



Develop Linux CAN device driver in the Linux i.MX RT BSP

Detailed Requirements and Design

can-drad-1_2.doc

<i>Revision:</i>	1.2
<i>Date:</i>	5/19/2023

TABLE OF CONTENTS

- 1. OVERVIEW3**
- 2. REQUIREMENTS3**
 - 2.1. Detailed Requirements.....3
 - 2.2. Detailed Non-Requirements3
- 3. DESIGN3**
 - 3.1. Detailed Design3
 - 3.1.1. *Design: Demo project*.....3
 - 3.1.2. *Design: Linux CAN Device Driver*4
 - 3.1.3. *Design: CANSocket*.....4
 - 3.1.4. *Design: CANSocket Test Suite*.....4
 - 3.2. Effect on Related Products.....4
 - 3.3. Changes to User Documentation.....4
 - 3.4. Alternative Design4
- 4. TEST PLAN4**
 - 4.1. Secure Download Area4
 - 4.2. Downloadable Files5
 - 4.3. Test Set-Up.....6
 - 4.3.1. *Hardware Setup*.....6
 - 4.3.2. *Software Setup*.....6
 - 4.4. Detailed Test Plan.....9
 - 4.4.1. *Test Plan: Demo Project*.....9
 - 4.4.2. *Test Plan: Linux CAN Driver*9
 - 4.4.3. *Test Plan: CANSocket*10
 - 4.4.4. *Test Plan: CANSocket Test Suite*11

1. Overview

This project develops Linux CAN device driver in the Linux BSP for the i.MX RT processor.

2. Requirements

2.1. Detailed Requirements

The following are the requirements for this project:

1. Provide a Linux demo project combining all the requirements in this project.
 - *Rationale:* Needed to let Customer integrate results of this project into target embedded application.
Implementation: Section: "Design: Demo project".
Test: Section: "Test Plan: Demo Project".
2. Develop Linux CAN device driver for the i.MX RT CAN controller.
 - *Rationale:* Explicit Customer requirement.
Implementation: Section: "Design: Linux CAN Device Driver".
Test: Section: "Test Plan: Linux CAN Driver".
3. Port CANSocket to the Linux i.MX RT BSP.
 - *Rationale:* Explicit Customer requirement.
Implementation: Section: "Design: CANSocket".
Test: Section: "Test Plan: CANSocket".
4. Validate successful execution of the test suite from the SocketCAN package.
 - *Rationale:* Explicit Customer requirement.
Implementation: Section: "Design: CANSocket Test Suite".
Test: Section: "Test Plan: CANSocket Test Suite".

2.2. Detailed Non-Requirements

The following are the non-requirements for this project that may otherwise not be obvious:

1. None

3. Design

3.1. Detailed Design

3.1.1. Design: Demo project

This project will enable the required CAN functionality in Linux configuration ("embedded project") called `rootfs`, which resides in a `projects/rootfs` directory, relative to the top of the Linux i.MX RT installation.

3.1.2. Design: Linux CAN Device Driver

The internal i.MX clocks necessary for the CAN interface will be enabled in U-Boot. The i.MX RT `flexcan` interface network driver `linux/drivers/net/can/flexcan.c` will be enabled, and the respective changes will be added to the kernel `.dts` file.

The `flexcan` functionality will be enabled in the Linux kernel configuration as follows:

- Go to Networking support -> CAN bus subsystem support -> CAN Device Drivers
- Enable Platform CAN drivers with Netlink support (`CONFIG_CAN_DEV`) and Support for Freescale FLEXCAN based chips (`CONFIG_CAN_FLEXCAN`)

3.1.3. Design: CANSocket

The CAN socket API, described in details in `linux/Documentation/networking/can.txt`, will be enabled using the Raw CAN Protocol (raw access with CAN-ID filtering) and Broadcast Manager CAN Protocol (with content filtering) configuration options in the Networking support -> CAN bus subsystem support configuration menu.

3.1.4. Design: CANSocket Test Suite

The `can-utils` and `can-tests` packages will be used for verification of the functionality implemented in this project.

3.2. Effect on Related Products

This project makes the following updates in the related products:

- None

3.3. Changes to User Documentation

This project updates the following user documents:

- None

3.4. Alternative Design

The following alternative design approaches were considered by this project but then discarded for some reason:

- None

4. Test Plan

4.1. Secure Download Area

The downloadable materials developed by this project are available from a secure Web page on the Emcraft Systems web site. Specifically, proceed to the following URL to download the software materials:

for the i.MX RT1050 BSP:

- <https://www.emcraft.com/imxrtaddon/imxrt1050/can>

for the i.MX RT1060 BSP:

- <https://www.emcraft.com/imxrtaddon/imxrt1060/can>

for the i.MX RT1170 BSP:

- <https://www.emcraft.com/imxrtaddon/imxrt1170/can>

The page is protected as follows:

- Login: *CONTACT EMCRAFT FOR DETAILS*
- Password: *CONTACT EMCRAFT FOR DETAILS*

4.2. Downloadable Files

The following files are available from the secure download area:

For release 2.5.1 (IMXRT1050-EVK BSP):

- 2.5.1/linux-flexcan-2.5.1.patch - patch to the Linux kernel sources;
- 2.5.1/projects-rootfs-2.5.1-addon.tar.bz2 - tarball with the `rootfs` project binary add-on;
- 2.5.1/projects-rootfs-2.5.1.patch - patch to the `rootfs` project;
- 2.5.1/u-boot-flexcan-2.5.1.patch - patch to the U-Boot sources;
- 2.5.1/rootfs.uImage - prebuilt bootable Linux image;
- 2.5.1/u-boot-dtb.imx - prebuilt bootable U-Boot image;

For release 2.5.2 (IMXRT1060-EVK BSP):

- 2.5.2/linux-flexcan-2.5.2.patch - patch to the Linux kernel sources;
- 2.5.2/projects-rootfs-2.5.2-addon.tar.bz2 - tarball with the `rootfs` project binary add-on;
- 2.5.2/projects-rootfs-2.5.2.patch - patch to the `rootfs` project;
- 2.5.2/u-boot-flexcan-2.5.2.patch - patch to the U-Boot sources;
- 2.5.2/rootfs.uImage
- 2.5.2/u-boot-dtb.imx

For release 2.5.3 (IMXRT1170-EVK BSP):

- 2.5.3/linux-flexcan-2.5.3.patch - patch to the Linux kernel sources;
- 2.5.3/projects-rootfs-2.5.2.patch - patch to the `rootfs` project;
- 2.5.3/projects-rootfs-2.5.3-addon.tar.bz2 - tarball with the `rootfs` project binary add-on;
- 2.5.3/u-boot-flexcan-2.5.3.patch - patch to the U-Boot sources;
- 2.5.3/rootfs.uImage - prebuilt bootable Linux image;
- 2.5.3/u-boot.flexspi - prebuilt bootable U-Boot image;

For release 2.5.4 (IMXRT1170-EVK BSP):

- 2.5.4/linux-flexcan-2.5.3.patch - patch to the Linux kernel sources;
- 2.5.4/projects-rootfs-2.5.4.patch - patch to the rootfs project;
- 2.5.4/u-boot-flexcan-2.5.3.patch - patch to the U-Boot sources;
- 2.5.4/rootfs.uImage - prebuilt bootable Linux image;
- 2.5.4/u-boot.flexspi - prebuilt bootable U-Boot image;

Refer to the below sections for the instructions on how to install and use these files.

4.3. Test Set-Up

4.3.1. Hardware Setup

The following hardware setup is required for the i.MX RT 1050/1060 boards:

- The i.MX RT EVK board, with the serial console attached as per <https://emcraft.com/imxrt1060-evk-board/connecting-serial-console-to-imxrt1060-evk>.
- A Linux PC with the `VSCOM USB-CAN USB to CAN Adapter` <http://www.vsc.com.de/vscom-usb-can.html> plugged into a USB port on the PC, and the following connections to the i.MX RT EVK board:
 - DB9.2 of USB-CAN connected to CANL J11.3 on the i.MX RT EVK board.
 - DB9.7 of USB-CAN connected to CANH J11.1 on the i.MX RT EVK board.
 - DB9.3 of USB-CAN connected to GND J11.2 on the i.MX RT EVK board.

The following hardware setup is required for the i.MX RT 1170 boards:

- The i.MX RT EVK board, with the serial console attached as per <https://emcraft.com/imxrt1170-evk-board/connecting-serial-console-to-imxrt1170-evk>.
- A Linux PC with the `VSCOM USB-CAN USB to CAN Adapter` <http://www.vsc.com.de/vscom-usb-can.html> plugged into a USB port on the PC, and the following connections to the i.MX RT EVK board:
 - DB9.2 of USB-CAN connected to CANL J47.3 on the i.MX RT EVK board.
 - DB9.7 of USB-CAN connected to CANH J47.1 on the i.MX RT EVK board.
 - DB9.3 of USB-CAN connected to GND J47.2 on the i.MX RT EVK board.

4.3.2. Software Setup

The following software setup is required:

1. Download the files listed in Section: "Downloadable Files" to the top of the Linux i.MX RT installation.
2. Install the BSP, as per the respective "Installing and activating cross development environment" document in the "Software" section on the Emcraft site.
3. From the top of the Linux installation, activate the Linux cross-compile environment by running:

```
$ . ACTIVATE.sh
```

4. From the top of the BSP installation, go to the U-Boot tree and install the U-Boot patch matching your BSP version, eg:

```
$ cd u-boot
$ patch -p1 < ../../u-boot-flexcan-2.5.2.patch
```

5. Build the updated U-Boot image as per the respective "Building and self-updating U-Boot" document, eg:

```
$ make mxrt106x-evk defconfig
$ make
```

6. Install the updated U-Boot image to the target board
 - o for the IMXRT1060/1050-EVK boards:
 1. Install the updated U-Boot image to microSD card:

```
$ sudo dd if=u-boot-dtb.imx of=/dev/sd<X> bs=1k seek=1
239+0 records in
239+0 records out
244736 bytes (245 kB) copied, 0.129487 s, 1.9 MB/s
```

2. Insert microSD card into the board and check that the board boots with the updated U-Boot image:

```
U-Boot 2017.09 (Feb 05 2019 - 10:10:48 +0300)
CPU: i.MX RT106x at 600MHz
Model: NXP i.RT1060 EVK
DRAM: 32 MiB
MMC: FSL_SDHC: 0
reading uboot.env
...
```

- o for the IMXRT1060/1050-EVK boards:
 1. Install the updated U-Boot image to the QSPI Flash:

```
=> sf probe 0
SF: Detected is25wp128a with page size 256 Bytes, erase size 64 KiB, total 16 MiB
=> run sf uboot update
Using ethernet@40424000 device
TFTP from server 192.168.1.79; our IP address is 192.168.1.15
Filename 'imxrt117x/u-boot.flexspi'.
Load address: 0x80007fc0
Loading: #####
          314.5 KiB/s

done
Bytes transferred = 258048 (3f000 hex)
SF: 327680 bytes @ 0x0 Erased: OK
device 0 offset 0x0, size 0x3f000
SF: 258048 bytes @ 0x0 Written: OK
=> reset
resetting ...
U-Boot 2017.09-rc1 (May 12 2023 - 17:30:25 +0300)
CPU: i.MX RT117x at 996MHz
Model: NXP i.MXRT1174 EVK
DRAM: 64 MiB
MMC: FSL_SDHC: 0
SF: Detected is25wp128a with page size 256 Bytes, erase size 64 KiB, total 16 MiB
In: serial@4007c000
Out: serial@4007c000
Err: serial@4007c000
Net: eth0: ethernet@40424000
Hit any key to stop autoboot: 0
```

- From the top of the BSP installation, go to the Linux kernel tree and install the kernel patch, eg:

```
$ cd linux/
$ patch -p1 < ../../linux-flexcan-2.5.2.patch
```

- From the top of the Linux installation, go to the `projects` sub-directory, and patch the `rootfs` project:

```
$ cd projects/
$ patch -p1 < ../../projects-rootfs-2.5.2.patch
```

- Additionally, unpack tarball with the binary add-on (this step is to be omitted for the 2.5.4 release):

```
$ pwd
/home/linux-cortexm-2.5.2/projects
$ tar xvj ../../projects-rootfs-2.5.3-addon.tar.bz2
```

- On the Linux PC intended for execution of the CANsocket test suite, ensure that the following software is installed (Emcraft used Linux PC running the Fedora 16 (3.1.0-7.fc16.i686.PAE) installation; the other Linux distributives should work too, but may require some additional steps like compilation and installation of the CAN framework kernel modules):

- Install `can-utils` package on the Linux PC (commands below are for a Fedora host):

```
$ sudo yum install can-utils
...
$
```

- Install and build `can-tests` on the Linux PC:

```
$ cd ~
$ git clone https://github.com/linux-can/can-tests.git
$ cd can-tests
$ make
$ sudo DESTDIR=/usr PREFIX= make install
```

- Load the CAN kernel modules on the Linux PC:

```
$ sudo modprobe can
$ sudo modprobe can-raw
$ sudo modprobe slcan
```

- Connect the VSCOM USB-CAN adapter to the Linux PC and configure it as follows:

- Get the VSCOM USB-CAN serial device name (in the example below it is `ttyUSB0`):

```
$ dmesg | tail
[ 2550.943538] usb 2-2.1: new full speed USB device number 4 using uhci_hcd
[ 2551.060476] usb 2-2.1: New USB device found, idVendor=0403, idProduct=6001
[ 2551.060493] usb 2-2.1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 2551.060495] usb 2-2.1: Product: USB-CAN
[ 2551.060505] usb 2-2.1: Manufacturer: VSCOM
[ 2551.060506] usb 2-2.1: SerialNumber: A4Z5RIHU
[ 2551.113648] USB Serial support registered for FTDI USB Serial Device
[ 2551.113891] ftdi_sio 2-2.1:1.0: FTDI USB Serial Device converter detected
[ 2551.113928] usb 2-2.1: Detected FT232RL
[ 2551.113929] usb 2-2.1: Number of endpoints 2
[ 2551.113930] usb 2-2.1: Endpoint 1 MaxPacketSize 64
[ 2551.113931] usb 2-2.1: Endpoint 2 MaxPacketSize 64
[ 2551.113932] usb 2-2.1: Setting MaxPacketSize 64
[ 2551.117949] usb 2-2.1: FTDI USB Serial Device converter now attached to ttyUSB0
[ 2551.118197] usbcore: registered new interface driver ftdi_sio
[ 2551.118199] ftdi_sio: v1.6.0:USB FTDI Serial Converters Driver
```

- Configure the VSCOM USB-CAN adapter to run with a 1Mbps CAN-bus speed (the `-s8` parameter in `slcan_attach`), enable the corresponding network interface:


```
$ sudo slcan attach -o -s8 /dev/ttyUSB0
attached tty /dev/ttyUSB0 to netdevice slcan0
$ sudo slcand -o -s8 -t hw -S 3000000 /dev/ttyUSB0
$ sudo ifconfig slcan0 up
```

3. If you have disconnected the VSCOM USB-CAN adapter from the Linux PC, before reconnecting it back run the following command:

```
$ sudo killall slcand
```

4.4. Detailed Test Plan

4.4.1. Test Plan: Demo Project

Perform the following step-wise test procedure:

1. Go to the `projects/rootfs` directory, build the loadable Linux image (`rootfs.uImage`) and copy it to the TFTP directory on the host:

```
$ cd projects/rootfs
$ make
```

2. Boot the loadable Linux image (`rootfs.uImage`) to the target via TFTP and validate that it boots to the Linux shell:

```
...
reading splash-imxrt-series_24.bmp
391736 bytes read in 115 ms (3.2 MiB/s)
reading mxrtl06x-evk.ini
101 bytes read in 12 ms (7.8 KiB/s)
fatexec: 'setenv ipaddr 172.17.44.111'
fatexec: 'setenv netmask 255.255.0.0'
Hit any key to stop autoboot: 0
=>
=> setenv tftpd_dir psl/imxrt6/
=> setenv image rootfs.uImage
=> setenv ipaddr 172.17.22.141
=> setenv serverip 172.17.0.1
=> save
=> run netboot
Using ethernet@402D8000 device
TFTP from server 172.17.0.1; our IP address is 172.17.22.141
Filename 'psl/imxrt6/rootfs.uImage'.
Load address: 0x80007fc0
Loading: #####
          #####
...
/ #
```

4.4.2. Test Plan: Linux CAN Driver

Perform the following step-wise test procedure:

1. In the kernel bootstrap messages, validate that the CAN driver has been successfully installed and activated:

```
...
CAN device driver interface
flexcan 401d4000.flexcan: device registered (reg base=401d4000, irq=53)
...
can: controller area network core (rev 20120528 abi 9)
NET: Registered protocol family 29
can: raw protocol (rev 20120528)
can: broadcast manager protocol (rev 20120528 t)
```

```
can: netlink gateway (rev 20130117) max hops=1
...
```

4.4.3. Test Plan: CANSocket

Perform the following step-wise test procedure:

1. On the target, configure the CAN network:

```
/ # ip link set can0 type can bitrate 1000000
/ # ifconfig can0 up
```

2. Test target to Linux PC transfers:
 - o Run the capture utility on the Linux PC:

```
$ candump slcan0
```

- o Send packets from the target to the host:

```
~ # cansend can0 12345678#99.AA.BB.CC.DD.EE.FF.00
~ # cansend can0 12345678#99.AA.BB.CC.DD.EE.FF.01
~ # cansend can0 12345678#99.AA.BB.CC.DD.EE.FF.02
~ # cansend can0 12345678#99.AA.BB.CC.DD.EE.FF.03
```

- o Validate that the packets have been captured on the Linux PC:

```
slcan0 12345678 [8] 99 AA BB CC DD EE FF 00
slcan0 12345678 [8] 99 AA BB CC DD EE FF 01
slcan0 12345678 [8] 99 AA BB CC DD EE FF 02
slcan0 12345678 [8] 99 AA BB CC DD EE FF 03
```

- o On the host, stop the capture utility by pressing `Ctrl-C`:

```
^C
$
```

3. Test Linux PC to target transfers:
 - o Run the capture utility on the target:

```
~ # candump can0
```

- o Send packets from the Linux PC to the target:

```
$ cansend slcan0 123abcde#11.22.33.44.56.78.90.01
$ cansend slcan0 123abcde#11.22.33.44.56.78.90.03
$ cansend slcan0 123abcde#11.22.33.44.56.78.90.05
$ cansend slcan0 123abcde#11.22.33.44.56.78.90.07
```

- o Validate that the packets have been captured on the target:

```
can0 123ABCDE [8] 11 22 33 44 56 78 90 01
can0 123ABCDE [8] 11 22 33 44 56 78 90 03
can0 123ABCDE [8] 11 22 33 44 56 78 90 05
can0 123ABCDE [8] 11 22 33 44 56 78 90 07
```

- o On the target, stop the capture utility by pressing `Ctrl-C`:

```
^C
~ #
```

4.4.4. Test Plan: CANSocket Test Suite

Perform the following step-wise test procedure:

1. Run the `tst-raw` Linux PC to target test:

- On the target:

```
~ # tst-raw -i can0
```

- On the Linux PC:

```
$ sudo tst-raw-sendto -i slcan0
```

- Observe the test data on the target, then press `Ctrl-C` and complete the test:

```
123 [3] 11 22 33  
^C  
~ #
```

2. Run the `tst-raw` target to Linux PC test:

- On the Linux PC:

```
$ sudo tst-raw -i slcan0
```

- On the target:

```
~ # tst-raw-sendto -i can0
```

- Observe test data on the Linux PC, then press `Ctrl-C` and complete the test:

```
123 [3] 11 22 33  
^C  
$
```

3. Run the `tst-packet` Linux PC to target test:

- On the target:

```
~ # tst-packet -i can0
```

- On the Linux PC send a packet, then press `Ctrl-C` and complete the test:

```
$ sudo tst-packet -i slcan0 -s  
^C  
$
```

- Observe the test packet on the target, then press `Ctrl-C` and complete the test:

```
123 [2] 11 22  
^C  
~ #
```

4. Run the `tst-packet` target to Linux PC test:

- On the Linux PC:

```
$ sudo tst-packet -i slcan0
```

- On the target, send a packet, then press `Ctrl-C` and complete the test:

```
~ # tst-packet -i can0 -s  
^C  
~ #
```

- Observe the test packet on the Linux PC, then press `Ctrl-C` and complete the test:

```
123 [2] 11 22
^C
$
```

5. Run the `tst-filter` test on the target:

```
~ # tst-filter can0
---
testcase 0 filters : can_id = 0x00000123 can_mask = 0x000007FF
testcase 0 sending patterns ... ok
testcase 0 rx : can_id = 0x00000123 rx = 1 rxbits = 1
testcase 0 rx : can_id = 0x40000123 rx = 2 rxbits = 17
testcase 0 rx : can_id = 0x80000123 rx = 3 rxbits = 273
testcase 0 rx : can_id = 0xC0000123 rx = 4 rxbits = 4369
testcase 0 ok
---
testcase 1 filters : can_id = 0x80000123 can_mask = 0x000007FF
testcase 1 sending patterns ... ok
testcase 1 rx : can_id = 0x00000123 rx = 1 rxbits = 1
testcase 1 rx : can_id = 0x40000123 rx = 2 rxbits = 17
testcase 1 rx : can_id = 0x80000123 rx = 3 rxbits = 273
testcase 1 rx : can_id = 0xC0000123 rx = 4 rxbits = 4369
testcase 1 ok
---
testcase 2 filters : can_id = 0x40000123 can_mask = 0x000007FF
testcase 2 sending patterns ... ok
testcase 2 rx : can_id = 0x00000123 rx = 1 rxbits = 1
testcase 2 rx : can_id = 0x40000123 rx = 2 rxbits = 17
testcase 2 rx : can_id = 0x80000123 rx = 3 rxbits = 273
testcase 2 rx : can_id = 0xC0000123 rx = 4 rxbits = 4369
testcase 2 ok
---
<...>
---
testcase 15 filters : can_id = 0xC0000123 can_mask = 0xC00007FF
testcase 15 sending patterns ... ok
testcase 15 rx : can_id = 0xC0000123 rx = 1 rxbits = 4096
testcase 15 ok
---
testcase 16 filters : can_id = 0x00000123 can_mask = 0xFFFFFFFF
testcase 16 sending patterns ... ok
testcase 16 rx : can_id = 0x00000123 rx = 1 rxbits = 1
testcase 16 ok
---
testcase 17 filters : can_id = 0x80000123 can_mask = 0xFFFFFFFF
testcase 17 sending patterns ... ok
testcase 17 rx : can_id = 0x80000123 rx = 1 rxbits = 256
testcase 17 ok
---
~ #
```

6. Run the `tst-rcv-own-msgs` test on the target:

```
~ # tst-rcv-own-msgs can0
Starting PF CAN frame flow test.
checking socket default settings ... ok.
check loopback 0 rcv own msgs 0 ... ok.
check loopback 0 rcv own msgs 1 ... ok.
check loopback 1 rcv_own_msgs 0 ... ok.
check loopback 1 rcv_own_msgs 1 ... ok.
PF CAN frame flow test was successful.
~ #
```