AN13515

USB CAN Adapter based on LPC55S16

Rev. 0 — 18 January 2022 Application Note

Contents

1	Introduction	1
1.1	Overview	1
2	Implementation	1
2.1	Overview	1
2.2	Related SDK example	1
2.3	Hardware	2
2.4	Serial communication protocol	2
3	Hands on with USBtinViewer	5
3.1	Connecting hardware to USBtinViewer	6
4	Third-party and community resources	9
5	Schematic of USB-CAN-adapter	10
6	Reference	10
7	Revision history	10

1 Introduction

1.1 Overview

This application note aims to build a USB-CAN adapter where the USB data retransmit to CAN-bus and vice versa. NXP LPC55S16 has a high-speed USB port and CANFD controllers. HSUSB can reach up to 480 Mbit/s transmission speed, which is enough for transmitting CANFD frame at highest CAN baud rate.

To make the system easy to use and compatible with other devices, we use USB CDC virtual COM port as communication interface with PC and a simple ASCII protocol inherited from open-source project USBtinViewer.

2 Implementation

2.1 Overview

As shown in Figure 1, USB CDC uses two USB physical buck endpoints to transfer data between PC and MCU. Each endpoint is responsible for uni-directional data transfer.

SDK already provides MCAN driver and USB Stack. In software, add two buffers for each pipe, one for USB -> CAN bus and the other for CAN bus -> USB. To ensure the best performance, the two pipes are independent.

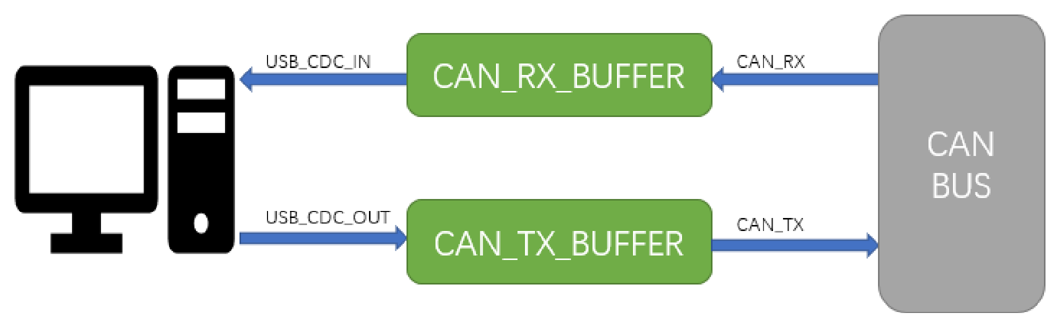

Figure 1. System block diagram

2.2 Related SDK example

Before continuing the task, we need the background knowledge of USB CDC and CAN usage. Fortunately, SDK provides two examples.

- MCAN loopback example

MCAN example is a simple CAN loopback example which demonstrates usage of LPC54608's CAN module. This example enables the internal loopback of CAN module and send a CAN frame. The CAN frame loops back into CAN receiver and MCU displays any received CAN frames on UART terminal. To get familiar with this example, read the `readme` documentation and run the example.

Example location: *\boards\lpcxpresso55s16\driver_examples\mcan\loopback*

- `usb_device_cdc_vcom` example

  This example is USB CDC class example to enumerate USB as a communication device class. When USB enumeration completes, a COM port pops out on the device. Any character sent through this COM port is loop back to display. See the `readme` documentation for this example for how to install device driver and run the demo.

  Example location: *\boards\lpcxpresso55s16\usb_examples\usb_device_cdc_vcom\bm*

Be familiar with above two examples before continue reading. Those two examples are building blocks for USB-CAN adapter design.

2.3 Hardware

Table 1 describe GPIO pins used in USB-CAN adapter.

Table 1. GPIO pins used in USB-CAN adapter

GPIO	Function	Description
`PIO1_2`	`CAN0_TX`	CAN bus signal
`PIO1_3`	`CAN0_RX`	CAN bus signal
`USB1_DM`	`USB1_DM`	HSUSB DM
`USB1_DP`	`USB1_DP`	HSUSB DP
`PIO0_29`	`UART_RXD`	Debug UART RXD
`PIO0_30`	`UART_TXD`	Debug UART TXD

For full schematic, see Schematic of USB-CAN-adapter.

2.4 Serial communication protocol

USB-CAN adapter registers as a virtual serial port on the host computer. With simple ASCII commands, CAN bus configuration can be controlled over this serial port. You can send/receive commands from any serial terminal program or from your own program.

Table 2. ASCII protocol commands list:

ASCII commands	Response	Description
O[CR]	[CR]	Open CAN channel
C[CR]	[CR]	Close CAN channel
tiiildd..[CR]	Transmit standard (11 bit) frame.	iii: Identifier in hexadecimal format (000-7FF) l: Data length (0-8) dd: Data byte value in hexadecimal format (00-FF)

Table continues on the next page...

Table 2. ASCII protocol commands list: (continued)

ASCII commands	Response	Description
Sx[CR]	Set baud rate	x: Bitrate id (0-8) S0 = 10 kBaud S1 = 20 kBaud S2 = 50 kBaud S3 = 100 kBaud S4 = 125 kBaud S5 = 250 kBaud S6 = 500 kBaud S7 = 800 kBaud S8 = 1 MBaud
Tiiiiiiiildd..[CR]	Transmit extended (29 bit) frame.	iiiiiiii: Identifier in hexadecimal format (0000000-1FFFFFFF) l: Data length (0-8) dd: Data byte value in hexadecimal format (00-FF)
riiil[CR]	Transmit standard RTR (11 bit) frame.	iii: Identifier in hexadecimal format (000-7FF) l: Data length (0-8)
Riiiiiiiil[CR]	Transmit extended RTR (29 bit) frame.	iiiiiiii: Identifier in hexadecimal format (0000000-1FFFFFFF) l: Data length (0-8)
mxxxxxxxx[CR	Set acceptance filter mask	SJA1000 format (AM0..AM3). Only first 11bit are relevant. xxxxxxxx: Acceptance filter mask
Mxxxxxxxx[CR]	Set acceptance filter code.	SJA1000 format (AC0..AC3). Only first 11bit are relevant. xxxxxxxx: Acceptance filter code

Example:

Set 10 kBaud, open CAN channel, send CAN message (id = 001 h, dlc = 4, data = 11 22 33 44), and close CAN.

Table 3. CAN message

Command	Response
S0[CR]	[CR]
O[CR]	[CR]

Table continues on the next page...

Table 3. CAN message (continued)

Command	Response
t001411223344[CR]	z[CR]
C[CR]	[CR]

With a state machine, the software accepts serial stream from CDC port, parses the ASCII, and applies the command. Below lists some of the important code snippet used in the software. For full source code, see AN13515SW.

- To send a CAN frame,

```
txFrame.xtd  = kMCAN_FrameIDStandard;
txFrame.rtr  = kMCAN_FrameTypeData;
txFrame.fdf  = 0;
txFrame.brs  = 0;
txFrame.dlc  = len;
txFrame.id   = id << STDID_OFFSET;
txFrame.data = buf;
txFrame.size = CAN_DATASIZE;

txXfer.frame     = &txFrame;
txXfer.bufferIdx = 0;
MCAN_TransferSendNonBlocking(EXAMPLE_MCAN, &mcanHandle, &txXfer);
```

- To receive a CAN frame,

```
static void mcan_callback(CAN_Type *base, mcan_handle_t *handle, status_t status, uint32_t
result, void *userData)
{
   switch (status)
   {
      case kStatus_MCAN_RxFifo0Idle:
      {
         memcpy(rx_data, rxFrame.data, rxFrame.size);
         MCAN_TransferReceiveFifoNonBlocking(EXAMPLE_MCAN, 0, &mcanHandle, &rxXfer);
         can_rx_cb(rxFrame.id >> STDID_OFFSET, rx_data, rxFrame.dlc);
       }
       break;

       case kStatus_MCAN_TxIdle:
       {

        }
                    break;

        default:
           break;
   }
}
```

- To send data via USB CDC,

```
void usbd_cdc_send(uint8_t *buf, uint32_t len)
{
    USB_DeviceCdcAcmSend(s_cdcVcom.cdcAcmHandle, USB_CDC_VCOM_BULK_IN_ENDPOINT, buf, len);
}
```

- To receive data from USB CDC,

```
usb_status_t USB_DeviceCdcVcomCallback(class_handle_t handle, uint32_t event, void *param)
{
   switch (event)
   {
   …
          case kUSB_DeviceCdcEventRecvResponse:
          {
             if ((1 == s_cdcVcom.attach) && (1 == s_cdcVcom.startTransactions))
             {
                uint8_t rx_size;
                rx_size = epCbParam->length;
                {
                    error = USB_DeviceCdcAcmRecv(handle, USB_CDC_VCOM_BULK_OUT_ENDPOINT,
cdc_rx_buf, g_UsbDeviceCdcVcomDicEndpoints[1].maxPacketSize);
                 }

                 cdc_rx_cb(cdc_rx_buf, rx_size);
              }
           }
           break;
   }
}
```

3 Hands on with USBtinViewer

To verify the functionality USB-CAN adapter, in this section, we use open-source software USBtinViewer and a commercial USB-CAN-adapter (PCAN-USB).

USBtinViewer can be download from https://www.fischl.de/usbtin/#usbtinviewer.

We use PCAN-USB for commercial USB-CAN adapter and busmaster for software.

Busmaster can be download from https://rbei-etas.github.io/busmaster/.

Figure 2 shows the hardware test environment.

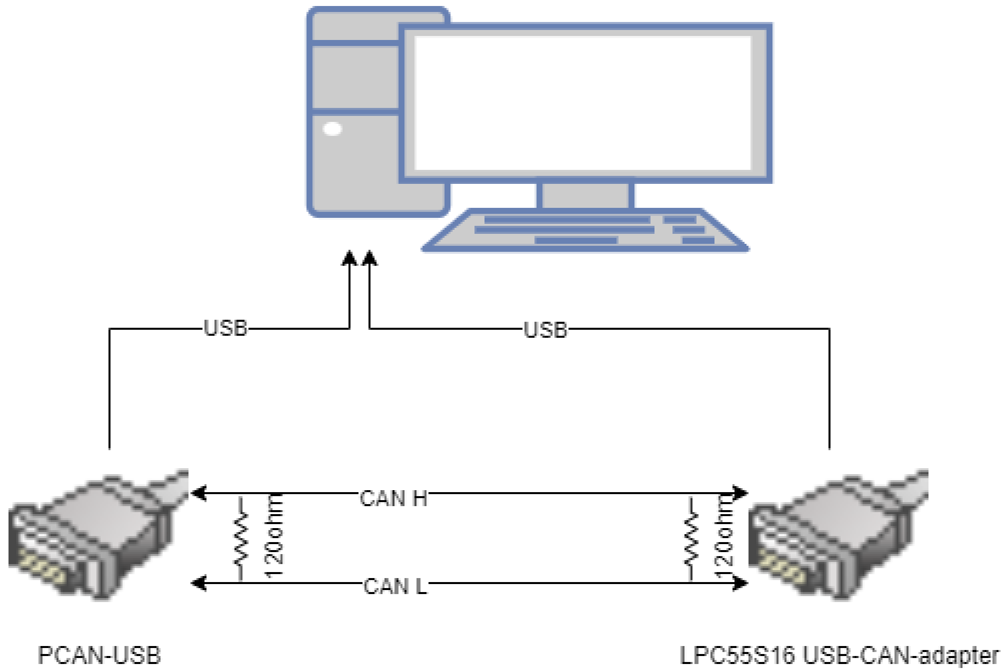

Figure 2. Test environment setup

3.1 Connecting hardware to USBtinViewer

1. Download USBtinViewer and connect the USB port of USB-CAN-adapter to the PC. A USB CDC COM port pops up.

NOTE

The COM port number varies from PC to PC.

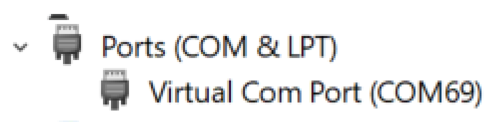

Figure 3. USB CDC port enumeration

Open USBtinViewer, select COM port and CAN baud rate (500 K in this example). Click **Connect**, and the USBtinViewer returns the firmware information, as shown in Figure 4. The information means that connection succeeds.

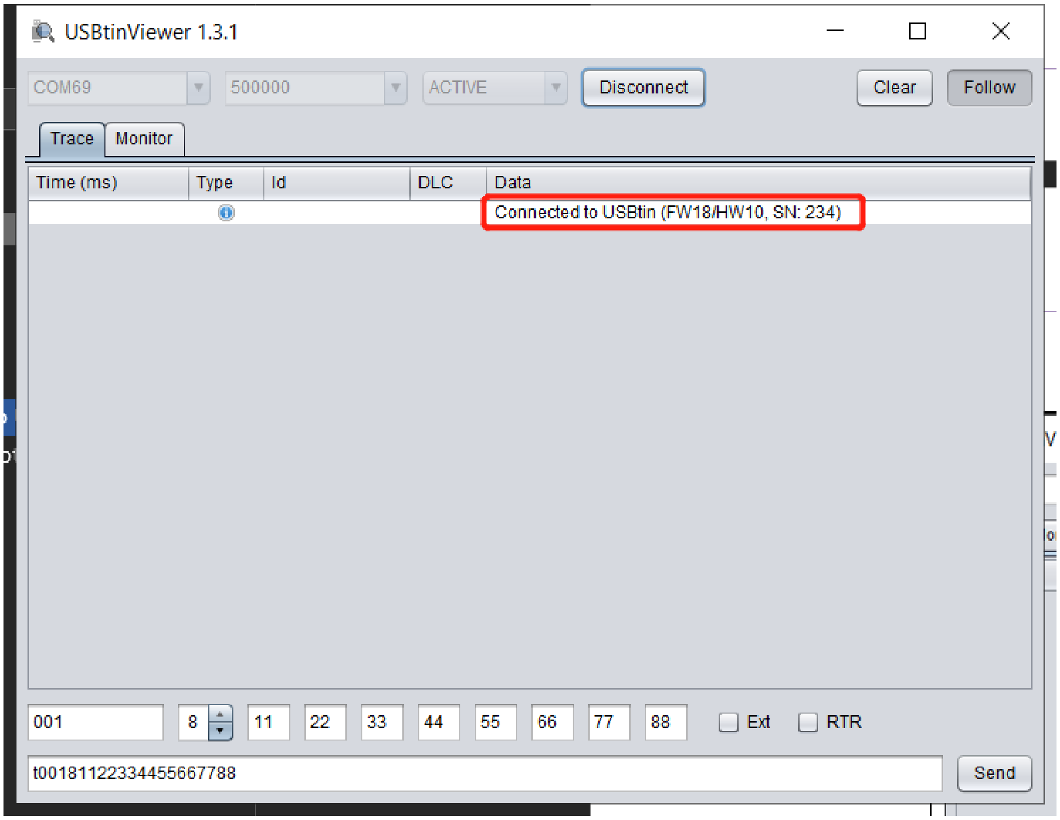

Figure 4. Connecting USB-CAN adapter to USBtin viewer

2. Open busmaster and connect PCAN-USB. Select 500 K baud rate, as shown in Figure 5.

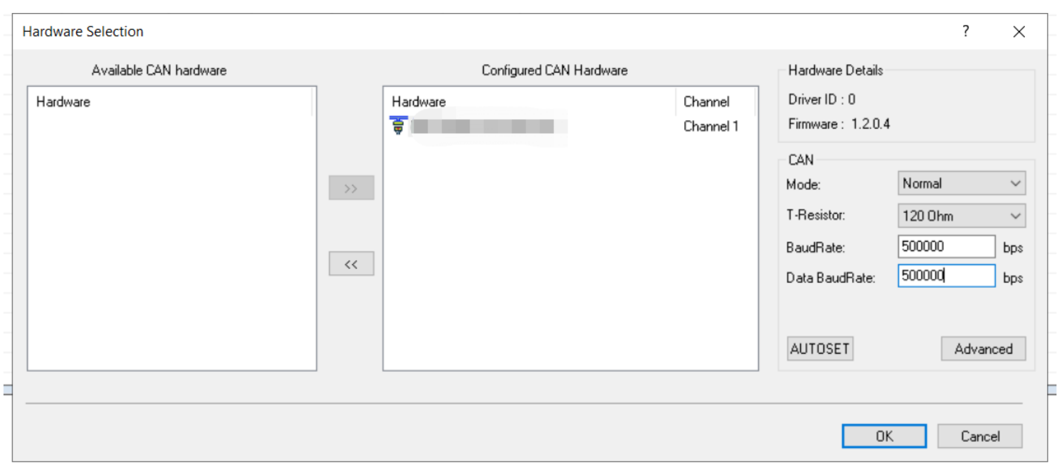

Figure 5. Connecting PCAN-USB to busmaster

3. Send CAN data from USBtinViewer and received by busmaster.

   Connect USB-CAN adapter and PCAN-USB. In the CAN TX box, at the bottom of USBtinViewer, enter the CAN message ID, DLC, and data field. Click **Send**, and the USB-CAN adapter sends the CAN message.

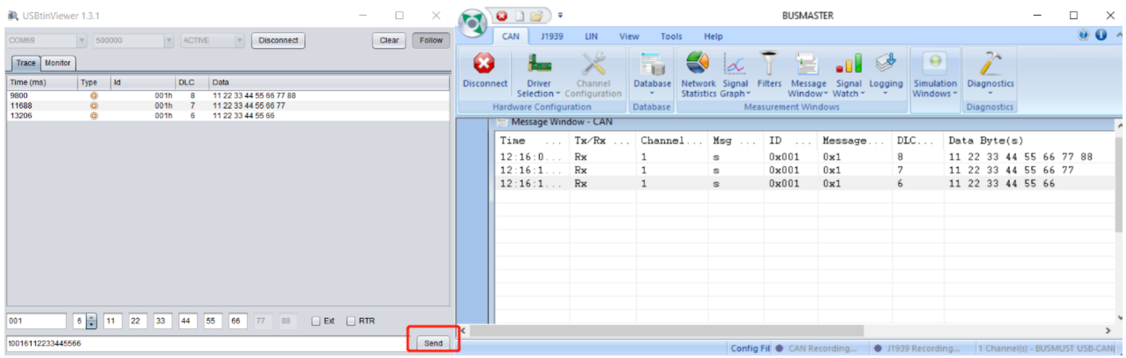

Figure 6. Sending CAN data from USBtinViewer and received by busmaster

In the **Message** window of busmaster, the same CAN message can be received, as shown in Figure 6.

4. Send CAN data from busmaster and verify by USBtinViewer.

   In busmaster, open **Transmit Window**. Click the empty space under the **Message Name** column. Enter the new message name, DLC, and frame data field. Click **Send message**, and the busmaster sends the CAN message. On USBtinViewer, this CAN message can be monitored, as shown in Figure 8.

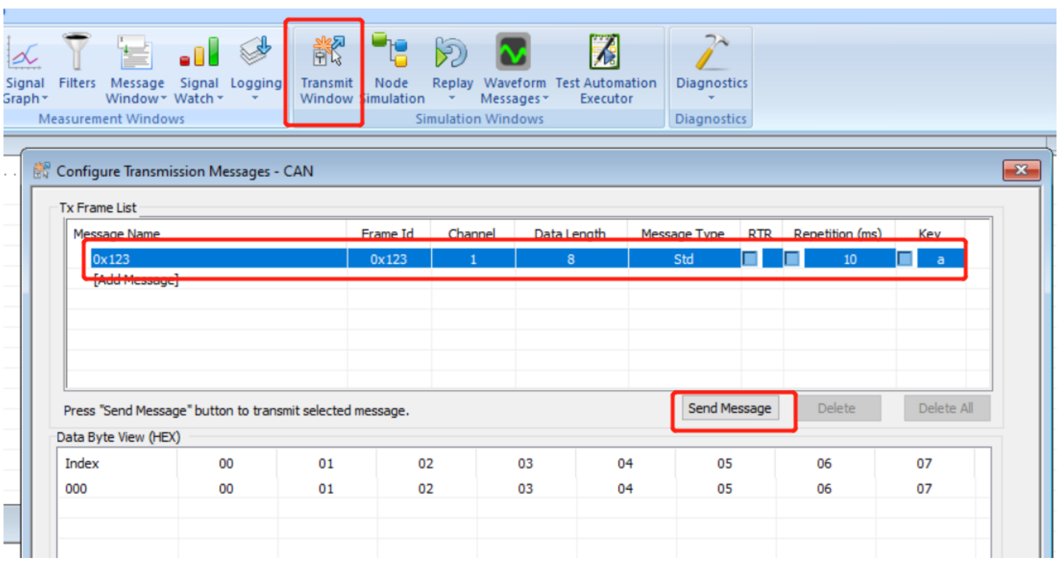

Figure 7. Sending CAN data from busmaster and received by USBtinViewer

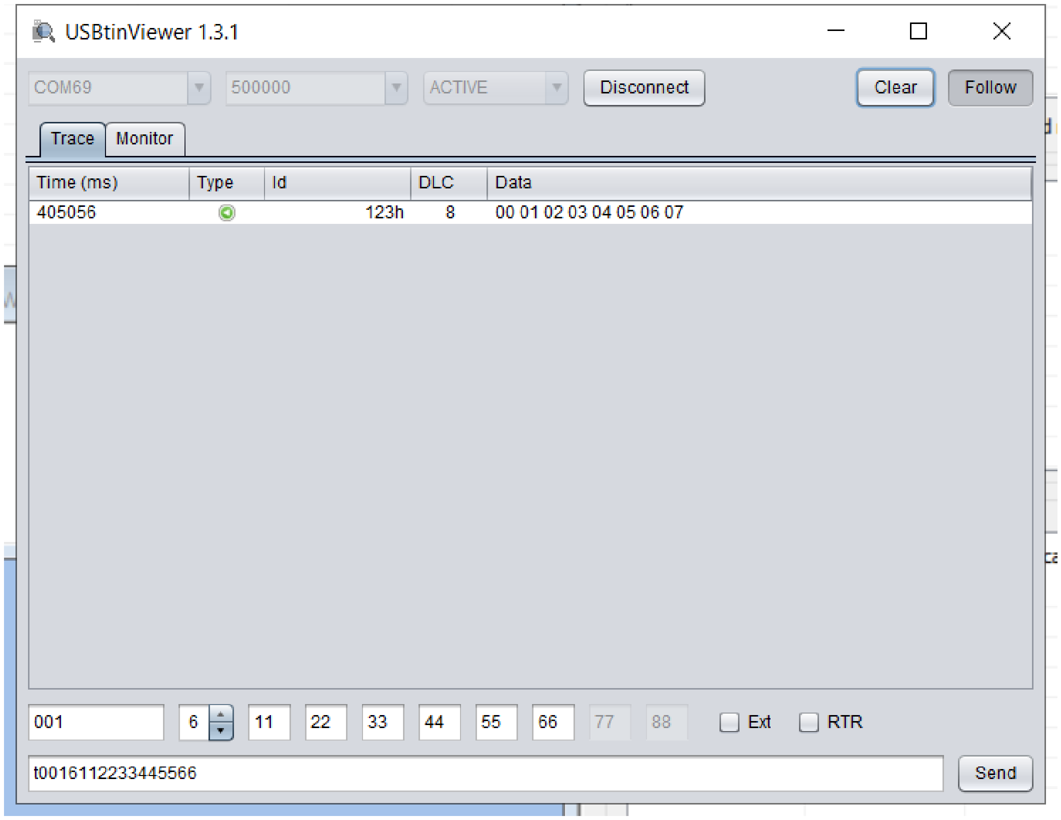

Figure 8. USBtinViewer received CAN message

4 Third-party and community resources

Many useful third-party resources are available on USBtin web page. It includes libraries and tools supporting USBtin. It provides rich resources and supports varies programming languages.

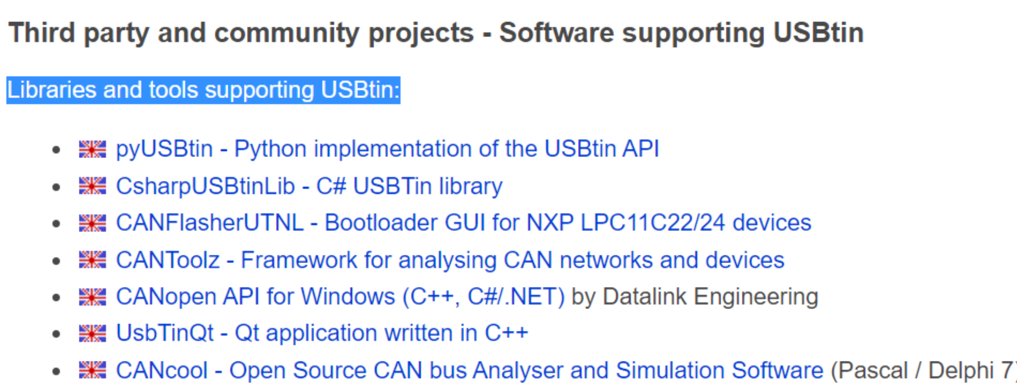

Figure 9. Third-party tools support USBtin

5 Schematic of USB-CAN-adapter

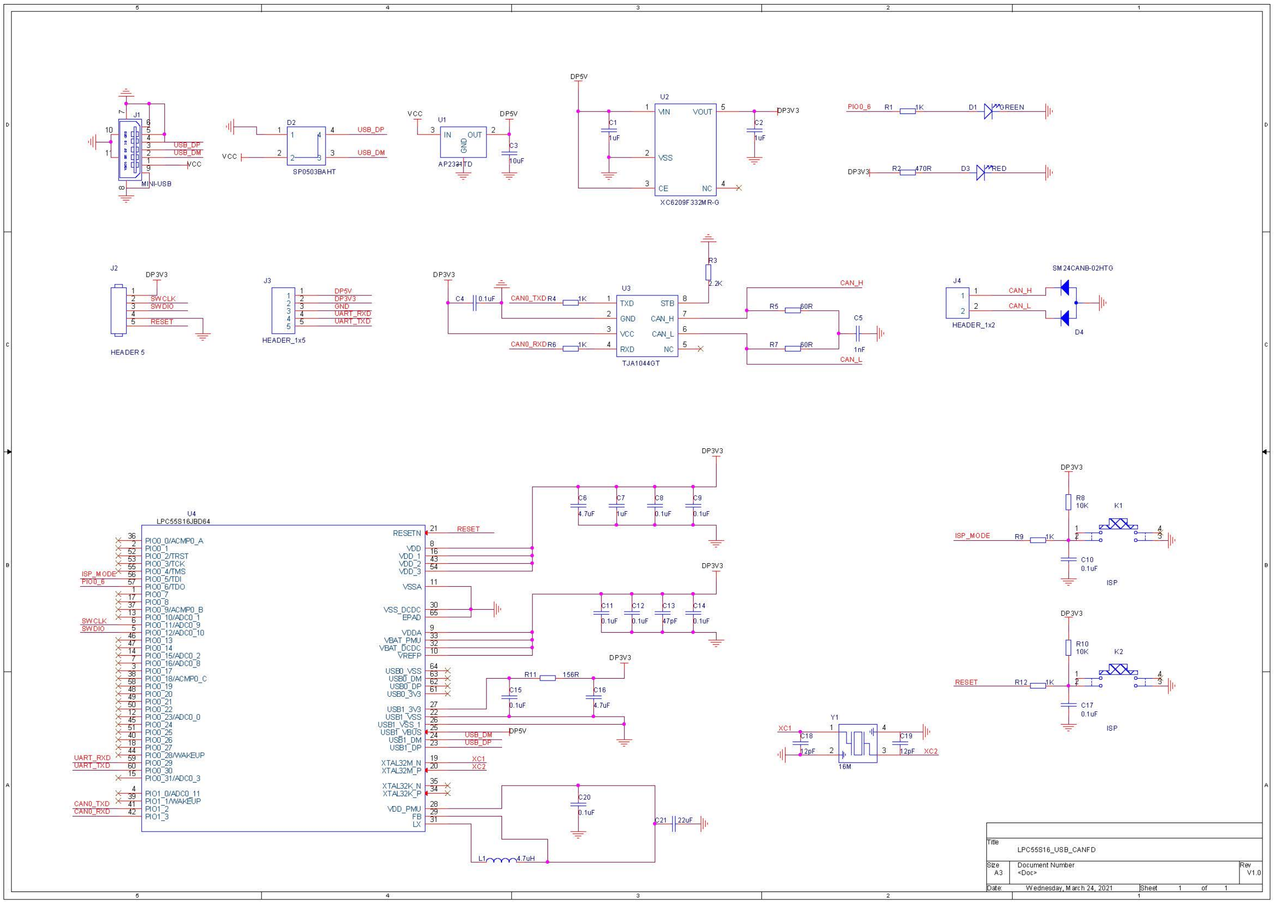

Figure 10. Schematic of USB-CAN-adapter

6 Reference

1. https://www.fischl.de/usbtin/#usbtinviewer
2. https://rbei-etas.github.io/busmaster/

7 Revision history

Rev.	Date	Description
0	18 January 2022	Initial release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 18 January 2022

Document identifier: AN13515