VANIER COLLEGE - Computer Engineering Technology
247-609 Networked Embedded Systems

Lab 4 : CAN on BBB

Leonardo Fusser (1946695)

Purpose:

a) To enable cape overlays: CAN bus.
b) To familiarize with the basic communications of CAN bus.
c) To observe CAN bus transactions using oscilloscope and protocol analyzer.

To be submitted before the deadline, via MS Teams Assignment:

1. **No formal report required**. Report must be in MS Word format, and includes the following:
    a. Answer questions, screen shots (clearly labelled).
    b. Include a final discussion and conclusion session.

All the following instructions are based on Debian Linux 8.6 (Jessie)

Lab Work:

Part A: Cape Overlays

1. Update the system

```
# apt-get update
```

2. Verify which capes are already installed

```
root@beaglebone:~# cat /sys/devices/platform/bone_capemgr/slots
 0: PF----  -1
 1: PF----  -1
 2: PF----  -1
 3: PF----  -1
 4: P-O-L- 0 Override Board Name,00A0,Override Manuf,BB-CAN1
```

3. If `BB-UART4,BB-CAN1,BB-ADC` are NOT installed then the cape overlays have to be loaded at boot time. To do so the following line must be added to `/etc/default/capemgr` file

```
CAPE=BB-UART4,BB-CAN1,BB-ADC
```

Note: overlays are separated by a comas.

4. After rebooting you will see that there are new virtual capes in the next free slots:

```
root@beaglebone:~# cat /sys/devices/platform/bone_capemgr/slots
 0: PF----  -1
 1: PF----  -1
 2: PF----  -1
 3: PF----  -1
 4: P-O-L-   0 Override Board Name,00A0,Override Manuf,BB-UART4
 5: P-O-L-   1 Override Board Name,00A0,Override Manuf,BB-CAN1
 6: P-O-L-   2 Override Board Name,00A0,Override Manuf,BB-ADC
```

5. If the BB-CAN1 overlay from part1 was not installed properly go the directory `/lib/firmware/` and look for the `DeviceTreeOverlay` for the DCAN1 "`BB-CAN1-00A0.dtbo`".

   If not present, then:

```
root@beaglebone:~# sudo apt-get install build-essential git-core
device-tree-compiler
root@beaglebone:~# git clone https://github.com/beagleboard/bb.org-
overlays/
root@beaglebone:~# cd ./bb.org-overlays/
root@beaglebone:~# ./install.sh
```

6. After a reboot you can check if the overlay is loaded as in step 4.

Part B: CAN bus

7. You can now load the kernel drivers for the CAN-bus:

```
root@beaglebone:~# sudo modprobe can
root@beaglebone:~# sudo modprobe can-dev
root@beaglebone:~# sudo modprobe can-raw
```

8. To bring up the interface and configure the bus speed to 500kBit/sec enter the following lines:

```
root@beaglebone:~/ sudo ip link set can0 up type can bitrate 500000
root@beaglebone:~/ sudo ifconfig can0 up
```

9. To check that the can interface is up you can use `ifconfig` and should get this output:

```
root@beaglebone:~# ifconfig
can0 Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00
      UP RUNNING NOARP MTU:16 Metric:1
      RX packets:0 errors:0 dropped:0 overruns:0 frame:0
      TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
      RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
      Interrupt:188
```

10. To start the CAN interface at system startup the `/etc/network/interfaces` configuration file can be used by adding the following lines:

```
allow-hotplug can0
iface can0 can static
bitrate 500000
```

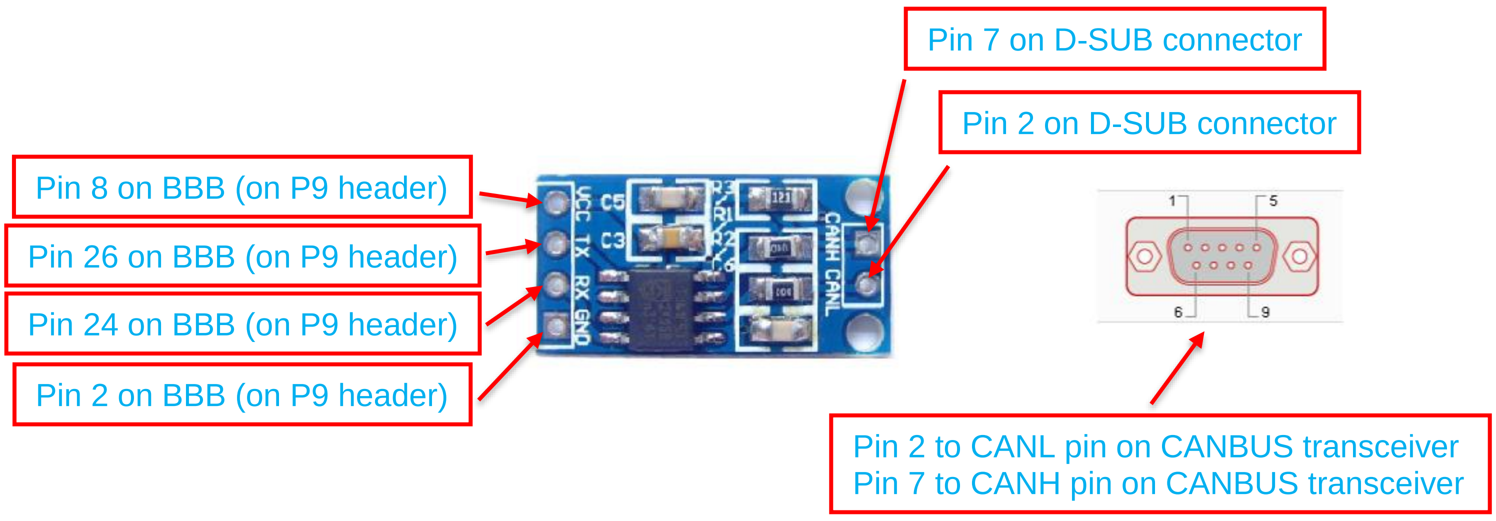
11. Which are the 2 pins on BBB that are configured as RX and TX for the CAN bus?

   - The 2 pins on the BBB that are configured as RX and TX for the CAN bus are pins 24 on P9 header (DCAN1_RX) and 26 on P9 header (DCAN1_TX). These two pins are also multiplexed with UART1_RXD (pin 26 on P9 header) and UART1_TXD (pin 24 on P9 header).
     This can be verified by looking inside the bone.js file (src/bone.js).

Part C: CAN bus transceiver interface

12. Obtain an ARD-617A high speed CANBUS transceiver board. You are required to design the connection between BBB and the transceiver board in order to generate the corresponding CANL and CANH signals. You will be using PEAK PCAN-USB (https://www.peak-system.com/PCAN-USB.199.0.html?&L=1) to send and receive CAN frames via a Windows software.

   a) Complete the figure below to illustrate your connections. Clearly indicate the pin number on BBB that should be connected.


Pin 7 on D-SUB connector
Pin 2 on D-SUB connector
Pin 8 on BBB (on P9 header)
Pin 26 on BBB (on P9 header)
Pin 24 on BBB (on P9 header)
Pin 2 on BBB (on P9 header)
VCC
TX
RX
GND
C5
C3
CANH
CANL
1
5
6
9
Pin 2 to CANL pin on CANBUS transceiver
Pin 7 to CANH pin on CANBUS transceiver

   b) Get approval from instructor on your design of connections.

13. To perform a basic test, CANSend and CANDump from the can-utils collection can be used. The building and installation of the can-utils is done by the following steps:

```
root@beaglebone:~# git clone https://github.com/linux-can/can-
utils.git
root@beaglebone:~# cd can-utils/
root@beaglebone:~# ./autogen.sh
root@beaglebone:~# ./configure
root@beaglebone:~# make
```

14. Connect up all the components as in step 13(a). Open PCAN-View application on your PC and select the corresponding hardware interface.

15. To perform test in sending CAN frames:

a) Connect up your oscilloscope to TX and CANH or CANL. Perform all the necessary setup on the protocol analyzer to analyze CAN bus. Select “Serial” button. On the screen menu:

   a. Mode: choose CAN protocol.

   b. Signals: setup your source, type of signal and baud rate accordingly.

b) Send a CAN-message to the address 0x247 with the data contains the 7 digits of your student numbers.

   Example: `root@beaglebone:~#./cansend can0 247#00.01.02.03.04.05.06`

c) You should be able to see the packet being recorded on PCAN-View, so as the oscilloscope. Perform the necessary screen shots with record of observation/analysis.

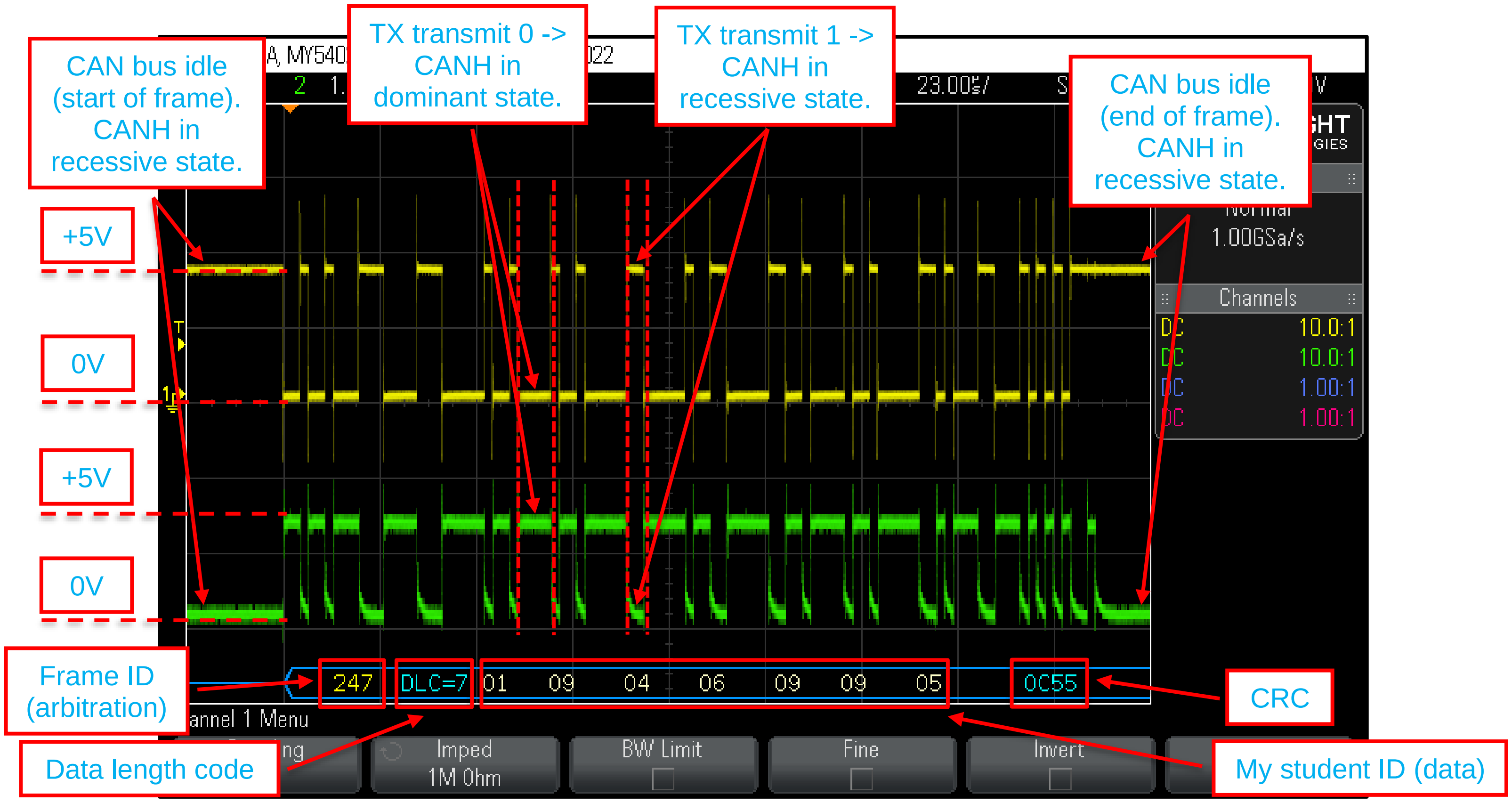


Figure 1. Yellow signal (channel 1): TX signal from DCAN1_TX pin on BBB. Green signal (channel 2): CANH signal from CANH pin on CAN transceiver board. Screenshot above shows complete CAN data frame being transmitted from the BBB. This is a standard CAN data frame.

16. To perform test in receiving CAN frames:

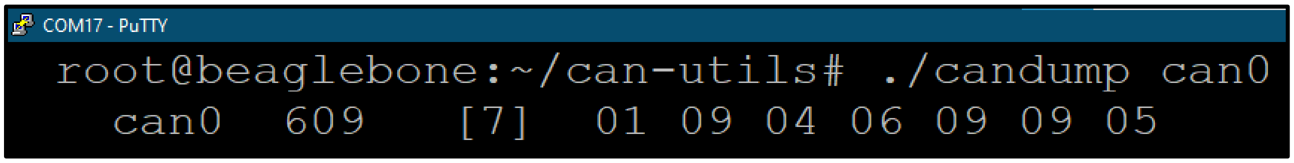
a) Connect up your oscilloscope to RX and CANH or CANL.

b) To log received message on BBB, you can use the CAN Dump program.

```
root@beaglebone:~#./candump can0
```

c) Send a CAN-message to the address 0x609 with the data contains the 7 digits of your student numbers.

d) You should be able to see the packet being recorded on candump, so as the oscilloscope. Perform the necessary screen shot with record of observation/analysis.



```
COM17 - PuTTY
root@beaglebone:~/can-utils# ./candump can0
  can0  609   [7]  01 09 04 06 09 09 05
```

Figure 2. CANdump observation from CAN0 (DCAN1_RX) pin on BBB. Received CAN data frame transmitted from PEAK CAN USB adapter connected to CAN transceiver board. Received CAN data frame contains my student ID (1946995).

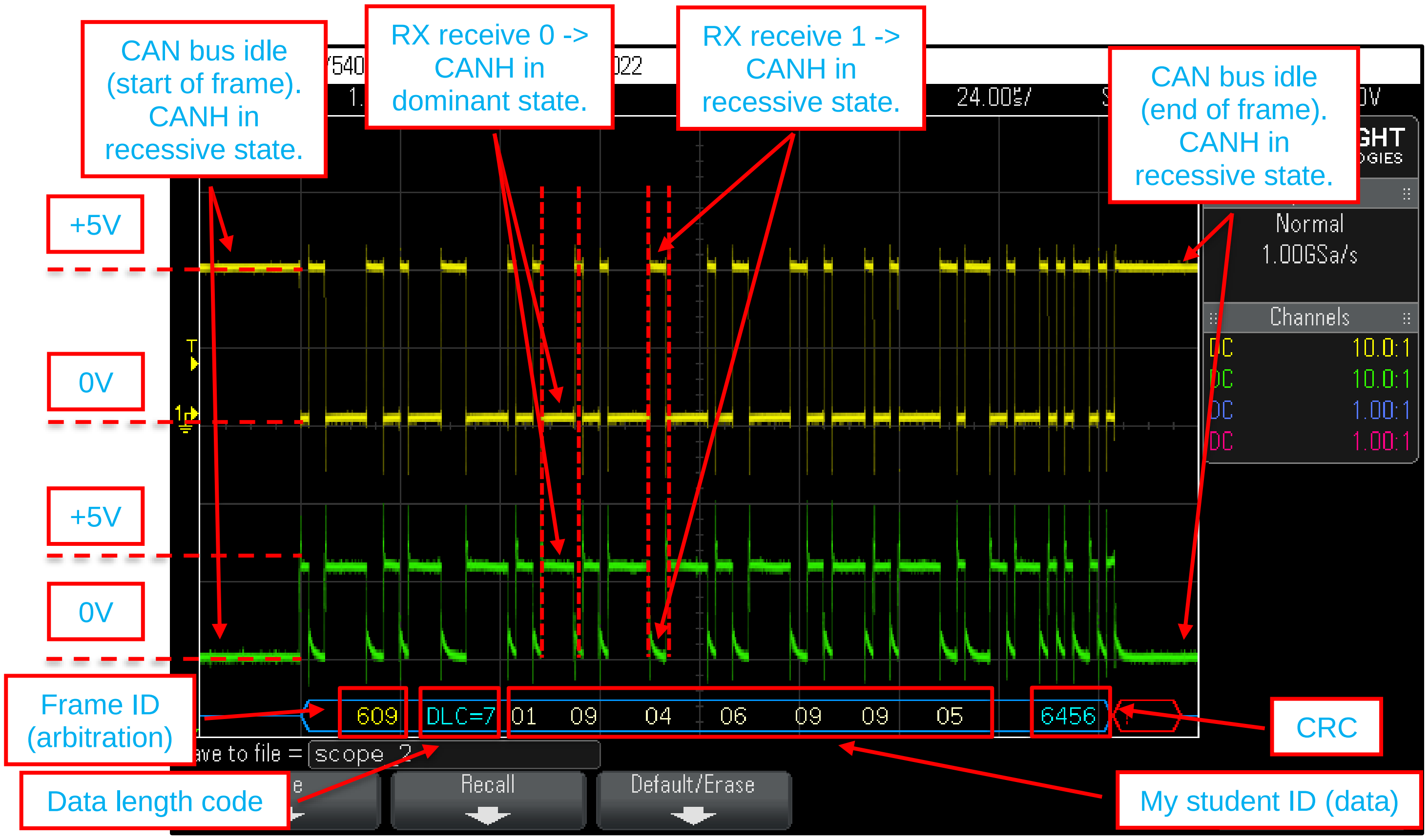


Figure 3. Yellow signal (channel 1): RX signal from DCAN1_RX pin on BBB. Green signal (channel 2): CANH signal from CANH pin on CAN transceiver board. Screenshot above shows complete CAN data frame being received from the PEAK CAN USB adapter. This is a standard CAN data frame.

17. CAN bus requires bit stuffing to maintain synchronization. Briefly explain how you could verify this via this setup. Perform the step accordingly and perform necessary screen shots.

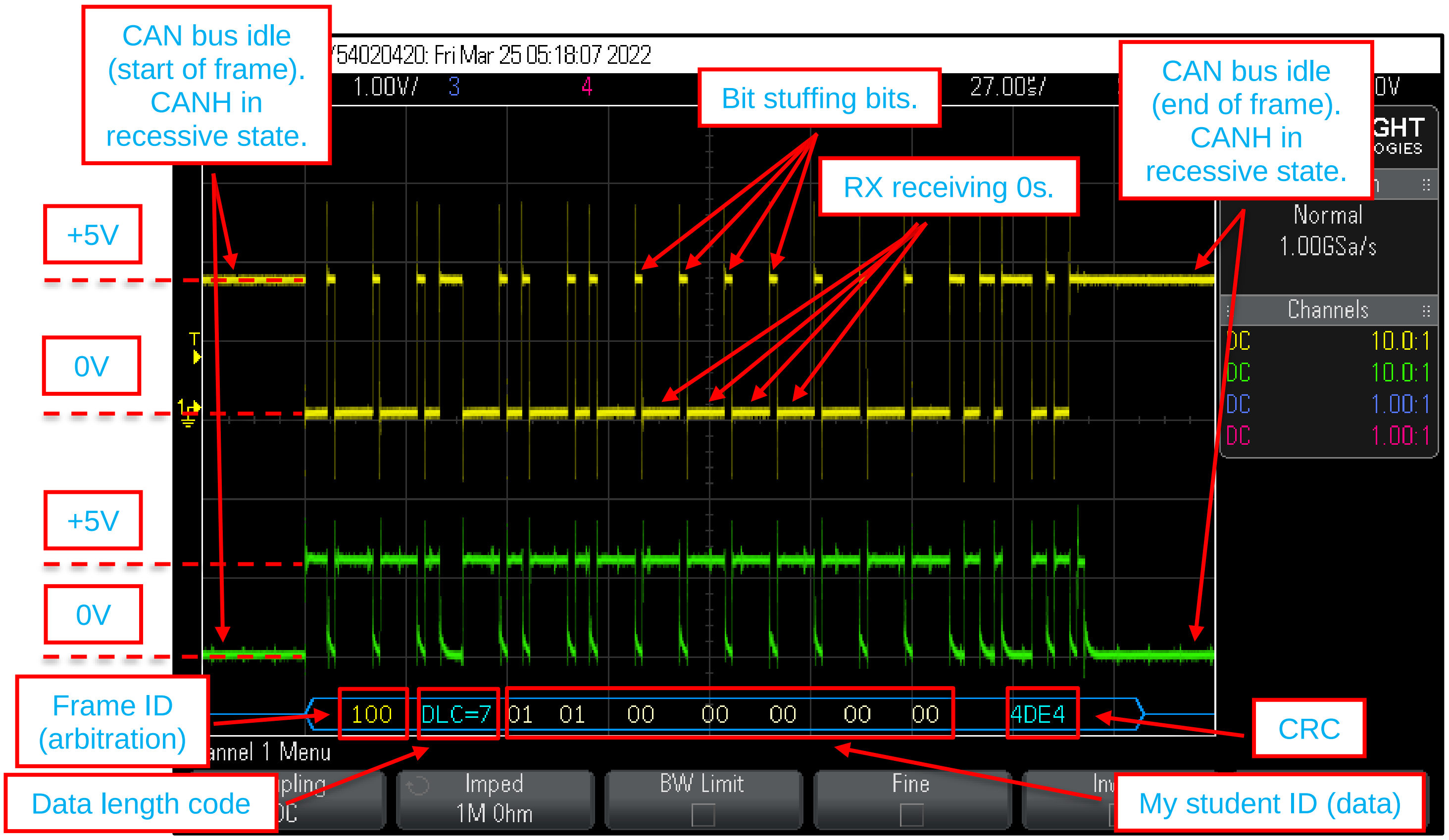


Figure 4. Yellow signal (channel 1): RX signal from DCAN1_RX pin on BBB. Green signal (channel 2): CANH signal from CANH pin on CAN transceiver board. Screenshot above shows complete CAN data frame being received from the PEAK CAN USB adapter. To view the bit stuffing process, a series of zeros were sent from the PEAK CAN USB adapter. This is a standard CAN data frame.

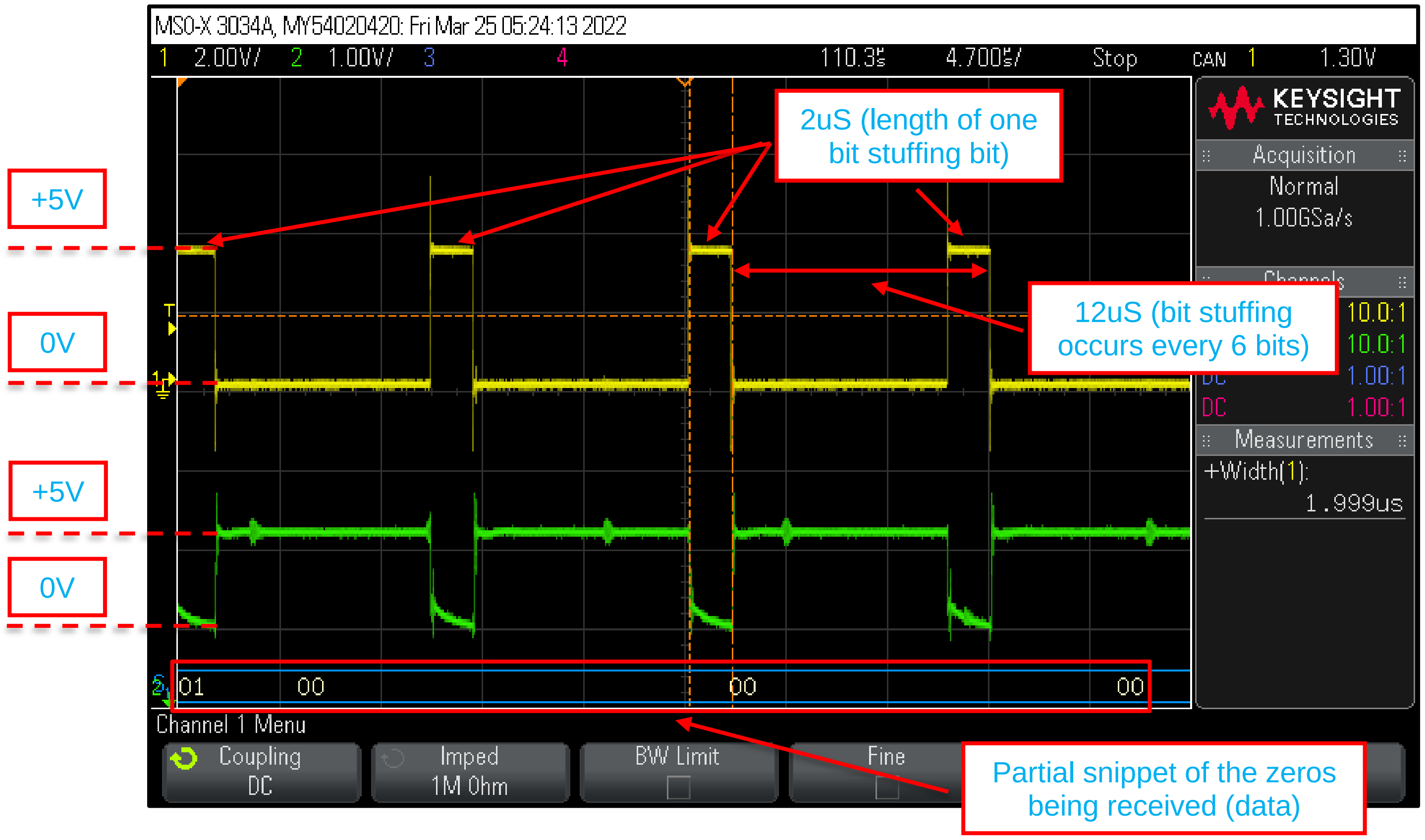

MSO-X 3034A, MY54020420: Fri Mar 25 05:24:13 2022
1 2.00V/ 2 1.00V/ 3 4 110.3s 4.700s/ Stop CAN 1 1.30V
KEYSIGHT TECHNOLOGIES
Acquisition
Normal
1.00GSa/s
Channels
10.0:1
10.0:1
DC 1.00:1
DC 1.00:1
Measurements
+Width(1):
1.999us
01 00 00 00
Channel 1 Menu
Coupling DC
Imped 1M Ohm
BW Limit
Fine
+5V
0V
+5V
0V
2uS (length of one bit stuffing bit)
12uS (bit stuffing occurs every 6 bits)
Partial snippet of the zeros being received (data)

Figure 5 (zoomed in signal of "Figure 4"). Yellow signal (channel 1): RX signal from DCAN1_RX pin on BBB. Green signal (channel 2): CANH signal from CANH pin on CAN transceiver board. Screenshot above shows partial CAN data frame being received from the PEAK CAN USB adapter. To view the bit stuffing process, a series of zeros were sent from the PEAK CAN USB adapter. The bit stuffing process occurs around every 6 bits that are received by the BBB (calculations shown below). This is a standard CAN data frame.

$$Bit\ time = \frac{1}{500kbps} = 2uS$$

$$Bit\ stuffing\ time = Bit\ time$$

$$Bit\ stuffing\ period = 12uS$$

$$Number\ of\ bits\ received\ before\ bit\ stuffing = \frac{12uS}{2uS} = 6\ bits$$

18. Connect up 2 or more BBB and perform communications. Show screen shots to illustrate the communications. Demo to the teacher.

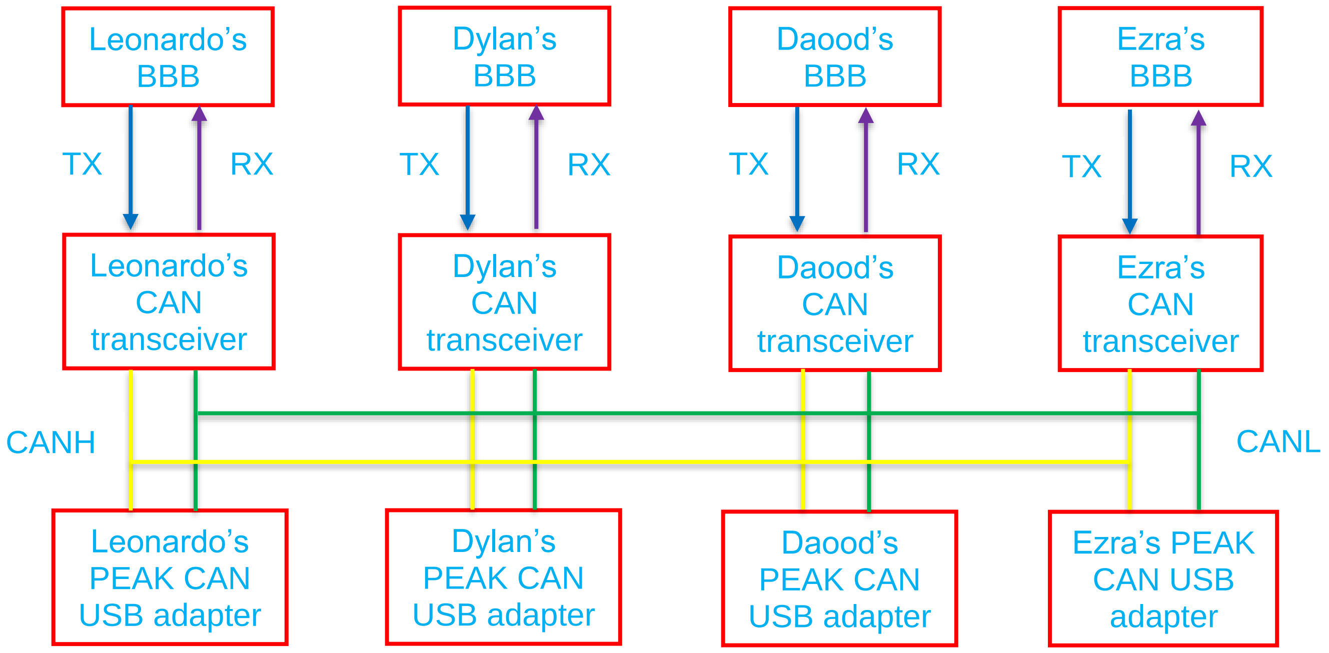
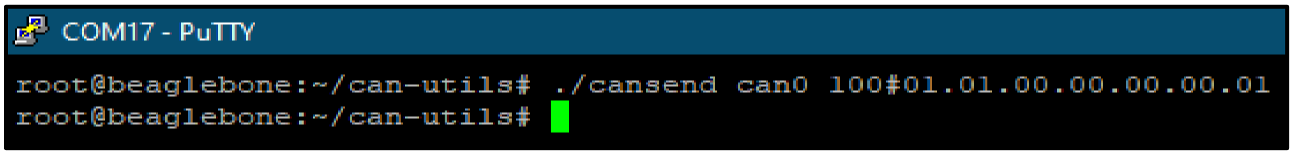

Leonardo's BBB
Dylan's BBB
Daood's BBB
Ezra's BBB
TX
RX
TX
RX
TX
RX
TX
RX
Leonardo's CAN transceiver
Dylan's CAN transceiver
Daood's CAN transceiver
Ezra's CAN transceiver
CANH
CANL
Leonardo's PEAK CAN USB adapter
Dylan's PEAK CAN USB adapter
Daood's PEAK CAN USB adapter
Ezra's PEAK CAN USB adapter

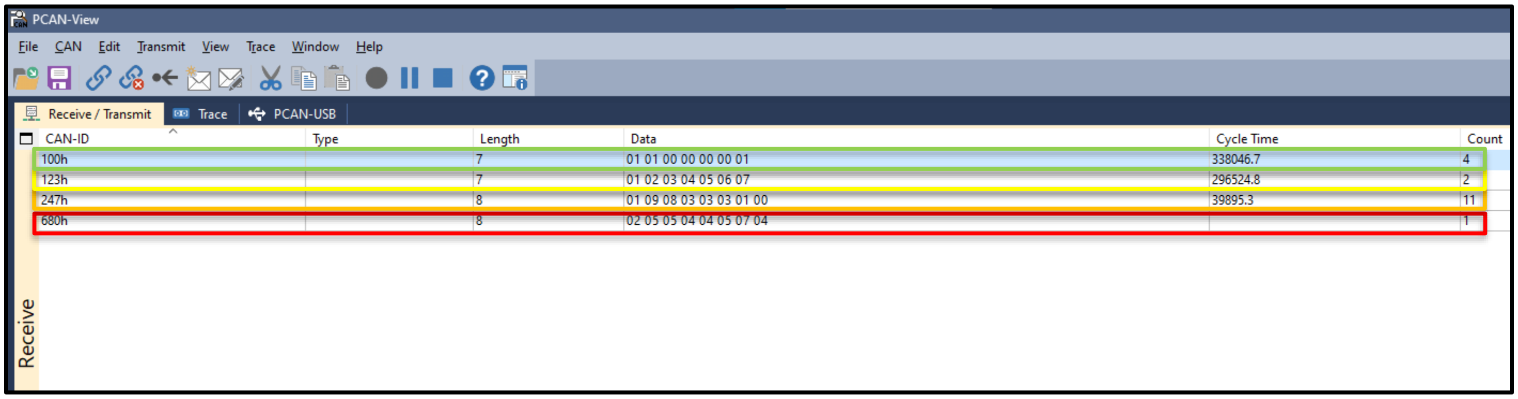
Figure 7. Physical hookup shown above.



COM17 - PuTTY

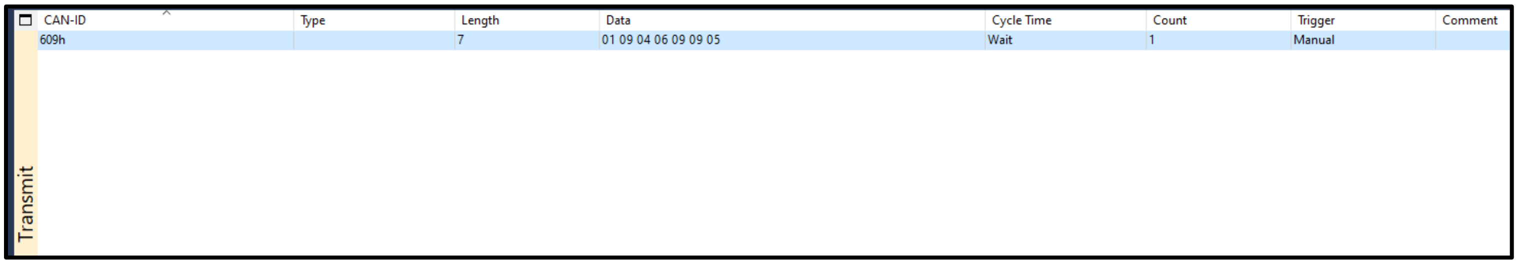
```
root@beaglebone:~/can-utils# ./cansend can0 100#01.01.00.00.00.00.01
root@beaglebone:~/can-utils# 
```

Figure 8. Using CANsend, a standard CAN data frame is sent out from my BBB shown above.



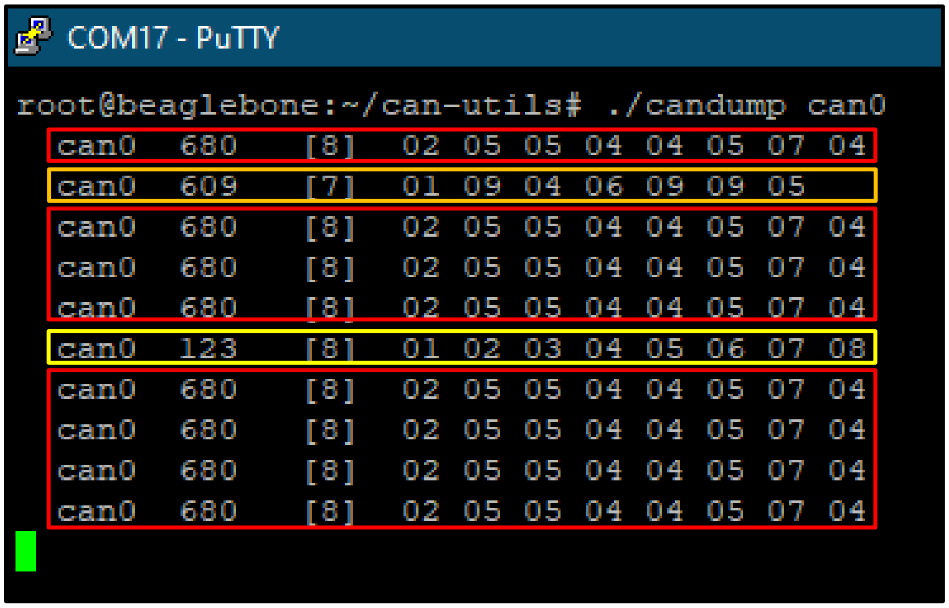
CAN-ID	Type	Length	Data	Cycle Time	Count
100h		7	01 01 00 00 00 00 01	338046.7	4
123h		7	01 02 03 04 05 06 07	296524.8	2
247h		8	01 09 08 03 03 03 01 00	39895.3	11
680h		8	02 05 05 04 04 05 07 04		1

Figure 9. Received CAN packets from mini CAN bus network on my BBB viewed in Pcan view software shown above. Green: myself, Yellow: Ezra, Orange: Dylan and Red: Daood.



CAN-ID	Type	Length	Data	Cycle Time	Count	Trigger	Comment
609h		7	01 09 04 06 09 09 05	Wait	1	Manual	

Figure 10. Using Pcan view software, a standard CAN data frame is sent out from my Peak CAN USB adapter shown above. It contains my student ID number (1946995).



```
root@beaglebone:~/can-utils# ./candump can0
  can0  680   [8]  02 05 05 04 04 05 07 04
  can0  609   [7]  01 09 04 06 09 09 05
  can0  680   [8]  02 05 05 04 04 05 07 04
  can0  680   [8]  02 05 05 04 04 05 07 04
  can0  680   [8]  02 05 05 04 04 05 07 04
  can0  123   [8]  01 02 03 04 05 06 07 08
  can0  680   [8]  02 05 05 04 04 05 07 04
  can0  680   [8]  02 05 05 04 04 05 07 04
  can0  680   [8]  02 05 05 04 04 05 07 04
  can0  680   [8]  02 05 05 04 04 05 07 04
```

Figure 11. Using CANdump, a complete list of received CAN packets on my BBB are shown above. CAN ID of 680: Daood (red), CAN ID of 609 (orange): Dylan and CAN ID of 123: Ezra (yellow).

Discussion:

- For Part A of the lab, a preliminary setup for the CAN hardware and software was done. A new cape was installed onto the BBB that allowed the BBB to perform CAN related tasks. The BBB was also checked for latest updates and if any were needed, they were installed onto the BBB.

  For Part B of the lab, the software configuration for the BBB CAN hardware was done. The kernel drivers for the CAN features on the BBB were loaded initially onto the BBB. The remaining two tasks were to configure the bit rate for the CAN communication and to ensure that the CAN interface on the BBB was up and running. An additional step was taken to ensure that the CAN interface would start up automatically upon each power cycle done on the BBB.

  For the final part of the lab, the setup done in the previous two parts of the lab were put to the test. Multiple tests were conducted to verify the CAN capabilities on the BBB. To begin these tests, a CAN transceiver board was connected to the RX and TX pins on the BBB (this board is responsible to convert CMOS levels from BBB to CAN signals on a CAN bus network). The TX and RX signals alongside the CANH signal were viewed on an oscilloscope. A CAN protocol analyzer on the scope was used to facilitate the observation of the CAN communication signals. CAN data frames were sent using the BBB as well as the Pcan view software. Likewise, CAN data frames were received on these two same devices. As a final test, multiple BBBs were put together to simulate a small-scale CAN bus network. These BBBs were sending and receiving multiple CAN packets concurrently.

  The lab was an overall success.

Conclusion:

- Successfully enabled CAN bus cape overlay on BBB.
- Successfully familiarized and understood the basic CAN bus communications.
- Successfully observed CAN bus communications using oscilloscope and protocol analyzer.