a virtual serial port on the host computer. To provide an easy human interaction with the interface, the CAN commands are received in the Python interface as ASCII characters. Similarly, the interface sends ASCII commands that are converted into CAN commands before being sent.

For this purpose, the frames must be created in the specific format displayed in Table 4.

**Table 4. Frame format**

| FD ID | Frame Start | CAN ID | DLC | Data |
|---|---|---|---|---|
| 2 characters | 1 character | 3 characters | 1 character | 2 to 128 characters depending on DLC |

- **FD ID**: Characters “FD” to identify if the frame is CAN-FD or not.
- **Frame Start**: ACII character ‘s’ or ‘S’ use to identify the start of CAN frame.
- **CAN ID**: 3 characters with valid values from “0 to 9” or “A to F” that corresponds to the hexadecimal value of the real CAN ID.
- **DLC**: A single character. Valid DLC options are listed in Table 5.

**Table 5. Valid DLC options**

| DLC value | Byte length | Number of characters |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 2 | 4 |

**Table 5. Valid DLC options**...*continued*

| DLC value | Byte length | Number of characters |
|---|---|---|
| 3 | 3 | 6 |
| 4 | 4 | 8 |
| 5 | 5 | 10 |
| 6 | 6 | 12 |
| 7 | 7 | 14 |
| 8 | 8 | 16 |
| 10 | 16 | 32 |
| 13 | 32 | 64 |
| 15 | 64 | 128 |

- Data: 2 to 128 characters with valid values from “0 to 9” or “A to F” that corresponds to the hexadecimal value in the CAN Frame.

An example of the frame format below is described in Table 6.

Frame example: FDs12381122334455667788

**Table 6. Example of frame format**

| FD ID | Frame Start | CAN ID | DLC | Data |
|---|---|---|---|---|
| FD | s | 123 | 8 | 1122334455667788 |

## 4.6 Python GUI Interface

Python is one of the programing languages that has developed much relevance in recent years. The community has developed useful libraries and tools that allow process automation and interface development.

This example uses Python revision 3.10.10 along with the Tkinter module and the pySerial library. All these tools are widely documented on the web and there are many good examples to take as a baseline. The code for this example is included in the project in the `python_gui` folder.

## 4.7 Interface description

Figure 3 shows the Python application GUI.

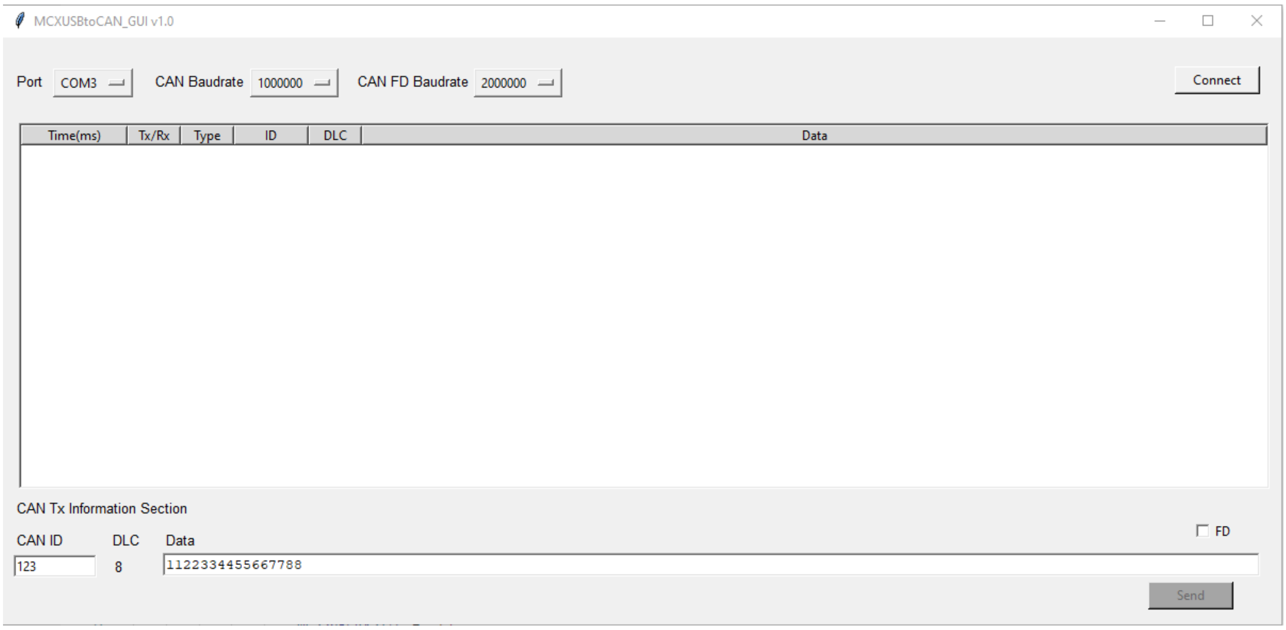



**Figure 3. Python interface GUI**

- **Port:** This listbox allows you to select the COM port for your USB CDC board.
- **CAN Baudrate**: Selects the arbitration phase baud rate.
- **CAN-FD Baudrate**: Selects the data phase baud rate.
- **Connect** button: Must be clicked once the port and baud rates are selected. This starts the serial communication with our device.
- **CAN Tx Information Section**: In this centre window, user is able to see the received and transmitted CAN messages
- **FD**: This checkbox allows to select either CAN or CAN-FD transmissions. This check box does not control the microcontroller configuration, only the serial message to be transmitted through serial.
- **CAN ID**: Select the CAN ID to send a message.
- **DLC**: Indicates the DLC for the length data. In case that data length is not allowed it shows an error.
- **Data**: Message to be transmitted. The length must be even numbers from 2 to 16, 32, 64, or 128 characters accordingly the DLC description.
- **Send** button:

# 5 Running the demo

The following examples demonstrate the use of the USB to CAN adapter to communicate with a CAN device or to monitor a communication in a CAN network.

## 5.1 Direct communication

This example requires two boards. One board runs the USB to CAN adapter code and the other runs `mcxn9xxevk_flexcan_interrupt_transfer` demo.

Prepare the example:

- Connect a USB cable between J5 debug USB port to the PC host in both boards.
- Connect a USB cable between the PC host and the J27 USB device port on the board that will run the USB to CAN code.
- Board to board CAN connections are described in Figure 4.

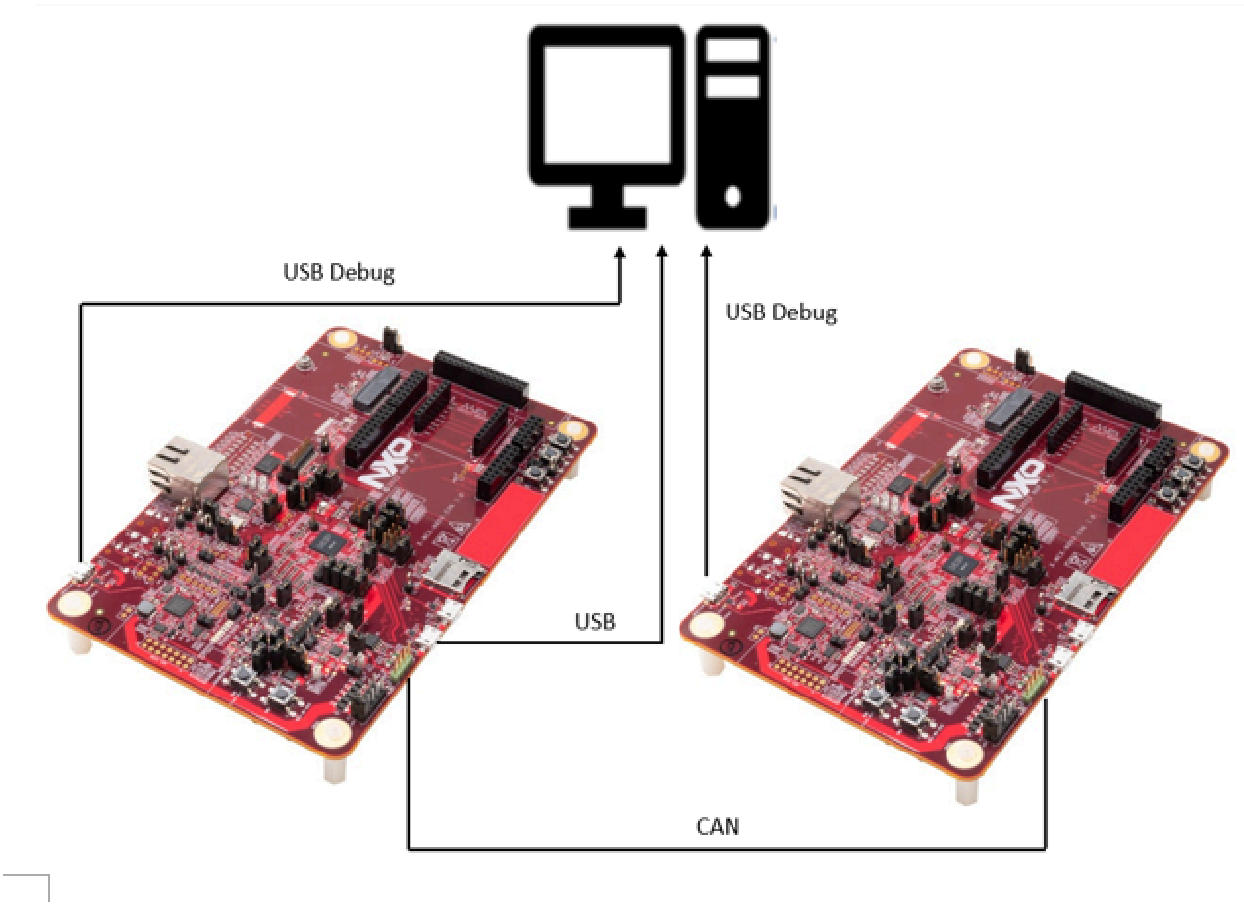



**Figure 4. Board to board CAN connections**

**Table 7. Serial Terminal after running the demo**

| Node A USB to CAN | | Node B CAN interrupt demo | |
|---|---|---|---|
| **Signal name** | **Board location** | **Signal name** | **Board location** |
| CANH | J29-1 | CANH | J29-1 |

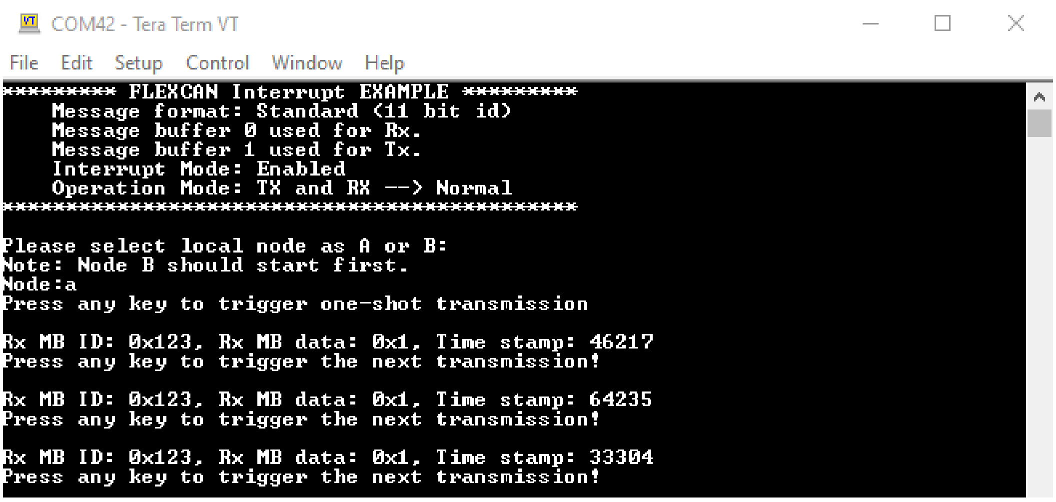
**Table 7. Serial Terminal after running the demo***...continued*

| Node A USB to CAN | | Node B CAN interrupt demo | |
|---|---|---|---|
| **Signal name** | **Board location** | **Signal name** | **Board location** |
| CANL | J29-2 | CANL | J29-2 |
| GND | J29-4 | GND | J29-4 |

- Download the example code to both the boards. One board must be programmed with the USB to CAN adapter source code that comes along with this application note. The other board must be programmed with the `flexcan_interrupt_transfer` demo imported directly from the MCXN9 SDK.
- From the board with the `mcxn9xxevk_flexcan_interrupt_transfer` demo, open a serial terminal on PC with the settings mentioned below:
  - 115200 baud rate
  - 8 data bits
  - No parity
  - One stop bit
  - No flow control
- Either press the reset button on your board or launch the debugger in your IDE to begin running the demos.

Run the example:

1. Open the Python interface `MCXUSBtoCAN_GUI.py` or `MCXUSBtoCAN_GUI.exe`.
2. Select the COM that corresponds to the USB CDC.
3. In this example, the CAN Baud rate used is 1000000 and CAN-FD Baud rate is 2000000.
4. Click the **Connect** button.
5. Set the **FD** checkbox.
6. On the `mcxn9xxevk_flexcan_interrupt_transfer` demo, select node A as the option.
7. Press any key on the serial terminal to send a CAN message.
8. Write the value 01 in Data section and click **Send** button.
9. Now, repeat steps 7 and 8. The **mcxn9xxevk_flexcan_interrupt_transfer** demo waits in a loop after sending a CAN message to receive a message and after receiving the message, it waits until a CAN message is sent using the terminal.



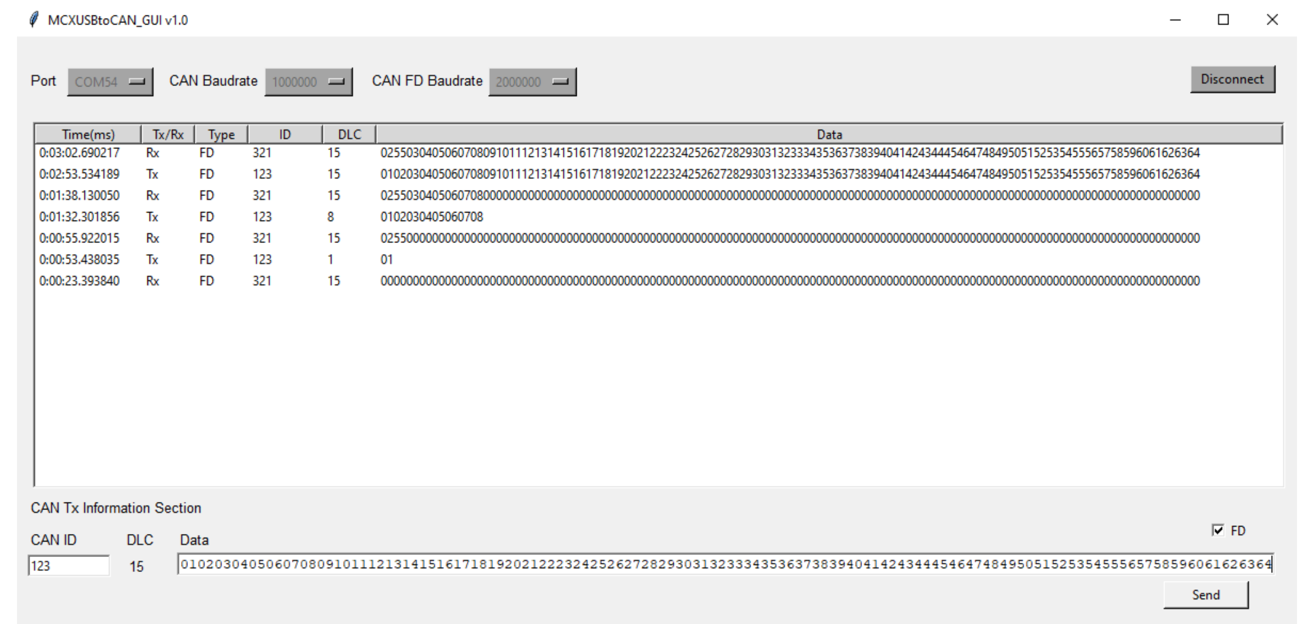



**Figure 5. Python GUI interface after running the demo**

## 5.2 Monitoring the CAN network

This example demonstrates a CAN network with two or more devices. The objective is to monitor all the traffic on the network. The example requires a MCX_N9XX_EVK or a MCX_N9XX_FDRM board with the USB to CAN adapter code connected to the network. This demonstration uses two NXP boards running the **can_interrupt_transfer** demo in both boards as the CAN network.

**Preparing the example:**

1. Connect a USB cable between the J5 debug USB port to the PC host on the board.
2. Connect a USB cable between the PC host and the J27 USB device port on the board.
3. CAN connections to the network is shown in Figure 6:

**Table 8. Can connections: Node A USB to CAN**

| Signal Name | Board Location |
| --- | --- |
| CANH | J29-1 |
| CANL | J29-2 |
| GND | J29-4 |

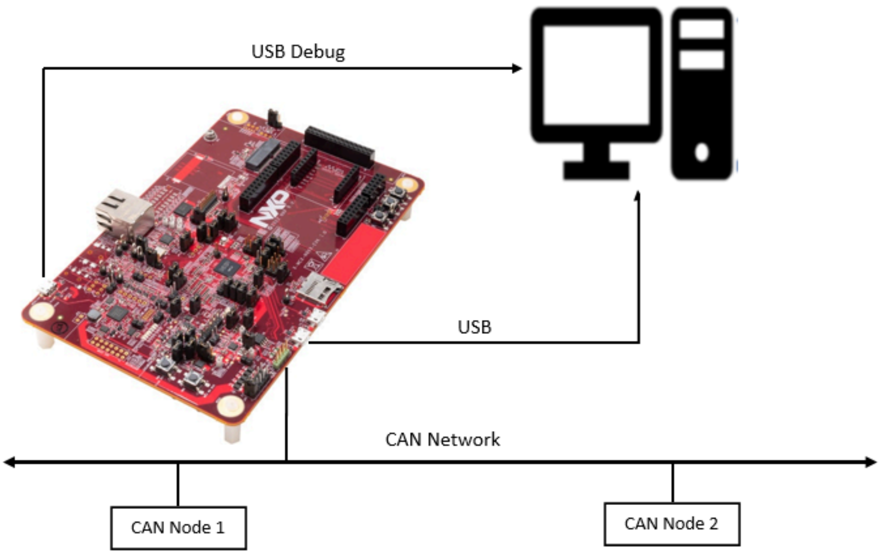



**Figure 6. CAN connections to the network**

4. Download the example code to the board.
5. Either press the **Reset** button on your board or launch the debugger in your IDE to begin running the demos.

### Running the example

1. Open Python interface `MCXUSBtoCAN_GUI.py` or `MCXUSBtoCAN_GUI.exe`
2. Select the COM that corresponds to USB CDC.
3. This example uses the CAN Baud rate of 1000000 and CAN-FD Baud rate as 2000000.
4. Click the **Connect** button.
5. Set the **FD** checkbox.
6. Start transmitting data on the CAN network and check the CAN traffic displayed in the window section. See Figure 7.

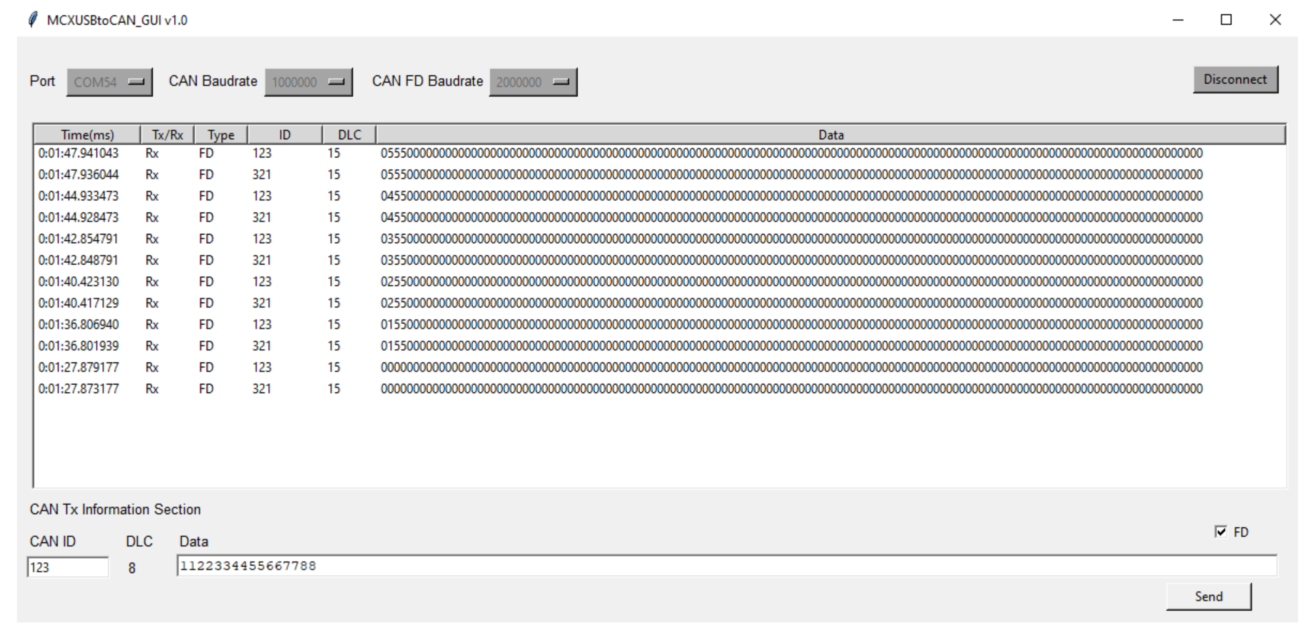



**Figure 7. Python Interface showing the CAN traffic on the network**

# 6 Acronyms

Table 9 lists and explains the acronyms and abbreviations used in this document.

**Table 9. Acronyms**

| Term | Description |
|---|---|
| CAN | Controller Area Network |
| CDC | Communications Device Class |
| CAN-FD | CAN with Flexible Data-Rate |
| IDE | Integrated Design Environment |
| MCU | Microcontroller Unit |
| SDK | Software Development Kit |
| USB | Universal Serial Bus |
| VCOM port | Virtual COM (communication) port |

## 7 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:



## 8 Revision history

Table 10 lists the revisions made to this document.

**Table 10. Revision history**

| Document ID | Release date | Description |
|---|---|---|
| AN14253 v.1.0 | 16 April 2024 | Initial public release |

# Legal information

## 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.

## 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 https://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 document 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.

## 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.

**MCX** — is a trademark of NXP B.V.

## Contents





**Date of release: 16 April 2024**
**Document identifier: AN14253**