International Journal of
Scientific Engineering and Technology Research

www.ijsetr.com

ISSN 2319-8885
Vol.03,Issue.31
October-2014,
Pages:6105-6111

Implementation of CAN Protocol for Industrial Automation

UBAID SAUDAGAR
Dept of ECE, Shadan College of Engineering and Technology, Peerancheru, Hyderabad,
Email: ubaid.saudagar@gmail.com.

Abstract: CAN protocol is a serial communication protocol and was designed for the automobile industry, but due to the flexibility of the protocol, it has found its way in other industries as well, such as industrial automation. But one thing is noticeable that CAN protocol uses CAN frame and is transmitted over CAN bus, requires CAN controller. Now a day's RISC processors are normally used in various sectors which have on chip CAN controller. Hence protocol development part doesn't come into play. But many industries still use CISC processors and even in the education sector, the curriculum focuses on CISC architecture. Such controllers do not have on chip CAN controller and hence needs to be interfaced externally. Since it is interfaced externally, its internal registers needs to be configured before the CAN frame is actually transmitted over the CAN bus. The aim of this project is to develop the CAN protocol for the automation industry as well as for the education sector where it can be implemented. For achieving this aim the CAN controller which is interfaced externally has different registers which needs to be configured. The configuration of registers is done using SPI protocol. Once the registers are configured the CAN frame is transmitted over the CAN bus at a specified Baud Rate. The protocol is transmitted over two lines viz. CANH and CANL. It is a message based protocol, hence no clock is required. The schematic of the hardware is made using Proteus 8 professional. The software programming is done in C language in Keil µvision 5.

Keywords: CAN Protocol, SPI Protocol, CISC Architecture, Proteus 8 Professional, Keil µvision 5.

I. INTRODUCTION

Controller Area Network (CAN) was initially created by German automotive system supplier Robert Bosch in the mid-1980s for automotive applications as a method for enabling robust serial communication. The goal was to make automobiles more reliable, safe and fuel-efficient while decreasing wiring harness weight and complexity. Since its inception, the CAN protocol has gained widespread popularity in industrial automation and automotive/truck applications. This paper presents the development of a serial communication protocol called the CAN Protocol. In general, RISC processors like ARM family and PIC family come along with on chip CAN controller, which actually generates the CAN frame. In such processors there is no point of development of protocol but just configuring the registers which are on chip. When it comes to CISC processors like 8051 and architectures based on it, CAN controller, in general is not present on chip. In the education industry, students are still taught the CISC architecture, hence development of CAN protocol so that it may work with CISC processors was a need in the education sector.

Secondly, many automation industries still use CISC processors like 89V51RD2. Since CAN protocol has now found its way into automation industry as well, there was a requirement to develop the protocol. In the scope of developing the protocol, we need to externally interface CAN controller and then configure it using the SPI protocol in mode 0,0 or 1,1. To configure the registers of CAN controllers, we need to use different SPI commands like RESET, READ and WRITE etc. and send these commands serially to the CAN controller. Once the registers are configured the CAN frame is transmitted at a specified Baud Rate on the CAN bus via a CAN Transreciever which will generate a differential voltage on the two lines of CAN bus i.e. CANH and CANL. The project aims at monitoring different devices in the system of an automation industry such as motors and LCD. The main aim of the project is to develop CAN protocol for CISC processors. Hence we need to interface externally CAN controller and then configure it using SPI protocol. Since CAN protocol has now found its way into automation industry as well, there was a requirement to develop the protocol. Therefore firstly the CAN controller registers are configured using SPI protocol and thereafter the CAN frame is generated on CAN bus.

CAN bus uses two dedicated wires for communication. The wires are called CAN high and CAN low. When the CAN bus is in idle mode, both lines carry 2.5V. When data bits are being transmitted, the CAN high line goes to 3.75V and the CAN low drops to 1.25V, thereby generating a 2.5V differential between the lines. Since communication relies on a voltage differential between the two bus lines, the CAN bus is not sensitive to inductive spikes, electrical fields or other noise. This makes CAN bus a reliable choice for networked communications on mobile equipment.

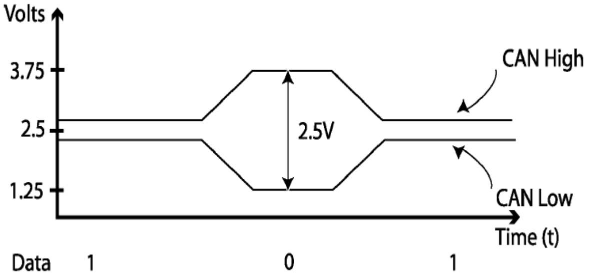


Fig1. Data on CAN bus.

The nature of CAN bus communications allows all modules to transmit and receive data on the bus. Any module can transmit data, which all the rest of the modules receive permitting both peer-to-peer and broadcast data transmissions.CAN bus can use multiple baud rates up to 1 Mbit/s. The most common baud rates are 125 Kbit/s and 250 Kbit/s. CAN is based on the “broadcast communication mechanism”, which is based on a message-oriented transmission protocol. It defines message contents rather than stations and station addresses. Every message has a message identifier, which is unique within the whole network since it defines content and also the priority of the message. The CAN protocol supports two message frame formats, the only essential difference being in the length of the identifier. The “CAN base frame” supports a length of 11 bits for the identifier, and the “CAN extended frame” supports a length of 29 bits for the identifier.

Fig2. CAN message frame.

A CAN base frame message begins with the start bit called "Start of Frame (SOF)", this is followed by the "Arbitration field" which consist of the identifier and the "Remote Transmission Request (RTR)" bit used to distinguish between the data frame and the data request frame called remote frame. The following "Control field" contains the "IDentifier Extension (IDE)" bit to distinguish between the CAN base frame and the CAN extended frame, as well as the "Data Length Code (DLC)" used to indicate the number of following data bytes in the "Data field". If the message is used as a remote frame, the DLC contains the number of requested data bytes. The "Data field" that follows is able to hold up to 8 data byte. The integrity of the frame is guaranteed by the following "Cyclic Redundant Check (CRC)" sum.The end of the message is indicated by "End of Frame (EOF)". The "Intermission Frame Space (IFS)" is the minimum number of bits separating consecutive messages. Unless another station starts transmitting, the bus remains idle after this.

II. HARDWARE DESIGN

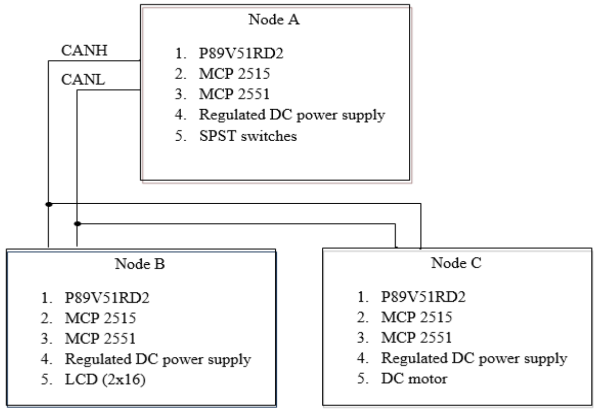


Fig3. Hardware overview.

The system consists of both hardware as well as software part. The hardware part consists of 3 nodes viz. Node A, Node B and Node C. The designing of the hardware includes the interfacing of components, designing of power supply and the value of components to be used in the designing process. After the designing process, schematic is prepared using Proteus 8 Professional.

A. Power supply design

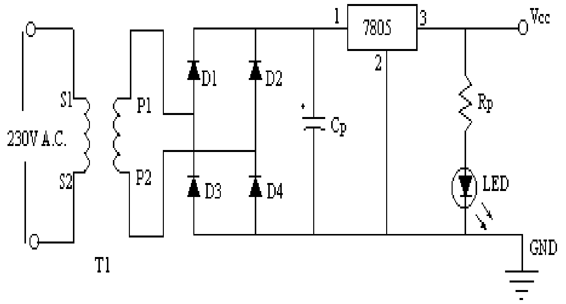


Fig4. Power supply circuit

i. Size of core is one of the first considerations in regard of weight and volume of transformer.

$$A_i = \sqrt{\frac{P_1}{0.87}} \qquad (1)$$

Ai = Area of cross - section in Sq. cm. and

P_1 = Primary voltage.

In transformer $P_1 = P_2$

For our project we required +5V regulated output. So transformer secondary rating is 12V, 500mA.

So secondary power wattage is,

$P_2 = 12 \times 500 \times 10^{-3}$ W= 6W.

Therefore Ai = 2.62

ii. Turns per volt of transformer are given by relation

$$\text{Turns / Volt} = \frac{10{,}000}{4.44\ f\ \text{Bm Ai}} \qquad (2)$$

Here,

f is the frequency in Hz

Bm is flux density in Wb/m^2

Ai is net area of cross section.

For project for 50 H_Z the turns per Volt for 0.91 Wb/m^2

Turns per Volt = 50 / Ai = 50 / 2.88

≅ 17

Thus for Primary winding = 220 x 17 = 3800

ForSecondary winding = 12 x 17 = 204

iii. R.M.S. Secondary voltage at secondary of transformer is 12V.So maximum voltage Vm across Secondary is

= Rms. Voltage x √2

= 12 x √2

= 16.97

D.C. O/p Voltage at rectifier O/p is

$$\text{Vdc} = \frac{2\ \text{Vm}}{\pi} \qquad (3)$$

$= (2 \times 16.97) / \pi$

$= 10.80$ V

PIV rating of each diode is

PIV = 2 Vm.

= 2 x 16.97

= 34 V

iv. Formula for calculating filter capacitor is,

$$C = \frac{1}{4\sqrt{3}\ r\ f\ R_L} \qquad (4)$$

r = ripple present at o/p of rectifier (Which is maximum 0.1 for full wave rectifier.)

F = frequency of mains A.C.

R_L = I/p impedance of voltage regulator IC.

$$C = \frac{1}{4\sqrt{3} \times 0.1 \times 50 \times 28} \qquad (5)$$

= 1030 μf

≅ 1000 μf

v. IC 7805 (Voltage Regulator IC)

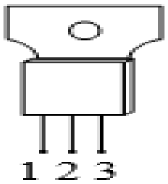


Fig5. Voltage regulator 7805.

Specifications:

Available o/p D.C. Voltage = + 5V.

Line Regulation = 0.03

Load Regulation= 0.5

V_{in} maximum= 35 V

Ripple Rejection= 66-80 (db)

B. Interfacing LCD display (2x16)

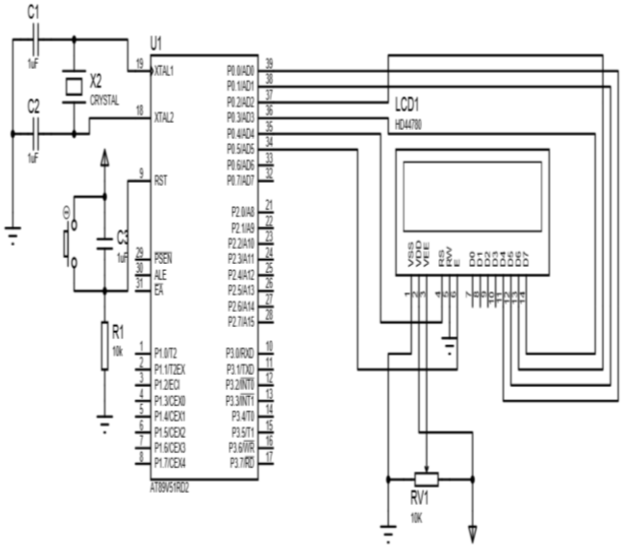


Fig6. Interfacing of LCD display with μc

LCD display contains one microcontroller which controls the various operations of LCD display. The microcontroller takes some time to execute the command. So as to synchronize the operation of LCD display with the microcontroller of target board some delay is introduced between the commands. The microcontroller also has RAM which stores the content of the cell, this is the reason that even after switching off the display using command its content is retained when it is switched ON again.

1. 4-bit programming of LCD

In 4-bit mode the data is sent in nibbles, first we send the higher nibble and then the lower nibble. To enable the 4-bit mode of LCD, we need to follow special sequence of initialization that tells the LCD controller that user has selected 4-bit mode of operation. We call this special sequence as resetting the LCD. Following is the reset sequence of LCD.

- Wait for about 20mS

- Send the first init value (0x30)
- Wait for about 10mS
- Send second init value (0x30)
- Wait for about 1mS
- Send third init value (0x30)
- Wait for 1mS
- Select bus width (0x30 - for 8-bit and 0x20 for 4-bit)
- Wait for 1mS

C. Interfacing DC motor

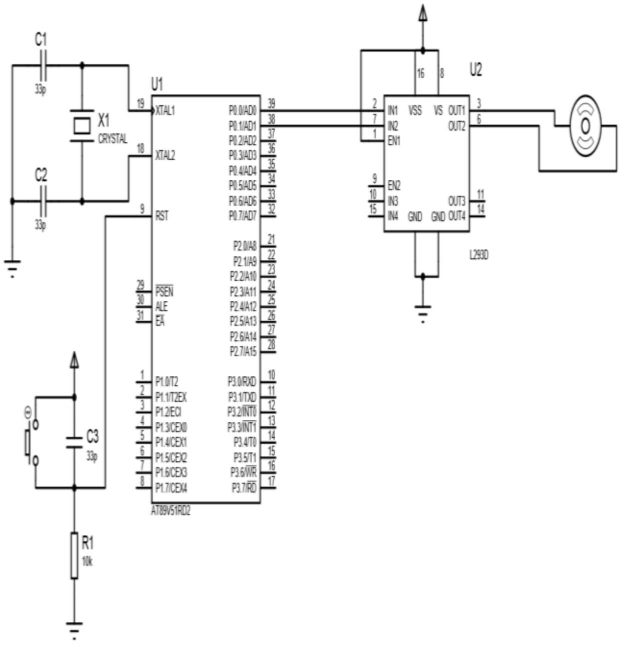


Fig7. DC motor interfaced with μc using L293D driver.

As you can see in the circuit, three pins are needed for interfacing a DC motor (A, B, Enable). If you want the o/p to be enabled completely then you can connect Enable to VCC and only 2 pins needed from controller to make the motor work.

Table1. Truth table for controlling DC motor

A	B	Description
0	0	Motor stops or breaks
0	1	Motor runs Anti clock wise
1	0	Motor runs Clock wise
1	1	Motor stops or breaks

As per the truth table the microcontroller is programmed either to rotate the motor or to stop the motor.

D. Interfacing SPST switch

As shown in the schematic above 4 SPST switches are interfaced with the microcontroller on P0.0 – P0.3. Ideally when the switches are in open state, logic 1 appears on the port pins. Whenever a switch is pressed, logic 0 appears on the port pins. Our aim is to send a message to the LCD which is at Node B when the first switch is pressed. When the second switch is pressed a second message is sent to the LCD over the CAN bus.

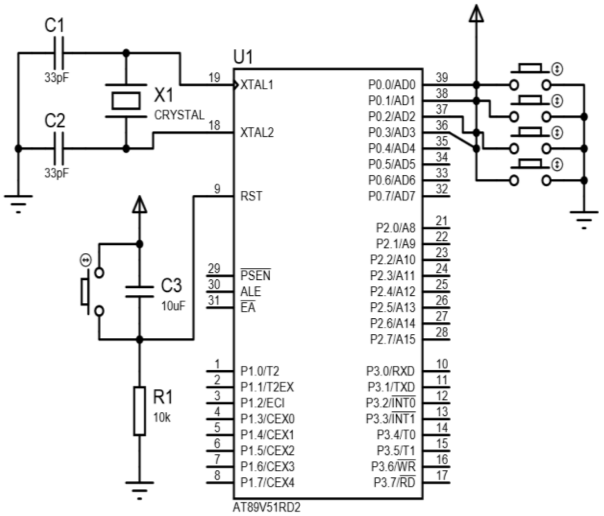


Fig8. SPST switches interfaced with Microcontroller.

On pressing the third switch, the DC motor switches on and rotates in the clockwise direction. When the fourth switch is pressed the DC motor stops rotating.

E. Interfacing P89V51RD2 with MCP2515 and MCP 2551

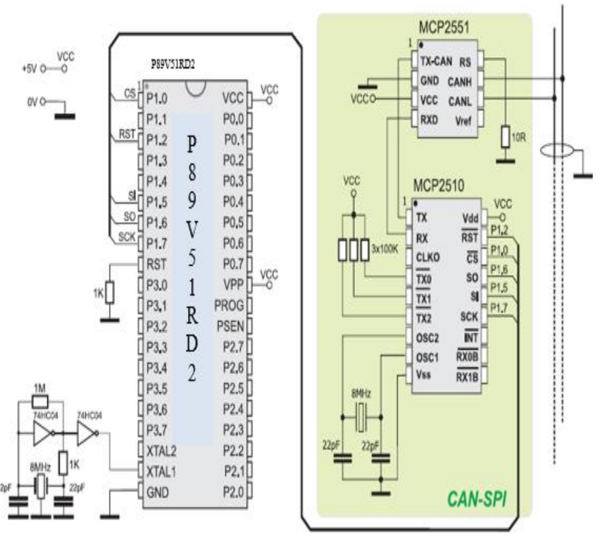


Fig9. Connecting CAN – SPI module with P89V51RD2

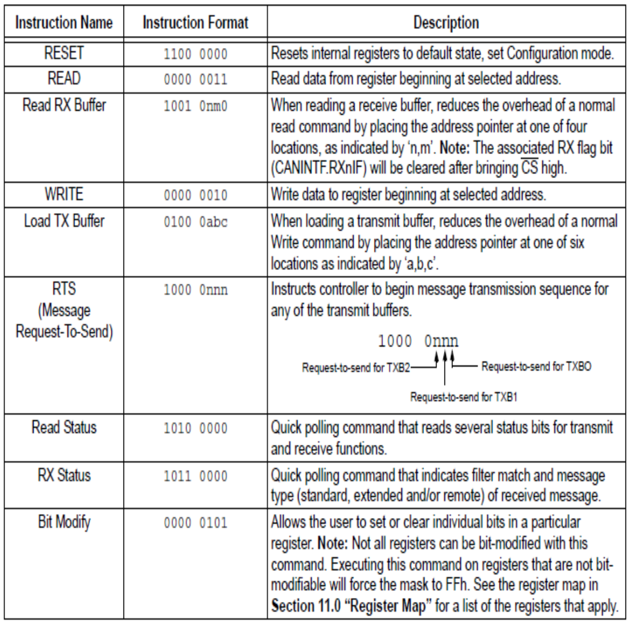
MCP2515 is a standalone CAN controller which is interfaced externally to microcontroller having CISC architecture. Controllers based on RISC architectures can also be interfaced, but RISC processors normally come with on chip CAN controller. The MCP2515 has numerous registers which needs to be configured before the actual CAN frame is transmitted over the CAN bus. To configure these registers from the microcontroller, we need to develop SPI protocol in mode 0,0 or 1,1 and using SPI RESET, WRITE commands, the registers are configured as per the datasheet. Once the registers are configured the

CAN frame which includes CAN ID of destination, DATA and DLC (Data length code). The frame is given to the CAN Transreciever which generates a differential voltage on the CAN bus. The receiver side also has the same module.

III. SOFTWARE DESIGN

The MCP2515 is designed to interface directly with the Serial Peripheral Interface (SPI) port available on many microcontrollers and supports Mode 0, 0 and Mode 1, 1. Commands and data are sent to the device via the SI pin, with data being clocked in on the rising edge of SCK. Data is driven out by the MCP2515 (on the SO line) on the falling edge of SCK. The CS pin must be held low while any operation is performed. The various SPI commands are tabulated below:

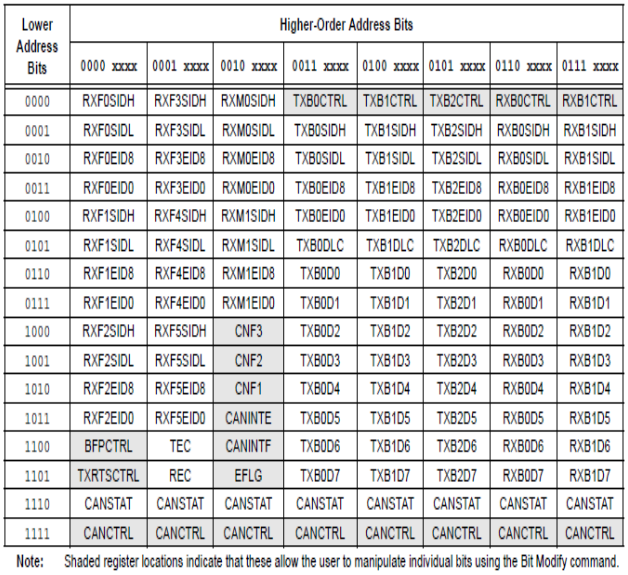
Table2. SPI instruction set



Instruction Name	Instruction Format	Description
RESET	1100 0000	Resets internal registers to default state, set Configuration mode.
READ	0000 0011	Read data from register beginning at selected address.
Read RX Buffer	1001 0nm0	When reading a receive buffer, reduces the overhead of a normal read command by placing the address pointer at one of four locations, as indicated by 'n,m'. **Note:** The associated RX flag bit (CANINTF.RXnIF) will be cleared after bringing $\overline{CS}$ high.
WRITE	0000 0010	Write data to register beginning at selected address.
Load TX Buffer	0100 0abc	When loading a transmit buffer, reduces the overhead of a normal Write command by placing the address pointer at one of six locations as indicated by 'a,b,c'.
RTS (Message Request-To-Send)	1000 0nnn	Instructs controller to begin message transmission sequence for any of the transmit buffers. 1000 0nnn Request-to-send for TXB2 Request-to-send for TXBO Request-to-send for TXB1
Read Status	1010 0000	Quick polling command that reads several status bits for transmit and receive functions.
RX Status	1011 0000	Quick polling command that indicates filter match and message type (standard, extended and/or remote) of received message.
Bit Modify	0000 0101	Allows the user to set or clear individual bits in a particular register. **Note:** Not all registers can be bit-modified with this command. Executing this command on registers that are not bit-modifiable will force the mask to FFh. See the register map in **Section 11.0 "Register Map"** for a list of the registers that apply.

The various registers which needs to be configured using SPI commands are tabulated below:

Table3. CAN controller Register Map



Lower Address Bits	Higher-Order Address Bits							
	0000 xxxx	0001 xxxx	0010 xxxx	0011 xxxx	0100 xxxx	0101 xxxx	0110 xxxx	0111 xxxx
0000	RXF0SIDH	RXF3SIDH	RXM0SIDH	TXB0CTRL	TXB1CTRL	TXB2CTRL	RXB0CTRL	RXB1CTRL
0001	RXF0SIDL	RXF3SIDL	RXM0SIDL	TXB0SIDH	TXB1SIDH	TXB2SIDH	RXB0SIDH	RXB1SIDH
0010	RXF0EID8	RXF3EID8	RXM0EID8	TXB0SIDL	TXB1SIDL	TXB2SIDL	RXB0SIDL	RXB1SIDL
0011	RXF0EID0	RXF3EID0	RXM0EID0	TXB0EID8	TXB1EID8	TXB2EID8	RXB0EID8	RXB1EID8
0100	RXF1SIDH	RXF4SIDH	RXM1SIDH	TXB0EID0	TXB1EID0	TXB2EID0	RXB0EID0	RXB1EID0
0101	RXF1SIDL	RXF4SIDL	RXM1SIDL	TXB0DLC	TXB1DLC	TXB2DLC	RXB0DLC	RXB1DLC
0110	RXF1EID8	RXF4EID8	RXM1EID8	TXB0D0	TXB1D0	TXB2D0	RXB0D0	RXB1D0
0111	RXF1EID0	RXF4EID0	RXM1EID0	TXB0D1	TXB1D1	TXB2D1	RXB0D1	RXB1D1
1000	RXF2SIDH	RXF5SIDH	CNF3	TXB0D2	TXB1D2	TXB2D2	RXB0D2	RXB1D2
1001	RXF2SIDL	RXF5SIDL	CNF2	TXB0D3	TXB1D3	TXB2D3	RXB0D3	RXB1D3
1010	RXF2EID8	RXF5EID8	CNF1	TXB0D4	TXB1D4	TXB2D4	RXB0D4	RXB1D4
1011	RXF2EID0	RXF5EID0	CANINTE	TXB0D5	TXB1D5	TXB2D5	RXB0D5	RXB1D5
1100	BFPCTRL	TEC	CANINTF	TXB0D6	TXB1D6	TXB2D6	RXB0D6	RXB1D6
1101	TXRTSCTRL	REC	EFLG	TXB0D7	TXB1D7	TXB2D7	RXB0D7	RXB1D7
1110	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT
1111	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL

Note: Shaded register locations indicate that these allow the user to manipulate individual bits using the Bit Modify command.

i. Software for Node A

Node A consists of a Node controller i.e. Microcontroller (based on 8051 architecture), CAN controller (MCP2510/15), CAN Transreciever (MCP2551), SPST switches. Once the CAN registers are configured the CAN frame is transmitted over the CAN bus with CAN_ID, DATA, and DLC (Data length code).

```
#include <stdio.h>
#include "89C51.h"
#include "Delay.h"
#include "MCP2510.h"
unsigned char MSG1[8]="Welcome.";
unsigned char MSG2[8]="CAN PRO.";
void main(void)
{
        mcp2510Init(250);
        canInit();
        while(1)
        {
                If (P0_0==0)
                {
                        canWrite (333, MSG1, 8);
                }
                if(P0_1==0)
                {
                        canWrite (333, MSG2, 8);
                }
                if(P0_2==0)
                {
                        canWrite (222, "O", 1);
                }
                if(P0_3==0)
                {
                        canWrite (222, "F", 1);
                }
        }
}
```

ii. *Software for Node B*

Node B consists of a Node controller i.e. Microcontroller (based on 8051 architecture), CAN controller (MCP2510/15), CAN Transreciever (MCP2551), LCD display (2x16).

```
#include <stdio.h>
#include "89C51.h"
#include "Delay.h"
#include "LCD4NW.h"
#include "MCP2510.h"
unsigned int n, id;
unsigned char i, RData[8], dlc;
int main(void)
{
        SetLCD();
        LCD(0);
        mcp2510Init(250);
        canInit();
        LCD(1);
        printf("Rxd Message.....");
        while(1)
        {
        if((n=canPoll())!=-1)
        {
                dlc = canRead(n, RData, &id);
                if(id==333)
                {
                        LCD(2);
                        for(i=0;i<dlc;i++)
                        {
                        printf("%c",RData[i]);
                        }
                }
        }
        }
}
```

iii. *Software for Node C*

Node C consists of a *Node controller* i.e. Microcontroller (based on 8051 architecture), CAN controller (MCP2510 /15), CAN Transreciever (MCP2551), DC motor along with its driver L293D.

```
#include <stdio.h>
#include "89C51.h"
#include "Delay.h"
#include "MCP2510.h"
unsigned int n, id;
unsigned char i, RData[8], dlc;
int main(void)
{
        //motor state initially in stop state
        P0_0=0;
        P0_1=0;
        mcp2510Init(250);
        Beep(1,300);
        canInit();
        Beep(1,300);
        while(1)
        {
                if((n=canPoll())!=-1)
                {
                dlc = canRead(n, RData, &id);
                if(id==222)
                {
                if(RData[0]=='O')
                {
                P0_0=1;
                P0_1=0;
                }
                if(RData[0]=='F')
                {
                P0_0=0;
                P0_1=0;
                }
                        }
                }
  }
}
```

IV. RESULTS

On pressing switch 1, a low signal is received at P0_0 which is identified in the program and a message is sent to Node B and Node C having CAN ID = 333, MSG1 = “Welcome.”, this message is send over the CAN bus. Since Node C is having CAN ID = 333, it accepts the frame and displays on the LCD screen. In the same way it happens for switch 2, but now the message changes and displays “CAN PRO.”.

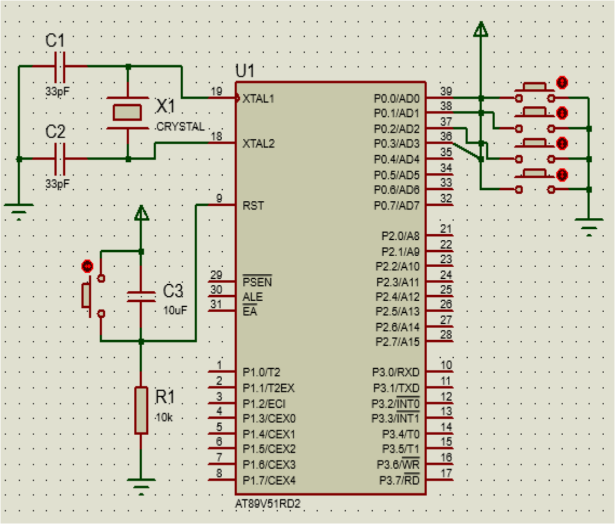


Fig10. Switch 1 is pressed.

On pressing switch 3, a low signal is received at P0_2 which is identified in the program and a message is sent to Node B and Node C having CAN ID = 222, MSG1 = “O”, this message is send over the CAN bus. Since Node B is having CAN ID = 222, it accepts the frame and rotates the DC motor in clockwise direction. In the same way it happens for switch 4, but now the message changes to “F” and the DC motor stops.

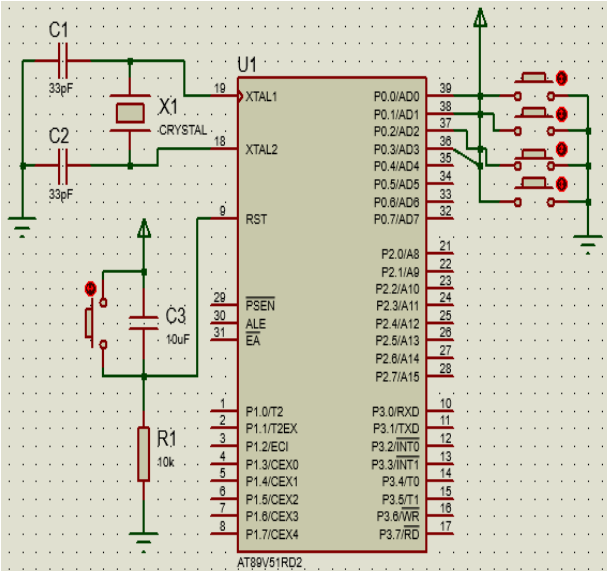


Fig11. Switch 3 is pressed.

V. CONCLUSION

The objective of the project is “Implementation of CAN protocol for industrial automation” was achieved and the protocol has been developed. Using this protocol any industry can establish communication amongst its various modules without depending on any particular controller. SPI protocol, which is required for configuring the CAN controller registers, has been developed instead of using the SPI interface so that the protocol is even independent of SPI interface. The protocol will also help the education sector where more focus is on CISC architecture like controllers which are based on architecture of 8051, hence in this project, 89V51RD2 is used which is having flash memory and additionally it’s an ISP chip.

VI. REFERENCE

[1] Presi.T.P “Design and Development of PIC Microcontroller Based Vehicle Monitoring System Using Controller Area Network (CAN) Protocol” Information Communication and Embedded Systems (ICICES), 2013 International Conference on 21-22 Feb. 2013, pg. 1070 – 1076, Chennai.

[2] CAN specification version 2.0. Robert Bosch GmbH, Stuttgart, Germany, 1991.

[3] Steve Corrigan, “Introduction to the Controller Area Network”, Published by Texas Instruments Application Report, SLOA101A, August 2002–Revised July 2008

[4] MCP 2551 High speed CAN Transceiver Datasheet.

[5] MCP 2515 Stand-Alone CAN Controller with SPI™ Interface datasheet

[6] Axiomatic Global electronic solutions, Finland, July 2006

[7] AN713 Controller Area Network (CAN) Basics by, Keith Pazul, Microchip Technology Inc.

[8] Marco Di Natale “Understanding and using the Controller Area Network”, October 30, 2008

[9] Steve Corrigan “Introduction to the Controller Area Network (CAN)”, Application Report, SLOA101A–August 2002–Revised July 2008

[10] Vikash Kumar Singh, Kumari Archana “Implementation Of 'CAN' Protocol In Automobiles Using Advance Embedded System”Lords Institute of Engineering and Technology, Himayat Sagar, Hyderabad, INDIA, International Journal of Engineering Trends and Technology (IJETT) – Volume 4 Issue 10- Oct 2013

[11] ZoranRistic*Easy8051B* development systems and *CAN-SPI* modules article, MikroElektronika - Software Department.