PICDEM™ LIN User's Guide

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.

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

Microchip received ISO/TS-16949:2002 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona and Mountain View, California in October 2003. The Company's quality system processes and procedures are for its PICmicro® 8-bit MCUs, 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.

Table of Contents

PICDEM™ LIN User's Guide

Chapter 1. PICDEM™ LIN Introduction

1.1 WELCOME

Thank you for purchasing the PICDEM LIN demonstration board from Microchip Technology Incorporated. The PICDEM LIN is a simple board, which demonstrates the capabilities of several Microchip microcontrollers, specifically from the PIC16C432, PIC16C433, PIC16C7XX, PIC16X8X, PIC18FXX8 families and the stand-alone LIN transceiver with built-in voltage regulator MCP201, using the LIN bus protocol.

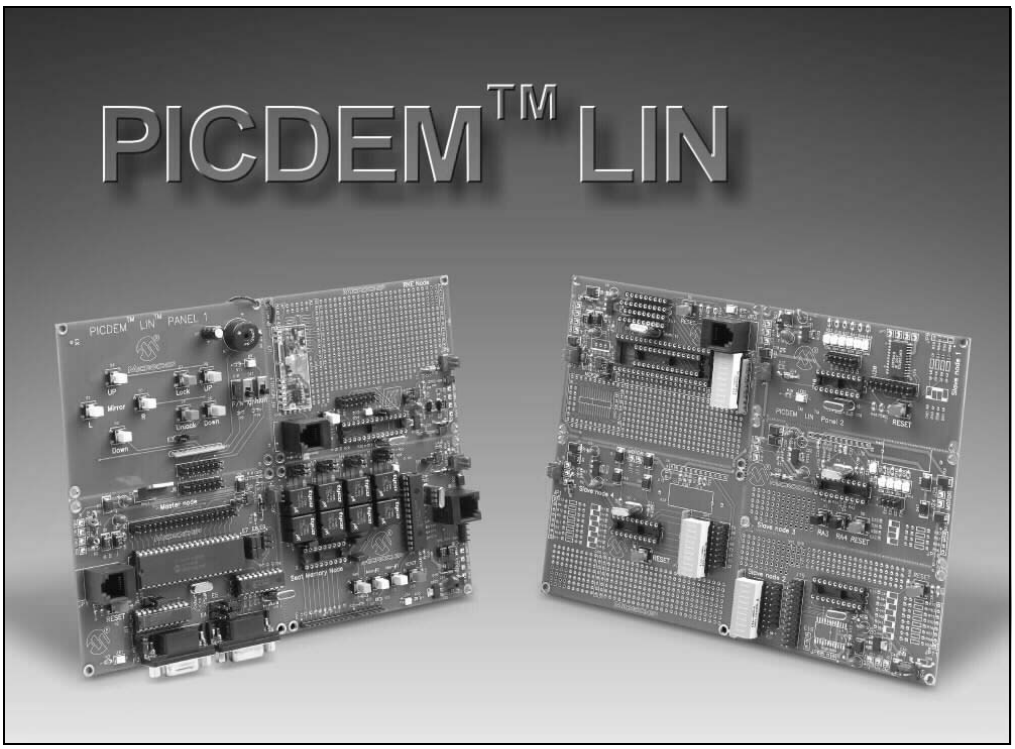
The PICDEM LIN can be used stand-alone with a programmed part, or with an emulator system, such as MPLAB® ICE. Sample programs are provided to demonstrate the unique features of the supported devices.

The PICDEM LIN Kit comes with the following:

1. PICDEM LIN Demonstration Board
2. Sample devices
3. Sample programs
4. PICDEM LIN Demonstration Board User's Guide (this document)

Note:	Please contact your nearest Microchip sales office for any missing parts from the kit.

FIGURE 1-1: PICDEM LIN KIT



1.2 PICDEM LIN DEMONSTRATION BOARD

The PICDEM LIN demonstration board has the following hardware features:

1. 18-pin, 28-pin and 40-pin DIP sockets (although 3 sockets are provided, only one device may be used at a time).
2. On-board +5V regulator for direct input from 12V.
3. RS-232C socket and associated hardware for direct connection to RS-232C interface.
4. CAN bus interface.
5. Control panel interface for LIN bus master.
6. RF stage for keyless entry function.
7. Seat memory unit.
8. Motor control slave node.
9. Jumper to disconnect on-board RC oscillator (approximately 2 MHz).
10. Prototype area for user hardware.

1.3 SAMPLE DEVICES

Several UV erasable and Flash devices are included. The device types may change. These devices are programmed with firmware to provide LIN bus communication. The supplied devices are typically one of the following:

- PIC16F874
- PIC16C432
- PIC16C433

1.4 SAMPLE PROGRAMS

The PICDEM LIN Kit includes a CD-ROM with sample demonstration programs on them. These programs may be used with the included sample devices or with an emulator system. The programs are:

- `demo432.asm` – PIC16C432 LIN bus Demo
- `demo433.asm` – PIC16C433 LIN bus Demo
- `demo874.asm` – PIC16F874 LIN bus Demo

1.5 PICDEM LIN USER'S GUIDE

This document describes the PICDEM LIN demonstration board, tutorial and demonstration software, to give the user a brief overview of the PICmicro® MCUs supported. Detailed information on individual microcontrollers may be found in the device's respective data sheet.

Chapter 1: Introduction – This chapter introduces the PICDEM LIN and provides a brief description of the hardware.

Chapter 2: Getting Started – This chapter goes through a basic step-by-step process for getting your PICDEM LIN up and running as a stand-alone board.

Chapter 3: Tutorials – This chapter provides a detailed description of the tutorial programs.

Appendix A: Hardware Detail - This appendix describes in detail the hardware of the PICDEM LIN board.

1.6 REFERENCE DOCUMENTS

Reference Documents may be obtained by contacting your nearest Microchip sales office, or by downloading via the Microchip web site www.microchip.com.

- Individual data sheets
- *MPLAB® IDE, Simulator and Editor User's Guide* (DS51025)
- *MPASM™ User's Guide with MPLINK™ and MPLIB™* (DS33014)
- *PRO MATE® II User's Guide* (DS30082)
- *MPLAB® ICE Emulator User's Guide* (DS51159)
- *PICmicro® Mid-Range Reference Manual* (DS33023)
- *LIN bus Specification Ver. 1.2* (can be obtained from www.lin-subbus.de)

NOTES:

Chapter 2. Getting Started

The PICDEM LIN may be used as a stand-alone board or with an emulator. The emulator discussed in this chapter is the PICMASTER® emulator. However, other emulators may be used. For a list of PICmicro® MCU compatible emulators, please refer to the *Microchip Development Systems Ordering Guide,* (DS30177).

2.1 PICDEM LIN AS A STAND-ALONE BOARD

The PICDEM LIN may be demonstrated immediately by following the steps listed below:

- Make sure the preprogrammed sample devices are in the appropriate socket on the PICDEM LIN board.
- Apply power to the PICDEM LIN (see Figure 2-1).
- Connect the two panels with a flat ribbon cable (see Figure 2-2).
- Press buttons S8 or S10 on the console panel to see ports on the slave nodes increment or decrement.

These steps are illustrated in Figure 2-1.

FIGURE 2-1: CONNECTING THE LIN BUS DEMO BOARD

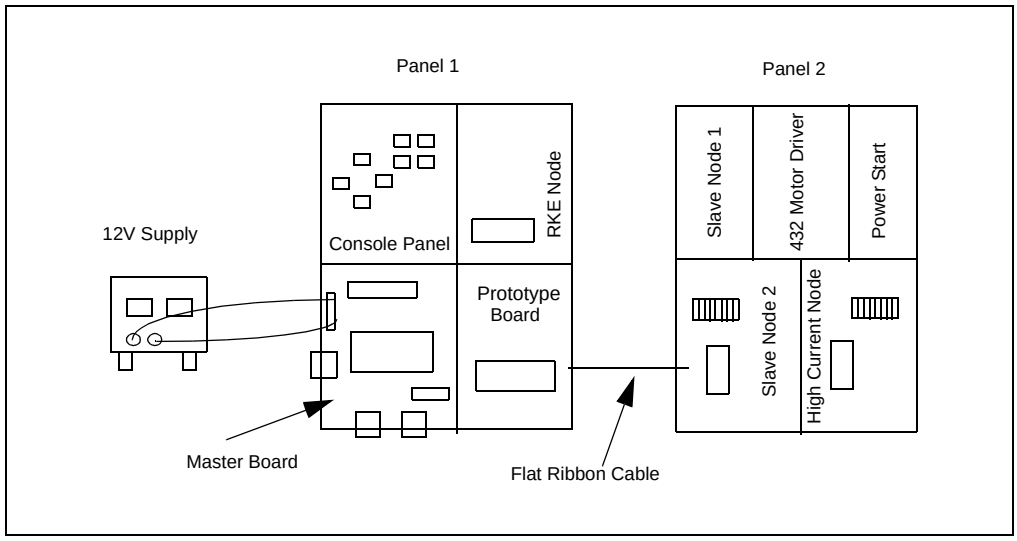
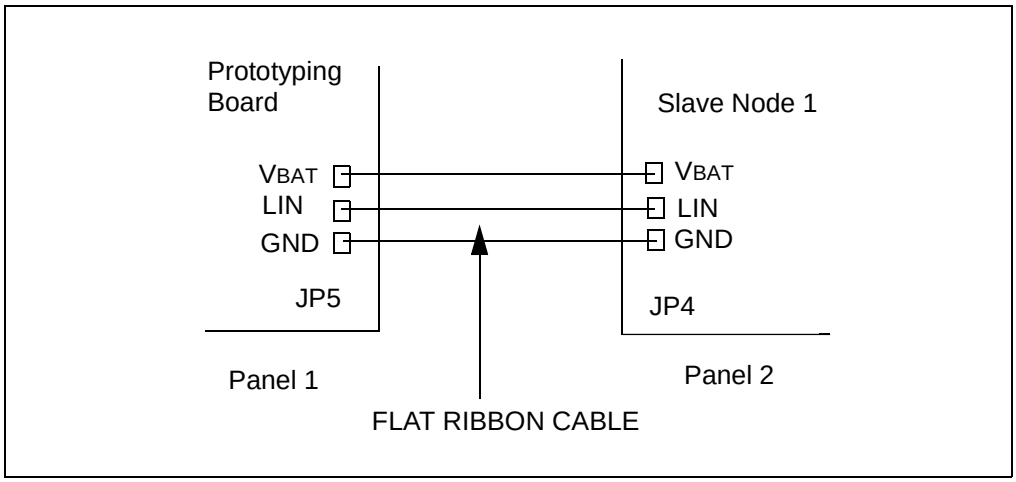


FIGURE 2-2: FLAT RIBBON CABLE CONNECTION



2.2 PICDEM LIN AS A STAND-ALONE BOARD – SAMPLE PROGRAMS

To demonstrate PICDEM LIN operation with one of the sample programs, the sample device will have to be erased and reprogrammed. Once the device has been reprogrammed:

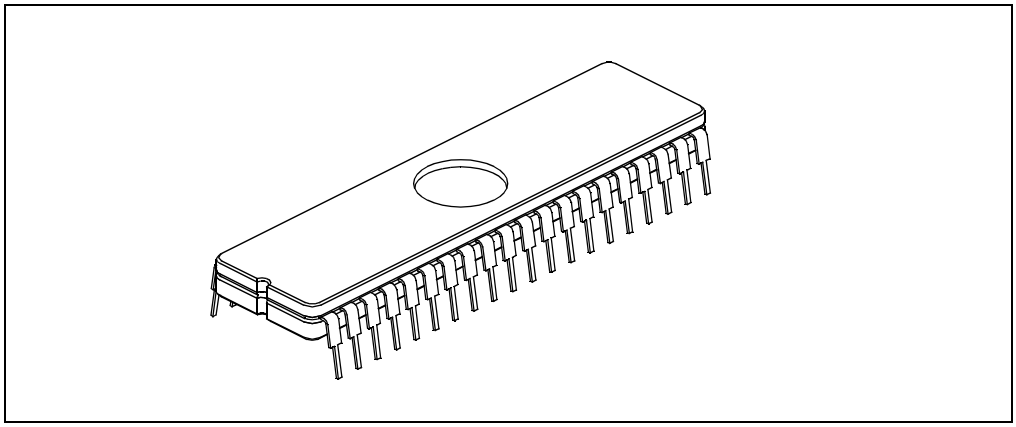
- Make sure the sample device is in the appropriate socket on the PICDEM LIN board.
- Apply power to the PICDEM LIN.
- Consult the appropriate chapter in this document for information on the execution of each demo.

2.2.1 Erasing the Sample Device

To erase an EPROM device:

- Remove any labels covering the device window. If you do not have a windowed device (Figure 2-3), you cannot reprogram it. A windowed version of all EPROM devices may be ordered by requesting the JW package.
- Place the device in an Ultraviolet (UV) EPROM Eraser. The amount of time required to completely erase a UV erasable device depends on: the wavelength of the light, its intensity, distance from UV source, and the process technology of the device (how small are the memory cells).
- Verify that the device is blank (e.g., perform a blank check) before attempting to program it.

FIGURE 2-3: WINDOWED DEVICE



To erase an EEPROM/Flash device:

- Enable your programmer in MPLAB IDE. If you change programmers, you will have to restart MPLAB IDE.
- Place the device in the programmer.
- Select *Erase Program Memory* or the equivalent erase command from the Programmer menu.

Note:	You do not have to erase an EEPROM/Flash device before reprogramming it.

2.2.2 Reprogramming the Sample Device

To reprogram the erased sample device, the following will be necessary:

1. Sample programs installed on the hard drive.

   The PICDEM LIN package includes a 3.5-inch disk, which contains sample programs for all the processor types supported. Instructions on how to install the programs can be found in the "README" file also on the disk.

2. An assembler, such as MPASM™ Assembler available with MPLAB IDE.

   Sample programs may be used to program the sample device once they have been assembled. Microchip Technology's MPLAB Integrated Development Environment (IDE) includes an assembler (MPASM). However, other assemblers may be used.

3. A device programmer, such as PRO MATE® II.
4. Once the sample program is in hex file format, a programmer may be used to program a blank device. Microchip Technology's PRO MATE II or PICSTART® Plus programmers may be used. Both are compatible with MPLAB IDE. However, other programmers may be used.

If the code protection bit(s) have not been programmed, the on-chip program memory can be read out for verification purposes.

Note:	Microchip does not recommend code protecting windowed devices.

2.3 HARDWARE DESCRIPTION

2.3.1 Circuits in Common

Much of the circuitry on each of the subsystem boards is the same. These building blocks will be referred to later in each of the node descriptions without further detail.

2.3.1.1 POWER SUPPLY

The power supplies on each board are built around an automotive grade voltage regulator. The voltage regulators are protected by a reverse polarity blocking diode and a 45V Zener diode, filtered on both the input and output. An LED is connected to the output for 'power-on' indication.

2.3.1.2 LIN BUS TRANSCEIVER

The Local Interface Network transceiver is socketed to accept an industry standard LIN or K/L-Line transceiver. The Microchip Technology MCP201 can be accommodated by removing the jumper connecting pin 3, V_{DD}, to the board V_{CC} power plane.

2.3.1.3 ICD/ICSP™ INTERFACE

The LIN bus master node, as well as the seat memory control node, are equipped with an ICSP interface connector. The ICSP interface allows programming the PICmicro devices in-circuit, without removing them from the board. When the ICSP interface is used, RB3, RB6 and RB7 are no longer available to the user on 28- and 40-pin devices.

2.3.2 The LIN bus Master Node

The LIN bus master node is designed to perform several tasks:

- Stand-alone LIN bus Master
- LIN-to-CAN Gateway
- RS-232-to-LIN Gateway

The CPU may be any of several PICmicro devices, in either 28- or 40-pin dual in-line packages. As a master node, the buttons and output devices on the console panel can be scanned and controlled, and LIN messages transmitted over the network. The schematic of the LIN bus master node is shown in Figure A-2.

The master node also has both a CAN controller and transceiver, as well as RS-232 buffers connected to the internal USART. These facilities allow the node to function as a CAN- or RS-232-to-LIN gateway.

2.3.2.1 PROCESSOR SOCKETS

Two sockets are provided, only one device may be used at a time.

- 28-pin socket
- 40-pin socket

2.3.2.2 RS-232 SERIAL PORT

An RS-232 level shifting IC has been provided with all the necessary hardware to support connection of a RS-232 host through the DB9 connector. The port is configured as DCE, and can be connected to a PC using a straight through cable.

28/40-pin devices have their RX and TX pins tied to the RX and TX lines of the MAX RS-232 via the Jumper E7.

The jumper E7 routes the RX and TX lines of the USART to either the LIN bus transceiver chip or the MAX RS-232 chip. Figure 2-4 shows the configuration if the USART is connected to the LIN bus transceiver. In this case, no pins will be connected to the MAX RS-232 chip.

Figure 2-5 shows the configuration where the USART is connected to the LIN bus transceiver chip, RB5 to the Transmit line, and RB4 to the Receive line of the RS-232 chip.

FIGURE 2-4: JUMPER E7 CONNECTS USART TO LIN BUS TRANSCEIVER

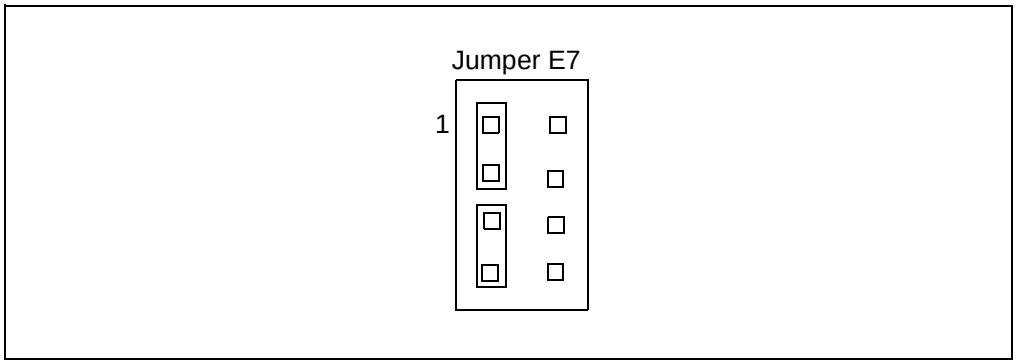


FIGURE 2-5: USART CONNECTED TO LIN BUS TRANSCEIVER, RB4 AND RB5 TO MAX RS-232

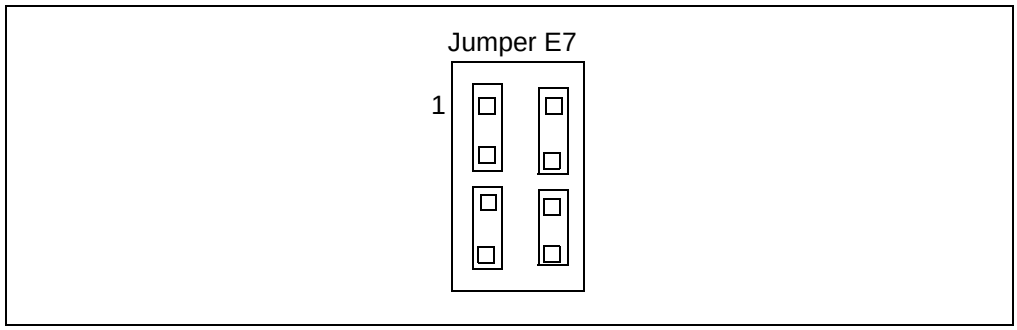


FIGURE 2-6: JUMPER E7 CONNECTS USART TO MAX RS-232

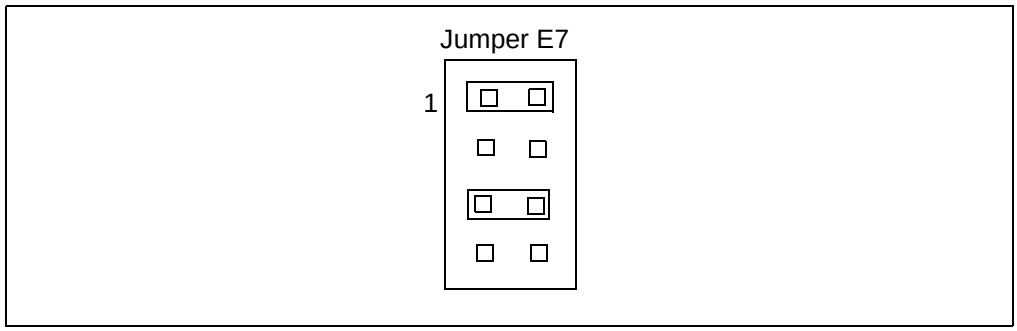
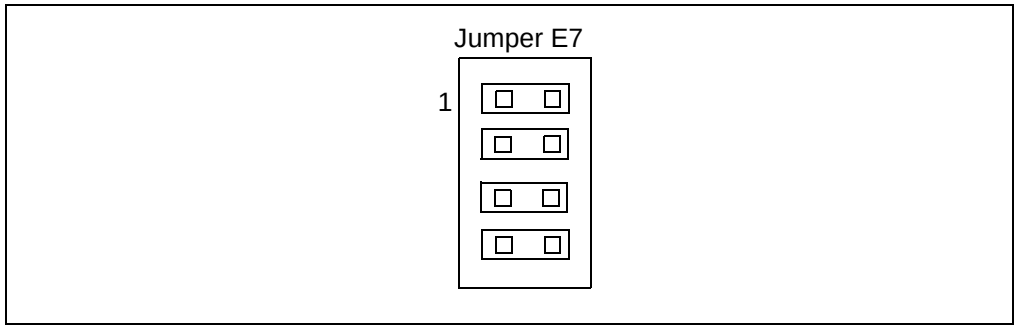


FIGURE 2-7: USART CONNECTED TO MAX RS-232 AND RB4 AND RB5 TO LIN BUS TRANSCEIVER



2.3.2.3 SWITCHES

One switch provides the following function:

- S2 – $\overline{\text{MCLR}}$ to hard reset the processor

Switch S2 has a debouncing capacitor. When pressed, the switch is grounded. When IDLE, it is pulled high (+5V).

2.3.2.4 OSCILLATOR OPTIONS

The LIN bus master runs off a 20 MHz crystal, but the oscillator can be changed from crystal to a resonator or RC oscillator. This is done by unsoldering the crystal and C20, and installing a 10K resistor on R36. This will allow the PICmicro MCU to run off an RC oscillator.

2.3.2.5 CAN BUS CONNECTIONS

There are two ways to connect to the CAN bus. One way is to use the MCP2510 stand-alone CAN controller, and the other way is to use a PICmicro MCU with an internal CAN interface (e.g., PIC18F258 or PIC18F458).

Jumper E4 and E5 will select whether the CAN transceiver chip will be connected to the MCP2510 stand-alone CAN controller, or pins RB2 and RB3 of the 40/28-pin socket.

RB3 of the 40/28-pin socket.

2.3.2.6 CONSOLE PANEL

The console panel board is connected to PORTD and PORTE of the 40-pin socket. The purpose of the control panel is to provide input and output to the master board, which could, for example, be transformed via LIN bus firmware to a LIN bus message.

If the console board is separated from the master board, then the console board can be connected with a flat ribbon cable via Jumper J6.

2.3.3 The Console Panel

The console panel provides input and output to the LIN bus Master. The schematic of the console panel is shown in Figure A-1.

The inputs from the console panel can be used to generate the LIN bus message. The LED, audio transducer and switches on the console panel interface to PORTD and PORTE of the master board. If the boards are separated, the console panel can be connected via a flat cable from J7 to the master board.

2.3.3.1 CONSOLE INPUTS

There are ten switches and one Jumper input. All the push button switches are pulled to VDD and closed to Ground. The IGNITION switch output is VDD when closed, and open circuit in the OFF position. The PARK/NEUTRAL switch is pulled up to VDD when OFF and closes to Ground. The jumper E3 is meant to simulate the diagnostic port jumper in a typical automotive diagnostic link. This jumper could, for example, cause the master to send data and error codes to a diagnostic unit.

2.3.3.2 CONSOLE OUTPUTS

The malfunction indicator lamp (MIL) is connected to VDD and turned on by taking the input to ground. The Chime audio beeper has a self-contained sound generator and only needs to be grounded to activate.

2.3.4 The Remote Keyless Entry Panel

The purpose of the Remote Keyless Entry Panel is to provide an interface from a key fob to the LIN bus. The schematic of the remote keyless entry node is shown in Figure A-4. The node can be used as either LIN bus master or LIN bus slave.The decision of which configuration is chosen depends on the firmware design.

The Remote Keyless Entry panel is equipped with a 28-pin socket for a PICmicro MCU, an RF board, a LEARN button and LED, and a LIN bus transceiver.

In order to program the PICmicro MCU to identify encoder transmitters, a LEARN switch input is routed to RB5 and a LED is routed to RB2. A high voltage tolerant input is routed to RB1.

2.3.4.1 RF RECEIVER

The RF receiver is a standard Telecontrolli module, as supplied in the KEELOQ® development kits. The module installed is for 433.9 MHz.

2.3.5 General Prototype Board

This circuit is designed for general purpose use. The schematic of the board is shown in Figure A-3. The general prototype area is assembled with four sockets. A 28-pin and 40-pin socket are for dual in-line packages. In the prototype area itself, are two areas where a 16-pin SOIC package can be installed. This area could be used, for example, for drivers.

Pins of RC6 and RC7 are routed to the single chip LIN bus transceiver.

2.3.6 Power Seat Panel

This circuit can be used to demonstrate a typical seat position memory module. With proper firmware, it can be commanded from either the on-board button inputs or via the LIN bus. The schematic of this panel is given in Figure A-5.

2.3.6.1 BUTTON AND SENSOR INPUTS

Three button inputs are connected to RA4, RA5 and RC3. These are labeled MEM#1, MEM#2 and SET, respectively. J12 is provided to enable external switch input connections. J12 also includes a signal to drive an external LED “Memory” function indicator.

Headers J9, J11, J15 and J16 connect to the external (user supplied) motors and have connections for analog feedback of motor position. These four analog inputs are routed through protective RC networks to RA0:3. These analog voltages may not exceed the range of 0V to 5.0V. If the potentiometers are connected ratio metrically between Vcc and ground, R17, R19, R27 and R31 may be omitted.

Header J8 is for external, manual motor control push button switches. These switches are connected directly to the relay driver circuit and operate the position motors independently from the microcontroller. These buttons can also be read through PORTB of the PIC® microcontroller.

A high voltage tolerant input is routed to RC5. This is labeled as IGNITION.

2.3.6.2 RELAY DRIVER

Power relays for the seat motors are provided on-board. These are driven by either a high- or low-side, octal 500 mA driver. Configuration of the board to accept a particular driver is done by jumpers J10, J13 and J14. Refer to the schematic to determine the proper driver device.

Jumper	Position	Description
J10	A-B	Relay common to V_{BAT}, U8 is a low-side driver
	B-C	Relay common to GND, U8 is a high-side driver
J13	A-B	Pin 9, U8 = V_{BAT}
	B-C	Pin 9, U8 = GND
J14	A-B	Pin 10, U8 = GND
	B-C	Pin 10, U8 = V_{BAT}

U13 and U14 provide high-voltage, open-collector isolation to the PIC microcontroller. The drivers are controlled by PORTB of the PICmicro MCU.

2.3.6.3 ICD/ICSP INTERFACE

The seat memory control node is equipped with an ICSP interface connector. The ICSP interface allows programming of the PICmicro MCU in-circuit without removing it from the board. When the ICSP interface is used for In-Circuit-Debug (ICD), RB3, RB6 and RB7 are no longer available to the user on 28- and 40-pin devices. Motor control headers J9 and J15 should not be used.

2.3.7 LIN bus Slave Node H-Bridge Drivers

2.3.7.1 SLAVE NODE 1

Slave node 1 is designed for the PIC16C433 and Allegro A3976KLB dual H-bridge driver. The I/O pins of the PIC16C433 can be connected to the driver device or any of the input circuits, by wire-wrapping the appropriate connections on the Jumper Field J20. Alternatively, the micro pins can be individually connected to an LED for indication by E18, E20-24. Two analog inputs are routed from Connector JP9 to the Jumper Field J20.

The schematic of this board is displayed in Figure A-6.

2.3.7.2 SLAVE NODE 3 HIGH VOLTAGE, HIGH CURRENT SLAVE NODE

This node is designed to demonstrate the PIC16C432 with a high voltage, high current motor driver. A high voltage and high current driver, such as a A3952SW, can be used.

The motor driver is controlled by pins RB4 through RB7. The output of the driver is routed to J24 and J25. RB0 to RB3 can be connected to LEDs via the Jumpers E25 to E28.

The schematic of this board is given in Figure A-8.

2.3.7.3 SLAVE NODE 4 HIGH VOLTAGE, HIGH CURRENT SLAVE NODE WITH ANALOG INPUTS

This slave node functions the same as the previous slave node, with the following additions:

- RB4 through RB7 can also be routed to LEDs
- RA0 to RA3 can be used as analog inputs. The analog inputs are routed to the Connector JP1.

The schematic of this board is shown in Figure A-9.

2.3.8 LIN bus Slave Node High-Side Driver

2.3.8.1 SLAVE NODE 2

Slave node 2 is designed for the PIC16C432 and a MC33143DW high-side driver. The I/O pins of the PIC16C432 can be connected to either the high-side driver and/or the LEDs.

In order to connect I/O pins from PORTB to the LEDs, Jumpers E10 to E17 have to be installed. If some of the I/O pins from PORTB are to be connected to the high-side driver, the respective Jumpers E10 to E17 should be removed and a connection has to be established on the Jumper Field J19.

Analog inputs to PORTA pins are provided through Connector JP11.

The schematic of this board is given in Figure A-7.

Chapter 3. Tutorial

3.1 TUTORIALS

The LIN bus master and slave programs are preprogrammed into the sample devices. These programs are listed on the included CD-ROM for user reference. For example, if the sample device has been reprogrammed with another sample program, the tutorial may be reassembled and reprogrammed into the device.

The tutorial program functions as follows.

For detailed information on the LIN bus hardware, please refer to Appendix A.

3.1.1 The Master Software

The source code of the master software is shown in Appendix B. The flow chart of the main routine is shown in Figure 3-1. In this code example, the LIN bus master is realized with the USART of the PIC16F874.

After initialization, the master routine reads in a key from the console panel. After a key is pressed and debounced, the parity bits, according to the LIN bus specification, are generated. The parity bits are inserted automatically into the identifier byte. The identifier byte includes the code for the key, which was pressed, and the parity bits. After the parity bits are generated, the checksum is computed. The checksum is appended after the last data byte. After the checksum computation, all data is transmitted via the LIN bus. Once the transmission is done, the routine waits until another key is pressed.

Following, the single subroutines are described.

Note:	The LIN firmware is only LIN 1.2 compliant.

3.1.1.1 THE KEYBOARD READ ROUTINE

This routine reads in one key from the console panel and debounces it. Depending on the key, an 8-bit value is returned to the main routine. This 8-bit value is stored in the ID register (ID_TEMP).

3.1.1.2 PARITY BIT GENERATION

This routine computes the parity bits for the identifier byte. The identifier byte, which contains only the code for the key, is used for this calculation. The calculation is done according to the LIN bus specification, Ver. 1.2. The calculate parity bits are stored in bit 6 and bit 7 of the identifier byte.

3.1.1.3 CHECKSUM CALCULATION

The checksum in this routine is calculated according to LIN bus specification, Ver. 1.2. The identifier byte contains a field that is used to determine the amount of data bytes calculated into the checksum. The number of data bytes to transmit is coded in the identifier byte. The FSR register points to the master routine in the data byte field. Once the checksum routine is called, the FSR register points to the first data byte to be transmitted. The data byte array is initialized during start-up. After the checksum is calculated, it is checked to see whether the CRC has to be appended to the data bytes (Master Transmission mode), or if the CRC has to be checked (Master Reception mode). The information, whether the CRC has to be checked or appended to the data bytes, is given by the identifier byte (ID_TEMP).

3.1.1.4 THE LIN BUS TRANSMISSION ROUTINE

In this routine, the ID data bytes and the CRC bytes are transmitted via the USART to the LIN bus.

First, the Synchronization byte is transmitted. This signal is 13 bits wide. Since the USART can only transmit 8 bits at a time, the baud rate is slowed to achieve a 13-bit signal at the nominal bit rate. The nominal bit rate in this example program is 19.2 Kbaud.

After the synchronization byte is transmitted to the USART, switched back to the nominal baud rate, and the synchronization field is transmitted, the synchronization field is followed by the identifier byte. After the identifier byte is transmitted, it is checked to see whether data has to be received from a slave node or data has to be transmitted to a slave node.

In this example code, the Master is a transmitter only.

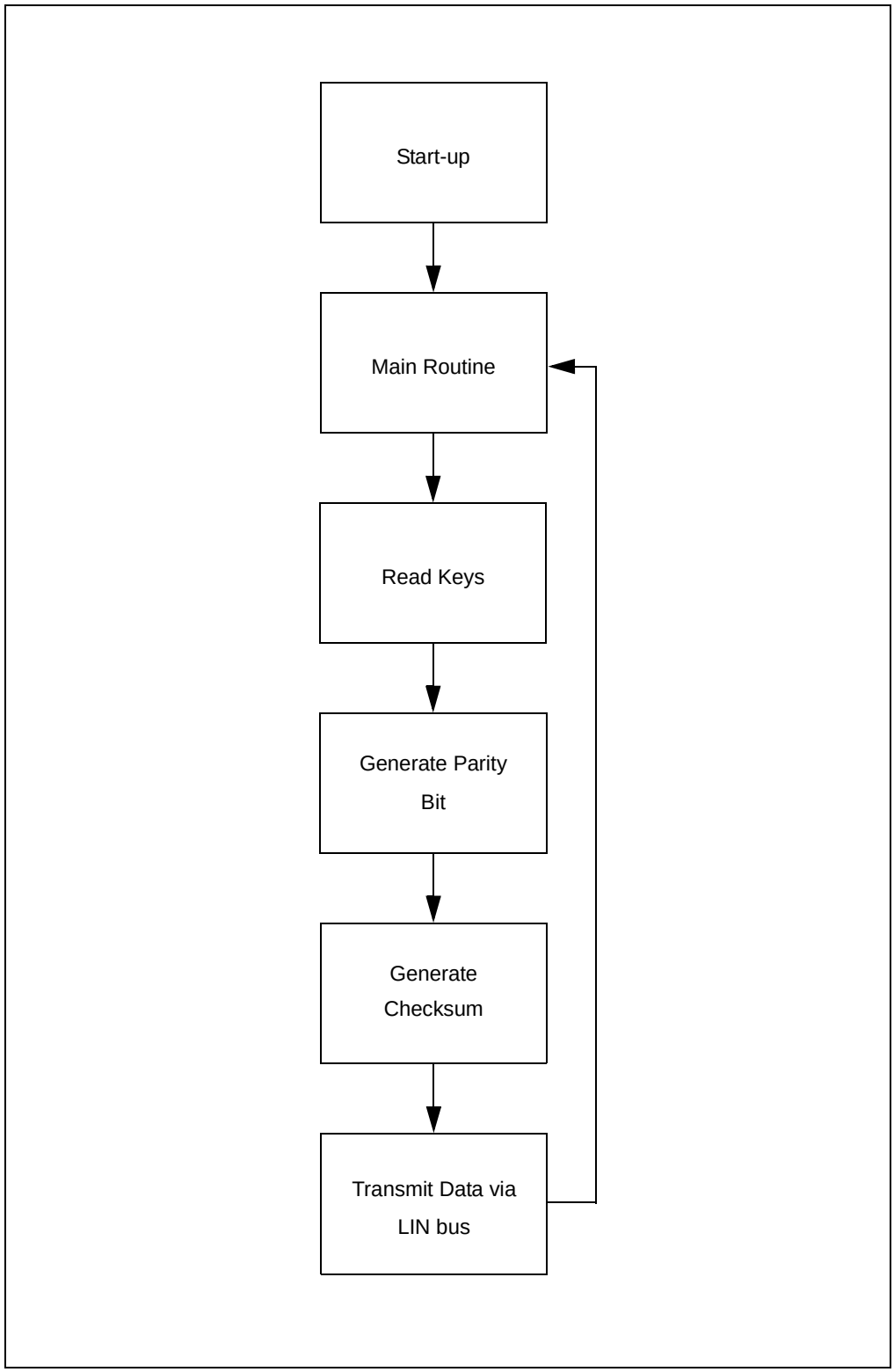
In Transmit mode, the master will transmit his data and CRC value. After everything is transmitted, the routine returns to the main routine.

In Receive mode, all data, including CRC, is received. The CRC value is checked and the routine returns to the main routine.

3.1.1.5 ERROR HANDLING

In the example, no error handling is implemented. The LIN bus specification gives the user the flexibility to handle errors.

FIGURE 3-1: MAIN ROUTINE OF LIN BUS MASTER



3.1.2 The PIC16C432 LIN bus Slave Code

The source code of the PIC16C432 LIN bus slave software is shown in Appendix C. The flowchart of the main routine is shown in Figure 3-2. In this code example, the LIN bus slave is realized entirely in software without hardware support.

After Reset, the PIC16C432 initializes its registers. After the initialization phase, the MCU waits for the synchronization break signal from the master. Once the signal is detected, the slave branches into the LIN handler subroutine. After receiving the transmitting data, the main routine checks what actions have to be taken upon the received identifier. Two actions are implemented:

- Pushing the WindowDown button on the console panel will decrement the LEDs connected to PORTB
- Pushing the WindowUp button on the console panel will increment the LEDs connected to PORTB

After PORTB is either incremented or decremented, the routine waits for the next synchronization break signal.

3.1.2.1 LIN BUS HANDLER

After the synchronization break signal was received, the PIC16C432 waits for the Start bit of the synchronization byte. Once the Start bit is detected, a software counter is incremented until all bits for the synchronization byte are received. After measuring the time, the baud rate is calculated by dividing the measured time by eight.

Following the calculation of the baud rate, the identifier byte is received. This identifier byte is used to determine whether the slave has to transmit data or receive data. Prior to decoding the identifier byte, the parity bits are checked. Error handling is not implemented, since the error handling is left to the user by the LIN bus specification.

In this example implementation, only the Receive mode is used.

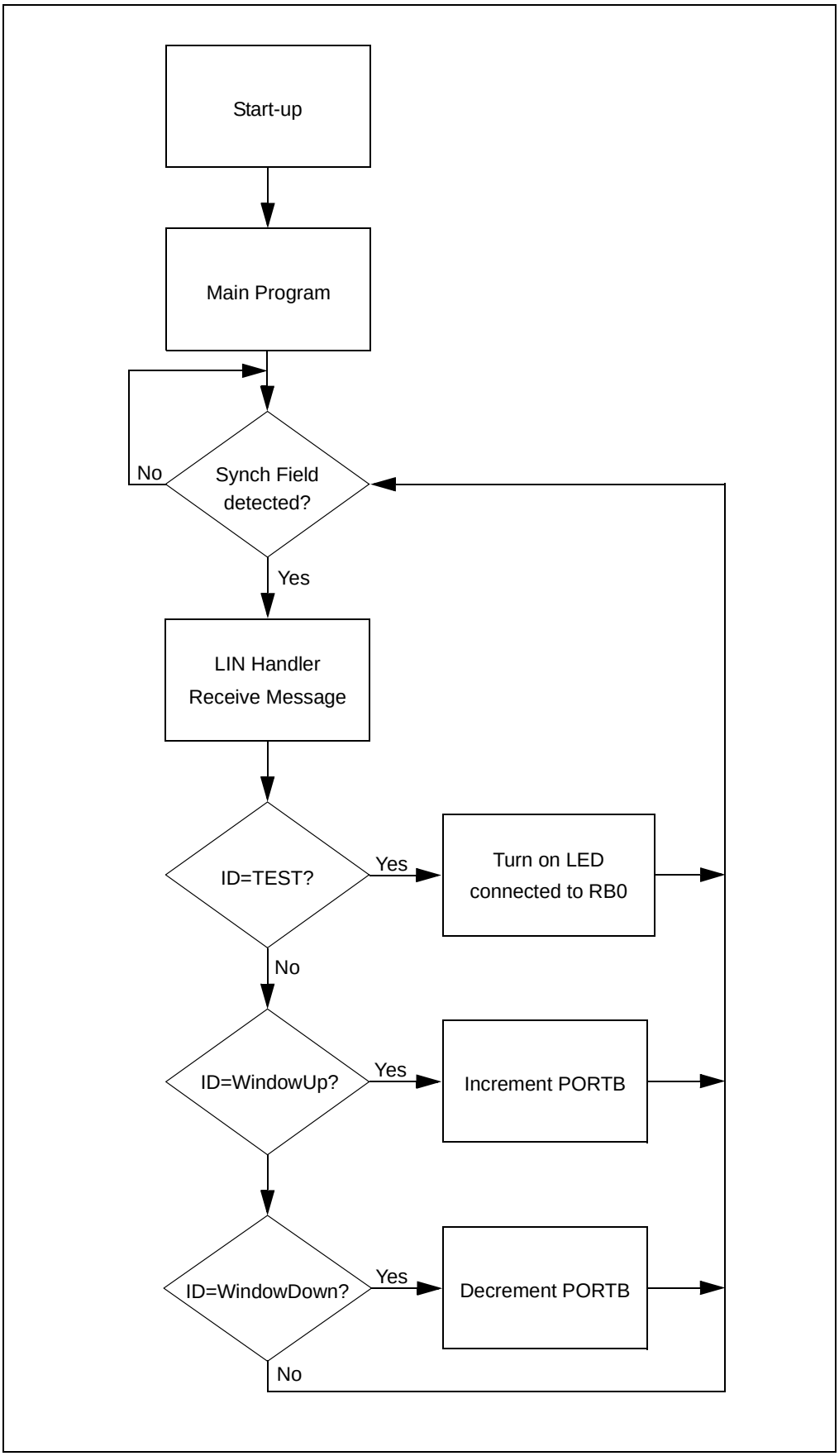
Transmit mode

Prior to transmitting all data, the CRC value for the data to be transmitted is generated. After all data is transmitted, the LIN handler routine returns to the main routine.

Receive mode

After receiving all data received, the CRC value is checked. After the CRC check, the LIN bus handler routine returns to the main routine.

FIGURE 3-2: PROGRAM FLOW OF THE PIC16C432 LIN SLAVE CODE



3.1.3 The PIC16C433 LIN bus Slave Code

The source code of the PIC16C433 LIN bus slave software is shown in Appendix D. The Program flow is almost identical to that described in **3.1.2 "The PIC16C432 LIN bus Slave Code"**, with the following exceptions:

- Pushing the WindowDown button on the console panel will turn on GPIO0
- Pushing the WindowUp button on the console panel will turn off GPIO0

Appendix A. Hardware Detail

A.1 BOARD LAYOUT AND SCHEMATICS

FIGURE A-1: H-BRIDGE DRIVER

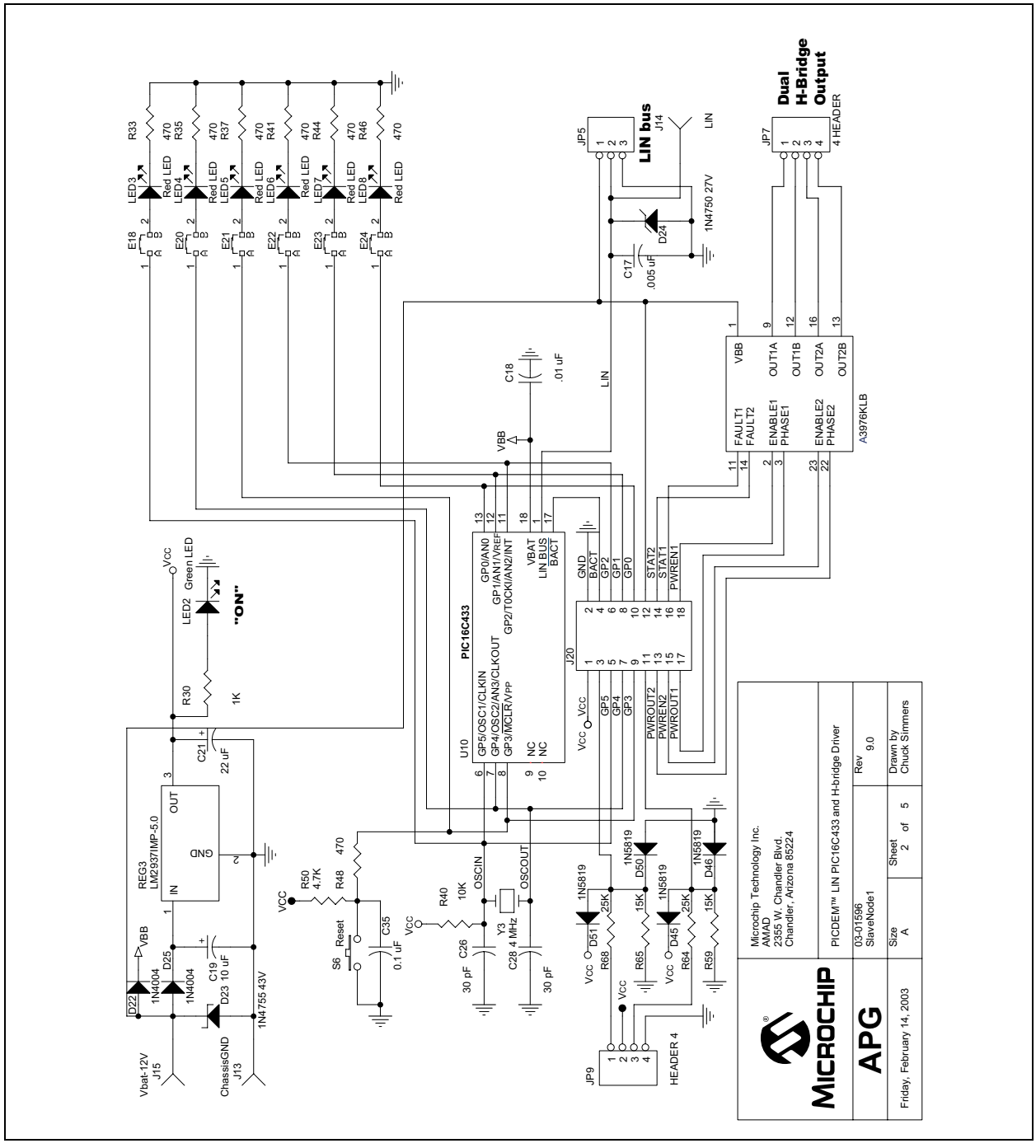


FIGURE A-2: HIGH-SIDE DRIVER

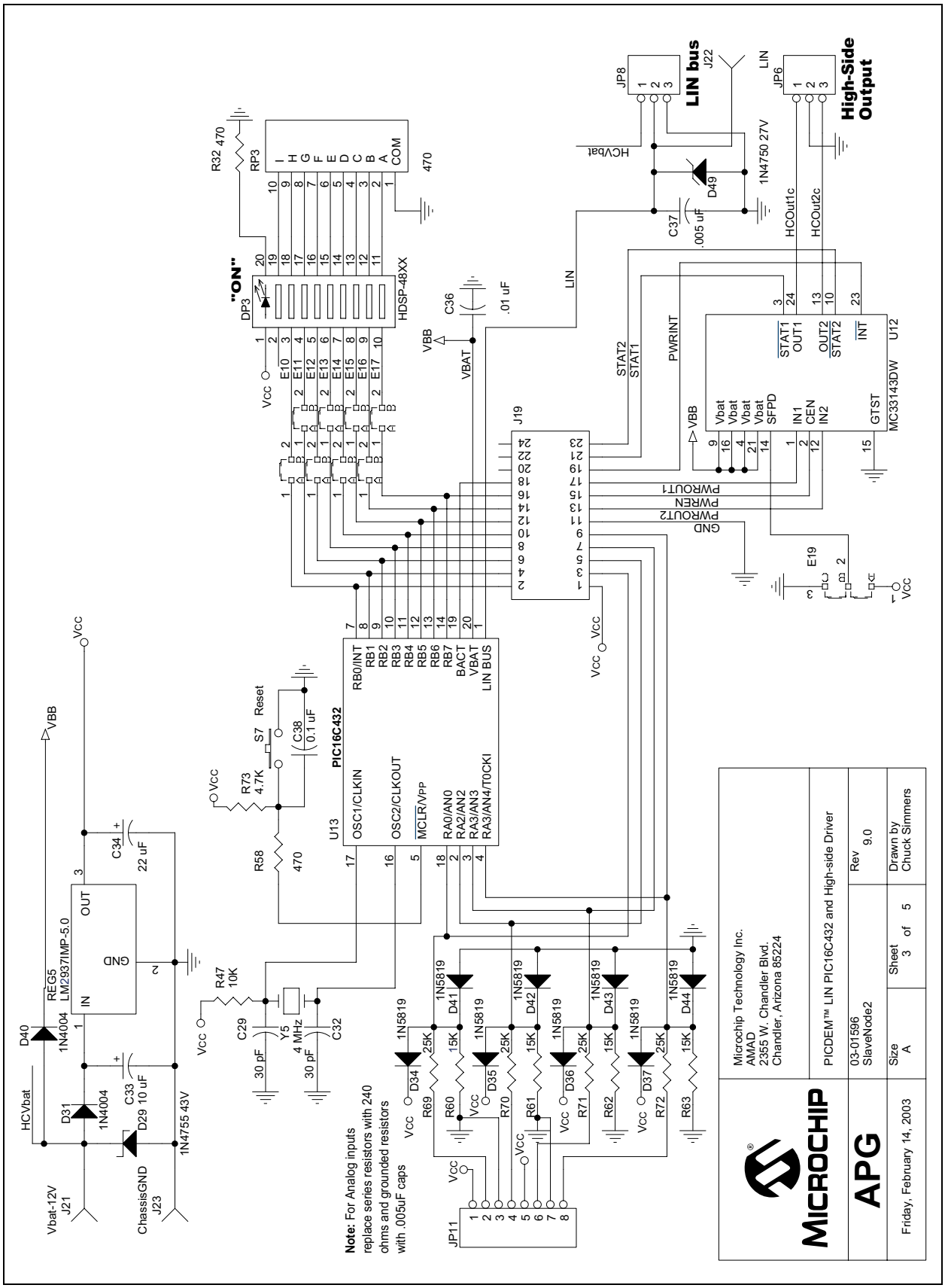


FIGURE A-3: GENERAL PROTOTYPING BOARD

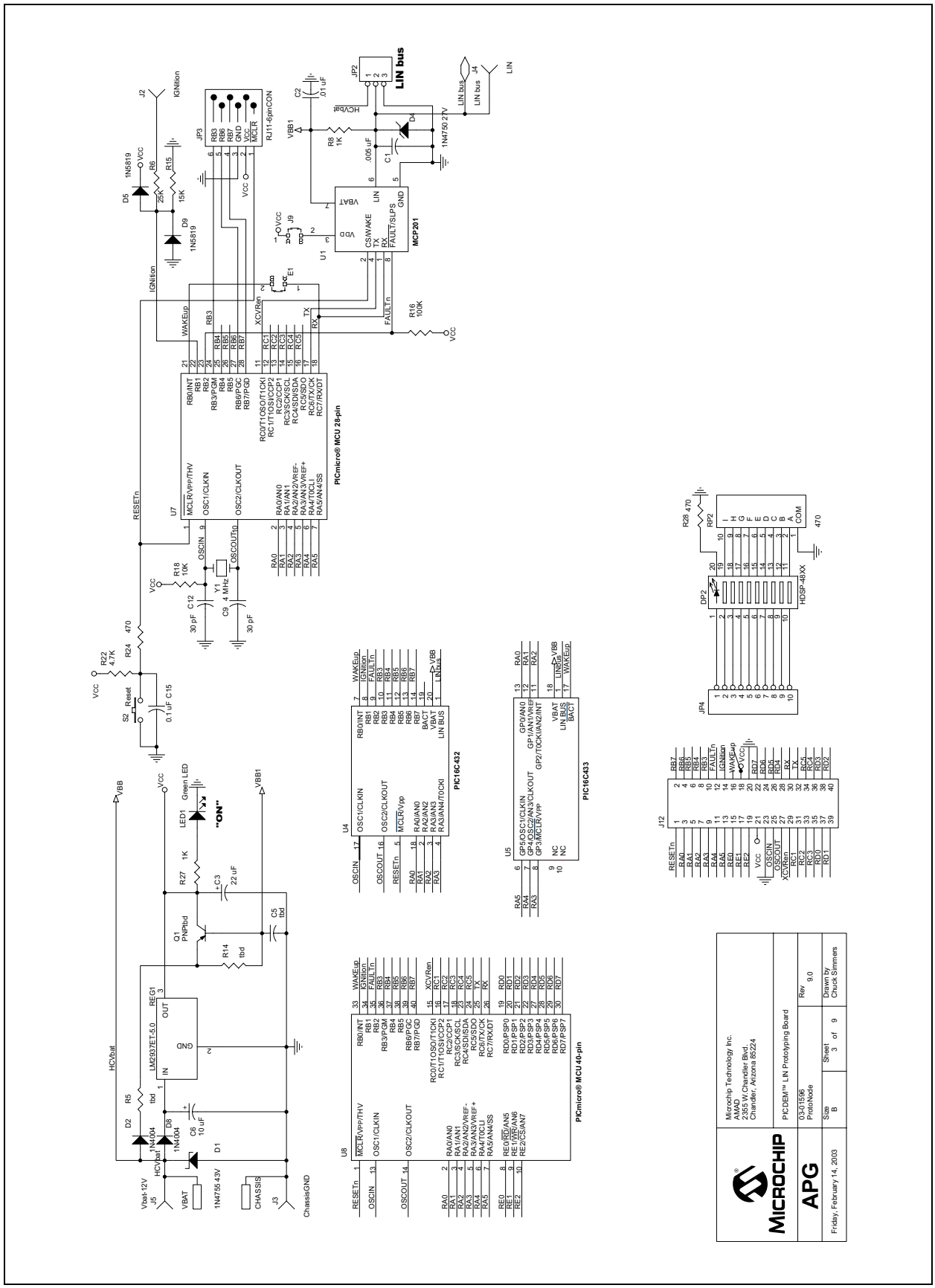


FIGURE A-4: H-BRIDGE DRIVER BOARD 1

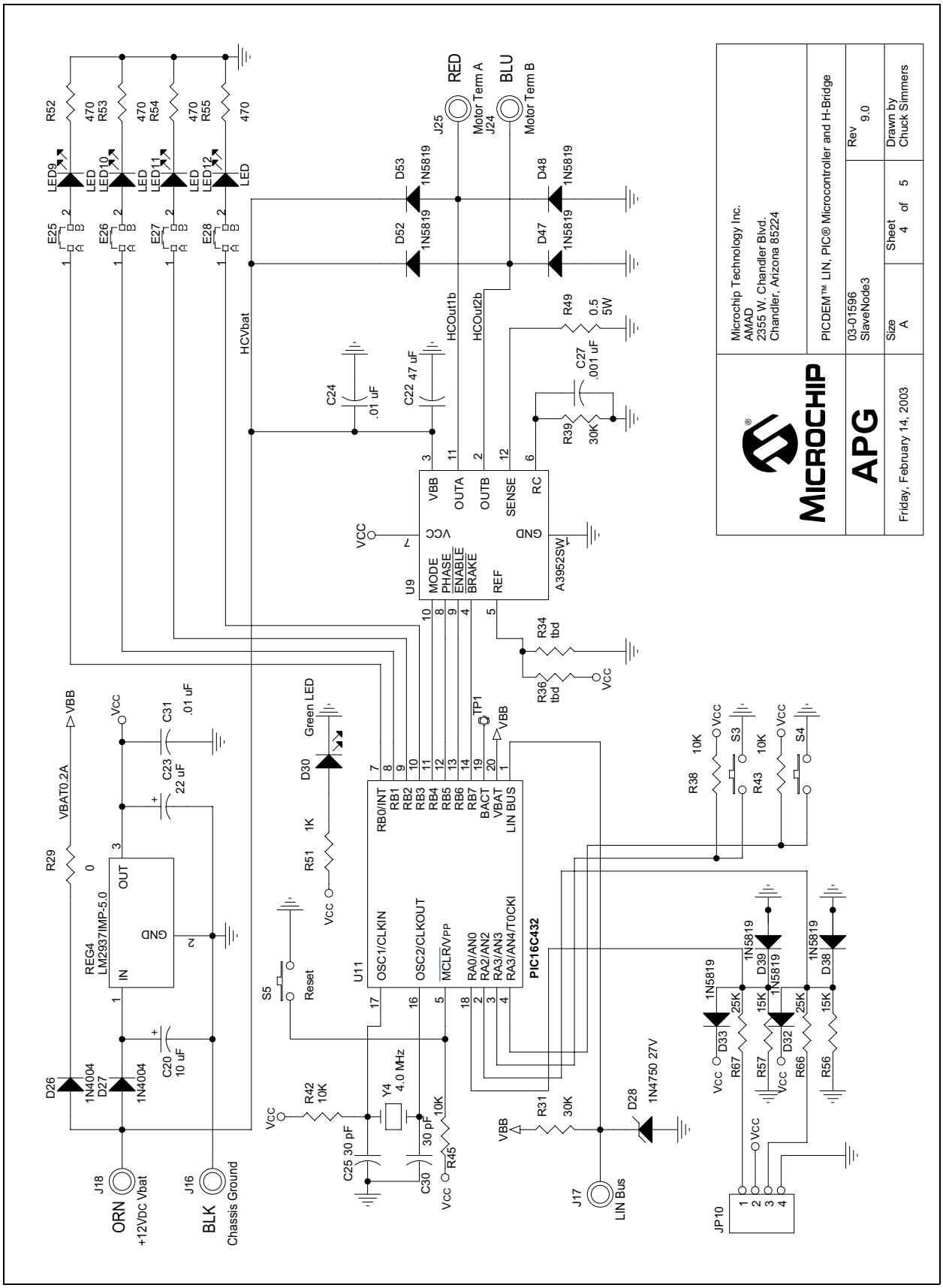


FIGURE A-5: H-BRIDGE DRIVER BOARD 2

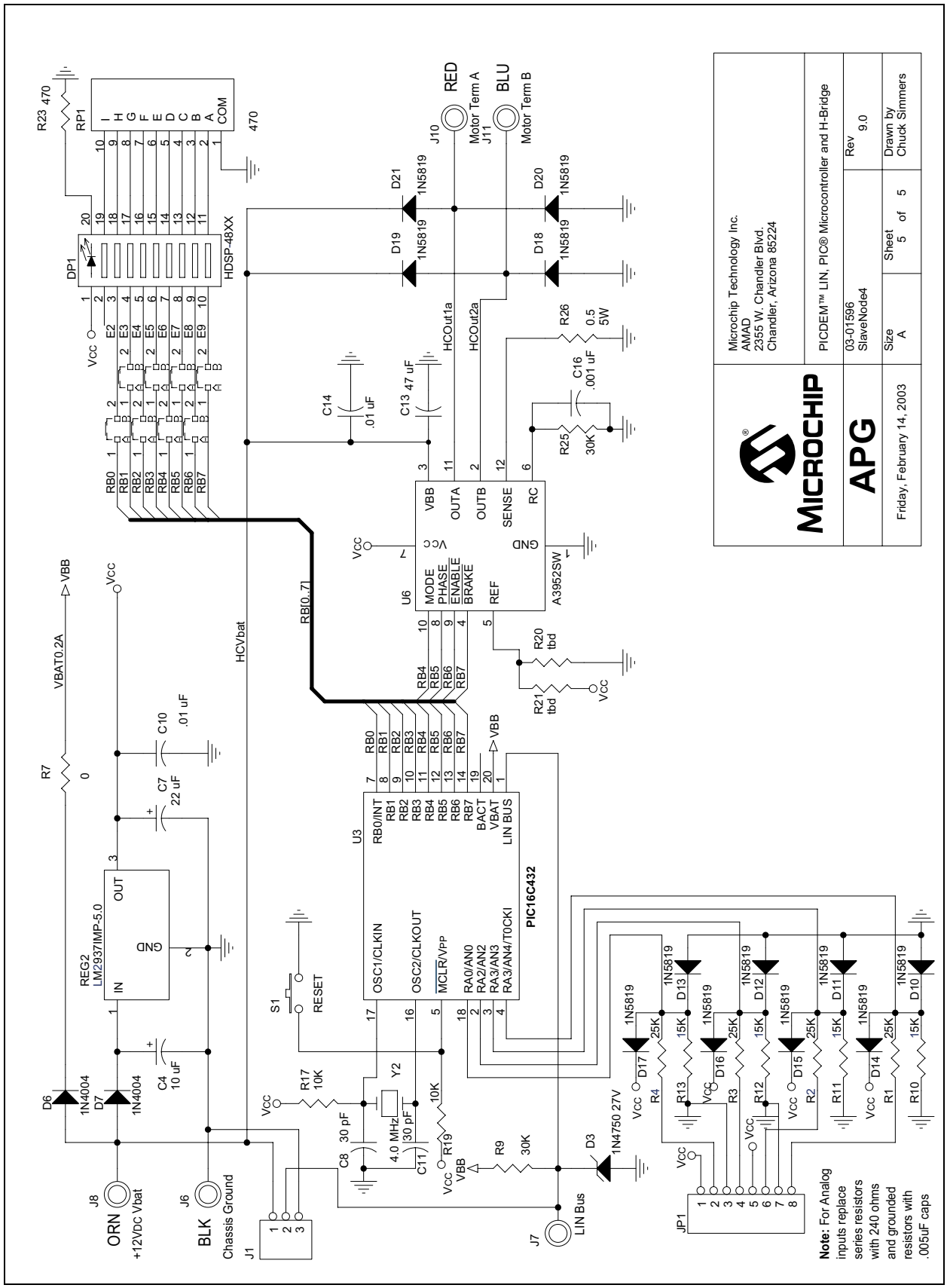


FIGURE A-6: INSTRUMENT CONTROL CONSOLE

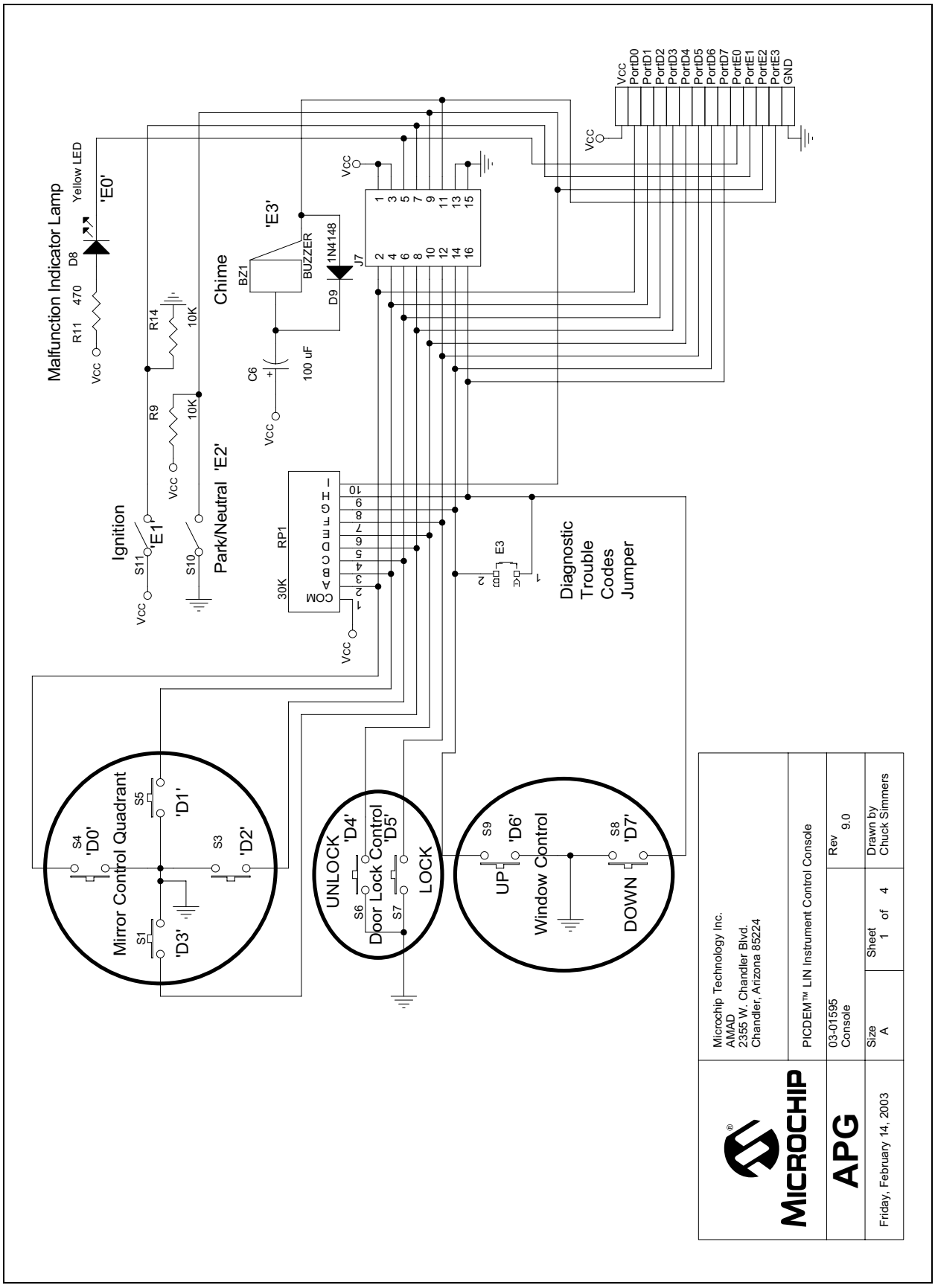


FIGURE A-7: MASTER NODE AND CAN INTERFACE

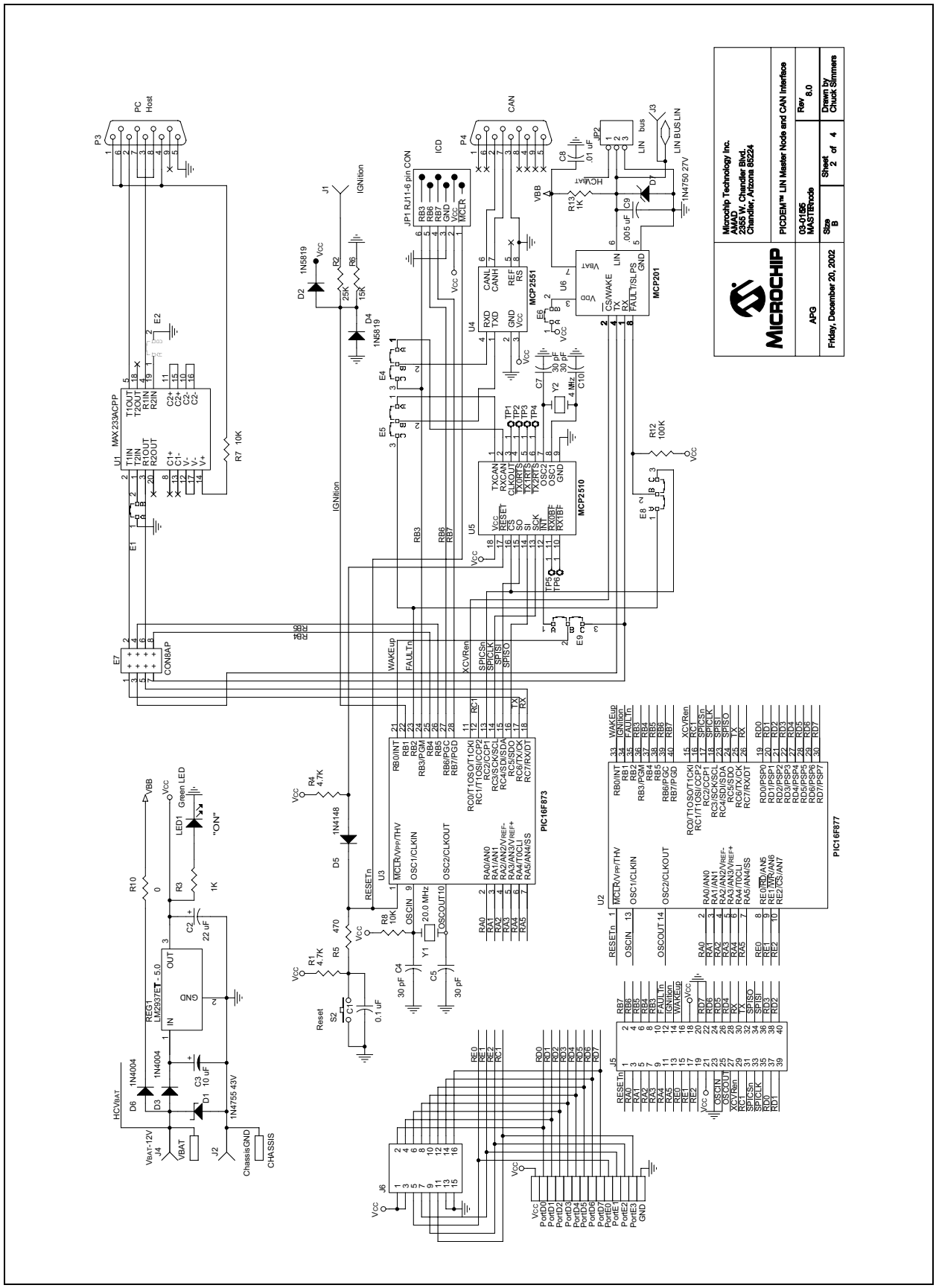


FIGURE A-8: REMOTE KEYLESS ENTRY DECODER

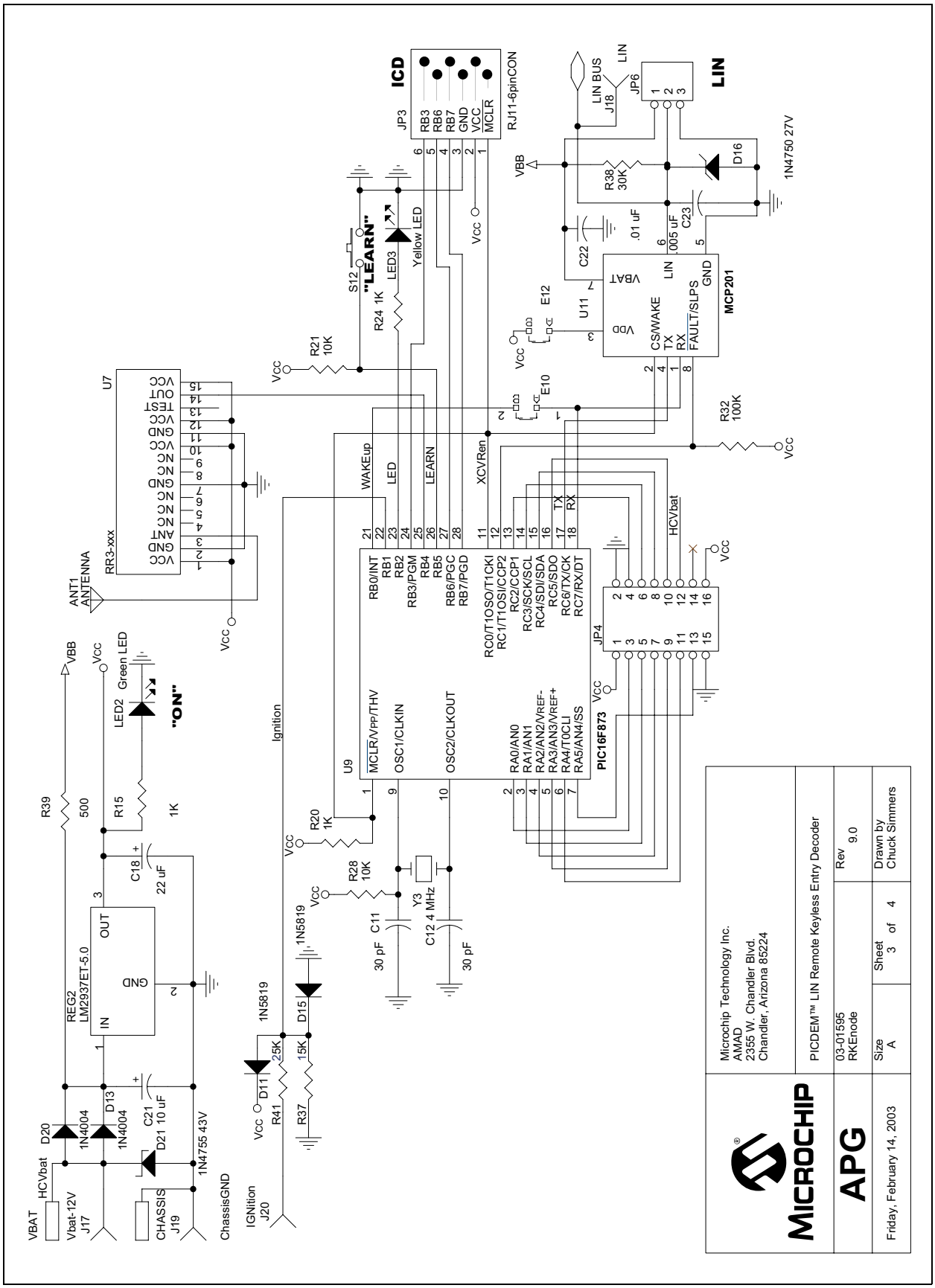
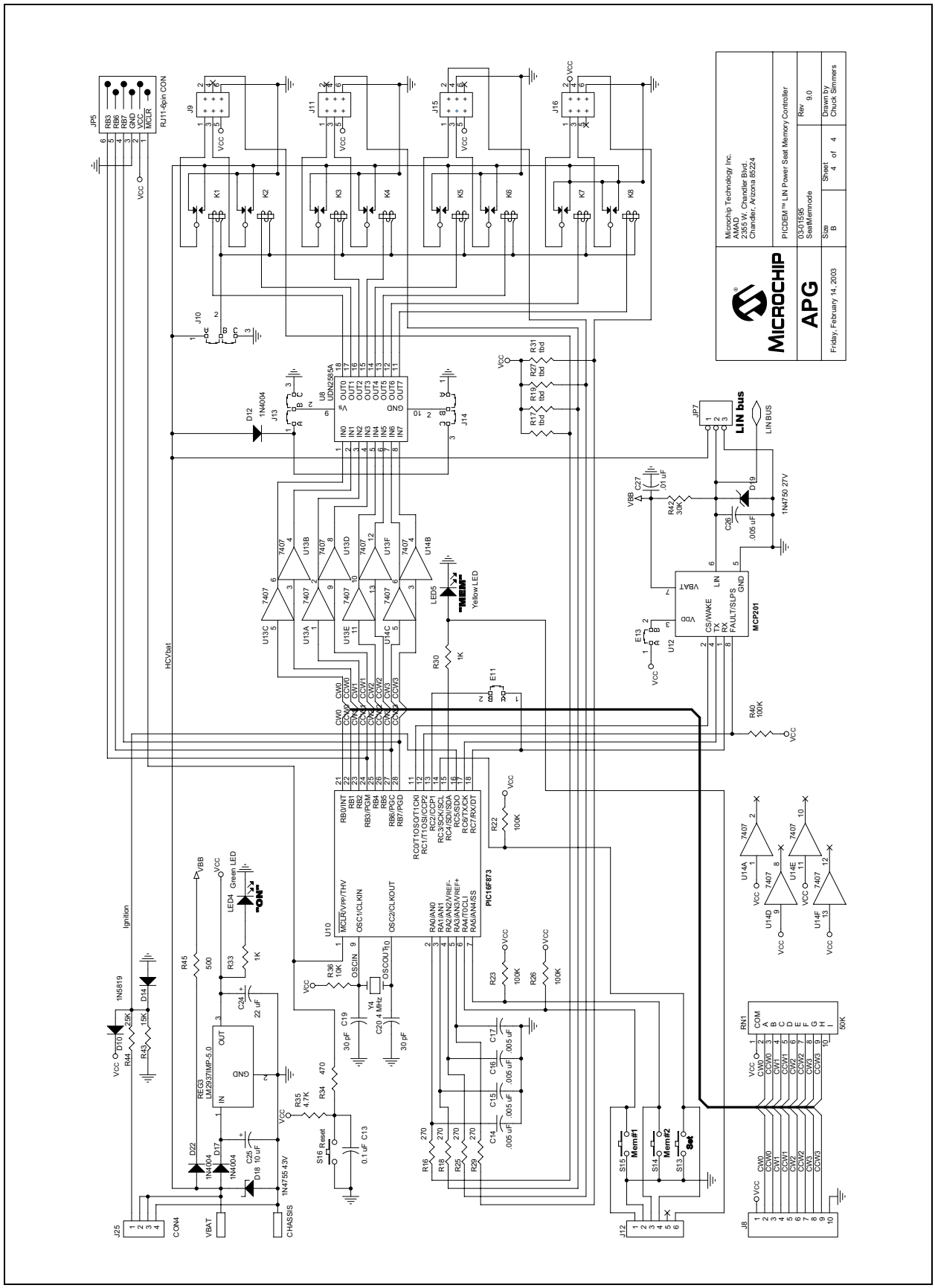


FIGURE A-9: POWER SEAT MEMORY CONTROLLER



NOTES:

Appendix B. LIN bus Master

B.1 LIN bus MASTER CODE


```
;*******************************************************************************
; * Title            : LIN Bus Master                                           *
; * Author           : Thomas Schmidt                                           *
; * Date             : 06.06.2000                                               *
; * Revision         : 0.1                                                      *
; * Last Modified    : 12.06.2000                                               *
; * Description      : This program implements the LIN Bus master based on USART *
;
;*******************************************************************************

       LIST p=16f874, r=hex

       #include <p16f874.inc>

       errorlevel  -302             ; suppress message 302 from list file

       ;*********************************************************************
       ;* Fuse configuration                                                *
       ;*********************************************************************
       __CONFIG
_CP_OFF&_WDT_OFF&_HS_OSC&_BODEN_OFF&_PWRTE_ON&_CPD_OFF&_DEBUG_OFF&_LVP_OFF&_WRT_ENABLE
_OFF

       #define     ABT_ERROR         0x01      ; Abritation error code
       #define     TRANSMIT_OK       0x02      ; Transmission was successful

       ; Definitions for Commands for Window. This definitions will be loaded
       ; into the first data byte
       #define RearViewLeft          0x14   ; Command for Rear view mirror left
       #define RearViewRight         0x24   ; Command for Rear view mirror right
       #define RearViewDown          0x34   ; Command for Rear view mirror down
       #define RearViewUp            0x04   ; Command for Rear view mirror up
       #define RearViewOff           0x00   ; Command for Rear view mirror off


       ; Definition of function IDs. These values are the ID values
       #define LockDoors             0x10   ; ID Value for LockDoors
       #define UnlockDoors           0x11   ; ID Value for Unlock Doors
```

```
        #define WindowsUp             0x12   ; ID Value for Windows up
        #define WindowsDown           0x13   ; ID Value for Windows down
        #define TiltSeatBack          0x18   ; ID Value for Tilt seat back
        #define TiltSeatForward       0x19   ; ID Value for Tilt seat forward
        #define SlideSeatForward      0x1a   ; ID Value for Slide seat forward
        #define SlideSeatBackward     0x1b   ; ID Value for Slide seat backward
        #define FrontEdgeSeatUp       0x1c   ; ID Value for Front Edge seat up
        #define FrontEdgeSeatDown     0x1d   ; ID Value for Front Edge Seat down
        #define BackEdgeSeatUp        0x1e   ; ID Value for Back edge seat up
        #define BackEdgeSeatDown      0x1f   ; ID Value for Back edge seat down

        #define LeftMirror            0x13   ; This ID selects the left mirror


        ; Other Definitions
        #define RXMODE                0         ; Receive mode
        #define TXMODE                1         ; Transmit mode
        #define DATAPOINTER           0x30      ; Address of data bytes
        #define CRC_ERROR             1         ; CRC_ERROR occured
        #define CRC_OK                0         ; No CRC_ERROR occured
        #define NoAction              0x00      ; No action required, because no key was
                                                ; pressed

        cblock      0x20
                    LINCOUNTER                  ; counter for LIN Bus bytes
                    ID_TEMP                     ; temporary register for ID byte
                    TEMP_COUNTER                ; temporary counter register
                    MESSAGE_COUNTER             ; number of bytes to receive or transmit
                    TEMP                        ; temporary register
                    TEMP1                       ; temporary register
                    TEMP2                       ; temporary register
        endc

                ORG 0x00
RESETVECTOR     goto   StartUp      ; goto StartUp routine


                ORG 0x04
INTVECTOR       goto   StartUp      ; no interrupts, therefore goto StartUp routine
```

```
      ; ***************************************************************************
      ; * ROUTINE             : StartUp                                           *
      ; * AUTHOR              : Thomas Schmidt                                    *
      ; * DATE LAST MODIFIED  : 08/14/2001                                        *
      ; * REVISION            : 1.0                                               *
      ; * CHANGES             : none                                              *
      ; * INPUT PARAMETER     : none                                              *
      ; * OUTPUT PARAMETER    : none                                              *
      ; * DESCRIPTION         : This routine is called after power-up. PORTS and  *
      ; *                       peripherals are initialized.                      *
      ; ***************************************************************************
StartUp   call    StartUpInit            ; Initialize PORTS and peripherals
          call    l_sys_init             ; initialize USART for LIN Bus communication
          call    StartUpRAMInit         ; Initialize RAM after power-up

          ; toggle CS line for transceiver
          bsf     PORTC, 0               ; set RC0 to 1
          bsf     STATUS, RP0            ; select bank1
          bcf     TRISC, 0               ; make RC0 an output
          bcf     STATUS, RP0            ; select bank0
          bcf     PORTC, 0               ; enable LIN Bus transceiver
          bsf     PORTC, 0               ;

     ; ***************************************************************************
      ; * ROUTINE             : Main Routine                                      *
      ; * AUTHOR              : Thomas Schmidt                                    *
      ; * DATE LAST MODIFIED  : 08/14/2001                                        *
      ; * REVISION            : 1.0                                               *
      ; * CHANGES             : none                                              *
      ; * INPUT PARAMETER     : none                                              *
      ; * OUTPUT PARAMETER    : none                                              *
      ; * DESCRIPTION         : This routine calls routines to Scan PORTD. After the  *
      ; *                       port is scan the LIN Bus data frame is transmitted.   *
      ; *                                                                         *
      ; ***************************************************************************
MainRoutine  call   ReadKeysPortD                ; read keys from PORTD
             movwf  TEMP                         ; copy result from Scan into TEMP
             movwf  ID_TEMP                      ; Load ID_TEMP with code from Key
             ; check if no key was pressed
             xorlw  0x00                         ; check if no action is required,
                                                 ; which means no key was pressed
             btfsc  STATUS, Z                    ; Check zero Flag
             goto   MainRoutine                  ; no key was pressed

             ; move command into first data byte
             movlw  DATAPOINTER                  ; point to frist data byte
             movwf  FSR                          ; with FSR register

             call   l_id_gen                     ; generated parity bits
             call   l_checksum                   ; generated checksum

             ; Reset Data pointer to Start Postion
             movlw  DATAPOINTER                  ; point data bytes
             movwf  FSR

             ; Transmit LIN Bus Data
             call   l_u8wr_sss                   ; transmit data

EndMainRoutine    goto     MainRoutine          ; do forever
```

```
      ; ************************************************************************
      ; * ROUTINE           : LIN Bus Transmission/Reception Routine           *
      ; * AUTHOR            : Thomas Schmidt                                   *
      ; * DATE LAST MODIFIED : 08/14/2001                                      *
      ; * REVISION          : 1.0                                              *
      ; * CHANGES           : none                                             *
      ; * INPUT PARAMETER   : none                                             *
      ; * OUTPUT PARAMETER  : none                                             *
      ; * DESCRIPTION       : This routine sends out a LIN Bus data frame and/or *
      ; *                     receives or transmits data. The reception or trans.*
      ; *                     of data is defined in the ID itself. After the ID-Byte*
      ; *                     is sent it is decoded in a look up table if data   *
      ; *                     should be transmitted or received. Depending on    *
      ; *                     the result either a transmit or receive mode is    *
      ; *                     activated.                                         *
      ; *                     ID_TEMP = holds the ID to send (parity is already  *
      ; *                               generated).                              *
      ; *                     DATAPOINTER = points to first data byte. Last byte *
      ; *                                   is CRC byte for transmission otherwise*
      ; *                                   last byte is data byte               *
      ; *                                                                        *
      ; ************************************************************************
l_u8_wr_sss  swapf   ID_TEMP, w         ; get ID4 and ID5 into lower bits
             andlw   b'00000011'        ; mask two bits
             call    table_tx           ; get data length code
             addlw   0x01               ; add one for CRC checksum
             movwf   MESSAGE_COUNTER    ; copy data length into TEMP register

             ; change baud rate for transmission
             bcf     RCSTA, SPEN   ; turn USART off
             bsf     STATUS, RP0   ; select Page 1
             ;movlw  0x13          ; Initialize USART for 12.5KBaud communication @4MHz
                                   ; Nominal Baudrate = 19.2KBaud @ 4MHz
             ;movlw  0x28          ; Initialize USART for 6KBaud
                                   ; communcation @ 4MHz
                                   ; Nominal Baudrate = 9.6KBaud @ 4MHz

             bcf     TXSTA, BRGH   ; select slow baud rate for 20MHz communciation
             movlw   0x81          ; Initialize USART for 2.4Kbaud @ 20MHz (BRGH=0)

             movwf   SPBRG         ; initialize SPBRG register
             bcf     STATUS, RP0   ; select Page 0
             bsf     RCSTA, SPEN   ; turn USART on

             ; transmit synchronization break
             movlw   0x00          ; Send Synchbreak Signal
             movwf   TXREG         ; Send 0x00
test_synchb  btfss   PIR1, RCIF    ; check if data was send
             goto    test_synchb   ; data not fully received

             movf    RCREG, w      ; copy data receive into w-register
             xorlw   0x00          ; check if data transmited is the same like
                                   ; data received.
             btfss   STATUS, Z     ; check zero flag
             retlw   ABT_ERROR     ; result is not the same, therefore an abritration
                                   ; error occured

             ; transmit synchronization field
             bcf     RCSTA, SPEN   ; turn USART off
```

```
       bsf     STATUS, RP0    ; select Page 1 in order to change baudrate to 19.2Kbaud
       ;movlw  0x19           ; change baudrate to 9.6KBaud @ 4MHz
       ;movlw  0x0c           ; change baudrate to 19.2Kbaud @ 4MHz

       bcf     TXSTA, BRGH    ; select slow rate for 20MHz communication
       movlw   0x1f           ; change baudrate to 9.6KBaud @ 20MHz (BRGH=0)

       movwf   SPBRG          ; change baudrate
       bcf     STATUS, RP0    ; select Page 0

       bsf     RCSTA, SPEN    ; turn USART on
       movlw   0x55           ; synch. field
       movwf   TXREG          ; send synch field
test_synchfield btfss PIR1, RCIF        ; check if data was send
                goto  test_synchfield   ; data not fully received
                movf  RCREG, w          ; copy data receive into w-register
                xorlw 0x55              ; check if data transmited is the same like
                                        ; data received.
                btfss STATUS, Z         ; check zero flag
                retlw ABT_ERROR         ; result is not the same, therefore an
                                        ; abritration error occured

                ; send identifier byte
                movf  ID_TEMP, w        ; copy ID_TEMP into
                movwf TXREG             ; transmit register
test_ID  btfss  PIR1, RCIF         ; check if data was send
         goto   test_ID            ; data not fully received
         movf   RCREG, w           ; copy data receive into w-register
         xorwf  RCREG, w           ; check if data transmited is the same like
                                   ; data received. (w-reg = transmited data)
         btfss  STATUS, Z          ; check zero flag
         retlw  ABT_ERROR          ; result is not the same, therefore an abritration
                                   ; error occured

         ; check if data has to be received or transmitted
         movf   ID_TEMP, w         ; copy ID into W register
         andlw  0x0f               ; Delete upper four bits of Identifier byte (these
                                   ; bit include parity bits and data lenght code).
                                   ; Lower bit are the indifier bits.

         call   DecodeIDTable      ; Decode Identifier bits
         bcf    PCLATH, 1          ; Reset PCLATH register after Look-up table call
         addwf  PCL, f             ; add mode to low byte of PC
         goto   ReceiveMode        ; receive data
         goto   TransmitMode       ; transmit data

         ; receive data
ReceiveMode  movlw   DATAPOINTER  ; Point to data
             movwf   FSR          ; point to location where received information
                                  ; is going to be stored
TestRXData   btfss   PIR1, RCIF                ; check if data was received
             goto    TestRXData                ; no data was not received yes
             movf    RCREG, w                  ; copy received data into w-register
             movwf   INDF                      ; copy into RAM
             incf    FSR, f                    ; point to next location
             decfsz  MESSAGE_COUNTER, f        ; decrement number of bytes to receive
             goto    TestRXData                ; read next data

             ; check CRC value
             call    l_checksum                ; check CRC value
```

```
            return                              ; return to main

            ; transmit data bytes
TransmitMode movf   INDF, w                     ; copy data byte transmit register
            movwf   TXREG                       ; and transmit data

test_databyte btfss  PIR1, RCIF         ; check if data was send
             goto   test_databyte       ; data not fully received
             movf   RCREG, w            ; copy data receive into w-register
             xorwf  RCREG, w            ; check if data transmited is the same like
                                        ; data received. (w-reg = transmitted data)
             btfss  STATUS, Z           ; check zero flag
             retlw  ABT_ERROR           ; result is not the same, therefore an
                                        ; abritration error occured
             incf   FSR, f              ; point to next data byte
             decfsz MESSAGE_COUNTER, f ;decrement number of transmitted data bytes by
                                        ; one
             goto   TransmitMode        ; get ready for the next data byte
             retlw  TRANSMIT_OK         ; the transmission was successful


table_tx  addwf   PCL, f                ; add value to low byte of program counter
          retlw   0x02                  ; data length = 2 bytes
          retlw   0x02                  ; data length = 2 bytes
          retlw   0x04                  ; data length = 4 bytes
          retlw   0x08                  ; data length = 8 bytes

      ; ************************************************************************
      ; * ROUTINE              : StartUp initialization                        *
      ; * AUTHOR               : Thomas Schmidt                                *
      ; * DATE LAST MODIFIED   : 08/14/2001                                    *
      ; * REVISION             : 1.0                                           *
      ; * CHANGES              : none                                          *
      ; * INPUT PARAMETER      : none                                          *
      ; * OUTPUT PARAMETER     : none                                          *
      ; * DESCRIPTION          : This routine initializes PORTs and peripherals after*
      ; *                        power-up.                                     *
      ; *                                                                      *
      ; ************************************************************************
StartUpInit   clrf   PORTA                  ; reset PORTA
              clrf   PORTB                  ; reset PORTB
              clrf   PORTC                  ; reset PORTC
              clrf   PORTD                  ; reset PORTD
              clrf   PORTE                  ; reset PORTD

              bsf    STATUS, RP0            ; select page 1
              bcf    TRISE, 0               ; make RE0 an output
              movlw  0xff                   ; make PORTD all inputs
              movwf  TRISD                  ; init TRISD register
              clrf   TRISA                  ; make all pins on portA outputs
              movlw  0xff                   ;
              movwf  ADCON1                 ; make all pin digital IOs
              bcf    STATUS, RP0            ; select page 0
              bcf    PORTE, 0               ; turn LED on
```

```
       ;***********************************************************************
       ; * ROUTINE              : RAM Initialization Routine                    *
       ; * AUTHOR               : Thomas Schmidt                                *
       ; * DATE LAST MODIFIED   : 08/14/2001                                    *
       ; * REVISION             : 1.0                                           *
       ; * CHANGES              : none                                          *
       ; * INPUT PARAMETER      : none                                          *
       ; * OUTPUT PARAMETER     : none                                          *
       ; * DESCRIPTION          : This routine initializes RAM (general purpose RAM)  *
       ; *                                                                      *
       ;***********************************************************************
StartUpRAMInit  movlw    DATAPOINTER            ; first address for data
                movwf    FSR                    ; load into FSR register
                movlw    0x0d                   ; initialize nine register
                movwf    TEMP                   ; TEMP is counter register
LoopInit        movlw    0x0d                   ; initialization value
                movwf    INDF                   ; use indirect addressing
                incf     FSR, f                 ; point to next RAM location
                decfsz   TEMP, f                ; decrement temp register
                goto     LoopInit               ; do until done

       ;***********************************************************************
       ; * ROUTINE              : System Initialization Routine for USART       *
       ; * AUTHOR               : Thomas Schmidt                                *
       ; * DATE LAST MODIFIED   : 08/14/2001                                    *
       ; * REVISION             : 1.0                                           *
       ; * CHANGES              : none                                          *
       ; * INPUT PARAMETER      : none                                          *
       ; * OUTPUT PARAMETER     : none                                          *
       ; * DESCRIPTION          : This routine initializes the USART for the LIN Bus  *
       ; *                        communication. BaudRate = 19.2KBaud           *
       ; *                                                                      *
       ;***********************************************************************
l_sys_init   bcf     RCSTA, SPEN            ; turn USART off
             bsf     PORTC, 6               ; set TX line to high
             bsf     STATUS, RP0            ; select Page 1
             movlw   b'00100100'            ; mask for TXSTA register
             movwf   TXSTA                  ; initialize TXSTA register
             movlw   b'10111111'            ; initialize TRISC register
             movwf   TRISC                  ; TRISC register
             bcf     STATUS, RP0            ; select Page 0
             movlw   b'00010000'            ; mask for RCSTA register
             movwf   RCSTA                  ; initialize RCSTA register
             bsf     RCSTA, SPEN            ; turn USART on
             retlw   0x00                   ; initialization sucessful => return 0x00
```

```
      ; ***************************************************************************
      ; * ROUTINE: ID Partiy Bit Generation                                       *
      ; * AUTHOR: Thomas Schmidt                                                  *
      ; * DATE LAST MODIFIED: 08/14/2001                                          *
      ; * REVISION: 1.0                                                           *
      ; * CHANGES: none                                                           *
      ; * INPUT PARAMETER: none                                                   *
      ; * OUTPUT PARAMETER: Updated ID with Parity bits in ID_TEMP                *
      ; * DESCRIPTION: This routine generated the parity bits P1 and P0           *
      ; *              for the identifier byte according to the LIN Bus           *
      ; *              specification 1.2.                                         *
      ; *                                                                         *
      ; ***************************************************************************
l_id_gen  movf      ID_TEMP, w    ; Temporary ID location

          ; calculate P0
          movwf     TEMP1         ; move ID value into TEMP1
          movwf     TEMP2         ; move ID into TEMP2
          rrf       TEMP1, f      ; rotate ID_TEMP one to the right (get ID1)
          movf      TEMP1, w      ; copy TEMP1 to w
          xorwf     TEMP2, f      ; TEMP2=ID0 XOR ID1
          rrf       TEMP1, f      ; get ID2
          movf      TEMP1, w      ; copy ID2 into w-register
          xorwf     TEMP2, f      ; TEMP2= TEMP2 XOR ID2
          bcf       STATUS, C     ; clear carry flag
          rrf       TEMP1, f      ; get ID3 into bit 0
          rrf       TEMP1, w      ; ID4 into bit 0 and store result in w-register
          xorwf     TEMP2, f      ; TEMP2 = TEMP2 XOR ID4

             btfsc  TEMP2, 0      ; test if bit is zero or one
             goto   set_p0        ; P0=1
             bcf    ID_TEMP, 6    ; P0=0
             goto   cal_p1        ;
set_p0       bsf    ID_TEMP, 6    ; set P0 to 1

             ; calculate P1
cal_p1    movf     ID_TEMP, w     ; copy ID_TEMP into w-register
          movwf    TEMP1          ; copy ID_TEMP into TEMP1
          movwf    TEMP2          ; and TEMP2
          rrf      TEMP2, f       ; ID1 into bit0
          rrf      TEMP1, f       ; ID1 into bit0
          rrf      TEMP1, f       ; ID2 into bit0
          rrf      TEMP1, f       ; ID3 into bit0
          movf     TEMP1, w       ; copy TEMP1 into w-register
          xorwf    TEMP2, f       ; TEMP2 = ID1 XOR ID3
          rrf      TEMP1, f       ; ID4 into bit0
          movf     TEMP1, w       ; TEMP1 into w-register
          xorwf    TEMP2, f       ; TEMP2 = TEMP2 XOR ID4
          rrf      TEMP1, w       ; ID5 into bit0
          xorwf    TEMP2, f       ; TEMP2 = TEMP2 XOR ID5
          comf     TEMP2, f       ; negate TEMP2

          btfsc    TEMP2, 0       ; check if P1=1
          goto     set_p1         ; P1=1
          bcf      ID_TEMP, 7     ; P1=0
          goto     testb          ; return
set_p1    bsf      ID_TEMP, 7     ; set P1

testb     return                  ; return to main
```

```
      ; ***************************************************************************
      ; * ROUTINE            : CRC Check and Generation                           *
      ; * AUTHOR             : Thomas Schmidt                                     *
      ; * DATE LAST MODIFIED : 08/14/2001                                         *
      ; * REVISION           : 1.0                                                *
      ; * CHANGES            : none                                               *
      ; * INPUT PARAMETER    : none                                               *
      ; * OUTPUT PARAMETER   : CRC Check :CRC check is appended after last Data Byte *
      ; *                       CRC Gen   :CRC is check and CRC_OK or CRC_ERROR is  *
      ; *                       into the w-register                                *
      ; * DESCRIPTION        : This routine generates or check CRC based on the   *
      ; *                       Modulo-256 checksum defined in the LIN Bus spec. 1.2 *
      ; *                                                                          *
      ; ***************************************************************************
l_checksum  movlw  DATAPOINTER     ; point to first data byte location
            movwf  FSR             ; initialize FSR register
            swapf  ID_TEMP, w      ; get ID4 and ID5 into bit0 and bit 1
            andlw  0x03            ; get rid of all other bits
            call   table_tx        ; get number of data bytes to be transmitted

            movwf  TEMP_COUNTER    ; copy number of data bytes into Temp-Counter

            decf   TEMP_COUNTER, f ; Number of data bytes - 1 = number of loop counts
            movf   INDF, w         ; copy first data byte into w-register
            movwf  TEMP            ; copy first data byte into temp register
next_calc   incf   FSR, f          ; point to next data memory location
            movf   INDF, w         ; move data into w-register
            addwf  TEMP, f         ; add data byte to temp and store in temp
            btfsc  STATUS, C       ; add with carry?
            incf   TEMP, f         ; yes, increment TEMP
            decfsz TEMP_COUNTER, f ; decrement bit counter
            goto   next_calc       ; calculate next

            ; check if data has to be received or transmited
            movf   ID_TEMP, w      ; copy ID into W register
            andlw  0x0f            ; Delete upper four bits of Identifier byte (these
                                   ; bit include parity bits and data lenght code).
                                   ; Lower bit are the indifier bits.

          call     DecodeIDTable   ; Decode Identifier bits
          bcf      PCLATH, 1       ; Reset PCLATH register

          addwf    PCL, f          ; add mode to low byte of PC
          goto     CRCCheck        ; data was received therefore check CRC
          goto     CRCAppend       ; data is going to be transmited, therefore
                                   ; append CRC

          ; generate CRC value
CRCAppend comf     TEMP, f        ; complement CRC value
          movf     TEMP, w        ; copy checksum into w-register
          incf     FSR, f         ; point to location for checksum
          movwf    INDF           ; copy checksum behind
          return                  ; return to main routine

CRCCheck  incf   FSR, f           ; point to CRC data
          movf   INDF, w          ; copy received CRC value into w-register
          addwf  TEMP, w          ; add received CRC to calculated CRC
          xorlw  0xff             ; Result should be 0xFF after XOR result is zero
          btfss  STATUS, Z        ; is result zero?
```

```
          retlw  CRC_ERROR        ; return with CRC_ERROR in w-register
          retlw   CRC_OK          ; return with CRC_OK in w-register

        ; ************************************************************************
        ; * ROUTINE               : PORTD Scan Routine                           *
        ; * AUTHOR                : Thomas Schmidt                               *
        ; * DATE LAST MODIFIED    : 08/14/2001                                   *
        ; * REVISION              : 1.0                                          *
        ; * CHANGES               : none                                         *
        ; * INPUT PARAMETER       : ID_TEMP                                      *
        ; * OUTPUT PARAMETER      : LIN Bus ID based on what key is pressed      *
        ; * DESCRIPTION           : This Routine scans PORTD for what key is pressed.  *
        ; *                         After the scan, the routine check what key is     *
        ; *                         pressed and return the ID for the key into the    *
        ; *                         w-register.                                    *
        ; *                                                                        *
        ; ************************************************************************
ReadKeysPortD  movf   PORTD, w            ; copy content of PORTD into W-register
               movwf  TEMP                ; store in temp register
               xorlw  0xff                ; see if anything changed
               btfsc  STATUS, Z           ; result is zero therefore no key was pressed
               goto   ReadKeysPortD       ; keep on reading


               call   Delay25ms           ; debounce key (delay for 25ms)

               ; read key again to see if it still pressed

               movf   TEMP, w             ; copy content of PORTD into W-register
               xorwf  PORTD, w            ; see if anything changed
               btfsc  STATUS, Z           ; result is zero therefore key is still
                                          ; pressed
               goto   ReadKeysPortD       ; keep on reading

               rrf    TEMP, f             ; rotate key register
               btfss  STATUS, C           ; was MirrorUp key pressed?
               retlw  RearViewUp          ; yes, return wiht RearViewUp value
               rrf    TEMP, f             ; check next key
               btfss  STATUS, C           ; was MirrorRight key pressed?
               retlw  RearViewRight       ; yes, return with RearViewRight value
               rrf    TEMP, f             ; check MirrorLeft key
               btfss  STATUS, C           ; was MirrorLeft key pressed?
               retlw  RearViewDown        ; yes, return with RearViewLeft value
               rrf    TEMP, f             ; check MirrorKeyDown
               btfss  STATUS, C           ; was MirrorKeyDown pressed?
               retlw  RearViewLeft        ; yes, return with RearViewDown value
               rrf    TEMP, f             ; check UNLOCK key
               btfss  STATUS, C           ; was UNLOCK key pressed?
               retlw  UnlockDoors         ; yes, return with UnlockDoors value
               rrf    TEMP, f             ; was LOCK doors key pressed
               btfss  STATUS, C           ; check if LOCK doors key was pressed
               retlw  LockDoors           ; yes, return with LockDoors value
               rrf    TEMP, f             ; check if WindowUp key was pressed
               btfss  STATUS, C           ; was WindowUp key pressed?
               retlw  WindowsUp           ; yes, return with WindowsUp value
               rrf    TEMP, f             ; check if WindowsDown key was pressed
               btfss  STATUS, C           ; was WindowsDown key pressed?
               retlw  WindowsDown         ; yes, return with WindowsDownValue
               retlw  NoAction            ; No Key was pressed
```

```
        ; ************************************************************************
        ; * ROUTINE             : Delay Routine for 25ms @ 4MHz                  *
        ; * AUTHOR              : Thomas Schmidt                                 *
        ; * DATE LAST MODIFIED  : 08/14/2001                                     *
        ; * REVISION            : 1.0                                            *
        ; * CHANGES             : none                                           *
        ; * INPUT PARAMETER     : none                                           *
        ; * OUTPUT PARAMETER    : none                                           *
        ; * DESCRIPTION         : This routine generates a delay for 25ms @ 4MHz  *
        ; *                                                                      *
        ; ************************************************************************
Delay25ms  movlw     0xff         ; intialize TEMP1
           movwf     TEMP1        ; with 0xff
           movlw     0x14         ; initialize TEMP2
           movwf     TEMP2        ; with 0x14
Loop1      decfsz    TEMP1, f     ; decrement TEMP1
           goto      ExLoop1      ; extent loop 1
           decfsz    TEMP2, f     ; Decrement TEMP2
           goto      ReloadTemp   ; Reload TEMP1
           return                 ; return to main routine

ReloadTemp movlw     0xff         ; load TEM1 with
           movwf     TEMP1        ; 0xff and go back
           goto      Loop1        ;

ExLoop1    goto      Loop1        ; add two insturction cycles to it
```

```
        ; ******************************************************************************
        ; * ROUTINE               : Decode ID Table                                      *
        ; * AUTHOR                : Thomas Schmidt                                       *
        ; * DATE LAST MODIFIED    : 08/14/2001                                           *
        ; * REVISION              : 1.0                                                  *
        ; * CHANGES               : Table offset value in w-register                     *
        ; * INPUT PARAMETER       : RXMODE (switch to receive mode) or TXMODE (switch    *
        ; *                         transmit mode)                                       *
        ; * OUTPUT PARAMETER      : Updated ID with Parity bits in ID_TEMP               *
        ; * DESCRIPTION           : This routine generated the parity bits P1 and P0     *
        ; *                         for the identifier byte according to the LIN Bus     *
        ; *                         specification 1.2.                                   *
        ; *                                                                              *
        ; ******************************************************************************
                org   0x200
DecodeIDTable   bsf     PCLATH, 1
                addwf   PCL, f      ; add to PC
                retlw   TXMODE      ; Receive data from bus
                retlw   TXMODE      ; Receive data from bus
                retlw   TXMODE      ; Receive Data from bus
                retlw   TXMODE      ; Receive Data from bus
                retlw   TXMODE      ; Receive data from bus
                retlw   TXMODE      ; Receive data from bus
                retlw   TXMODE      ; Receive Data from bus
                retlw   TXMODE      ; Receive Data from bus
                retlw   TXMODE      ; Receive data from bus
                retlw   TXMODE      ; Receive data from bus
                retlw   TXMODE      ; Receive Data from bus
                retlw   TXMODE      ; Receive Data from bus
                retlw   TXMODE      ; Receive data from bus
                retlw   TXMODE      ; Receive data from bus
                retlw   TXMODE      ; Receive Data from bus
                retlw   TXMODE      ; Receive Data from bus


        END
```

Appendix C. LIN bus Slave for the PIC16C432

C.1 LIN bus SLAVE FOR THE PIC16C432 CODE


```
;*****************************************************************************
; * Title        : LIN bus Slave for the PIC16C432
; * Author       : Thomas Schmidt
; * Date         : 05.12.2000
; * Revision     : 0.4
; * Last Modified:
; * Description  :
; *****************************************************************************

                LIST P=16C432, r=hex

                ; *************************************************************************
                ; * Include files           *
                ; *************************************************************************
                #include <P16C432.INC>

                ; *************************************************************************
                ; * Register definitions    *
                ; *************************************************************************
                cblock   0x20
                         AUTOBAUD_LOW       ; low byte of bit-time coutner
                         AUTOBAUD_HIGH      ; high byte of bit-time counter
                         AUTOHALF_LOW       ; low byte of half the bit time
                         AUTOHALF_HIGH      ; high byte for half the bit time
                         TEMP1, TEMP2       ; temporary registers
                         RXTX_REG           ; receive register
                         MESSAGE_COUNTER    ; Countes number of bytes to receive or transmit
                         COUNTER            ; receive & Transmit counter register
                         ID_TEMP            ; Register for ID byte
                endc
```

```
                ; *********************************************************************
                ; * Other definitions
                ; *********************************************************************
                #define BITS            0x08     ; number of bits to receive
                #define RXMODE          0x00     ; Receive mode
                #define TXMODE          0x01     ; Transmit mode
                #define LISTENMODE      0x02     ; Listen only mode
                #define CRC_ERROR       0x01     ; CRC_ERROR occured
                #define CRC_OK          0x00     ; No CRC_ERROR occured
                #define PARITY_OK       0x00     ; No Parity error occcured
                #define PARITY_ERROR_P0 0x01     ; Parity error on P0 occured
                #define PARITY_ERROR_P1 0x02     ; Parity error on P1 occured
                #define DATAPOINTER     0x35     ; address of first data byte
                #define IGNORE          0x00     ; ignorê received command
                #define INCREMENT       0x01     ; increment PORTB
                #define DECREMENT       0x02     ; decrement PORTB



                ; *********************************************************************
                ; * Fuse configuration       *
                ; *********************************************************************
                __CONFIG _CP_OFF&_WDT_OFF&_XT_OSC&_PWRTE_ON&_BODEN_OFF

                ; *********************************************************************
                ; * Reset vector             *
                ; *********************************************************************
                ORG 0x00
                goto     MainRoutine


                ;*************************************************************************
                ; * ROUTINE                         : MainRoutine*
                ; * AUTHOR                          : Thomas Schmidt*
                ; * DATE LAST MODIFIED              : 08/14/2001*
                ; * REVISION                        : 1.0*
                ; * CHANGES                         : none*
                ; * INPUT PARAMETER                 : none*
                ; * OUTPUT PARAMETER                : none*
                ; * DESCRIPTION                     : Main routine. The main routine detects
                ; *                                   first the transmission time of the incom-
                                                     ming calibration character. After that the
                ; *                                   routine receives and transmits incomming
                ; *                                   characters.
                ;*************************************************************************
                ORG 0x06
MainRoutine     call     StartUp            ; Call Startup Routine

TestSynchByte   btfsc    PORTA, LINRX       ; Check for synch byte
                goto     TestSynchByte      ; Synch byte not received
TestEndSynch    btfss    PORTA, LINRX       ; check for end of synchbyte
                goto     TestEndSynch       ; end of synchronization byte not received


                ; Receive message on LIN bus
                call     LinHandler         ; call LIN bus handler
```

```
                ; Decode message and take action upon receive message Identifier
                movf     ID_TEMP, w         ; copy ID_TEMP into w-register
                andlw    0x0f               ; decode only lower four nibbles
                call     DecodeAction       ; see if LED has to be turn on
                clrf     PCLATH             ; reset PCLATH register
                addwf    PCL, f             ; add to PC
                goto     TestSynchByte      ; Ignore all other IDs
                goto     WindowUp           ; Window up function
                goto     WindowDown         ; Window down function



                ; WindowUp command was receive from Master
WindowUp        incf     PORTB, f           ; increment PORTB
                goto     TestSynchByte      ; Go back

                ; WindowDown command was receive from Master
WindowDown      decf     PORTB, f           ; decrement PORTB
                goto     TestSynchByte      ; go back


                ;*******************************************************************************
                ; * ROUTINE                 : StartUp
                ; * AUTHOR                  : Thomas Schmidt
                ; * DATE LAST MODIFIED      : 08/14/2001
                ; * REVISION                : 1.0
                ; * CHANGES                 : none
                ; * INPUT PARAMETER         : none
                ; * OUTPUT PARAMETER        : none
                ; * DESCRIPTION             : This routine is called after power-up. PORTS and
                ; *                           peripherals are initialized.
                ; *
                ;*******************************************************************************
StartUp         clrf     PORTA              ; set all latches of PORTA to zero
                clrf     PORTB              ; set all latches of PORTB to zero
                movlw    0x07               ; Make all pins on PORTA
                movwf    CMCON              ; digitial I/Os.
                bsf      STATUS,RP0         ; select page 1
                movlw    b'00000010         ; make RA1 an input
                movwf    PORTA              ; make all other pins on PORTA outputs
                clrf     PORTB              ; make all PORTB pins outputs
                bsf      LININTF, LINTX     ; set Transmit line high
                bsf      LININTF, LINVDD    ; turn LIN Vdd on.
                bcf      STATUS,RP0         ; select page 0

                ; Initialize Data RAM for LIN bus Communication
InitData        movlw    DATAPOINTER        ; point to first data location
                movwf    FSR                ; and initialize FSR register
                movlw    0x0f               ; Initialize 16 register
                movwf    TEMP1              ; temporary counter register
Count           movlw    0xaa               ; init value is 0xaa
                movwf    INDF               ; copy into location where FSR points to
                incf     FSR, f             ; point to next location
                decfsz   TEMP1, f           ; decrement temporary counter
                goto     Count              ; counter is not zero, therefore keep on
                                            ; initializing
                retlw    0x00               ; return to Main Routine
```

```
                ;*******************************************************************************
                ; * ROUTINE                  : LIN bus Transmission/Reception Routine
                ; * AUTHOR                   : Thomas Schmidt
                ; * DATE LAST MODIFIED       : 08/14/2001
                ; * REVISION                 : 1.0
                ; * CHANGES                  : none
                ; * INPUT PARAMETER          : none
                ; * OUTPUT PARAMETER         : none
                ; * DESCRIPTION              : This routine sends out a LIN bus data frame and/or
                ; *                            receives or transmits data. The reception or trans.
                ; *                            of data is defined in the ID itself. After the ID
                ; *                            -Byte is sent it is decoded in a look up table if
                : *                            data should be transmitted or received. Depending
                ; *                            on the result either a transmit or receive mode is
                ; *                            activated.
                ; *                            ID_TEMP = holds the ID to send (parity
                ; *                                    is already generated).
                ; *                            DATAPOINTER = points to first data byte. Last byte
                ; *                                        is CRC byte for transmission otherwise
                ; *                                        last byte is data byte
                ; *
                ;*******************************************************************************
LinHandler      clrf     AUTOBAUD_LOW       ; reset register
                clrf     AUTOBAUD_HIGH      ; reset register
                clrf     AUTOHALF_LOW       ; reset register
                clrf     AUTOHALF_HIGH      ; reset register
                clrf     COUNTER            ; reset counter
                clrf     MESSAGE_COUNTER    ; reset message counter register

WaitStartBit    btfsc    PORTA, LINRX       ; was start bit there
                goto     WaitStartBit       ; no, keep on looking

                ; yes, measure first low bit time
TestClear       btfss    PORTA, LINRX       ; was there a transition from low to high?
                goto     AutoMeasureSet     ; no, increment Autobaud counter registers
                incf     COUNTER,f          ; increment counter register
                goto     TestSet            ; yes, therefore increment bit counter and measure
                                              high time

                         ; Autobaud counter. The signal is sampled every 6 Instruction cycles
                         ; This means the number of countes equals 8*Tbit/6Tcy. Example:
                         ; Transmission rate 19.2Kbaud, Fosc=4MHz => Tbit=51us, 8*51us/6*1us = 68
                         ; counts. Counting the Transitions takes 9Tcy. This value has to be
                         ; added to the number of counts. Example from above: 68 Couts + 5*9Tcy
                         ; (5 because five transitions are counted.
AutoMeasureSet  incfsz   AUTOBAUD_LOW, f    ; increment Autobaud low register
                goto     TestClear          ; keep on testing
                incf     AUTOBAUD_HIGH,f    ; increment high byte of autobaud register
                goto     TestClear          ; keep on testing

AutoMeasureClr  incfsz   AUTOBAUD_LOW, f    ; increment Autobaud low register
                goto     TestSet            ; keep on testing
                incf     AUTOBAUD_HIGH,f    ; increment high byte of autobaud register
                goto     TestSet            ; keep on testing
```

```
TestSet         btfsc    PORTA, LINRX       ; check if pin is still high
                goto     AutoMeasureClr     ; no, there was not, measure time

                incf     COUNTER, f         ; increment counter register
                movf     COUNTER, w         ; check if all 8 bits were received
                xorlw    0x08               ;
                btfss    STATUS, Z          ; is result zero?
                goto     TestClear          ; No, therefore keep on measuring

                ; It takes 6Tcy to process the counter register. This is executed
                ; eight times => 8*6Tcy=48Tcy where the low byte is not update according
                ; to 6 countes. Therefore 48Tcy/6countes= 8. Eight countes have
                ; to be added to the low byte of the auto_baud_low register
                ; add one count to low byte, because auf LIN bus propagation time
                ; 4us to drive LIN bus high and 2V/us raise time. Therefore we have
                ; 10us delay before a slave sees a high value. 10us is two counts
                ; Therefore the total counts to be added are 10 counts (=0x0a)

                movlw    0x09
                addwf    AUTOBAUD_LOW, f    ; adjust low byte

                ; Calculation of transmission time for one bit
Divide          bcf      STATUS, C          ; clear carry bit
                rrf      AUTOBAUD_HIGH,f    ; rotate autobaud high register
                rrf      AUTOBAUD_LOW, f    ; rotate autobaud low register
                bcf      STATUS, C          ; clear carry bit
                rrf      AUTOBAUD_HIGH,f    ; rotate autobaud high register
                rrf      AUTOBAUD_LOW, f    ; rotate autobaud low register
                bcf      STATUS, C          ; clear carry bit
                rrf      AUTOBAUD_HIGH,f    ; rotate autobaud high register
                rrf      AUTOBAUD_LOW, f    ; rotate autobaud low register

                ; Calculate the transmission time for half the bit time (means
                ; divide transmission time of one bit by two).
CalcHalfBit     bcf      STATUS, C          ; clear carry bit
                rrf      AUTOBAUD_HIGH,w    ; rotate autobaud high register
                movwf    AUTOHALF_HIGH      ; copy result into AUTOHALF_HIGH register
                rrf      AUTOBAUD_LOW, w    ; rotate autobaud high register
                movwf    AUTOHALF_LOW       ; copy result into AUTOHALF_LOW register

                ; Adjust 16-bit counter for receive and transmit routine. This means
                ; that the overhead of instruction cycles in of the receive/transmit
                ; routine has to be substracted from the transmission time of one bit
                ; and half a bit.
AdjustLowByte   movlw    0x02               ; 18-19 instruction cycles overhead from
                                            ; transmit/receive routine. This overhead
                                            ; must be substracted from iterations
                subwf    AUTOBAUD_LOW, f    ; adjust low byte from Autobaud counter
                movlw    0x02               ; substract 2 from low byte of half the bit time
                subwf    AUTOHALF_LOW, f    ; substract from low byte of half the bit time

                         ; wait until Stop-bit comes in. Otherwise the reception of the next
                         ; bit will start to early.


EndStopBit      btfss    PORTA, LINRX       ; Wait for stop bit to be finished
                goto     EndStopBit         ; Stop bit not finished yet



                ; receive Identifier byte
                call     Receive            ; Receive Identifier byte

                ; store Identifier byte
                movf     RXTX_REG, w        ; copy Identifier byte into w-register
                movwf    ID_TEMP            ; copy Identifier into ID_TEMP register
```

```
                call     CheckParityBits    ; check if parity bits are correct

                ; Decode ID4 and ID5. These two bits indicate how many bytes of
                ; data have to be transmitted or received.
                swapf    RXTX_REG, w        ; change ID4 and ID5 to lower nibble
                andlw    0x03               ; delete rest
                call     DecDataLength      ; decode data length
                clrf     PCLATH             ; reset PCLATH register
                movwf    MESSAGE_COUNTER    ; store length of message into MESSAGE_LENGTH
                                            ; register
                incf     MESSAGE_COUNTER,f  ; increment message counter by one for
                                            ; receiving or transmitting CRC byte

                ; Decode Identifier byte.
                movf     RXTX_REG, w        ; load RXTX_REG into w-register (Identifierbyte into
                                            ; w-register)
                andlw    0x0f               ; Delete upper four bits of Identifier byte (these
                                            ; bit include parity bits and data lenght code).
Lower
                                            ; bit are the indifier bits.

                call     DecodeIDTable      ; Decode Identifier bits
                clrf     PCLATH             ; reset PCLATH register

                addwf    PCL, f             ; add mode to low byte of PC
                goto     ReceiveMode        ; receive data
                goto     TransmitMode       ; transmit data
                goto     ReceiveMode        ; receive data

TransmitMode    call     CheckCRC           ; generated CRC
                movlw    DATAPOINTER        ; point to first data byte
                movwf    FSR                ; initialize FSR register

TransmitNextD   movf     INDF, w            ; copy data byte into w-register
                movwf    RXTX_REG           ; copy data in RXTX_REG

                call     Transmit           ; transmit data
                incf     FSR, f             ; point to next location
                decfsz   MESSAGE_COUNTER, f ; decrement Message Counter by one
                goto     TransmitNextD      ; transmit next data
                retlw    0x00               ; return to main

                ; Receive Mode in this sequence data is received
ReceiveMode     movlw    DATAPOINTER        ; point to data location
                movwf    FSR                ; where data should be stored
ReceiveNextData call     Receive            ; receive next data
                movf     RXTX_REG, w        ; copy data into w-register
                movwf    INDF               ; copy data into data area
                incf     FSR, f             ; point to next location
                decfsz   MESSAGE_COUNTER, f ; decrement number of bytes to receive by one
                goto     ReceiveNextData    ; receive next data byte

                call     CheckParityBits    ; check if parity bits are correct
                call     CheckCRC           ; check if checksum is correct
                retlw    0x00               ; return to main
```

```
                ;*******************************************************************************
                ; * ROUTINE           : LIN bus Receive Routine
                ; * AUTHOR            : Thomas Schmidt
                ; * DATE LAST MODIFIED: 08/14/2001
                ; * REVISION          : 1.0
                ; * CHANGES           : none
                ; * INPUT PARAMETER   : none
                ; * OUTPUT PARAMETER  : none
                ; * DESCRIPTION       : This routine receives an 8-bit value via the LIN
                                        Bus
                ;*******************************************************************************
Receive         clrf    RXTX_REG          ; clear receive register
                movlw   BITS              ; number of bits to receive
                movwf   COUNTER           ; load number of bits into counter register


ReceiveStartBit btfsc   PORTA, LINRX      ; test for falling edge
                goto    ReceiveStartBit   ; start-bit not found
                call    DelayHalfBit      ; wait until middle of start-bit
                call    DelayFullBit      ; ignore start-bit and sample first
                                          ; data bit in the middle of the bit
ReceiveNext     btfsc   PORTA, LINRX      ; is LINRX zero or a one?
                bsf     STATUS,C          ; bit is a one => set carry bit
                btfss   PORTA, LINRX      ; is LINRX one or a zero?
                bcf     STATUS,C          ; LINRX is zero => clear carry bit
                rrf     RXTX_REG, f       ; rotate value into receive register
                call    DelayFullBit      ; call Delay routine
                decfsz  COUNTER, f        ; decrement receive count register by one
                goto    ReceiveNext       ; receive next bit
                retlw   0x00              ; return


                ;*******************************************************************************
                ; * ROUTINE              : LIN bus Transmit Routine
                ; * AUTHOR               : Thomas Schmidt
                ; * DATE LAST MODIFIED   : 08/14/2001
                ; * REVISION             : 1.0
                ; * CHANGES              : none
                ; * INPUT PARAMETER      : none
                ; * OUTPUT PARAMETER     : none
                ; * DESCRIPTION          : This routine transmits an 8-bit value via the LIN bus
                ; *******************************************************************************
Transmit        movlw   BITS              ; number of bit's to transmit
                movwf   COUNTER           ; initialize count register
                bsf     STATUS, RP0       ; select page 1
                bcf     LININTF, LINTX    ; generate start-bit
                bcf     STATUS, RP0       ; select page 0
                call    DelayFullBit      ; generate Delay for one bit-time
TransmitNext    rrf     RXTX_REG, f       ; rotate receive register
                bsf     STATUS, RP0       ; select page 1
                btfss   STATUS, C         ; test bit to be transmited
                bcf     LININTF, LINTX    ; Send a zero
                btfsc   STATUS, C         ; Check if a high has to be transmited
                bsf     LININTF, LINTX    ; send a one
                bcf     STATUS, RP0       ; select page 0
                call    DelayFullBit      ; call Delay routine
                decfsz  COUNTER, f        ; decrement counter register
                goto    TransmitNext      ; transmit next bit
                bsf     STATUS, RP0       ; select page 1
                bsf     LININTF, LINTX    ; generate Stop bit
                bcf     STATUS, RP0       ; select page 0
                call    DelayFullBit      ; delay for Stop bit
                retlw   0x00              ; return to main routine
```

```
                ;*******************************************************************************
                ; * ROUTINE             : LIN bus Delay Routinefor Full bit time
                ; * AUTHOR              : Thomas Schmidt
                ; * DATE LAST MODIFIED  : 08/14/2001
                ; * REVISION            : 1.0
                ; * CHANGES             : none
                ; * INPUT PARAMETER     : none
                ; * OUTPUT PARAMETER    : none
                ; * DESCRIPTION         : This routine generates a delay for a full bit time. The
                                          routine is called from the transmit or receive routine.
                ;*******************************************************************************
DelayFullBit    movf     AUTOBAUD_HIGH,w    ; copy content of Autobaud high register into
                btfss    STATUS, Z          ; is high byte = 0?
                goto     LoadHighByte       ; no, high byte is not zero
                goto     DecLowByteOnly     ; decrement only low byte

LoadHighByte    movwf    TEMP2              ; load TEMP2 with content of AUTOBAUD_HIGH
                clrf     TEMP1              ; reset TEMP1 register
DecLowByte1     decfsz   TEMP1, f           ; decrement low byte
                goto     DecLowByte11       ; do until result is zero
                decfsz   TEMP2, f           ; decrement low byte
                goto     DecLowByte1        ; decrement low byte again

DecLowByteOnly  movf     AUTOBAUD_LOW, w    ; copy low byte from autobaud register
                movwf    TEMP1              ; into TEMP1
DecLowByte2     decfsz   TEMP1, f           ; decrement low byte until zero
                goto     DecLowByte22       ; extra two cycle delay
                retlw    0x00               ; return from subroutine
DecLowByte11    nop                         ; stretch time
                goto     DecLowByte1        ; additional two cycle delay
DecLowByte22    nop
                goto     DecLowByte2        ; additional two cycle delay

                ;*******************************************************************************
                ; * ROUTINE           : LIN bus Delay Routine Half Bit time
                ; * AUTHOR            : Thomas Schmidt
                ; * DATE LAST MODIFIED : 08/14/2001
                ; * REVISION          : 1.0
                ; * CHANGES           : none
                ; * INPUT PARAMETER   : none
                ; * OUTPUT PARAMETER  : none
                ; * DESCRIPTION       : This routine generates a delay for a half a bit time.
                                        The routine is called from the receive routine.
                ;*******************************************************************************
DelayHalfBit    movf     AUTOHALF_HIGH,w    ; copy content of Autobaud high register into
                btfss    STATUS, Z          ; is high byte = 0?
                goto     LoadHighByteH      ; no, high byte is not zero
                goto     DecLowByteOnlyH    ; decrement only low byte

LoadHighByteH   movwf    TEMP2              ; load TEMP2 with content of AUTOHALF_HIGH
                clrf     TEMP1              ; reset TEMP1 register
DecLowByteH1    decfsz   TEMP1, f           ; decrement low byte
                goto     DecLowByteH11      ; do until result is zero
                decfsz   TEMP2, f           ; decrement low byte
                goto     DecLowByteH1       ; decrement low byte again

DecLowByteOnlyH movf     AUTOHALF_LOW, w    ; copy low byte from autobaud register
                movwf    TEMP1              ; into TEMP1
DecLowByteH2    decfsz   TEMP1, f           ; decrement low byte until zero
                goto     DecLowByteH22      ; extra two cycle delay
                retlw    0x00               ; return from subroutine
DecLowByteH11   nop
                goto     DecLowByteH1       ; additional two cycle delay
DecLowByteH22   nop
                goto     DecLowByteH2       ; additional two cycle delay
```

```
                ;*******************************************************************************
                ; * ROUTINE              : CRC Check and Generation *
                ; * AUTHOR               : Thomas Schmidt*
                ; * DATE LAST MODIFIED   : 08/14/2001*
                ; * REVISION             : 1.0*
                ; * CHANGES              : none*
                ; * INPUT PARAMETER      : none*
                ; * OUTPUT PARAMETER     : CRC Check: CRC check is appended after last Data Byte
                                           CRC Gen  : CRC is check and CRC_OK or CRC_ERROR is into
                                           the w-register
                ; * DESCRIPTION          : This routine generates or check CRC based on the Mod-
                                           ulo-256 checksum defined in the LIN bus spec. 1.2
                ;*******************************************************************************
CheckCRC        movlw     DATAPOINTER       ; point to first data byte
                movwf     FSR               ;
                swapf     ID_TEMP, w        ; get ID4 and ID5 into bit0 and bit 1
                andlw     0x03              ; get rid of all other bits
                call      DecDataLength     ; get number of data bytes to be transmitted
                clrf      PCLATH            ; reset PCLATH register

                movwf     TEMP1             ; copy number of data bytes into Temp-Counter
                decf    TEMP1, f              ; decrement number of data bytes by one, because
                                            ; TEMP2 register is preloaded

                movf      INDF, w           ; copy first data byte into w-register
                movwf     TEMP2             ; copy first data byte into temp register
NextCalc        incf      FSR, f            ; point to next data memory location
                movf      INDF, w           ; move data into w-register
                addwf     TEMP2, f          ; add data byte to temp and store in temp
                btfsc     STATUS, C         ; add with carry?
                incf      TEMP2, f          ; yes, increment TEMP
                decfsz    TEMP1, f          ; decrement bit counter
                goto      NextCalc          ; calculate next

                call      DecodeIDTable     ; Decode Identifier bits
                clrf      PCLATH            ; Reset PCLATH register

                addwf     PCL, f            ; add mode to low byte of PC
                goto      CRCCheck          ; data was received therefore check CRC
                goto      CRCAppend         ; data is going to be transmited, therefore
                                            ; append CRC

CRCAppend       comf      TEMP2, f          ; complement CRC value
                movf      TEMP2, w          ; copy checksum into w-register
                incf      FSR, f            ; point to location for checksum
                movwf     INDF              ; copy checksum behind
                retlw     0x00              ; return from subroutine

CRCCheck        incf      FSR, f            ; point to CRC byte
                movf      INDF, w           ; copy received CRC value into w-register
                addwf     TEMP2, w          ; add received CRC to calculated CRC
                xorlw     0xff              ; Result should be 0xFF after XOR result is zero
                btfss     STATUS, Z         ; is result zero?
                retlw     CRC_ERROR         ; return with CRC_ERROR
                retlw     CRC_OK
```

```
                ;*******************************************************************************
                ; * ROUTINE             : ID Partiy Bit Generation
                ; * AUTHOR              : Thomas Schmidt
                ; * DATE LAST MODIFIED  : 08/14/2001
                ; * REVISION            : 1.0
                ; * CHANGES             : none
                ; * INPUT PARAMETER     : none
                ; * OUTPUT PARAMETER    : Updated ID with Parity bits in ID_TEMP
                ; * DESCRIPTION         : This routine generated the parity bits P1 and P0 for
                                          the identifier byte according to the LIN bus specifica-
                                          tion 1.2.
                ;*******************************************************************************
CheckParityBits  movf     ID_TEMP, w         ; copy ID value into w-register
                                             ; calculate P0

                 movwf    TEMP1              ; move ID value into TEMP1
                 movwf    TEMP2              ; move ID into TEMP2
                 rrf      TEMP1, f           ; rotate ID_TEMP one to the right (get ID1)
                 movf     TEMP1, w           ; copy TEMP1 to w
                 xorwf    TEMP2, f           ; TEMP2=ID0 XOR ID1
                 rrf      TEMP1, f           ; get ID2
                 movf     TEMP1, w           ; copy ID2 into w-register
                 xorwf    TEMP2, f           ; TEMP2= TEMP2 XOR ID2
                 bcf      STATUS, C          ; clear carry flag
                 rrf      TEMP1, f           ; get ID3 into bit 0
                 rrf      TEMP1, w           ; ID4 into bit 0 and store result in w-register
                 xorwf    TEMP2, f           ; TEMP2 = TEMP2 XOR ID4

                 btfsc    TEMP2, 0           ; test if bit is zero or one
                 goto     CheckP0            ; check if received P0=1
                 btfss    ID_TEMP, 6         ; Check if received P0=0
                 goto     CalcP1             ;
                 retlw    PARITY_ERROR_P0    ; parity error occured

CheckP0          btfss    ID_TEMP, 6         ; check if P0 to 1
                 retlw    PARITY_ERROR_P0    ; P1=0 therefore parity error occrured

                 ; calculate P1
CalcP1           movf     ID_TEMP, w         ; copy ID_TEMP into w-register
                 movwf    TEMP1              ; copy ID_TEMP into TEMP1
                 movwf    TEMP2              ; and TEMP2
                 rrf      TEMP2, f           ; ID1 into bit0
                 rrf      TEMP1, f           ; ID1 into bit0
                 rrf      TEMP1, f           ; ID2 into bit0
                 rrf      TEMP1, f           ; ID3 into bit0
                 movf     TEMP1, w           ; copy TEMP1 into w-register
                 xorwf    TEMP2, f           ; TEMP2 = ID1 XOR ID3
                 rrf      TEMP1, f           ; ID4 into bit0
                 movf     TEMP1, w           ; TEMP1 into w-register
                 xorwf    TEMP2, f           ; TEMP2 = TEMP2 XOR ID4
                 rrf      TEMP1, w           ; ID5 into bit0
                 xorwf    TEMP2, f           ; TEMP2 = TEMP2 XOR ID5
                 comf     TEMP2, f           ; negate TEMP2

                 btfsc    TEMP2, 0           ; check if P1=1
                 goto     CheckP1            ; check if received P1=1
                 btfsc    ID_TEMP, 7         ; check if P1=0
                 retlw    PARITY_ERROR_P1    ; received P1 is not 0 therefore parity error
                 retlw    PARITY_OK          ; received P1 is 0 therefore no parity error

CheckP1          btfss    ID_TEMP, 7         ; set P1
                 retlw    PARITY_ERROR_P1    ; parity error occured
                 retlw    PARITY_OK          ; no parity error occured
```

```
                ;*********************************************************************
                ; Data ID Table. This table is called from the receive routine after
                ; the Identifier byte is received
                ; ********************************************************************
                org      0x200
DecodeIDTable   bsf      PCLATH, 1          ; set PCLATH register
                addwf    PCL, f             ; add to PC
                retlw    RXMODE             ; Receive data from bus
                retlw    RXMODE             ; Receive data from bus
                retlw    RXMODE             ; Receive Data from bus
                retlw    RXMODE             ; Receive Data from bus
                retlw    RXMODE             ; Receive data from bus
                retlw    RXMODE             ; Receive data from bus
                retlw    RXMODE             ; Receive Data from bus
                retlw    RXMODE             ; Receive Data from bus
                retlw    RXMODE             ; Receive data from bus
                retlw    RXMODE             ; Receive data from bus
                retlw    RXMODE             ; Receive Data from bus
                retlw    RXMODE             ; Receive Data from bus
                retlw    RXMODE             ; Receive data from bus
                retlw    RXMODE             ; Receive data from bus
                retlw    RXMODE             ; Receive Data from bus
                retlw    RXMODE             ; Receive Data from bus
                ;****************************************************************************
                ; * ROUTINE             : Decode ID Table
                ; * AUTHOR              : Thomas Schmidt
                ; * DATE LAST MODIFIED  : 08/14/2001
                ; * REVISION            : 1.0
                ; * CHANGES             : Table offset value in w-register
                ; * INPUT PARAMETER     : RXMODE (switch to receive mode) or TXMODE (switch
                                          transmit mode)
                ; * OUTPUT PARAMETER    : Updated ID with Parity bits in ID_TEMP
                ; * DESCRIPTION         : This routine generated the parity bits P1 and P0 for
                                          the identifier byte according to the LIN bus specifica-
                                          tion 1.2.
                ;****************************************************************************
DecodeAction    bsf      PCLATH, 1          ;
                addwf    PCL, f             ; add to PC
                retlw    IGNORE             ; Lock Doors
                retlw    IGNORE             ; Unlock Doors
                retlw    INCREMENT          ; Windows Up
                retlw    DECREMENT          ; Windows Down
                retlw    IGNORE             ; Mirror Left
                retlw    IGNORE             ; Mirror Right
                retlw    IGNORE             ; Mirror Down
                retlw    IGNORE             ; Mirror Up
                retlw    IGNORE             ; Tilt Seat Back
                retlw    IGNORE             ; Tilt Seat Forward
                retlw    IGNORE             ; Slide Seat Forward
                retlw    IGNORE             ; Slide Seat Backward
                retlw    IGNORE             ; Front Edge Set Up
                retlw    IGNORE             ; Front Edge Set Down
                retlw    IGNORE             ; Back Edge  Seat Up
                retlw    IGNORE             ; Back Edge Seat Down
                ; ********************************************************************
                ; Data length Table. This table is called from the receive routine after
                ; the Identifier byte is received
                ; ********************************************************************
DecDataLength   bsf      PCLATH, 1          ; Set PCLATH register
                addwf    PCL, f             ; add to PC
                retlw    0x02               ; data length is 2
                retlw    0x02               ; data length is 2
                retlw    0x04               ; data length is 4
                retlw    0x08               ; data length is 8

                END
```

NOTES:

Appendix D. LIN bus Slave for the PIC16C432 (slave2.asm)

D.1 LIN bus SLAVE FOR THE PIC16C432 (SLAVE2.ASM)


```
; *************************************************************************************************
; * Title       : LIN bus Slave for the PIC16C433 (slave2.asm)
; * Author      : Thomas Schmidt
; * Date        : 6/26/2001
; * Revision    : 1.0
; * Description :
; *************************************************************************************************

               LIST P=16C433, r=hex

               ; *********************************************************************
                         ; * Include files
               ; *********************************************************************
                         #include <P16C433.INC>

               ; *********************************************************************
               ; * Pin definitions
               ; *********************************************************************


               ; *********************************************************************
               ; * Register definitions
               ; *********************************************************************
               cblock 0x20
                     AUTOBAUD_LOW            ; low byte of bit-time coutner
                     AUTOBAUD_HIGH           ; high byte of bit-time counter
                     AUTOHALF_LOW            ; low byte of half the bit time
                     AUTOHALF_HIGH           ; high byte for half the bit time
                     TEMP1, TEMP2            ; temporary registers
                     RXTX_REG                ; receive register
                     MESSAGE_COUNTER         ; Counts number of bytes to receive or transmit
                     COUNTER                 ; receive & Transmit counter register
                     ID_TEMP                 ; Register for ID byte
                     endc
```

```
                ; *********************************************************************
                ; * Other definitions
                ; *********************************************************************
                #define BITS         0x08      ; number of bits to receive
                #define RXMODE       0x00      ; Receive mode
                #define TXMODE       0x01      ; Transmit mode
                #define LISTENMODE   0x02      ; Listen only mode
                #define CRC_ERROR    0x01      ; CRC_ERROR occurred
                #define CRC_OK       0x00      ; No CRC_ERROR occured
                #define PARITY_OK    0x00      ; No Parity error occrured
                #define PARITY_ERROR_P00x01    ; Parity error on P0 occured
                #define PARITY_ERROR_P10x02    ; Parity error on P1 occured
                #define DATAPOINTER  0x35      ; address of first data byte
                #define IGNORE       0x00      ; ignore received command
                #define INCREMENT    0x01      ; increment PORTB
                #define DECREMENT    0x02      ; decrement PORTB



                ; *********************************************************************
                ; * Fuse configuration
                ; *********************************************************************
                __CONFIG _CP_OFF&_XT_OSC&_PWRTE_ON&_MCLRE_ON&_WDT_OFF

                ; ********************************************************************************
                ; * Reset vector
                ; ********************************************************************************
                ORG 0x00
                goto MainRoutine

                ; ********************************************************************************
                ; * ROUTINE             : MainRoutine
                ; * AUTHOR              : Thomas Schmidt
                ; * DATE LAST MODIFIED  : 08/14/2001
                ; * REVISION            : 1.0
                ; * CHANGES             : none
                ; * INPUT PARAMETER     : none
                ; * OUTPUT PARAMETER    : none
                ; * DESCRIPTION         : Main routine. The main routine detects first the trans-
                                          mission time of the incomming calibration character.
                                          After that the routine receives and transmits incomming
                                          characters.
                ;*********************************************************************************
                ORG 0x06
MainRoutine     call        StartUp            ; Initialize Registers

TestSynchByte   btfsc       GPIO, LINRX        ; Check for synch byte
                goto        TestSynchByte      ; Synch byte not received
TestEndSynch    btfss       GPIO, LINRX        ; check for end of synchbyte
                goto        TestEndSynch       ; end of synchronization byte not received

                ; Receive message on LIN bus
                call        LinHandler         ; call LIN bus handler
                    ; Decode message and take action upon receive message Identifier
                            ; Decode message and take action upon receive message Identifier
                    movf    ID_TEMP, w           ; copy ID_TEMP into w-register
                    andlw   0x0f                 ; decode only lower four nibbles
                    call    DecodeAction         ; see if LED has to be turn on
                    clrf    PCLATH               ; reset PCLATH register
                    addwf   PCL, f               ; add to PC
                    goto    TestSynchByte        ; Ignore all other IDs
                    goto    WindowUp             ; Window up function
                    goto    WindowDown           ; Window down function
```

```
                ; WindowUp command was receive from Master
WindowUp        incf   GPIO, f                  ; increment GPIO register
                goto   TestSynchByte            ; Go back

                ; WindowDown command was receive from Master
WindowDown      decf   GPIO, f                  ; decrement GPIO register
                goto   TestSynchByte            ; go back

                ;*****************************************************************************
                ; * ROUTINE            : StartUp
                ; * AUTHOR             : Thomas Schmidt
                ; * DATE LAST MODIFIED : 08/14/2001
                ; * REVISION           : 1.0
                ; * CHANGES            : none
                ; * INPUT PARAMETER    : none
                ; * OUTPUT PARAMETER   : none
                ; * DESCRIPTION        : This routine is called after power-up. PORTS and
                                         peripherals are initialized.
                ;*****************************************************************************
StartUp         movlw   0xC0                    ; set all latches of GPIO except
                movwf   GPIO                    ; LINTX & LINRX to zero
                bsf     STATUS,RP0              ; select page 1
                movlw   0x07                    ; Make all pins on GPIO
                movwf   ADCON1                  ; digitial I/Os.
                movlw   0xf8                    ; make GP0, GP1 and GP2 outputs
                movwf   TRISIO                  ; initialize TRISIO register
                bcf     STATUS,RP0              ; select page 0

                ; Initialize Data RAM for LIN bus Communication
InitData        movlw   DATAPOINTER             ; point to first data location
                movwf   FSR                     ; and initialize FSR register
                movlw   0x0f                    ; Initialize 16 register
                movwf   TEMP1                   ; temporary counter register
Count           movlw   0xaa                    ; init value is 0xaa
                movwf   INDF                    ; copy into location where FSR points to
                incf    FSR, f                  ; point to next location
                decfsz  TEMP1, f                ; decrement temporary counter
                goto    Count                   ; counter is not zero, therefore keep on
                                                ; initializing
                retlw   0x00                    ; return to main routine
```

```
                ;*************************************************************************
                ; * ROUTINE              : LIN bus Transmission/Reception Routine
                ; * AUTHOR               : Thomas Schmidt
                ; * DATE LAST MODIFIED   : 08/14/2001
                ; * REVISION             : 1.0
                ; * CHANGES              : none
            ; * INPUT PARAMETER      : none
            ; * OUTPUT PARAMETER     : none
            ; * DESCRIPTION          : This routine sends out a LIN bus data frame and/or
                                       receives or transmits data. The reception or trans. of
                                       data is defined in the ID itself. After the ID-Byte is
                                       sent it is decoded in a look up table if data should be
                                       transmitted or received. Depending on the result either
                                       a transmit or receive mode is activated.
                                       ID_TEMP =holds the ID to send (parity is already gener-
                                               ated
                                       DATAPOINTER =points to first data byte. Last bytes CRC
                                                    byte for transmission otherwise last byte
                                                    is data byte
            ; *************************************************************************
LinHandler  clrf    AUTOBAUD_LOW        ; reset register
            clrf    AUTOBAUD_HIGH       ; reset register
            clrf    AUTOHALF_LOW        ; reset register
            clrf    AUTOHALF_HIGH       ; reset register
            clrf    COUNTER             ; reset counter
            clrf    MESSAGE_COUNTER     ; reset message counter register

WaitStartBit btfsc  GPIO, LINRX         ; was start bit there
            goto    WaitStartBit        ; no, keep on looking

            ; yes, measure first low bit time
TestClear   btfss   GPIO, LINRX         ; was there a transition from low to high?
            goto    AutoMeasureSet      ; no, increment Autobaud counter registers

            incf    COUNTER,f           ; increment counter register
            goto    TestSet             ; yes, therefore increment bit counter and measure
                                        ; high time

            ; Autobaud counter. The signal is sampled every 6 Instruction cycles
            ; This means the number of countes equals 8*Tbit/6Tcy. Example:
            ; Transmission rate 19.2Kbaud, Fosc=4MHz => Tbit=51us, 8*51us/6*1us = 68 counts.
            ; Counting the Transitions takes 9Tcy. This value has to be added to the
            ; number of counts. Example from above: 68 Couts + 5*9Tcy (5 because five
            ; transitions are counted.
AutoMeasureSet incfsz AUTOBAUD_LOW, f   ; increment Autobaud low register
            goto    TestClear           ; keep on testing
            incf    AUTOBAUD_HIGH,f     ; increment high byte of autobaud register
            goto    TestClear           ; keep on testing

AutoMeasureClr incfsz AUTOBAUD_LOW, f   ; increment Autobaud low register
            goto    TestSet             ; keep on testing
            incf    AUTOBAUD_HIGH,f     ; increment high byte of autobaud register
            goto    TestSet             ; keep on testing

TestSet     btfsc   GPIO, LINRX         ; check if pin is still high
            goto    AutoMeasureClr      ; no, there was not, measure time
            incf    COUNTER, f          ; increment counter register
            movf    COUNTER, w          ; check if all 8 bits were received
            xorlw   0x08                ;
            btfss   STATUS, Z           ; is result zero?
            goto    TestClear           ; No, therefore keep on measuring
```

```
                    ; It takes 6Tcy to process the counter register. This is executed
                    ; eight times => 8*6Tcy=48Tcy where the low byte is not update according
                    ; to 6 countes. Therefore 48Tcy/6countes= 8. Eight countes have
                    ; to be added to the low byte of the auto_baud_low register
                    ; add one count to low byte, because auf LIN bus propagation time
                    ; 4us to drive LIN bus high and 2V/us raise time. Therefore we have
                    ; 10us delay before a slave sees a high value. 10us is two counts
                    ; Therefore the total counts to be added are 10 counts (=0x0a)
                    movlw   0x09
                    addwf   AUTOBAUD_LOW, f       ; adjust low byte

                    ; Calculation of transmission time for one bit
Divide              bcf   STATUS, C               ; clear carry bit
                    rrf   AUTOBAUD_HIGH,f         ; rotate autobaud high register
                    rrf   AUTOBAUD_LOW, f         ; rotate autobaud low register
                    bcf   STATUS, C               ; clear carry bit
                    rrf   AUTOBAUD_HIGH,f         ; rotate autobaud high register
                    rrf   AUTOBAUD_LOW, f         ; rotate autobaud low register
                    bcf   STATUS, C               ; clear carry bit
                    rrf   AUTOBAUD_HIGH,f         ; rotate autobaud high register
                    rrf   AUTOBAUD_LOW, f         ; rotate autobaud low register

                    ; Calculate the transmission time for half the bit time (means
                    ; divide transmission time of one bit by two).
CalcHalfBit         bcf     STATUS, C             ; clear carry bit
                    rrf     AUTOBAUD_HIGH,w       ; rotate autobaud high register
                    movwf   AUTOHALF_HIGH         ; copy result into AUTOHALF_HIGH register
                    rrf     AUTOBAUD_LOW, w       ; rotate autobaud high register
                    movwf   AUTOHALF_LOW          ; copy result into AUTOHALF_LOW register

                    ; Adjust 16-bit counter for receive and transmit routine. This means
                    ; that the overhead of instruction cycles in of the receive/transmit
                    ; routine has to be substracted from the transmission time of one bit
                    ; and half a bit.
AdjustLowByte       movlw   0x2                   ; 18-19 instruction cycles overhead from
                                                  ; transmit/receive routine. This overhead
                                                  ; must be substracted from iterations
                    subwf   AUTOBAUD_LOW, f       ; adjust low byte from Autobaud counter
                    movlw   0x02                  ; substract 2 from low byte of half the bit time
                    subwf   AUTOHALF_LOW, f       ; substract from low byte of half the bit time

                    ; wait until Stop-bit comes in. Otherwise the reception of the next
                    ; bit will start to early.

Wait4Stop           btfss   GPIO, LINRX
                    goto    Wait4Stop

                    ; receive Identifier byte

                    call    Receive               ; Receive Identifier byte
                                                  ; store Identifier byte
                    movf    RXTX_REG, w           ; copy Identifier byte into w-register
                    movwf   ID_TEMP               ; copy Identifier into ID_TEMP register
                    call    CheckParityBits       ; check if parity bits are correct

                    ; Decode ID4 and ID5. These two bits indicate how many bytes of
                    ; data have to be transmitted or received.
                    swapf   RXTX_REG, w           ; change ID4 and ID5 to lower nibble
                    andlw   0x03                  ; delete rest
                call    DecDataLength         ; decode data length
                clrf    PCLATH                ; reset PCLATH register
                movwf   MESSAGE_COUNTER       ; store length of message into MESSAGE_LENGTH
                                              ; register
                incf    MESSAGE_COUNTER,f     ; increment message counter by one for
                                              ; receiving or transmitting CRC byte
```

```
                ; Decode Identifier byte.
                movf    RXTX_REG, w           ; load RXTX_REG into w-register (Identifierbyte
                                              ; into
                                              ; w-register)
                andlw   0x0f                  ; Delete upper four bits of Identifier byte (these
                                              ; bit include parity bits and data lenght code
                                              ; Lower
                                              ; bit are the indifier bits.

                call    DecodeIDTable         ; Decode Identifier bits
                clrf    PCLATH                ; reset PCLATH register

                addwf   PCL, f                ; add mode to low byte of PC
                goto    ReceiveMode           ; receive data
                goto    TransmitMode          ; transmit data
                goto    ReceiveMode           ; Listen to the bus

TransmitMode    call    CheckCRC              ; generated CRC
                movlw   DATAPOINTER           ; point to first data byte
                movwf   FSR                   ; initialize FSR register

TransmitNextD   movf    INDF, w               ; copy data byte into w-register
                movwf   RXTX_REG              ; copy data in RXTX_REG

                call    Transmit              ; transmit data
                incf    FSR, f                ; point to next location
                decfsz  MESSAGE_COUNTER, f    ; decrement Message Counter by one
                goto    TransmitNextD         ; transmit next data
                retlw   0x00                  ; return to main

                ; Receive Mode in this sequence data is received
ReceiveMode     movlw     DATAPOINTER         ; point to data location
                movwf     FSR                 ; where data should be stored
ReceiveNextData call      Receive             ; receive next data
                movf      RXTX_REG, w         ; copy data into w-register
                movwf     INDF                ; copy data into data area
                incf      FSR, f              ; point to next location
                decfsz    MESSAGE_COUNTER, f  ; decrement number of bytes to receive by one
                goto      ReceiveNextData     ; receive next data byte

                call      CheckParityBits     ; check if parity bits are correct
                call      CheckCRC            ; check if checksum is correct
                retlw     0x00                ; return to main
```

```
                ; ************************************************************************
                ; * ROUTINE             : LIN bus Receive Routine
                ; * AUTHOR              : Thomas Schmidt
                ; * DATE LAST MODIFIED  : 08/14/2001
                ; * REVISION            : 1.0
                ; * CHANGES             : none
                ; * INPUT PARAMETER     : none
                ; * OUTPUT PARAMETER    : none
                ; * DESCRIPTION         : This routine receives an 8-bit value via the LIN bus
                   ;************************************************************************
Receive            clrf    RXTX_REG              ; clear receive register
                   movlw   BITS                  ; number of bits to receive
                   movwf   COUNTER               ; load number of bits into counter register


ReceiveStartBit    btfsc   GPIO, LINRX           ; test for falling edge
                   goto    ReceiveStartBit       ; start-bit not found
                   call    DelayHalfBit          ; wait until middle of start-bit
                   call    DelayFullBit          ; ignore start-bit and sample first
                                                 ; data bit in the middle of the bit

ReceiveNext        btfsc   GPIO, LINRX           ; is LINRX zero or a one?
                   bsf     STATUS,C              ; bit is a one => set carry bit
                   btfss   GPIO, LINRX           ; is LINRX one or a zero?
                   bcf     STATUS,C              ; LINRX is zero => clear carry bit
                   rrf     RXTX_REG, f           ; rotate value into receive register
                   call    DelayFullBit          ; call Delay routine
                   decfsz  COUNTER, f            ; decrement receive count register by one
                   goto    ReceiveNext           ; receive next bit

                   retlw   0x00                  ; return


                   ;************************************************************************
                   ; * ROUTINE              : LIN bus Transmit Routine
                   ; * AUTHOR               : Thomas Schmidt
                   ; * DATE LAST MODIFIED   : 08/14/2001
                   ; * REVISION             : 1.0
                   ; * CHANGES              : none
                   ; * INPUT PARAMETER      : none
                   ; * OUTPUT PARAMETER     : none
                   ; * DESCRIPTION          : This routine transmits an 8-bit value via the LIN
bus
                   ;************************************************************************
Transmit           movlw   BITS                  ; number of bit's to transmit
                   movwf   COUNTER               ; initialize count register
                   bcf     GPIO, LINTX           ; generate start-bit
                   call    DelayFullBit          ; generate Delay for one bit-time
TransmitNext       rrf     RXTX_REG, f           ; rotate receive register
                   btfss   STATUS, C             ; test bit to be transmited
                   bcf     GPIO, LINTX           ; Send a zero
                   btfsc   STATUS, C             ; Check if a high has to be transmited
                   bsf     GPIO, LINTX           ; send a one
                   call    DelayFullBit          ; call Delay routine
                   decfsz  COUNTER, f            ; decrement counter register
                   goto    TransmitNext          ; transmit next bit
                   bsf     GPIO, LINTX           ; generate Stop bit
                   call    DelayFullBit          ; delay for Stop bit
                   retlw   0x00                  ; return to main routine
```

```
                ;*************************************************************************
                ; * ROUTINE              : LIN bus Delay Routinefor Full bit time
                ; * AUTHOR               : Thomas Schmidt
                ; * DATE LAST MODIFIED   : 08/14/2001
                ; * REVISION             : 1.0
                ; * CHANGES              : none
                ; * INPUT PARAMETER      : none
                ; * OUTPUT PARAMETER     : none
            ; * DESCRIPTION          : This routine generates a delay for a full bit time.
            ; *                        The routine is called from the transmit or receive
            ; *                        routine.
            ; *************************************************************************
DelayFullBit    movf      AUTOBAUD_HIGH,w     ; copy content of Autobaud high register into
                btfss     STATUS, Z           ; is high byte = 0?
                goto      LoadHighByte        ; no, high byte is not zero
                goto      DecLowByteOnly      ; decrement only low byte

LoadHighByte    movwf     TEMP2               ; load TEMP2 with content of AUTOBAUD_HIGH
                clrf      TEMP1               ; reset TEMP1 register
DecLowByte1     decfsz    TEMP1, f            ; decrement low byte
                goto      DecLowByte11        ; do until result is zero
                decfsz    TEMP2, f            ; decrement low byte
                goto      DecLowByte1         ; decrement low byte again

DecLowByteOnly  movf      AUTOBAUD_LOW, w     ; copy low byte from autobaud register
                movwf     TEMP1               ; into TEMP1
DecLowByte2     decfsz    TEMP1, f            ; decrement low byte until zero
                goto      DecLowByte22        ; extra two cycle delay
                retlw     0x00                ; return from subroutine
DecLowByte11    nop                           ; stretch time
                goto      DecLowByte1         ; additional two cycle delay
DecLowByte22    nop
                goto      DecLowByte2         ; additional two cycle delay

                ; *************************************************************************
                ; * ROUTINE              : LIN bus Delay RoutineHalf Bit time
                ; * AUTHOR               : Thomas Schmidt
                ; * DATE LAST MODIFIED   : 08/14/2001
                ; * REVISION             : 1.0
                ; * CHANGES              : none
                ; * INPUT PARAMETER      : none
                ; * OUTPUT PARAMETER     : none
                ; * DESCRIPTION          : This routine generates a delay for a half a bit time.
                                          The routine is called from the receive routine.
                ; *************************************************************************
DelayHalfBit    movf      AUTOHALF_HIGH,w     ; copy content of Autobaud high register into
                btfss     STATUS, Z           ; is high byte = 0?
                goto      LoadHighByteH       ; no, high byte is not zero
                goto      DecLowByteOnlyH     ; decrement only low byte

LoadHighByteH   movwf     TEMP2               ; load TEMP2 with content of AUTOHALF_HIGH
                clrf      TEMP1               ; reset TEMP1 register
DecLowByteH1    decfsz    TEMP1, f            ; decrement low byte
                goto      DecLowByteH11       ; do until result is zero
                decfsz    TEMP2, f            ; decrement low byte
                goto      DecLowByteH1        ; decrement low byte again

DecLowByteOnlyH movf      AUTOHALF_LOW, w     ; copy low byte from autobaud register
                movwf     TEMP1               ; into TEMP1
DecLowByteH2    decfsz    TEMP1, f            ; decrement low byte until zero
                goto      DecLowByteH22       ; extra two cycle delay
                retlw     0x00                ; return from subroutine
DecLowByteH11   nop
                goto      DecLowByteH1        ; additional two cycle delay
DecLowByteH22   nop
                goto      DecLowByteH2        ; additional two cycle delay
```

```
                ;*************************************************************************
                ; * ROUTINE             : CRC Check and Generation
                ; * AUTHOR              : Thomas Schmidt
                ; * DATE LAST MODIFIED  : 08/14/2001
                ; * REVISION            : 1.0
                ; * CHANGES             : none
                ; * INPUT PARAMETER     : none
                ; * OUTPUT PARAMETER    : CRC Check: CRC check is appended after last Data Byte
                                          CRC Gen  : CRC is check and CRC_OK or CRC_ERROR is
                ; *                                  into the w-register
                ; * DESCRIPTION         : This routine generates or check CRC based on the
                ; *                       Modulo-256 checksum defined in the LIN bus spec. 1.2
                ; *
                ; *************************************************************************
CheckCRC        movlw   DATAPOINTER         ; point to first data byte
                movwf   FSR                 ;
                swapf   ID_TEMP, w          ; get ID4 and ID5 into bit0 and bit 1
                andlw   0x03                ; get rid of all other bits
                call    DecDataLength       ; get number of data bytes to be transmitted
                clrf    PCLATH              ; reset PCLATH register

                movwf   TEMP1               ; copy number of data bytes into Temp-Counter
                decf    TEMP1, f            ; decrement number of data bytes by one,
because                                     ; TEMP2 register is preloaded

                movf    INDF, w             ; copy first data byte into w-register
                movwf   TEMP2               ; copy first data byte into temp register
NextCalc        incf    FSR, f              ; point to next data memory location
                movf    INDF, w             ; move data into w-register
                addwf   TEMP2, f            ; add data byte to temp and store in temp
                btfsc   STATUS, C           ; add with carry?
                incf    TEMP2, f            ; yes, increment TEMP
                decfsz  TEMP1, f            ; decrement bit counter
                goto    NextCalc            ; calculate next

                call    DecodeIDTable       ; Decode Identifier bits
                clrf    PCLATH              ; reset PCLATH register

                addwf   PCL, f              ; add mode to low byte of PC
                goto    CRCCheck            ; data was received therefore check CRC
                goto    CRCAppend           ; data is going to be transmited, therefore
                                            ; append CRC

CRCAppend       comf    TEMP2, f            ; complement CRC value
                movf    TEMP2, w            ; copy checksum into w-register
                incf    FSR, f              ; point to location for checksum
                movwf   INDF                ; copy checksum behind
                return                      ; return from subroutine

CRCCheck        incf    FSR, f              ; point to CRC byte
                movf    INDF, w             ; copy received CRC value into w-register
                addwf   TEMP2, w            ; add received CRC to calculated CRC
                xorlw   0xff                ; Result should be 0xFF after XOR result is
zero
                btfss   STATUS, Z           ; is result zero?
                retlw   CRC_ERROR           ; return with CRC_ERROR
                retlw   CRC_OK
```

```
                ; ***************************************************************************
                ; * ROUTINE                    : ID Partiy Bit Generation
                ; * AUTHOR                     : Thomas Schmidt
                ; * DATE LAST MODIFIED         : 08/14/2001
                ; * REVISION                   : 1.0
                ; * CHANGES                    : none
                ; * INPUT PARAMETER            : none
                ; * OUTPUT PARAMETER           : Updated ID with Parity bits in ID_TEMP
                ; * DESCRIPTION                : This routine generated the parity bits P1 and P0
                ; *                              for the identifier byte according to the LIN bus
                ; *                              specification 1.2.
                ; *
                ; ***************************************************************************
CheckParityBits movf    ID_TEMP, w            ; copy ID value into w-register
                ; calculate P0

                movwf   TEMP1                 ; move ID value into TEMP1
                movwf   TEMP2                 ; move ID into TEMP2
                rrf     TEMP1, f              ; rotate ID_TEMP one to the right (get ID1)
                movf    TEMP1, w              ; copy TEMP1 to w
                xorwf   TEMP2, f              ; TEMP2=ID0 XOR ID1
                rrf     TEMP1, f              ; get ID2
                movf    TEMP1, w              ; copy ID2 into w-register
                xorwf   TEMP2, f              ; TEMP2= TEMP2 XOR ID2
                bcf     STATUS, C             ; clear carry flag
                rrf     TEMP1, f              ; get ID3 into bit 0
                rrf     TEMP1, w              ; ID4 into bit 0 and store result in w-register
                xorwf   TEMP2, f              ; TEMP2 = TEMP2 XOR ID4

                btfsc   TEMP2, 0              ; test if bit is zero or one
                goto    CheckP0               ; check if received P0=1
                btfss   ID_TEMP, 6            ; Check if received P0=0
                goto    CalcP1                ;
                retlw   PARITY_ERROR_P0       ; parity error occured

CheckP0         btfss   ID_TEMP, 6            ; check if P0 to 1
                retlw   PARITY_ERROR_P0       ; P1=0 therefore parity error occrured

                ; calculate P1
CalcP1          movf    ID_TEMP, w            ; copy ID_TEMP into w-register
                movwf   TEMP1                 ; copy ID_TEMP into TEMP1
                movwf   TEMP2                 ; and TEMP2
                rrf     TEMP2, f              ; ID1 into bit0
                rrf     TEMP1, f              ; ID1 into bit0
                rrf     TEMP1, f              ; ID2 into bit0
                rrf     TEMP1, f              ; ID3 into bit0
                movf    TEMP1, w              ; copy TEMP1 into w-register
                xorwf   TEMP2, f              ; TEMP2 = ID1 XOR ID3
                rrf     TEMP1, f              ; ID4 into bit0
                movf    TEMP1, w              ; TEMP1 into w-register
                xorwf   TEMP2, f              ; TEMP2 = TEMP2 XOR ID4
                rrf     TEMP1, w              ; ID5 into bit0
                xorwf   TEMP2, f              ; TEMP2 = TEMP2 XOR ID5
                comf    TEMP2, f              ; negate TEMP2

                btfsc   TEMP2, 0              ; check if P1=1
                goto    CheckP1               ; check if received P1=1
                btfsc   ID_TEMP, 7            ; check if P1=0
                retlw   PARITY_ERROR_P1       ; received P1 is not 0 therefore parity error
                    retlw   PARITY_OK           ; received P1 is 0 therefore no parity error

CheckP1             btfss   ID_TEMP, 7          ; set P1
                    retlw   PARITY_ERROR_P1     ; parity error occured
                    retlw   PARITY_OK           ; no parity error occured
```

```
                ; ***********************************************************************
                ; Data length Table. This table is called from the receive routine after
                ; the Identifier byte is received
                ; ***********************************************************************
                org     0x200
DecDataLength   bsf     PCLATH, 1           ; Set PCLATH register
                addwf   pcl, f              ; add to pc
                retlw   0x02                ; data length is 2
                retlw   0x02                ; data length is 2
                retlw   0x04                ; data length is 4
                retlw   0x08                ; data length is 8


                ; ***********************************************************************
                ; Data ID Table. This table is called from the receive routine after
                ; the Identifier byte is received
                ; ***********************************************************************

DecodeIDTable   bsf     PCLATH, 1           ; set PCLATH register
                addwf   PCL, f              ; add to PC
                retlw   RXMODE              ; Receive data from bus
                retlw   RXMODE              ; Receive data from bus
                retlw   RXMODE              ; Receive Data from bus
                retlw   RXMODE              ; Receive Data from bus
                retlw   RXMODE              ; Receive data from bus
                retlw   RXMODE              ; Receive data from bus
                retlw   RXMODE              ; Receive Data from bus
                retlw   RXMODE              ; Receive Data from bus
                retlw   RXMODE              ; Receive data from bus
                retlw   RXMODE              ; Receive data from bus
                retlw   RXMODE              ; Receive Data from bus
                retlw   RXMODE              ; Receive Data from bus
                retlw   RXMODE              ; Receive data from bus
                retlw   RXMODE              ; Receive data from bus
                retlw   RXMODE              ; Receive Data from bus
                retlw   RXMODE              ; Receive Data from bus
```

```
                ;*************************************************************************
                ; * ROUTINE            : Decode ID Table
                ; * AUTHOR             : Thomas Schmidt
                ; * DATE LAST MODIFIED : 08/14/2001
                ; * REVISION           : 1.0
                ; * CHANGES            : Table offset value in w-register
                ; * INPUT PARAMETER    : RXMODE (switch to receive mode) or TXMODE (switch
                ; *                      transmit mode)
                ; * OUTPUT PARAMETER   : Updated ID with Parity bits in ID_TEMP
                ; * DESCRIPTION        : This routine generated the parity bits P1 and P0
                ; *                      for the identifier byte according to the LIN bus
                ; *                      specification 1.2.
                ; *************************************************************************
DecodeAction    bsf       PCLATH, 1          ; Set PCLATH register
                addwf     PCL, f             ; add to PC
                retlw     IGNORE             ; Lock Doors
                retlw     IGNORE             ; Unlock Doors
                retlw     INCREMENT          ; Windows Up
                retlw     DECREMENT          ; Windows Down
                retlw     IGNORE             ; Mirror Left
                retlw     IGNORE             ; Mirror Right
                retlw     IGNORE             ; Mirror Down
                retlw     IGNORE             ; Mirror Up
                retlw     IGNORE             ; Tilt Seat Back
                retlw     IGNORE             ; Tilt Seat Forward
                retlw     IGNORE             ; Slide Seat Forward
                retlw     IGNORE             ; Slide Seat Backward
                retlw     IGNORE             ; Front Edge Set Up
                retlw     IGNORE             ; Front Edge Set Down
                retlw     IGNORE             ; Back Edge  Seat Up
                retlw     IGNORE             ; Back Edge Seat Down


                END
```

Index

NOTES:

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:
480-792-7627
Web Address:
www.microchip.com

Atlanta
Alpharetta, GA
Tel: 770-640-0034
Fax: 770-640-0307

Boston
Westford, MA
Tel: 978-692-3848
Fax: 978-692-3821

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

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

San Jose
Mountain View, CA
Tel: 650-215-1444
Fax: 650-961-0286

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

ASIA/PACIFIC

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: 86-28-8676-6200
Fax: 86-28-8676-6599

China - Fuzhou
Tel: 86-591-750-3506
Fax: 86-591-750-3521

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

China - Shanghai
Tel: 86-21-6275-5700
Fax: 86-21-6275-5060

China - Shenzhen
Tel: 86-755-8290-1380
Fax: 86-755-8295-1393

China - Shunde
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

China - Qingdao
Tel: 86-532-502-7355
Fax: 86-532-502-7205

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-2229-0061
Fax: 91-80-2229-0062

India - New Delhi
Tel: 91-11-5160-8632
Fax: 91-11-5160-8632

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

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

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

Taiwan - Kaohsiung
Tel: 886-7-536-4816
Fax: 886-7-536-4817

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

Taiwan - Hsinchu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

EUROPE

Austria - Weis
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark - Ballerup
Tel: 45-4420-9895
Fax: 45-4420-9910

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

Germany - Ismaning
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

England - Berkshire
Tel: 44-118-921-5869
Fax: 44-118-921-5820

08/24/04