**INTRODUCTION**

Advances in data communications have created efficient methods for several devices to communicate over a minimum number of system wires. The Controller Area Network (CAN) is one of these methods. CAN sends and receives messages over a two-wire CAN bus. The nodes broadcast their individual messages over the CAN bus, while the receivers are setup to accept the message and anticipate an acknowledgment (ACK) signal indicating the receipt of a non-corrupted message. The protocol of the CAN has two states and the bits are either dominant (logic '0') or recessive (logic '1'). Nodes may attempt to transmit a message at the same time. To ensure that collisions do not reduce the throughput of the bus, there is an arbitration scheme in which a node will continue to transmit until a dominant bit is detected, while that node is expecting a recessive bit (in the ID field) on the CAN bus. The node(s) that lose arbitration will automatically terminate their transmission and switch to receive mode. Once the CAN bus enters an idle state, these nodes attempt to re-transmit. If the node does not lose arbitration, it completes its transmission. For additional information on the CAN protocol, refer to AN713, "Controller Area Network (CAN) Basics", DS00713.

The bus configuration operates by the multi-master principle and allows several node boards to connect directly to the bus. If one node board fails in the system, the other node boards are not affected. The probability of the entire network failing is extremely low compared to ring type networks. Ring type networks have a high probability failure rate, due to the fact that if one node malfunctions, the entire network becomes inoperable. The CAN controller seeks to solve this problem.

**MCP2510 CAN Controller Benefits**

- Monitors Several Devices
- Individual Node Programming
- Replaces a Large Wiring Harness

---

**MODULE OVERVIEW**

The module hardware can be divided into two components. These are:
- CAN-NET Node Board
- CAN-NET Analog Input Board

These boards can be purchased from Diversified Engineering by ordering the CAN-NET Analog Input Node Kit. The CAN-NET Analog Input Board also requires that some of the options be installed by the customer. Two additional components are: a 14.5-PSI Pressure Transducer and a LED. Table 1 gives the part numbers for these components.

**TABLE 1: COMPONENT PART NUMBERS**

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Component</th>
<th>Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diversified Engineering</td>
<td>CAN-NET Analog Input Node Kit</td>
<td>905190</td>
</tr>
<tr>
<td>Motorola</td>
<td>Pressure Transducer</td>
<td>MPX2010DP</td>
</tr>
</tbody>
</table>

This module has several key features. These include:

- High-Speed SPI™ Interface
- MPLAB® ICD Debugging Tool
- Low Power CMOS Technology
- PWM Output for Driving a Lamp
- Supports SPI modes 0,0 and 1,1
CAN-NET Node Board

The CAN-NET Node Board consists of hardware devices that are used in conjunction with software programming techniques to achieve an optimal Controller Area Network. The versatility of the CAN controller enables a wide variety of applications to be created based on the concept of this particular design.

The MCP2510 CAN controller is the heart of the CAN interface. It handles all transmitting and receiving of message packets that contain useful information for other nodes on the network via the CAN bus. The MCP2510 CAN controller is also designed to interface with the Serial Peripheral Interface (SPI) port. The SPI port is available on the PIC16F876 microcontroller and the MCP3201 Analog-to-Digital Converter (ADC).

The PIC16F876 microcontroller stores the program in memory and reads the DIP switch settings for sending and receiving messages. It controls the PWM output and enables the MPLAB ICD to be used as a debugging tool.

CAN-NET Analog Input Board

The MCP3201 ADC accepts input signals from the pressure sensor, utilizing a differential amplifier configuration. The MCP602 amplifier uses single-supply CMOS operational amplifier (op amp) technology.

HARDWARE OVERVIEW

This section describes the CAN-NET Node board hardware and how the CAN functions in the node board system. Schematics can be found in Appendix A.

MCP2510 CAN Controller

The high-level design of this system is shown in Figure 1. The concept is to enable the MCP2510 CAN controller, the PIC16F876 microcontroller and the MCP3201 ADC to efficiently communicate with each other utilizing the SPI. The MCP2510 handles the lower-level protocols.

FIGURE 1: BLOCK DIAGRAM OF THE CAN NODE BOARD
The PIC16F876 microcontroller stores the program in memory and constantly polls the MCP3201 ADC, along with the reference A/D.

In the main loop of the program, a variable is toggled. When the value of the variable is a logic ‘0’, the PICmicro® device reads the pressure sensor and, once the value of the variable is a logic ‘1’, the PICmicro device reads the reference A/D. The microcontroller also reads the settings of the input switches. The first two (of four) switches tells the microcontroller which message the node is allowed to receive. The last two (of four) switches tell the microcontroller the transmit address of the node. The configuration shown in Figure 2 illustrates three node boards on a CAN bus set to transmit and receive certain messages.

FIGURE 2: THREE NODE BOARDS CONNECTED TO THE CAN BUS
In this case, each node transmits its own pressure sensor value and is set to receive a value of the pressure sensor from a different node. The identification for each node board is 01, 10 and 11. These settings are transmit and receive identifiers. Node Board 1 is set to receive the pressure sensor value from Node Board 2. Node Board 2 is set to receive the pressure sensor value from Node Board 3, while Node Board 3 is set to receive the pressure sensor value from Node Board 1. The pressure sensor value of each node board is directly proportional to the PWM output of the corresponding microcontroller.

The CAN driver chip converts the input and output to the CAN bus voltages ranging from 0 to 5 volts, with a shift of ±12V.

The MCP3201 is a 12-bit ADC with on-board sample and hold circuitry.

The input to the device comes from a differential amplifier circuit that communicates over the serial interface using the SPI protocol. The MCP602 op amp is used to design a suitable differential amplifier.

The gain of the amplifier is determined by the following equation:

**EQUATION 1: AMPLIFIER GAIN**

\[
\text{Gain} = \frac{R_{14}}{R_{13}} + \frac{R_{13}}{R_{P1}}
\]

Figure 3 shows a differential amplifier circuit. The input to this amplifier ranges from 0 to 5 volts and is useful for pressure applications. The pressure can be referred to as “zero pressure”. The normalized pressure setting consists of negative pressure and positive pressure. The pressure sensor produces a negative voltage when there is negative pressure and a positive voltage when there is positive pressure. The reference for the differential amplifier is 2.5 volts. Above 2.5 volts, it indicates a positive pressure. Below 2.5 volts, it indicates a negative pressure. The CAN-NET Node Board with the Analog I/O Board is designed specifically for pressure, but can easily be altered to do both.

**Hardware Tools**

The MPLAB ICD is a tool which enhances the code development and hardware debugging process. The debugger uses a PIC16F877 device and operates in “real time”. This low cost tool saves engineering time (“money”) by allowing the application program and circuit to be evaluated and enhanced in real time. The ICD interface also allows the PIC16F87X devices to be programmed after the board has been manufactured. This allows software changes or updates to be programmed into the device. The ICD uses the RB6 and RB7 pins of the PIC16F87X for this. For that reason, these pins are not used for any other purpose in this system. For additional information on In-Circuit Serial Programming™, please refer to Microchip’s In-Circuit Serial Programming (ICSP™) Guide, (DS30277).

**FIGURE 3: DIFFERENTIAL AMPLIFIER CIRCUIT**

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
<th>Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>R11</td>
<td>30.1 kΩ</td>
<td>1%</td>
</tr>
<tr>
<td>R12</td>
<td>10.0 kΩ</td>
<td>1%</td>
</tr>
<tr>
<td>R13</td>
<td>10.0 kΩ</td>
<td>1%</td>
</tr>
<tr>
<td>R14</td>
<td>30.1 kΩ</td>
<td>1%</td>
</tr>
<tr>
<td>R15</td>
<td>1.0 KΩ</td>
<td>1%</td>
</tr>
<tr>
<td>RP1</td>
<td>0.0 to 50.0 kΩ</td>
<td>N.A.</td>
</tr>
</tbody>
</table>
SOFTWARE OVERVIEW

Programming Style

The code for the node board is written in the PICmicro device instruction set to be assembled using Microchip’s MPLAB environment. There is a significant use of macros to make the code more readable and less error prone. The macros are defined in three files:

1. Near the top of the main file,
2. CANLIB.ASM (file contains the CAN macros)
3. MACROS16.INC

If an unfamiliar instruction is found, it is probably made up of a set of familiar instructions in one of the macros. The macros in the MACROS16.INC file are used extensively in writing code for the PICmicro microcontroller family, because they increase readability and greatly reduce programming errors.

Common Code

The node board uses common software files to maximize the program’s efficiency. The routines that enable communication with the MCP2510 CAN chip are in the file CANLIB.ASM, while the definitions for the MCP2510 registers are in MCP2510.INC. The common macros are in MACROS16.INC.

SPI Communications

Communication from a device on the node (such as a microcontroller) to the MCP2510 is accomplished through the SPI bus. The PICmicro device used on the node board fully supports the SPI in the master mode. Command strings are sent and received using a single software buffer. To send a string, the software buffer (called pSPIBufBase) is loaded with the bytes to send and the SPI interrupt is turned on. The interrupt handler exchanges bytes with the MCP2510. The bytes received from the MCP2510 replace the bytes that were sent from the software buffer so that, once the string has been sent, the buffer will contain the bytes received from the MCP2510. All communication with the MCP2510 is handled in this manner and is encapsulated in the routines in the CANLIB.ASM.

General ID Structure

The ID structure used by the node boards is determined by the settings on the DIP switches on power-up or after a reset. Changing the DIP switches while running has no effect on the ID structure.

Receive ID Structure

The node board uses the settings in Table 2 for receiving:

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RxMask0</td>
<td>0xFFF</td>
</tr>
<tr>
<td>RxMask1</td>
<td>0xFFF</td>
</tr>
<tr>
<td>RxFilter0</td>
<td>0xFFF</td>
</tr>
<tr>
<td>RxFilter1</td>
<td>0xFFF</td>
</tr>
<tr>
<td>RxFilter2</td>
<td>0x000(1)</td>
</tr>
<tr>
<td>RxFilter3</td>
<td>0x10</td>
</tr>
<tr>
<td>RxFilter4</td>
<td>0xFFF</td>
</tr>
<tr>
<td>RxFilter5</td>
<td>0xFFF</td>
</tr>
</tbody>
</table>

Note: This value is the Base Receive ID for receiving. The DIP #1 and DIP #2 settings are used to determine this value.

The DIP settings for receiving are shown in Table 3.

<table>
<thead>
<tr>
<th>DIP #1</th>
<th>DIP #2</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0x000</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0x100</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0x200</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0x300</td>
</tr>
</tbody>
</table>

A message received for RxFilter2 (Base Receive ID) is assumed to be a two-byte integer that contains a 12-bit value between 0 and 4095. The 12-bit data is used to generate a PWM output, where a ‘0’ generates a 0% duty cycle and 0xFFF generates a 100% duty cycle.
Transmit ID Structure

The node board transmits a CAN message every 131 ms. A message contains two data bytes that represent a 12-bit value, with least significant byte (LSB) sent first.

The pressure switch is assigned to the Base Transmit ID and is measured and transmitted with that ID every 393 ms as a two-byte integer in the range 0 to 4095. The reader should note that, while the A/D measurement is 8 bits, it is shifted by 4 bits before transmission. Thus, its actual range is 0x0000 to 0x0FF0.

Each data source has its own unique Base Transmit ID obtained from the settings of DIP #3 and DIP #4. These settings are shown in Table 4.

<table>
<thead>
<tr>
<th>DIP #3</th>
<th>DIP #4</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>All transmissions are disabled</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0x100</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0x200</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0x300</td>
</tr>
</tbody>
</table>

The MCP2510 CAN controller has a 125 kbit rate and the polling method is used. The use of interrupts would be easier in the system, but polling allows the interrupt pins to remain free for other potential functions in the system.

There are three methods for transmitting information:
1. Responding to an external event (event driven).
2. Sending messages at regular intervals (timed transmission). The time of the event may be unknown.
3. A combination of the first two. The receiver can expect messages at a maximum known interval.

The flowcharts for the operation of the source code are shown in Figure 4 through Figure 24. The subroutines contain the actual name and the function it performs within the flowchart, so that it can be easily referenced with the source code. Table 5 gives the function names used and a brief description of the function. In the electronic version of this document, clicking of the function name will take you (link you) to the page for that function.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function Description</th>
<th>Figure Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td>This is the main loop of the program.</td>
<td>Figure 4</td>
</tr>
<tr>
<td>Hardstart</td>
<td>Does a full initialization of the system.</td>
<td>Figure 5</td>
</tr>
<tr>
<td>Init</td>
<td>Initializes the PIC16F87X registers.</td>
<td>Figure 6</td>
</tr>
<tr>
<td>InitSPIPort</td>
<td>Initializes the PIC16F87X SPI port.</td>
<td>Figure 7</td>
</tr>
<tr>
<td>Init2510</td>
<td>Initializes the MCP2510’s registers.</td>
<td>Figure 8</td>
</tr>
<tr>
<td>Read3201</td>
<td>Reads the specified register in the MCP3201 (A/D converter).</td>
<td>Figure 9</td>
</tr>
<tr>
<td>ReadA2D</td>
<td>Reads the specified register in the MCP3201 (A/D converter).</td>
<td>Figure 10</td>
</tr>
<tr>
<td>WaitANDeqZ</td>
<td>Waits for pending messages.</td>
<td>Figure 11</td>
</tr>
<tr>
<td>CheckCANMsg</td>
<td>Checks for messages in the receive buffer.</td>
<td>Figure 12</td>
</tr>
<tr>
<td>ParseCAN</td>
<td>Setup messages for the PWM output.</td>
<td>Figure 13</td>
</tr>
<tr>
<td>Reset2510</td>
<td>Resets the MCP2510.</td>
<td>Figure 14</td>
</tr>
<tr>
<td>BitMod2510</td>
<td>Modifies the value of a specified bit in the MCP2510.</td>
<td>Figure 15</td>
</tr>
<tr>
<td>Wrt2510Reg</td>
<td>Writes the specified register in the MCP2510 (CAN interface).</td>
<td>Figure 16</td>
</tr>
<tr>
<td>SetNormalMode</td>
<td>Sets the MCP2510 to normal operating mode.</td>
<td>Figure 17</td>
</tr>
<tr>
<td>Rd2510Reg</td>
<td>Reads the specified register in the MCP2510 (CAN interface).</td>
<td>Figure 18</td>
</tr>
<tr>
<td>OutputPWM</td>
<td>Loads the PWM duty cycle registers with the values in the specified registers.</td>
<td>Figure 19</td>
</tr>
<tr>
<td>InitSPIBuf</td>
<td>Initializes SPI buffer for transaction.</td>
<td>Figure 20</td>
</tr>
<tr>
<td>LoadSPIByte</td>
<td>Loads the value in the W register into the SPI buffer.</td>
<td>Figure 21</td>
</tr>
<tr>
<td>ExchangeSPI</td>
<td>Initiates the SPI transaction.</td>
<td>Figure 22</td>
</tr>
<tr>
<td>WaitSPIExchange</td>
<td>Waits for the SPI transaction to be completed.</td>
<td>Figure 23</td>
</tr>
<tr>
<td>LoadSPIZeros</td>
<td>Clears the value in the SPI buffer.</td>
<td>Figure 24</td>
</tr>
</tbody>
</table>
CONCLUSION

The MCP2510 offers a simple method to interface a CAN network in order to maximize the transmitting and receiving of data via the CAN Bus. This efficient method allows a wide variety of I/O devices to be connected to the network using a node board. An advantage in utilizing this type of system is the ability to monitor several node boards at any given time. If an error occurs, it is detected and re-transmitted over the bus line until the receiver acknowledges the message. Another advantage is that several node Boards can work from one bus line rather than using a large wiring harness that connects to a main control panel. Our design demonstrates a way to implement a simple input pressure switch connected to a node board, along with a visual light source to display the value in terms of brightness. By this example, several uses for different types of inputs and outputs can be implemented by using the basic techniques from this design.

Table 6 shows the resource requirements for the major functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Memory</th>
<th>Instruction Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assemble Message</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>Transmit</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>Receive</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>Disassemble Message</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>Test Error Conditions</td>
<td>TBD</td>
<td>TBD</td>
</tr>
</tbody>
</table>

CONTACTING DIVERSIFIED ENGINEERING

Additional information and CAN related products may be acquired from Diversified Engineering. You may contact them by either calling:
(203) 799-7875
or by visiting their web site:
www.DiversifiedEngineering.net
FIGURE 4: MAIN PROGRAM LOOP (Main)

Main

Initialize System from POR (Hardstart)

No

Transmit CAN Message?

Yes

Reload Counter

No

Transmission Turned off?

Yes

Check CAN Message (CheckCANMsg)

No

New CAN Message Received?

Yes

Parse the Message (CheckCANMsg)

No

Message Type = ‘1’?

Yes

Read A/D Reference (ReadA2D)

No (≠ ’0’)

Read A/D Pressure (Read3201)

Use TX DIP address for Transmission

Assemble new Message using new Source and ID

Wait for Pending Messages (WaitANDeq2)

Output CAN Message
FIGURE 5: HARDSTART (Hardstart)

- Initialize PICmicro® MCU (Init)
- Deselect devices on SPI™ bus
- Read the DIP switches
- Create ID for Transmit and Receive
- Initialize PICmicro SPI Port (InitSPIPort)
- Delay 28 msec (for MCP2510)
- Initialize MCP2510 Registers (Init2510)
- Return
FIGURE 6: INITIALIZE PICmicro® (Init)

Init

- Clear Peripheral Interrupt bits
- Clear GPR registers in Bank 0
- Clear GPR registers in Bank 1
- Turn on A/D Conversion
- Initialize Ports (A, B, and C)
- Configure Timer
- Initialize PWM
- Initialize Ports (A, B, and C)
- Enable Peripherals only
- Return
AN212

FIGURE 7: SETUP SPI™ PORT (InitSPIPort)

InitSPIPort

- Disable SPI™ module
- Configure as Master Mode SPI
- Enable SPI
- Clear SPI Interrupt flag (SSPIF)
- SPP Enable Bank 1
- Return
FIGURE 8: SETUP MCP2510 REGISTERS (Init2510)

1. **Reset MCP2510 Registers**
   - (Reset2510)

2. **Set clock output prescaler to divide by 4**

3. **Write Data in Register using Mask**
   - (BitMod2510)

4. **Set Physical Layer Configuration**

5. **Configure Receive Buffer 0**
   - Mask and Filters

6. **Configure Receive Buffer 1**
   - Mask and Filters

7. **Configure Filter 2 to match ID from DIP switch settings**

8. **Configure Filter 3 to match ID from DIP switch settings**

9. **Disable MCP2510 Interrupts**

10. **Write W Register to MCP2510 Register**
    - (Wrt2510Reg)

11. **Set Normal Mode**
    - (SetNormalMode)

12. **Return**
FIGURE 9: READ A/D PRESSURE (Read3201)

1. **Read3201**
2. Initialize SPI™ Buffer
   - (InitSPIBuf)
3. Clear the SPI buffer
   - (LoadSPIZeros)
4. Initiate SPI Transaction
5. Load first byte to begin Exchange
6. Wait for SPI completion
   - (WaitSPIExchange)
7. Shift byte right one bit
   - (to remove extra bit)
8. Clear upper 4 bits
9. Return
FIGURE 10: READ A/D REFERENCE (ReadA2D)

1. Configure A/D input channel
2. Turn on A/D
3. Start A/D conversion
4. Check if A/D conversion is complete?
   - Yes: Convert result to 12 bits
   - No: Repeat steps 1-3
5. Return
FIGURE 11: WAIT FOR PENDING MESSAGES (WaitANDeqZ)

1. Put Address in Register
2. Save Address in Register
3. Read MCP2510 register at address pointed to in W Register (Rd2510Reg)
4. Address Match?
   - No
   - Yes
5. Return

Which Address, Which Register
What matches what
FIGURE 12: CHECK CAN MESSAGE (CheckCANMsg)

CheckCANMsg

New CAN Message Received?

Yes

Return

No

Message Pending?

Yes

Calculate Receive Buffer location

Get Data Byte

Read MCP2510 register at address pointed to in W Register (Rd2510Reg)

Save Data

Calculate Destination Buffer location

Store Data in Buffer

Clear Receive Buffer

Signal Data Pending

Return

No More Data?

Yes

Get ID out of Message Source

Get Number of Bytes of Data
FIGURE 13: PARSE THE MESSAGE (ParseCAN)

ParseCAN

Received ID Base? No

Yes

Extract Data (12-bits) from Message

Convert Data to 8-bit PWM output (by masking lower 4 bits)

Output PWM (OutputPWM)

Data ≥ 0x800? No

Yes

Extract Data (12-bits) from Message

Turn RX LED on

Return

Turn RX LED off
FIGURE 14:  RESET MCP2510 REGISTERS (Reset2510)

1. Reset2510
2. Initialize SPI™ Buffer ([InitSPIBuf])
3. Reset MCP2510
4. Load W Register into SPI Buffer ([LoadSPIByte])
5. Initialize SPI Buffer ([InitSPIBuf])
6. Initiate SPI Transaction ([ExchangeSPI])
7. Wait for SPI Completion ([WaitSPIExchange])
8. Return
FIGURE 15: WRITE DATA IN REGISTER USING MASK (BitMod2510)

BitMod2510

Save Message Address

Initialize SPI™ Buffer
(InitSPIBuf)

Send MCP2510
Modify Register Command

Load W Register into SPI Buffer
(LoadSPIByte)

Load Address into
W Register

Load W Register into SPI Buffer
(LoadSPIByte)

Load Mask into
W Register

Load W Register into SPI Buffer
(LoadSPIByte)

Load Data into
W Register

Load W Register into SPI Buffer
(LoadSPIByte)

Initiate SPI Transaction
(ExchangeSPI)

Wait for SPI Completion
(WaitSPIExchange)

Return
FIGURE 16: WRITE BYTE IN MCP2510 REGISTER IN W (Wrt2510Reg)

1. Save Message Address
2. Initialize SPI™ Buffer (InitSPIBuf)
3. Send MCP2510 Write Register Command
4. Load W Register into SPI Buffer (LoadSPIByte)
5. Load Address into W Register
6. Load W Register into SPI Buffer (LoadSPIByte)
7. Load Data into W Register
8. Load W Register into SPI Buffer (LoadSPIByte)
9. Initiate SPI Transaction (ExchangeSPI)
10. Wait for SPI Completion (WaitSPIExchange)
11. Return
FIGURE 17: SET NORMAL MODE (SetNormalMode)

```
SetNormalMode

Configure Mask and Data Addresses

Write data in CANCNTL Register (BitMod2510)

Read MCP2510 register at address pointed to in W Register (Rd2510Reg)

Value ANDed 0x0E = 0?

Yes

No

Return
```
FIGURE 18: READ REGISTER ADDRESS IN W (Rd2510Reg)

1. Save Message Address
2. Initialize SPI™ Buffer (InitSPIBuf)
3. Send MCP2510 Read Register Command
4. Load W Register into SPI Buffer (LoadSPIByte)
5. Load Address into W Register
6. Load W Register into SPI Buffer (LoadSPIByte)
7. Clears the value in the SPI Buffer (LoadSPIZeros)
8. Initiate SPI Transaction (ExchangeSPI)
9. Wait for SPI Completion (WaitSPIExchange)
10. Return
**FIGURE 19: OUTPUT PWM (OutputPWM)**

1. **Extract Data (8-bits) from Message**
2. **Load Data into PWM register**
3. **Turn on PWM output**
4. **Load upper 8-bits into CCPR1 register**
5. **Return**

**FIGURE 20: INITIALIZE SPI BUFFER (InitSPIBuf)**

1. **Load FSR with start address of SPI Buffer**
2. **Return**

**FIGURE 21: LOAD BYTE IN W TO SPI BUFFER (LoadSPIByte)**

1. **Increment FSR Register**
2. **Return**
FIGURE 22: INITIATE SPI TRANSACTION (ExchangeSPI)

ExchangeSPI

Get Number of Bytes to Exchange

Bytes $\geq 0$?

Yes

No

Load Number of Bytes in Buffer

Load Byte to begin Exchange

Send Byte(s)

Return

FIGURE 23: WAIT FOR SPI COMPLETION (WaitSPIExchange)

WaitSPIExchange

SPI communication completed?

No

Yes

Return
FIGURE 24: LOAD NUMBER OF ZEROS IN W TO SPI BUFFER (LoadSPIZeros)

LoadSPIZeros

W Register = 0?

Yes

Clear Address pointed to by FSR

Increment FSR

Add 0xFF to W Register

No

W Register = 0?

Yes

Return
APPENDIX A: SCHEMATICS

FIGURE 1: CAN NODE BOARD
FIGURE 2: ANALOG INPUT BOARD
## APPENDIX B: BILL OF MATERIALS

### TABLE 1: BILL OF MATERIALS (PAGE 1 OF 2)

<table>
<thead>
<tr>
<th>Qty</th>
<th>Reference</th>
<th>Description</th>
<th>Mfg Part</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PC BRD</td>
<td>PC BROAD</td>
<td>110501 A/A</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>C9-C12</td>
<td>CAP, MLC, 50V, NPO, 5%, 0805, 18PF</td>
<td>MCH215A180JK</td>
<td>ROHM</td>
</tr>
<tr>
<td>1</td>
<td>C8</td>
<td>CAP, MLC, 50V, X7R, 10%, 0805, 1000PF</td>
<td>MCH215C102KK</td>
<td>ROHM</td>
</tr>
<tr>
<td>5</td>
<td>C1, C3-C6</td>
<td>CAP, MLC, 50V, X7R, 10%, 0805, .1UF</td>
<td>C0805C104K5RAC7210</td>
<td>KEMET</td>
</tr>
<tr>
<td>1</td>
<td>C2</td>
<td>CAP, ELEC, 16V, 20%, VA-B, 10UF</td>
<td>EEV-HB1C100R</td>
<td>PANASONIC</td>
</tr>
<tr>
<td>1</td>
<td>D1</td>
<td>DIODE, 200V, 1A, DO-214BA</td>
<td>GF1D</td>
<td>GI</td>
</tr>
<tr>
<td>1</td>
<td>(J4)</td>
<td>DIP SHUNT, GOLD, 2 POS.</td>
<td>SNT-100-BK-G</td>
<td>SAMTEC</td>
</tr>
<tr>
<td>1</td>
<td>GND</td>
<td>TEST POINT</td>
<td>2305-3-00-44-0000070</td>
<td>MILLMAX</td>
</tr>
<tr>
<td>1</td>
<td>(U6)</td>
<td>SOCKET, 250V, DIP18, LIN BER COP</td>
<td>2-641611-1</td>
<td>AMP</td>
</tr>
<tr>
<td>1</td>
<td>(U4)</td>
<td>SOCKET, 250V, ROUND PIN, 28PIN, F</td>
<td>110-99-328-41-001</td>
<td>MILLMAX</td>
</tr>
<tr>
<td>2</td>
<td>J1, J2</td>
<td>CONN, MTA100, 4 PIN MALE</td>
<td>640456-4</td>
<td>AMP</td>
</tr>
<tr>
<td>1</td>
<td>J4</td>
<td>CONN, .025SQX.1, 250V, 3A, 2PIN, M, .230</td>
<td>*TSW-102-07-T-S</td>
<td>SAMTEC</td>
</tr>
<tr>
<td>1</td>
<td>J3</td>
<td>CONN, SINGLE ROW, RA, 15POS, F</td>
<td>CES-115-02-T-S-RA</td>
<td>SAMTEC</td>
</tr>
<tr>
<td>5</td>
<td>RX, TX, ICD, CAN-IN, U5</td>
<td>DO NOT INSTALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>R6, R7</td>
<td>RES, .1W, 5%, 0805, 1K</td>
<td>MCR10J102</td>
<td>ROHM</td>
</tr>
<tr>
<td>4</td>
<td>R2-R5</td>
<td>RES, .1W, 5%, 0805, 10K</td>
<td>MCR10J103</td>
<td>ROHM</td>
</tr>
<tr>
<td>1</td>
<td>R8</td>
<td>RES, .1W, 5%, 0805, 120OHM</td>
<td>MCR10J121</td>
<td>ROHM</td>
</tr>
<tr>
<td>1</td>
<td>R1</td>
<td>RES, .1W, 5%, 0805, 47K</td>
<td>MCR10J473</td>
<td>ROHM</td>
</tr>
<tr>
<td>1</td>
<td>S1</td>
<td>SWITCH, PUSH, MOM, 6MM</td>
<td>TL1105EF250</td>
<td>E-SWITCH</td>
</tr>
<tr>
<td>1</td>
<td>S4</td>
<td>SWITCH, DIP8, SPST, 50V, 100MA, 4.POS</td>
<td>206-4ST</td>
<td>CTS</td>
</tr>
<tr>
<td>1</td>
<td>U6</td>
<td>IC, CAN_CONTROLLER, DIP18, CMOS, I</td>
<td>MCP2510-I/P</td>
<td>MICROCHIP TECHNOLOGY INC.</td>
</tr>
<tr>
<td>1</td>
<td>U4</td>
<td>IC, MICRO, DIP28, FLASH</td>
<td>PIC16F873-20/P</td>
<td>MICROCHIP TECHNOLOGY INC.</td>
</tr>
<tr>
<td>1</td>
<td>U7</td>
<td>IC, CAN_INTERFACE, SO8</td>
<td>MCP2551</td>
<td>MICROCHIP TECHNOLOGY INC.</td>
</tr>
<tr>
<td>1</td>
<td>Q5</td>
<td>REG, 5V, .1A, SOT-89</td>
<td>NJM78L05UA</td>
<td>NJR</td>
</tr>
<tr>
<td>1</td>
<td>X2</td>
<td>CRYSTAL, 4MHZ, CSM-7</td>
<td>ECS-40-20-5P</td>
<td>ECS</td>
</tr>
<tr>
<td>1</td>
<td>X1</td>
<td>CRYSTAL, PARALLEL, CSM-7, 16.00MHZ</td>
<td>ECS-160-20-5P</td>
<td>ECS</td>
</tr>
<tr>
<td>1</td>
<td>110500</td>
<td>OSDA LABOR</td>
<td>110500</td>
<td>OSDA</td>
</tr>
</tbody>
</table>
### TABLE 7: BILL OF MATERIALS (PAGE 2 OF 2)

<table>
<thead>
<tr>
<th>Qty</th>
<th>Reference</th>
<th>Description</th>
<th>Mfg Part</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PCB</td>
<td>PCB, CAN-NE ANALOG IN</td>
<td>110511 A/A</td>
<td>PCB</td>
</tr>
<tr>
<td>2</td>
<td>C6, C10</td>
<td>CAP, MLC, 16V, Y5V, +80-20, 0805, 1UF</td>
<td>MCH213F105ZP</td>
<td>ROHM</td>
</tr>
<tr>
<td>2</td>
<td>C5, C13</td>
<td>CAP, MLC, 50V, X7R, 10%, 0805, .01UF</td>
<td>MCH215C103KK</td>
<td>ROHM</td>
</tr>
<tr>
<td>7</td>
<td>C1-C4, C8, C11, C12</td>
<td>CAP, MLC, 50V, X7R, 10%, 0805, .01UF</td>
<td>C0805C104K5RAC7210</td>
<td>KEMET</td>
</tr>
<tr>
<td>1</td>
<td>C7</td>
<td>CAP, ELEC, 16V, 20%, VA-B, 10UF</td>
<td>EEV-HB1C100R</td>
<td>PANASONIC</td>
</tr>
<tr>
<td>1</td>
<td>D1</td>
<td>DIODE, 200V, 1A, DO-214BA</td>
<td>GF1D</td>
<td>GI</td>
</tr>
<tr>
<td>1</td>
<td>TB1</td>
<td>TERM BLOCK, TH, 5POS, .1PITCH</td>
<td>1725685</td>
<td>PHOENIX</td>
</tr>
<tr>
<td>(J8)</td>
<td></td>
<td>SHUNT, DUAL, TIN, 2X2</td>
<td>MNT-102-BK-T</td>
<td>SAMTEC</td>
</tr>
<tr>
<td>1</td>
<td>J8</td>
<td>CONN, DUALROW, .025SQ, 3A, 8POS</td>
<td>TSW-104-07-T-D</td>
<td>SAMTEC</td>
</tr>
<tr>
<td>1</td>
<td>J3</td>
<td>CONN, SINGLE_ROW, RA, 15POS, M</td>
<td>TSW-115-08-T-S-RA</td>
<td>SAMTEC</td>
</tr>
<tr>
<td>2</td>
<td>XDUCE1, R11</td>
<td>DO NOT INSTALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>R8, R9</td>
<td>RES, .1W, 1%, 0805, 1K</td>
<td>MCR10F1001</td>
<td>ROHM</td>
</tr>
<tr>
<td>5</td>
<td>R1, R3, R5, R6, R10</td>
<td>RES, .1W, 1%, 0805, 10.0K</td>
<td>MCR10F1002</td>
<td>ROHM</td>
</tr>
<tr>
<td>2</td>
<td>R4, R7</td>
<td>RES, .1W, 1%, 0805, 30.1K</td>
<td>MCR10F3012</td>
<td>ROHM</td>
</tr>
<tr>
<td>1</td>
<td>R13</td>
<td>RES, 1/10W, 1%, 0805, 39.2K</td>
<td>MCR10F3922</td>
<td>ROHM</td>
</tr>
<tr>
<td>1</td>
<td>R12</td>
<td>RES, 1/10W, 1%, 0805, 61.9K</td>
<td>MCR10F6192</td>
<td>ROHM</td>
</tr>
<tr>
<td>1</td>
<td>RP3</td>
<td>RES, POT, 1/2W, 10%, 20K</td>
<td>3299Y-203</td>
<td>BOURNS</td>
</tr>
<tr>
<td>1</td>
<td>RP1</td>
<td>RES, POT, 1/2W, 10%, 50K</td>
<td>3299Y-503</td>
<td>BOURNS</td>
</tr>
<tr>
<td>1</td>
<td>R2</td>
<td>THERMISTOR, tc-4.6, 10k, 1s, .1&quot;, disc, TH</td>
<td>ERT-D2FHL103S</td>
<td>PANASONIC</td>
</tr>
<tr>
<td>1</td>
<td>U4</td>
<td>IC, MEMORY, DIP8, E2PROM, 512X8</td>
<td>25C040/P</td>
<td>MICROCHIP TECHNOLOGY INC.</td>
</tr>
<tr>
<td>1</td>
<td>U3</td>
<td>IC, ADC, DIP8, 12BIT, CMOS, +/-2LSB</td>
<td>MCP3201-C/P</td>
<td>MICROCHIP TECHNOLOGY INC.</td>
</tr>
<tr>
<td>2</td>
<td>U1, U2</td>
<td>IC, CMOS, DIPS8, DUAL_OPAMP, LOW_POWER</td>
<td>MCP602/P</td>
<td>MICROCHIP TECHNOLOGY INC.</td>
</tr>
<tr>
<td>1</td>
<td>REF1</td>
<td>REG, VOLTAGE_REFERENCE, 4_096V, SO8, F</td>
<td>REF198FA</td>
<td>ANALOG_DE</td>
</tr>
<tr>
<td>1</td>
<td>VR1</td>
<td>REG, 5V, .1A, SOT-89</td>
<td>NJM78L05UA</td>
<td>NJR</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>CAN-NET ANALOG IN</td>
<td>110510</td>
<td>OSDA</td>
</tr>
</tbody>
</table>