Krishna Sandilya Garimella Date : 10th June 2021

18ES701 : Embedded System for Automotive

Assignment 3 : Implementation of CAN2A and CAN2B

1) Introduction :

Can2.0A protocol and CAN2.0B protocol is used predominantly in the automotive networks field for communication between various components of the automobile. It works on the principle of differential voltage for dominant and recessive bits. Each message is broadcast and it is the duty of the node to listen to messages and decipher if they need the message or not based on the message ID. Message ID can be used to transmit messages of certain types and can be managed using a CAN bus arbitration mechanism. This mechanism gives highest priority with the message with the smallest ID and lowest priority to the message with the largest ID.

CAN2.0A has 11bits which means it can support 2^11 message ID's = 2048 unique messages at max.
CAN2.0B has support for 29bits and 11 bits so at max can support 2^29 message ID's= 536870912 unique messages at max.

2) Implementation of CAN2.0 in MATLAB

The code is split into a main function which creates a receiving channel,starts it and transmits and receives messages on the channel and plots the messages received.

There is another function which generates messages in a periodic fashion and runs the message passing.

The main function:

```
%% BASIC CAN PROTOCOL Message Passing
%Below code uses CAN channel functionality in MATLAB for transmission and Reception of CAN Messages in loopback configuration.
%% Receiving Channel (rxCh) being created
rxCh = canChannel('MathWorks', 'Virtual 1', 2);

%% Starting the receiving Channel
start(rxCh);

%% Transmit Messages
%generateMsgsMod function transmits CAN messages at periodic rates by creating
%them
generateMsgsMod();

%% Receive Messages
% Receive all the messages generated and transmitted periodically by the generateMsgsMod function above.
rxMsg = receive(rxCh, Inf, 'OutputFormat', 'timetable');
rxMsg

%% Receiving channel being stopped
stop(rxCh);
```

```
%% Plot Received Messages
plot(rxMsg.Time, rxMsg.ID, 'x')
ylim([0 3500])
xlabel('Timestamp')
ylabel('CAN Identifier')
```

Note :
CAN2.0A Maximum ylim is [0 2047]
CAN2.0B Maximum ylim is [0 536870911]

The message generating function(generateMsgsMod):

```
function generateMsgsMod()
   %Create the CAN message with following arguments
   %canMessage(<Message ID>,<Enable CAN 2.0B>,Data Length in bits)
   msgTx01 = canMessage(01, true, 4);
   msgTx500 = canMessage(500, true, 0);
   msgTx800 = canMessage(800, true, 6);
   msgTx1110 = canMessage(1110, true, 0);
   msgTx3000 = canMessage(3000, true, 8);

   % Create a CAN channel on which to transmit.
   txCh = canChannel('MathWorks', 'Virtual 1', 1);

   % Register each message on the channel at a specified periodic rate with Arguments
   %transmitPeriodic(<NameOfTransmissionChannel,MessageVariableName,'ON',<Periodicity in seconds>)
   transmitPeriodic(txCh, msgTx01, 'On', 0.500)
   transmitPeriodic(txCh, msgTx500, 'On', 0.500)
   transmitPeriodic(txCh, msgTx800, 'On', 0.500);
   transmitPeriodic(txCh, msgTx1110, 'On', 1);
   transmitPeriodic(txCh, msgTx3000, 'On', 1);

   % Start the CAN transmission channel.
   start(txCh);

   % Run for several seconds incrementing the message data regularly.
   for ii = 1:40
      % Increment the message data bytes.
      msgTx01.Data = msgTx01.Data + 1
      msgTx500.Data = msgTx500.Data + 1
      msgTx800.Data = msgTx800.Data + 1
      msgTx1110.Data = msgTx1110.Data + 1
      msgTx3000.Data = msgTx3000.Data + 1

      % Wait for a time period.
      pause(0.1);
   end

   % Stop the CAN transmission channel.
   stop(txCh);
end
```

Note :
CAN2.0A set Argument as False
CAN2.0B set Argument as True

3) Output of CAN2.0 in MATLAB

CAN2.0A Output

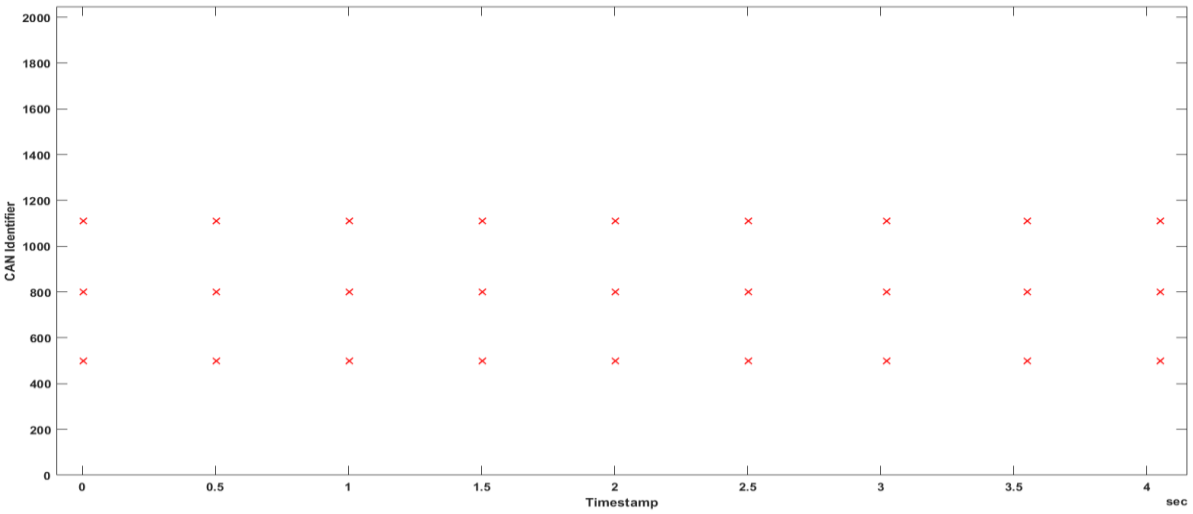


Figure 1 : Messages in CAN2.0A (X:Time(s) Y:MsgID)

It is seen above that there are 4 messages of ID 500,800,1100 respectively which have a periodic rate of 0.5 implemented using the CAN2.0A module. On a cursory glance it may seen like as if the messages are all occurring together without any arbitration. However, seeing the below table of details of Receiver we find out that the arbitration has occurred.

rxMsg =

27×8 timetable

	Time	ID	Extended	Name	Data	Length	Signals	Error	Remote
1	0.0031931 sec	500	false	{0×0 char}	{1×0 uint8}	0	{0×0 struct}	false	false
	0.0031957 sec	800	false	{0×0 char}	{1×4 uint8}	4	{0×0 struct}	false	false
	0.0031978 sec	1110	false	{0×0 char}	{1×4 uint8}	4	{0×0 struct}	false	false
2	0.50341 sec	500	false	{0×0 char}	{1×0 uint8}	0	{0×0 struct}	false	false
	0.50342 sec	800	false	{0×0 char}	{1×4 uint8}	4	{0×0 struct}	false	false
	0.50343 sec	1110	false	{0×0 char}	{1×4 uint8}	4	{0×0 struct}	false	false
	1.0033 sec	500	false	{0×0 char}	{1×0 uint8}	0	{0×0 struct}	false	false
	1.0033 sec	800	false	{0×0 char}	{1×4 uint8}	4	{0×0 struct}	false	false
	1.0033 sec	1110	false	{0×0 char}	{1×4 uint8}	4	{0×0 struct}	false	false
	1.5034 sec	500	false	{0×0 char}	{1×0 uint8}	0	{0×0 struct}	false	false
	1.5034 sec	800	false	{0×0 char}	{1×4 uint8}	4	{0×0 struct}	false	false
	1.5034 sec	1110	false	{0×0 char}	{1×4 uint8}	4	{0×0 struct}	false	false
	2.0033 sec	500	false	{0×0 char}	{1×0 uint8}	0	{0×0 struct}	false	false
	2.0033 sec	800	false	{0×0 char}	{1×4 uint8}	4	{0×0 struct}	false	false
	2.0033 sec	1110	false	{0×0 char}	{1×4 uint8}	4	{0×0 struct}	false	false

Figure 2 : CAN2.0A Message Instances Received in a Table

Seeing the first instance of the messages we can see that the message with lowest ID 500 gets maximum priority followed by 800 followed by 1110. The first instances of the messages have a deadline of 0.5 seconds and the second instance has a deadline of 1 second and both message sets occur within deadline following full arbitration.

CANN2.0B Output

Note :
Message of ID 3000 not possible in CAN2.0A so it shows CAN2.0B works

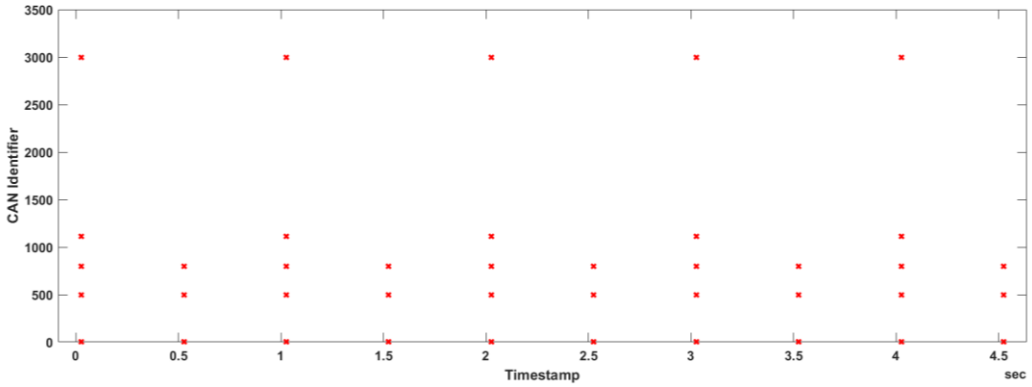


Figure 3 : Messages in CAN2.0B (X:Time(s) Y:MsgID)

It is seen above that there are 3 messages of ID 1,500,800 having periodicity 0.5 second and 2 messages of ID 1110,3000 which have a periodicity 1 second implemented using the CAN2.0B module. On a cursory glance it may seem like as if the messages are occurring together without any arbitration. However, seeing the below table of details of Receiver we find out that the arbitration has occurred.

rxMsg =

40×8 timetable

	Time	ID	Extended	Name	Data	Length	Signals	Error	Remote
1	0.009127 sec	1	true	{0×0 char}	{1×4 uint8}	4	{0×0 struct}	false	false
	0.0091385 sec	500	true	{0×0 char}	{1×0 uint8}	0	{0×0 struct}	false	false
	0.0091471 sec	800	true	{0×0 char}	{1×6 uint8}	6	{0×0 struct}	false	false
	0.0091544 sec	1110	true	{0×0 char}	{1×0 uint8}	0	{0×0 struct}	false	false
	0.0091612 sec	3000	true	{0×0 char}	{1×8 uint8}	8	{0×0 struct}	false	false
2	0.50904 sec	1	true	{0×0 char}	{1×4 uint8}	4	{0×0 struct}	false	false
	0.50904 sec	500	true	{0×0 char}	{1×0 uint8}	0	{0×0 struct}	false	false
	0.50905 sec	800	true	{0×0 char}	{1×6 uint8}	6	{0×0 struct}	false	false
3	1.0091 sec	1	true	{0×0 char}	{1×4 uint8}	4	{0×0 struct}	false	false
	1.0091 sec	500	true	{0×0 char}	{1×0 uint8}	0	{0×0 struct}	false	false
	1.0091 sec	800	true	{0×0 char}	{1×6 uint8}	6	{0×0 struct}	false	false
	1.0091 sec	1110	true	{0×0 char}	{1×0 uint8}	0	{0×0 struct}	false	false
	1.0091 sec	3000	true	{0×0 char}	{1×8 uint8}	8	{0×0 struct}	false	false
	1.5095 sec	1	true	{0×0 char}	{1×4 uint8}	4	{0×0 struct}	false	false
	1.5095 sec	500	true	{0×0 char}	{1×0 uint8}	0	{0×0 struct}	false	false
	1.5095 sec	800	true	{0×0 char}	{1×6 uint8}	6	{0×0 struct}	false	false
	2.0091 sec	1	true	{0×0 char}	{1×4 uint8}	4	{0×0 struct}	false	false
	2.0091 sec	500	true	{0×0 char}	{1×0 uint8}	0	{0×0 struct}	false	false
	2.0091 sec	800	true	{0×0 char}	{1×6 uint8}	6	{0×0 struct}	false	false
	2.0091 sec	1110	true	{0×0 char}	{1×0 uint8}	0	{0×0 struct}	false	false
	2.0091 sec	3000	true	{0×0 char}	{1×8 uint8}	8	{0×0 struct}	false	false

Figure 4 : CAN2.0B Message Instances Received in a Table

Seeing the first instance of the messages we can see that the message with lowest ID 1 gets maximum priority followed by 500,800 followed by 1110,3000. This demonstrates the Bus arbitration mechanism where lower Message ID's get maximum priority. In the 2nd instance only messages with ID1,500,800 only occur as their periodicity is 0.5 seconds. In both first and second instances all messages meet their deadlines. In the 3rd instance we can see the message 1110 and 3000 occurring again as their period is 1second and they meet the deadline by occurring once within 0-1second and 1-2second interval.

4) Implementation of CAN2.0 in SIMULINK

CAN 2.0A and CAN2.0B in Simulink can be implemented using the Vehicle network toolbox. The blocks used and their respective settings are given below.

CAN Transmission Unit.

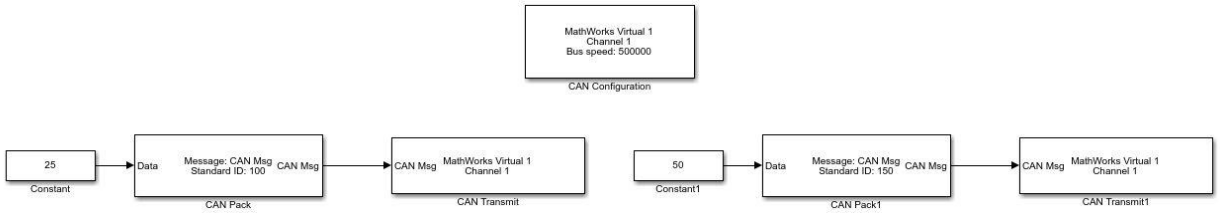
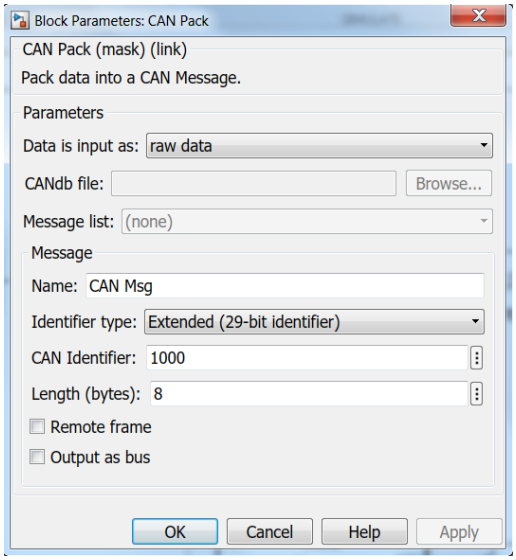


Figure 5 : CAN Transmission Unit

It consists of a CAN configuration block which chooses the CAN channel to communicate on. Since we have two messages to show arbitration mechanism as well we would need two sets of CAN Pack and Can Transmit blocks on the transmission channel.



The input to the CAN Pack block would be a constant data value chosen for both messages as 25 and 50 respectively. These constant numeric values are packed in CAN format by the CAN pack module using the CAN2.0A or CAN2.0B methods. The output of the CAN Pack is given to the CAN Transmit block where the Periodicity for both messages is set to 0.5 seconds and in the CAN Pack block the length of the data is set as 8 bytes.

The Identifier type is important as that can help change the protocol to CAN2.0A or CAN2.0B. If Extended 29bit identifier is selected then CAN2.0B communication is being implemented else Standard 11 bit CAN2.0A communication is being implemented.

The Bus speed and Channel is set as default in the CAN configuration block as 500000 and Virtual Channel 1

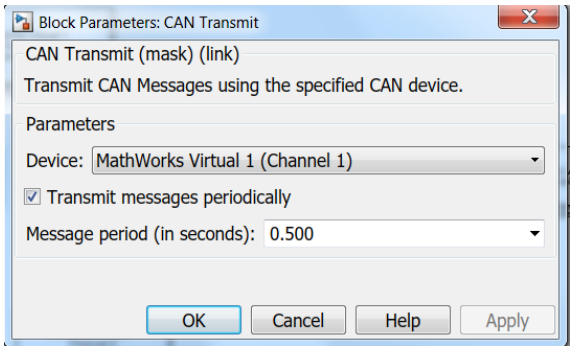


Figure 6 : Block Parameters of CAN Pack and CAN Transmit

CAN Reception Unit.

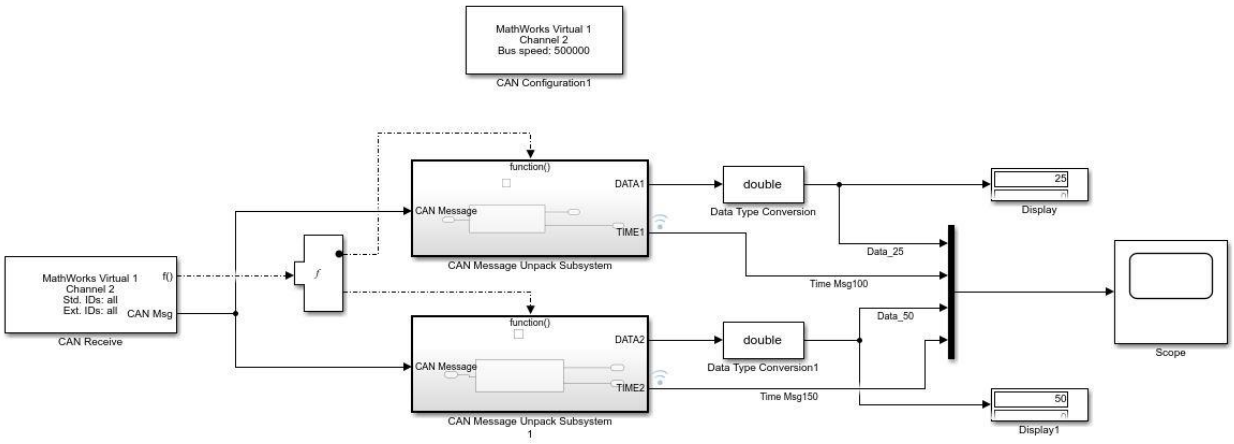
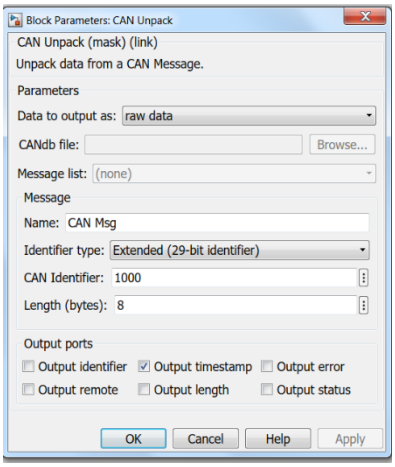


Figure 7 : CAN Reception Unit

The CAN reception unit captures the transmitted message using the CAN2.0A or CAN2.0B protocol and sends it across for unpacking using the message Unpack subsystem routed through a function select block which will unpack message depending on the message ID. Both the subsystems have the same setting except that one of them is used for unpacking for message of id 3000 and the other of message id 1000. The output of this subsystem is the data and the time of reception. Since the time is in double format, the data is converted to a double format too to ensure scaling while plotting.



The CAN Unpack subsystem is given in the figure below where the CAN unpack block settings are set to Extended(29bit) in case of CAN2.0B communication and Standard(11bit) in case of CAN2.0A communication.

The CAN receive Block parameters are set on Channel 2 for a separate reception channel same as a separate configuration block for reception with same bus speed as transmission configuration block but different channel in this case. Which is Channel 2.

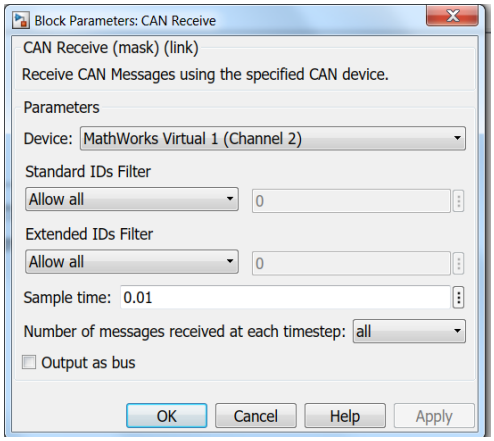


Figure 8 : Block Parameters for CAN Receive and CAN Unpack

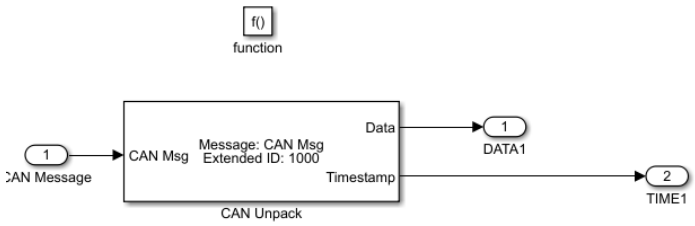


Figure 9 : CAN Unpack Subsystem

5) Output of CAN2.0 in SIMULINK

CAN2.0A Output

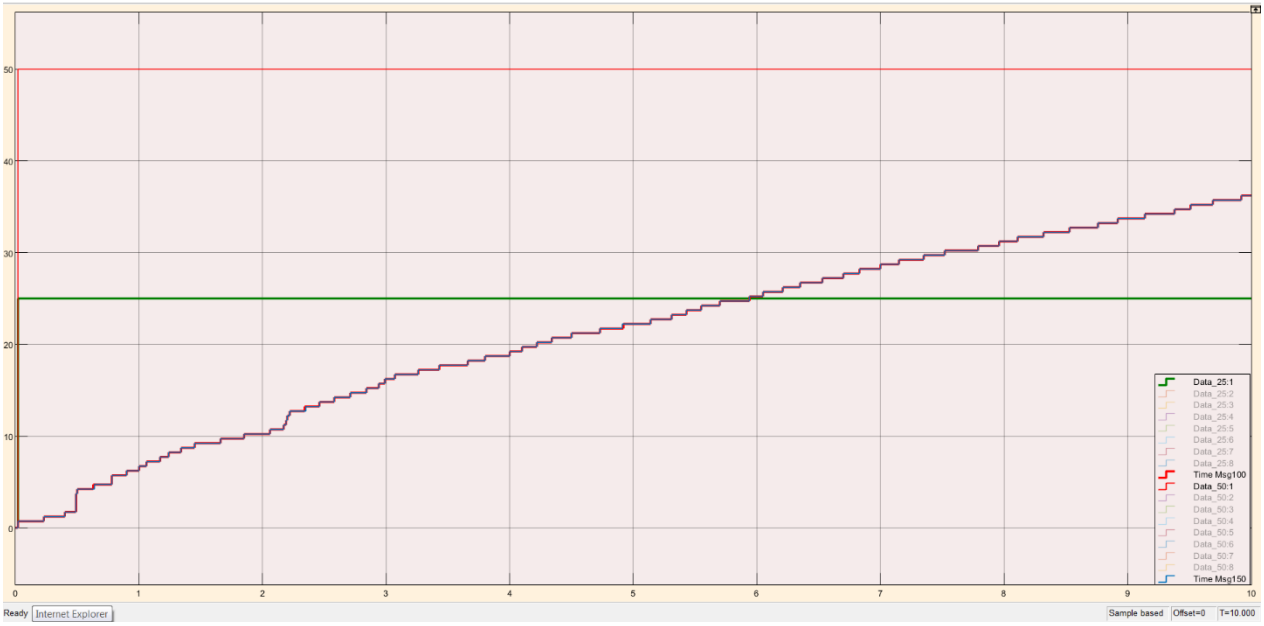


Figure 10 : CAN2.0A Output

On simulation of CAN2.0A protocol on Simulink and observing the graphs we can see the data of both the channels being received which is 25 and 50 respectively denoted by the horizontal red and green line. The time is shown as two signals for two transmitting channels with message ID 100 and 150 overlapping each other as red and blue signals and here each message if configured to be having a periodicity of 0.5seconds.

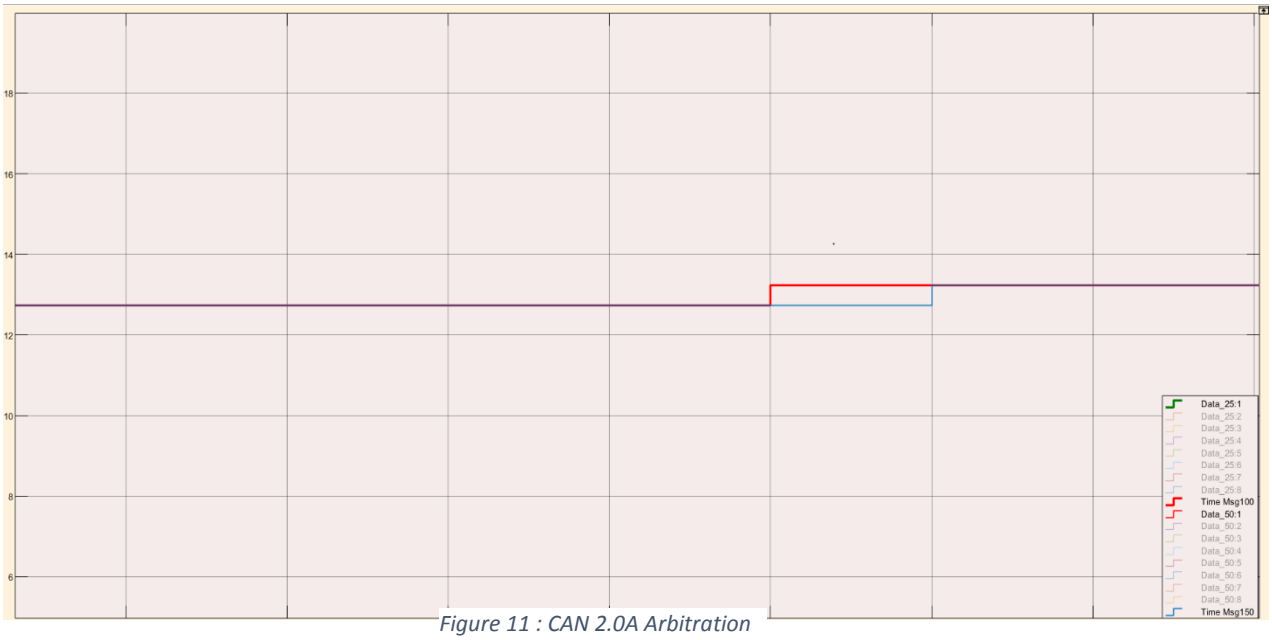


Figure 11 : CAN 2.0A Arbitration

On a cursory glance the arbitration mechanism which is the signal of 100 message id must be occurring first as a time signal over the signal with message id 150. We may have to zoom in a bit into the signal to prove the phenomenon as shown in the image below.

This is the zoomed in version of the time signals of the message id 100 and message id 150 signal shown as red and blue respectively. It is easy to see that the signal containing the message id 100 occurs first which is the proof of arbitration in the CAN2.0 A channel.

CAN2.0B Output (Message ID set as 3000 as it is not possible in CAN2.0A and shows CAN2.0B)

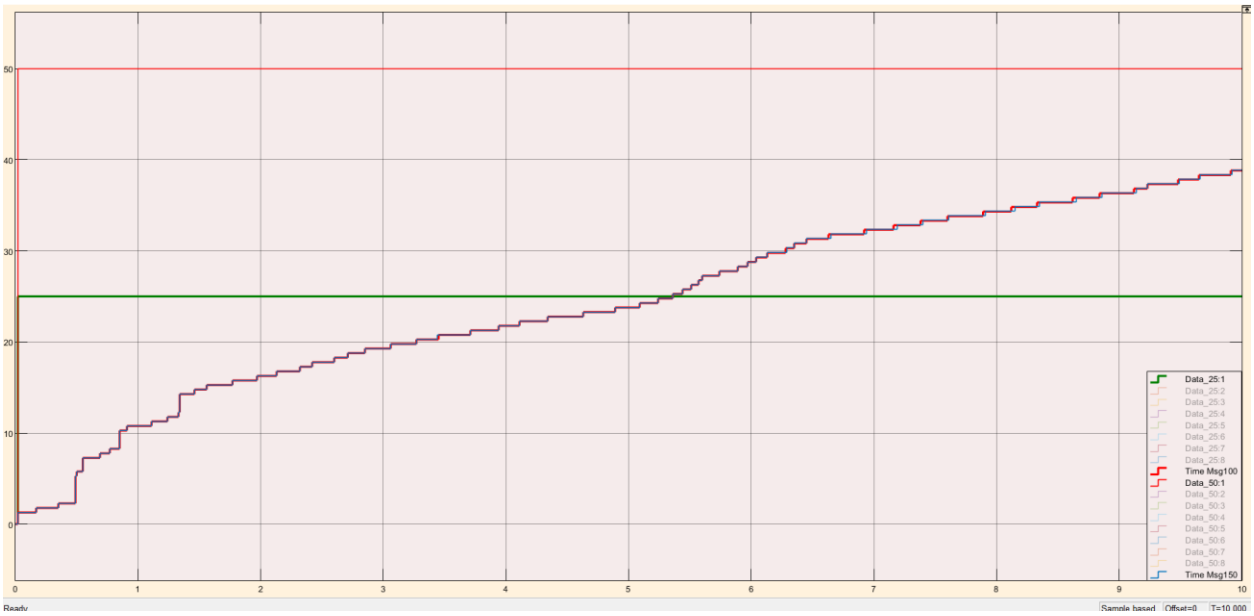


Figure12 : CAN2.0B Output

On simulation of CAN2.0B protocol on Simulink and observing the graphs we can see the data of both the channels being received which is 25 and 50 respectively denoted by the horizontal red and green line. The time is shown as two signals for two transmitting channels with message ID 1000 and 3000 overlapping each other as red and blue singals and here each message is configured with a periodicity of 0.5 seconds.

On a cursory glance the arbitration mechanism which is the signal of 1000 message id must be occurring first as a time signal over the signal with message id 3000. We may have to zoom in a bit into the signal to prove the phenomenon as shown in the image below.

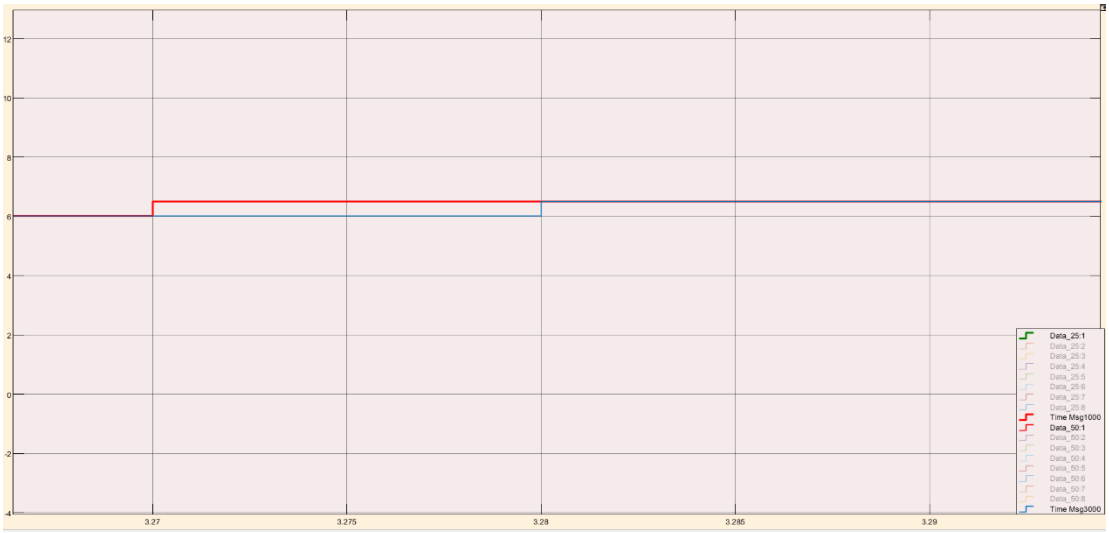


Figure 13 : CAN2.0B Arbitration

This is the zoomed in version of the time signals of the message id 1000 and message id 3000 signal shown as red and blue respectively. It is easy to see that the signal containing the message id 1000 occurs first which is the proof of arbitration in the CAN2.0 B channel.

6) References

[1] https://in.mathworks.com/help/vnt/ug/transmit-and-receive-can-messages.html

[2] https://in.mathworks.com/help/vnt/ug/build-can-communication-simulink-models.html