Smart Sensor CAN Node Using the MCP2515 and PIC16F876

Author: Mike Stanczyk
Diversified Engineering, Inc.

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. Meanwhile, the receivers are set up 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 this scheme, 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 lost arbitration will automatically terminate their transmission and switch to Receive mode. After the CAN bus enters an Idle state, these nodes attempt to retransmit. If the node did 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.

MCP2515 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 an LED. Table 1 gives the part numbers for these components.

TABLE 1: COMPONENT PART NUMBERS

Manufacturer	Component	Part Number
Diversified Engineering	CAN-NET Analog Input Node Kit	905190
Motorola®	Pressure Transducer	MPX2010DP

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 MCP2515 CAN controller is the heart of the CAN interface. It handles all of the transmitting and receiving of message packets that contain useful information for other nodes on the network via the CAN bus. The MCP2515 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.

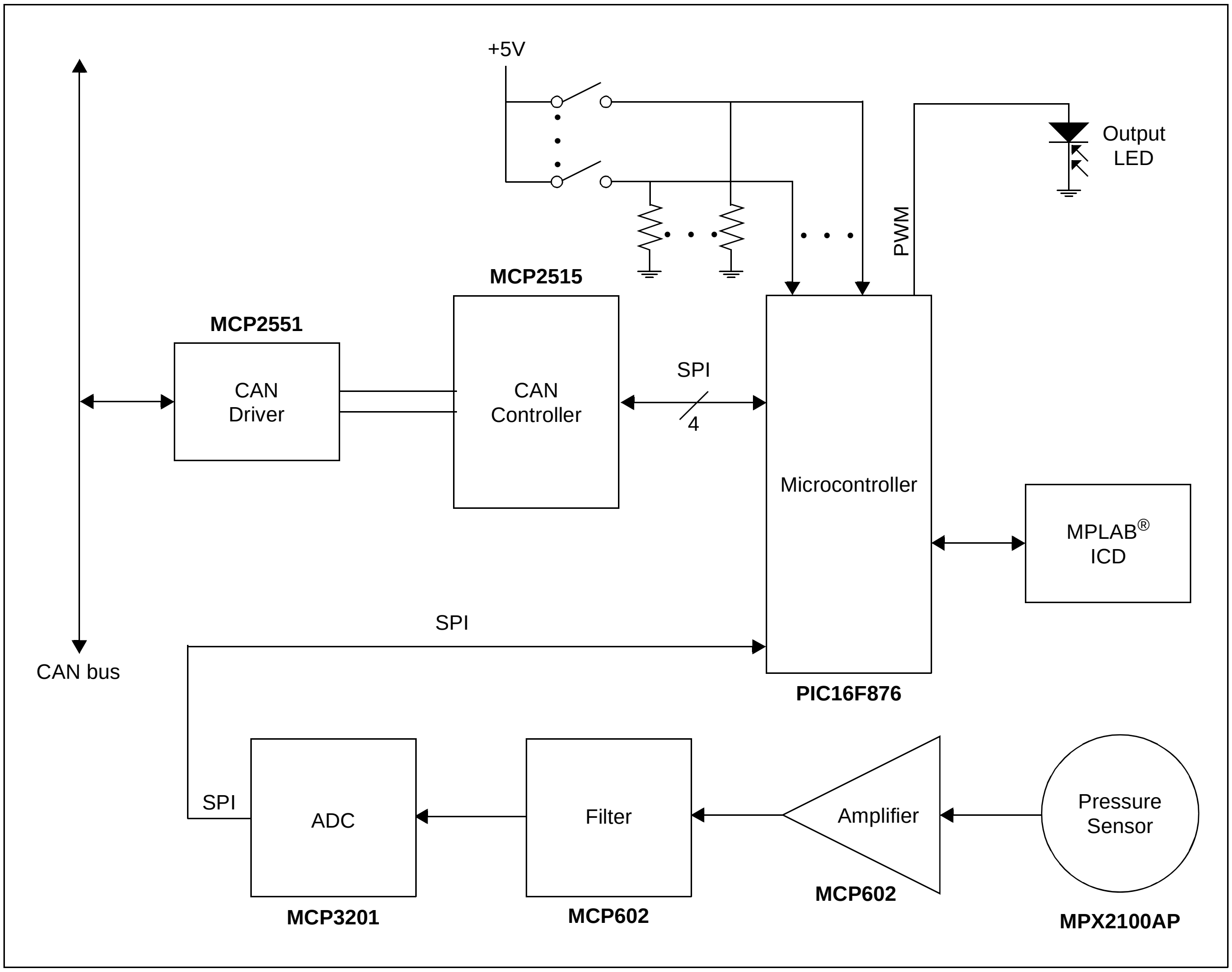
HARDWARE OVERVIEW

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: “Schematics”**.

MCP2515 CAN Controller

The high-level design of this system is shown in Figure 1. The concept is to enable the MCP2515 CAN controller, the PIC16F876 microcontroller and the MCP3201 ADC to efficiently communicate among each other, utilizing the SPI. The MCP2515 handles the lower level protocols.

FIGURE 1: BLOCK DIAGRAM OF THE CAN NODE BOARD

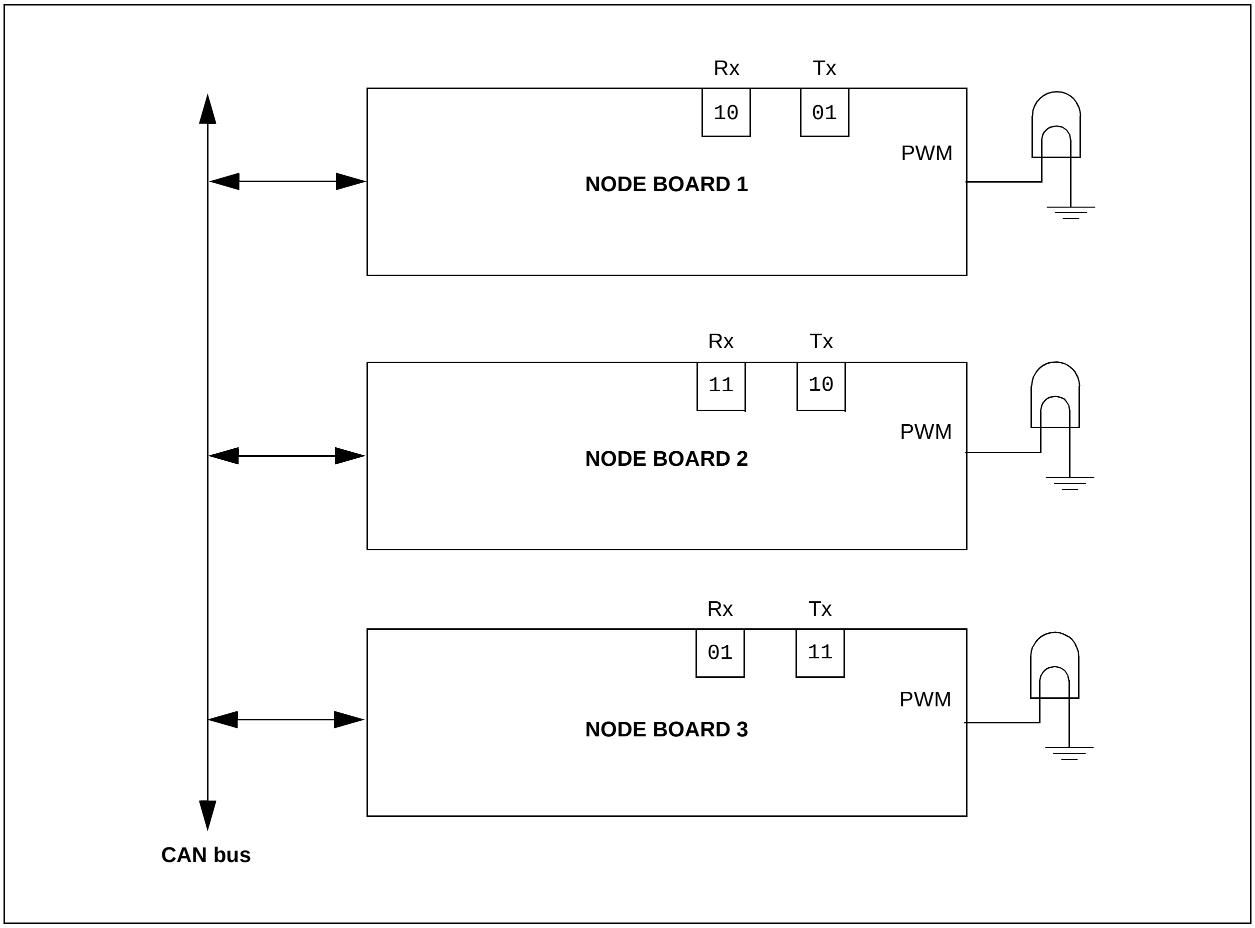


The PIC16F876 microcontroller stores the program in memory and constantly polls the MCP3201 ADC, along with the reference A/D.

In the main loop of the program, a variable is toggled. When the value of the variable is a logic '0', the PIC® device reads the pressure sensor, and when the value of the variable is a logic '1', the PIC® device reads the reference A/D. The microcontroller also reads the settings of the input switches.

The first two (of four) switches tell 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 3, illustrates three Node Boards on a CAN bus, and they are 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 each node 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, and 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, which 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

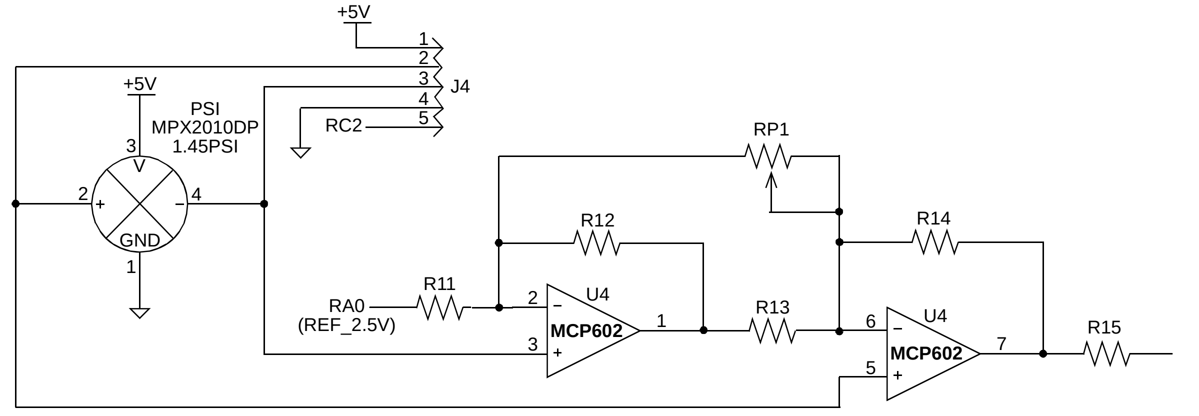
$$Gain = 1 + \frac{R14}{R13} + 2\frac{R13}{RP1}$$

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 (when there is intake) and positive pressure (when there is outtake). 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 (“expenses”) 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



Component	Value	Tolerance
R11	30.1 kΩ	1%
R12	10.0 kΩ	1%
R13	10.0 kΩ	1%
R14	30.1 kΩ	1%
R15	1.0 KΩ	1%
RP1	0.0 to 50.0 kΩ	N.A.

SOFTWARE OVERVIEW

Programming Style

The code for the Node Board is written in the PIC® 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 PIC® 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 MCP2515 CAN chip are in the file `canli.asm` and the definitions of the MCP2515 registers are in `mcp2515.inc`. The common macros are in `macros16.inc`.

SPI Communications

Communications from a device on the node (such as a microcontroller) to the MCP2515 are through the SPI bus. The PIC® 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 MCP2515. The bytes received from the MCP2515 replace the bytes that were sent from the software buffer, so that after the string has been sent, the buffer will contain the bytes received from the MCP2515. All communications with the MCP2515 are 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 following setting in Table 2 for receiving:

TABLE 2: RECEIVING SETTINGS FOR THE NODE BOARD

Register	Value
RxMask0	0xFFF
RxMask1	0xFFE
RxFilter0	0xFFF
RxFilter1	0xFFF
RxFilter2	0xn00 (1)
RxFilter3	0xn10
RxFilter4	0xFFF
RxFilter5	0xFFF

Note 1: 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 3: DIP SWITCH ID SETTINGS FOR RECEIVING

DIP #1	DIP #2	ID
0	0	0x000
0	1	0x100
1	0	0x200
1	1	0x300

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 the 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 of 0 to 4095. Note that the A/D measurement is 8 bits, which is then shifted by 4 bits before transmission; hence, 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 4: DIP SWITCH ID SETTINGS FOR TRANSMITTING

DIP #3	DIP #4	ID
0	0	All transmissions are disabled
0	1	0x100
1	0	0x200
1	1	0x300

The MCP2515 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 ink you to the page for that function.

TABLE 5: SOFTWARE FUNCTION DESCRIPTIONS

Function Name	Function Description	Figure Number
Main	This is the main loop of the program.	Figure 4
Hardstart	Does a full initialization of the system.	Figure 5
Init	Initializes the PIC16F87X registers.	Figure 6
InitSPIPort	Initializes the PIC16F87X SPI port.	Figure 7
Init2515	Initializes the MCP2515 registers.	Figure 8
Read3201	Reads the specified register in the MCP3201 (A/D Converter).	Figure 9
ReadA2D	Reads the specified register in the MCP3201 (A/D Converter).	Figure 10
WaitANDeqZ	Waits for pending messages.	Figure 11
CheckCANMsg	Checks for messages in the receive buffer.	Figure 12
ParseCAN	Set up messages for the PWM output.	Figure 13
Reset2515	Resets the MCP2515.	Figure 14
BitMod2515	Modifies the value of a specified bit in the MCP2515.	Figure 15
Wrt2515Reg	Writes the specified register in the MCP2515 (CAN interface).	Figure 16
SetNormalMode	Sets the MCP2515 to normal operating mode.	Figure 17
Rd2515Reg	Reads the specified register in the MCP2515 (CAN interface).	Figure 18
OutputPWM	Loads the PWM Duty Cycle registers with the values in the specified registers.	Figure 19
InitSPIBuf	Initializes SPI buffer for transaction.	Figure 20
LoadSPIByte	Loads the value in the W register into the SPI buffer.	Figure 21
ExchangeSPI	Initiates the SPI transaction.	Figure 22
WaitSPIExchange	Waits for the SPI transaction to be completed.	Figure 23
LoadSPIZeros	Clears the value in the SPI buffer.	Figure 24

CONCLUSION

The MCP2515 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 retransmitted 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 demonstrated 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.

CONTACTING DIVERSIFIED ENGINEERING

Additional information and CAN related products may be acquired from Diversified Engineering, Inc. You may contact them by either calling:

(202) 726-7676

or by visiting their web site:

www.diveng.com

FIGURE 4: MAIN PROGRAM LOOP (Main)

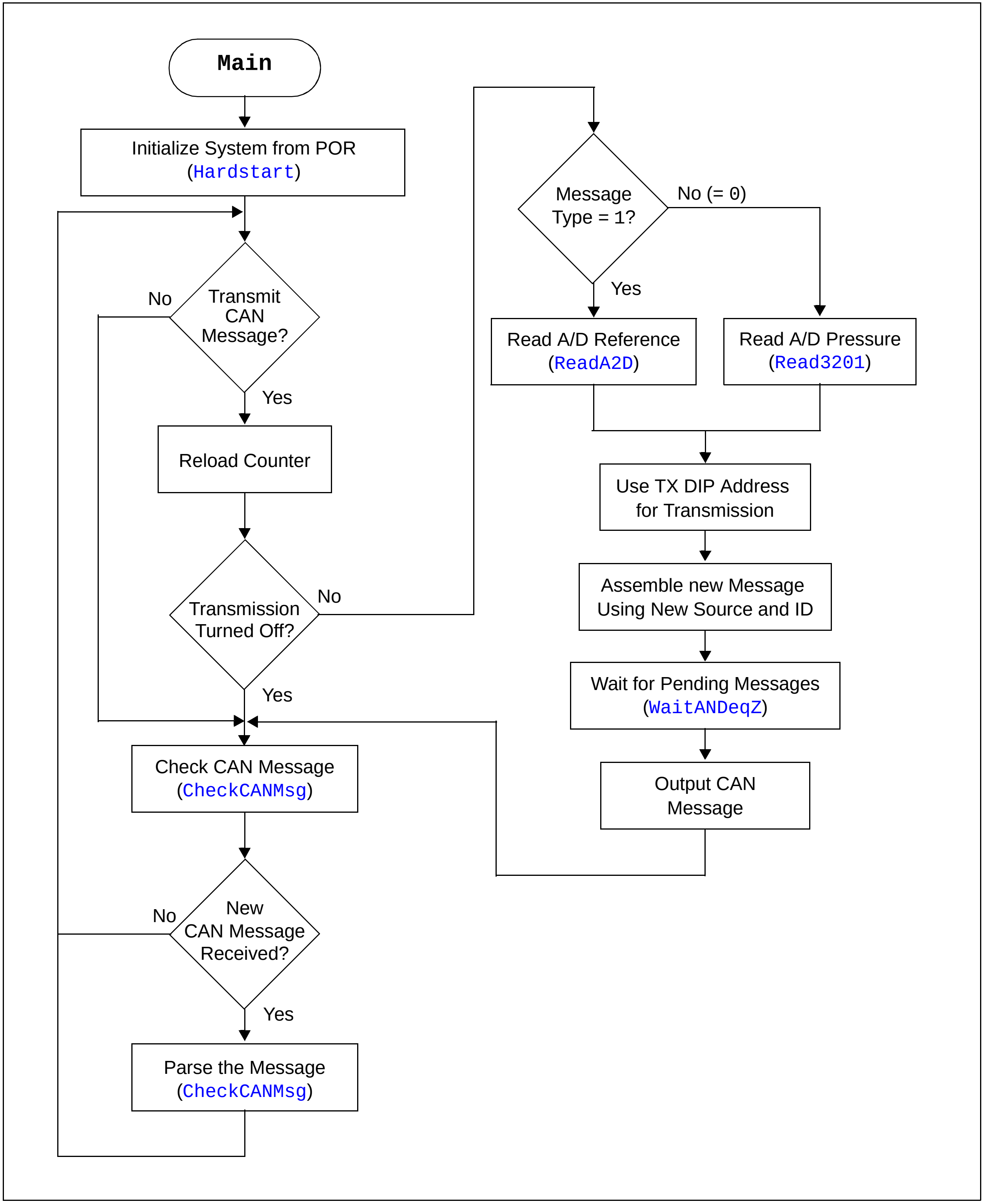

Main
Initialize System from POR (Hardstart)
Transmit CAN Message?
No
Yes
Reload Counter
Transmission Turned Off?
No
Yes
Check CAN Message (CheckCANMsg)
New CAN Message Received?
No
Yes
Parse the Message (CheckCANMsg)
Message Type = 1?
No (= 0)
Yes
Read A/D Reference (ReadA2D)
Read A/D Pressure (Read3201)
Use TX DIP Address for Transmission
Assemble new Message Using New Source and ID
Wait for Pending Messages (WaitANDeqZ)
Output CAN Message

FIGURE 5: HARDSTART (`Hardstart`)

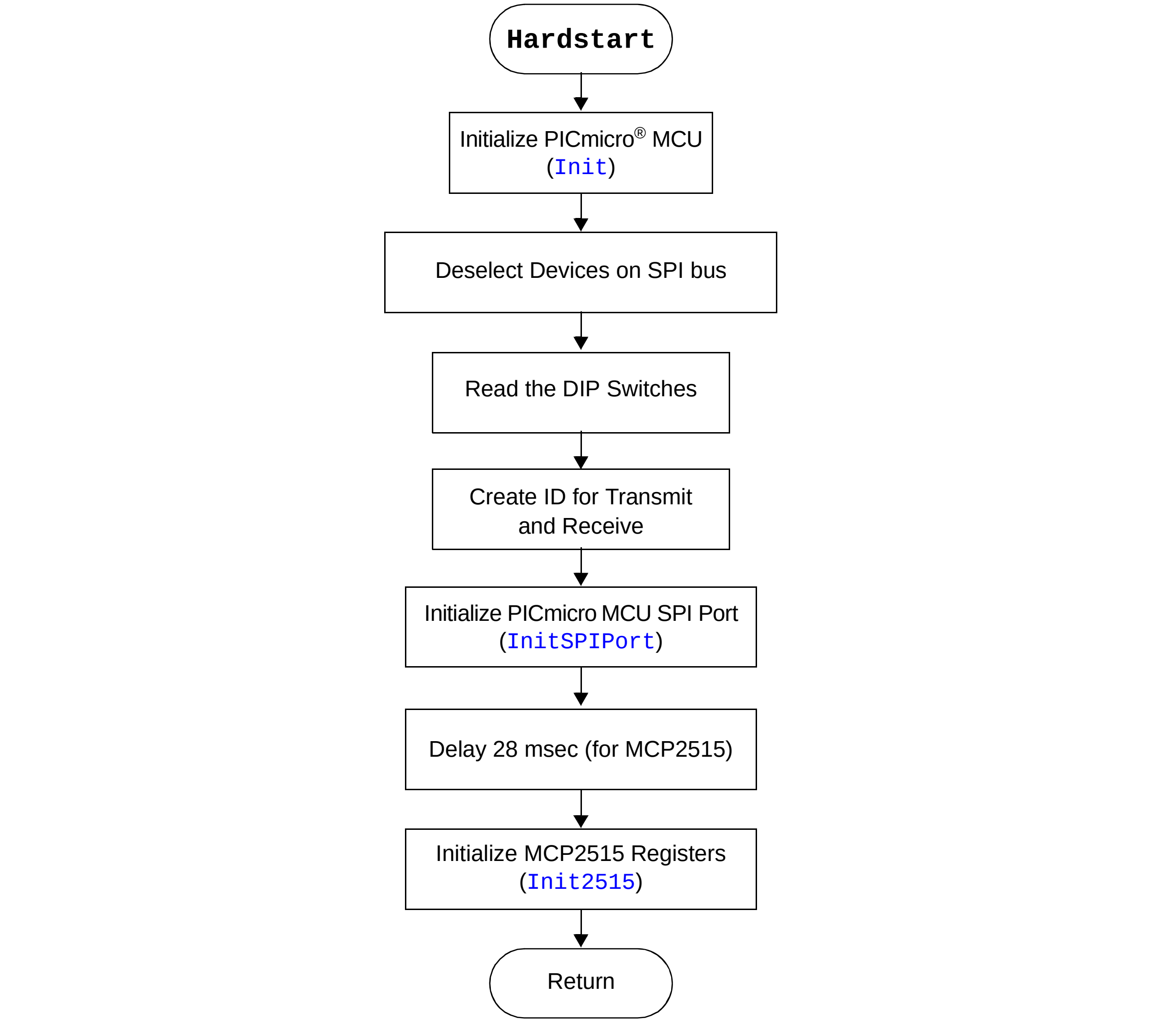


FIGURE 6: INITIALIZE PICmicro® MCU (`Init`)

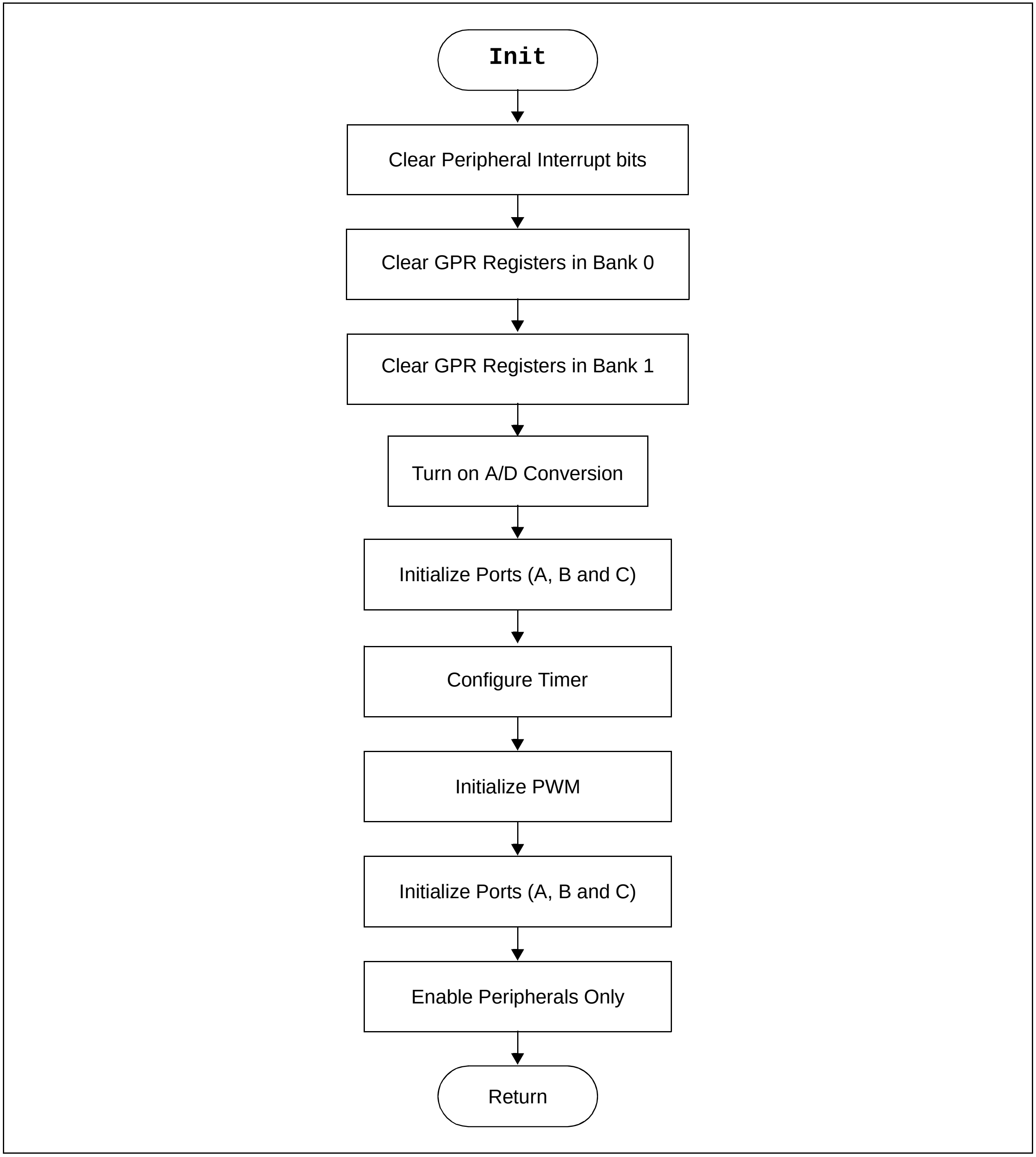

Init
Clear Peripheral Interrupt bits
Clear GPR Registers in Bank 0
Clear GPR Registers in Bank 1
Turn on A/D Conversion
Initialize Ports (A, B and C)
Configure Timer
Initialize PWM
Initialize Ports (A, B and C)
Enable Peripherals Only
Return

FIGURE 7: SETUP SPI PORT (`InitSPIPort`)

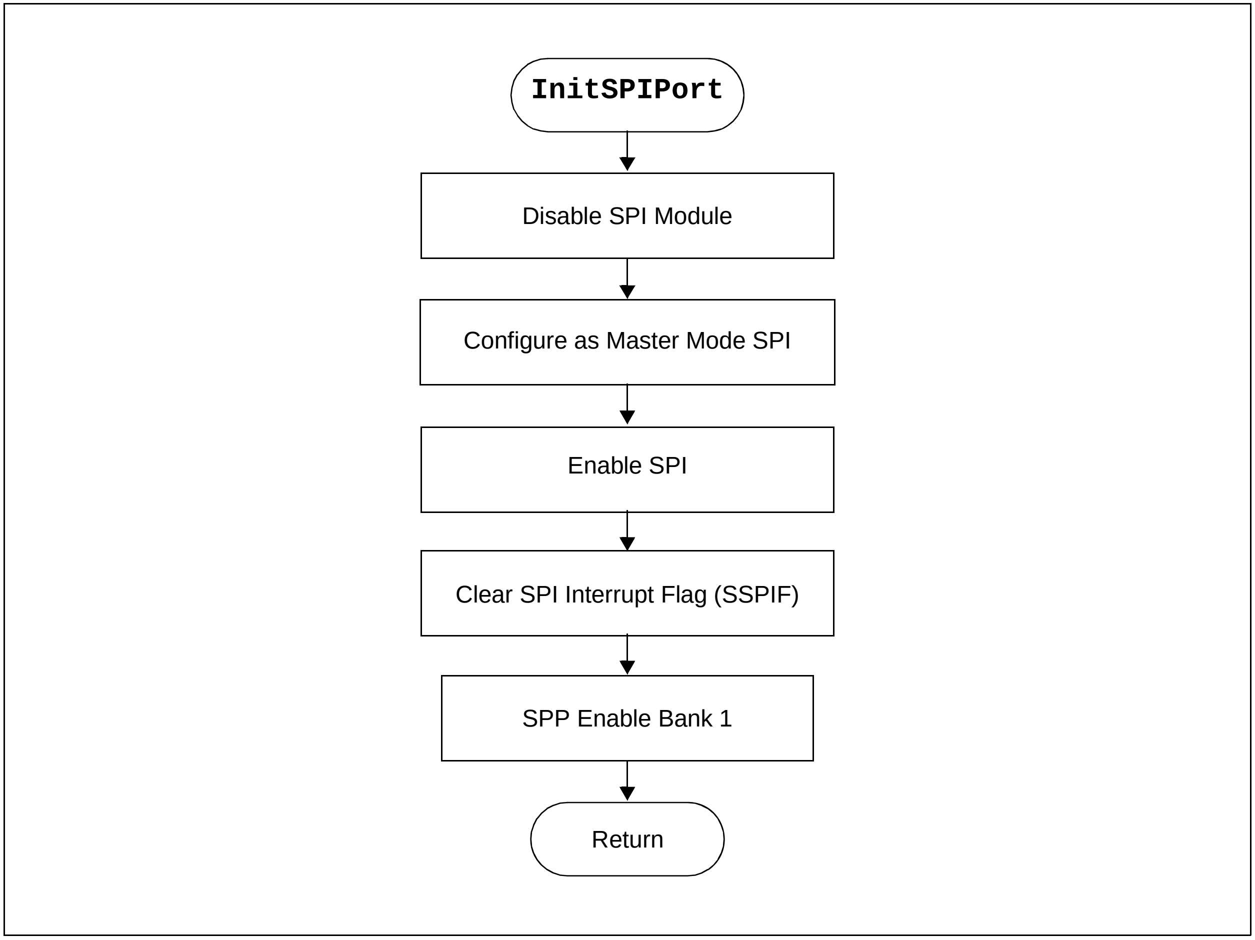


FIGURE 8: SETUP MCP2515 REGISTERS (`Init2515`)

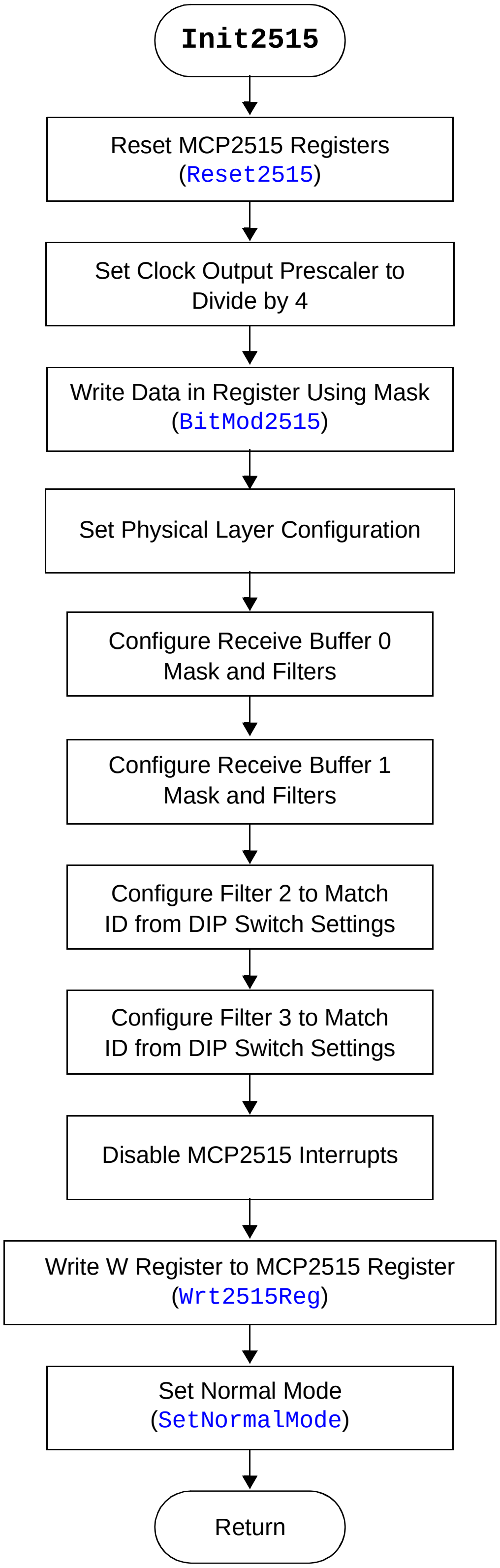

Init2515
Reset MCP2515 Registers (Reset2515)
Set Clock Output Prescaler to Divide by 4
Write Data in Register Using Mask (BitMod2515)
Set Physical Layer Configuration
Configure Receive Buffer 0 Mask and Filters
Configure Receive Buffer 1 Mask and Filters
Configure Filter 2 to Match ID from DIP Switch Settings
Configure Filter 3 to Match ID from DIP Switch Settings
Disable MCP2515 Interrupts
Write W Register to MCP2515 Register (Wrt2515Reg)
Set Normal Mode (SetNormalMode)
Return

FIGURE 9: READ A/D PRESSURE (Read3201)

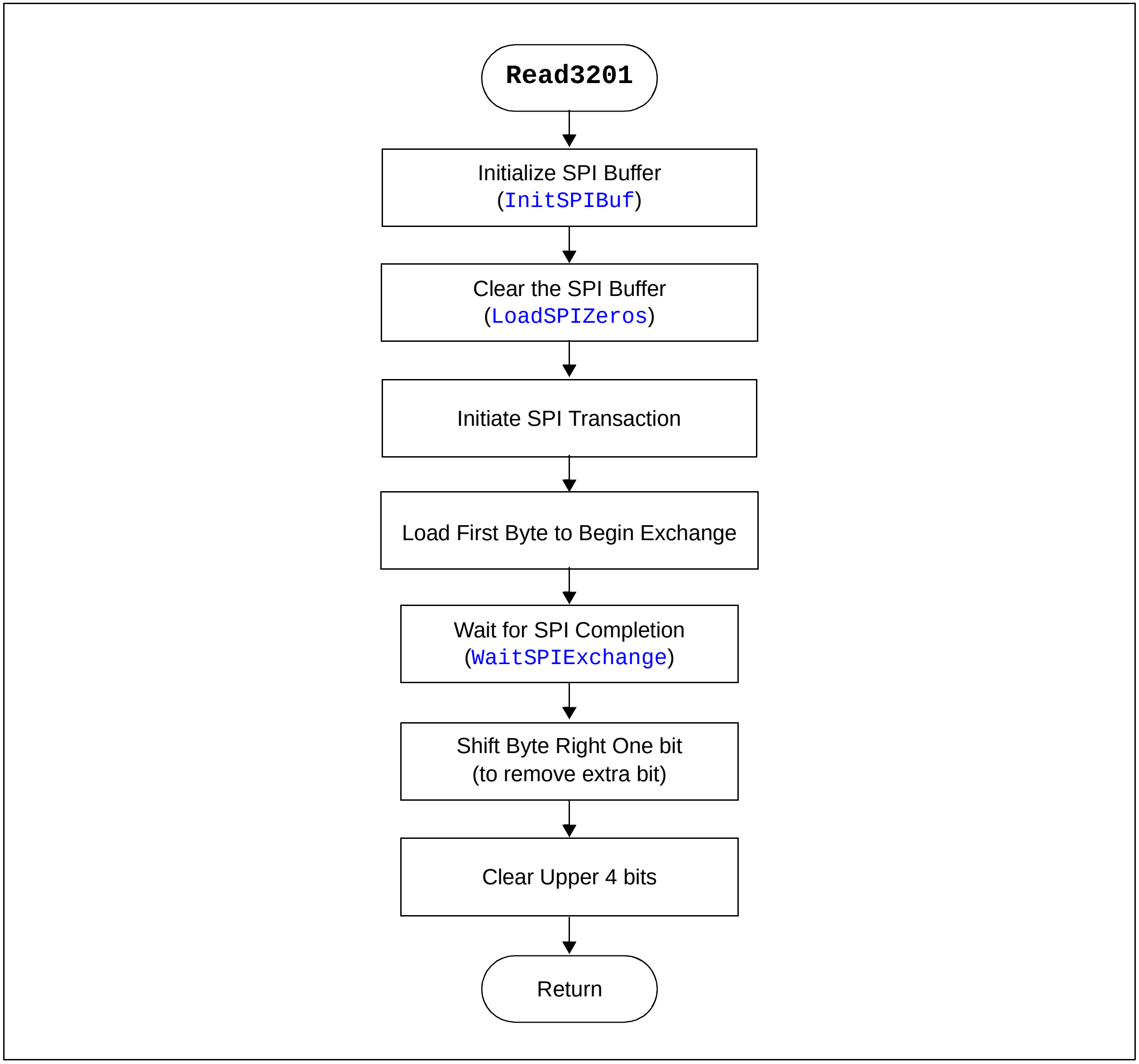

Read3201
Initialize SPI Buffer (InitSPIBuf)
Clear the SPI Buffer (LoadSPIZeros)
Initiate SPI Transaction
Load First Byte to Begin Exchange
Wait for SPI Completion (WaitSPIExchange)
Shift Byte Right One bit (to remove extra bit)
Clear Upper 4 bits
Return

FIGURE 10: READ A/D REFERENCE (ReadA2D)

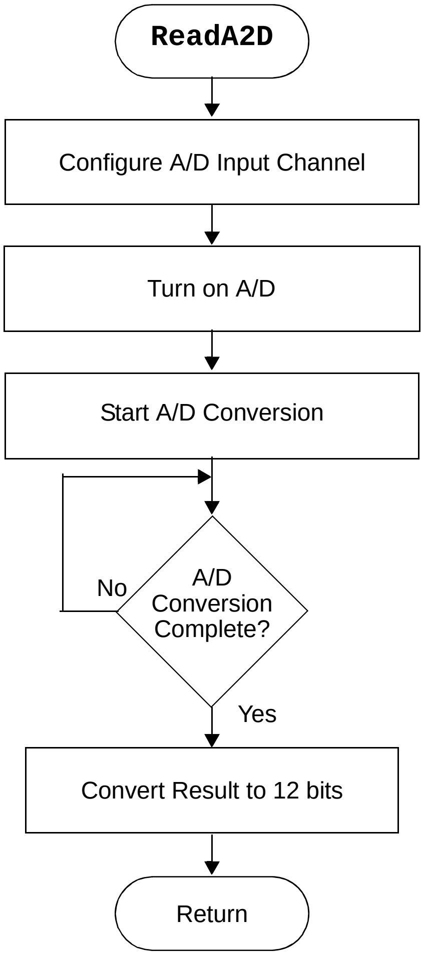


FIGURE 11: WAIT FOR PENDING MESSAGES (WaitANDeqZ)

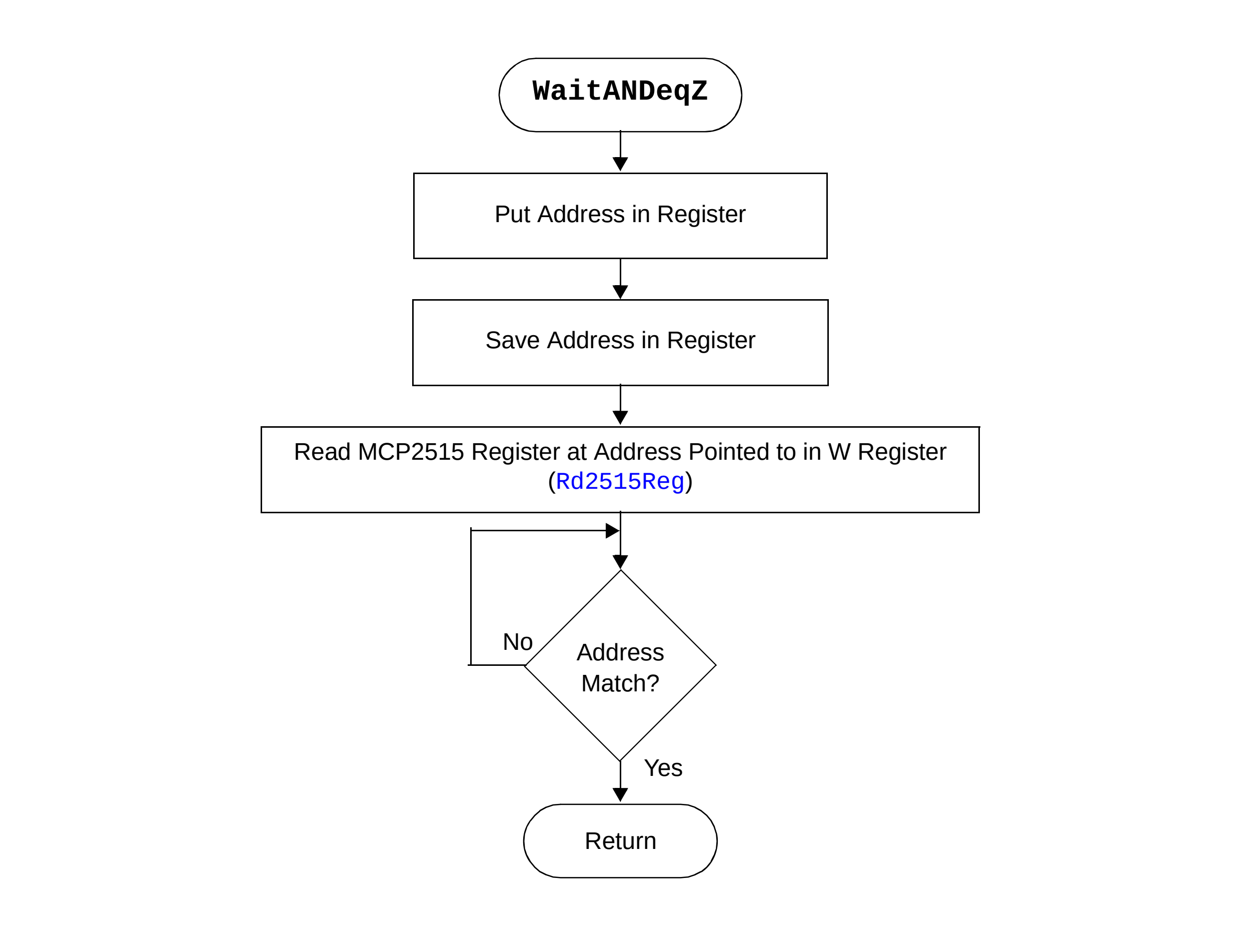


FIGURE 12: CHECK CAN MESSAGE (CheckCANMsg)

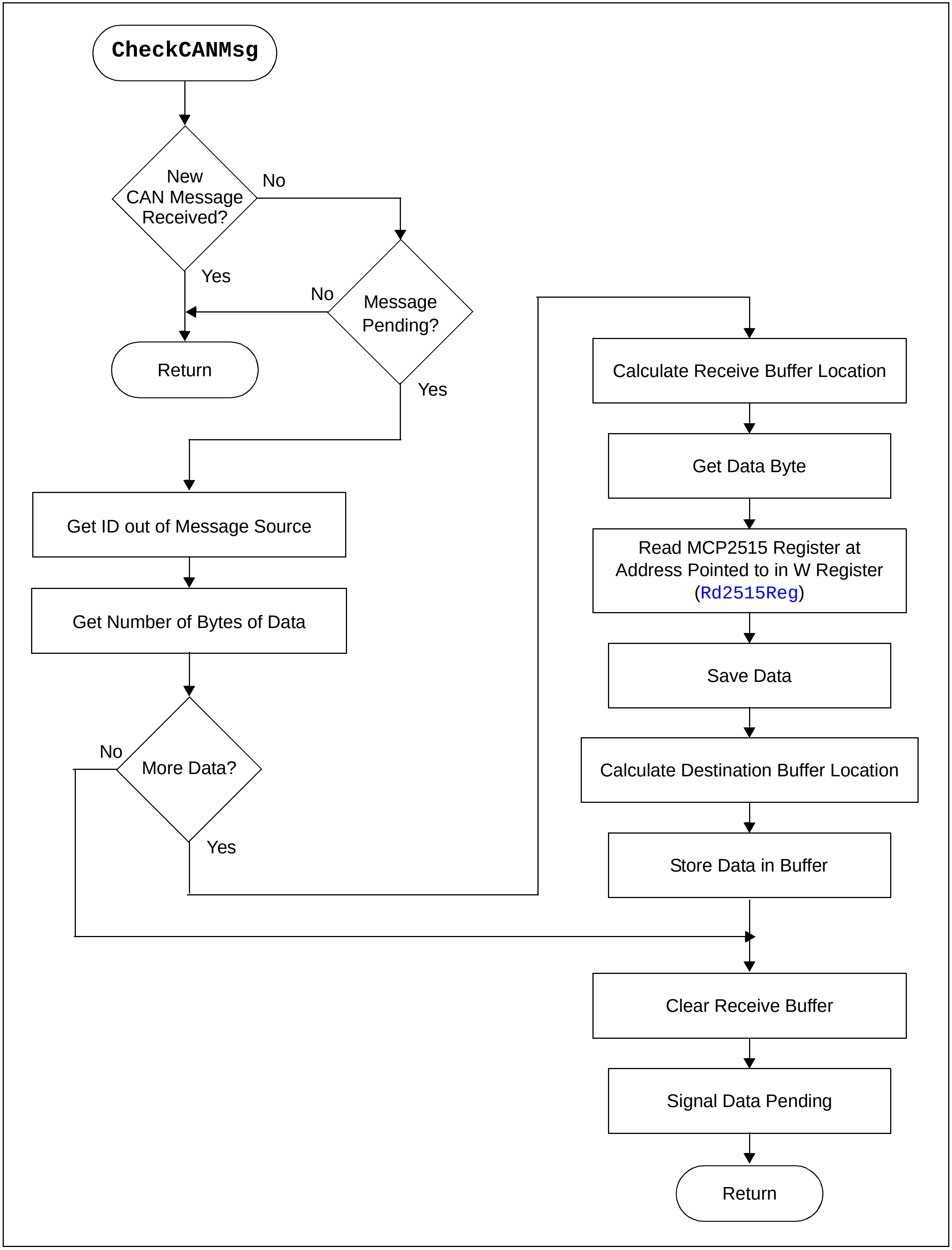

CheckCANMsg
New CAN Message Received?
No
Yes
No
Message Pending?
Return
Yes
Calculate Receive Buffer Location
Get Data Byte
Get ID out of Message Source
Read MCP2515 Register at Address Pointed to in W Register (Rd2515Reg)
Get Number of Bytes of Data
Save Data
No
More Data?
Calculate Destination Buffer Location
Yes
Store Data in Buffer
Clear Receive Buffer
Signal Data Pending
Return

FIGURE 13: PARSE THE MESSAGE (`ParseCAN`)

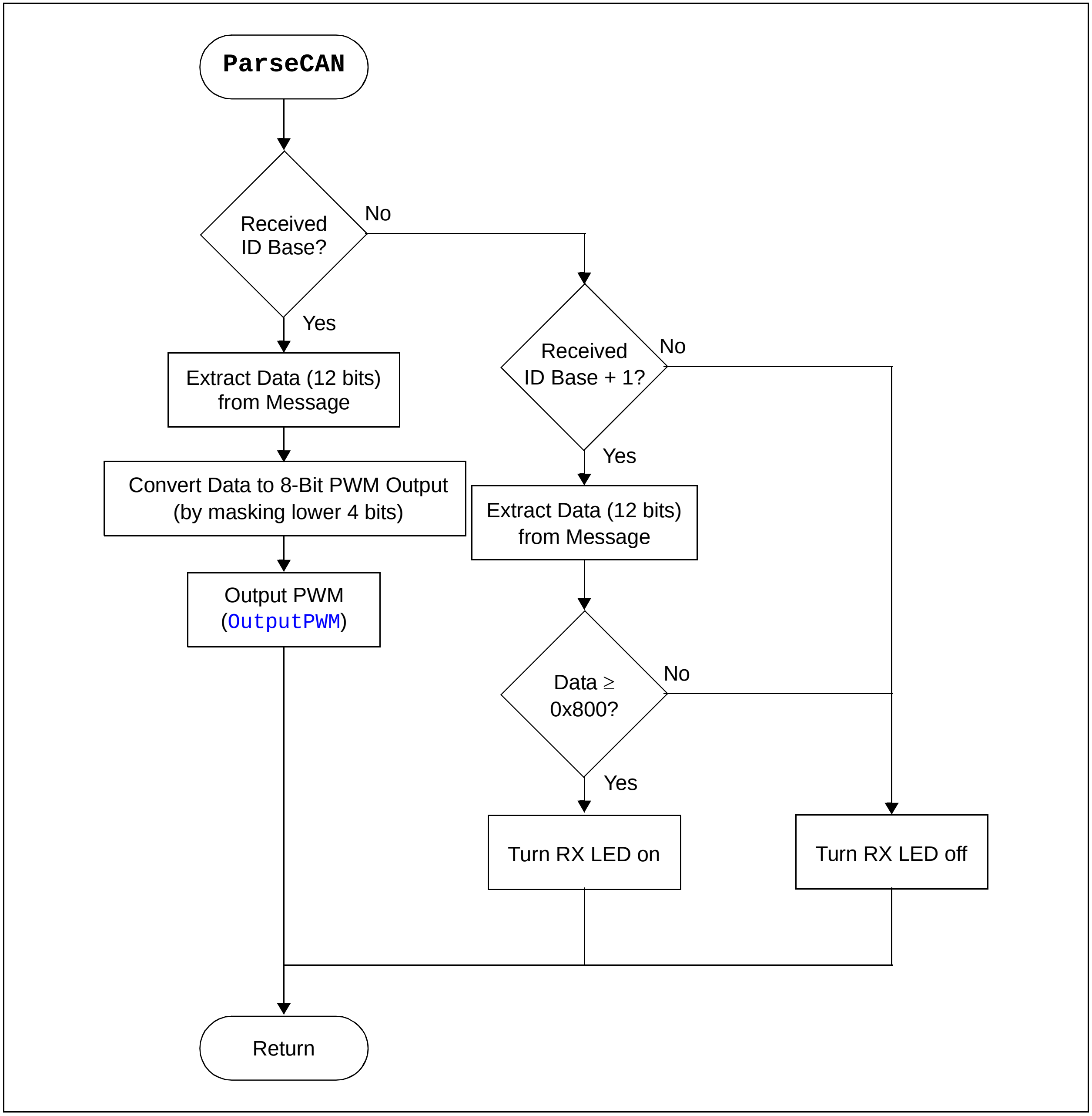


FIGURE 14: RESET MCP2515 REGISTERS (Reset2515)

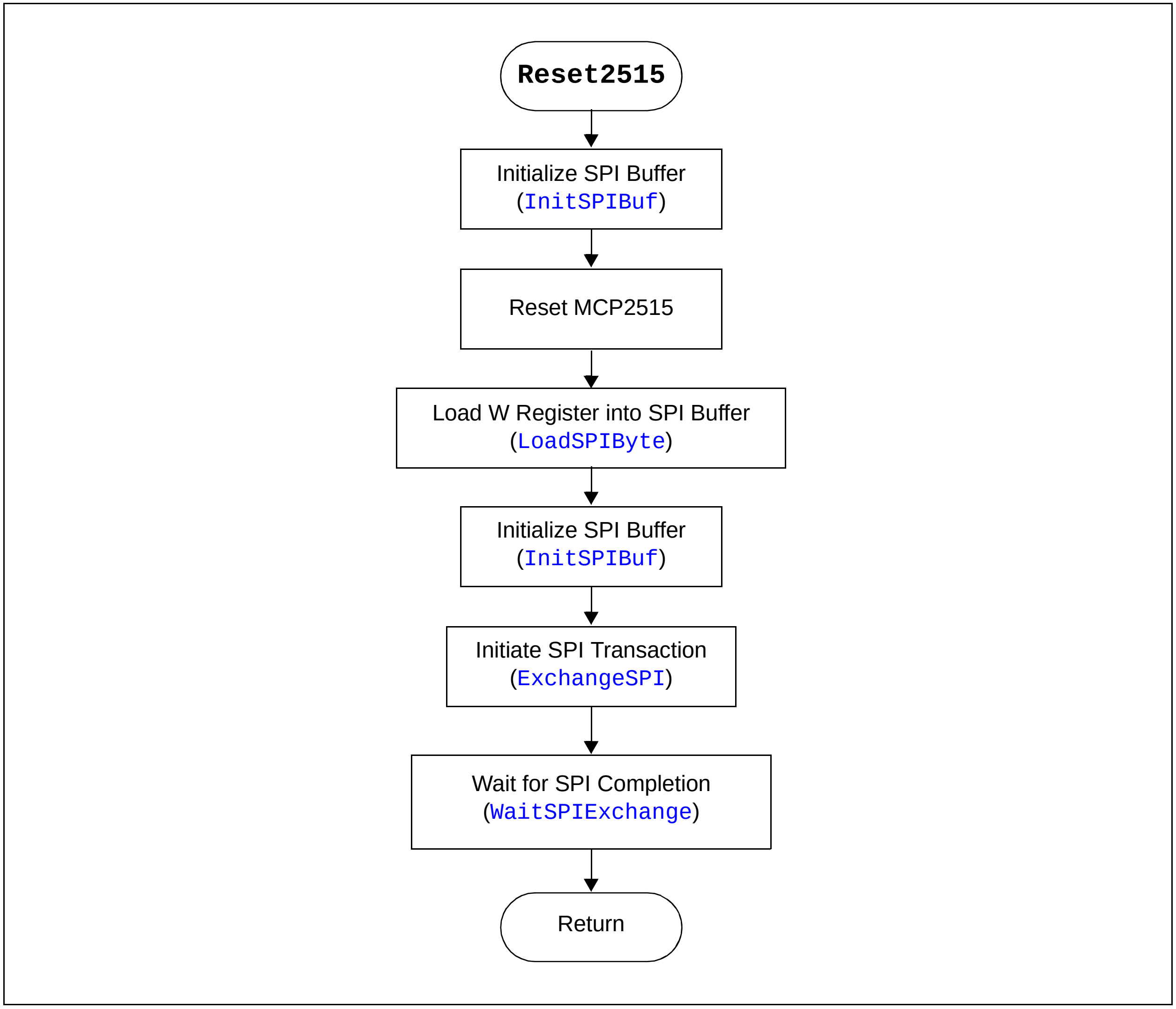

Reset2515
Initialize SPI Buffer (InitSPIBuf)
Reset MCP2515
Load W Register into SPI Buffer (LoadSPIByte)
Initialize SPI Buffer (InitSPIBuf)
Initiate SPI Transaction (ExchangeSPI)
Wait for SPI Completion (WaitSPIExchange)
Return

FIGURE 15: WRITE DATA IN REGISTER USING MASK (`BitMod2515`)

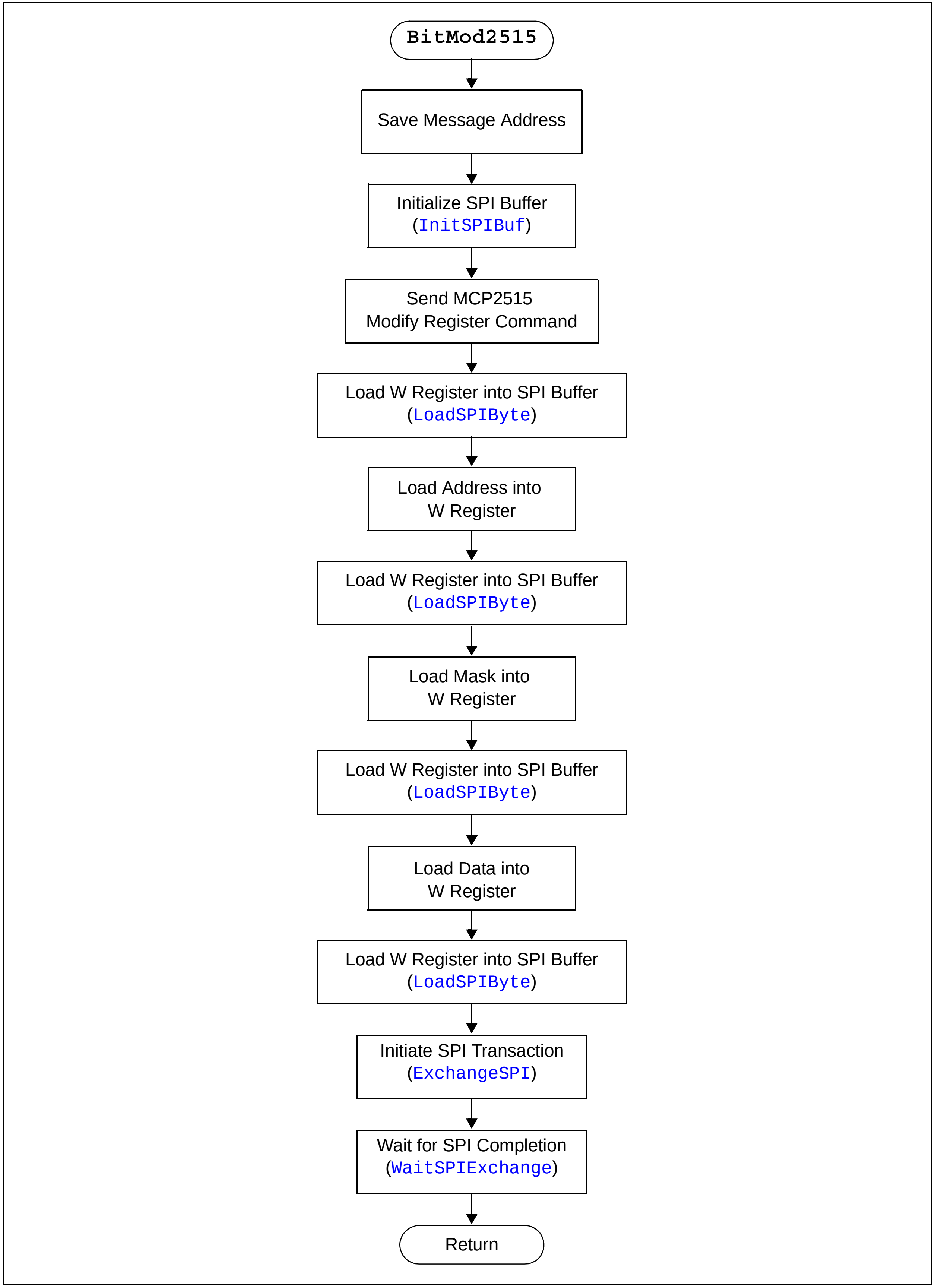

BitMod2515
Save Message Address
Initialize SPI Buffer (InitSPIBuf)
Send MCP2515 Modify Register Command
Load W Register into SPI Buffer (LoadSPIByte)
Load Address into W Register
Load W Register into SPI Buffer (LoadSPIByte)
Load Mask into W Register
Load W Register into SPI Buffer (LoadSPIByte)
Load 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 MCP2515 REGISTER IN W (`Wrt2515Reg`)

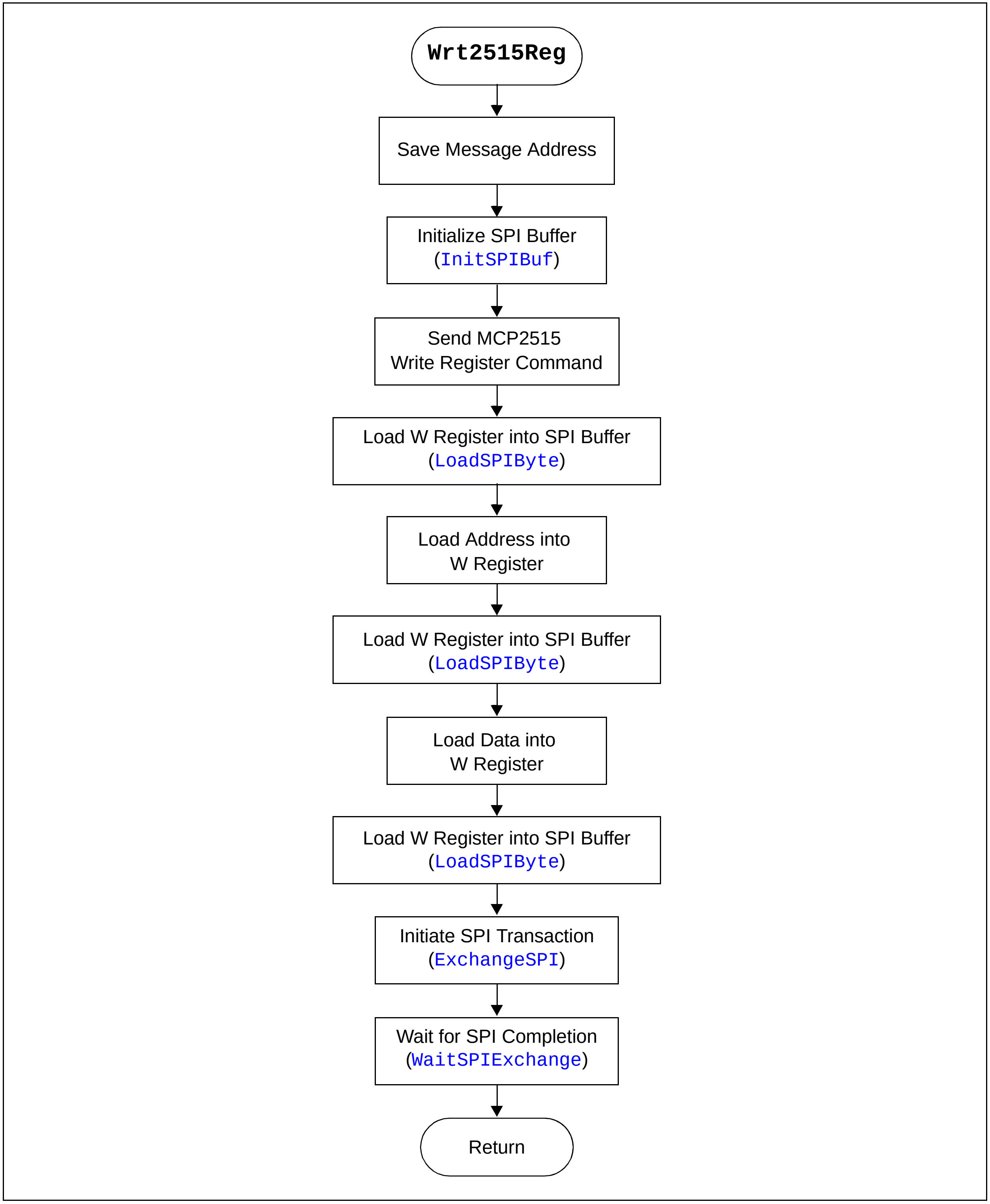

Wrt2515Reg
Save Message Address
Initialize SPI Buffer (InitSPIBuf)
Send MCP2515 Write Register Command
Load W Register into SPI Buffer (LoadSPIByte)
Load Address into W Register
Load W Register into SPI Buffer (LoadSPIByte)
Load Data into W Register
Load W Register into SPI Buffer (LoadSPIByte)
Initiate SPI Transaction (ExchangeSPI)
Wait for SPI Completion (WaitSPIExchange)
Return

FIGURE 17: SET NORMAL MODE (SetNormalMode)

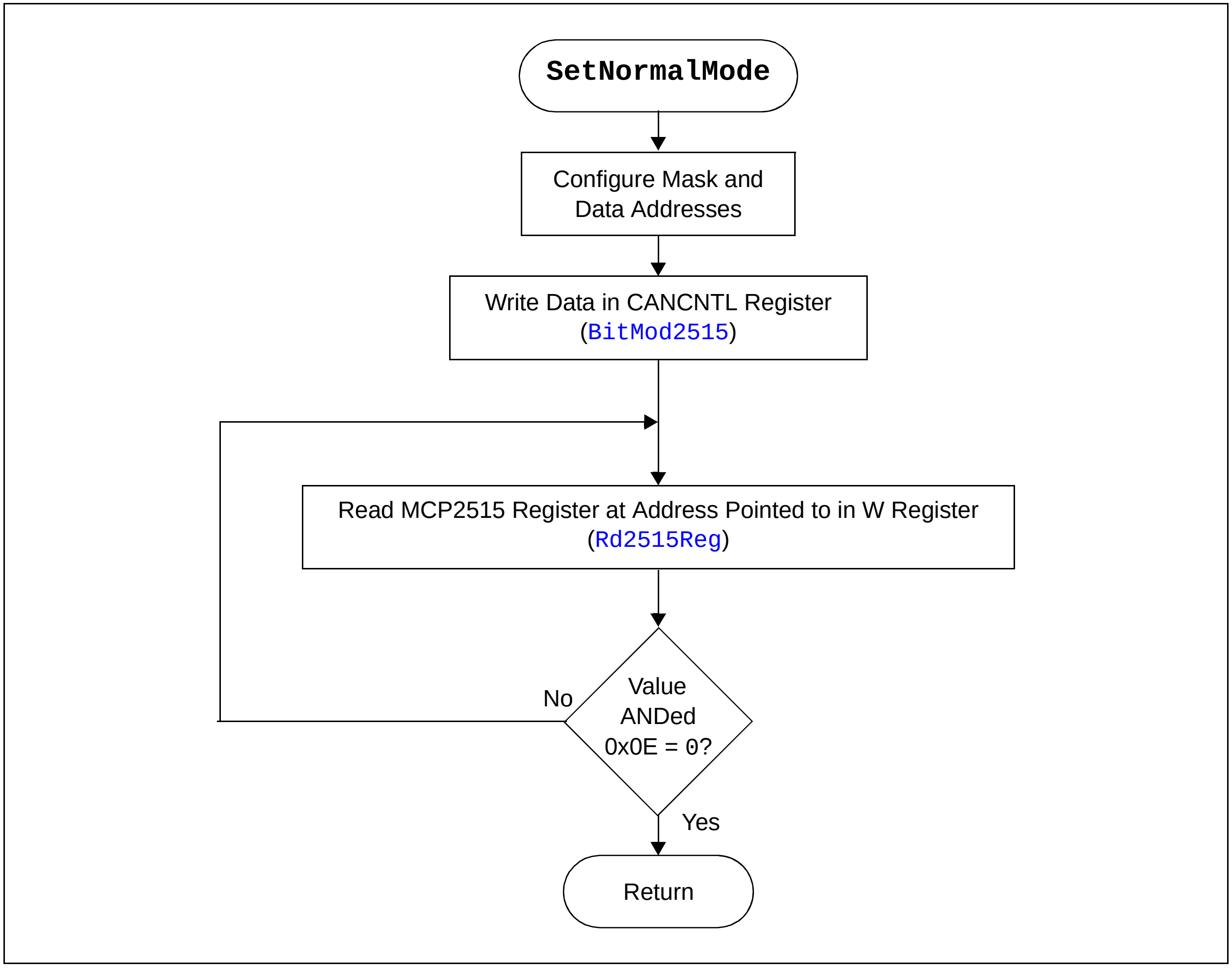

SetNormalMode
Configure Mask and Data Addresses
Write Data in CANCNTL Register (BitMod2515)
Read MCP2515 Register at Address Pointed to in W Register (Rd2515Reg)
No
Value ANDed 0x0E = 0?
Yes
Return

FIGURE 18: READ REGISTER ADDRESS IN W (Rd2515Reg)

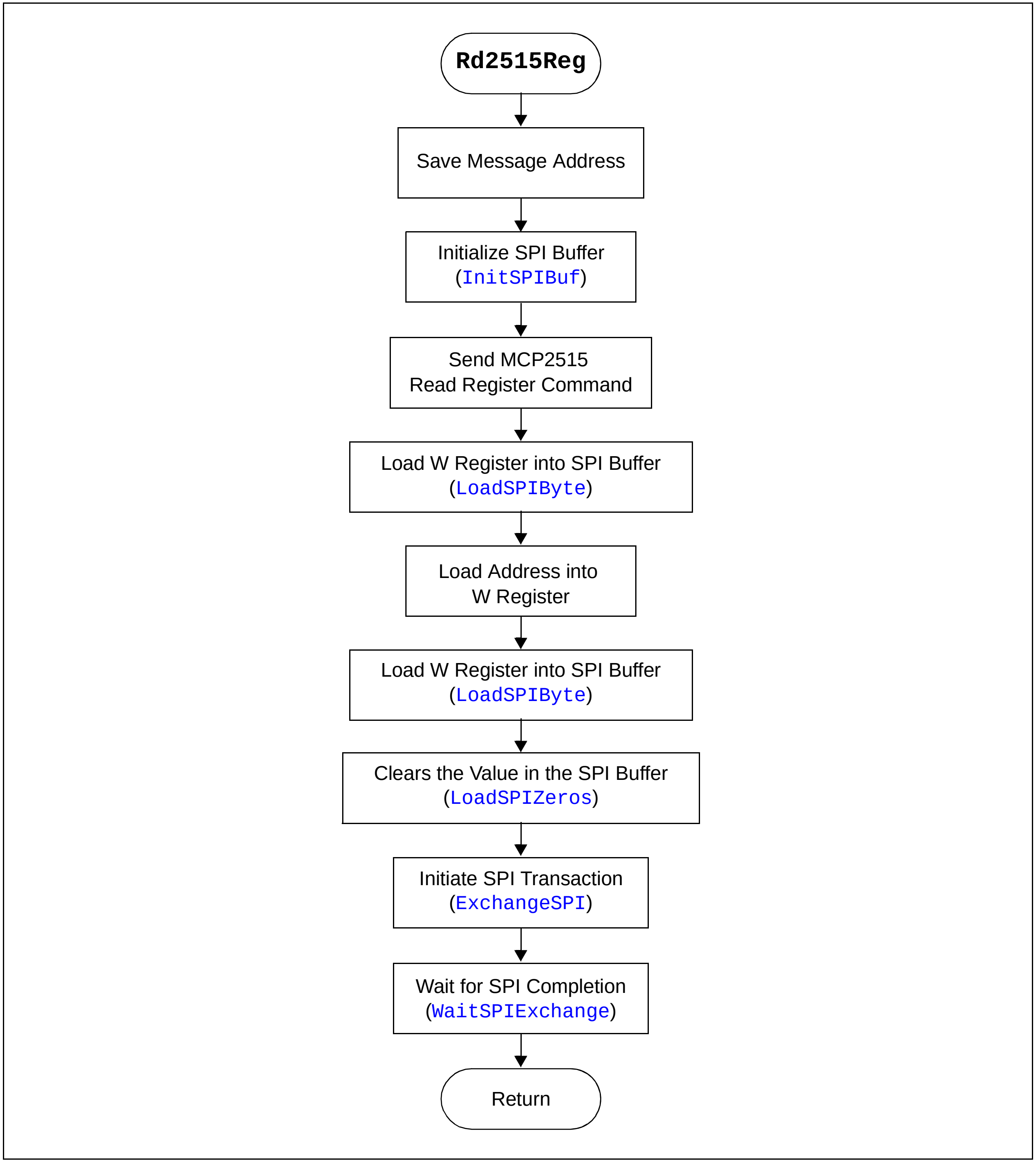

Rd2515Reg
Save Message Address
Initialize SPI Buffer (InitSPIBuf)
Send MCP2515 Read Register Command
Load W Register into SPI Buffer (LoadSPIByte)
Load Address into W Register
Load W Register into SPI Buffer (LoadSPIByte)
Clears the Value in the SPI Buffer (LoadSPIZeros)
Initiate SPI Transaction (ExchangeSPI)
Wait for SPI Completion (WaitSPIExchange)
Return

FIGURE 19: OUTPUT PWM (`OutputPWM`)

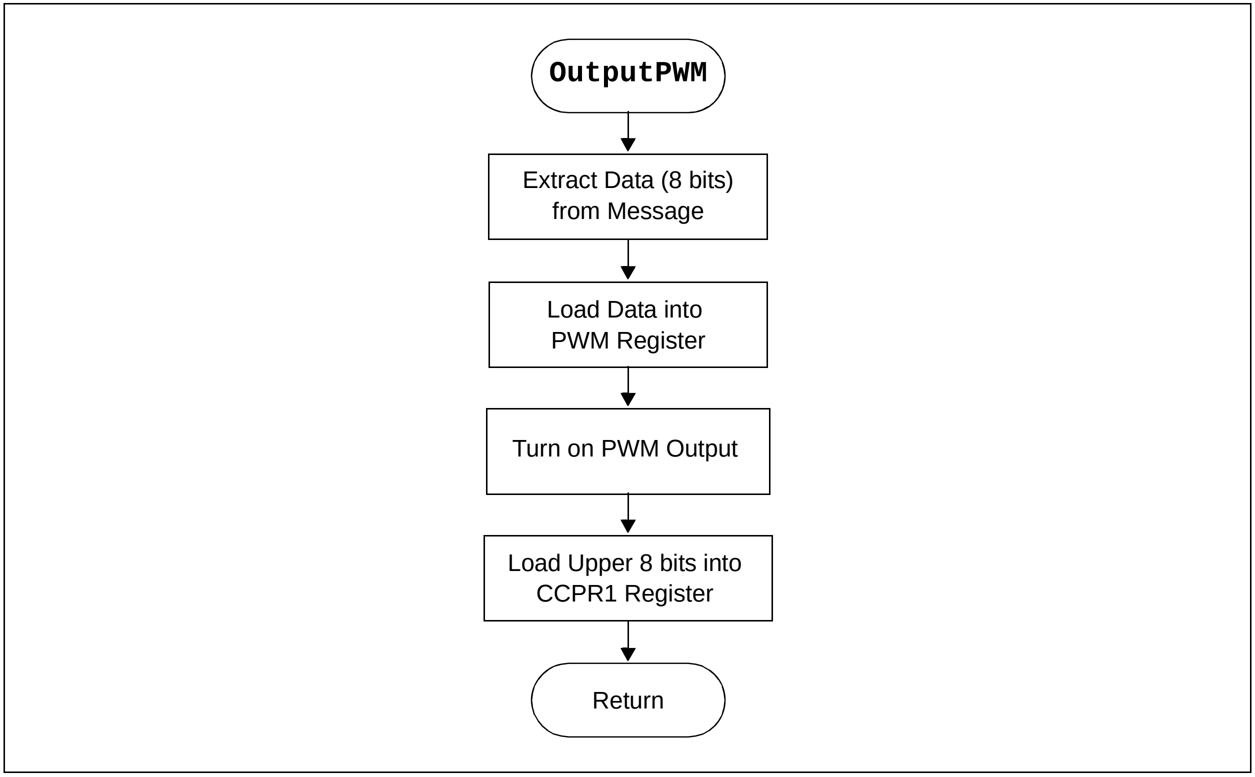

OutputPWM
Extract Data (8 bits) from Message
Load Data into PWM Register
Turn on PWM Output
Load Upper 8 bits into CCPR1 Register
Return

FIGURE 20: INITIALIZE SPI BUFFER (`InitSPIBuf`)

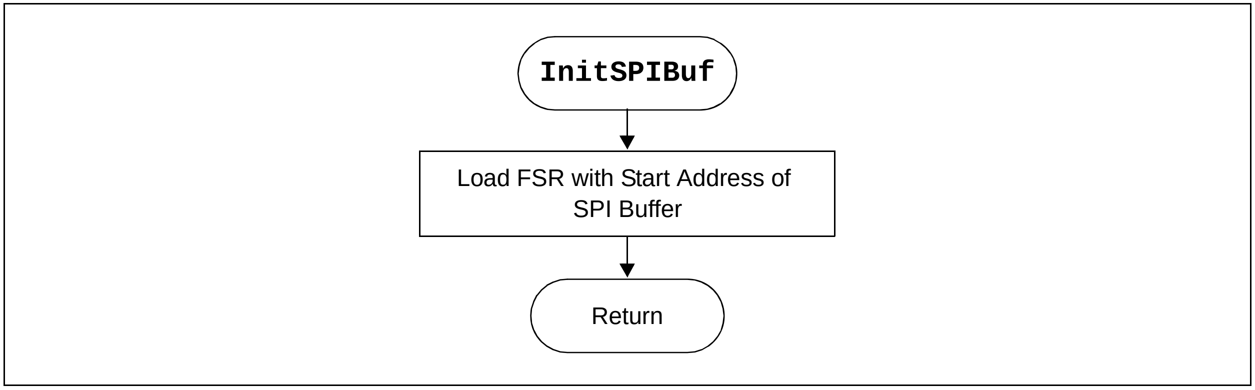


FIGURE 21: LOAD BYTE IN W TO SPI BUFFER (`LoadSPIByte`)

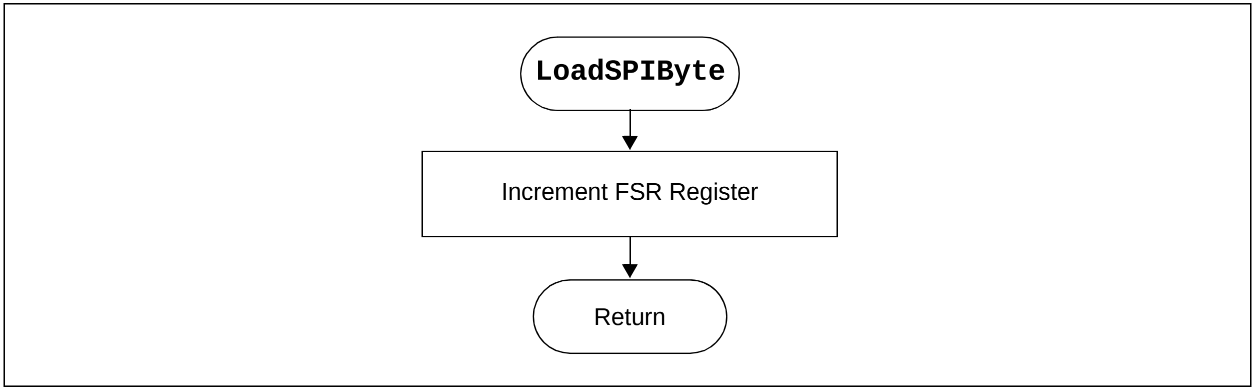


FIGURE 22: INITIATE SPI TRANSACTION (`ExchangeSPI`)

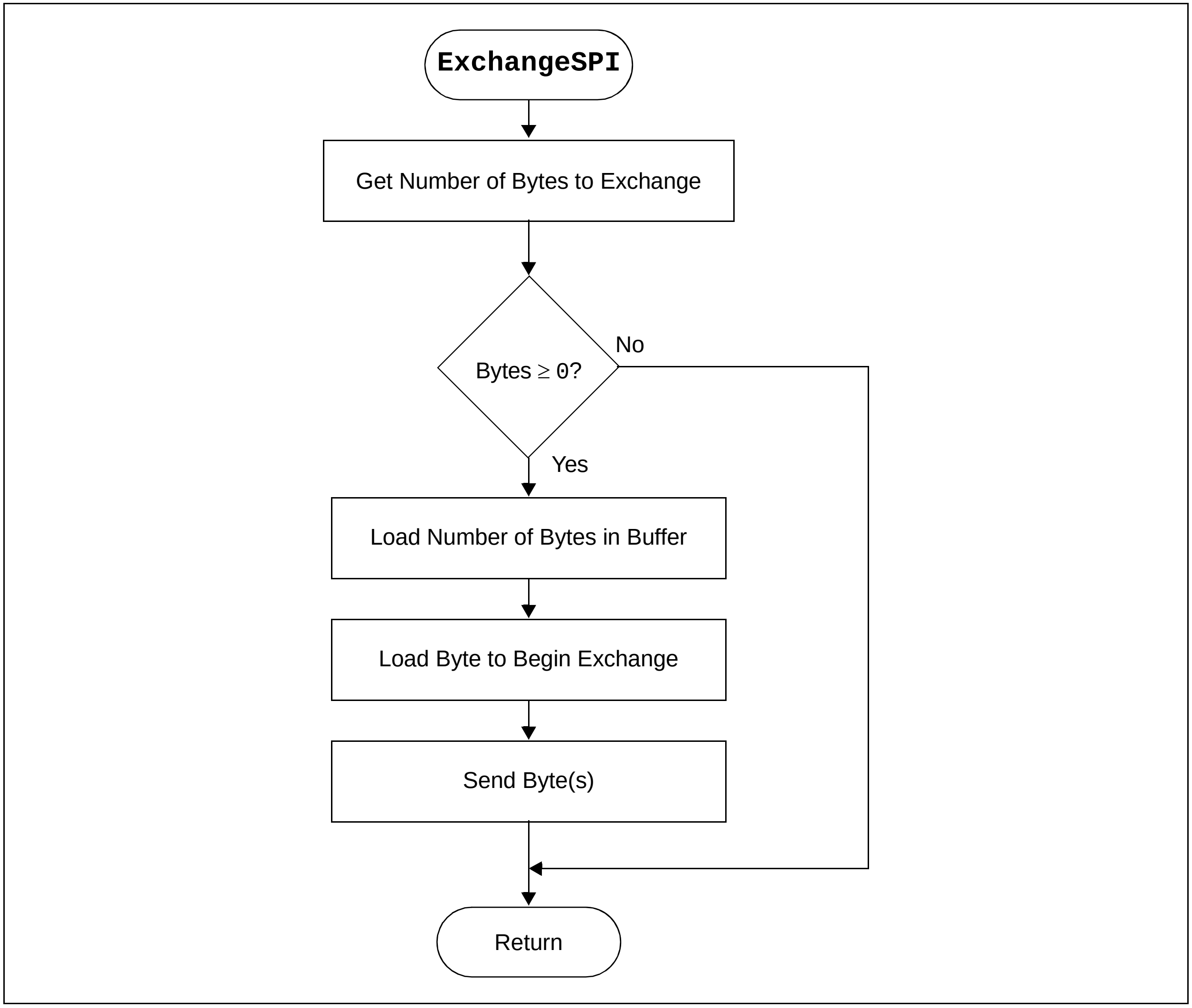

ExchangeSPI
Get Number of Bytes to Exchange
Bytes ≥ 0?
No
Yes
Load Number of Bytes in Buffer
Load Byte to Begin Exchange
Send Byte(s)
Return

FIGURE 23: WAIT FOR SPI COMPLETION (`WaitSPIExchange`)

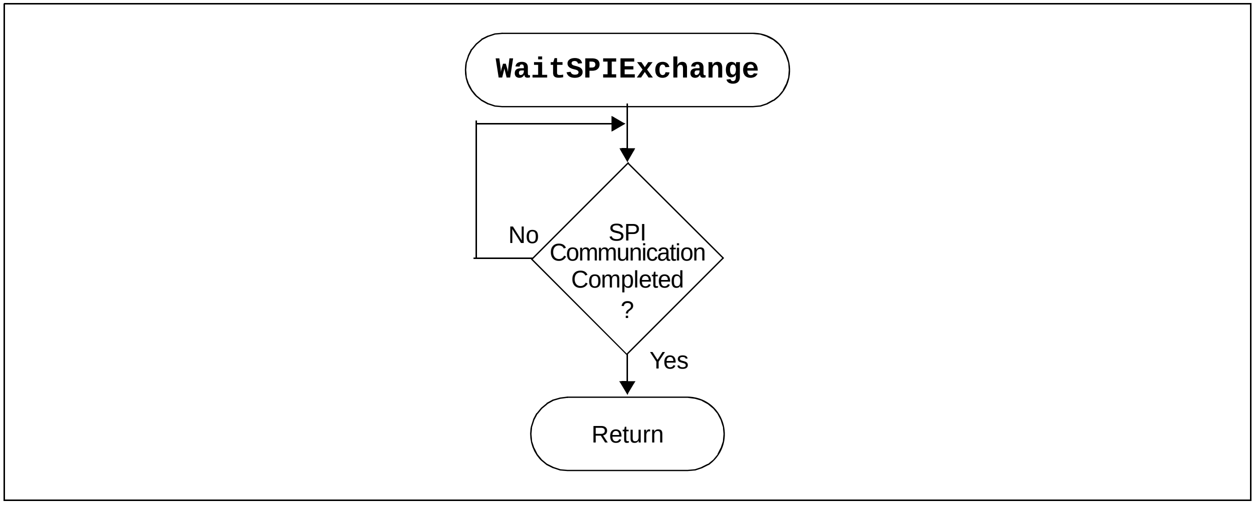
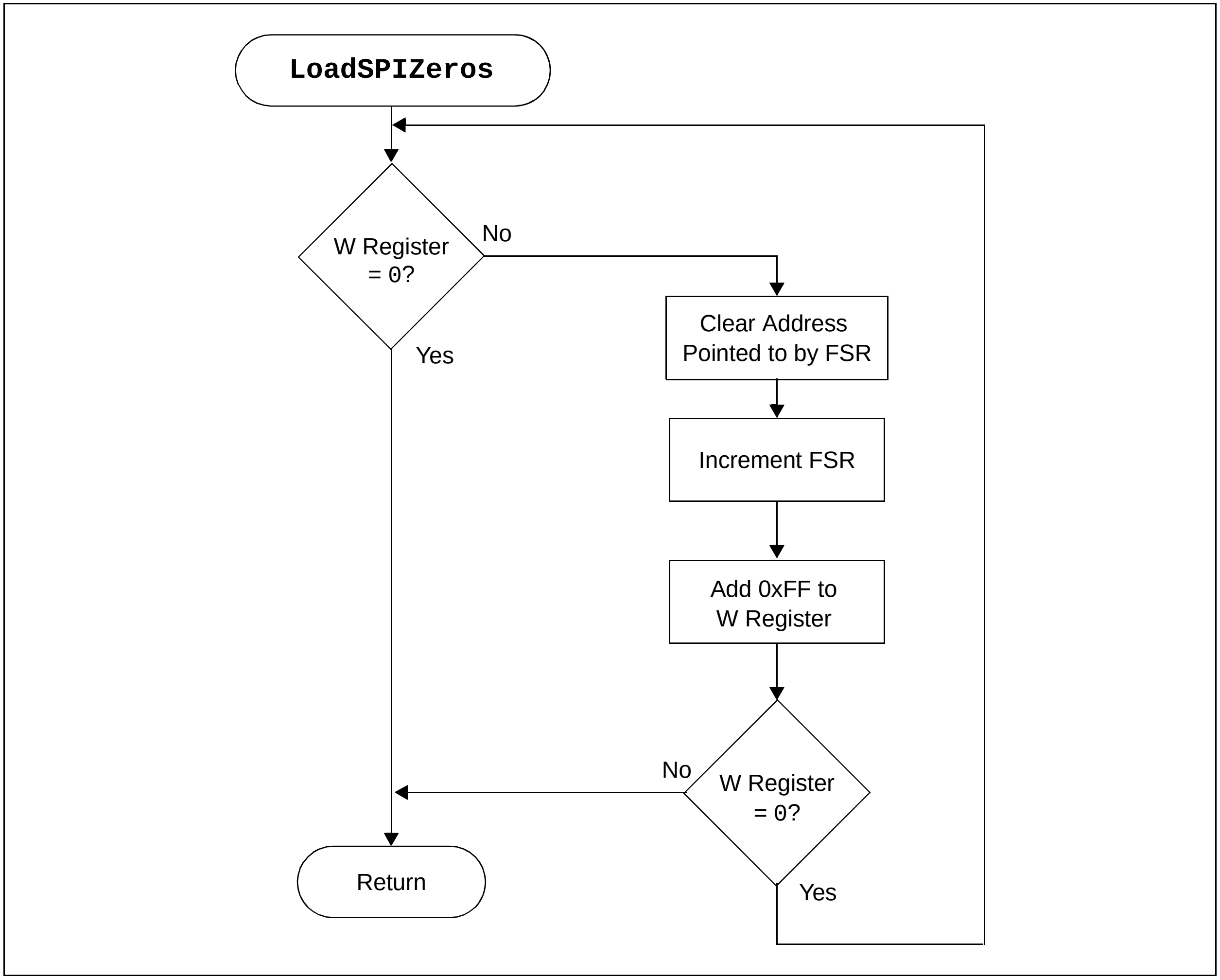


FIGURE 24: LOAD NUMBER OF ZEROS IN W TO SPI BUFFER (`LoadSPIZeros`)



APPENDIX A: SCHEMATICS

FIGURE 1: CAN NODE BOARD

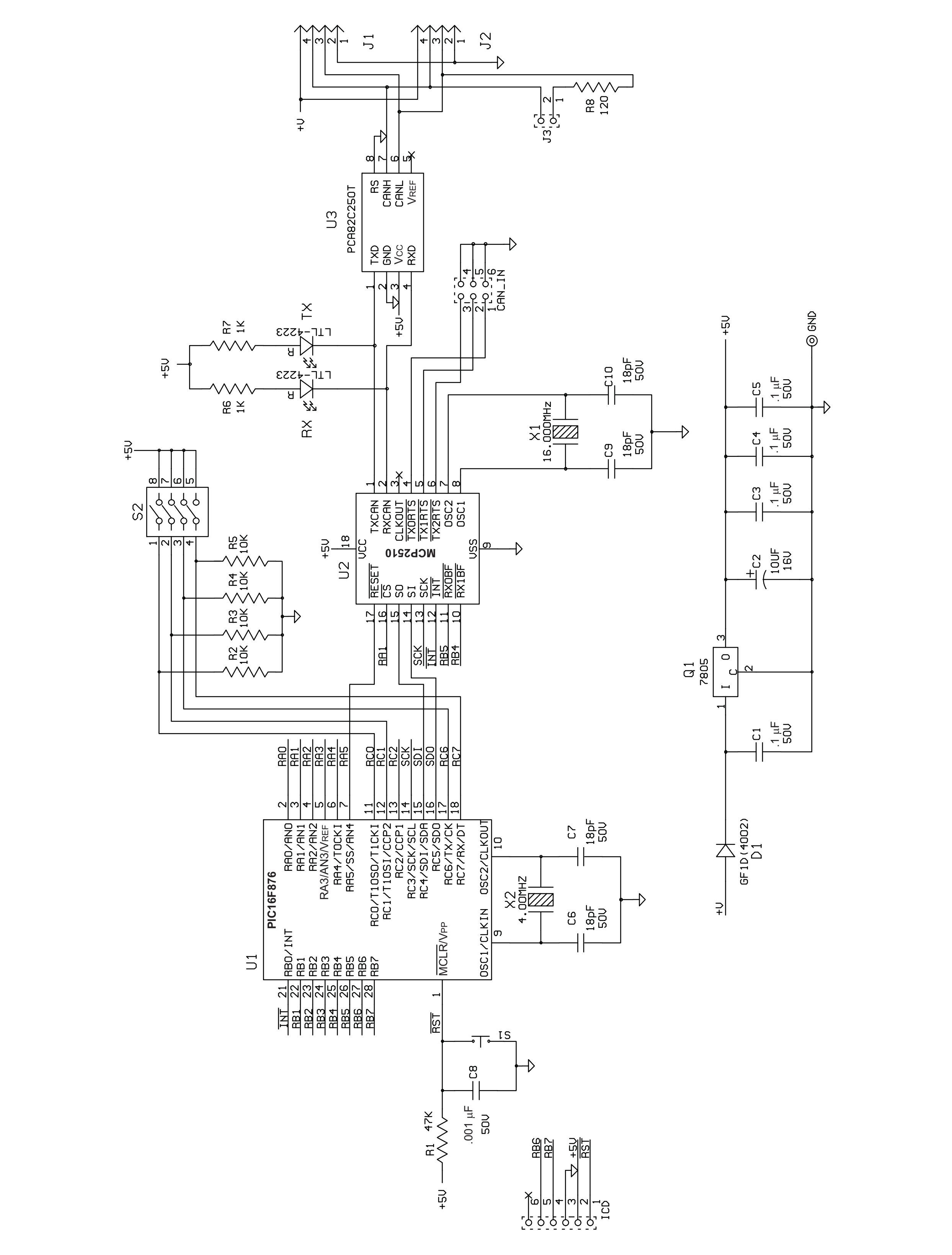
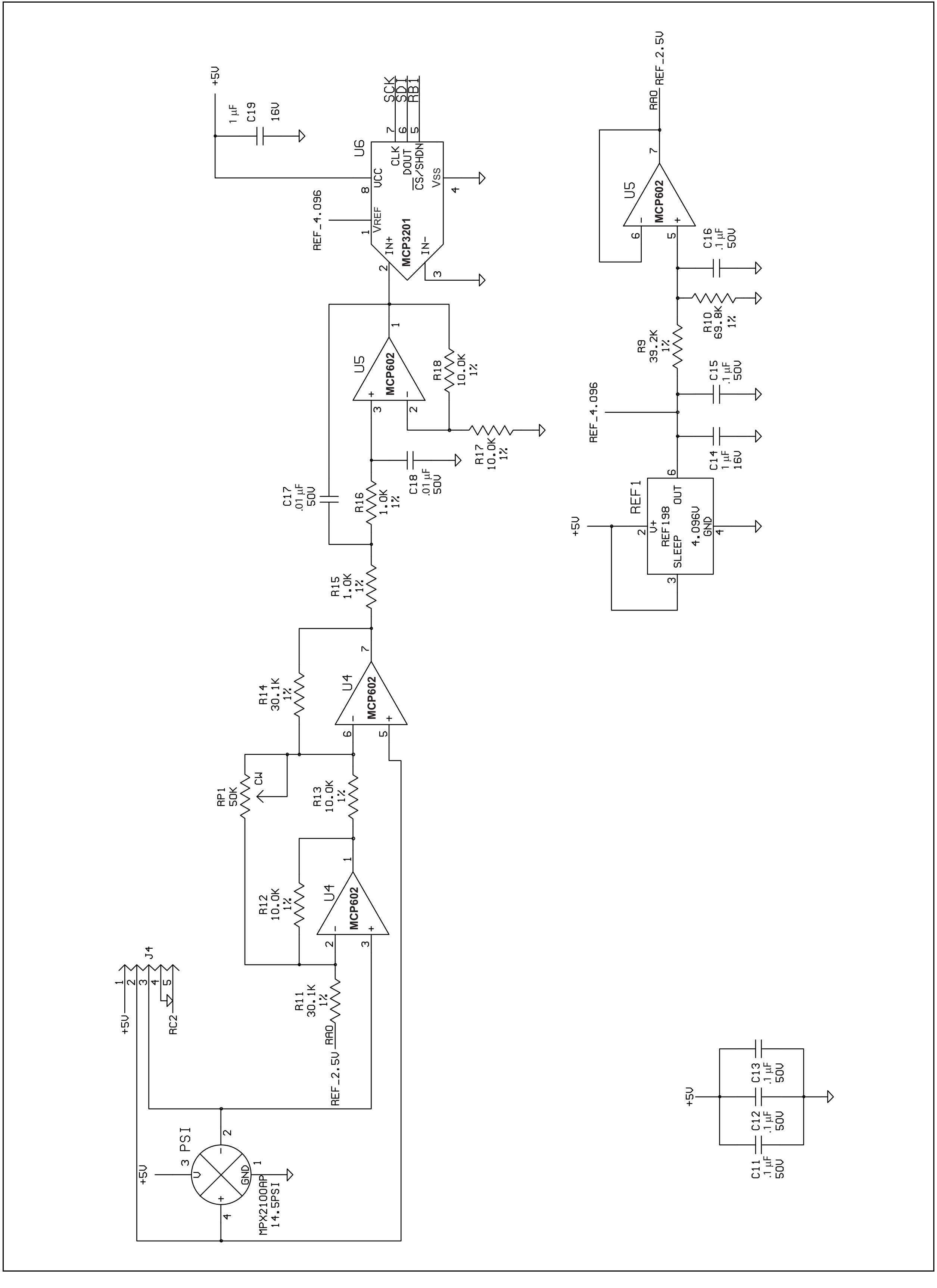


FIGURE 2: ANALOG INPUT BOARD



APPENDIX B: BILL OF MATERIALS

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

BILL OF MATERIALS (NODE BOARD)				
Qty	Reference	Description	Mfg Part	Manufacturer
1	PC BRD	PC BOARD	110501 A/A	
4	C9-C12	CAP, MLC,50V, NPO, 5%, 0805, 18 pF	MCH215A180JK	ROHM
1	C8	CAP, MLC, 50V, X7R, 10%, 0805, 1000 pF	MCH215C102KK	ROHM
5	C1, C3-C6	CAP, MLC, 50V, X7R, 10%, 0805, .1 μF	C0805C104K5RAC7210	KEMET
1	C2	CAP, ELEC, 16V, 20%, VA-B, 10 μF	EEV-HB1C100R	PANASONIC
1	D1	DIODE, 200V, 1A, DO-214BA	GF1D	GI
1	(J4)	DIP SHUNT, GOLD, 2 POS.	SNT-100-BK-G	SAMTEC
1	GND	TEST POINT	2305-3-00-44-0000070	MILLMAX
1	(U6)	SOCKET, 250V, DIP18, TIN BER COP	2-641611-1	AMP
1	(U4)	SOCKET, 250V, ROUND PIN, 28-PIN, F	110-99-328-41-001	MILLMAX
2	J1, J2	CONN, MTA100, 4-PIN MALE	640456-4	AMP
1	J4	CONN, .025SQX.1, 250V, 3A, 2PIN, M, .230	"TSW-102-07-T-S	SAMTEC
1	J3	CONN, SINGLE ROW, RA, 15POS, F	CES-115-02-T-S-RA	SAMTEC
5	RX, TX, ICD, CAN-IN, U5	DO NOT INSTALL		
2	R6, R7	RES, .1W, 5%, 0805, 1K	MCR10J102	ROHM
4	R2-R5	RES, .1W, 5%, 0805, 10K	MCR10J103	ROHM
1	R8	RES, .1W, 5%, 0805, 120 OHM	MCR10J121	ROHM
1	R1	RES, .1W, 5%, 0805, 47K	MCR10J473	ROHM
1	S1	SWITCH, PUSH, MOM, 6 MM	TL1105EF250	E-SWITCH
1	S4	SWITCH, DIP8, SPST, 50V, 100 mA, 4_POS	206-4ST	CTS
1	U6	IC, CAN_CONTROLLER, DIP18, CMOS, I	MCP2515-I/P	MICROCHIP
1	U4	IC, MICRO, DIP28, FLASH	PIC16F873-20/P	MICROCHIP
1	U7	IC, CAN_INTERFACE, SO8	MCP2551	MICROCHIP
1	Q5	REG, 5V, .1A, SOT-89	NJM78L05UA	NJR
1	X2	CRYSTAL, 4 MHz, CSM-7	ECS-40-20-5P	ECS
1	X1	CRYSTAL, PARALLEL, CSM-7, 16.00 MHz	ECS-160-20-5P	ECS
1	110500	OSDA LABOR	110500	OSDA

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

BILL OF MATERIALS (NODE BOARD)				
Qty	Reference	Description	Mfg Part	Manufacturer
1	PCB	PCB, CAN-NE ANALOG IN	110511 A/A	PCB
2	C6, C10	CAP, MLC,16V, Y5V, +80-20, 0805, 1 μF	MCH213F105ZP	ROHM
2	C5, C13	CAP, MLC, 50V, X7R, 10%, 0805, .01 μF	MCH215C103KK	ROHM
7	C1-C4, C8, C11, C12	CAP, MLC, 50V, X7R, 10%, 0805, .1 μF	C0805C104K5RAC7210	KEMET
1	C7	CAP, ELEC, 16V, 20%, VA-B, 10 μF	EEV-HB1C100R	PANASONIC
1	D1	DIODE, 200V, 1A, DO-214BA	GF1D	GI
1	TB1	TERM BLOCK, TH, 5POS, .1PITCH	1725685	PHEONIX
1	(J8)	SHUNT, DUAL, TIN, 2X2	MNT-102-BK-T	SAMTEC
1	J8	CONN, DUALROW, .025SQ, 3A, 8POS	TSW-104-07-T-D	SAMTEC
1	J3	CONN, SINGLE_ROW, RA, 15POS, M	TSW-115-08-T-S-RA	SAMTEC
2	XDUCER1, R11	DO NOT INSTALL		
2	R8, R9	RES, .1W, 1%, 0805, 1K	MCR10F1001	ROHM
5	R1, R3, R5, R6, R10	RES, .1W, 1%, 0805, 10.0K	MCR10F1002	ROHM
2	R4, R7	RES, .1W, 1%, 0805, 30.1K	MCR10F3012	ROHM
1	R13	RES, 1/10W, 1%, 0805, 39.2K	MCR10F3922	ROHM
1	R12	RES, 1/10W, 1%, 0805, 61.9K	MCR10F6192	ROHM
1	RP3	RES, POT, 1/2W, 10%, 20K	3299Y-203	BOURNS
1	RP1	RES, POT, 1/2W, 10%, 50K	3299Y-503	BOURNS
1	R2	THERMISTOR, tc-4.6, 10k, 1s .1", disc, TH	ERT-D2FHL103S	PANASONIC
1	U4	IC, MEMORY, DIP8, E2PROM, 512X8	25C040/P	MICROCHIP
1	U3	IC, ADC, DIP8, 12-BIT, CMOS, +/- 2 LSB	MCP3201-C/P	MICROCHIP
2	U1, U2	IC, CMOS, DIPS8, DUAL_OPAMP, LOW_POWER	MCP602/P	MICROCHIP
1	REF1	REG, VOLTAGE_REFERENCE, 4_096V, SO8, F	REF198FA	ANALOG_DE
1	VR1	REG, 5V, .1A, SOT-89	NJM78L05UA	NJR
1		CAN-NET ANALOG IN	110510	OSDA

APPENDIX C: SOURCE CODE

```
;**********************************************************************
; Microchip CAN Reference Design
; node.asm
; Mike Richitelli
; Diversified Engineering
; 283 Indian River Road
; Orange, CT 06477
; (203)-799-7875 fax(203)799-7892
; WWW.DIVERSIFIEDENGINEERING.NET
;
;**********************************************************************
;**********************************************************************

          TITLE " CAN_Ref Design "

;**********************************************************************

dVersion       equ  1
dRelease       equ  5

;======================================================================
; Transmits CAN message every 131 mSec.  Message contains two data
; bytes
;         that represent a 12 bit value with least significant byte
;         sent first.
; Cycles between three outputs:
;         Pot:  Value goes from 0 to 0xFF0 as Pot is turned clockwise.
;               ID is selected from DIP switches #3 and #4 as follows:
;                   #3 #4  ID
;                   0  0   transmission disabled
;                   0  1   0x100
;                   1  0   0x200
;                   1  1   0x300
;
;         Push button switch: Switch open => 0, Switch closed => 0xFFF
;               ID is Pot ID + 0x010
;
;         CdS:  Value goes from 0 to 0xFF0 as Pot is turned clockwise.
;               ID is Pot ID + 0x020
;
; CAN messages received are assumed to be 12 bit data sent as two ; ;
;bytes,
;         least significant byte first.
;
;         The base ID for receiving CAN messages is specified by DIP
;         switches #1 and #2:
;                   #1 #2  ID
;                   0  0   0x000
;                   0  1   0x100
;                   1  0   0x200
;                   1  1   0x300
```

```
;
;         Lamp: If the message ID matches the ID selected by the DIP
;               switches the 12 bit data is used to generate a PWM
;output
;               where a 0 value gives a zero duty cycle and 0xFFF
;generates
;               a 100% duty cycle.  The lamp output is proportional to
;               the duty cycle.
;               ID is Base ID
;
;         LED: On if value received is >= 0x800 and off if < 0x800.
;              ID is Base ID + 0x010
;
;======================================================================

;----- PIC16F876 Micro -----;

          LIST P=16F876
          LIST r=dec,x=on,t=off

#include "P16F876.INC"

  __CONFIG
_BODEN_ON&_CP_OFF&_WRT_ENABLE_ON&_PWRTE_ON&_WDT_OFF&_HS_OSC&_DEBUG_OFF&_CPD_OFF&_LVP_OFF
  __IDLOCS (dVersion<<8)|dRelease  ; version: vvrr , vv- version, rr - release

;---------------------------;

#include "MACROS16.INC"
#include "MCP2515.INC"

;          errorlevel 0,-306,-302,-305


;******** constants

;Crystal freq 4.00 MHz, Fosc/4 = 1 uS
;
; Timer 1: Uses no prescale => Tic is 1 uSec
;       8 bit rollover 256 uSec
;       16 bit rollover 65.536 mSec

;   8 bit timers
;    TMR1L: 1 uSec tics with maximum of 1/2 rollover = 128 uSec maximum
;    TMR1H: 256 uSec tics with maximum of 1/2 rollover = 32.768 msec
;maximum



;======================================================================


; A/D selection ( value of ADCON0 )

#define dA2DRA0     B'01000000'         ; fosc/8 clk, RA0, A/D off
#define dA2DRA3     B'01011000'         ; fosc/8 clk, RA3, A/D off


;**********************************************************************
; special function defines

#define _SSPEN      SSPCON,SSPEN        ; SPI enable

;**********************************************************************
```

```
;; General control flags definitions

#define tbWork          bGenFlags1,0   ; working bit
#define tbReset         bGenFlags1,1   ; must reset
#define tbNewSPI        bGenFlags1,2   ; new SPI data available
#define tbRxMsgPend     bGenFlags1,3   ; new CAN message received
#define tbTxMsg         bGenFlags1,4   ; xmit next CAN message
#define tbRC2NowHigh    bGenFlags1,5   ; Robot PWM signal high


;*****************  PIN DEFINITIONS   *********************************
#define tp2510_CS_   PORTA,1       ; CS_ for MCP2515 chip

;; I/O
#define tpSwitch_    PORTB,1       ; Push button switch, Open => high
#define tpLED        PORTB,2       ; LED

;; Analog In
#define tpA2D_CS_    PORTB,1       ; CS_ for 3201 A2D chip
#define tpEE_CS_     PORTB,2       ; CS_ for 25040 E2

;*****************  LOCAL REGISTER STORAGE ****************************
;
;
;============ BANK 0 ==================================================

 cblock   0x20
     ;; interrupt variables
          bIntSaveSt       ; save Status
          bIntSaveFSR      ; save FSR
          bIntSavPCLATH    ; interrupt storage for PCLATH
          bIntWork         ; working
          iIntWork:2       ; working

     ;; general work space
    bGenFlags1       ; general control flags 1
    bGenFlags2       ; general control flags 2
          bWork            ; work byte
          bWork1           ; work byte
    iWork:2          ; work integer
    bCnt             ; work byte counter

     ;; Arithmetic
          iA:2             ; 2 byte integer
          iB:2             ; 2 byte integer

     ;; Timer1
          iTimer1:2        ; counts Timer1 rollover
          bGenClk          ; general clock
          bXmitClk         ; Countdown to xmit next message

     ;; In/Out variables
          iA2DValue:2      ; 12 bit or 8 bit A2D value
          bPWMValue        ; 8 bit PWM value
          iRecValue:2      ; 12 bit received value

     ;; general control
          bSwXmitID        ; ID for transmission from DIP switch
          bBaseRecID       ; ID for reception from DIP switch
          bRecIDNext       ; Rec Base ID + 1
          bXmitID          ; ID for transmission of next msg
          bNextMsgType     ; Select next msg

     ;; Received CAN message
          iRecID_L         ; ID of received message (3 bits left
;justified)
          iRecID_H         ; ID of received message (8 bits left
;justified)
          bRecCount        ; number of bytes received
          pRecDataBase:8   ; received data

     ;; Low level SPI interface
          b2510RegAdr      ; Register address
          b2510RegData     ; Data sent/received
          b2510RegMask     ; Bit Mask

       ; following used in interrupt
          bSPICnt          ; # bytes remaining to receive
          pSPIBuf          ; Pointer into buffer
          pSPIBufBase:12   ; Base of SPI receive/xmit buffer

 endc


; storage for interrupt service routine
; W saved in one of these locations depending on the page selected
; at the time the interrupt occured

bIntSaveW0 equ  0x7F       ; interrupt storage for W

;============ BANK 1 =================
bIntSaveW1 equ  0xFF       ; interrupt storage for W

;**********************************************************************
;********** LOCAL MACROS **********************************************
;**********************************************************************
;
; Shift left 2 byte integer once.
iShiftL macro    iVar
        bcf      _C                 ; clear carry bit
        rlf      iVar,F
        rlf      iVar+1,F
        endm

; Shift right 2 byte integer once.
iShiftR macro    iVar
        bcf      _C                ; clear carry bit
        rrf      iVar+1,F
        rrf      iVar,F
        endm

; Increment 2 byte integer
intInc  macro   iVar
        incf    iVar,F
        skipNZ
        incf    iVar+1,F
        endm

;
;; --------------------------------------------------------------------
;; Set TRM1H 8 bit clock
;    TMR1H: 256 uSec tics with maximum of 1/2 rollover = 32.768 msec ;
;    maximum
;; --------------------------------------------------------------------
Set1HClock macro bClk,Value
          movfw     TMR1H
          addlw     Value
          movwf     bClk
          endm
```

```
;; --------------------------------------------------------------------
;; Jump to jLabel if TMR1H (low byte) < bClk
;; --------------------------------------------------------------------
jmp1HNotYet macro bClk,jLabel

          movfw     TMR1H
          subwf     bClk,W
          andlw     0x80
          jmpZ      jLabel
          endm

;; -----------------------------------------------------------------------------
;; Jump to jLabel if TMR1H (low byte) < bClk
;; --------------------------------------------------------------------
jmp1HDone macro bClk,jLabel

          movfw     TMR1H
          subwf     bClk,W
          andlw     0x80
          jmpNZ     jLabel
          endm
```

```
;**********************************************************************
;      Begin Program Code
;**********************************************************************

          ORG      0x0            ;memory @ 0x0
          nop                     ;nop ICD!!
    goto     HardStart

    ORG     04h            ;Interrupt Vector @ 0x4
;**********************************************************************
; Interrupt service routine - must be at location 4 if page 1 is used
; Context save & restore takes ~20 instr
;**********************************************************************
     ;; Global int bit, GIE, has been reset.
     ;; W saved in bIntSaveW0 or bIntSaveW1 depending on the bank
;selected at
     ;; the time the interrupt occured.
          movwf     bIntSaveW0      ; save W in either of two locations
                                    ; depending on bank currently
;selected

     ;; only way to preserve Status bits (since movf sets Z) is with a
     ;; swapf command now
          swapf     STATUS,W        ; Status to W with nibbles swapped
          BANK0
          movwf     bIntSaveSt
          movfw     FSR
          movwf     bIntSaveFSR     ; save FSR
          movf      PCLATH,W
          movwf     bIntSavPCLATH   ; interrupt storage for PCLATH
          clrf      PCLATH          ; set to page 0

     ;; Must determine source of interrupt

     ;; SPI interrupt
    btfsc    _SSPIF        ; SPI interrupt
    goto     IntSPI

          jmpSet   _TMR1IF,jIntTimer1  ; Timer1 overflow interrupt flag

     ;; unknown

     ;; restore registers and return
IntReturn
          BANK0
          movf      bIntSavPCLATH,W   ; interrupt storage for PCLATH
          movwf     PCLATH
          movf      bIntSaveFSR,W  ; restore FSR
          movwf     FSR
          swapf     bIntSaveSt,W ; get swapped Status (now unswapped)
          movwf     STATUS       ; W to Status  ( bank select restored )
          swapf     bIntSaveW0,F ; swap original W in place
          swapf     bIntSaveW0,W ; now load and unswap ( no status
;change)
          retfie                 ; return from interrupt
```

```
;***************** ID TABLE *******************************************
; Look up ID associated with bits 0,1 in W
RxIDTable addwf    PCL,F           ;Jump to char pointed to in W reg
                                      ;( adds 5bits from PCLATH )
          retlw    0x00  ; 0
          retlw    0x20  ; 1
          retlw    0x10  ; 2
          retlw    0x30  ; 3
RxIDTable_End
#if ( (RxIDTable & 0xF00) != (RxIDTable_End & 0xF00) )
       MESSG   "Warning - Table crosses page boundry in computed jump"
#endif

; Look up ID associated with bits 0,1 in W
TxIDTable addwf    PCL,F           ;Jump to char pointed to in W reg
                                      ;( adds 5bits from PCLATH )
          retlw    0xFF  ; 0
          retlw    0x20  ; 1
          retlw    0x10  ; 2
          retlw    0x30  ; 3
TxIDTable_End
#if ( (TxIDTable & 0xF00) != (TxIDTable_End & 0xF00) )
       MESSG   "Warning - Table crosses page boundry in computed jump"
#endif

;***************** LIBRARY STORAGE & FUNCTIONS ************************

#include "CanLib.asm"         ; basic MCP2515 interface routines
#include "a2d3201.asm"        ; MCP3201 AD routines

;***************** Local Interrupt Handlers ***************************


;**********************************************************************
;jIntTimer1
;         Timer1 rollover interrupt.
;
;**********************************************************************
jIntTimer1  ; Timer1 overflow interrupt flag
          bcf       _TMR1IF        ; timer1 rollover interrupt flag
          intInc    iTimer1

          jmpFeqZ   bXmitClk,IntReturn

          decfsz    bXmitClk,F     ; Countdown to xmit next message
          goto      IntReturn

     ; Countdown to xmit next message

          bsf       tbTxMsg        ; xmit next CAN msg
          goto      IntReturn
```

```
;**********************************************************************
;IntSPI
;
; A single buffer, at pSPIBufBase, is used for both SPI receive and
; transmit.  When a byte is removed from the buffer to transmit it is
; replaced by the byte received.
;
; When here the buffer pointer, pSPIBuf, points to the last byte loaded
; for transmission. This is the location that the received byte will be stored.
;
; When here the count, bSPICnt, contains the number of bytes remaining
; to be received.  This is one less then the number remaining to be
; transmitted.  When bSPICnt reaches zero the transaction is complete.
;
;
;**********************************************************************IntSPI
          bcf       _SSPIF              ; clear interrupt flag

     ;; Transfer received byte to the next location in the buffer
          bV2bV     pSPIBuf,FSR
          incf      pSPIBuf,F

          movfw     SSPBUF              ; get data & clear buffer flag
          movwf     INDF                ; put it into SPI buffer

          decfsz    bSPICnt,F
          goto      jIntSPI1            ; More bytes to send

     ;; Last transaction completed
          bsf       tp2510_CS_           ; CS_ for MCP2515 chip
          goto      IntReturn

jIntSPI1
     ;; Fetch next byte from buffer and load it for transmission
          incf      FSR,F
          movfw     INDF                ; get byte from buffer
          movwf     SSPBUF              ; send it
          goto      IntReturn
```

```
;**********************************************************************
;**********************************************************************
;**********************************************************************

HardStart
          call      Init

     ;; Make sure no chips are selected
          bsf       tp2510_CS_   ; CS_ for MCP2515 chip

     ;; Read DIP switch and create bBaseRecID and bSwXmitID

     ;; Rec ID bits are at pins 0,1 of PORTC and are logic high = 1
          movfw     PORTC
          andlw     0x03
          call      RxIDTable
          movwf     bBaseRecID

     ;; Xmit ID bits are at pins 6,7 of PORTC and are logic low = 1
          swapf     PORTC,W
          movwf     bSwXmitID
          rrfbSwXmitID,F
          rrfbSwXmitID,W
          andlw     0x03
          call      TxIDTable
          movwf     bSwXmitID


     ;; ----------------- One time calculations -----------------------

     ;; Setup SPI port
          call      InitSPIPort

     ;; Wait 28 mS for MCP2515 to initialize ( there is no significance to 28 mS -
     ;; we just selected a large time since time is not critical)
          Set1HClock bGenClk,100   ; 277.77 uSec tics
jInit5
          jmp1HNotYet bGenClk,jInit5


     ;; Setup all MCP2515 registers
          call      Init2510

          bsf       tbTxMsg             ; xmit flag
```

```
;; --------------------------------------------------------------------
;; ----------- MAIN LOOP -----------------------------------------------
;; --------------------------------------------------------------------

jMainLoop clrwdt


;;====================== XMIT CODE ====================================

          jmpClr    tbTxMsg,jMain10     ; not time to xmit next CAN msg
;yet
          bcf       tbTxMsg             ; reset xmit flag

     ;; Reload counter
          movlw     2              ; 65 mS units
          movwf     bXmitClk       ; Countdown to xmit next message

          jmpFeqL   bXmitID,0xFF,jMain10 ; Transmission turned off

     ;; Time to xmit next CAN message.  Select source of message and ID
     ;; to use for transmission

;; <<<<< Analog Input Board >>>>>

          jmpFeqL   bNextMsgType,0,Xmit3201
          jmpFeqL   bNextMsgType,1,XmitRA0
          goto      jMain8

;********** POT ********************************************************

Xmit3201
          call      Read3201            ; read 3201 AD
          movfw     bSwXmitID           ; Use DIP Tx address for transmission
          movwf     bXmitID
          incf      bNextMsgType,F      ; Next time use next source and ID
          goto      jMain8

XmitRA0                                 ;; Use Pot as input
          movlw     dA2DRA0             ; fosc/8 clk, RA0, A/D off
          call      ReadA2D             ; Read A/D port in W
          movfw     bSwXmitID           ; Use DIP Tx address for transmission
          addlw     0x01
          movwf     bXmitID
          clrf      bNextMsgType        ; clear next message
          goto      jMain8

jMain8

     ;; Wait for pending messages to be sent (ALL BUFFERS)
          bL2bV     0x08,b2510RegMask
          movlw     TXB0CTRL
          call      WaitANDeqZ

     ;; Send CAN message with
     ;;    ID = bXmitID
     ;;    Two data bytes: iA2DValue,iA2DValue+1

          SPI_WriteV TXB0SIDH,bXmitID    ; Message ID
          SPI_WriteL TXB0SIDL,0x00       ; Send message - lower bits 0
          SPI_WriteL TXB0DLC,0x02        ; 2 data bytes

     ;; Send least significant byte first
          SPI_WriteV TXB0D0,iA2DValue
          SPI_WriteV TXB0D1,iA2DValue+1
          SPI_Rts   RTS0                ; Transmit buffer 0
```

```
jMain10

;;====================== RECEIVE CODE =================================

          call      CheckCANMsg

          jmpClr    tbRxMsgPend,jMainLoop

     ;; new CAN message received

          call      ParseCAN
          bcf       tbRxMsgPend         ; new CAN message received
          goto      jMainLoop

;**********************************************************************
;OutputPWM
;       OutputPWM - Uses PWM1 output.
;
;**********************************************************************
OutputPWM
          movfw     bPWMValue   ; 8 bit PWM value
          movwf     iA
          clrf      iA+1

     ;; W = 0 - 255.  Load into PWM register.
     ;; load LOWER 2 bits into CCP1CON bits 5,4
     ;; load UPPER 6 bits (shifted right by 2 ) into CCPR1
     ;; this is high res 8 bit mode (upper 2 bits of 10 bit word are
;zero)
          movf     iA,W       ; low byte to W
          clrf     iA+1
          rrf      iA,F       ; low bit to carry
          rrf      iA+1,F      ; move carry into upper bit
          rrf      iA,F       ; low bit to carry
          rrf      iA+1,F      ; move carry into upper bit
          rrf      iA+1,F      ; move to 6,5
          rrf      iA+1,W      ; move to 5,4 in W
          andlw    B'00110000'   ; mask other bits
          iorlw    B'00001100'   ; turn on PWM mode
          movwf    CCP1CON       ; set PWM1 and lower 2 bits
     ;; get upper 6 bits
          movf     iA,W
          andlw    B'00111111'   ; mask upper 2 bits
          movwf    CCPR1L

          return
```

```
;**********************************************************************
;ReadA2D
;         This functions reads analog input and stores the result
;         in iA2DValue as a 12 bit value.  Value in W is used to set
;         ADCON0 to select correct port.
;**********************************************************************
ReadA2D

     ;; setup A/D to select port, etc
          movwf     ADCON0

     ;; turn on A/D
          bsf      ADCON0,0        ;A/D on
     ;; allow 50us for settling
          movlw     25
          movwf     bCnt
ReadAD10
          decfsz    bCnt,F
          goto      ReadAD10

     ;; begin conversion
          bsf       ADCON0,2           ; GO
ReadAD20  jmpSet    ADCON0,2,ReadAD20   ; wait for done bit

          movf      ADRES,W
          movwf     iA2DValue
          clrf      iA2DValue+1

     ;; Convert to 12 bit
          iShiftL   iA2DValue
          iShiftL   iA2DValue
          iShiftL   iA2DValue
          iShiftL   iA2DValue
          return
```

```
;**********************************************************************
;ParseCAN <<<<<INPUTS>>>>>
;         Parse message. Assumes message is two byte 12 bit data.
;         Uses bBaseRecID and bRecIDNext to accept message for PWM
;output.
;**********************************************************************
ParseCAN

;********** Analog In Board *******************************************

          movlw     b'00001111'
          andwf     iRecID_H,W
          movwf     iA
          jmpFeqL   iA,0x0,IDx0             ;check for ID x0
          goto      jParCANRet

IDx0                                              ;; x0 input send PWM
;to lamp
     ;; 12 bits of data
          bV2bV     pRecDataBase,iRecValue
          bV2bV     pRecDataBase+1,iRecValue+1

     ;; new PWM value pending

          bV2bV     iRecValue,iA
          bV2bV     iRecValue+1,iA+1              ; 12 bit received value

     ;; convert to 8 for PWM out
          iShiftR   iA
          iShiftR   iA
          iShiftR   iA
          iShiftR   iA

     ;; Convert to 8 bit
          bV2bV     iA,bPWMValue
          call      OutputPWM
          goto      jParCANRet

jParCANRet
          return
```

```
;**********************************************************************
;Init2510
;*  Function:   Init_MCP2515()
;*      Place MCP2515 initialization here...
;**********************************************************************
Init2510
     ;; Reset MCP2515
          call      Reset2510

     ;; set CLKOUT prescaler to div by 4
          bL2bV     0x03,b2510RegMask
          bL2bV     0x02,b2510RegData
          movlw     CANCTRL
          call      BitMod2510

;Set physical layer configuration
;
;     Fosc = 16MHz
;     BRP        =   7  (divide by 8)
;     Sync Seg   = 1TQ
;     Prop Seg   = 1TQ
;     Phase Seg1 = 3TQ
;     Phase Seg2 = 3TQ
;
;    TQ = 2 * (1/Fosc) * (BRP+1)
;     Bus speed = 1/(Total # of TQ) * TQ
;
          SPI_WriteL CNF1,0x07           ; set BRP to div by 8

;#define BTLMODE_CNF3    0x80
;#define SMPL_1X         0x00
;#define PHSEG1_3TQ      0x10
;#define PRSEG_1TQ       0x00
          SPI_WriteL CNF2,0x90

;#define PHSEG2_3TQ      0x02
          SPI_WriteL CNF3,0x02

;
     ;; Configure Receive buffer 0 Mask and Filters
     ;; Receive buffer 0 will not be used
          SPI_WriteL RXM0SIDH,0xFF
          SPI_WriteL RXM0SIDL,0xFF

          SPI_WriteL RXF0SIDH,0xFF
          SPI_WriteL RXF0SIDL,0xFF

          SPI_WriteL RXF1SIDH,0xFF
          SPI_WriteL RXF1SIDL,0xFF

     ;; Configure Receive Buffer 1 Mask and Filters
          SPI_WriteL RXM1SIDH,0xFF
          SPI_WriteL RXM1SIDL,0xE0

     ;; Initialize Filter 2 to match x0 bBaseRecID from  DIP switch
          SPI_WriteV RXF2SIDH,bBaseRecID
          SPI_WriteL RXF2SIDL,0x00       ; Make sure EXIDE bit (bit 3)
;is set correctly in filter

     ;; Initialize Filter 3 to match x1 from DIP switch
          incf       bBaseRecID,F
          SPI_WriteV RXF3SIDH,bBaseRecID
          SPI_WriteL RXF3SIDL,0x00

     ;; Initialize Filter 4 to match x2 from DIP switch
```

```
          incf       bBaseRecID,F
          SPI_WriteV RXF4SIDH,bBaseRecID
          SPI_WriteL RXF4SIDL,0x00

     ;; Initialize Filter 5 to match x3 from DIP switch
          incf       bBaseRecID,F
          SPI_WriteV RXF5SIDH,bBaseRecID
          SPI_WriteL RXF5SIDL,0x00

          movlw     b'11110000'
          andwf     bBaseRecID,F

     ;; Disable all MCP2515 Interrupts
          bL2bV     0x00,b2510RegData
          movlw     CANINTE
          call      Wrt2510Reg

     ;; Sets normal mode
          call      SetNormalMode
          return

;**********************************************************************
;ProcessSPI
;
;**********************************************************************
ProcessSPI
          skipSet   bSPICnt,2
     ;; buffer not full yet
          return

     ;; disable SPI interupt
          BANK1
          bcf       _SSPIE_P       ; SSP int enable (BANK 1)
          BANK0


     ;; enable SPI
          BANK1
          bsf       _SSPIE_P  ; SSP int enable (BANK 1)
          BANK0
          return


;**********************************************************************
;WaitMSec
;   Delay W number of Msec Routines (255 max)
;         Actually slightly larger than 1 mS
;**********************************************************************

WaitMSec
    movwfbCnt  ;store Msec -> bCnt

jWaitMSec0
    clrwdt         ;clear wdt

; TMR1H: 256 uSec tics with maximum of 1/2 rollover = 32.768 msec
;maximum
          Set1HClock bGenClk,4          ; 256 uS

jWaitMSec1
          jmp1HNotYet bGenClk,jWaitMSec1

          decfsz    bCnt,F
          goto      jWaitMSec0
    return
```

```
;
;**********************************************************************
;Init
;         Initialize
;
;**********************************************************************
Init
          clrwdt                ; required before changing wdt to timer0

     ;; clear peripheral interrupts
    BANK1
          clrf      PIE1_P

     ;; OPTION_REG: PortB Pullups on.
     ;; no prescale for WDT -> should always > 7 mSec  ( 18 mS nominal)
     ;; Timer 0:  Use 64 prescale for 0.27127 * 64 = 17.361 uSec tics

          movlw     B'01000101'        ; Timer0 prescale 64
          movwf     OPTION_REG_P

     ;; clear bank 0
          movlw     0x20
          movwf     FSR
jInitClr1 clrf      INDF
          incf      FSR,F
          jmpClr    FSR,7,jInitClr1

     ;; clear bank 1
          movlw     0xA0
          movwf     FSR
jInitClr2 clrf      INDF
          incf      FSR,F
          jmpSet    FSR,7,jInitClr2

          call      InitIO              ;initalize IO of microcontroller

     ;; configure Timer1:
          BANK0
          movlw     B'00000001'     ; Prescale = 1, Timer enabled
          movwf     T1CON

    BANK1
          bsf       _TMR1IE_P      ; timer1 rollover interrupt enable
;(page 1)
    BANK0

     ;; init output PWM1 ( uses timer2 )
    BANK0
            ;; Timer2 ( 8 bit timer ) set for
    movlw    B'00000100'     ; prescale of 1, internal clk, enable
;timer2
    movwf    T2CON
              ; load PWM counter(PR2) with 0x3F ( 8 bit high res mode)
              ; this gives a 15.625KHz signal with 4MHz crystal
    BANK1
    movlw    0x3F
    movwf    PR2_P
    BANK0


     ;; for testing
          clrf      TMR1L
          clrf      TMR1H
     ;; turn on interrupts
    BANK0
          movlw    B'11000000'     ; Enable interrupts ( Periphrals
;only )
          movwf    INTCON

          return


;; INITIALIZE I/O OF MICROCONTROLLER

InitIO
          BANK1
           movlw    b'00000100'    ;turn on A/D conversion RA0, RA1, RA3
           movwf    ADCON1_P

    ;; Port A
    ;;      0  in     <*>A2D input POT<*>
    ;;      1  out(1) MCP2515 chip select
    ;;      2  in     <*>open<*>
    ;;      3  in     <*>open<*>
    ;;      4  in     <*>open<*>
    ;;      5  out(1) RST MCP2515

          BANK0
    movlwB'00000010';; initialize Port A outputs
           movfw  PORTA
          BANK1
           movlw   B'11111101'
           movwf   TRISA_P         ;; set Port A


    ;; Port B
    ;;      0  in     Interrupt from MCP2515
    ;;      1  out(1) <*>CS' MCP3201<*>
    ;;      2  out(1) <*>CS' 25C04<*>
    ;;      3  in     <*>open<*>
    ;;      4  in     <*>open<*>
    ;;      5  in     RX0BF from MCP2515
    ;;      6  in     ICD
    ;;      7  in     ICD

          BANK0
    movlwB'00000110';; initialize Port B outputs
           movfw  PORTB
          BANK1
           movlw   B'11111001'
          movwf     TRISB_P         ;; set Port B

    ;; Port C
    ;;      0  in     DIP #1
    ;;      1  in     DIP #2
    ;;      2  out(0) <*>PWM Out<*>
    ;;      3  out(0) SPI clock - master
    ;;      4  in     SPI data in
    ;;      5  out(0) SPI data out
    ;;      6  in     DIP #3
    ;;      7  in     DIP #4

          BANK0
          movlw     B'00000000'
          movwf     PORTC
          BANK1

          movlw     B'11010011'
          movwf     TRISC_P          ;; set Port C

          BANK0
          return

 ifdef ROBUST
;; robust design - force WDT reset
          FILL (goto WDTReset1),(0xFFF-$)
WDTReset1  goto      WDTReset1
 endif


         END
```

```
;---------------------------------------------------------------------
;MCP2515.INC
; Description:  This file contains the definitions for the MicroChip
; standalone CANbus controller.
;
; 07/17/99 JPF Original Version
; 09/11/99 JCT Modified for ASM
;---------------------------------------------------------------------

#define RXF0SIDH0x00
#define RXF0SIDL0x01
#define RXF0EID80x02
#define RXF0EID00x03
#define RXF1SIDH0x04
#define RXF1SIDL0x05
#define RXF1EID80x06
#define RXF1EID00x07
#define RXF2SIDH0x08
#define RXF2SIDL0x09
#define RXF2EID80x0A
#define RXF2EID00x0B
#define BFPCTRL0x0C
#define TXRTSCTRL0x0D
#define CANSTAT0x0E
#define CANCTRL0x0F

#define RXF3SIDH0x10
#define RXF3SIDL0x11
#define RXF3EID80x12
#define RXF3EID00x13
#define RXF4SIDH0x14
#define RXF4SIDL0x15
#define RXF4EID80x16
#define RXF4EID00x17
#define RXF5SIDH0x18
#define RXF5SIDL0x19
#define RXF5EID80x1A
#define RXF5EID00x1B
#define TEC0x1C
#define REC0x1D
#define CANSTAT10x1E
#define CANCTRL10x1F

#define RXM0SIDH0x20
#define RXM0SIDL0x21
#define RXM0EID80x22
#define RXM0EID00x23
#define RXM1SIDH0x24
#define RXM1SIDL0x25
#define RXM1EID80x26
#define RXM1EID00x27
#define CNF30x28
#define CNF20x29
#define CNF10x2A
#define CANINTE0x2B
#define CANINTF0x2C
#define EFLG0x2D
#define CANSTAT20x2E
#define CANCTRL20x2F

#define TXB0CTRL0x30
#define TXB0SIDH0x31
#define TXB0SIDL0x32
#define TXB0EID80x33
#define TXB0EID00x34
```

```
#define TXB0DLC0x35
#define TXB0D00x36
#define TXB0D10x37
#define TXB0D20x38
#define TXB0D30x39
#define TXB0D40x3A
#define TXB0D50x3B
#define TXB0D60x3C
#define TXB0D70x3D
#define CANSTAT30x3E
#define CANCTRL30x3F

#define TXB1CTRL0x40
#define TXB1SIDH0x41
#define TXB1SIDL0x42
#define TXB1EID80x43
#define TXB1EID00x44
#define TXB1DLC0x45
#define TXB1D00x46
#define TXB1D10x47
#define TXB1D20x48
#define TXB1D30x49
#define TXB1D40x4A
#define TXB1D50x4B
#define TXB1D60x4C
#define TXB1D70x4D
#define CANSTAT40x4E
#define CANCTRL40x4F

#define TXB2CTRL0x50
#define TXB2SIDH0x51
#define TXB2SIDL0x52
#define TXB2EID80x53
#define TXB2EID00x54
#define TXB2DLC0x55
#define TXB2D00x56
#define TXB2D10x57
#define TXB2D20x58
#define TXB2D30x59
#define TXB2D40x5A
#define TXB2D50x5B
#define TXB2D60x5C
#define TXB2D70x5D
#define CANSTAT50x5E
#define CANCTRL50x5F

#define RXB0CTRL0x60
#define RXB0SIDH0x61
#define RXB0SIDL0x62
#define RXB0EID80x63
#define RXB0EID00x64
#define RXB0DLC0x65
#define RXB0D00x66
#define RXB0D10x67
#define RXB0D20x68
#define RXB0D30x69
#define RXB0D40x6A
#define RXB0D50x6B
#define RXB0D60x6C
#define RXB0D70x6D
#define CANSTAT60x6E
#define CANCTRL60x6F

#define RXB1CTRL0x70
#define RXB1SIDH0x71
```

```
#define RXB1SIDL0x72
#define RXB1EID80x73
#define RXB1EID00x74
#define RXB1DLC0x75
#define RXB1D00x76
#define RXB1D10x77
#define RXB1D20x78
#define RXB1D30x79
#define RXB1D40x7A
#define RXB1D50x7B
#define RXB1D60x7C
#define RXB1D70x7D
#define CANSTAT70x7E
#define CANCTRL70x7F

;; Bit definitions

;; Bit definitionsBFPCTRL
#define trB1BFSBFPCTRL,5
#define trB0BFSBFPCTRL,4
#define trB1BFEBFPCTRL,3
#define trB0BFEBFPCTRL,2
#define trB1BFMBFPCTRL,1
#define trB0BFMBFPCTRL,0

;; Bit definitionsTXRTSCTRL
#define trB2RTSBFPCTRL,5
#define trB1RTSBFPCTRL,4
#define trB0RTSBFPCTRL,3
#define trB2RTSMBFPCTRL,2
#define trB1RTSMBFPCTRL,1
#define trB0RTSMBFPCTRL,0

;; Bit definitionsCANSTAT
#define trOPMOD2CANSTAT,7
#define trOPMOD1CANSTAT,6
#define trOPMOD0CANSTAT,5
#define trICOD2CANSTAT,3
#define trICOD1CANSTAT,2
#define trICOD0CANSTAT,1

;; Bit definitionsCANCTRL
#define trREQOP2CANCTRL,7
#define trREQOP1CANCTRL,6
#define trREQOP0CANCTRL,5
#define trABATCANCTRL,4
#define trCLKENCANCTRL,2
#define trCLKPRE1CANCTRL,1
#define trCLKPRE0CANCTRL,0

;; Dit definitionsCNF3
#define trWAKFILCNF3,6
#define trPHSEG22CNF3,2
#define trPHSEG21CNF3,1
#define trPHSEG20CNF3,0

;; Bit definitionsCNF2
#define trBTLMODECNF2,7
#define trSAMCNF2,6
#define trPHSEG12CNF2,5
#define trPHSEG11CNF2,4
#define trPHSEG10CNF2,3
#define trPHSEG2CNF2,2
#define trPHSEG1CNF2,1
#define trPHSEG0CNF2,0
```

```
;; Bit definitionsCNF1
#define trSJW1CNF1,7
#define trSJW0CNF1,6
#define trBRP5CNF1,5
#define trBRP4CNF1,4
#define trBRP3CNF1,3
#define trBRP2CNF1,2
#define trBRP1CNF1,1
#define trBRP0CNF1,0

;; Bit definitions CANINTE
#define trMERRECANINTE,7
#define trWAKIECANINTE,6
#define trERRIECANINTE,5
#define trTX2IECANINTE,4
#define trTX1IECANINTE,3
#define trTX0IECANINTE,2
#define trRX1IECANINTE,1
#define trRX0IECANINTE,0

;; Bit definitions CANINTF
#define trMERRFCANINTF,7
#define trWAKIFCANINTF,6
#define trERRIFCANINTF,5
#define trTX2IFCANINTF,4
#define trTX1IFCANINTF,3
#define trTX0IFCANINTF,2
#define trRX1IFCANINTF,1
#define trRX0IFCANINTF,0

;; Bit definitions EFLG
#define trRX1OVREFLG,7
#define trRX0OVREFLG,6
#define trTXB0EFLG,5
#define trTXEPEFLG,4
#define trRXEPEFLG,3
#define trTXWAREFLG,2
#define trRXWAREFLG,1
#define trEWARNEFLG,0

;; Bit definitions TXB0CTRL
#define trABTF0TXB0CTRL,6
#define trMLOA0TXB0CTRL,5
#define trTXERR0TXB0CTRL,4
#define trTXREQ0TXB0CTRL,3
#define trTXP10TXB0CTRL,1
#define trTXP00TXB0CTRL,0

;; Bit definitions TXB1CTRL
#define trABTF1TXB1CTRL,6
#define trMLOA1TXB1CTRL,5
#define trTXERR1TXB1CTRL,4
#define trTXREQ1TXB1CTRL,3
#define trTXP11TXB1CTRL,1
#define trTXP01TXB1CTRL,0

;; Bit definitions TXB2CTRL
#define trABTF2TXB2CTRL,6
#define trMLOA2TXB2CTRL,5
#define trTXERR2TXB2CTRL,4
#define trTXREQ2TXB2CTRL,3
#define trTXP12TXB2CTRL,1
#define trTXP02TXB2CTRL,0
```

```
;; Bit definitions RXB0CTRL
#define trRXM10RXB0CTRL,6
#define trRXM00RXB0CTRL,5
#define trRXRTR0RXB0CTRL,3
#define trBUKT01RXB0CTRL,2
#define trBUKT00RXB0CTRL,1
#define trFILHIT00RXB0CTRL,0

;; Bit definitionsRXB1CTRL
#define trRXM11RXB1CTRL,6
#define trRXM01RXB1CTRL,5
#define trRXRTR1RXB1CTRL,3
#define trFILHIT12RXB1CTRL,2
#define trFILHIT11RXB1CTRL,1
#define trFILHIT10RXB1CTRL,0


;; use with SPI_Rts function
#define RTS00x01
#define RTS10x02
#define RTS20x04




;**********************************************************************
;**********************************************************************


; MCP2515 Instructions
#define d2510Rd      0x03      ; MCP2515 read instruction
#define d2510Wrt     0x02      ; MCP2515 write instruction
#define d2510Reset   0xC0      ; MCP2515 reset instruction
#define d2510RTS     0x80      ; MCP2515 RTS instruction
#define d2510Status  0xA0      ; MCP2515 Status instruction
#define d2510BitMod  0x05      ; MCP2515 bit modify instruction
```

```
;**********************************************************************
;*************** SPECIAL CAN MACROS ********************************************
;**********************************************************************

; Read MCP2515 register Reg and return data in W.
SPI_Read macro Reg
          movlw     Reg
          call      Rd2510Reg
          endm

; Write literal byte to MCP2515 register Reg.
SPI_WriteL macro Reg,LitData
          movlw     LitData
          movwf     b2510RegData
          movlw     Reg
          call      Wrt2510Reg
          endm

; Write Data byte to MCP2515 register Reg.
SPI_WriteV macro Reg,RegData
          movfw     RegData
          movwf     b2510RegData
          movlw     Reg
          call      Wrt2510Reg
          endm

; Write W byte to MCP2515 register Reg.
SPI_WriteW macro Reg
          movwf     b2510RegData
          movlw     Reg
          call      Wrt2510Reg
          endm


; Write bits determined by Mask & Data to MCP2515 register Reg.
SPI_BitMod macro Reg,Mask,Data
          movlw     Mask
          movwf     b2510RegMask
          movlw     Data
          movwf     b2510RegData
          movlw     Reg
          call      BitMod2510
          endm

; Arm xmit buffers for xmission
SPI_Rts macro Data
          movlw     Data
          call      Rts2510
          endm
```

```
;**********************************************************************
;**********************************************************************
;Support routines for communicating with MCP2515 chip
;**********************************************************************
;**********************************************************************

;**********************************************************************
;CheckCANMsg
;
; Checks for message in Receive Buf 1.  If no message pending return
; with Z flag set.
;
; If message pending:
;     Load iRecID_L,iRecID_H with ID.
;     Load bRecCount with number of bytes of data received.
;     Load buffer at pRecDataBase with data
;     Clear MCP2515 Receive Buffer 1 interrupt flag
;     Set tbRxMsgPend flag and clear Z flag.
;
; NOTE: If message already pending doesn't check for new message.
;
;**********************************************************************
CheckCANMsg

          bcf       _Z                  ; for return
          skipClr   tbRxMsgPend         ; new CAN message received
          return                        ; Message already pending

     ;; Test for Message pending in Receive Buffer 1
          SPI_Read  CANINTF
          andlw     0x02

          skipNZ
          return              ; Nothing in Rec Buf 1

          bsf       tbRxMsgPend         ; new CAN message received

     ;; Get ID of message source
          SPI_Read  RXB1SIDH
          movwf     iRecID_H
          SPI_Read  RXB1SIDL
          andlw     0xE0
          movwf     iRecID_L

     ;; Get number of bytes of data
          SPI_Read  RXB1DLC
          andlw     0x0F
          movwf     bRecCount

     ;; Get data from buffer. Up to 8 bytes based on
          clrf      bCnt

jRxChk11  jmpFeqF   bCnt,bRecCount,jRxChk90            ; no data left

     ;; Calculate correct MCP2515 receive buffer location
          movlw     RXB1D0
          addwf     bCnt,W

     ;; Get data byte
          call      Rd2510Reg
          movwf     b2510RegData     ; temporary save

     ;; Calculate destination buffer location
          movlw     pRecDataBase
          addwf     bCnt,W
          movwf     FSR

     ;; Store data in buffer
          movfw     b2510RegData     ; temporary save
          movwf     INDF
          incf      bCnt,F
          goto      jRxChk11

jRxChk90
          SPI_BitMod CANINTF,0x02,0     ; Clear receive buffer 1 ;interrupt
          bcf       _Z                  ; signal data pending
          return
```

```
;**********************************************************************
;SetConfigMode
;
;// Function Name: Set_Config_Mode()
;**********************************************************************
SetConfigMode
;  SPI_BitMod(CANCTRL, 0xE0, 0x80);    //Config. mode/
          bL2bV     0xE0,b2510RegMask
          bL2bV     0x80,b2510RegData
          movlw     CANCTRL
          call      BitMod2510

jSetConfigM1
          movlw     CANSTAT
          call      Rd2510Reg
          andlw     0xE0
          xorlw     0x80
          jmpNZ     jSetConfigM1

          return


;**********************************************************************
;SetNormalMode
;
;// Function Name: Set_Normal_Mode()
;**********************************************************************
SetNormalMode

          bL2bV     0xE0,b2510RegMask
          bL2bV     0x00,b2510RegData
          movlw     CANCTRL
          call      BitMod2510

jSetNormalM1
          movlw     CANSTAT
          call      Rd2510Reg
          andlw     0xE0
          jmpNZ     jSetNormalM1

          return

;**********************************************************************
;WaitANDeqZ
;         Wait for byte from address in W to AND with mask in
;         b2510RegMask to be zero. Uses b2510RegAdr to hold address.
;
;**********************************************************************
WaitANDeqZ
          movwf     b2510RegAdr         ; save

jWaitANDeqZ
          movfw     b2510RegAdr         ; save
          call      Rd2510Reg
          andwf     b2510RegMask,W
          jmpNZ     jWaitANDeqZ
          return
```

```
;**********************************************************************
;**********************************************************************


;**********************************************************************
;**************** BASIC COMMUNICATION *********************************
;**********************************************************************


;**********************************************************************
;Get2510Status
;         Get Status byte from MCP2515.
;// Function Name: SPI_ReadStatus()
;**********************************************************************
Get2510Status
          call      InitSPIBuf
          movlw     d2510Status         ; MCP2515 Status instruction
          call      LoadSPIByte
          movlw     1                   ; expect 1 byte answer
          call      LoadSPIZeros
          call      ExchangeSPI
          call      WaitSPIExchange
          return

;**********************************************************************
;Rd2510Reg
;         Read MCP2515 register at address in W. Return results
;         in W. Uses b2510RegAdr to hold address.
;// Function Name: SPI_Read(uint address)
;**********************************************************************
Rd2510Reg
          movwf     b2510RegAdr         ; save
          call      InitSPIBuf
          movlw     d2510Rd             ; MCP2515 read instruction
          call      LoadSPIByte
          movfw     b2510RegAdr         ; get address
          call      LoadSPIByte
          movlw     1                   ; expect 1 byte answer
          call      LoadSPIZeros
          call      ExchangeSPI
          call      WaitSPIExchange
          movfw     pSPIBufBase+2
          return

;**********************************************************************
;Wrt2510Reg
;         Write byte in b2510RegData to MCP2515 register at location in W.
;         Uses b2510RegAdr to hold address.
;// Function Name: SPI_Write(uint address)
;**********************************************************************
Wrt2510Reg
          movwf     b2510RegAdr         ; save
          call      InitSPIBuf
          movlw     d2510Wrt            ; MCP2515 write instruction
          call      LoadSPIByte
          movfw     b2510RegAdr         ; get address
          call      LoadSPIByte
          movfw     b2510RegData        ; get data
          call      LoadSPIByte
          call      ExchangeSPI
          call      WaitSPIExchange
          return
```

```
;**********************************************************************
;BitMod2510
;// Function Name: SPI_BitMod()
;         Write data in b2510RegData using mask in b2510RegMask to
;         address in W. Uses b2510RegAdr to hold address.
;**********************************************************************
BitMod2510
          movwf     b2510RegAdr         ; save
          call      InitSPIBuf

          movlw     d2510BitMod         ; MCP2515 bit modify ;instruction
          call      LoadSPIByte

          movfw     b2510RegAdr         ; address
          call      LoadSPIByte

          movfw     b2510RegMask        ; mask
          call      LoadSPIByte

          movfw     b2510RegData        ; data
          call      LoadSPIByte

          call      ExchangeSPI
          call      WaitSPIExchange
          return


;**********************************************************************
;Rts2510
;         Request to send to MCP2515.
;         Send the request to send instruction to the CANbus Controller ORed
;         with value in W.  Uses b2510RegData.
;// Function Name: SPI_Reset()
;**********************************************************************
Rts2510
          movwf     b2510RegData
          call      InitSPIBuf

          movlw     d2510RTS            ; MCP2515 RTS instruction
          iorwf     b2510RegData,W      ; get data and OR it with RTS
          call      LoadSPIByte

          call      ExchangeSPI
          call      WaitSPIExchange
          return


;**********************************************************************
;Reset2510
;         Reset MCP2515.
;// Function Name: SPI_Reset()
;**********************************************************************
Reset2510
          call      InitSPIBuf
          movlw     d2510Reset           ; MCP2515 reset instruction
          call      LoadSPIByte
          call      ExchangeSPI
          call      WaitSPIExchange
          return
```

```
;**********************************************************************
;***************** LOCAL - DON'T CALL DIRECTLY ************************
;**********************************************************************

;**********************************************************************
;InitSPIPort
;         Intialize SPI port
;**********************************************************************
InitSPIPort
    BANK0
          bcf       _SSPEN         ; disable SPI
          movlw     0x11           ; SPI Master, Idle high, Fosc/16
          movwf     SSPCON
          bsf       _SSPEN         ; enable SPI
          bcf       _SSPIF         ; clear interrupt flag
          BANK1
          bsf       _SSPIE_P       ; SSP int enable (BANK 1)
          BANK0
          return

;**********************************************************************
;InitSPIBuf
;         Initializes SPI buffer for transaction.  Sets up
;         FSR as buffer pointer.
;**********************************************************************
InitSPIBuf
          clrf      bSPICnt
          movlw     pSPIBufBase
          movwf     pSPIBuf
          movwf     FSR
          return

;**********************************************************************
;LoadSPIByte
;         Load byte in W to SPI buffer.  Assumes FSR is pointer.
;**********************************************************************
LoadSPIByte
          movwf     INDF
          incf      FSR,F
          return

;**********************************************************************
;LoadSPIZeros
;         Load number of zeros in W to SPI buffer.
;         Assumes FSR is pointer.
;**********************************************************************
LoadSPIZeros
          andlw     0xFF
          skipNZ
          return                        ; finished
          clrf      INDF
          incf      FSR,F
          addlw     0xFF                ; Subtract 1 from W
          jmpNZ     LoadSPIZeros
          return
```

```
;**********************************************************************
;ExchangeSPI
;         Initiate SPI transaction.
;**********************************************************************
ExchangeSPI
     ;; Get number of bytes to exchange
          bV2bV     FSR,bSPICnt
          movlw     pSPIBufBase
          subwf     bSPICnt,F

          skipNZ
          return                        ; nothing to exchange

          movlw     pSPIBufBase
          movwf     pSPIBuf

     ;; Load 1st byte to begin exchange
          bcf       tp2510_CS_          ; CS_ for MCP2515 chip
          movfw     pSPIBufBase         ; get 1st byte in buffer
          movwf     SSPBUF              ; send it
          return


;**********************************************************************
;WaitSPIExchange
;         Wait for SPI transaction to be completed.
;**********************************************************************
WaitSPIExchange
          jmpFneZ   bSPICnt,WaitSPIExchange
          return


;**********************************************************************
;**********************************************************************


; MCP2515 Instructions
#define d2510Rd      0x03      ; MCP2515 read instruction
#define d2510Wrt     0x02      ; MCP2515 write instruction
#define d2510Reset   0xC0      ; MCP2515 reset instruction
#define d2510RTS     0x80      ; MCP2515 RTS instruction
#define d2510Status  0xA0      ; MCP2515 Status instruction
#define d2510BitMod  0x05      ; MCP2515 bit modify instruction
```

```
;**********************************************************************
;*************** SPECIAL CAN MACROS ********************************************
;**********************************************************************

; Read MCP2515 register Reg and return data in W.
SPI_Read macro Reg
          movlw     Reg
          call      Rd2510Reg
          endm

; Write literal byte to MCP2515 register Reg.
SPI_WriteL macro Reg,LitData
          movlw     LitData
          movwf     b2510RegData
          movlw     Reg
          call      Wrt2510Reg
          endm

; Write Data byte to MCP2515 register Reg.
SPI_WriteV macro Reg,RegData
          movfw     RegData
          movwf     b2510RegData
          movlw     Reg
          call      Wrt2510Reg
          endm

; Write W byte to MCP2515 register Reg.
SPI_WriteW macro Reg
          movwf     b2510RegData
          movlw     Reg
          call      Wrt2510Reg
          endm


; Write bits determined by Mask & Data to MCP2515 register Reg.
SPI_BitMod macro Reg,Mask,Data
          movlw     Mask
          movwf     b2510RegMask
          movlw     Data
          movwf     b2510RegData
          movlw     Reg
          call      BitMod2510
          endm

; Arm xmit buffers for xmission
SPI_Rts macro Data
          movlw     Data
          call      Rts2510
          endm
```

```
;**********************************************************************
;**********************************************************************
;Support routines for communicating with MCP2515 chip
;**********************************************************************
;**********************************************************************

;**********************************************************************
;CheckCANMsg
;
; Checks for message in Receive Buf 1.  If no message pending return
; with Z flag set.
;
; If message pending:
;     Load iRecID_L,iRecID_H with ID.
;     Load bRecCount with number of bytes of data received.
;     Load buffer at pRecDataBase with data
;     Clear MCP2515 Receive Buffer 1 interrupt flag
;     Set tbRxMsgPend flag and clear Z flag.
;
; NOTE: If message already pending doesn't check for new message.
;
;**********************************************************************
CheckCANMsg

          bcf       _Z                  ; for return
          skipClr   tbRxMsgPend         ; new CAN message received
          return                        ; Message already pending

     ;; Test for Message pending in Receive Buffer 1
          SPI_Read  CANINTF
          andlw     0x02

          skipNZ
          return                        ; Nothing in Rec Buf 1

          bsf       tbRxMsgPend         ; new CAN message received

     ;; Get ID of message source
          SPI_Read  RXB1SIDH
          movwf     iRecID_H
          SPI_Read  RXB1SIDL
          andlw     0xE0
          movwf     iRecID_L

     ;; Get number of bytes of data
          SPI_Read  RXB1DLC
          andlw     0x0F
          movwf     bRecCount

     ;; Get data from buffer. Up to 8 bytes based on
          clrf      bCnt

jRxChk11  jmpFeqF   bCnt,bRecCount,jRxChk90            ; no data left

     ;; Calculate correct MCP2515 receive buffer location
          movlw     RXB1D0
          addwf     bCnt,W

     ;; Get data byte
          call      Rd2510Reg
          movwf     b2510RegData     ; temporary save

     ;; Calculate destination buffer location
          movlw     pRecDataBase
          addwf     bCnt,W
          movwf     FSR

     ;; Store data in buffer
          movfw     b2510RegData     ; temporary save
          movwf     INDF
          incf      bCnt,F
          goto      jRxChk11

jRxChk90
          SPI_BitMod CANINTF,0x02,0     ; Clear receive buffer 1 ;interrupt
          bcf       _Z                  ; signal data pending
          return
```

```
;**********************************************************************
;SetConfigMode
;
;// Function Name: Set_Config_Mode()
;**********************************************************************
SetConfigMode
;  SPI_BitMod(CANCTRL, 0xE0, 0x80);    //Config. mode/
          bL2bV     0xE0,b2510RegMask
          bL2bV     0x80,b2510RegData
          movlw     CANCTRL
          call      BitMod2510

jSetConfigM1
          movlw     CANSTAT
          call      Rd2510Reg
          andlw     0xE0
          xorlw     0x80
          jmpNZ     jSetConfigM1

          return


;**********************************************************************
;SetNormalMode
;
;// Function Name: Set_Normal_Mode()
;**********************************************************************
SetNormalMode

          bL2bV     0xE0,b2510RegMask
          bL2bV     0x00,b2510RegData
          movlw     CANCTRL
          call      BitMod2510

jSetNormalM1
          movlw     CANSTAT
          call      Rd2510Reg
          andlw     0xE0
          jmpNZ     jSetNormalM1

          return

;**********************************************************************
;WaitANDeqZ
;         Wait for byte from address in W to AND with mask in
;         b2510RegMask to be zero. Uses b2510RegAdr to hold address.
;
;**********************************************************************
WaitANDeqZ
          movwf     b2510RegAdr         ; save

jWaitANDeqZ
          movfw     b2510RegAdr         ; save
          call      Rd2510Reg
          andwf     b2510RegMask,W
          jmpNZ     jWaitANDeqZ
          return
```

```
;**********************************************************************
;**********************************************************************


;**********************************************************************
;**************** BASIC COMMUNICATION *********************************
;**********************************************************************


;**********************************************************************
;Get2510Status
;         Get Status byte from MCP2515.
;// Function Name: SPI_ReadStatus()
;**********************************************************************
Get2510Status
          call      InitSPIBuf
          movlw     d2510Status         ; MCP2515 Status instruction
          call      LoadSPIByte
          movlw     1                   ; expect 1 byte answer
          call      LoadSPIZeros
          call      ExchangeSPI
          call      WaitSPIExchange
          return

;**********************************************************************
;Rd2510Reg
;         Read MCP2515 register at address in W. Return results
;         in W. Uses b2510RegAdr to hold address.
;// Function Name: SPI_Read(uint address)
;**********************************************************************
Rd2510Reg
          movwf     b2510RegAdr         ; save
          call      InitSPIBuf
          movlw     d2510Rd             ; MCP2515 read instruction
          call      LoadSPIByte
          movfw     b2510RegAdr         ; get address
          call      LoadSPIByte
          movlw     1                   ; expect 1 byte answer
          call      LoadSPIZeros
          call      ExchangeSPI
          call      WaitSPIExchange
          movfw     pSPIBufBase+2
          return

;**********************************************************************
;Wrt2510Reg
;         Write byte in b2510RegData to MCP2515 register at location in W.
;         Uses b2510RegAdr to hold address.
;// Function Name: SPI_Write(uint address)
;**********************************************************************
Wrt2510Reg
          movwf     b2510RegAdr         ; save
          call      InitSPIBuf
          movlw     d2510Wrt            ; MCP2515 write instruction
          call      LoadSPIByte
          movfw     b2510RegAdr         ; get address
          call      LoadSPIByte
          movfw     b2510RegData        ; get data
          call      LoadSPIByte
          call      ExchangeSPI
          call      WaitSPIExchange
          return
```

```
;**********************************************************************
;BitMod2510
;// Function Name: SPI_BitMod()
;         Write data in b2510RegData using mask in b2510RegMask to
;         address in W. Uses b2510RegAdr to hold address.
;**********************************************************************
BitMod2510
          movwf     b2510RegAdr         ; save
          call      InitSPIBuf

          movlw     d2510BitMod         ; MCP2515 bit modify ;instruction
          call      LoadSPIByte

          movfw     b2510RegAdr         ; address
          call      LoadSPIByte

          movfw     b2510RegMask        ; mask
          call      LoadSPIByte

          movfw     b2510RegData        ; data
          call      LoadSPIByte

          call      ExchangeSPI
          call      WaitSPIExchange
          return


;**********************************************************************
;Rts2510
;         Request to send to MCP2515.
;         Send the request to send instruction to the CANbus Controller ORed
;         with value in W.  Uses b2510RegData.
;// Function Name: SPI_Reset()
;**********************************************************************
Rts2510
          movwf     b2510RegData
          call      InitSPIBuf

          movlw     d2510RTS            ; MCP2515 RTS instruction
          iorwf     b2510RegData,W      ; get data and OR it with RTS
          call      LoadSPIByte

          call      ExchangeSPI
          call      WaitSPIExchange
          return


;**********************************************************************
;Reset2510
;         Reset MCP2515.
;// Function Name: SPI_Reset()
;**********************************************************************
Reset2510
          call      InitSPIBuf
          movlw     d2510Reset           ; MCP2515 reset instruction
          call      LoadSPIByte
          call      ExchangeSPI
          call      WaitSPIExchange
          return
```

```
;**********************************************************************
;***************** LOCAL - DON'T CALL DIRECTLY ************************
;**********************************************************************

;**********************************************************************
;InitSPIPort
;         Intialize SPI port
;**********************************************************************
InitSPIPort
    BANK0
          bcf       _SSPEN         ; disable SPI
          movlw     0x11           ; SPI Master, Idle high, Fosc/16
          movwf     SSPCON
          bsf       _SSPEN         ; enable SPI
          bcf       _SSPIF         ; clear interrupt flag
          BANK1
          bsf       _SSPIE_P       ; SSP int enable (BANK 1)
          BANK0
          return

;**********************************************************************
;InitSPIBuf
;         Initializes SPI buffer for transaction.  Sets up
;         FSR as buffer pointer.
;**********************************************************************
InitSPIBuf
          clrf      bSPICnt
          movlw     pSPIBufBase
          movwf     pSPIBuf
          movwf     FSR
          return

;**********************************************************************
;LoadSPIByte
;         Load byte in W to SPI buffer.  Assumes FSR is pointer.
;**********************************************************************
LoadSPIByte
          movwf     INDF
          incf      FSR,F
          return

;**********************************************************************
;LoadSPIZeros
;         Load number of zeros in W to SPI buffer.
;         Assumes FSR is pointer.
;**********************************************************************
LoadSPIZeros
          andlw     0xFF
          skipNZ
          return                        ; finished
          clrf      INDF
          incf      FSR,F
          addlw     0xFF                ; Subtract 1 from W
          jmpNZ     LoadSPIZeros
          return
```

```
;**********************************************************************
;ExchangeSPI
;         Initiate SPI transaction.
;**********************************************************************
ExchangeSPI
     ;; Get number of bytes to exchange
          bV2bV     FSR,bSPICnt
          movlw     pSPIBufBase
          subwf     bSPICnt,F

          skipNZ
          return                        ; nothing to exchange

          movlw     pSPIBufBase
          movwf     pSPIBuf

     ;; Load 1st byte to begin exchange
          bcf       tp2510_CS_          ; CS_ for MCP2515 chip
          movfw     pSPIBufBase         ; get 1st byte in buffer
          movwf     SSPBUF              ; send it
          return


;**********************************************************************
;WaitSPIExchange
;         Wait for SPI transaction to be completed.
;**********************************************************************
WaitSPIExchange
          jmpFneZ   bSPICnt,WaitSPIExchange
          return
```

```
;-----------------------------------------------------------------------
;MCP2515.INC
; Description:  This file contains the definitions for the MicroChip
; standalone CANbus controller.
;
; 07/17/99 JPF Original Version
; 09/11/99 JCT Modified for ASM
;-----------------------------------------------------------------------

#define RXF0SIDH0x00
#define RXF0SIDL0x01
#define RXF0EID80x02
#define RXF0EID00x03
#define RXF1SIDH0x04
#define RXF1SIDL0x05
#define RXF1EID80x06
#define RXF1EID00x07
#define RXF2SIDH0x08
#define RXF2SIDL0x09
#define RXF2EID80x0A
#define RXF2EID00x0B
#define BFPCTRL0x0C
#define TXRTSCTRL0x0D
#define CANSTAT0x0E
#define CANCTRL0x0F

#define RXF3SIDH0x10
#define RXF3SIDL0x11
#define RXF3EID80x12
#define RXF3EID00x13
#define RXF4SIDH0x14
#define RXF4SIDL0x15
#define RXF4EID80x16
#define RXF4EID00x17
#define RXF5SIDH0x18
#define RXF5SIDL0x19
#define RXF5EID80x1A
#define RXF5EID00x1B
#define TEC0x1C
#define REC0x1D
#define CANSTAT10x1E
#define CANCTRL10x1F

#define RXM0SIDH0x20
#define RXM0SIDL0x21
#define RXM0EID80x22
#define RXM0EID00x23
#define RXM1SIDH0x24
#define RXM1SIDL0x25
#define RXM1EID80x26
#define RXM1EID00x27
#define CNF30x28
#define CNF20x29
#define CNF10x2A
#define CANINTE0x2B
#define CANINTF0x2C
#define EFLG0x2D
#define CANSTAT20x2E
#define CANCTRL20x2F

#define TXB0CTRL0x30
#define TXB0SIDH0x31
#define TXB0SIDL0x32
#define TXB0EID80x33
#define TXB0EID00x34
```

```
#define TXB0DLC0x35
#define TXB0D00x36
#define TXB0D10x37
#define TXB0D20x38
#define TXB0D30x39
#define TXB0D40x3A
#define TXB0D50x3B
#define TXB0D60x3C
#define TXB0D70x3D
#define CANSTAT30x3E
#define CANCTRL30x3F

#define TXB1CTRL0x40
#define TXB1SIDH0x41
#define TXB1SIDL0x42
#define TXB1EID80x43
#define TXB1EID00x44
#define TXB1DLC0x45
#define TXB1D00x46
#define TXB1D10x47
#define TXB1D20x48
#define TXB1D30x49
#define TXB1D40x4A
#define TXB1D50x4B
#define TXB1D60x4C
#define TXB1D70x4D
#define CANSTAT40x4E
#define CANCTRL40x4F

#define TXB2CTRL0x50
#define TXB2SIDH0x51
#define TXB2SIDL0x52
#define TXB2EID80x53
#define TXB2EID00x54
#define TXB2DLC0x55
#define TXB2D00x56
#define TXB2D10x57
#define TXB2D20x58
#define TXB2D30x59
#define TXB2D40x5A
#define TXB2D50x5B
#define TXB2D60x5C
#define TXB2D70x5D
#define CANSTAT50x5E
#define CANCTRL50x5F

#define RXB0CTRL0x60
#define RXB0SIDH0x61
#define RXB0SIDL0x62
#define RXB0EID80x63
#define RXB0EID00x64
#define RXB0DLC0x65
#define RXB0D00x66
#define RXB0D10x67
#define RXB0D20x68
#define RXB0D30x69
#define RXB0D40x6A
#define RXB0D50x6B
#define RXB0D60x6C
#define RXB0D70x6D
#define CANSTAT60x6E
#define CANCTRL60x6F

#define RXB1CTRL0x70
#define RXB1SIDH0x71
```

```
#define RXB1SIDL0x72
#define RXB1EID80x73
#define RXB1EID00x74
#define RXB1DLC0x75
#define RXB1D00x76
#define RXB1D10x77
#define RXB1D20x78
#define RXB1D30x79
#define RXB1D40x7A
#define RXB1D50x7B
#define RXB1D60x7C
#define RXB1D70x7D
#define CANSTAT70x7E
#define CANCTRL70x7F

;; Bit definitions

;; Bit definitions BFPCTRL
#define trB1BFSBFPCTRL,5
#define trB0BFSBFPCTRL,4
#define trB1BFEBFPCTRL,3
#define trB0BFEBFPCTRL,2
#define trB1BFMBFPCTRL,1
#define trB0BFMBFPCTRL,0

;; Bit definitions TXRTSCTRL
#define trB2RTSBFPCTRL,5
#define trB1RTSBFPCTRL,4
#define trB0RTSBFPCTRL,3
#define trB2RTSMBFPCTRL,2
#define trB1RTSMBFPCTRL,1
#define trB0RTSMBFPCTRL,0

;; Bit definitions CANSTAT
#define trOPMOD2CANSTAT,7
#define trOPMOD1CANSTAT,6
#define trOPMOD0CANSTAT,5
#define trICOD2CANSTAT,3
#define trICOD1CANSTAT,2
#define trICOD0CANSTAT,1

;; Bit definitions CANCTRL
#define trREQOP2CANCTRL,7
#define trREQOP1CANCTRL,6
#define trREQOP0CANCTRL,5
#define trABATCANCTRL,4
#define trCLKENCANCTRL,2
#define trCLKPRE1CANCTRL,1
#define trCLKPRE0CANCTRL,0

;; Dit definitions CNF3
#define trWAKFILCNF3,6
#define trPHSEG22CNF3,2
#define trPHSEG21CNF3,1
#define trPHSEG20CNF3,0

;; Bit definitions CNF2
#define trBTLMODECNF2,7
#define trSAMCNF2,6
#define trPHSEG12CNF2,5
#define trPHSEG11CNF2,4
#define trPHSEG10CNF2,3
#define trPHSEG2CNF2,2
#define trPHSEG1CNF2,1
#define trPHSEG0CNF2,0
```

```
;; Bit definitions CNF1
#define trSJW1CNF1,7
#define trSJW0CNF1,6
#define trBRP5CNF1,5
#define trBRP4CNF1,4
#define trBRP3CNF1,3
#define trBRP2CNF1,2
#define trBRP1CNF1,1
#define trBRP0CNF1,0

;; Bit definitions CANINTE
#define trMERRECANINTE,7
#define trWAKIECANINTE,6
#define trERRIECANINTE,5
#define trTX2IECANINTE,4
#define trTX1IECANINTE,3
#define trTX0IECANINTE,2
#define trRX1IECANINTE,1
#define trRX0IECANINTE,0

;; Bit definitions CANINTF
#define trMERRFCANINTF,7
#define trWAKIFCANINTF,6
#define trERRIFCANINTF,5
#define trTX2IFCANINTF,4
#define trTX1IFCANINTF,3
#define trTX0IFCANINTF,2
#define trRX1IFCANINTF,1
#define trRX0IFCANINTF,0

;; Bit definitions EFLG
#define trRX1OVREFLG,7
#define trRX0OVREFLG,6
#define trTXB0EFLG,5
#define trTXEPEFLG,4
#define trRXEPEFLG,3
#define trTXWAREFLG,2
#define trRXWAREFLG,1
#define trEWARNEFLG,0

;; Bit definitions TXB0CTRL
#define trABTF0TXB0CTRL,6
#define trMLOA0TXB0CTRL,5
#define trTXERR0TXB0CTRL,4
#define trTXREQ0TXB0CTRL,3
#define trTXP10TXB0CTRL,1
#define trTXP00TXB0CTRL,0

;; Bit definitions TXB1CTRL
#define trABTF1TXB1CTRL,6
#define trMLOA1TXB1CTRL,5
#define trTXERR1TXB1CTRL,4
#define trTXREQ1TXB1CTRL,3
#define trTXP11TXB1CTRL,1
#define trTXP01TXB1CTRL,0

;; Bit definitions TXB2CTRL
#define trABTF2TXB2CTRL,6
#define trMLOA2TXB2CTRL,5
#define trTXERR2TXB2CTRL,4
#define trTXREQ2TXB2CTRL,3
#define trTXP12TXB2CTRL,1
#define trTXP02TXB2CTRL,0
```

```
;; Bit definitions RXB0CTRL
#define trRXM10RXB0CTRL,6
#define trRXM00RXB0CTRL,5
#define trRXRTR0RXB0CTRL,3
#define trBUKT01RXB0CTRL,2
#define trBUKT00RXB0CTRL,1
#define trFILHIT00RXB0CTRL,0

;; Bit definitions RXB1CTRL
#define trRXM11RXB1CTRL,6
#define trRXM01RXB1CTRL,5
#define trRXRTR1RXB1CTRL,3
#define trFILHIT12RXB1CTRL,2
#define trFILHIT11RXB1CTRL,1
#define trFILHIT10RXB1CTRL,0


;; use with SPI_Rts function
#define RTS00x01
#define RTS10x02
#define RTS20x04
```

```
;Basic macros for PIC16C series
;6/20/98

#ifdef __16C77
#define _COMMONBANK  ; use common upper 16 bytes in 4 banks
#endif
#ifdef __16C76
#define _COMMONBANK  ; use common upper 16 bytes in 4 banks
#endif

TRUE            equ     1
FALSE           equ     0

; Page 1 register definitions to avoid page warning

OPTION_REG_P                 EQU     H'0081'-0x80
TRISA_P                      EQU     H'0085'-0x80
TRISB_P                      EQU     H'0086'-0x80
TRISC_P                      EQU     H'0087'-0x80
TRISD_P                      EQU     H'0088'-0x80
TRISE_P                      EQU     H'0089'-0x80
PIE1_P                       EQU     H'008C'-0x80
PIE2_P                       EQU     H'008D'-0x80
PCON_P                       EQU     H'008E'-0x80
PR2_P                        EQU     H'0092'-0x80
SSPADD_P                     EQU     H'0093'-0x80
SSPSTAT_P                    EQU     H'0094'-0x80
TXSTA_P                      EQU     H'0098'-0x80
SPBRG_P                      EQU     H'0099'-0x80
ADCON1_P                     EQU     H'009F'-0x80



; Special register bit definition pairs

;     STATUS bit definitions

#define _C              STATUS,0
#define _DC             STATUS,1
#define _Z              STATUS,2
#define _PD             STATUS,3
#define _TO             STATUS,4
#define _RP0            STATUS,5
#define _RP1            STATUS,6
#define _IRP            STATUS,7


#define _INTE       INTCON,INTE   ; External interrupt enable
#define _INTF       INTCON,INTF   ; External interrupt flag
#define _RBIE       INTCON,RBIE   ; Port B pins 4-7 edge interrupt enable
#define _RBIF       INTCON,RBIF   ; Port B pins 4-7 edge interrupt flag
#define _T0IE       INTCON,T0IE   ; Timer 0 interrupt enable
#define _T0IF       INTCON,T0IF   ; Timer 0 interrupt flag

#define _CCP1IE_P   PIE1_P,CCP1IE ; Timer 1 compare int enable (page 1)
#define _CCP1IF     PIR1,CCP1IF   ; Timer 1 compare int flag

#define _RCIE_P     PIE1_P,RCIE   ; async rec interrupt enable (page 1)
#define _RCIF       PIR1,RCIF     ; async rec interrupt flag

#define _TXIE_P     PIE1_P,TXIE   ; async xmit interrupt enable (page
;1)
#define _TXIF       PIR1,TXIF     ; async xmit interrupt flag

#define _SSPIE_P    PIE1_P,SSPIE  ; SSP int enable (page 1)
```

```
#define _SSPIF      PIR1,SSPIE    ; SSP interrupt flag

#define _TMR1IE_P   PIE1_P,TMR1IE ; Timer 1 enable (page 1)
#define _TMR1IF     PIR1,TMR1IF   ; Timer1 interrupt flag

#define _TMR2IE_P   PIE1_P,TMR2IE ; Timer 2 enable (page 1)
#define _TMR2IF     PIR1,TMR2IF   ; Timer2 interrupt flag

#ifdef _COMMONBANK ; use common upper 16 bytes in 4 banks

PAGE3   macro
        bsf     PCLATH,4
        bsf     PCLATH,3
        endm

PAGE2   macro
        bsf     PCLATH,4
        bcf     PCLATH,3
        endm

PAGE1   macro
        bcf     PCLATH,4
        bsf     PCLATH,3
        endm

PAGE0   macro
        bcf     PCLATH,4
        bcf     PCLATH,3
        endm


BANK3    macro
        bsf      STATUS,6
        bsf      STATUS,5
        endm

BANK2    macro
        bsf      STATUS,6
        bcf      STATUS,5
        endm

BANK1    macro
        bcf      STATUS,6
        bsf      STATUS,5
        endm

BANK0    macro
        bcf      STATUS,6
        bcf      STATUS,5
        endm


FSRBank23 macro
        bsf      STATUS,7
        endm

FSRBank01 macro
        bcf      STATUS,7
        endm
#else


PAGE1   macro
        bsf     PCLATH,3
        endm
```

```
PAGE0  macro
       bcf     PCLATH,3
       endm

BANK0   macro
       bcf     STATUS,5    ; Select page 0
       endm

BANK1   macro
       bsf     STATUS,5    ; Select page 1
       endm
#endif

enableInt macro
       bsf     INTCON,GIE
       endm

disableInt macro
          local    Loop
Loop    bcf     INTCON,GIE
        btfsc   INTCON,GIE
        goto    Loop
       endm



; Byte logical & arithmetic macros

bV2bV   macro   bSource,bDest
        movf    bSource,W
        movwf   bDest
        endm

bL2bV   macro   bVal,bDest
        movlw   bVal
        movwf   bDest
        endm

jmpFeqZ macro   Reg,Label
        movf    Reg,F
        btfsc   _Z
        goto    Label
        endm

jmpFneZ macro   Reg,Label
        movf    Reg,F
        btfss   _Z
        goto    Label
        endm

jmpFgtL macro   Reg1,bVal,Label
          movfw     Reg1
          jmpWgtL   bVal,Label
    endm

jmpFgeL macro   Reg1,bVal,Label
          movfw     Reg1
          jmpWgeL   bVal,Label
    endm

jmpFeqL macro   Reg,bVal,Label
    movf    Reg,W
          sublw   bVal
    btfsc   _Z
    goto    Label
    endm

jmpFneL macro   Reg,bVal,Label
    movf    Reg,W
          sublw   bVal
          btfss   _Z
          goto    Label
          endm

jmpFleL macro   Reg1,bVal,Label
          movfw     Reg1
          jmpWleL   bVal,Label
    endm

jmpFltL macro   Reg1,bVal,Label
          movfw     Reg1
          jmpWltL   bVal,Label
    endm

jmpFeqF macro   Reg1,Reg2,Label
    movf    Reg1,W
        subwf   Reg2,W
    btfsc   _Z
    goto    Label
    endm

jmpFneF macro   Reg1,Reg2,Label
    movf    Reg1,W
        subwf   Reg2,W
    btfss   _Z
    goto    Label
    endm

jmpFleF macro   Reg1,Reg2,Label
          movfw     Reg1
          jmpWleF   Reg2,Label
    endm

jmpFltF macro   Reg1,Reg2,Label
          movfw     Reg1
          jmpWltF   Reg2,Label
    endm

jmpWeqZ macro   Label        ; jmp if W == 0
        andlw   0xFF
        jmpZ    Label
        endm

jmpWneZ macro   Label        ; jmp if W != 0
        andlw   0xFF
        jmpNZ   Label
        endm

skipFeqZ macro   Reg
    movf    Reg,F
    btfss   _Z
    endm

skipFneZ macro   Reg
    movf    Reg,F
    btfsc   _Z
    endm

skipFeqL macro   Reg,bVal
```

```
   movf    Reg,W
       sublw   bVal
   btfss   _Z
   endm

skipFneL macro   Reg,bVal
   movf    Reg,W
       sublw   bVal
   btfsc   _Z
   endm

skipFeqF macro   Reg1,Reg2
   movf    Reg1,W
       subwf   Reg2,W
   btfss   _Z
   endm

skipFneF macro   Reg1,Reg2
   movf    Reg1,W
       subwf   Reg2,W
   btfsc   _Z
   endm

skipWeqZ macro
        andlw   0xFF
        btfss   _Z
        endm

skipWneZ macro
        andlw   0xFF
        btfsc   _Z
        endm

jmpWgtL macro   bVal,Label
   sublw   bVal
   btfss   _C
   goto    Label
   endm

jmpWgeL macro   bVal,Label
   sublw   bVal
   btfss   _C
   goto    Label
   btfsc   _Z
   goto    Label
   endm

jmpWeqL macro   bVal,Label
   sublw   bVal
   btfsc   _Z
   goto    Label
   endm

jmpWneL macro   bVal,Label
   sublw   bVal
   btfss   _Z
   goto    Label
   endm

jmpWleL macro   bVal,Label
   sublw   bVal
   btfsc   _C
   goto    Label
   endm
```

```
jmpWltL macro   bVal,Label
    sublw   bVal
        skipC
        bsf     _Z
    jmpNZLabel
    endm

jmpWgtF macro   Reg,Label
    subwf   Reg,W
    btfss   _C
    goto    Label
    endm

jmpWgeF macro   Reg,Label
    subwf   Reg,W
    btfss   _C
    goto    Label
    btfsc   _Z
    goto    Label
    endm

jmpWeqF macro   Reg,Label
    subwf   Reg,W
    btfsc   _Z
    goto    Label
    endm

jmpWneF macro   Reg,Label
    subwf   Reg,W
    btfss   _Z
    goto    Label
    endm

jmpWleF macro   Reg,Label
    subwf   Reg,W
    btfsc   _C
    goto    Label
    endm

jmpWltF macro   Reg,Label
    subwf   Reg,W
        skipC
        bsf     _Z
    jmpNZ Label
    endm


jmpClr  macro Reg,Bit,Label

    btfss   Reg,Bit
    goto    Label
    endm

jmpSet  macro Reg,Bit,Label
    btfsc   Reg,Bit
    goto    Label
    endm

jmpZ    macro Label
    btfsc   _Z
    goto    Label
    endm

jmpNZ   macro Label
    btfss   _Z
    goto    Label
    endm

jmpC    macro Label
    btfsc   _C
    goto    Label
    endm

jmpNC   macro Label
    btfss   _C
    goto    Label
    endm

skipClr macro Reg,Bit
    btfsc   Reg,Bit
    endm

skipSet macro Reg,Bit
    btfss   Reg,Bit
    endm

skipNZ  macro
    btfsc   _Z
    endm

skipZ   macro
    btfss   _Z
    endm

skipNC  macro
    btfsc   _C
    endm

skipC   macro
    btfss   _C
    endm

toggle  macro Reg,Bit
        local    SLabel,Label
    btfss   Reg,Bit
        goto    SLabel
        bcf     Reg,Bit
        goto    Label
SLabel  bsf     Reg,Bit
Label
        endm


tb2tb macro RegS,BitS,RegD,BitD
          local     jLab1,jLab2
          jmpSet    RegS,BitS,jLab1
          bcf       RegD,BitD
          goto      jLab2
jLab1     bsf       RegD,BitD
jLab2
          endm

tb2Nottb macro RegS,BitS,RegD,BitD
          local     jLab1,jLab2
          jmpClr    RegS,BitS,jLab1
          bcf       RegD,BitD
          goto      jLab2
jLab1     bsf       RegD,BitD
jLab2
          endm
```

```
;**********************************************************************
;Read3201
;         This functions reads MCP3201 and store the result
;         in iA2DValue as a 12 bit value.
;**********************************************************************
Read3201

          bcf       tpA2D_CS_           ; CS_ for 3201 A2D chip

          call      InitSPIBuf
          movlw     2                   ; expect 2 bytes
          call      LoadSPIZeros

     ;; Initiate SPI transaction.
     ;; Get number of bytes to exchange
          bV2bV     FSR,bSPICnt
          movlw     pSPIBufBase
          subwf     bSPICnt,F

          movlw     pSPIBufBase
          movwf     pSPIBuf

     ;; Load 1st byte to begin exchange
          movfw     pSPIBufBase         ; get 1st byte in buffer
          movwf     SSPBUF              ; send it

          call      WaitSPIExchange

          bsf       tpA2D_CS_           ; CS_ for 3201 A2D chip

          bV2bV     pSPIBufBase,iA2DValue+1
          bV2bV     pSPIBufBase+1,iA2DValue

     ;; Shift right by 1 to remove extra b1 bit
          iShiftR   iA2DValue

     ;; remove dummy upper 4 bits
          movlw     0x0F
          andwf     iA2DValue+1,F
          return
```

NOTES:

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE**.** Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Printed on recycled paper.

ISBN: 978-1-60932-648-1

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

MICROCHIP

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-6578-300
Fax: 886-3-6578-370

Taiwan - Kaohsiung
Tel: 886-7-213-7830
Fax: 886-7-330-9305

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

08/04/10