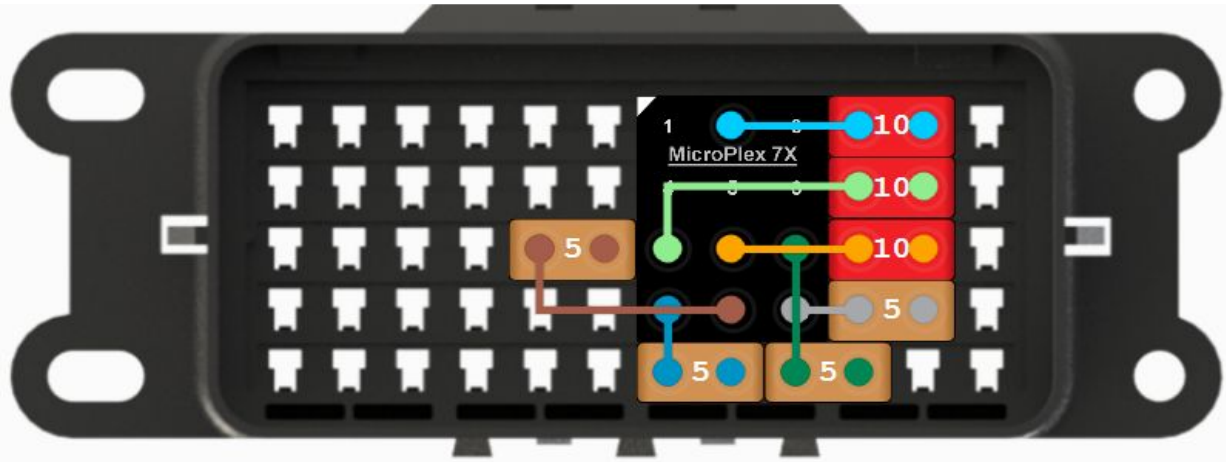
Introduction

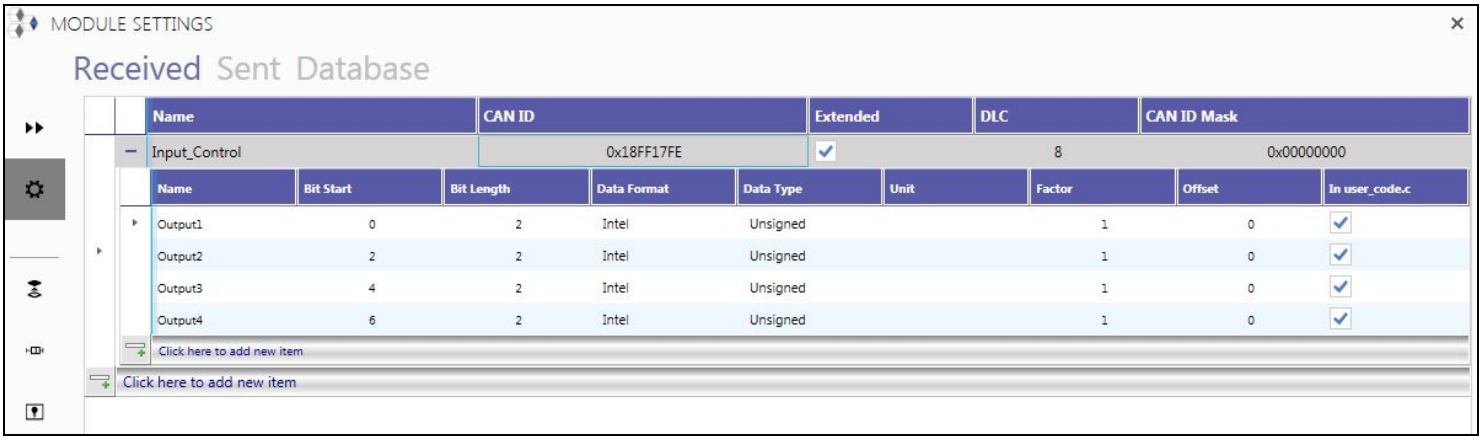
This sample program will go through the 7X programming process for a practical application. The 7 I/O pins on the module are each connected to a fuse and a solenoid. The MicroPlex is responsible for having the correct outputs when a CAN bus message gives the signal, and the module also sends a message to the CAN bus with information collected from the input pins.

Component Layout and Module Settings

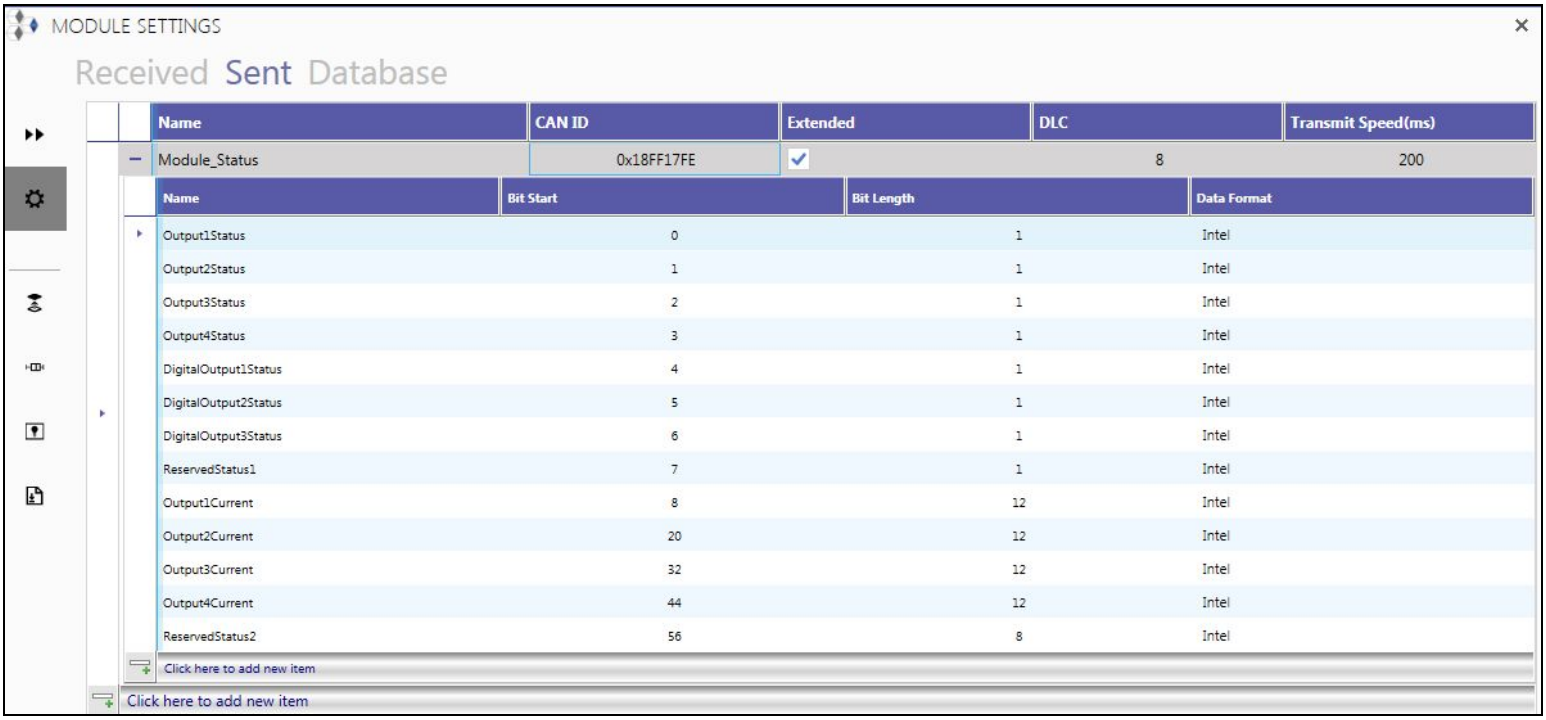
First we must place the 7X module in the case along with three 10 Amp and four 5 Amp fuses (change the rated current by right clicking on the fuse). The following image shows which fuses should connect to what pins.



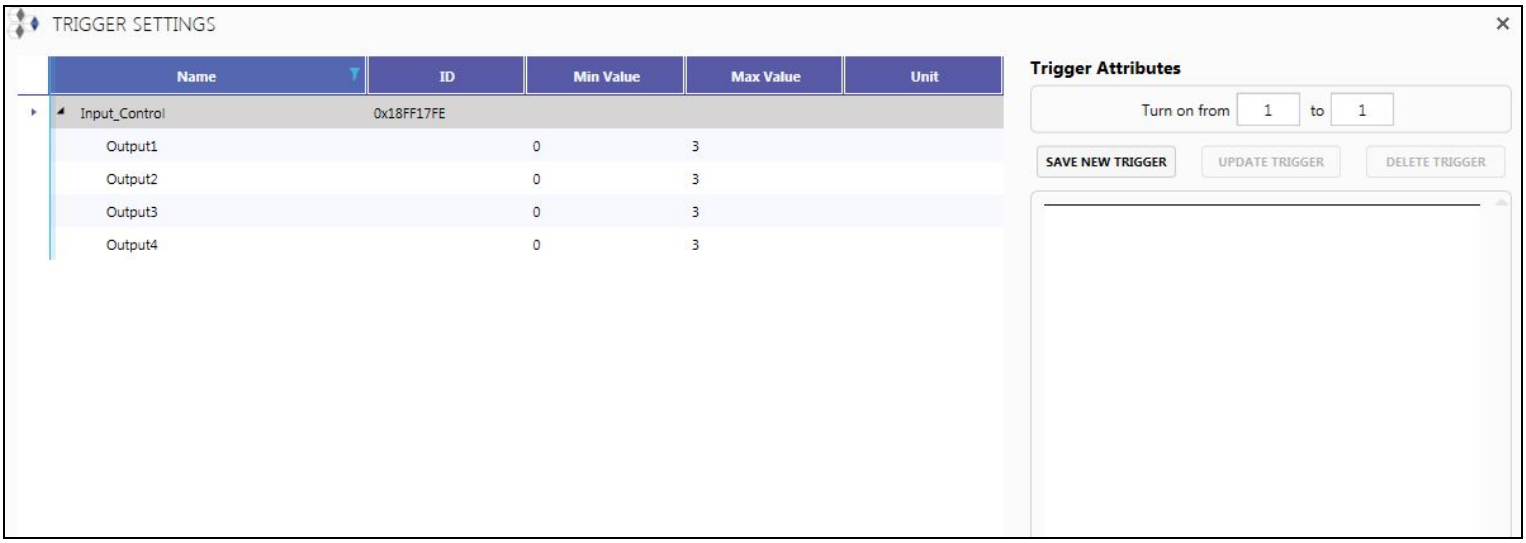
Now we will set the module settings which can be found by right clicking on the MicroPlex module that was already placed. The ‘Received’ settings should be set up so that the module knows the exact message, and bits in that message, which indicate whether or not the solenoid should be on. This can be seen below.



Next, click on the ‘Sent’ tab. This is where we set up the messages that need to be sent back to the CAN bus.



The next step is setting up the triggers. The 7X module will only allow triggers to be set up on pins 9-12 because pins 2, 7, and 8 are input only. To set up a trigger, double click on one of the wires connected to pins 9-12. The following window will appear.



Once this window appears, you can select one of the received variables we set up earlier to control the trigger.

After selecting which bit grouping will control the trigger, the range of values for which the trigger should be on can be set. For our purposes all triggers should be on from 1 to 1. Then you click on ‘SAVE NEW TRIGGER’. The trigger on the wire coming from pin 9 looks like this.

The remaining 3 triggers should be set up exactly like the first one except that the signal used for the triggers on pins 10, 11, and 12 should be Output2, Output3, and Output4, respectively.

Coding the MicroPlex Module

Once you have set all four triggers, you will need to write the user code. To get to this window, right click on the MicroPlex and select “Open user_code.c”. There will be a lot of default code and almost all of it should be commented out. This default code should have reserved places for the user to write certain functions; the “void usercode(void){}” function will be where we write our code. The commented out portions inform the user about predefined variables and functions that you may use while writing your own code.

The first thing the user must do in our example is declare the needed variables. This can be done anywhere but we chose to put it after the comments about digital and analog inputs. The declared variables can be seen below:

```
// 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


// ------------------------------------------------------------------------------
// Example variables
// ------------------------------------------------------------------------------
uint16_t variable1, variable2, Output1value, Output2value, Output3value, Output4value;
uint16_t DigitalInput1value, DigitalInput2value, DigitalInput3value, OutputCurrent1, OutputCurrent2, OutputCurrent3, OutputCurrent4;
uint32_t time_val;
```

The rest of the code we will write is going to be within the “void usercode{}” function. The first part of the code is setting the values of the Output Status message based on whether or not the output is turned on. This is done with the following code:

```
//Output Status


//set Output value variables (defined above) to the data point from corresponding message inputs
Output1value = can_db_get_value(0, Output1);
Output2value = can_db_get_value(0, Output2);
Output3value = can_db_get_value(0, Output3);
Output4value = can_db_get_value(0, Output4);

//set value of Output Status messages to the value of the corresponding variable from above
can_db_set_value(0, Output1Status, Output1value);
can_db_set_value(0, Output2Status, Output2value);
can_db_set_value(0, Output3Status, Output3value);
can_db_set_value(0, Output4Status, Output4value);

//note: the above can be combined into simpler lines of code with no variables, but they are separated for clarity.
//For example the status of Output 1 can be set and sent via the following:
//can_db_set_value(Output1Status, (can_db_get_value(0,Output1)));
```

Next, we want to write code that will set the values of the digital input status message. By using the predefined “os_digin” function and “AI_1” variable, we assign a value of either 1 or 0 to the “DigitalInputvalue” variables. This is then used to set the value of the digital input status:

```
//Digital Input Status


//Set digital input variables (defined above) to the value obtained from the "os_digin" function
//Sets Variable to input from pin 2
DigitalInput1value = os_digin(AI_1);

//Sets Variable to input from pin 7
DigitalInput2value = os_digin(AI_3);

//Sets Variable to input from pin 8
DigitalInput3value = os_digin(AI_2);

//set value of Digital Input Status messages to the value of the corresponding variable from above
can_db_set_value(0, DigitalInput1Status, DigitalInput1value);
can_db_set_value(0, DigitalInput2Status, DigitalInput2value);
can_db_set_value(0, DigitalInput3Status, DigitalInput3value);

//note: the above can be combined into simpler lines of code with no variables, but they are separated for clarity.
//For example the status of digital input 1 can be set and sent via the following:
//can_db_set_value(DigitalInput1Status, (os_digin(AI_1)));
```

The final part of code we need to implement is something that will set the values of solenoid current bits in the output message. This is done in a very similar way as the previous code:

```
//Output Current


//current output on pins 9, 10, 11, and 12 are treated as analog inputs, os_align_mv function reads that "input"
//sets Output current variables (defined above) to the value obtained form the os_align_mv function of each pin

//pin 9
OutputCurrent1 = os_algin_mv(AI_CS_4);

//pin 10
OutputCurrent2 = os_algin_mv(AI_CS_3);

//pin 11
OutputCurrent3 = os_algin_mv(AI_CS_2);

//pin 12
OutputCurrent4 = os_algin_mv(AI_CS_1);

//set value of Output Current Status messages to the value of the corresponding variable from above
can_db_set_value(0, Output1Current, OutputCurrent1);
can_db_set_value(0, Output2Current, OutputCurrent2);
can_db_set_value(0, Output3Current, OutputCurrent3);
can_db_set_value(0, Output4Current, OutputCurrent4);

//note: the above can be combined into simpler lines of code with no variables, but they are separated for clarity.
//For example the status of Output current 1 can be set and sent via the following:
//can_db_set_value(Output1Current, (os_algin(AI_CS_4)));
```

We have now completed the code which marks the end of a completed MicroPlex program. To find definitions of the predefined functions and variables that were used, just press ‘Ctrl + f’ and search for the description within the comments of the code.

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!