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

# Lab 5 : SocketCAN

Leonardo Fusser (1946995)

## Purpose:

a) To familiarize with SocketCAN in Linux.
b) To write C programming to transmit and receive of CAN frames.
c) To utilize mask and filter in CAN kernel.
d) To understand the operations of multi-master message-based CAN bus.

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

1. **No formal report required**. Report must be in a single MS Word document, and includes the following:

    a. Answer questions, screen shots (clearly labelled, with analysis/explanation).
    b. Include a final discussion and conclusion session.

2. Well commented, final C code.

## Lab Work:

**Part A : CAN communications in loopback**

1. To ease the process of development, this part shall be done in loopback mode. No hardware CAN interface is required at all. Setup your CAN interface for loopback mode. Refer to theory notes on the procedure.

2. Which command should you use to display CAN device details and statistics? Perform a screen shot on the command the corresponding results. Based on this information, has the loopback mode successfully enabled? Explain.

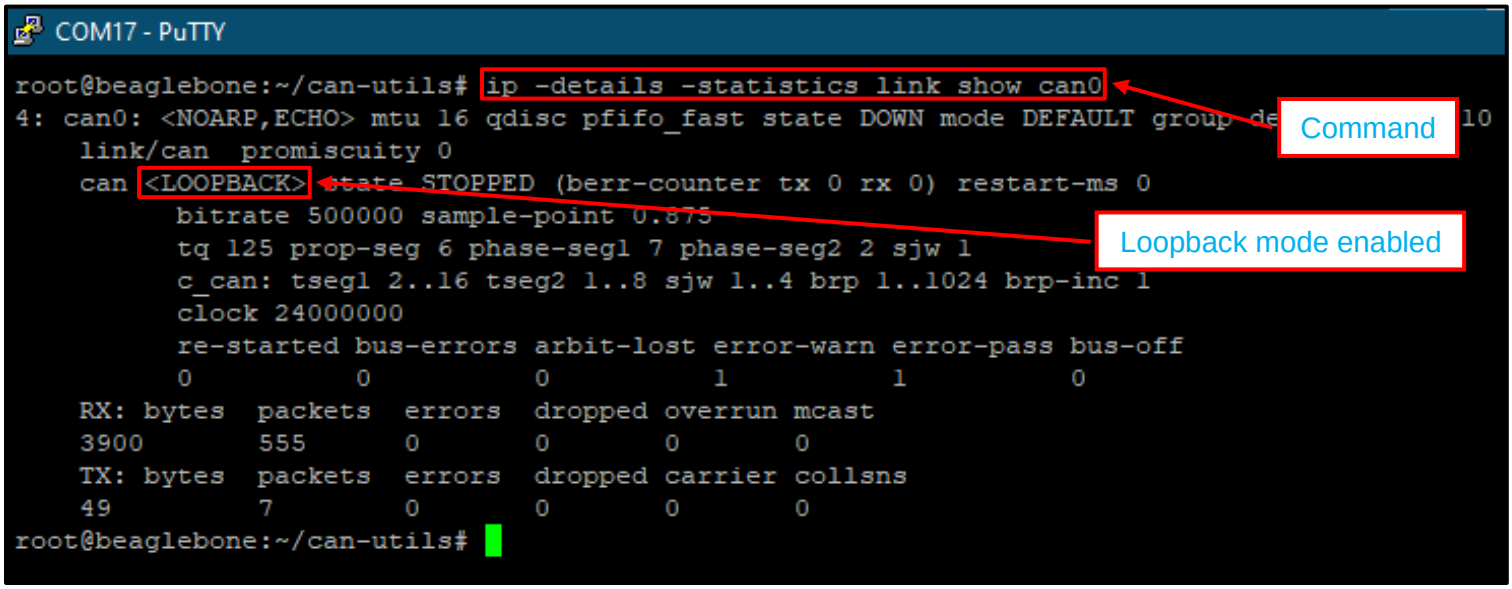


*Figure 1. The screenshot above shows a brief description of the CAN0 interface hardware status on the BBB. The command that was used for this was "`ip -details -statistics link show can0`". As shown above, the loopback mode was successfully enabled. State shows "STOPPED" because CAN interface was not turned on after enabling loopback mode.*

3. Copy cantest.c attached in this lab into your can-utils directory in BBB. Perform a compilation to ensure that there is no pre-existing error in the code.

4. Modify the code to add in the following session, as commented in the code:

    a. Setup transmit frame info. This includes can_id (your bench number) and data length.
    b. In the while loop:
        i. Setup your data to be transmitted. It must contain a running sequence number, followed by your student number.
        ii. Write the code to print out info of your transmit frame.
        iii. Write code to perform receiving of CAN frame.
        iv. Print out the details of the received CAN frame.

5. Compile your code and run. You should be able to receive all the CAN frame you have transmitted. Example of screen shot as shown below. Perform a screen shot on your output.

```
Transmit CAN frame :
         0x555 [8] 32 31 32 33 34 35 36 37
Receiving CAN frame:
         0x555 [8] 32 31 32 33 34 35 36 37

Transmit CAN frame :
         0x555 [8] 33 31 32 33 34 35 36 37
Receiving CAN frame:
         0x555 [8] 33 31 32 33 34 35 36 37

Transmit CAN frame :
         0x555 [8] 34 31 32 33 34 35 36 37
Receiving CAN frame:
         0x555 [8] 34 31 32 33 34 35 36 37
```

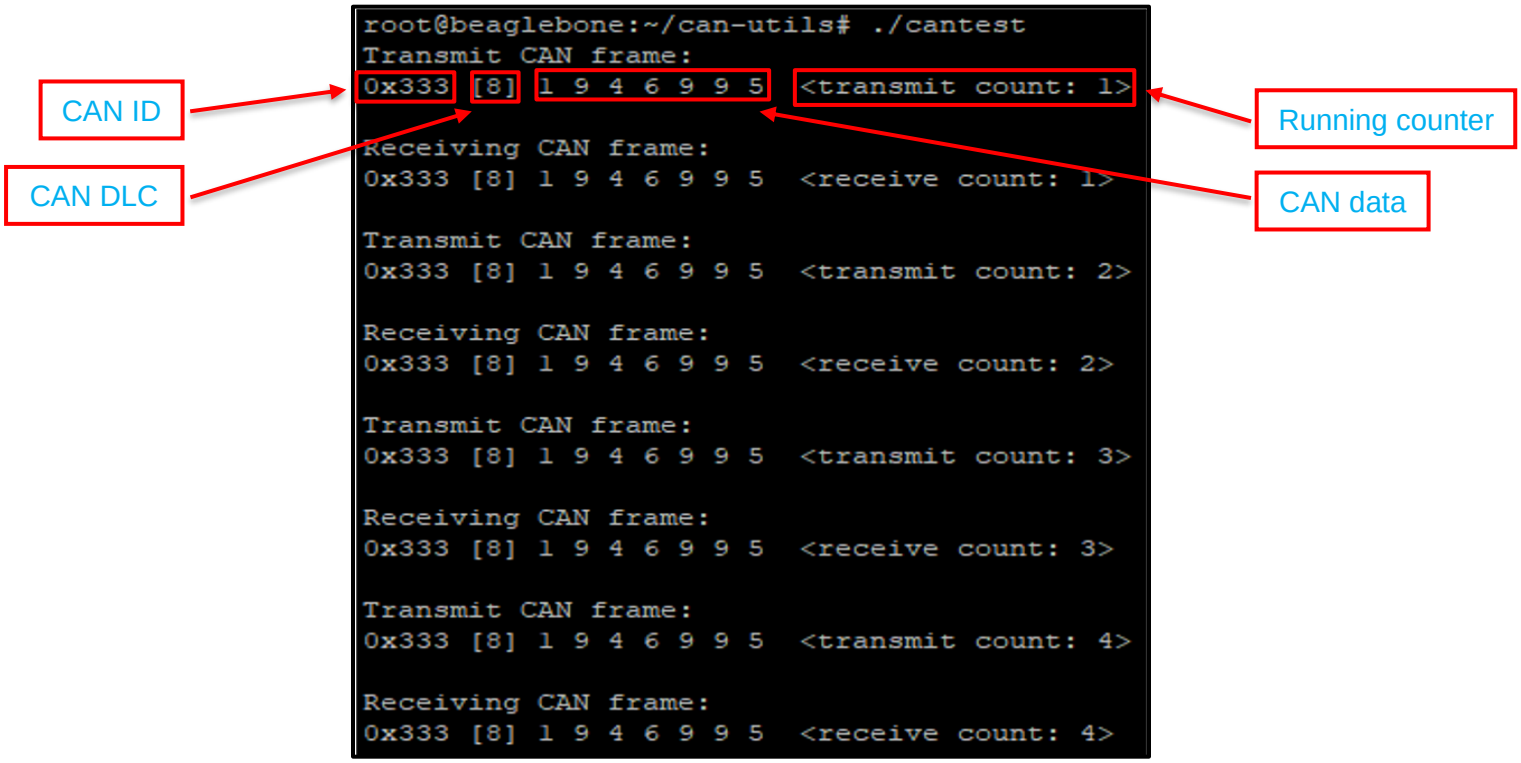


*Figure 2. Screenshot above shows both transmitting and receiving CAN packets with running counter. CAN ID is 0x333, CAN DCL is 8 and data is my student ID.*

6. Modify your code to setup receive filter, to receive message from your CAN_ID. Explain the code changed. Verify that the code works as expected. Perform screen shot(s) on your test output.
7.

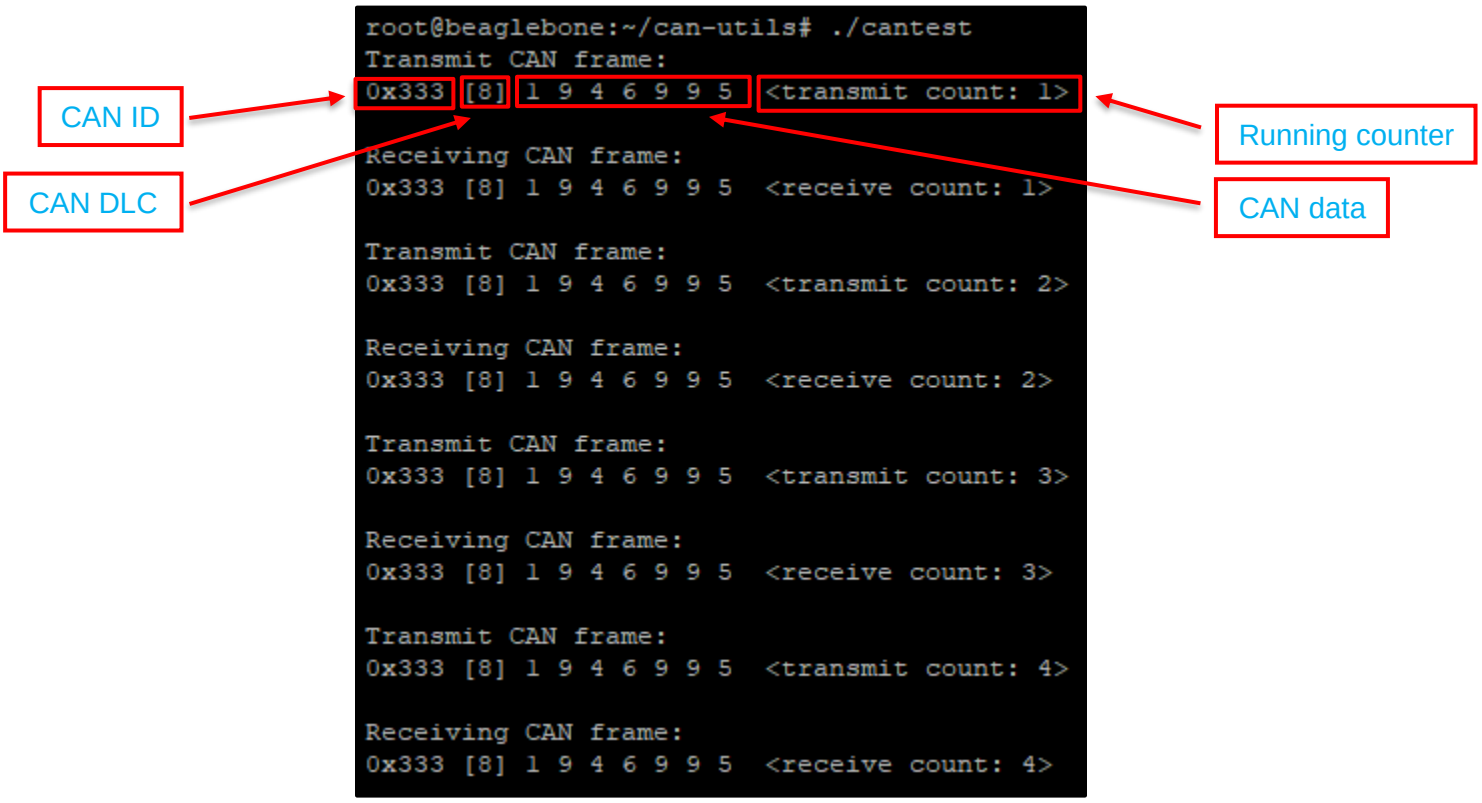


*Figure 3. Screenshot above shows when receiving filter is setup to only accept my CAN ID. CAN ID is 0x333, CAN DCL is 8 and data is my student ID. Code changed was to specify an ID of 0x333 and mask for standard CAN frames.*

8. Modify your receive filter, to receive message from other CAN _ID. Explain the code changed. Verify that the message would be rejected by the receiver due to the mismatch of message ID than expected. Perform screen shot(s) on your test output.

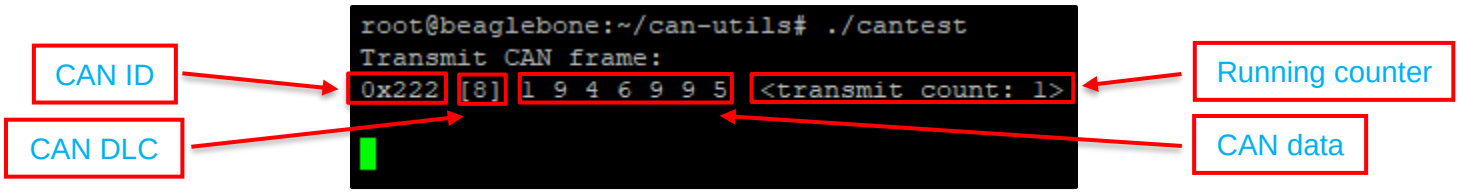


*Figure 4. Screenshot above shows when receiving filter is setup to only accept my CAN ID. Receiving filter is setup to only accept CAN frames with a CAN ID of 0x333. As shown above, receiving packet not shown because transmitted CAN frame has an ID of 0x222. CAN DCL is 8 and data is my student ID.*

**Part B : CAN bus communication with filter**

9. Turn off the local loopback. Setup all the necessary CAN bus connection. You must work in at least a group of 2.

10. Modify your code to setup receive filter. You are required to receive CAN message sent out by your teammate.

11. Compile your code and run. Perform a screen shot of your output, and a screen shot of the corresponding result on protocol analyzer. Explain your screen shots.

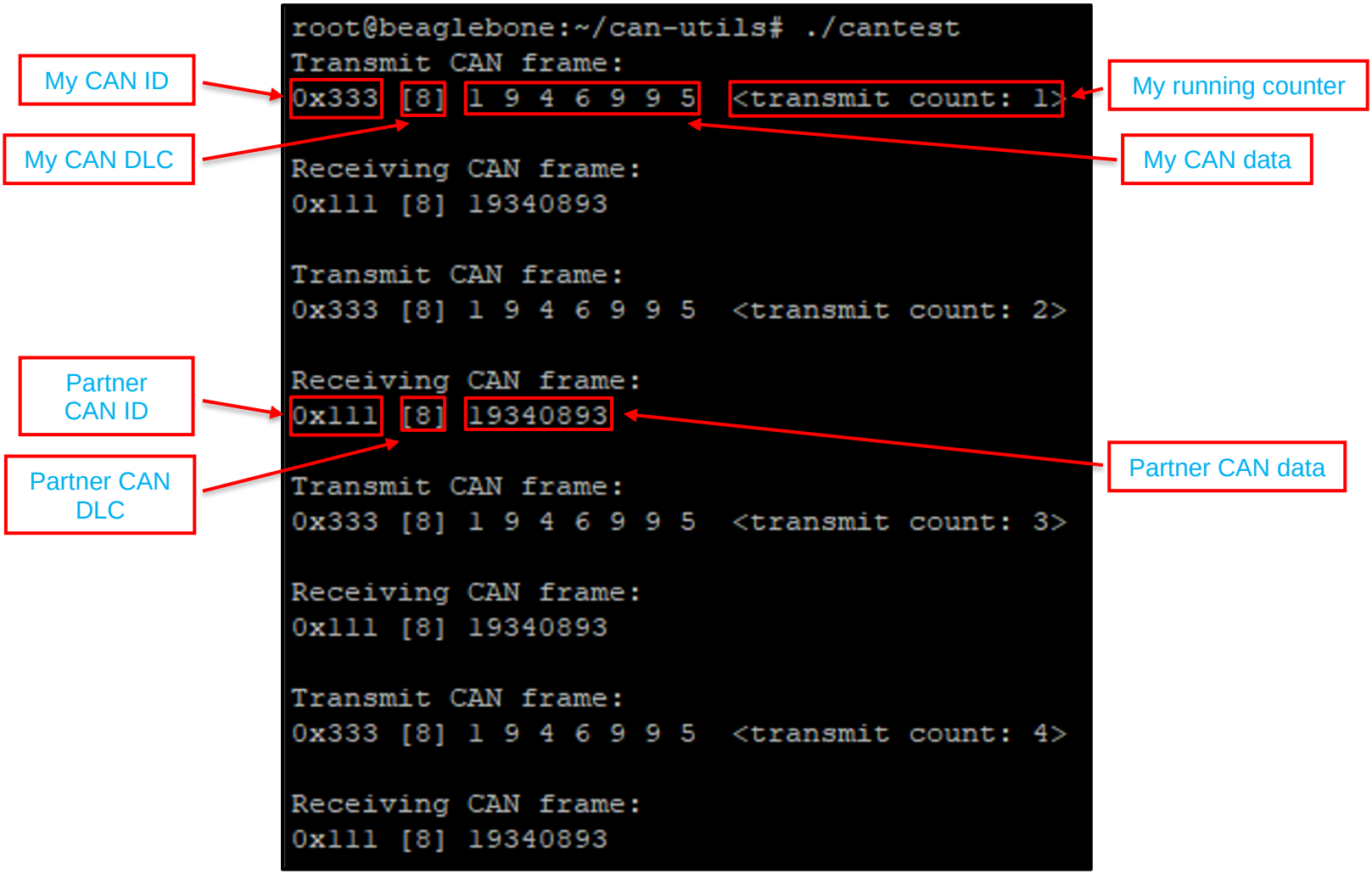


*Figure 5. Screenshot above shows transmitted and received CAN frames between my partner and I. Transmitted CAN frames belong to me and received CAN frames belong to my partner. Transmitted CAN frames: ID is 0x333, DLC is 8 and data is my student Id. Received CAN frames: ID is 0x111, DLC is 9 and data is my partner's student ID. Receiving running counter not shown but was added afterward.*

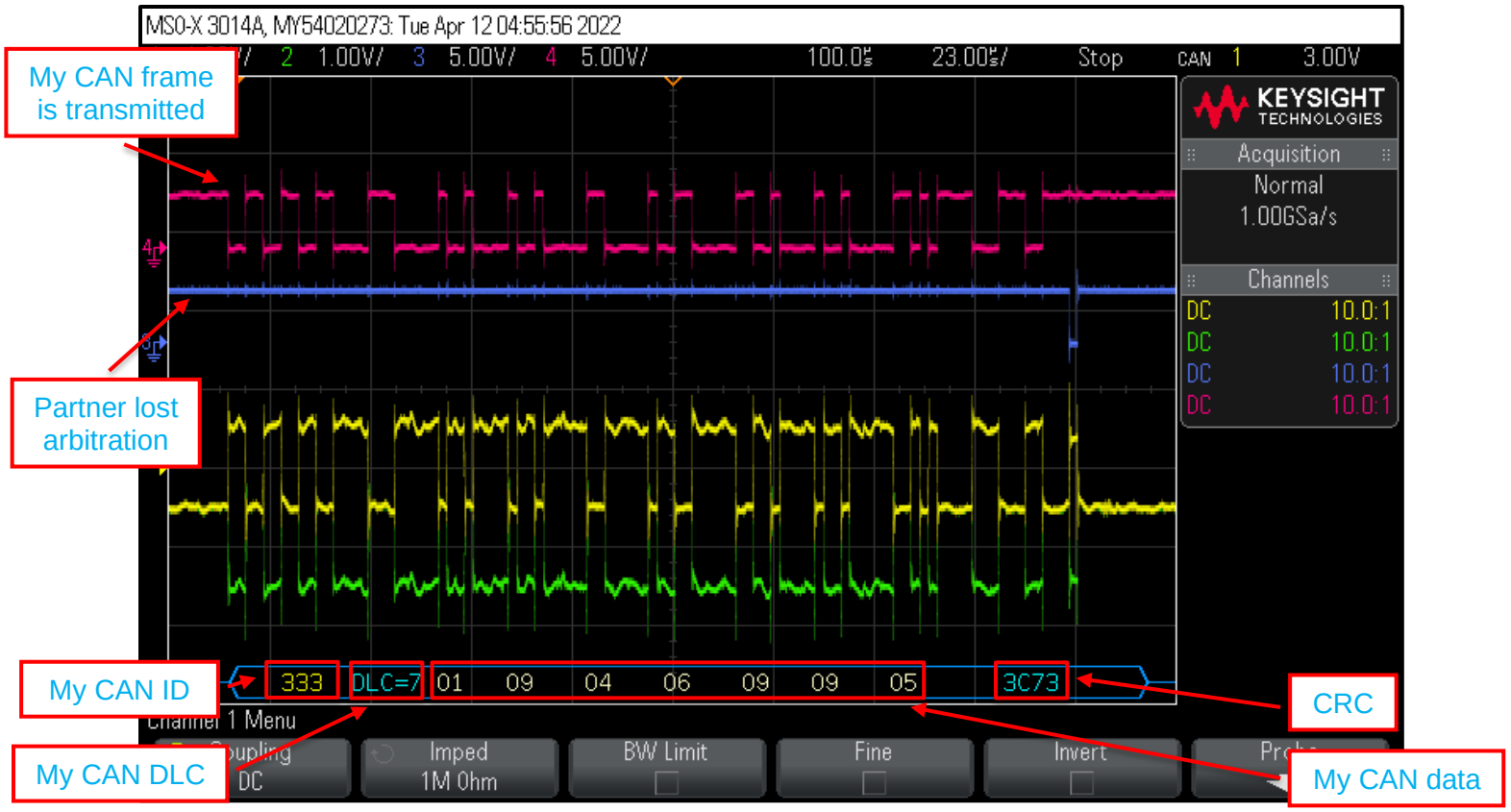


*Figure 6. Screenshot above shows captured CAN frame using CAN protocol analyzer on oscilloscope. Green signal is CAN L, yellow signal is CAN H, blue signal is TX pin on my partner's BBB and pink signal is TX pin on my BBB. My CAN frame is transmitted, and my partner's CAN frame is not being transmitted. This is due to my partner losing arbitration.*

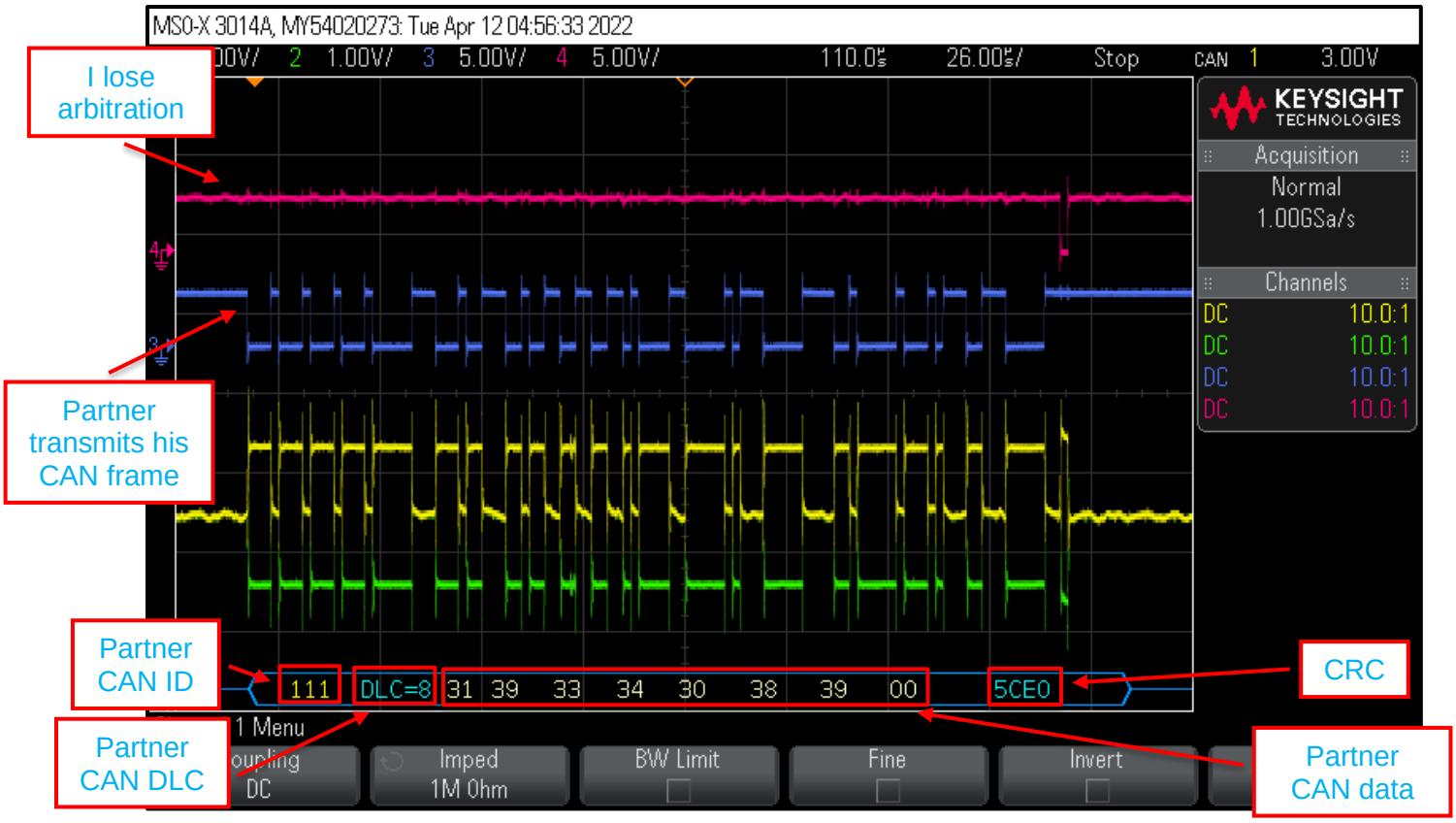


*Figure 7. Screenshot above shows captured CAN frame using CAN protocol analyzer on oscilloscope. Green signal is CAN L, yellow signal is CAN H, blue signal is TX pin on my partner's BBB and pink signal is TX pin on my BBB. My partner's CAN frame is transmitted, and my CAN frame is not being transmitted. This is due to my CAN frame losing arbitration.*

12. Connect the CAN bus together (per row of benches?). Set up your filter to accept only either odd or even can_id. Explain how that could be achieved. Perform a screen shot to illustrate the outcome.

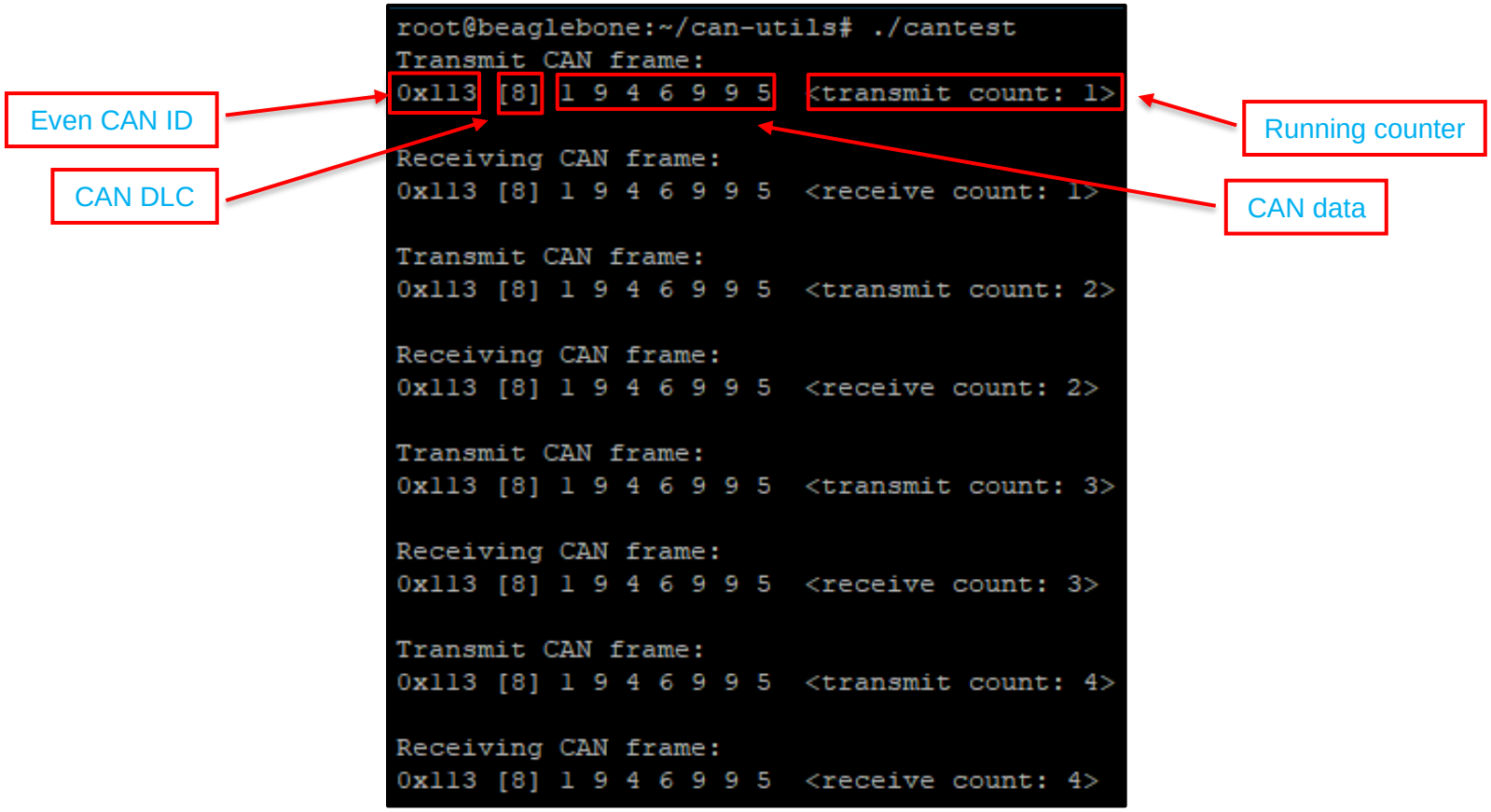


*Figure 8. Screenshot above shows how the receiving filter behaves when an odd CAN ID is received. This was achieved by setting the receiving filter's ID and mask to 0x001. This is because all odd CAN IDs will end with a "1", so even CAN IDs will be rejected by the receiving filter. CAN ID is 0x113, CAN DCL is 8 and data is my student ID.*

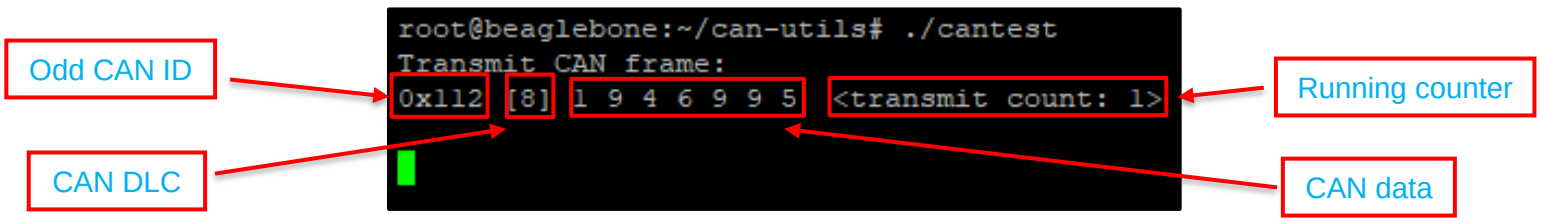


*Figure 9. Screenshot above shows how the receiving filter behaves when an even CAN ID is received. As explained above, since the receiving filter will reject all even CAN IDs, there is nothing received above. CAN ID is 0x112, CAN DCL is 8 and data is my student ID.*

Discussion:

- For the first part of the lab, two main things were done. Initially, the CAN hardware was initialized to be used in loopback mode. The CAN interface was disabled, loopback option was turned on and the interface was re-enabled. Afterwards, a simple C program was written to transmit and receive CAN frames. The C program was tested with a CAN dataframe that contained my student ID and a running counter. A receiving filter was initially setup to test if certain CAN frames would get rejected or not.

  For the second part of the lab, loopback mode was no longer needed. The opposite steps mentioned above were taken to disable loopback mode. A teammate was partnered with me to test our C program in sending out and receiving each other's CAN frames. This was also captured on an oscilloscope using a protocol analyzer. A final test was to see if we could only receive even CAN IDs by modifying the receiving filter to only accept even CAN IDs.

  The overall lab was a success.

Conclusion:

- Successfully familiarized SocketCAN in Linux OS.
- Successfully wrote a program to transmit and receive CAN frames in C.
- Successfully utilized receiving filter in CAN kernel (ID and mask).
- Successfully understood operations behind multi-master messaged-based CAN bus.