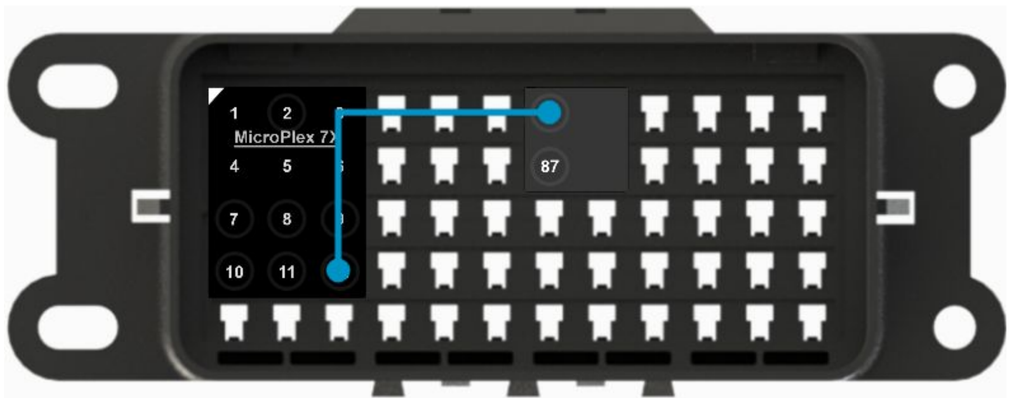
TD-137 Microplex 7X Sample Program Description

Introduction

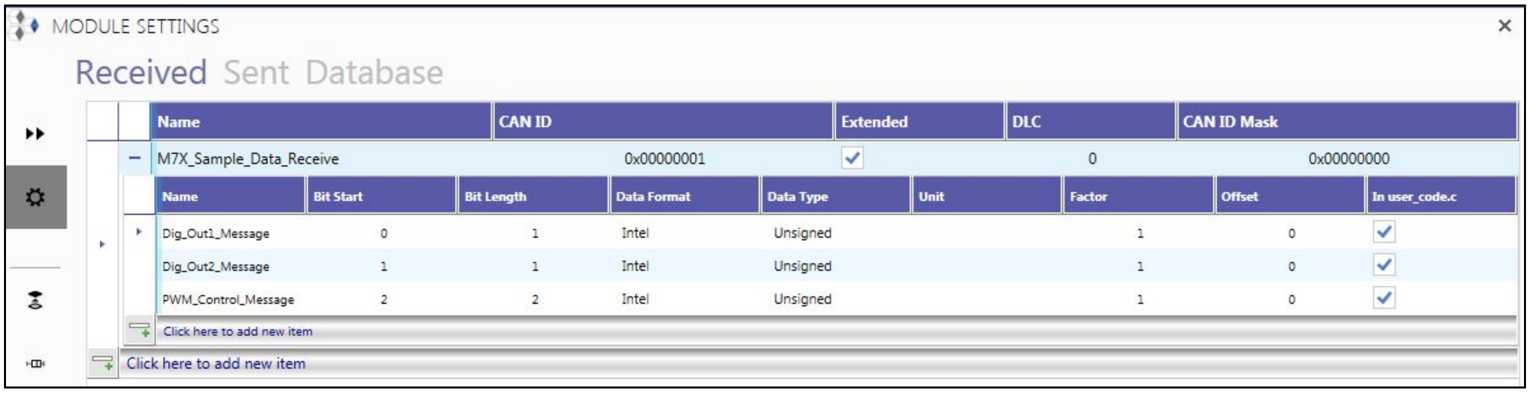
In this sample program for the MicroPlex 7X, we will walk through the set-up of the module settings in the MicroPlex Lab and explain what each part of the code does. This is a very basic program and is intended to help new users get a better understanding of how the most simple features of the MicroPlex Lab are used. The code is written to use input messages, specified in the module settings, to determine the outputs of the pins on the MicroPlex. The 7X also allows us to collect input from the pins and send the information in a message to the CAN bus.

Component Layout and Module Settings

The following image shows the layout of the components; it is simply the MicroPlex 7X connected to an Ultra Micro relay.



Next, we must initialize the data in the Module Settings. This can be accessed by right clicking on the MicroPlex that has already been placed in the Lab. Below are the settings for example messages that are being received from the CAN bus.



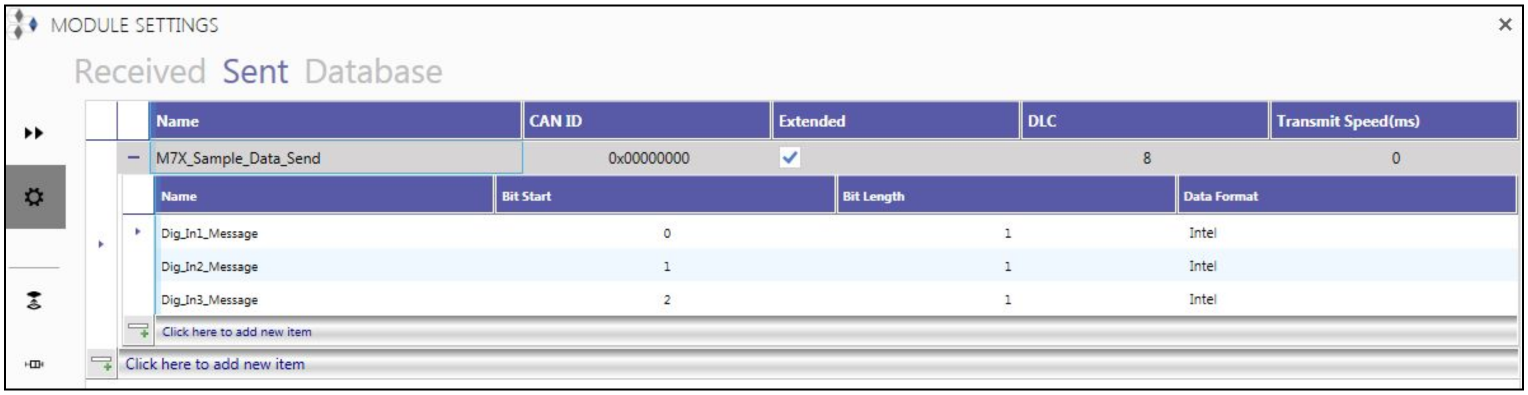
Name	CAN ID	Extended	DLC	CAN ID Mask
M7X_Sample_Data_Receive	0x00000001	✓	0	0x00000000

Name	Bit Start	Bit Length	Data Format	Data Type	Unit	Factor	Offset	In user_code.c
Dig_Out1_Message	0	1	Intel	Unsigned		1	0	✓
Dig_Out2_Message	1	1	Intel	Unsigned		1	0	✓
PWM_Control_Message	2	2	Intel	Unsigned		1	0	✓

Click here to add new item

Click here to add new item

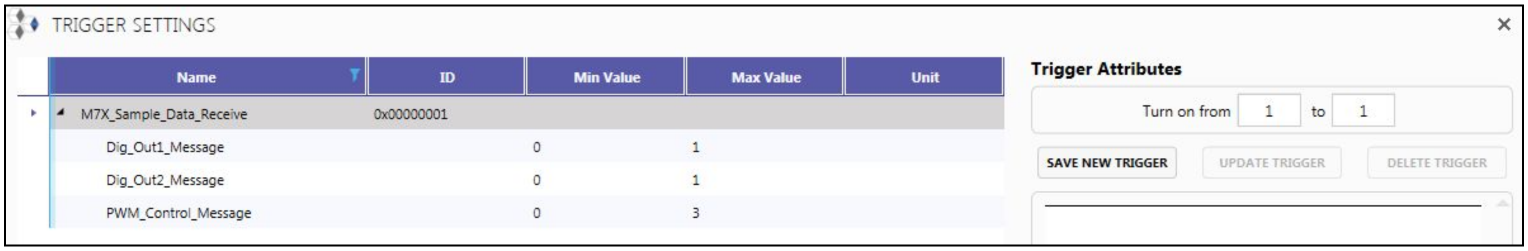
The next image is taken from the module settings, but they are the settings for example messages that are being sent back out to the CAN bus from the module. This page is found by clicking on the "Sent" tab in the module settings window.



Name	CAN ID	Extended	DLC	Transmit Speed(ms)
M7X_Sample_Data_Send	0x00000000	✓	8	0

Name	Bit Start	Bit Length	Data Format
Dig_In1_Message	0	1	Intel
Dig_In2_Message	1	1	Intel
Dig_In3_Message	2	1	Intel

Once this is all setup, we can add a trigger by double clicking on the wire that connects the MicroPlex to the relay. The following window will pop up once the wire has been double-clicked.

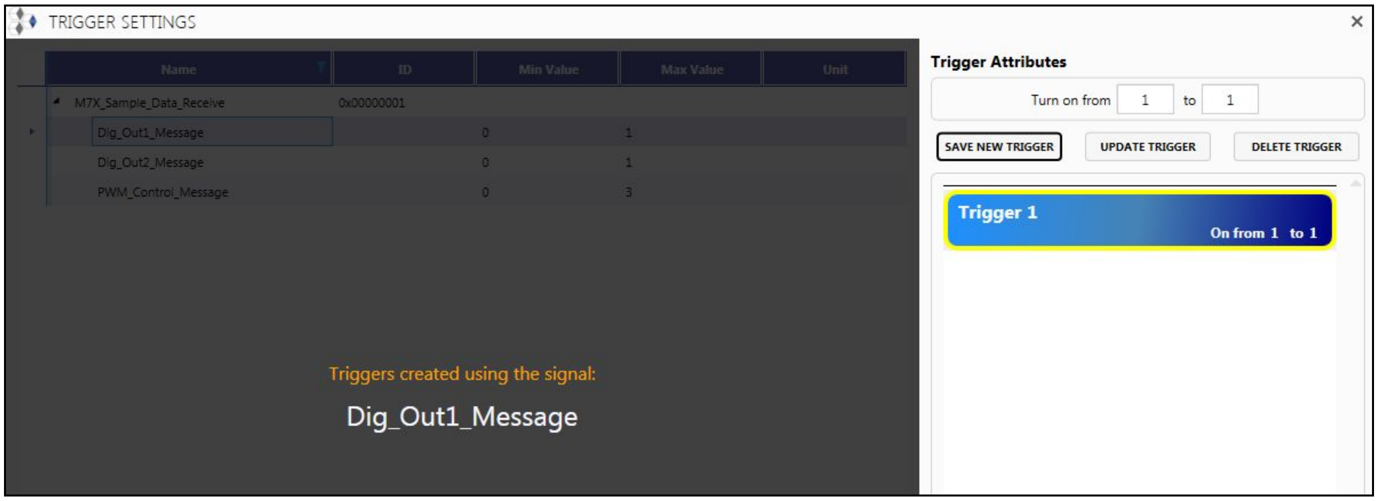


Name	ID	Min Value	Max Value	Unit
M7X_Sample_Data_Receive	0x00000001			
Dig_Out1_Message		0	1	
Dig_Out2_Message		0	1	
PWM_Control_Message		0	3	

Turn on from 1 to 1

By then selecting a data point, which we defined earlier in particular CAN messages from the Module Settings, we decide what controls this trigger. This means that triggers use predetermined bit(s) from a CAN bus message to control the output of a pin on the MicroPlex.

The sample program has a trigger assigned to pin 12 whose output now depends on the bit that was labeled as "Dig_Out1_Message". The picture below shows what the window looks like after this trigger is set.



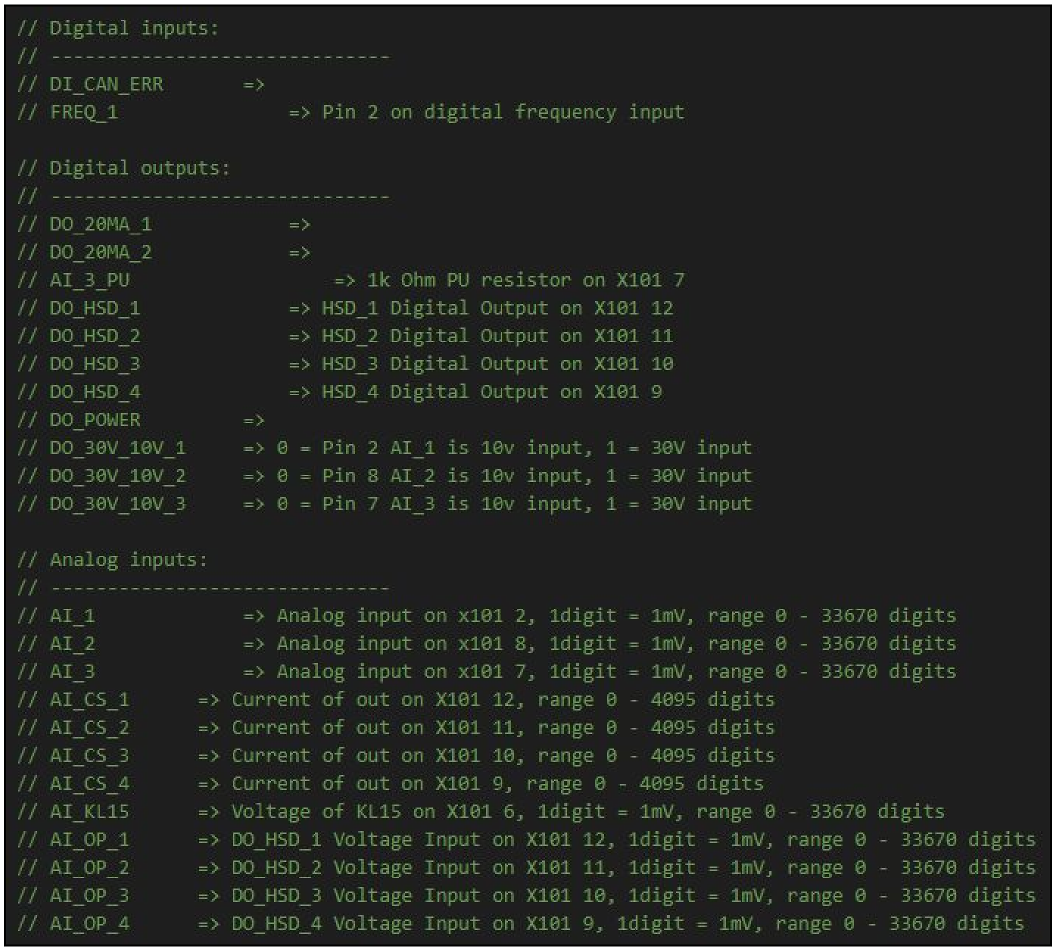
Name	ID	Min Value	Max Value	Unit
M7X_Sample_Data_Receive	0x00000001			
Dig_Out1_Message		0	1	
Dig_Out2_Message		0	1	
PWM_Control_Message		0	3	

Turn on from 1 to 1

Coding the MicroPlex Module

Now that we have finished designing the MicroPlex circuit and set up the module settings and triggers, we are ready to input code that is a little more complicated, but adds additional functionality. To open up the coding window, right click on the module and select "Open user_code.c".

There are many things that will already be written and commented out in this file. Much of this is an explanation of default functions and variables that the user can utilize. All of the functions used below have explanations of how to use them correctly in these comments. One of the first things you will see when opening the usercode file is a list of the names used to label certain inputs and outputs. A screenshot of this is found below. ('X101' is used to refer to the module).

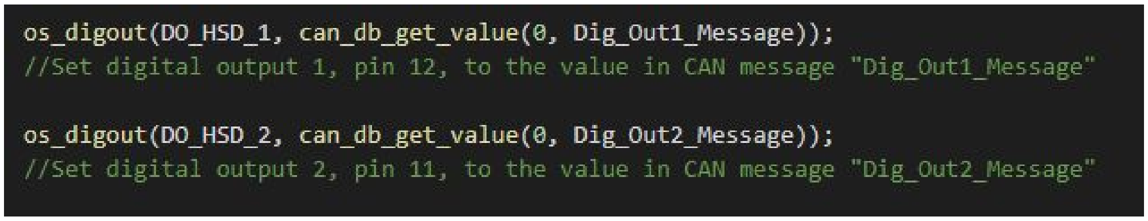


```
// Digital inputs:
// ------------------------------
// DI_CAN_ERR       =>
// FREQ_1               => Pin 2 on digital frequency input

// Digital outputs:
// ------------------------------
// DO_20MA_1            =>
// DO_20MA_2            =>
// AI_3_PU                  => 1k Ohm PU resistor on X101 7
// DO_HSD_1             => HSD_1 Digital Output on X101 12
// DO_HSD_2             => HSD_2 Digital Output on X101 11
// DO_HSD_3             => HSD_3 Digital Output on X101 10
// DO_HSD_4             => HSD_4 Digital Output on X101 9
// DO_POWER         =>
// DO_30V_10V_1     => 0 = Pin 2 AI_1 is 10v input, 1 = 30V input
// DO_30V_10V_2     => 0 = Pin 8 AI_2 is 10v input, 1 = 30V input
// DO_30V_10V_3     => 0 = Pin 7 AI_3 is 10v input, 1 = 30V input

// Analog inputs:
// ------------------------------
// AI_1             => Analog input on x101 2, 1digit = 1mV, range 0 - 33670 digits
// AI_2             => Analog input on x101 8, 1digit = 1mV, range 0 - 33670 digits
// AI_3             => Analog input on x101 7, 1digit = 1mV, range 0 - 33670 digits
// AI_CS_1      => Current of out on X101 12, range 0 - 4095 digits
// AI_CS_2      => Current of out on X101 11, range 0 - 4095 digits
// AI_CS_3      => Current of out on X101 10, range 0 - 4095 digits
// AI_CS_4      => Current of out on X101 9, range 0 - 4095 digits
// AI_KL15      => Voltage of KL15 on X101 6, 1digit = 1mV, range 0 - 33670 digits
// AI_OP_1      => DO_HSD_1 Voltage Input on X101 12, 1digit = 1mV, range 0 - 33670 digits
// AI_OP_2      => DO_HSD_2 Voltage Input on X101 11, 1digit = 1mV, range 0 - 33670 digits
// AI_OP_3      => DO_HSD_3 Voltage Input on X101 10, 1digit = 1mV, range 0 - 33670 digits
// AI_OP_4      => DO_HSD_4 Voltage Input on X101 9, 1digit = 1mV, range 0 - 33670 digits
```

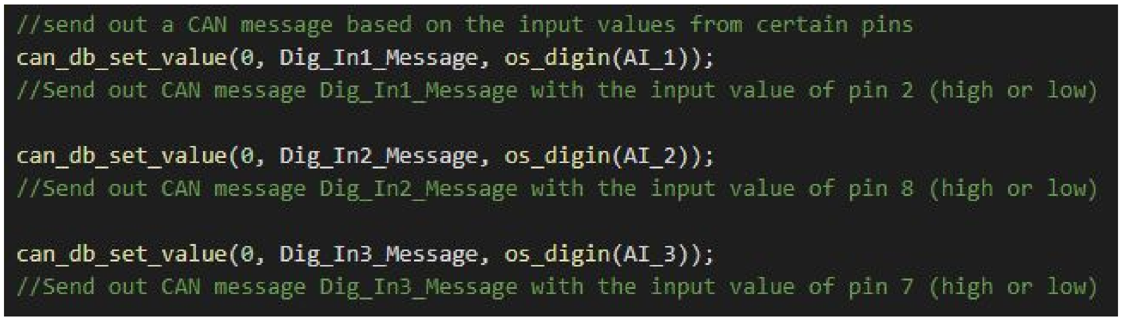
The majority of code we write goes in the function “void usercode(void){ }”. You can use “Ctrl+f” to search for this function within the code. In this sample program’s user code, we have used the default function “os_digout()” to turn on an output pin based on the bit value specified by the second argument of the function “can_db_get_value() ”. (Note, this can also be accomplished by utilizing the “Triggers” as explained above.)



```
os_digout(DO_HSD_1, can_db_get_value(0, Dig_Out1_Message));
//Set digital output 1, pin 12, to the value in CAN message "Dig_Out1_Message"

os_digout(DO_HSD_2, can_db_get_value(0, Dig_Out2_Message));
//Set digital output 2, pin 11, to the value in CAN message "Dig_Out2_Message"
```

We are also able to get input from a pin, using ‘os_digin()’, and then set the value of a bit(s) in a message through the use of ‘can_db_set_value()’. The code below shows how it is written:

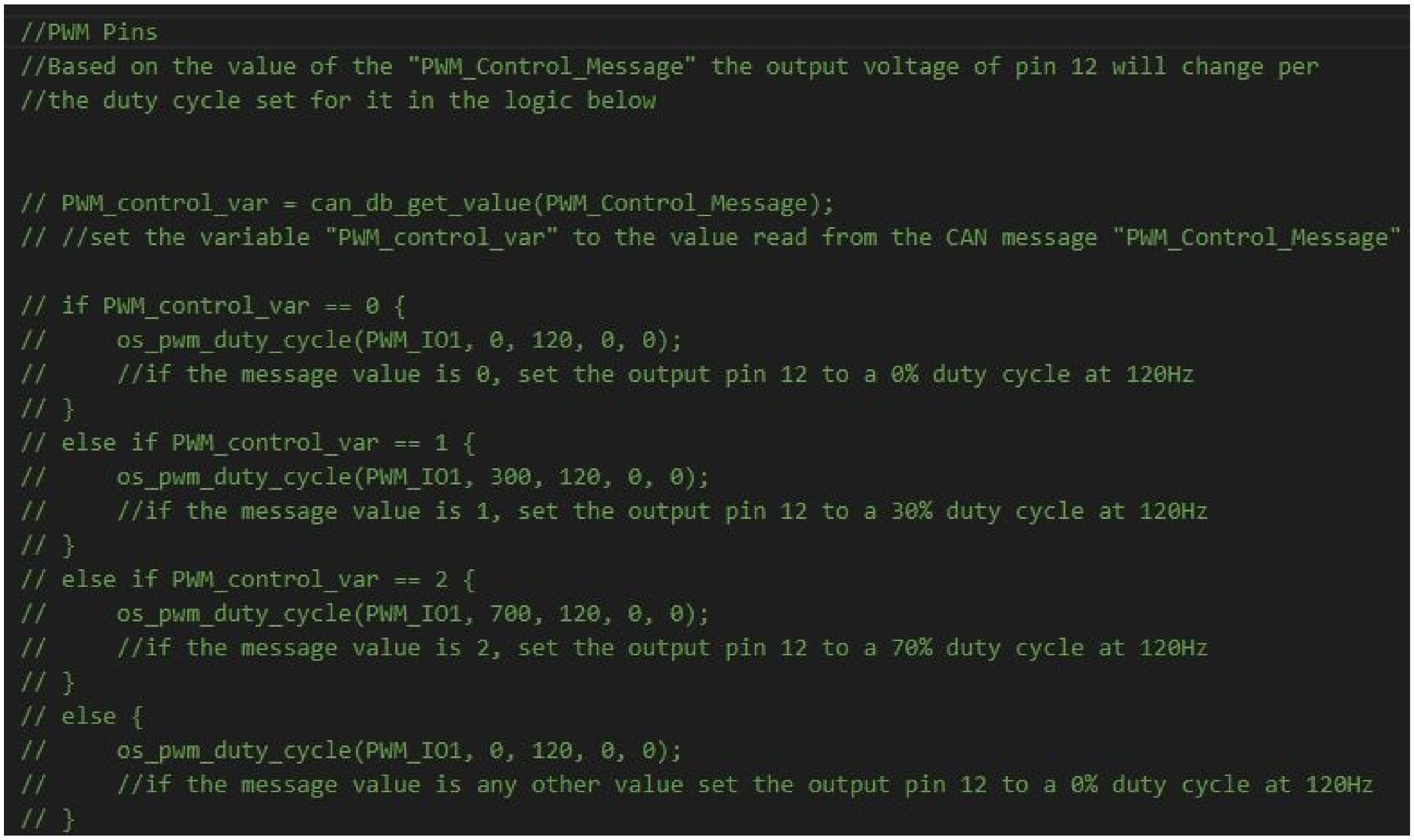


```
//send out a CAN message based on the input values from certain pins
can_db_set_value(0, Dig_In1_Message, os_digin(AI_1));
//Send out CAN message Dig_In1_Message with the input value of pin 2 (high or low)

can_db_set_value(0, Dig_In2_Message, os_digin(AI_2));
//Send out CAN message Dig_In2_Message with the input value of pin 8 (high or low)

can_db_set_value(0, Dig_In3_Message, os_digin(AI_3));
//Send out CAN message Dig_In3_Message with the input value of pin 7 (high or low)
```

Additionally, we have provided example code that uses pulse width modulation (PWM). If you can recall from earlier, when we established the modulation settings, we set “PWM_Control_Message” to have a bit length of 2. This allows it to have a value in the range of 0-3, which we can then use to have different duty cycles sent out. Our example may be seen in the image below:



```
//PWM Pins
//Based on the value of the "PWM_Control_Message" the output voltage of pin 12 will change per
//the duty cycle set for it in the logic below


// PWM_control_var = can_db_get_value(PWM_Control_Message);
// //set the variable "PWM_control_var" to the value read from the CAN message "PWM_Control_Message"

// if PWM_control_var == 0 {
//     os_pwm_duty_cycle(PWM_IO1, 0, 120, 0, 0);
//     //if the message value is 0, set the output pin 12 to a 0% duty cycle at 120Hz
// }
// else if PWM_control_var == 1 {
//     os_pwm_duty_cycle(PWM_IO1, 300, 120, 0, 0);
//     //if the message value is 1, set the output pin 12 to a 30% duty cycle at 120Hz
// }
// else if PWM_control_var == 2 {
//     os_pwm_duty_cycle(PWM_IO1, 700, 120, 0, 0);
//     //if the message value is 2, set the output pin 12 to a 70% duty cycle at 120Hz
// }
// else {
//     os_pwm_duty_cycle(PWM_IO1, 0, 120, 0, 0);
//     //if the message value is any other value set the output pin 12 to a 0% duty cycle at 120Hz
// }
```

As you can see, there are many ways to utilize the MicroPlex. If you have any questions or concerns regarding the product, please feel free to reach out to our team at Chief Enterprises!