PCAN Driver Installation and Initialization on Jetson Nano

Table of contents

1. Introduction

The Jetson Nano, a small single-board computer running Ubuntu Linux, enables developers to implement advanced applications. Integrating a CAN interface on Jetson Nano allows it to communicate with automotive controllers, industrial sensors, or other CAN-enabled devices, enabling monitoring, control, and data acquisition over the CAN bus.

This document describes the **complete procedure to install, configure, and initialize PCAN drivers** on Jetson Nano. It also includes instructions for sending and receiving CAN messages using Python, providing a foundation for building CAN-enabled applications.

2. Objective

The main objectives of this setup are:

1. **Driver Installation:**
   - Download, compile, and install PEAK PCAN Linux drivers on Jetson Nano.
   - Enable SocketCAN support to interface with standard Linux CAN tools.
2. **System Preparation:**
   - Install all required build tools, kernel headers, and Python libraries.
   - Verify kernel, GCC, and JetPack versions to ensure driver compatibility.
3. **PCAN Initialization:**
   - Load and verify the PCAN driver in the kernel.
   - Bring up the CAN interface (can0) at a specified bitrate.
4. **Testing and Communication:**

   - Test the CAN bus using can-utils (listen/send messages).
   - Initialize PCAN-USB in Python and send CAN frames programmatically.

3. PCAN Drivers Installation

- Install build tools + kernel headers (or get L4T headers)
- Download peak-linux-driver from PEAK and build it
- (Optional) Install with DKMS so it rebuilds on kernel updates
- Load module, verify /proc/pcan, find pcan.ko location
- Rebuild with SocketCAN (NET=NETDEV_SUPPORT)
- Bring up can0 and test with can-utils
- Install PCAN-Basic (C/Python) and sample usage
- Make pcan auto-load and optionally auto bring up can0 at boot

3.1 check your system

```
uname -r           # kernel version (needed for headers)
gcc --version       # GCC used to compile modules
lsb_release -a      # Ubuntu version / JetPack info
```

Step 1 - Install required packages (build tools + utilities)

```
-sudo apt update
-sudo apt install -y build-essential dkms can-utils python3 python3-pip git libusb-1.0-0-dev
```

3.2 Install kernel headers

Driver compilation requires headers matching your running kernel.

```
-  sudo apt install -y linux-headers-$(uname -r)
```

Explanation:

- uname -r shows your current kernel version (e.g., 4.9.253-tegra).
- Kernel headers allow the driver to compile against your exact kernel.

3.3 Download PEAK Linux driver

```
-cd ~/Downloads
-wget
https://www.peak-system.com/fileadmin/media/linux/files/peak-linux-driver-8.15.2.tar.gz
-tar -xzf peak-linux-driver-*.tar.gz
-cd peak-linux-driver-*
```

3.4 Build and install driver

```
-make clean
-make
-sudo make install
```

Explanation:

- make clean: cleans previous builds.
- make: compiles pcan.ko kernel module.
- sudo make install: installs the module into /lib/modules/$(uname -r)/kernel/drivers/net/can/.

Expected output:

- Compilation messages, ending with pcan.ko installed.
- Example location: /lib/modules/4.9.253-tegra/kernel/drivers/net/can/pcan.ko.

3.5 Load the driver

```
-sudo modprobe pcan
```

Explanation:

- Loads the driver into the kernel.
- PCAN devices will now be recognized.

3.6 Verify driver and device

```
-lsmod | grep pcan

-cat /proc/pcan

-ip link show | grep can
```

3.7 Enable SocketCAN

If you want standard Linux can0 interface:

1. Rebuild driver with SocketCAN support:

   ```
   -sudo rmmod pcan

   -make clean

   -make NET=NETDEV_SUPPORT

   -sudo make install

   -sudo modprobe pcan
   ```

Expected output:

- No errors.
- After modprobe, ip link show will list can0.

3.8 Bring CAN interface up

```
-sudo ip link set can0 up type can bitrate 500000

-ip -details link show can0
```

Expected output:

```
4: can0: <NOARP,ECHO> mtu 16 qdisc pfifo_fast state UP mode DEFAULT
group default qlen 10

    link/can  promiscuity 0

    can state UP restart-ms 100

    bitrate 500000 sample-point 0.875

    tq 125 prop-seg 6 phase-seg1 7 phase-seg2 2 sjw 1
```

3.9 Test CAN bus

1) Listen on can0:

```
-candump can0
```

2) Send a frame

```
-cansend can0 123#1122334455667788
```

3.10 Make pcan auto-load at boot

```
-echo pcan | sudo tee /etc/modules-load.d/pcan.conf
```

Then enable:

```
-sudo systemctl daemon-reload

-sudo systemctl enable --now pcan-netup.service
```

```
Sudo systemctl enable –now pcan-netup.service

GPIO.setmode(GPIO.BOARD)
```

- **Pcan Initialization**

Step 1: Install prerequisites

```
-sudo apt update
-sudo apt install -y build-essential dkms can-utils python3 python3-pip
libusb-1.0-0-dev git
-pip3 install python-can
```

Step 2 - code for pacn initialization

```
 import can
 import time

def send_can_frames():
        # Initialize PCAN-USB bus
        bus = can.Bus(interface="pcan", channel="PCAN_USBBUS1",
bitrate=250000)

        frames = [
        # (Arbitration ID, Data bytes, Description)
        (0x307, [0x00, 0x00, 0x67, 0x12, 0x00, 0x00, 0x00, 0x00], "Warning
message to Driver"),
        (0x307, [0x00, 0x00, 0x00, 0x00, 0x00, 0x3E, 0x00, 0x00], "Steering and
meter sign"),
        (0x307, [0x00, 0x00, 0x00, 0x00, 0x67, 0x00, 0x00, 0x00], "Front view
camera is Faulty")
        ]

        print("Starting CAN frame transmission...\n")

        for arb_id, data, desc in frames:
        msg = can.Message(arbitration_id=arb_id,
                        data=data,
                        is_extended_id=False)



        try:
        bus.send(msg)
        print(f"Sent Frame → ID: {hex(arb_id)}  Data: {[hex(b) for b in data]}  |
{desc}")
        except can.CanError:
        print("Error: Message NOT sent. Check CAN connection or driver.")
        time.sleep(1)  # delay between messages



        print("\n All frames sent. Closing CAN bus.")
        bus.shutdown()

if __name__ == "__main__":
        send_can_frames()
```