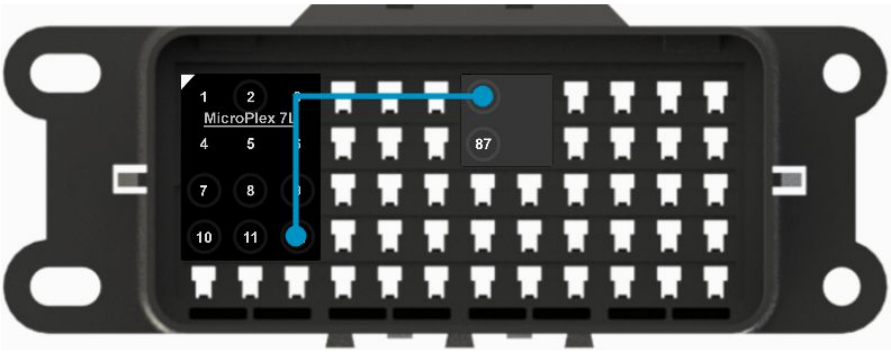
TD-139 MicroPlex 7L Sample Program Description

Introduction

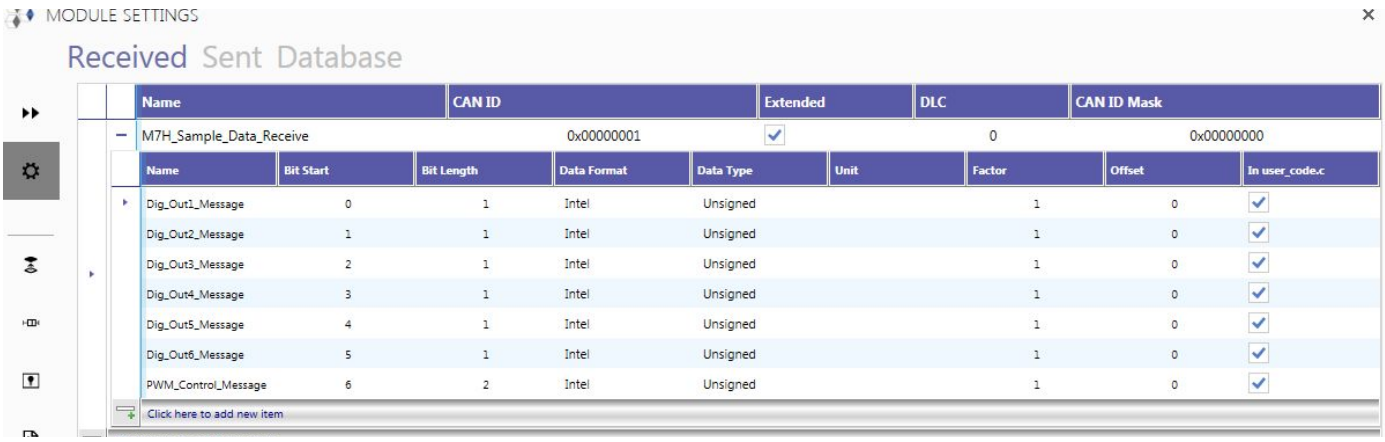
In this sample program for the MicroPlex 7L, we will walk through the set-up of the module settings in the MicroPlex Lab and explain what each part of the code does. This is a very basic program and is intended to help new users get a better understanding of how the most simple features of the MicroPlex Lab are used. It will be very similar to the program for the 7H module. The main difference between the two is that the 7H uses high side driver outputs and the 7L uses low side driver. The code is written to use input messages, specified in the module settings, to determine the outputs of the pins on the MicroPlex.

Component Layout and Module Settings

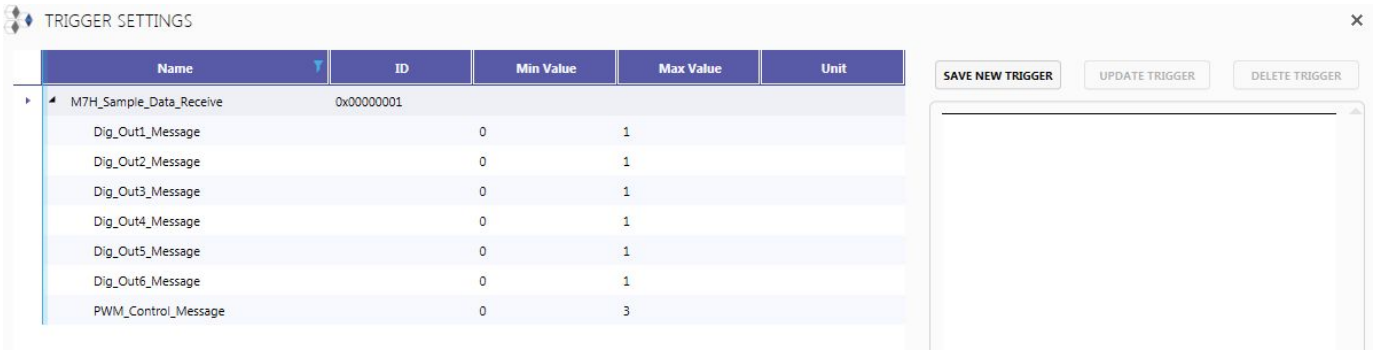
The following image shows the layout of the components; it is simply pin 12 on the MicroPlex 7L connected to an Ultra Micro relay. To place the components just drag and drop them, and to place a wire just click on a module pin.



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. The settings used in the sample program are shown below.

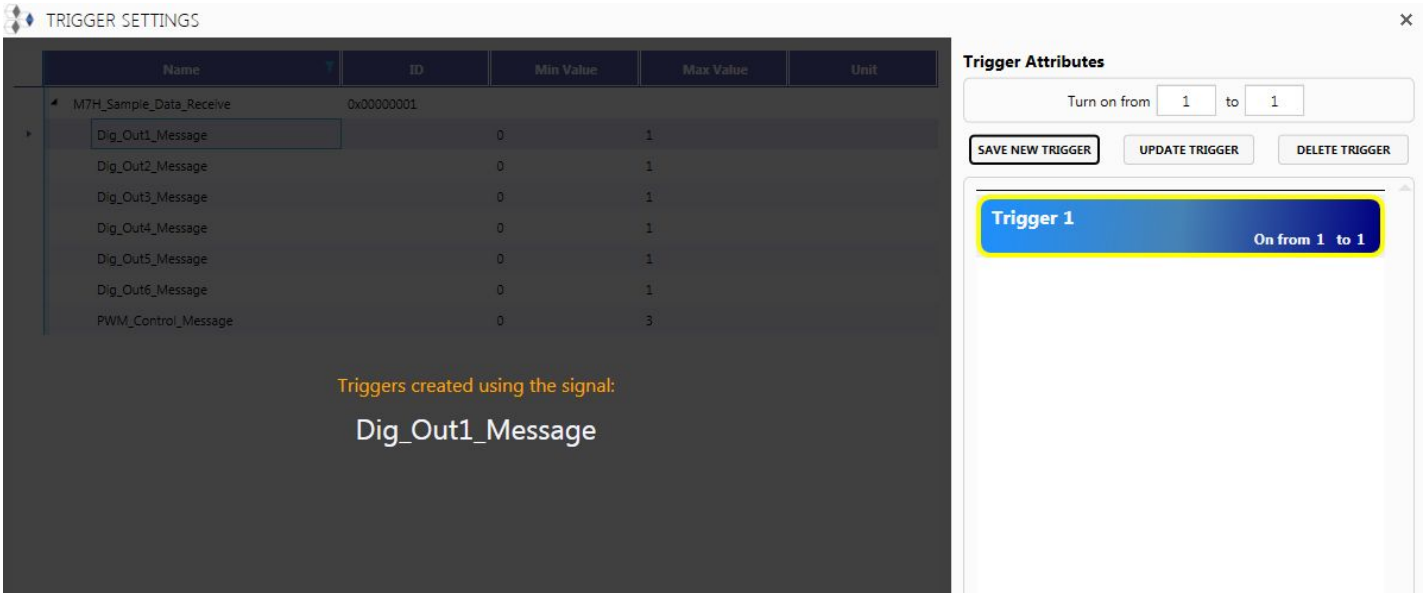


Once this is all setup, we can add a trigger by double clicking on the wire that connects the MicroPlex to the relay. The following window will pop up once the wire has been double-clicked.



By then selecting a message, which we defined earlier as certain bits in a particular message from the CAN bus, 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.



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. 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. For instance, one of the first things you will see is a list of the names used to label certain outputs. A screenshot of this is found below. (‘X101’ is used to refer to the module)

```
// Digital inputs:
// ----------------------------
// DI_CAN_ERR       =>

// Digital outputs:
// ----------------------------
// DO_LSD_1         => LSD_1 Digital Output on X101 12
// DO_LSD_2         => LSD_2 Digital Output on X101 11
// DO_LSD_3         => LSD_3 Digital Output on X101 10
// DO_LSD_4         => LSD_4 Digital Output on X101 9
// DO_LSD_5         => LSD_5 Digital Output on X101 8
// DO_LSD_6         => LSD_6 Digital Output on X101 2
// DO_LSD_7         => LSD_7 Digital Output on X101 7
// DO_POWER     =>

// Analog Inputs:
// ----------------------------
// AI_KL15      => Voltage of KL15 on X101 6, 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() ”.

```
//Digital output examples
//set the digital outputs based on the CAN messages received
os_digout(DO_LSD_1, can_db_get_value(0, Dig_Out1_Message));
//set digital output 1, pin 12, to the value of Dig_Out1_Message


os_digout(DO_LSD_2, can_db_get_value(0, Dig_Out2_Message));
//set digital output 2, pin 11, to the value of Dig_Out2_Message


os_digout(DO_LSD_3, can_db_get_value(0, Dig_Out3_Message));
//set digital output 3, pin 10, to the value of Dig_Out3_Message


os_digout(DO_LSD_4, can_db_get_value(0, Dig_Out4_Message));
//set digital output 4, pin 9, to the value of Dig_Out4_Message


os_digout(DO_LSD_5, can_db_get_value(0, Dig_Out5_Message));
//set digital output 5, pin 8, to the value of Dig_Out5_Message

os_digout(DO_LSD_6, can_db_get_value(0, Dig_Out6_Message));
//set digital output 6, pin 2, to the value of Dig_Out6_Message
```

The above code could also be implemented, without writing any code, through the use of triggers.

We have also 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 is seen in the image below:

```
//PWM Pins Example
//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

//the logic below would be used instead of line 82 above as it uses pin 12 as well


// 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!