Guide Of Installation For CAN Utility Under Linux

Environment:

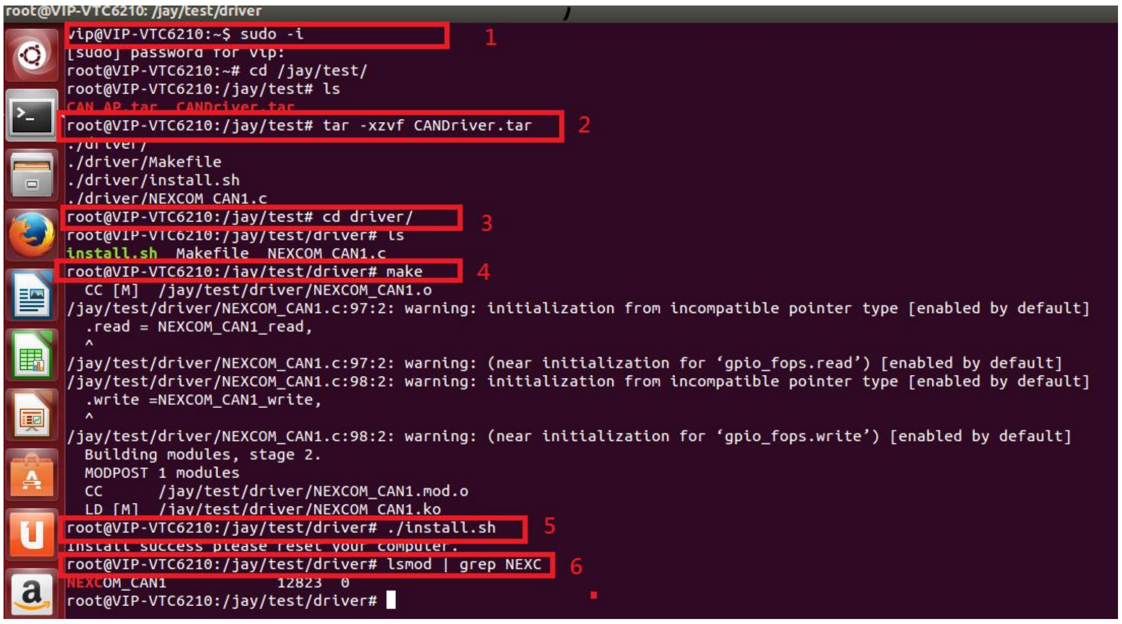
Test Machine	VTC6210 /VTC1010
Operation System	6Ubuntu13.04 (kernel 3.11.0-24-genenric)
Language	English
Browser	Terminal

(1) There are two tar file for using CAN device under Linux. One is "CANDriver.tar" and other is "CAN_AP.tar".
The "CANDriver.tar" is driver for CAN device, so you need to have root authority to install this driver.
The "CAN_AP.tar" has a demo program for how to access CAN message from CAN device.
Next we will explain how to use above two files. The item (2) will explain how to install CAN driver. And item (3) will explain how to write an application program from demo program.

(2) Please copy "CANDriver.tar" to Linux OS, then follow six steps to install driver to OS.

1. Change the user to root
2. Unzip "CANDriver.tar" on Linux OS.
3. Go to "driver" folder.
4. "make" driver source code and generate "NEXCOM_CAN1.ko" file.
5. Execute "install.sh" file then "NEXCOM_CAN1" driver will be installed to Linux OS

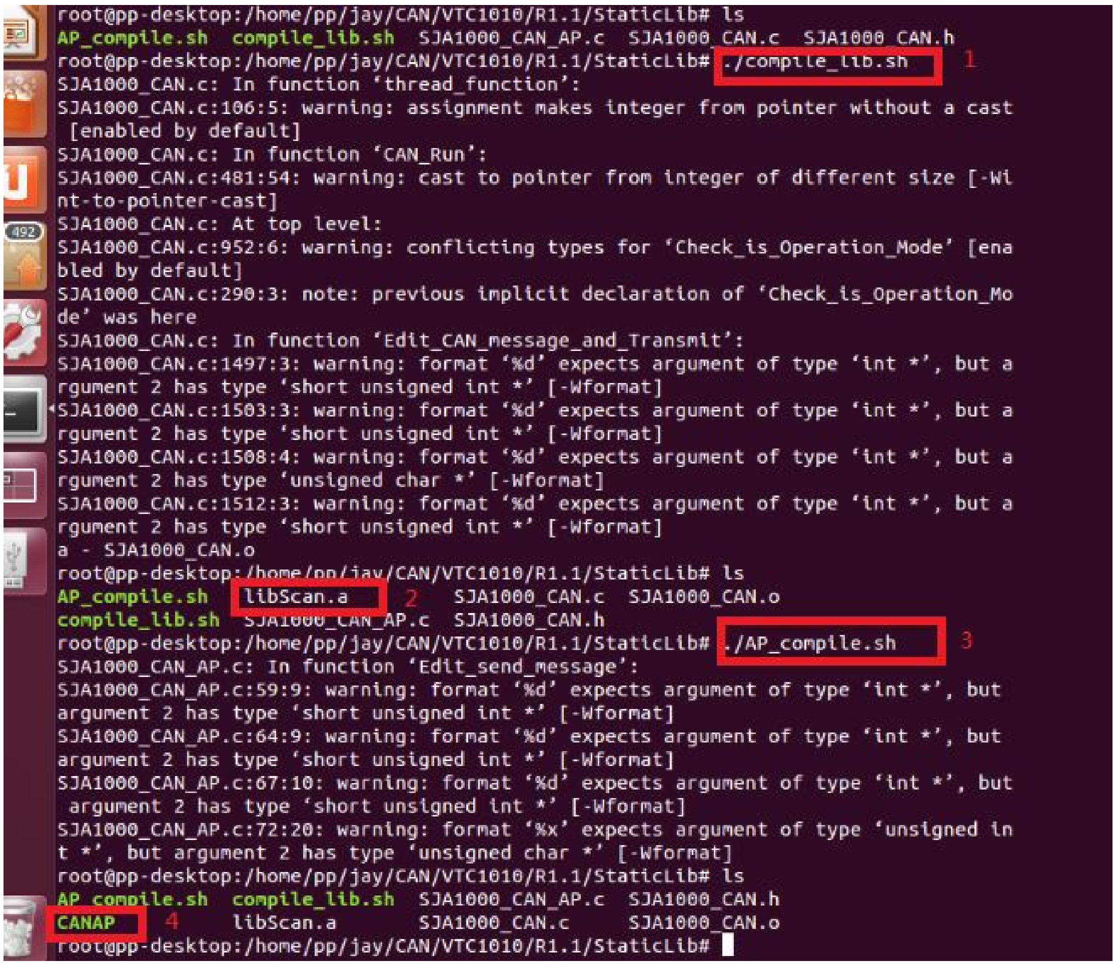
6. Please use "lsmod" to check "NEXCOM_CAN1" whether has been installed to Linux OS.



```
vip@VIP-VTC6210:~$ sudo -i
[sudo] password for vip:
root@VIP-VTC6210:~# cd /jay/test/
root@VIP-VTC6210:/jay/test# ls
CAN_AP.tar  CANDriver.tar
root@VIP-VTC6210:/jay/test# tar -xzvf CANDriver.tar
./driver/
./driver/Makefile
./driver/install.sh
./driver/NEXCOM_CAN1.c
root@VIP-VTC6210:/jay/test# cd driver/
root@VIP-VTC6210:/jay/test/driver# ls
install.sh  Makefile  NEXCOM_CAN1.c
root@VIP-VTC6210:/jay/test/driver# make
  CC [M]  /jay/test/driver/NEXCOM_CAN1.o
/jay/test/driver/NEXCOM_CAN1.c:97:2: warning: initialization from incompatible pointer type [enabled by default]
  .read = NEXCOM_CAN1_read,
  ^
/jay/test/driver/NEXCOM_CAN1.c:97:2: warning: (near initialization for ‘gpio_fops.read’) [enabled by default]
/jay/test/driver/NEXCOM_CAN1.c:98:2: warning: initialization from incompatible pointer type [enabled by default]
  .write =NEXCOM_CAN1_write,
  ^
/jay/test/driver/NEXCOM_CAN1.c:98:2: warning: (near initialization for ‘gpio_fops.write’) [enabled by default]
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /jay/test/driver/NEXCOM_CAN1.mod.o
  LD [M]  /jay/test/driver/NEXCOM_CAN1.ko
root@VIP-VTC6210:/jay/test/driver# ./install.sh
install success please reset your computer.
root@VIP-VTC6210:/jay/test/driver# lsmod | grep NEXC
NEXCOM_CAN1            12823  0
root@VIP-VTC6210:/jay/test/driver#
```

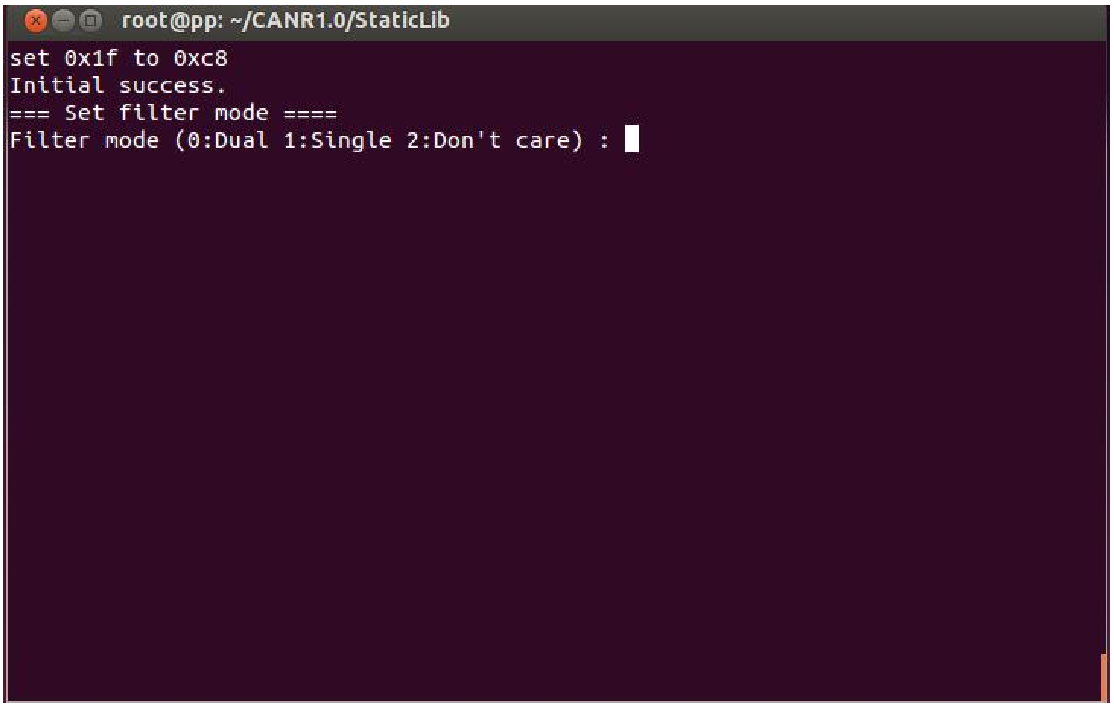
(3) Please copy “CAN_AP.tar” to Linux OS, then follow four steps to compile demo program.

1. Unzip “CAN_AP.tar” on Linux OS.
2. Go to “StaticLib” folder.
3. Execute “./compile_lib.sh” file and then generate “libScan.a” library.
4. Execute “AP_compile.sh.sh” file and then generate “CANAP” executable file on this folder.
5. Please execute “CANAP” on Linux OS, then you can access CAN message.
6. You can refer to explanation of library function on appendix A if you want to develop application program of CAN bus function.



```
root@pp-desktop:/home/pp/jay/CAN/VTC1010/R1.1/StaticLib# ls
AP_compile.sh  compile_lib.sh  SJA1000_CAN_AP.c  SJA1000_CAN.c  SJA1000_CAN.h
root@pp-desktop:/home/pp/jay/CAN/VTC1010/R1.1/StaticLib# ./compile_lib.sh     1
SJA1000_CAN.c: In function ‘thread_function’:
SJA1000_CAN.c:106:5: warning: assignment makes integer from pointer without a cast
 [enabled by default]
SJA1000_CAN.c: In function ‘CAN_Run’:
SJA1000_CAN.c:481:54: warning: cast to pointer from integer of different size [-Wi
nt-to-pointer-cast]
SJA1000_CAN.c: At top level:
SJA1000_CAN.c:952:6: warning: conflicting types for ‘Check_is_Operation_Mode’ [ena
bled by default]
SJA1000_CAN.c:290:3: note: previous implicit declaration of ‘Check_is_Operation_Mo
de’ was here
SJA1000_CAN.c: In function ‘Edit_CAN_message_and_Transmit’:
SJA1000_CAN.c:1497:3: warning: format ‘%d’ expects argument of type ‘int *’, but a
rgument 2 has type ‘short unsigned int *’ [-Wformat]
SJA1000_CAN.c:1503:3: warning: format ‘%d’ expects argument of type ‘int *’, but a
rgument 2 has type ‘short unsigned int *’ [-Wformat]
SJA1000_CAN.c:1508:4: warning: format ‘%d’ expects argument of type ‘int *’, but a
rgument 2 has type ‘unsigned char *’ [-Wformat]
SJA1000_CAN.c:1512:3: warning: format ‘%d’ expects argument of type ‘int *’, but a
rgument 2 has type ‘short unsigned int *’ [-Wformat]
a - SJA1000_CAN.o
root@pp-desktop:/home/pp/jay/CAN/VTC1010/R1.1/StaticLib# ls
AP_compile.sh   libScan.a   2   SJA1000_CAN.c   SJA1000_CAN.o
compile_lib.sh  SJA1000_CAN_AP.c  SJA1000_CAN.h
root@pp-desktop:/home/pp/jay/CAN/VTC1010/R1.1/StaticLib# ./AP_compile.sh     3
SJA1000_CAN_AP.c: In function ‘Edit_send_message’:
SJA1000_CAN_AP.c:59:9: warning: format ‘%d’ expects argument of type ‘int *’, but
argument 2 has type ‘short unsigned int *’ [-Wformat]
SJA1000_CAN_AP.c:64:9: warning: format ‘%d’ expects argument of type ‘int *’, but
argument 2 has type ‘short unsigned int *’ [-Wformat]
SJA1000_CAN_AP.c:67:10: warning: format ‘%d’ expects argument of type ‘int *’, but
 argument 2 has type ‘short unsigned int *’ [-Wformat]
SJA1000_CAN_AP.c:72:20: warning: format ‘%x’ expects argument of type ‘unsigned in
t *’, but argument 2 has type ‘unsigned char *’ [-Wformat]
root@pp-desktop:/home/pp/jay/CAN/VTC1010/R1.1/StaticLib# ls
AP_compile.sh  compile_lib.sh  SJA1000_CAN_AP.c  SJA1000_CAN.h
CANAP    4     libScan.a       SJA1000_CAN.c     SJA1000_CAN.o
root@pp-desktop:/home/pp/jay/CAN/VTC1010/R1.1/StaticLib#
```

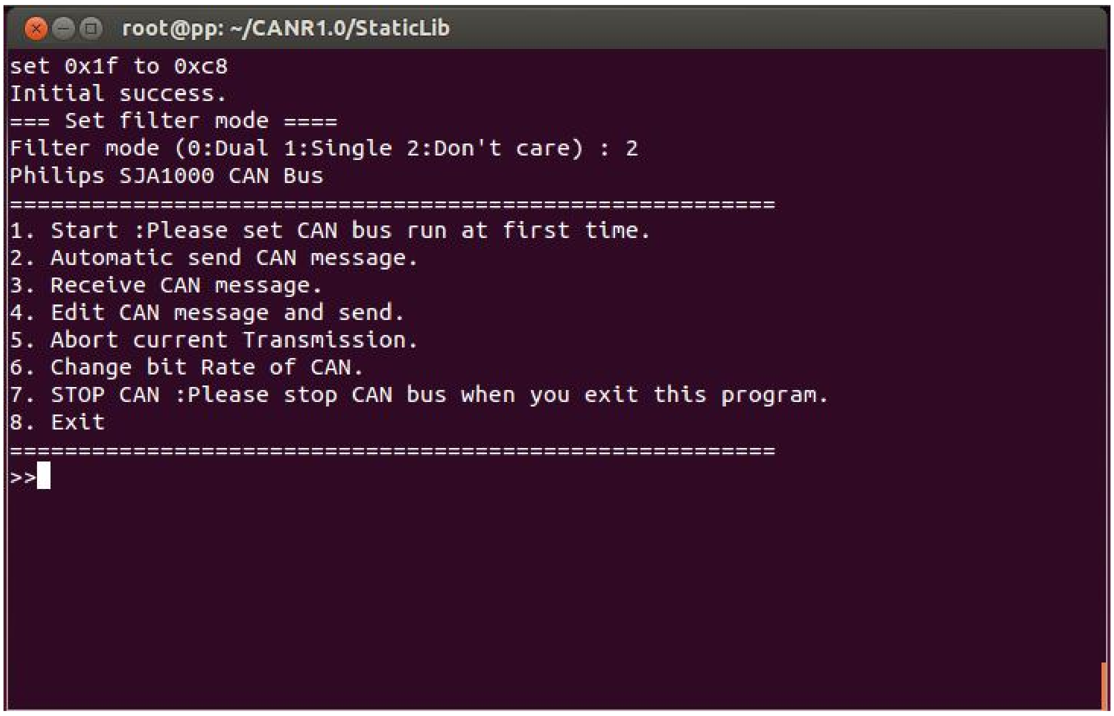
7. You can see the screen as below after you execute "./CANAP".



```
set 0x1f to 0xc8
Initial success.
=== Set filter mode ====
Filter mode (0:Dual 1:Single 2:Don't care) :
```

You can select whether you need to set CAN message filter. You can select "2" if you want to receive all messages from other CAN device. Or you can select "1" to set "Single" filter and "2" for "Dual" filter. Please refer to page44 of SJA1000 spec if you need to know more detail.

8. You can see the screen as below after you set or ignore filter setting.



```
set 0x1f to 0xc8
Initial success.
=== Set filter mode ====
Filter mode (0:Dual 1:Single 2:Don't care) : 2
Philips SJA1000 CAN Bus
========================================================
1. Start :Please set CAN bus run at first time.
2. Automatic send CAN message.
3. Receive CAN message.
4. Edit CAN message and send.
5. Abort current Transmission.
6. Change bit Rate of CAN.
7. STOP CAN :Please stop CAN bus when you exit this program.
8. Exit
========================================================
>>
```

There are eight items for your selection.

(1) Start : this is an initial process, please set this at first time when you use this AP.

(2) Automatic send CAN message:

    You can use this item to send message for testing. (PS: the message is standard message 11bits)

(3) Receive CAN message:

    You can use this item to receive CAN message.

(4) Enter CAN message and send:

    You can use this item to set which message you want to send.

(5) Abort current transmission:

    You can use this item to abort current sending message.

(6) Change bit Rate of CAN:

    You can use this item to change bit rate of CAN.

    PS: you need to set right bit rate firstly when you use CAN function.

(7) STOP CAN:

    Because you need to close some driver before you exit this AP.

    So you need using this item when you want to exit this AP before.

    The CAN function will be not workable again if you don’t do “STOP CAN” when you exit this AP.

(8) Exit:

    You can use this item to exit this AP

Appendix A : Explanation of library function

There are 13 functions can be used to access CAN message. Next will explain each function how to using.

(1) void CAN_Reset(int fh)

This function can reset CAN device. You need to send "CAN driver id" to "fh" argument, then CAN device will be reset to inactive state.

(2) int CAN_SetBaud(int fh, int baudrate)

You can use this function to set baudrate of CAN device.
You need send "CAN driver id" to "fh" argument, and baudrate to second argument. Then CAN device will be programed baudrate to your setting.
PS: There are total 20 baudrate you can set on "SJA1000_CAN.h". Please let us know if you need to use other baudrate except those.

(3) void CAN_Dual_Filter (int fh, can_rx_dual_filer_t *filter)

You can use this function to set filter of CAN device.
You need send "CAN driver id" to "fh" argument, and "can_rx_daul_filter structure" to second argument.

(4) void CAN_Single_Standard_Filter(int fh,can_rx_single_standard_filter_t *filter)

You can use this function to set filter of CAN device.
You need send "CAN driver id" to "fh" argument, and "can_rx_single_standard_filter structure" to second argument.

(5) void CAN_Single_Extended_Filter(int fh,can_rx_single_extended_filter_t *filter)

You can use this function to set filter of CAN device.
You need send "CAN driver id" to "fh" argument, and "can_rx_single_standard_filter structure" to second argument.
PS: You can refer to filter setting of CAN device on page44~48 of "SJA1000" spec if you don't know how to set filter.

(6) void Abort_Transmission(int fh)

You can use this function to abort a sending message of CAN device. For example, you can abort a sending message if no any device to reply "acknowledge phase" for this message.
You need send "CAN driver id" to "fh" argument.

(7) int CAN_Transmission(can_msg_t *msg)

You can use this function to send a message to other CAN device.
You need send "can_msg_t structure" to "msg" argument.
The return value : 1. The buffer is full if return value is "-1"
2. The value indicate how many transmitted messages on

buffer if the value is a positive number.

(8) int CAN_Receive_nonblock(can_msg_t *msg)

You can use this function to receive a message from CAN device.

You need get a CAN message from CAN device on “msg” argument.

The return value : 1. The buffer is empty if return value is “-1”

2. The value indicate how many received messages on buffer if the value is a positive number.

(9) int CAN_Error_States_Get(int fh,can_error_t *msg)

You can use this function to get error messages from CAN device.

This function is reserved for future debug.

(10) void Automatic_Transmit_CAN_message()

This function is for testing only. You can use this function to automatically send message form CAN device.

(11) int CAN_Initial(int fh, int baudrate)

You must use this function to initial CAN device before you want to use it.

The return value : 1. Initiation is fail if return value is “-1”.

2. Initiation is success if the return value is “1”.

(12) int CAN_Run(int fh)

Set device to run. You must use this function to start ability of CAN device before you want to access CAN message (transmit or receive).

The return value : 1. It is fail if return value is “-1”.

2. It is success if the return value is “1”.

(12) int CAN_STOP()

Set device to stop. You must use this function to terminate ability of CAN device before you want to stop CAN device or exit your AP.

The return value : 1. It is fail if return value is “-1”.

2. It is success if the return value is “1”.

(13) void Clear_Buffer(int fh)

You can use this function to clear “transmit” and “receive” buffer.