

MEEM/EE-5750: Distributed Embedded Control Systems

LAB REPORT– 5

Remote Electronic Throttle Control Via CAN

Submitted by:



Syed Anisur Rehaman
Mechanical Engineering



Rubeena Parveez
Electrical Engineering



Raveena Pai
Electrical Engineering



Dhruv Mehta
Mechanical Engineering

LAB BATCH – L05

GROUP – 03 and 01

INSTRUCTOR

Dr. BO CHEN

Department of Mechanical Engineering – Engineering Mechanics

Department of Electrical and Computer Engineering

Michigan Technological University

Table of Contents

Introduction.....	3
Input-Output and Control Logic.....	3
Model Development.....	4
Results and Validation.....	9
Conclusion.....	24
References.....	25

Introduction:

Controller Area Network (CAN) is a multi-master priority based serial communication protocol for connecting two or more ECU's. CAN network has 2 parts- Part A- data communication using 11-bit message ID known as standard format and Part-B – Data communication using 29-bit message ID known as extended format. This lab was performed to develop a remote electronic throttle control using CAN communication between two ECU's using standard data format. Sensors inputs and actuation signals were communicated between the two ECU's. Two message definition files were generated for the two message files to be transmitted between ECU's to define the message ID, endianness, payload size, ID filter and scale/offset of payload. The CAN bus was monitored in the CANKing output window.

Input/output and Control Logic:

Accelerator Pedal Position:

The driver generates commands for the throttle opening through an accelerator pedal which contains a pair of sensors i.e. potentiometers that ensure a fail-safe operation. These signals are input to the ECU-2 through analog input channel. As the accelerator is pressed, voltage signals are read at the ECU-2 and these signals are sent to ECU-1 through CAN network conveying the desired position.

Throttle Position Sensor:

The sensor is placed at the butterfly shaft to monitor the position of the throttle with a pair of potentiometers that provide variable resistance depending on the throttle valve position. These signals are analog input to the MotoTron ECU-2 that provides the actual position of the throttle valve. A pair of sensors is used to ensure fail-safe operation. These sensor signals are transmitted to ECU-1 over the CAN communication network.

DC Motor:

The DC motor of the throttle body receives signal from the H-bridge in the MotoTron ECU-2 depending on the output of the PID controller from ECU-1 to drive the throttle plate to the required position.

CAN Bus:

Controller Area Network protocol is used to communicate between two ECUs. To monitor bus, Kvaser CANKing software for windows is used. It is an application that allows to send and receive CAN messages.

Control Logic:

The control objective is to open and close the throttle plate to the required position rapidly via CAN communication. To achieve this, CAN messages for sensor signals and actuation signal are defined. The signals from sensors i.e. TPS and PPS read at ECU-2 are sent over CAN bus to ECU-1. A closed-loop discrete PID controller uses the received accelerator pedal position and throttle position feedback to control the current required for the H-bridge that drives the

DC motor. As shown in Figure 1, the sensor input data from ECU-2 is packed in Message 0x500 and is sent to ECU-1 via CAN bus. The ECU-1 receives the message and the error between the accelerator pedal position signal and throttle position signal is fed to the PID controller. The output of the PID controller is saturated such that the output current is within 1.5A and is transmitted to ECU-2 in Message 0x600 where it is fed to the H-bridge that drives the DC motor to open/close the throttle plate to the desired position.

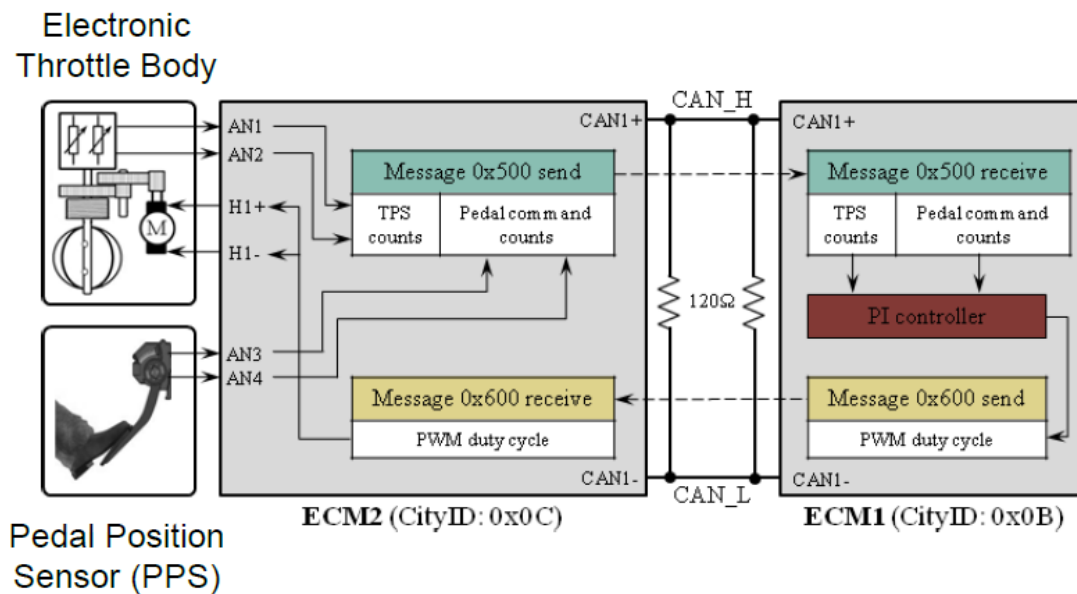


Figure 1: Remote Electronic Throttle Control via CAN

Model Development:

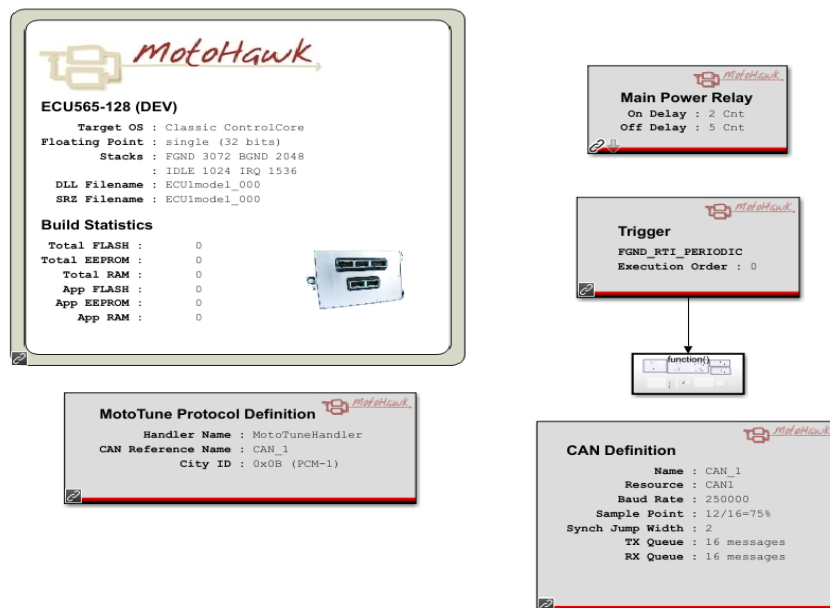


Figure 2: Top Layer of MotoHawk model for ECU-1

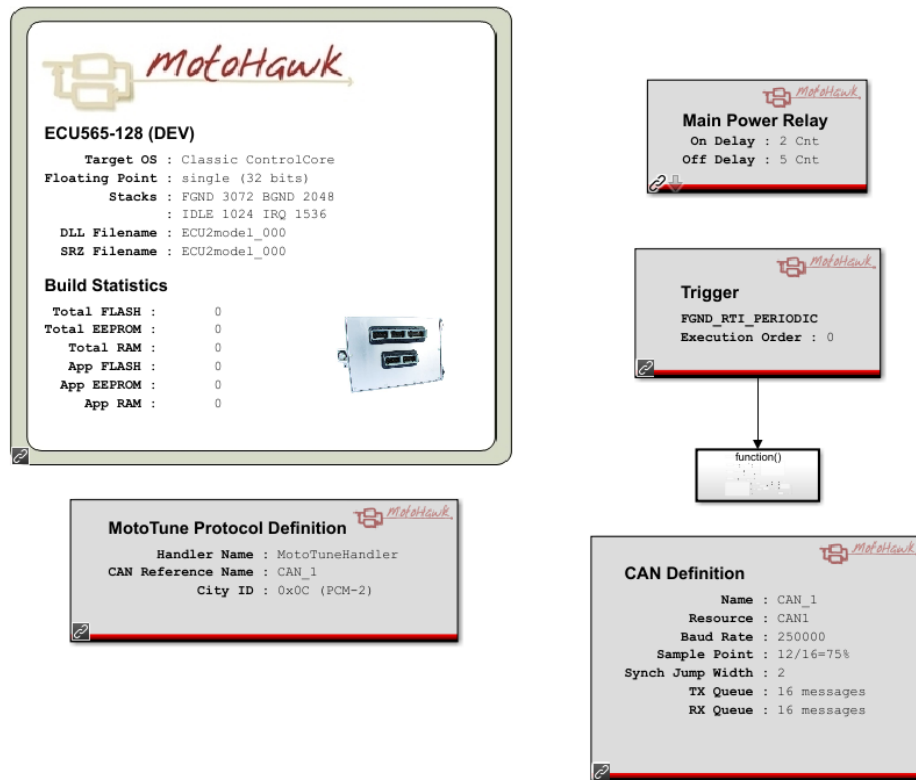


Figure 3: Top Layer of MotoHawk model for ECU-2

Figure 2 and Figure 3 shows the top layer of MotoHawk model for both ECUs that consists of Target definition block, Main power relay power, Trigger, Function block, CAN definition and MotoTune Protocol definition block. A MotoTune Protocol definition block is used to define city ID to identify different software modules. The city ID – 0x0B is used for ECU-1 and city ID – 0x0C for ECU-2. A CAN definition block is used to define baud rate and message buffer size. In this experiment, CAN communication is achieved at 250K baud rate.

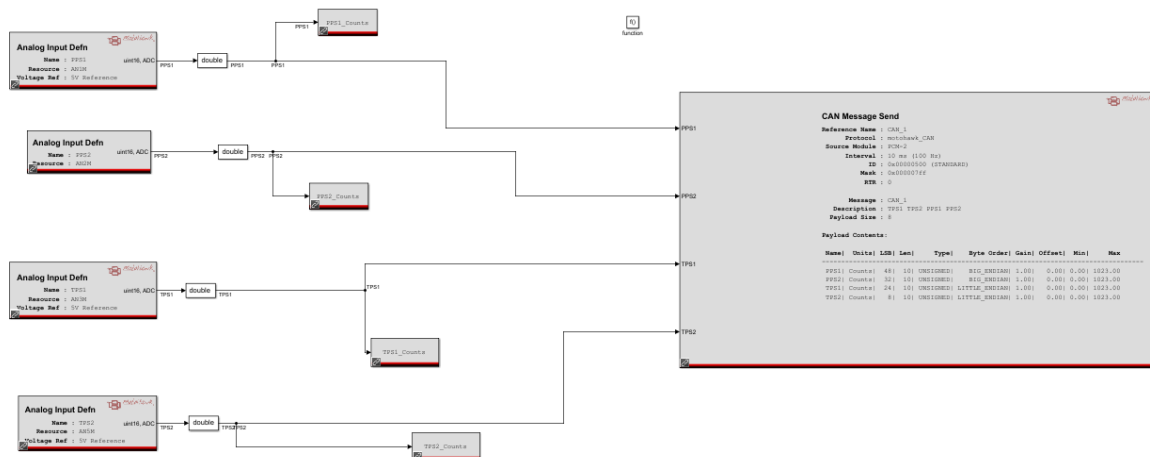


Figure 4: CAN Message Send Block for ECU-2

In ECU-2, four MotoHawk analog input definition blocks are used to read the data from a pair of TPS and PPS sensors. The data read is converted to standard format using Simulink data conversion blocks. An advanced CAN send block with message ID 0x500 is used to transmit data via CAN bus. The parameters for the message IDs are defined using message definition

m-file in MATLAB which consists of ID formatting and payload information like payload size = 8bytes, bit length = 10bits each, byte order for TPS = little endian and for PPS = big endian, payload units = counts, data type = unsigned and start bit as specified.

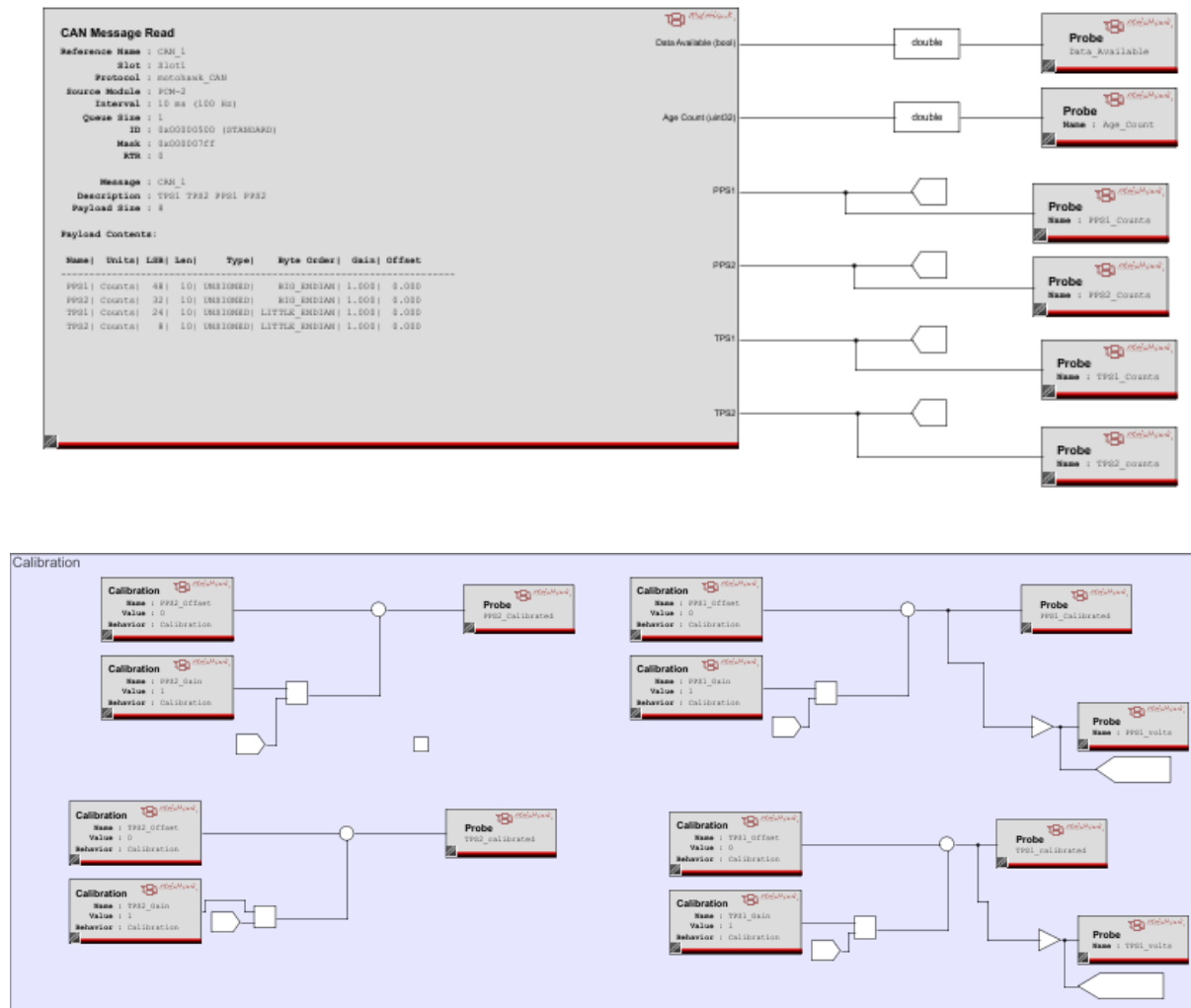


Figure 5: CAN Message Read Block and linear interface model for ECU-1

In ECU-1, an advanced CAN read block with message ID 0x500 is used to read the sensor data. A linear interface model for TPS1 and PPS1 sensor data in counts is designed to match both the data to common level (0-1023) as shown in Figure 5. For this purpose, calibration blocks are used to calibrate the gain and offset values. The calibrated values are converted to voltage using a gain block. The difference between the PPS1 and TPS1 voltages i.e. error between these is given as reference input to the PID controller whose gains are calibrated in real time to obtain required response. The output of the PID controller is saturated to keep the current of the H-bridge within 1.5A. To obtain the saturation limits, a MotoHawk override block is used between PID controller output and CAN send block. In real-time the duty cycle is overridden to obtain the wide-open throttle position and this value is set as saturation upper limit.

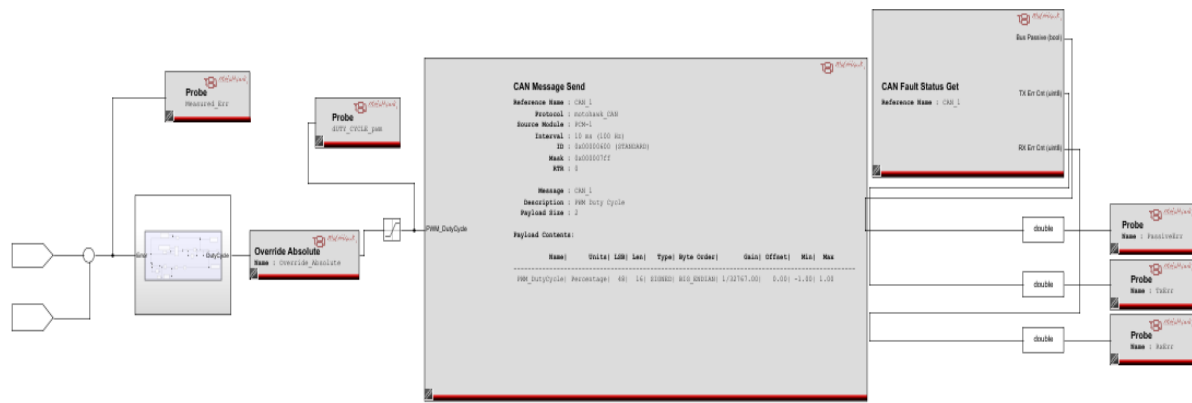


Figure 6: CAN Message Send Block for ECU-1

An advanced CAN send block with message ID 0x600 is used to transmit the controller output i.e. PWM duty cycle to ECU-2 via CAN bus. The message 0x600 consist of PWM duty cycle information whose parameters are defined as – payload size = 2bytes, bit length = 16bits, byte order = big endian, payload units = percentage, data type = signed and start bit = 48.

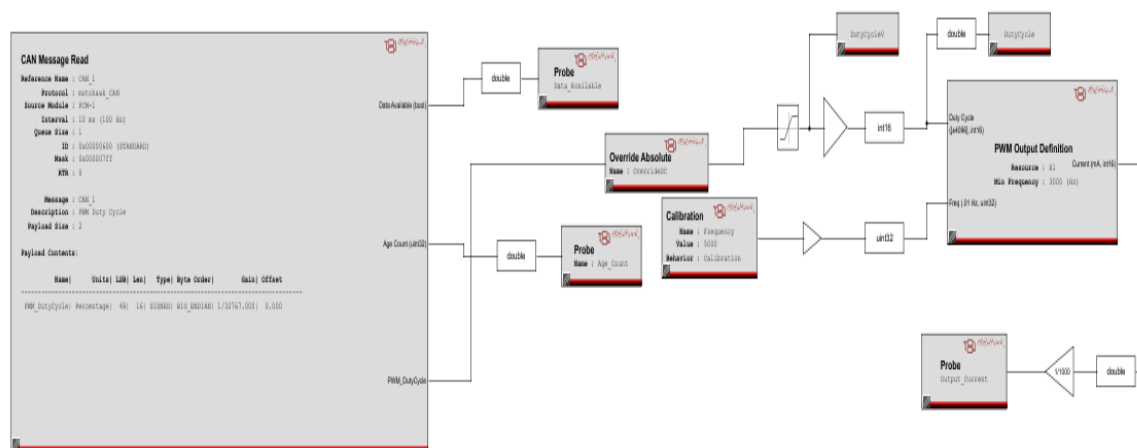


Figure 7: CAN Message Read Block for ECU-2

In ECU-2, the PWM duty cycle is received using an advanced CAN read block with message ID 0x600. This is fed to the MotoHawk PWM Output Definition block which provides signals to the H-bridge of the ECU-2 to control the DC motor. The probe blocks are used to display the values of the variables read i.e. TPS and PPS and to display duty-cycle and PWM output current.

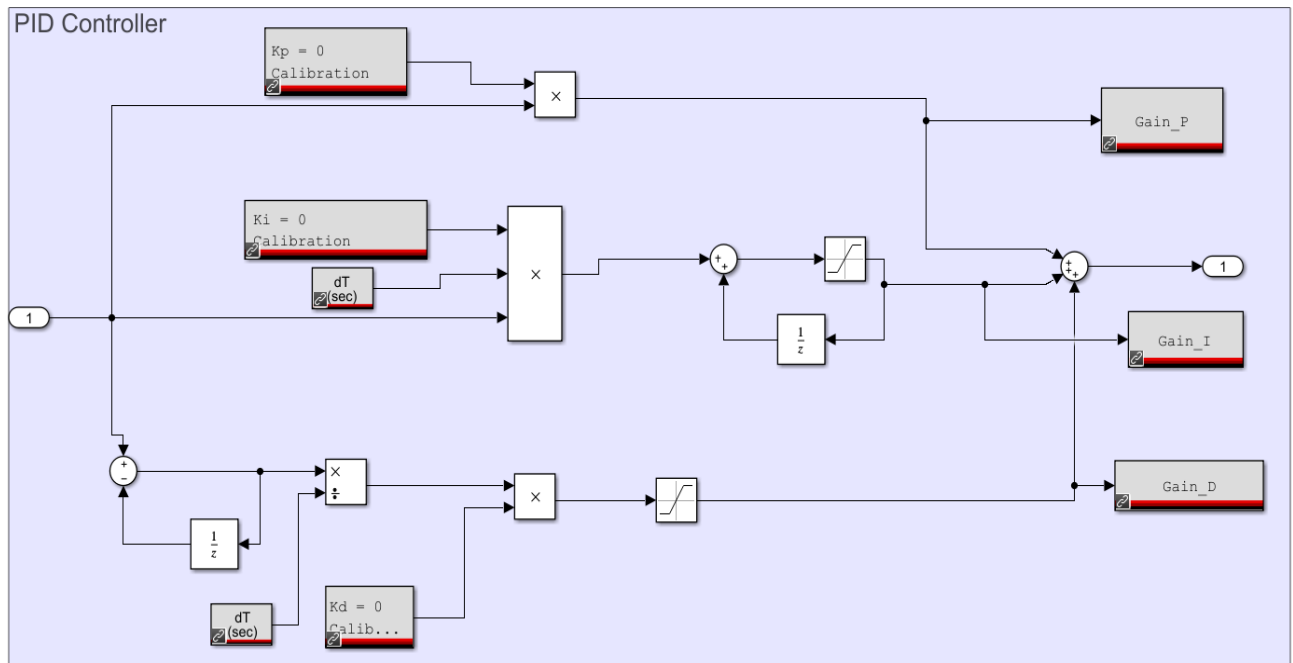


Figure 8: PID Controller

Figure 8 shows the discrete PID controller designed for ETC. The MotoHawk calibration blocks are used to calibrate the P, I and D gains in real-time. For the proportional term, the error signal is multiplied with gain P (Kp). For integral term, a MotoHawk time constant block is multiplied with the error signal and integral gain I (Ki) and then added to delayed signals. To obtain the derivative term, the delayed signal is multiplied with the derivative gain D (Kd) and divided with the time constant. The following discrete time equation is used to implement the PID controller.

$$D(z) = Kp + Ki \left(\frac{zT_s}{z-1} \right) + Kd \left(\frac{z-1}{T_s z} \right)$$

where, K_p = proportional gain

$K_i = \text{integral gain}$

Kd = derivative gain

CAN Communication:

Data Communication	ECU-2 → ECU-1	ECU-1 → ECU-2
Baud Rate	250kbps	250kbps
Bus Channel	CAN-1	CAN-1
Resource	CAN-1	CAN-1
City ID	ECU-2 – 0x0C	ECU-1 – 0x0B

City ID Access Level	4	4
ID - Type	Standard	Standard
Message ID	0x500	0x600
Message ID Mask	0x7FF	0x7FF
Data Field	TPS and PPS (ADC counts)	PWM Duty Cycle (%)
Repeating Rate	10ms	10ms
Payload Size	8 bytes	2 bytes
Bit length	10 bits each	16 bits
Data Type	Unsigned	Signed
Byte Order	TPS – Little Endian PPS- Big Endian	PWM Duty Cycle – Big Endian

Results and Validation:

The limits of the PWM duty cycle were calibrated using an override block at the output of the controller in ECU-1. With an override value of zero, the TPS 1 and PPS 1 counts were displayed in MotoTune of ECU-2 and compared with received data in ECU-1. Figure 9 shows the counts of TPS 1 and PPS 1 as observed in the MotoTune display of ECU-2 and Figure 10 shows the counts of TPS 1 and PPS 1 as observed in the MotoTune display of ECU-1.

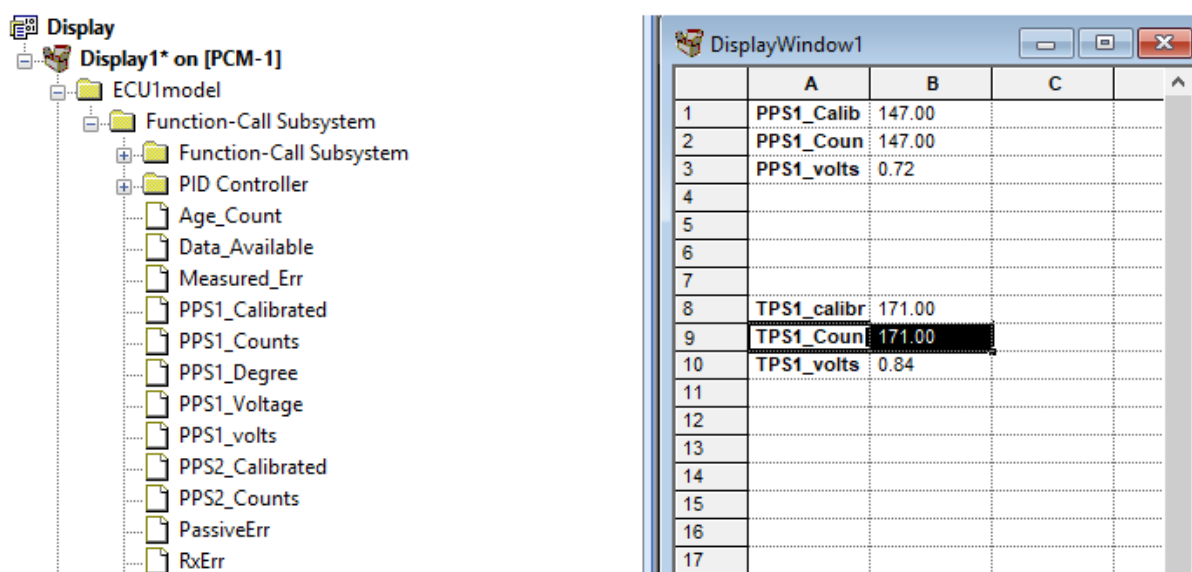


Figure 9: Display window for PPS1 and TPS1 counts on PCM-1 at idle position

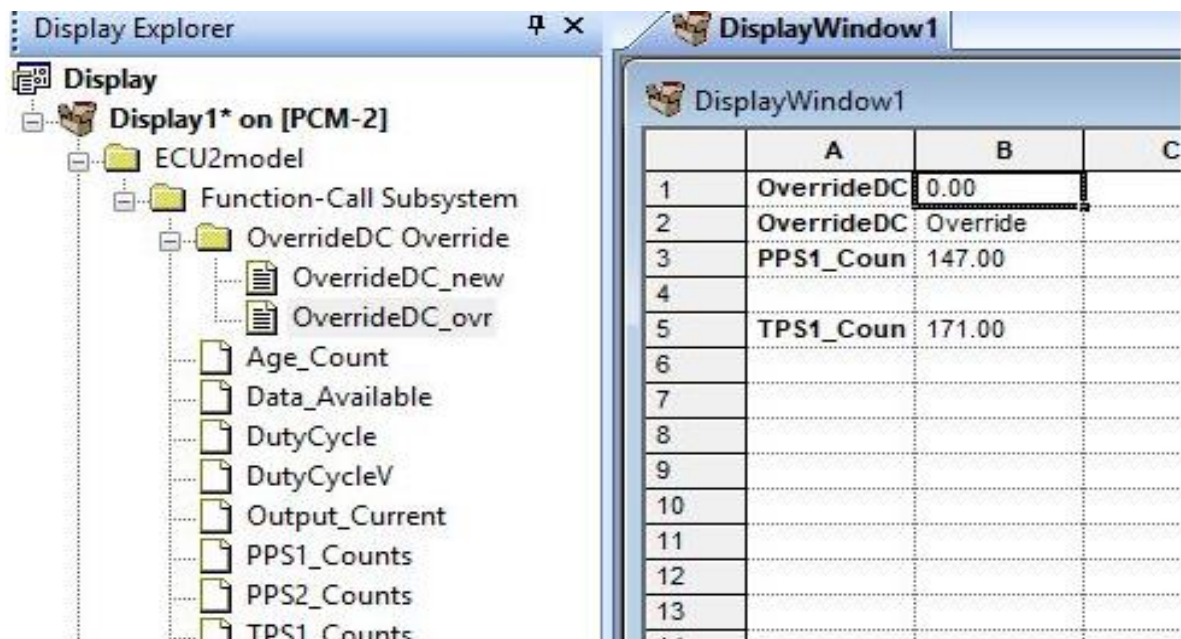


Figure 10: Display window for PPS1 and TPS1 counts on PCM-2 at idle position

The upper limit of the PWM duty cycle was calibrated using an override value of 0.33 for wide-open throttle. The TPS 1 and PPS 1 counts were displayed on MotoTune. Figure 9 shows the display window of TPS 1 and PPS 1 as observed in the MotoTune display of PCM 1 and Figure 10 shows the display window of TPS 1 and PPS1 as observed in display of PCM 2.

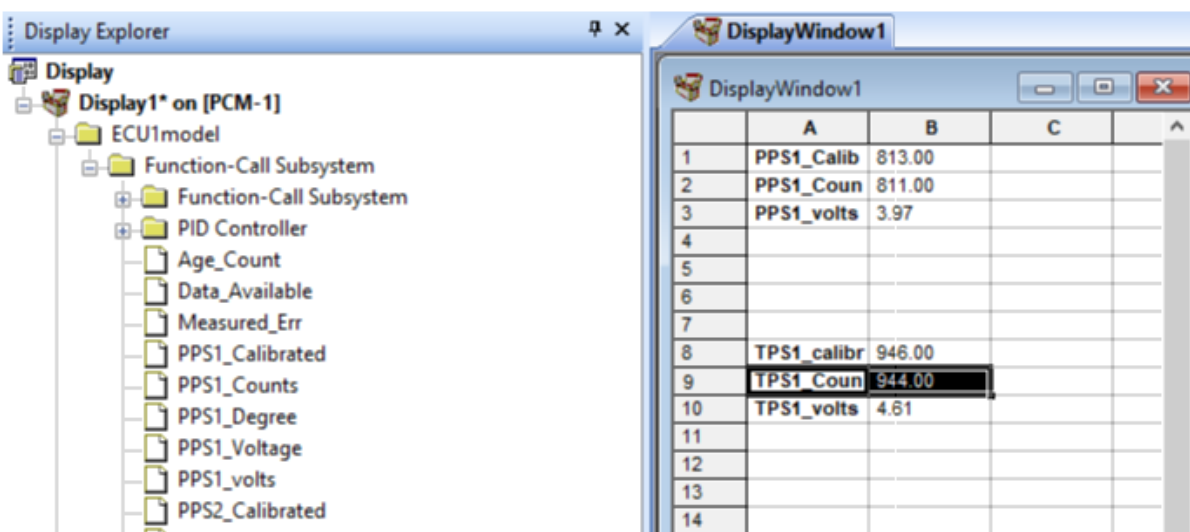


Figure 11: Calibration window for PPS1 and TPS1 counts on PCM-1 at WOP position

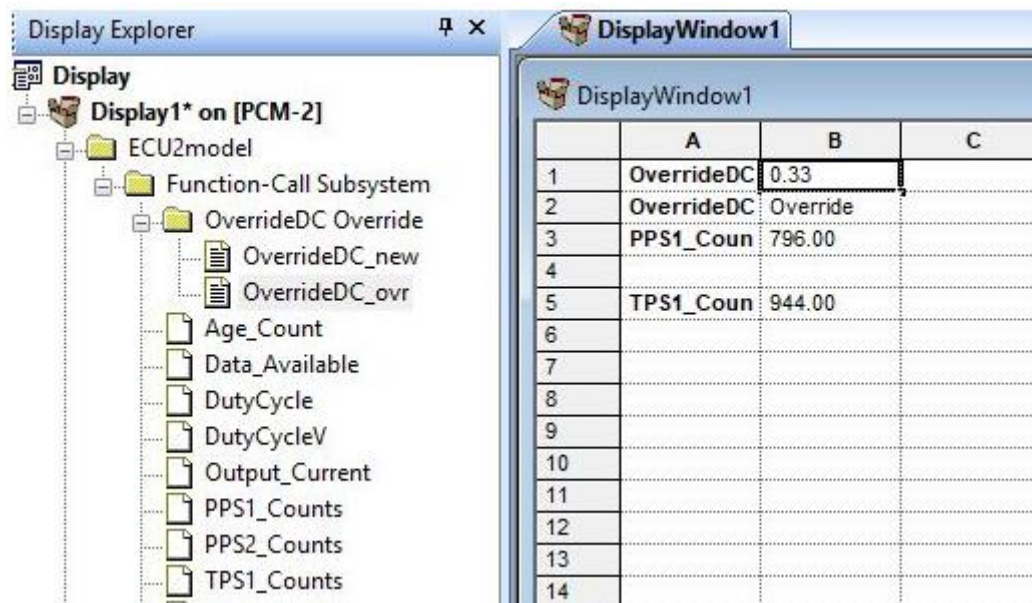


Figure 12: Calibration window for PPS1 and TPS1 counts on PCM-2at WOP position

Message Calibration:

There are two methods to calibrate the payload data,

In method-1 the data is calibrated in real-time using the MotoHawk calibration block for gain and offset. Each sensor data is read for idle and full throttle position, these values are used to obtain the desired reading using the relation below,

$$\text{desired value} = \text{read value} * \text{gain} + \text{offset}$$

The computed gain and offset are defined in the calibration window. It is observed that the data read in CANKing output window at ECU-2 is same as the data received at ECU-1. The calibrated values on ECU-1 shows that the calibration is done after the data communication.

In method-2 the above calibrated values of gain and offset are defined in the m-files of both the ECUs and the values of gain and offset in calibration window are 1 and 0 respectively. In this case, the CAN messages are communicated with gain and offset.

Table 1: Calibration for PPS 1 and TPS 1 counts

Position	TPS1 (counts)	PPS1 (counts)
Override = 0 (Idle position)	171	147
Override = 0.33 (WOT)	944	796

To obtain linear relationship between TPS and PPS counts, the gain and offset are calculated by mapping ADC counts to 0 at idle position and 1023 at WOP.

For TPS,

$$\text{Desired count} = \text{actual count} * \text{gain} + \text{offset}$$

$$0 = 171 * m + c$$

$$1023 = 944 * m + c$$

By solving above equations, we get, $m = 1.323$ and $c = -226.304$

For PPS,

$$0 = 147 * m + c$$

$$1023 = 796 * m + c$$

By solving above equations, we get, $m = 1.576$ and $c = -231.71$

Per the method-1, these values are defined in calibration window of MotoTune, Figure 13 shows the calibrated results for idle position and Figure 14 shows the calibrated values for WOP.

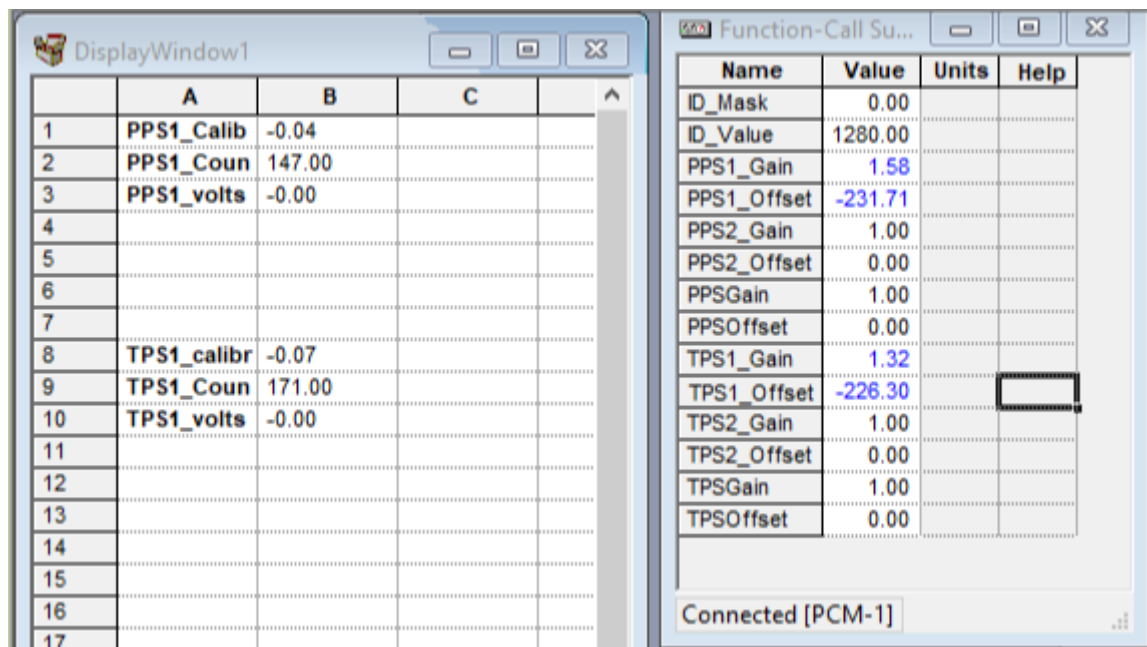


Figure 13: Calibration window for linear interfacing at idle position

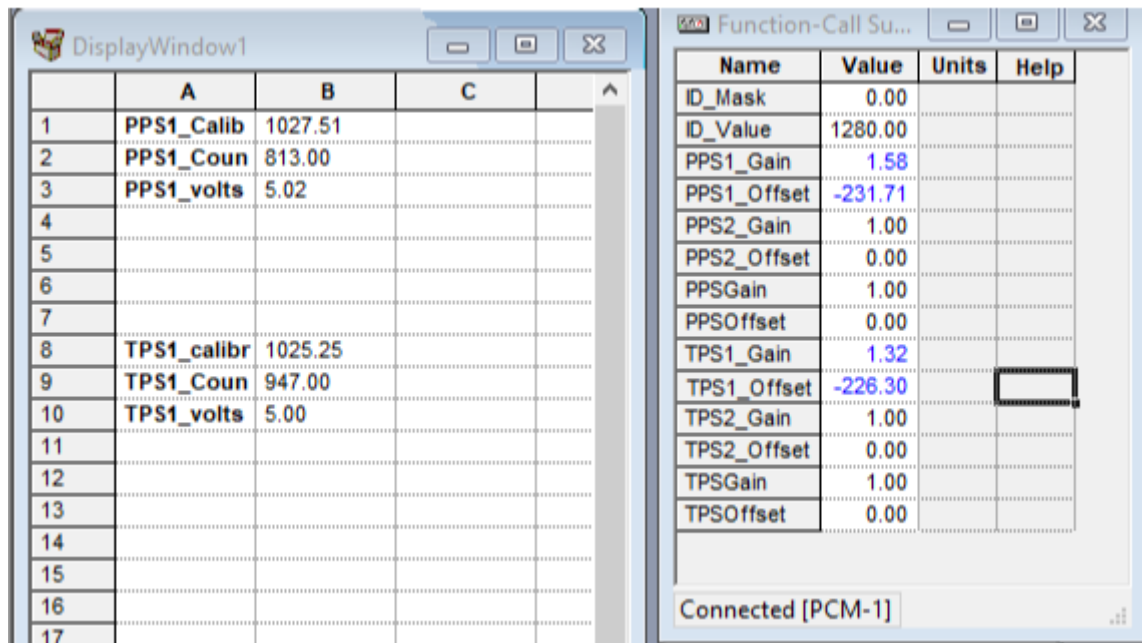


Figure 14: Calibration window for linear interfacing at WOP

After calibrating the TPS and PPS counts, the PID controller gains were tuned to obtain an optimal performance. The optimal controller gains were as follows - $K_p = 0.8$, $K_i = 1$ and $K_d = 0$. The derivative gain was kept zero to avoid system noise. Figure 14, Figure 15 and Figure 16 shows the calibration, display window and charts of throttle position signals to accelerator pedal command signals. It can be observed from the figures below, the throttle signal follows the commanded pedal signal.

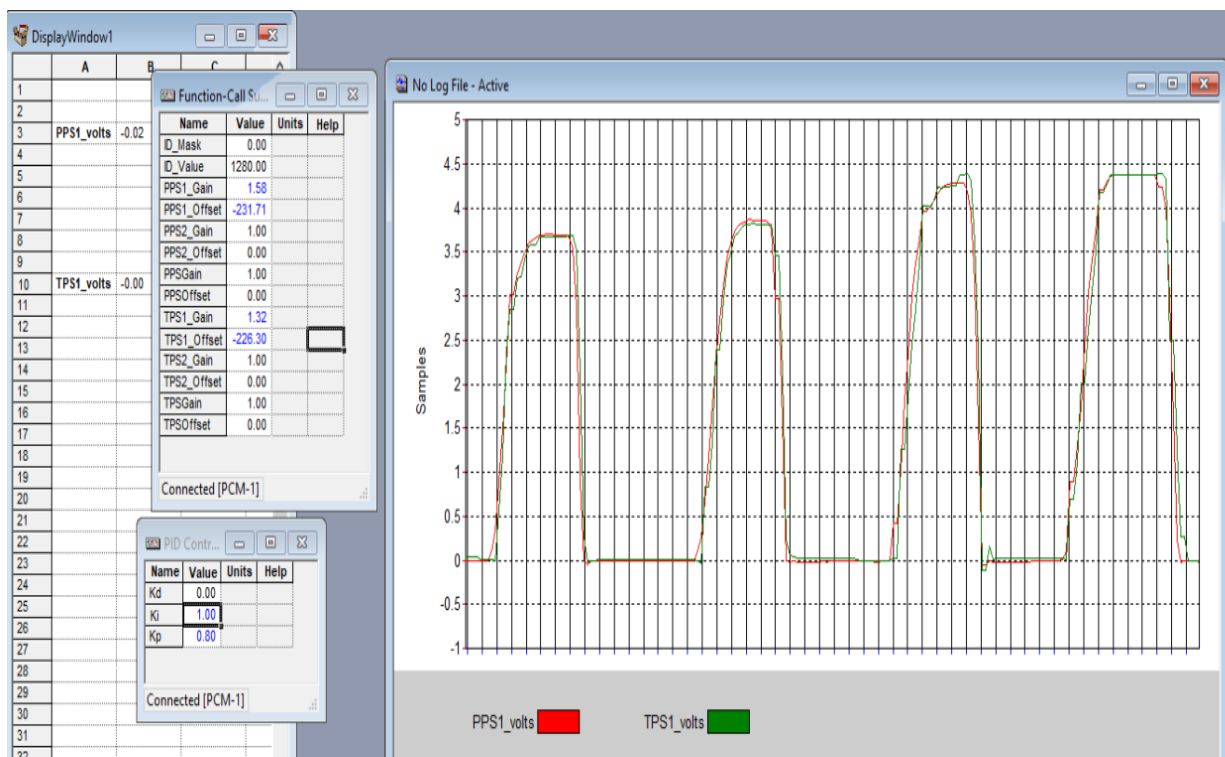


Figure 15: TPS1 and PPS1 response 1

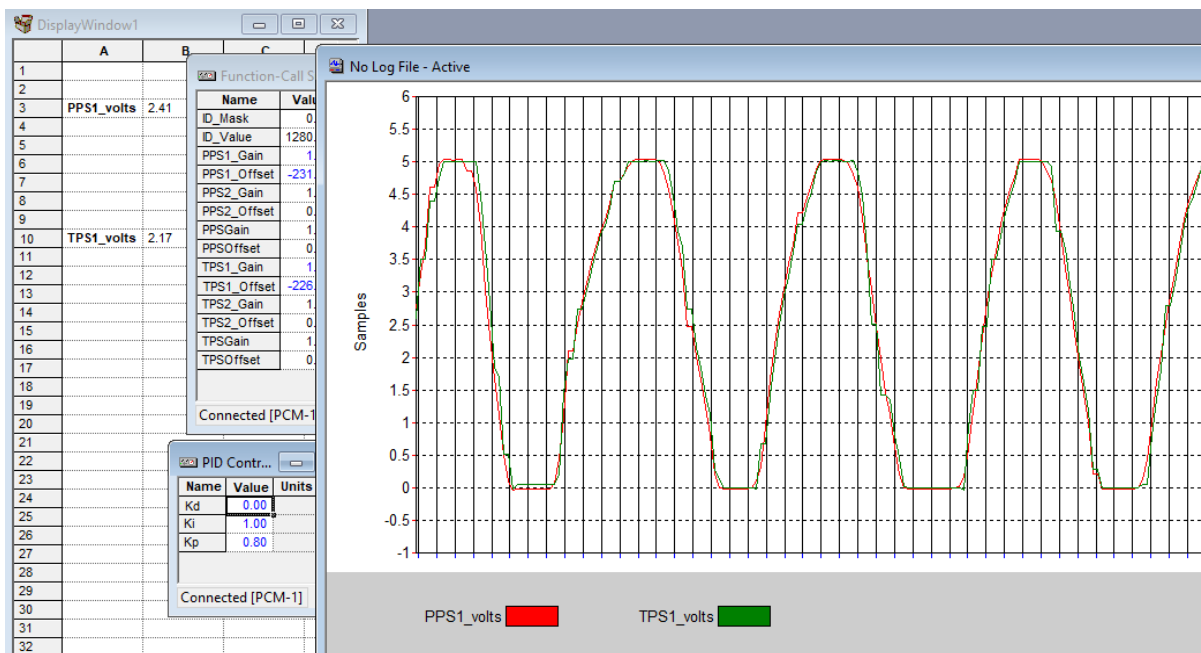


Figure 16: TPS1 and PPS1 response 2

Output Window

Chn	Identifier	Flg	DLC	D0...	1...	2...	3...	4...	5...	6...	D7	Time	Dir
0	00000500		8	00	93	00	42	AD	00	2F	03	26080.000850	R
0	00000600		2	00	00							26079.999470	R

Figure 17: CANKing Output Window for idle position

DisplayWindow1

	A	B	C
1	Override_A	0.33	
2	Override_A	Pass-Thr	
3			
4			
5	Age_Count	4114.00	
6	Data_Availa	0.00	
7			
8	PPS1_Coun	147.00	
9	PPS1_volts	-0.00	
10			
11			
12			
13	dUTY_CYCL	0.00	
14			
15	TPS1_Coun	173.00	
16	TPS1_Volts	0.85	
17			

Figure 18: Display window for idle position

The CANKing output window shows the CAN messages having IDs and corresponding payload size for PCM1 and PCM2. The message ID 0x500 with payload size of 8bytes corresponds to the message sent from ECU-2 to ECU-1 displaying TPS and PPS sensor data. As defined in the m-file:

- The PPS1 counts are displayed in D0 and D1 data fields (bit 48 to 63) and stored in Big Endian format. Therefore, the start bit for PPS1 is 48th bit.
- The PPS2 counts are displayed in D2 and D3 data fields (bit 32 to 47) and stored in Big Endian format. Therefore, the start bit for PPS2 is 32nd bit.
- The TPS1 counts are displayed in D4 and D5 data fields (bit 16 to 31) and stored in Little Endian format. Therefore, the start bit for TPS1 is 24th bit.
- The TPS2 counts are displayed in D6 and D7 data fields (bit 0 to 15) and stored in Little Endian format. Therefore, the start bit for TPS2 is 8th bit.

The message ID 0x600 with payload size of 2bytes corresponds to the message sent from ECU-1 to ECU-2 displaying PWM duty cycle. As defined in the m-file, the PWM duty cycle is displayed in D0 and D1 data fields (bit 48 to 63) and stored in Big Endian format. Hence, the start bit for PWM duty cycle will be 48th bit.

For Big Endian, the LSB is stored in the higher data byte and the MSB is stored in the lower data byte. Hence, for PPS1 the LSB=0x93 and MSB=0x00.

⇒ So, the obtained value is 0x0093.

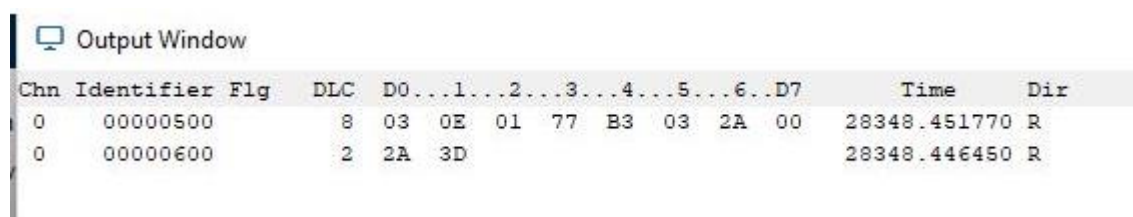
⇒ This corresponds to decimal value of 147 which matches with the PPS1 counts as shown in the display window in Figure 18.

For PWM duty cycle, the LSB= 0x00 and MSB= 0x00 which corresponds to decimal value of 0 in decimal. This matches with duty cycle shown in the display window in Figure 17.

For Little Endian, the LSB is stored in the lower data byte and the MSB is stored in the higher data byte. Hence, for TPS1 the LSB=0xAD and MSB=0x00.

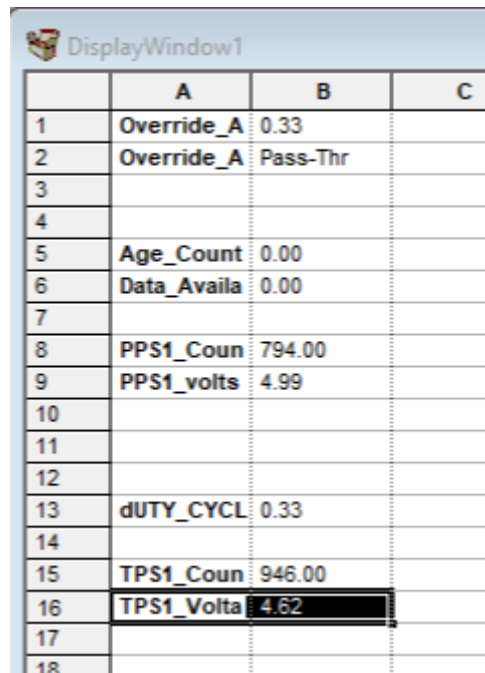
⇒ So, the obtained value is 0x00AD.

⇒ This corresponds to decimal value of 173 which matches with the TPS1 counts as shown in the display window in Figure18.



Chn	Identifier	Flg	DLC	D0...	1...	2...	3...	4...	5...	6...	D7	Time	Dir
0	00000500		8	03	0E	01	77	B3	03	2A	00	28348.451770	R
0	00000600		2	2A	3D							28348.446450	R

Figure 19: CANKing Output Window for WOP



	A	B	C
1	Override_A	0.33	
2	Override_A	Pass-Thr	
3			
4			
5	Age_Count	0.00	
6	Data_Availa	0.00	
7			
8	PPS1_Coun	794.00	
9	PPS1_volts	4.99	
10			
11			
12			
13	dUTY_CYCL	0.33	
14			
15	TPS1_Coun	946.00	
16	TPS1_Volts	4.62	
17			
18			

Figure 20: Display window for WOP

Figure19 shows CANKing output window and Figure 20shows display window at WOP.

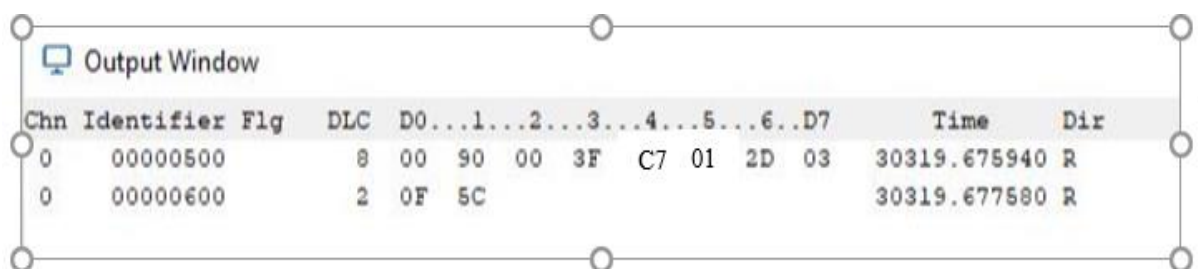
For PPS1, the LSB=0x0E and MSB=0x03.

- ⇒ So, the obtained value is 0x030E.
- ⇒ This corresponds to decimal value of 782 which matches with the PPS1 counts as shown in the display window in Figure 20.

For PWM duty cycle, the LSB= 0x3D and MSB= 0x2A which corresponds to decimal value of 10813 in decimal. Gain for PWM duty cycle is defined in the m-file. So, to obtain the actual value, the obtained value is to be multiplied with gain of 1/32767. Hence, the actual value of PWM duty cycle is $10813 * \frac{1}{32767} = 0.33$. This matches with duty cycle shown in the display window in Figure 20.

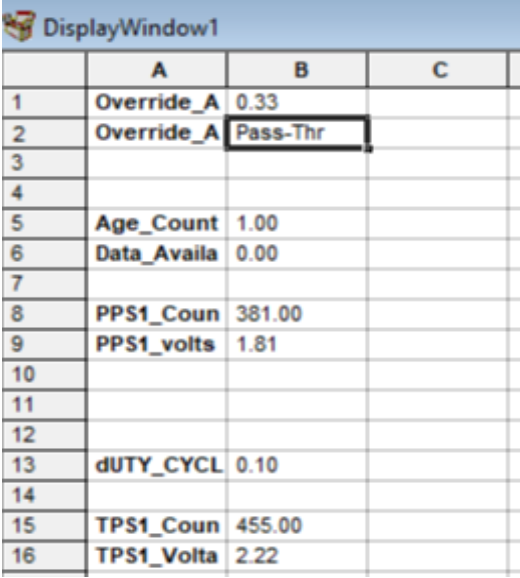
For TPS1, the LSB=0xB3 and MSB=0x03.

- ⇒ So, the obtained value is 0x03B3.
- ⇒ This corresponds to decimal value of 947 which matches with the TPS1 counts as shown in the display window in Figure 20.



Chn	Identifier	Flg	DLC	D0...	1...	2...	3...	4...	5...	6...	D7	Time	Dir
0	00000500		8	00	90	00	3F	C7	01	2D	03	30319.675940	R
0	00000600		2	0F	5C							30319.677580	R

Figure 21: CANKing Output Window for duty cycle =0.1



	A	B	C
1	Override_A	0.33	
2	Override_A	Pass-Thr	
3			
4			
5	Age_Count	1.00	
6	Data_Availa	0.00	
7			
8	PPS1_Coun	381.00	
9	PPS1_volts	1.81	
10			
11			
12			
13	dUTY_CYCL	0.10	
14			
15	TPS1_Coun	455.00	
16	TPS1_Volta	2.22	

Figure 22: Display window at duty cycle = 0.1

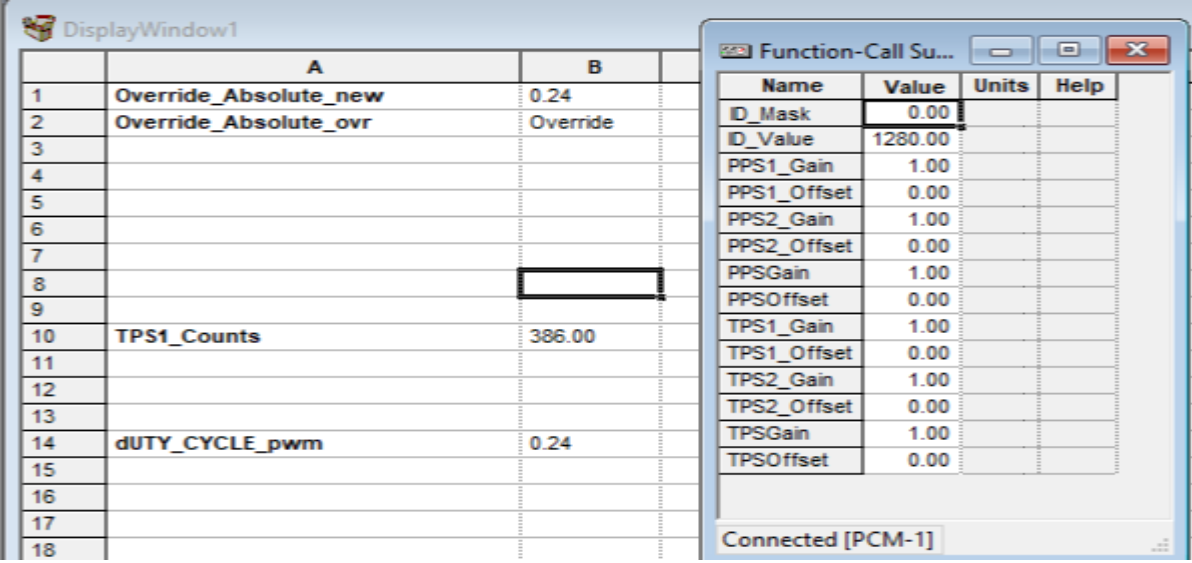
Figure 21 shows CANKing output window and Figure 22 shows display window at duty cycle=0.1.

For PWM duty cycle, the LSB= 0x5C and MSB= 0x0F which corresponds to decimal value of 3932 in decimal. Gain for PWM duty cycle is defined in the m-file. So, to obtain the actual value, the obtained value is to be multiplied with gain of 1/32767. Hence, the actual value of PWM duty cycle is $3932 * \frac{1}{32767} = 0.11$. This matches with duty cycle shown in the display window in Figure 22.

For TPS1, the LSB=0xC7 and MSB=0x01.

⇒ So, the obtained value is 0x01C7.

This corresponds to decimal value of 455 which matches with the TPS1 counts as shown in the display window in Figure 23.

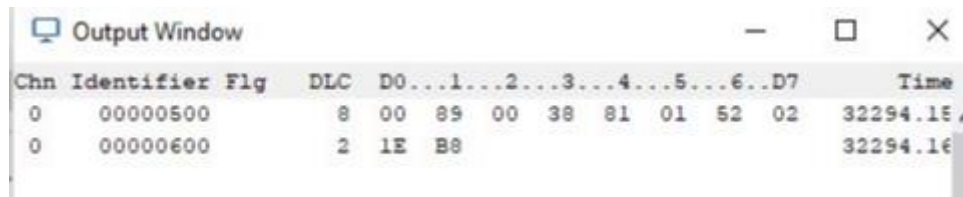


	A	B
1	Override_Absolute_new	0.24
2	Override_Absolute_ovr	Override
3		
4		
5		
6		
7		
8		
9		
10	TPS1_Counts	386.00
11		
12		
13		
14	dUTY_CYCLE_pwm	0.24
15		
16		
17		
18		

Name	Value	Units	Help
ID_Mask	0.00		
ID_Value	1280.00		
PPS1_Gain	1.00		
PPS1_Offset	0.00		
PPS2_Gain	1.00		
PPS2_Offset	0.00		
PPSGain	1.00		
PPSOffset	0.00		
TPS1_Gain	1.00		
TPS1_Offset	0.00		
TPS2_Gain	1.00		
TPS2_Offset	0.00		
TPSGain	1.00		
TPSOffset	0.00		

Connected [PCM-1]

Figure 23: Display window for TPS calibrated value as per m-file



Chn	Identifier	Flg	DLC	D0	1	2	3	4	5	6	D7	Time
0	00000500		8	00	89	00	38	81	01	52	02	32294.15
0	00000600		2	1E	B8							32294.16

Figure 24: CANKing window for TPS calibrated value as per m-file

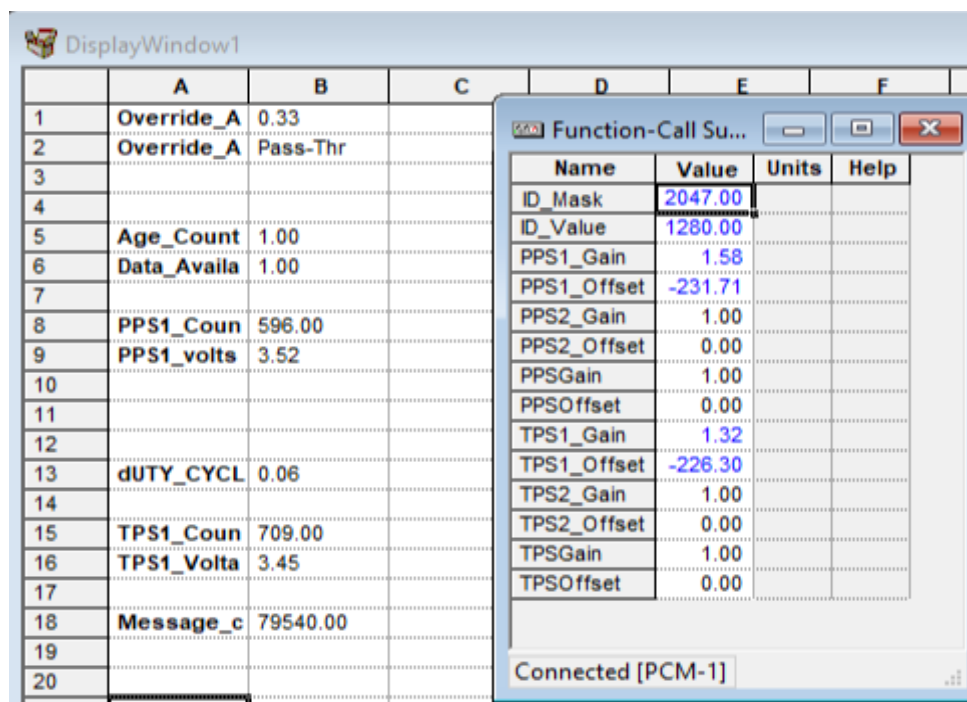
For TPS1, the LSB=0x01 and MSB=0x81.

⇒ So, the obtained value is 0x0181.

This corresponds to decimal value of 385 which matches with the TPS1 counts as shown in the display window in Figure 23. As we can see in the display window, the gain=1 and the offset is 0. But still this value of TPS1 count is calibrated. This is because we have defined the gain and offset in the m-file.

Message ID and ID mask manipulation in real-time:

A MotoHawk CAN Receive slot properties block is used to change the value of message ID and ID mask in real-time. A MotoHawk trigger slot block is used which triggers a function call whenever a message is received. The function that is to be triggered upon reception of message is the counter. To test this function, the message ID and ID mask is set to 0x500 and 0x7FF respectively as shown in Figure 25. The Age count, message count and data available is observed in display window of MotoTune. As shown in Figure 25, the incoming message matches the message ID based on the defined ID mask, therefore message count keeps increasing and data available is 1 with age count as 0. The ID mask is set to 0x7FF which means that all the bits of incoming message ID must match the defined message ID.



	A	B	C	D	E	F
1	Override_A	0.33				
2	Override_A	Pass-Thr				
3						
4						
5	Age_Count	1.00				
6	Data_Availa	1.00				
7						
8	PPS1_Coun	596.00				
9	PPS1_volts	3.52				
10						
11						
12						
13	dUTY_CYCL	0.06				
14						
15	TPS1_Coun	709.00				
16	TPS1_Volta	3.45				
17						
18	Message_c	79540.00				
19						
20						

Name	Value	Units	Help
ID_Mask	2047.00		
ID_Value	1280.00		
PPS1_Gain	1.58		
PPS1_Offset	-231.71		
PPS2_Gain	1.00		
PPS2_Offset	0.00		
PPSGain	1.00		
PPSOffset	0.00		
TPS1_Gain	1.32		
TPS1_Offset	-226.30		
TPS2_Gain	1.00		
TPS2_Offset	0.00		
TPSGain	1.00		
TPSOffset	0.00		

Connected [PCM-1]

Figure 25: Received ID for defined ID mask

However, when message ID is changed, say 0x700 shown in Figure 26 then the message ID defined in m-file (ID filter) does not match the message ID of the incoming message. Therefore, the age count increases rapidly with data available = 0.

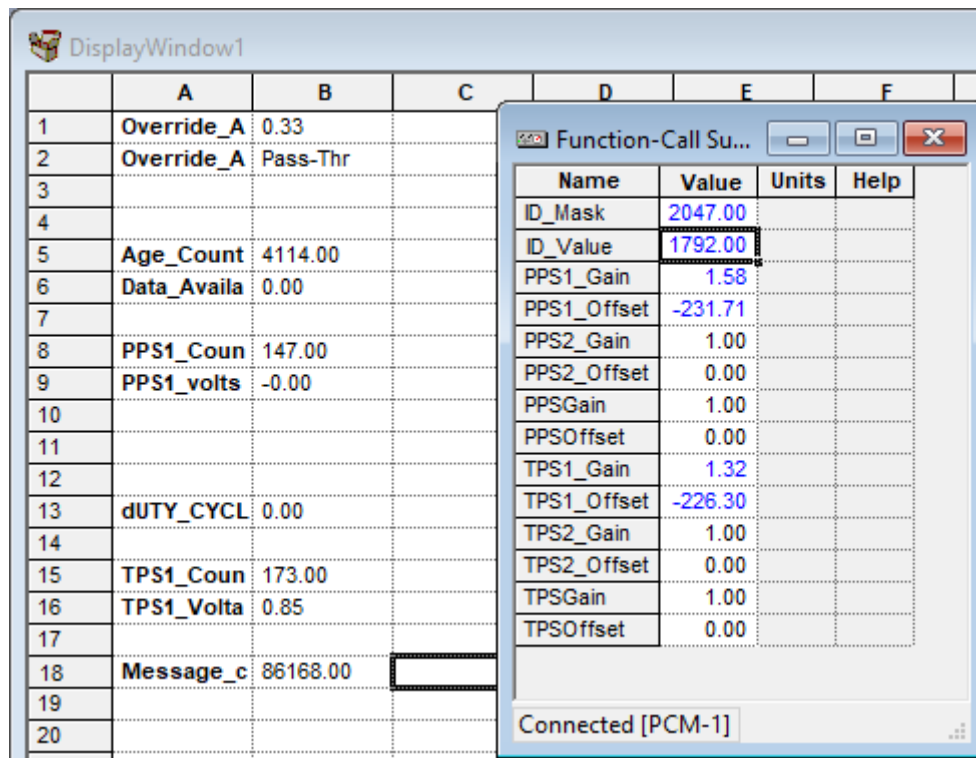


Figure 26: Results for Declined Message ID

To receive the message ID 0x700, the ID mask is modified according to the filter 0x500 and incoming message ID (0x700) i.e.,

ID Filter	0x500	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1280
ID Mask	0x7FF	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	2047
Message ID	0x700	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1792
New ID mask		1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	-1781

The new ID mask value is defined in the calibration window. Now, the incoming ID matches the filter and mask value. There the message ID = 0x700 is received by the ECU and the age count is reset to 0. The data available parameter is changed to 1 indicating that the data is available at the CAN bus.

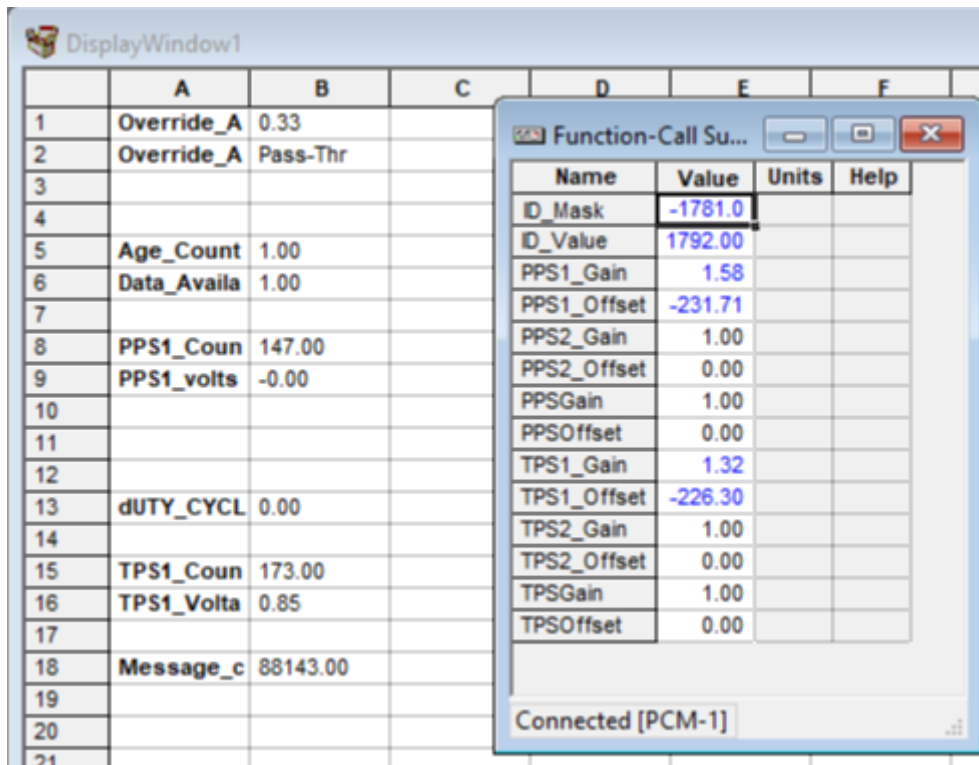


Figure 27: Modified ID mask to receive declined message ID

Message Definition Files:

To control the electronic throttle remotely via CAN bus two messages are used, one to transmit the sensor data like TPS and PPS and the other to receive the PWM duty cycle. The m- files contain the information about the message module and communication channel used for communication. Also, it contains the information about message id format, ID filter, ID mask, payload size and repeating rate of the message ID.

For message 0x500, the m-file contains the TPS and PPS message parameters such as the units, bit length, byte order, start bit, data type, scale and offset. While using the method-2 for calibration, the scale and offset are defined in this file. For method-1, the scale and offset are 1 and 0 respectively. Figure 28 shows the m-file described for Message ID = 0x500 with method-1 calibration. Figure 29 shows the m-file for Message ID = 0x500 using method-2 calibration.

For message 0x600, the m-file defines the PWM duty cycle parameters such as unit, start bit, bit length, byte order, data type, scale and offset. The scale for PWM duty cycle is (1/32767). Figure 30 shows the m-file for Message ID = 0x600.

```

msg.name                = 'CAN_1'
msg.description          = 'TPS1 TPS2 FPS1 FPS2'
msg.protocol             = 'motehawk_CAN'
msg.module               = 'ECM-2'
msg.channel              = 1

msg.idext                = 'STANDARD'
msg.id                   = hex2dec('0600')
msg.idmask               = hex2dec('07ff')
msg.payload_size         = 8
msg.interval             = 10

msg.fields{1}.name       = 'FPS1'
msg.fields{1}.units      = 'Counts'
msg.fields{1}.start_bit  = 48
msg.fields{1}.bit_length = 10
msg.fields{1}.byte_order = 'BIG_ENDIAN'
msg.fields{1}.data_type  = 'UNSIGNED'
msg.fields{1}.scale      = 1
msg.fields{1}.offset     = 0

msg.fields{2}.name       = 'FPS2'
msg.fields{2}.units      = 'Counts'
msg.fields{2}.start_bit  = 32
msg.fields{2}.bit_length = 10
msg.fields{2}.byte_order = 'BIG_ENDIAN'
msg.fields{2}.data_type  = 'UNSIGNED'
msg.fields{2}.scale      = 1
msg.fields{2}.offset     = 0

msg.fields{3}.name       = 'TPS1'
msg.fields{3}.units      = 'Counts'
msg.fields{3}.start_bit  = 24
msg.fields{3}.bit_length = 10
msg.fields{3}.byte_order = 'LITTLE_ENDIAN'
msg.fields{3}.data_type  = 'UNSIGNED'
msg.fields{3}.scale      = 1
msg.fields{3}.offset     = 0

msg.fields{4}.name       = 'TPS2'
msg.fields{4}.units      = 'Counts'
msg.fields{4}.start_bit  = 8
msg.fields{4}.bit_length = 10
msg.fields{4}.byte_order = 'LITTLE_ENDIAN'
msg.fields{4}.data_type  = 'UNSIGNED'
msg.fields{4}.scale      = 1
msg.fields{4}.offset     = 0

```

Figure 28 : Matlab m-file for Message ID = 0x500 using Method-1 Calibration

```

msg.name = 'CAN_1';
msg.description = 'TPS1 TPS2 PPS1 PPS2';
msg.protocol = 'motehawk_CAN';
msg.module = 'PCM-2';
msg.channel = 1;

msg.idext = 'STANDARD';
msg.id = hex2dec('0500');
msg.idmask = hex2dec('07ff');
msg.payload_size = 5;
msg.interval = 10;

msg.fields{1}.name = 'PPS1';
msg.fields{1}.units = 'Counts';
msg.fields{1}.start_bit = 48;
msg.fields{1}.bit_length = 10;
msg.fields{1}.byte_order = 'BIG_ENDIAN';
msg.fields{1}.data_type = 'UNSIGNED';
msg.fields{1}.scale = 1.576;
msg.fields{1}.offset = -231.71;

msg.fields{2}.name = 'PPS2';
msg.fields{2}.units = 'Counts';
msg.fields{2}.start_bit = 32;
msg.fields{2}.bit_length = 10;
msg.fields{2}.byte_order = 'BIG_ENDIAN';
msg.fields{2}.data_type = 'UNSIGNED';
msg.fields{2}.scale = 1;
msg.fields{2}.offset = 0;

msg.fields{3}.name = 'TPS1';
msg.fields{3}.units = 'Counts';
msg.fields{3}.start_bit = 24;
msg.fields{3}.bit_length = 10;
msg.fields{3}.byte_order = 'LITTLE_ENDIAN';
msg.fields{3}.data_type = 'UNSIGNED';
msg.fields{3}.scale = 1.323;
msg.fields{3}.offset = -226.304;

msg.fields{4}.name = 'TPS2';
msg.fields{4}.units = 'Counts';
msg.fields{4}.start_bit = 8;
msg.fields{4}.bit_length = 10;
msg.fields{4}.byte_order = 'LITTLE_ENDIAN';
msg.fields{4}.data_type = 'UNSIGNED';
msg.fields{4}.scale = 1;
msg.fields{4}.offset = 0;

```

Figure 29: Matlab m-file for Message ID = 0x500 using Method-2 Calibration

```

msg.name = 'CAN_1';
msg.description = 'PWM Duty Cycle';
msg.protocol = 'motehawk_CAN';
msg.module = 'PCM-1';
msg.channel = 1;

msg.idext = 'STANDARD';
msg.id = hex2dec('0600');
msg.idmask = hex2dec('07ff');
msg.payload_size = 2;
msg.interval = 10;

msg.fields{1}.name = 'PWM_DutyCycle';
msg.fields{1}.units = 'Percentage';
msg.fields{1}.start_bit = 48;
msg.fields{1}.bit_length = 16;
msg.fields{1}.byte_order = 'BIG_ENDIAN';
msg.fields{1}.data_type = 'SIGNED';
msg.fields{1}.scale = 1/32767;
msg.fields{1}.offset = 0;

```

Figure 30: Matlab m-file for Message ID = 0x600

Additional Feature:

A CAN fault status get block is used to detect the fault in the CAN communication. This block reports the current fault status of the referenced CAN definition. Three fault status are reported by the block. The passive bus error is detected when there is mismatch of the baud rate, bus is improperly terminated or by any hardware failure. The transmit error is reported when module is unable to communicate transmitted messages and receive error is reported when module is unable to receive the messages. In this experiment, the feature was tested for bus fault status. The baud rate of one of the ECU was changed to 150Kbps. There was a conflicting baud-rate between the sender and receiver. Hence, an error frame is reported showing the hardware failure of the bus. Figure 31 shows the Error Frame received when baud rate was mismatched.

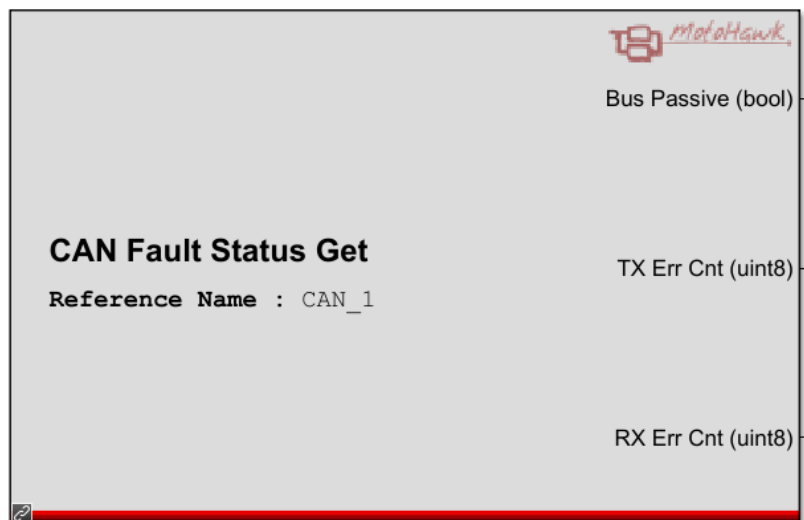


Figure 31: CAN Fault Status block

Chn	Identifier	Flg	DLC	D0...1...2...3...4...5...6..D7	Time
0	00000000			ErrorFrame	33039.73
0	00000500		8	00 93 00 42 AC 00 30 03	33087.85
0	00000600		2	00 00	32706.50

Figure 32: Output window showing hardware failure

Discussion:

We've understood how CAN stores and transmits data for our electronic throttle control body using two ECUs. This remote control of ECU method has its own advantages and disadvantages over the ETC method performed in Lab 4.

Advantages-

- It gives high data transmission rate as compared to normal ETC control which means that we can have a faster response when giving accelerator pedal commands for the opening of our throttle.
- Reduced wiring as compared to the lab 4 experiment.
- It has more amount of error detection capabilities like bit error, transmission and receiving error, etc. whereas in Lab 4 we had only one type of error between TPS1 and TPS2 signals and PPS1 and PPS2 signals.
- Supports auto-retransmission of lost messages

Disadvantages-

- Communication protocols are more complex as they involve use of multiple parameters like message ID, ID mask, start bit, etc.
- There is an undesirable interaction between the nodes which causes an error in the values as compared to the previous lab experiment
- Higher gains can cause the system to respond with a lot of disturbance

Conclusion

Basic understanding of CAN communication was gained in this experiment. We realized how data is packed, transmitted and unpacked when received. The use of ID mask and payload manipulation was done to understand working with message filters and what relevant messages to accept and reject irrelevant message IDs. A closed loop electronic throttle was developed using CAN communication between two ECU's. APPS and TPS sensors are the inputs to the ECU 2 where the data is packed and sent to ECU 1. The difference between the commanded pedal position and desired throttle position is fed as input to the digital controller in ECU 1. The output of the Digital controller i.e. duty cycle is packed in ECU 1 and sent to ECU 2 which drives the DC motor through H-Bridge to obtain desired position. Advanced CAN Send and Advanced CAN Receive were used to send and receive the required data. Advanced CAN blocks were used over Raw blocks as they have more features related to the data. A linear interface model was developed to match the entire operating range of TPS counts with PPS counts. CAN bus was monitored with the output window of CANKing and the obtained results were validated based on their endianness and gain/offset with the display window in the MotoTune. Two methods of implementing gain and offset was studied and functions of changing ID mask and message ID using CAN receive slot properties block was analysed. The function of the slog trigger block was verified with every incoming message. It could be observed that age count remains zero for every incoming message that matches with the message ID and age count rapidly increases when the incoming message doesn't match with the message ID. Use of CAN Fault Status block was understood to detect an error during transmission or receiving when there are conflicting message IDs.

References

- [1] Lucian R. da Silva, R. C. (n.d.). Analysis of Anti-windup Techniques in PID Control of Processes with Measurement Noise. *IFAC PapersOnly*, 51-54.
- [2] Park, K., Lee, J., & J., P. (2013). Torque control of a vehicle with electronic throttle control using an input shaping method. *Springer*.
- [3] Chen, Bo, Lecture 6- Part 2 Mototron CAN-1, 2019