DRM003/D

Parallel or Serial CAN Connection Using External CAN Standalone CAN Controllers with the Motorola M68HC908 Embedded FLASH Microcontrollers

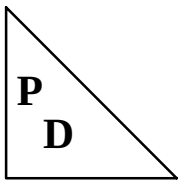
Designer Reference Manual

Digital DNA™ from Motorola

Parallel or Serial CAN Connection Using External CAN Standalone CAN Controllers with the Motorola M68HC908 Embedded FLASH Microcontrollers

By: Peter Dilger
Ing.-Buro Dilger
Kohlgasse 20
D-77743 Neuried
Germany

Telephone: +49 (0) 7807 955 432
Fax: +49 (0) 7807 955 432
Email: ib.dilger@t-online.de
Web: www.ib-dilger.de



Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

List of Sections

Section 1. General Description 17

Section 2. Overview of Embedded FLASH MCUs Used 23

Section 3. Printed Circuit Board Structure 37

Section 4. Pin Assignments and Jumper Settings 41

Section 5. Jumper Description 45

Section 6. Working with the Reference Design 55

Section 7. Circuit Description of the Monitor Mode Area 61

Section 8. Software Description 63

Appendix A. Bill of Materials and Schematic 73

Appendix B. Flowcharts and Source Code Files 87

Appendix C. Motorola Embedded CAN MCU Selector Guide. 153

Appendix D. MC33388 Motorola CAN Interface 155

List of Sections

Table of Contents

Section 1. General Description

1.1 Contents 17
1.2 Introduction 17
1.3 Reference Design Overview 20
1.3.1 Option 1: MC68HC908GR8 + MCP2510 + MC33388 20
1.3.2 Option 2: MC68HC908GP32 + MCP2510 + MC33388 21
1.3.3 Option 3: MC68HC908GP32 + SJA1000 + PCA82C250 . . 21

Section 2. Overview of Embedded FLASH MCUs Used

2.1 Contents 23
2.2 Introduction 23
2.3 MC68HC908GR8 24
2.3.1 MC68HC908GR8 Application Notes and Engineering Bulletins 28
2.3.2 MC68HC908GR8 Development Tool Kits 28
2.3.3 MC68HC908GR8 Individual Development Tool Components 29
2.3.4 MC68HC908GR8 Package Options 29
2.3.5 MC68HC908GR8 Sample Packs 29
2.4 MC68HC908GP32 30
2.4.1 MC68HC908GP32 Application Notes and Engineering Bulletins 34
2.4.2 MC68HC908GP32 Development Tool Kits 34
2.4.3 MC68HC908GP32 Individual Development Tool Components 35
2.4.4 MC68HC908GP32 Package Options 35
2.4.5 MC68HC908GP32 Sample Packs 35

Section 3. Printed Circuit Board Structure

3.1 Contents 37
3.2 Introduction 37
3.3 Monitor Mode Interface (Area 1) 38
3.4 Controller (Area 2) 39
3.5 Application (Area 3) 39

Section 4. Pin Assignments and Jumper Settings

4.1 Contents 41
4.2 Introduction 41
4.3 Supply Voltage X3 42
4.4 Monitor Mode X2 42
4.5 SCI X1 42
4.6 CAN1 X5 43
4.7 CAN2 X4 43
4.8 Motor SV5 44

Section 5. Jumper Description

5.1 Contents 45
5.2 Introduction 45
5.3 Jumpers in the Monitor Mode Area 45
5.4 Jumpers in the Controller Area 47
5.5 Jumpers in the Application Area 49
5.6 Motor Driver L6202 53

Section 6. Working with the Reference Design

6.1 Contents 55
6.2 Introduction 55
6.3 Connecting the CPU to the CAN Controller 55

Section 7. Circuit Description of the Monitor Mode Area

Section 8. Software Description

8.1 Contents 63
8.2 Introduction 63
8.3 CodeWarrior IDE 64
8.4 Module Description 65
8.4.1 MCU_Select.h 65
8.4.2 CAN Driver.C. 68
8.4.3 MCU_Hardware_Description_HC08.h 69
8.4.4 MCU_Assembler_Instruction.h 69
8.4.5 Hardware_HC08.c. 69
8.4.6 IRQtab.c 69
8.4.7 CAN_Controller.h 70
8.4.8 CAN_Controller_SJA1000.C 70
8.4.9 CAN_Controller_MCP2510.C 71
8.4.10 cptest.C 71

Appendix A. Bill of Materials and Schematic

Appendix B. Flowcharts and Source Code Files

B.1 Contents 87
B.2 Introduction 87
B.3 Flowcharts 88
B.4 Source Code Files 91

Appendix C. Motorola Embedded CAN MCU Selector Guide

Appendix D. MC33388 Motorola CAN Interface

D.1 Contents 155

D.2 Introduction 156

D.3 Features 157

D.4 Pin Connections and Ordering Information. 158

D.5 Electrical Characteristics 159
D.5.1 Maximum Ratings. 159
D.5.2 Thermal Ratings 159
D.5.3 DC Characteristics 160
D.5.4 AC Characteristics 163

D.6 Device Description 165
D.6.1 Packaging. 165
D.6.2 Transmitter Function 165
D.6.3 Receiver Function 166
D.6.4 Noise Filtering 166
D.6.5 Device Operation Modes 166
D.6.6 Operation Modes 168
D.6.7 System Power On 169
D.6.8 V_{DD} Reset Function 170
D.6.9 Battery Fail Flag 170
D.6.10 Bus Failure Detection 170
D.6.11 TX Permanent Dominant Detection 171
D.6.12 Behavior Under Faults Condition 171
D.6.13 Detailed Description of Error Detections 171
D.6.14 Wakeup Events 173
D.6.15 Fault Operation Table. 173

D.7 Pin Function Description 174
D.7.1 V_{BAT} (Input) 174
D.7.2 V_{DD} (Input) 174
D.7.3 CANH 174
D.7.4 RTH 175
D.7.5 CANL 175
D.7.6 RTL 175
D.7.7 STB and EN 176
D.7.8 INH 176
D.7.9 WAKE 176
D.7.10 TX 177
D.7.11 RX 177
D.7.12 NERR 177

D.8 Application 178

D.9 Electromagnetic Compatibility 180
D.9.1 EMI Noise 180
D.9.2 EMC Susceptibility Performances 181
D.9.3 Susceptibility Evaluation with BCI 182
D.9.4 Results 182

D.10 Case Outline Dimensions 183

Table of Contents

List of Figures

Figure	Title	Page
1-1	Motorola CAN Reference Design Board	19
1-2	Partitioning of the Reference Design Board	19
1-3	Option 1: Block Diagram Using the MC68HC908GR8	20
1-4	Option 2: Block Diagram Using the MC68HC908GP32 with a Serial Controller	21
1-5	Option 3: Block Diagram Using the MC68HC908GP32 with a Parallel CAN Controller	21
2-1	MC68HC908GR8 Block Diagram	25
2-2	MC68HC908GP32 Block Diagram	31
3-1	Motorola CAN Reference Design Board Modules	38
5-1	Jumper Connections	50
5-2	TP1 Example	52
5-3	Motor Control Circuitry	52
5-4	Motor Layout	54
6-1	MCP2510 Block Diagram	56
6-2	Block Diagram MC33388	57
6-3	Block Diagram MC68HC908GR8 and MC68HC908GP32 + MCP2510 + MC33388	57
6-4	Block Diagram SJA1000	58
6-5	Block Diagram of M68HC908GP32 Connection to Philips SJA1000	59

List of Figures

Figure	Title	Page
8-1	Option 1 — MC68HC908GR8 and MCP2510	66
8-2	MC68HC908GP32 and MCP2510	67
8-3	MC68HC908GP32 and SJA1000	68
A-1	Sheet 1 of CAN Reference Design Schematic with MAX232	80
A-2	CAN Reference Design Schematic with MC145487	81
B-1	CAN Reference Design Flowchart	88
D-1	Simplified Block Diagram	157
D-2	Pin Connections	158
D-3	Device Signal Waveforms	164
D-4	Test Circuit for AC Characteristics	164
D-5	State Machine and Operation Modes	167
D-6	Typical Application Schematic	178
D-7	Minimum Application Schematic	180
D-8	Typical Common Mode Glitch Measured at CANL and CANH	181
D-9	Minimum Susceptibility Level with Modulation	183
D-10	Minimum Susceptibility Level without Modulation	183
D-11	Outline Dimensions for Case 751A-03	184

List of Tables

Table	Title	Page
2-1	MC68HC908GR8 Features and Benefits	26
2-2	MC68HC908GR8 Package Options	29
2-3	MC68HC908GR8 Sample Packs	29
2-4	MC68HC908GP32 Features and Benefits	32
2-5	MC68HC908GP32 Package Options	35
2-6	MC68HC908GP32 Sample Packs	35
5-1	MC68HC908GR8 in Monitor Mode Block	46
5-2	MC68HC908GP32 in Monitor Mode Block	46
5-3	MC68HC908GR8 in Controller Area	47
5-4	MC68HC908GP32 in Controller Area	48
5-5	MC68HC908GP32 CAN Controller Selection	48
5-6	MC68HC908GR8 and MC68HC908GP32 in Application Area	51
5-7	Table of Test Points (TP)	52
5-8	Switching Possibilities	53
6-1	SJA1000 Control Bus	59
6-2	SJA1000 Address and Data Bus	59
8-1	Clock Generation Configurations	65
A-1	Bill of Materials	74
C-1	Embedded CAN Microcontroller Selector Guide	154
D-1	Ordering Information	158
D-2	Truth Table	168
D-3	Detail Fault Operation Table	172
D-4	Truth Table of RX and TX CAN Bus States	177

List of Tables

Section 1. General Description

1.1 Contents

1.2 Introduction 17

1.3 Reference Design Overview 20
1.3.1 Option 1: MC68HC908GR8 + MCP2510 + MC33388 20
1.3.2 Option 2: MC68HC908GP32 + MCP2510 + MC33388 21
1.3.3 Option 3: MC68HC908GP32 + SJA1000 + PCA82C250 . . 21

1.2 Introduction

The low-cost controller area network (CAN) reference design described in this document shows the coupling possibilities of both the Motorola low-cost MC68HC908GR8 and MC68HC908GP32 to a CAN bus system using either serial or parallel communication.

The M68HC08 Family derivatives used in this design do not have an internal CAN module. This method of combining a general-purpose Motorola microcontroller unit (MCU) with a standalone CAN controller illustrates how performance and flexibility can be achieved at a very low cost. The method and software described here can, of course, be used with many different members of the Motorola M68HC08 Family which allows you to choose different versions (depending on your system needs) while completely reusing software and methods. A single common MCU could then be used in both CAN and non-CAN platforms.

Motorola also has the most extensive portfolio of FLASH- and ROM-based integrated CAN 8-, 16-, and 32-bit MCUs in the world, with more on the way! **Appendix C. Motorola Embedded CAN MCU Selector Guide** shows an overview of these derivatives. More detailed information can be found on the World Wide Web at:

http://www.motorola.com/semiconductors/

NOTE: *Several more derivatives are in design, so check with your local Motorola representative for details.*

The hardware board developed to complement and demonstrate the functionality of the design includes:

- A universal monitor debug interface
- An application area including:
  - Sensors
  - Push buttons
  - Light-emitting diodes (LED)
  - A motor driver which can simulate many applications
- An RS232 interface using the internal serial communications interface (SCI) of the MCU which can be used to connect the board to a personal computer

The printed circuit board illustrated in **Figure 1-1** can be separated into three areas to allow for “mix and match” reuse on further designs. Partitioning of the reference design board is shown in **Figure 1-2**.

Figure 1-1. Motorola CAN Reference Design Board

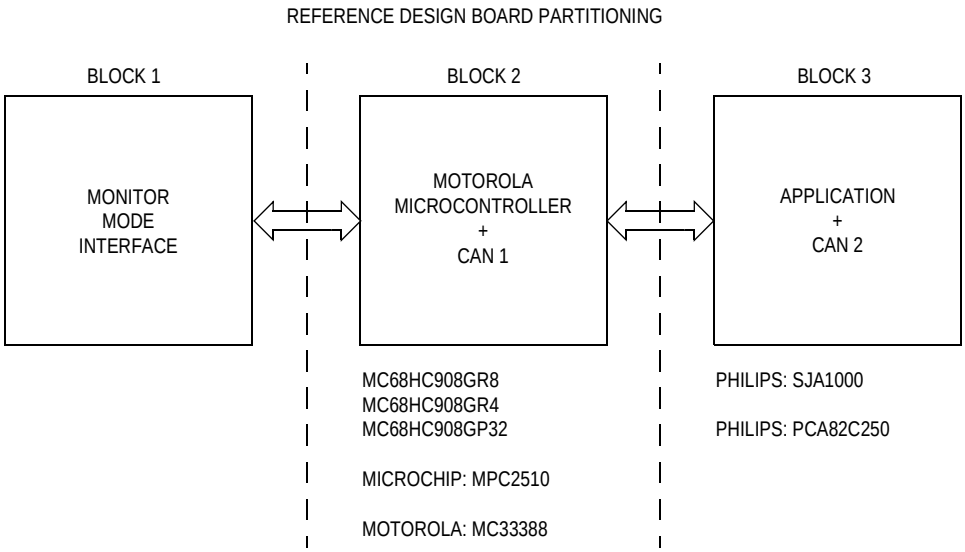


Figure 1-2. Partitioning of the Reference Design Board

1.3 Reference Design Overview

Depending on the user's preferred configuration and cost targets, the reference design uses either:

- The MC68HC908GR8 8-K embedded FLASH controller in a 28-pin or 32-pin package, or
- The MC68HC908GP32 32-K embedded FLASH controller in a 40-pin, 42-pin, or 44-pin package

In all examples, the special characteristics of the two MCUs relevant to this use are highlighted.

In both cases we have a simple 3-chip solution for coupling to the CAN bus:

Microcontroller + CAN Controller +
CAN Transceiver (physical interface)

1.3.1 Option 1: MC68HC908GR8 + MCP2510 + MC33388

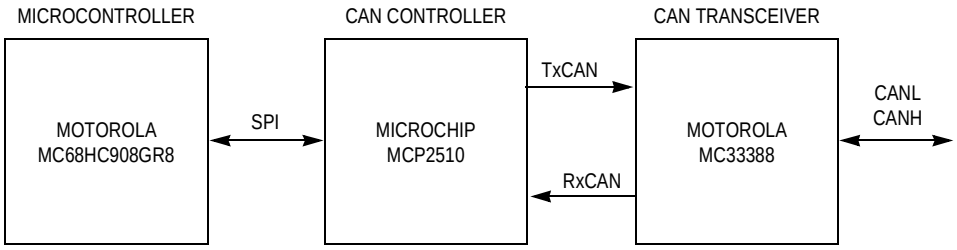


Figure 1-3. Option 1: Block Diagram Using the MC68HC908GR8

Communication between the MCU (MC68HC908GR8) and the external CAN controller (Microchip MCP2510) is done via the serial peripheral interface (SPI) of the MCU. A minimum of four SPI pins of the controller is used for the connection. This is an appropriate solution for low pin count microcontrollers.

By using the higher integrated MC68HC908GP32, we can show two variants. They are selected by the positioning of jumpers (15 and 2). See **1.3.2 Option 2: MC68HC908GP32 + MCP2510 + MC33388** and **1.3.3 Option 3: MC68HC908GP32 + SJA1000 + PCA82C250**.

1.3.2 Option 2: MC68HC908GP32 + MCP2510 + MC33388

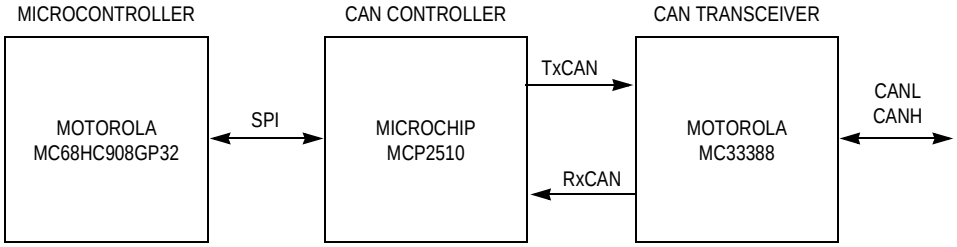


Figure 1-4. Option 2: Block Diagram Using the MC68HC908GP32 with a Serial Controller

In this option, communication between the MCU and the external CAN controller is also done via the SPI interface. Again, a minimum of four SPI lines of the controller are used for the connection. This allows most of the microcontroller I/Os (inputs/outputs) to be available for the application.

1.3.3 Option 3: MC68HC908GP32 + SJA1000 + PCA82C250

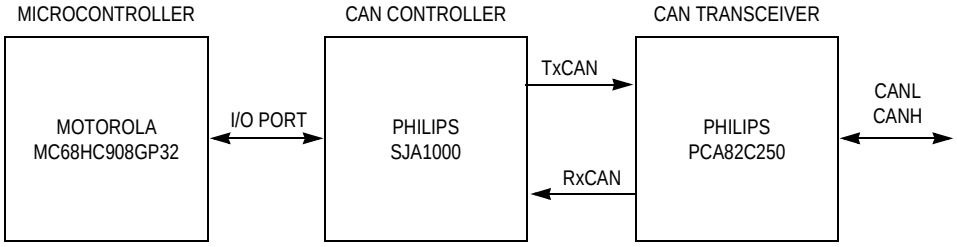


Figure 1-5. Option 3: Block Diagram Using the MC68HC908GP32 with a Parallel CAN Controller

The CAN controller used here has a parallel interface to the microcontroller. A minimum of eight address data lines and four control lines are necessary. Since the MC68HC908GP32 does not have an external address and data bus, these are builtin I/O port functions and software.

The selection between the two CAN controllers (for example, the MCP2510) and the SJA1000 is made possible by using jumpers (15 and 22). The allocation of the CAN controller and the CAN transceiver is defined as follows:

- The MCP2510 is used together with the MC33388.
- The SJA1000 is used together with the PCA82C250.

The MC33388 is a fault tolerant CAN transceiver offering baud rates up to 250 kbaud on the CAN bus. With the PCA82C250, baud rates up to 1 Mbaud are possible.

Section 2. Overview of Embedded FLASH MCUs Used

2.1 Contents

2.2 Introduction 23
2.3 MC68HC908GR8 24
2.3.1 MC68HC908GR8 Application Notes and Engineering Bulletins 28
2.3.2 MC68HC908GR8 Development Tool Kits 28
2.3.3 MC68HC908GR8 Individual Development Tool Components 29
2.3.4 MC68HC908GR8 Package Options 29
2.3.5 MC68HC908GR8 Sample Packs 29
2.4 MC68HC908GP32 30
2.4.1 MC68HC908GP32 Application Notes and Engineering Bulletins 34
2.4.2 MC68HC908GP32 Development Tool Kits 34
2.4.3 MC68HC908GP32 Individual Development Tool Components 35
2.4.4 MC68HC908GP32 Package Options 35
2.4.5 MC68HC908GP32 Sample Packs 35

2.2 Introduction

This section provides an overview of the Motorola MC68HC908GR8 and MC68HC908GP32 embedded FLASH microcontrollers (MCU) used in this reference design.

2.3 MC68HC908GR8

The MC68HC908GR8 utilizes integrated second generation FLASH and is enhanced with embedded, on-chip functions that eliminate the need for external serial components. The 32-kHz phase-locked loop (PLL) provides cost savings by replacing the need for expensive, high-speed crystals or noisy oscillators. The on-chip timebase module (TBM) further reduces costs by eliminating the need for external real-time clock and wakeup circuitry. See **Figure 2-1**.

Other features are:

- Analog-to-digital converter (ADC)
- Serial communications interface (SCI)
- Serial peripheral interface (SPI)
- Low-voltage inhibit (LVI)
- Computer operating properly (COP) watchdog timer

Table 2-1 provides an overview of the features and benefits of the MC68HC908GR8 broken down by major components.

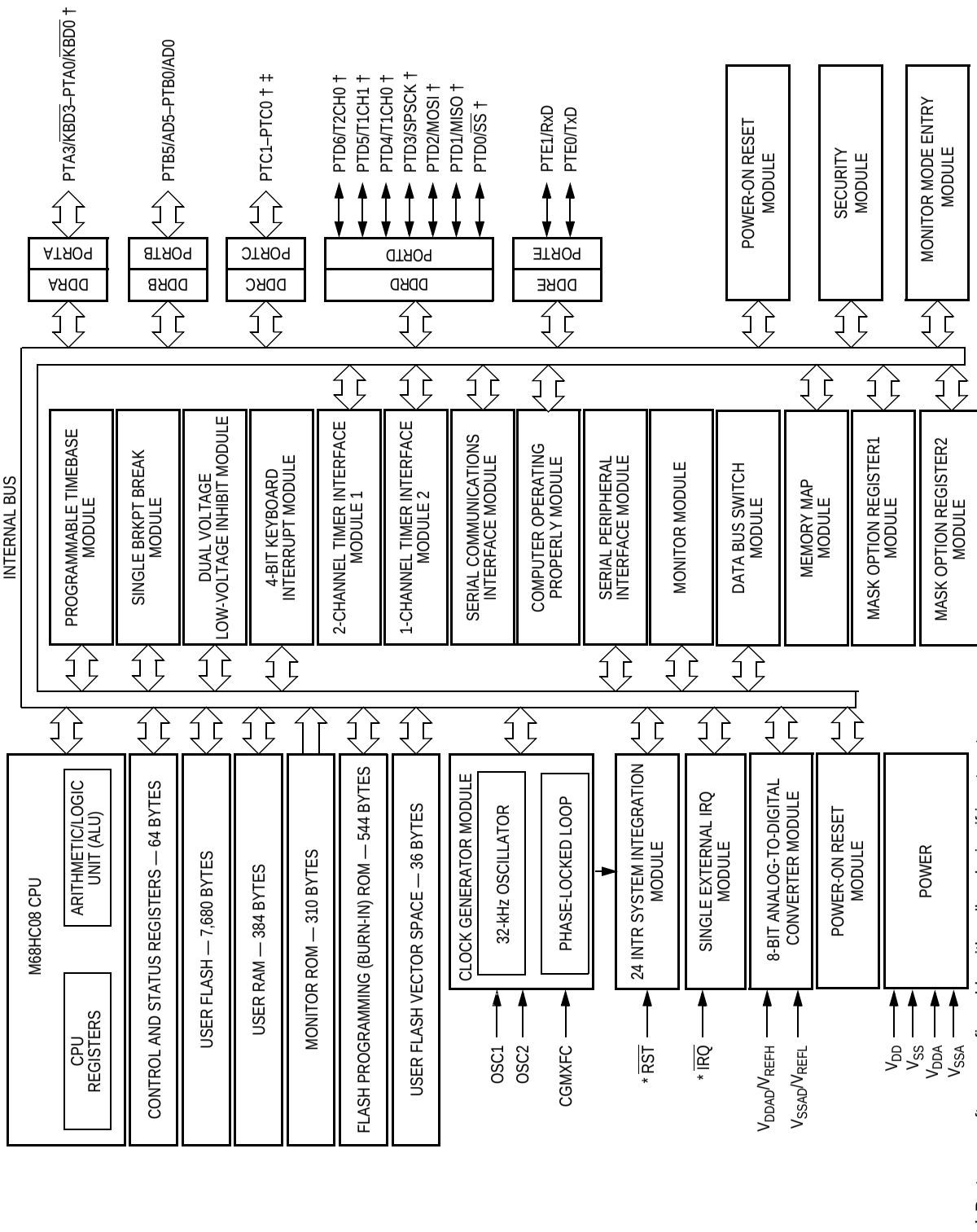


Figure 2-1. MC68HC908GR8 Block Diagram

Table 2-1. MC68HC908GR8 Features and Benefits

Features	Benefits
High-performance M68HC08 central processor unit (CPU) core	
• High-performance M68HC08 central processor unit (CPU) core • 8-MHz bus operation at 5-V operation for 125 ns minimum instruction cycle time • 4-MHz bus operation at 3 V for 250 ns minimum instruction cycle time • Efficient instruction set including multiply and divide • 16 flexible addressing modes including stack relative with 16-bit stack pointer • Fully static low-voltage, low-power design with wait and stop modes	• Object code compatible with the M68HC05 Family • Easy to learn and use architecture • C optimized architecture provides compact code
Integrated second generation FLASH memory	
• In-application re-programmable • Extremely fast programming, encoding 64 bytes in as fast as 2 ms • FLASH programming across the M68HC08's full operating supply voltage with no extra programming voltage • 10 K write/erase cycles minimum over temperature • 100 K write/erase cycles typical • Flexible block protection and security	• Cost-effective programming changes and field software upgrades via in-application • Reduces production programming costs through ultra-fast programming • Allows re-programmable battery-powered applications • Byte-writable for data as well as program memory • Protects code from unauthorized reading and to guard against unintentional erasing/writing of user-programmable segments of code
8-bit analog-to-digital converter (ADC)	
• Eight/six channels • Single conversion in 17 µs	• Fast, easy conversion from analog inputs like temperature, pressure, and fluid levels to digital values for CPU processing
Clock generation module with phase-locked loop (PLL)	
• Programmable clock frequency in integer multiples of external crystal reference • Crystal reference of 32 kHz to 100 kHz • External clock option with or without PLL	• Provides high performance using low-cost, low-frequency reference crystals • Reduces generated noise while still providing high-performance (up to 32 MHz internal clock)
Four programmable 16-bit timer channels	
• 125 ns resolution at 8 MHz bus • Free-running counter or modulo up-counter	• Each channel independently programmable for input capture, output compare, or unbuffered PWM • Pairing timer channels provides a buffered PWM function

Table 2-1. MC68HC908GR8 Features and Benefits (Continued)

Features	Benefits
Timebase module (TBM)	
• Eight user-selectable periodic real-time interrupts • Optionally operate in low-power stop mode	• Provides auto wakeup from low-power stop mode to maintain real-time clock or check external device status such as sensors
Serial communications interface (SCI)	
• UART asynchronous communications system • Flexible baud rate generator • Double buffered transmit and receive • Optional hardware parity check and generation	• Enables high-speed asynchronous communication
Serial peripheral interface (SPI)	
• Full-duplex 3-wire synchronous transfers • Maximum master bit rate of 4 MHz for 8 MHz system clock	• High-speed synchronous communication between multiple MCUs or between MCU and serial peripherals • Cost-effective serial peripheral expansion to EEPROM, high-precision A/D and D/A converters, real-time clocks, etc.
Computer operating properly (COP) watchdog timer	
	• Issues reset in the event of runaway code
Selectable trip point low-voltage inhibit (LVI)	
	• Improves reliability by resetting the MCU when voltage drops below trip point • Two trip points allow optimum operation in both 5 V and 3 V nominal systems • Integration reduces system cost
Up to 21 bidirectional input/output (I/O) lines	
• 10 mA sink/source capability on all I/O pins • 15 mA sink capability on two I/O pins • Keyboard scan with selectable interrupts for four I/O pins • Software programmable pullups on 13 I/O pins	• High-current I/O allows direct drive of light-emitting diodes (LED) and other circuits to eliminate external drivers and reduce system costs • Keyboard scan with programmable pullups eliminates external glue logic when interfacing to simple keypads

2.3.1 MC68HC908GR8 Application Notes and Engineering Bulletins

For additional information, these reference documents are available:

- *In-Circuit Programming of 68HC908GR8 FLASH Memory*, Motorola order number EB368/D
- *Using MC68HC908 On-Chip Programming Routines*, Motorola order number AN1831/D
- *Creating Efficient C Code for the HC08*, Motorola order number AN2093/D
- *M68HC08 Integer Math Routines*, Motorola order number AN1219/D
- *HC05 to HC08 Optimization*, Motorola order number AN1218/D
- *Non-Volatile Memory Technology Review*, Motorola order number AN1837/D
- *Data Structures for 8-Bit MCUs*, Motorola order number AN1742/D
- *Noise ReductionTechniques for MCU-Based Systems*, Motorola order number AN1704/D

These documents and more are available on the World Wide Web at:
http://www.motorola.com/semiconductors/

2.3.2 MC68HC908GR8 Development Tool Kits

Part Number	Description
M68ICS08GR	68HC908GRx programmer/in-circuit debug kit
KITMMEVS08GR	Cost-effective real-time in-circuit emulator kit
KITMMDS08GR	High-performance real-time in-circuit emulator kit

2.3.3 MC68HC908GR8 Individual Development Tool Components

Part Number	Description
M68MMDS508	High-performance emulator
M68MMPFB0508	MMEVS platform board
M68EML08GP32	Emulation module daughter board
M68CBL05C	Low-noise flex cable
M68TC08GR8FA32	32-pin QFP target head adapter
M68TC08GR8P28	28-pin DIP target head adapter
M68TQS032SAG1	32-pin TQ socket with guides
M68TQP032SA1	32-pin TQPACK
M68DIP28SOIC	28-pin surface mount adapter

2.3.4 MC68HC908GR8 Package Options

Table 2-2. MC68HC908GR8 Package Options

Part Number	Package	Temperature Range
MC68HC908GR8CFA	32-pin quad flat pack (QFP)	–40° C to +85° C
MC68HC908GR8CP	28-pin plastic dual in-line (DIP)	–40° C to +85° C
MC68HC908GR8CDW	28-pin small outline integrated circuit (SOIC)	–40° C to +85° C

2.3.5 MC68HC908GR8 Sample Packs

Table 2-3. MC68HC908GR8 Sample Packs

Part Number	Package	Temperature Range
KMC908GR8CFA	32-pin quad flat pack (QFP)	–40° C to +85° C
KMC908GR8CP	28-pin plastic dual in-line (DIP)	–40° C to +85° C
KMC908GR8CDW	28-pin small outline integrated circuit (SOIC)	–40° C to +85° C

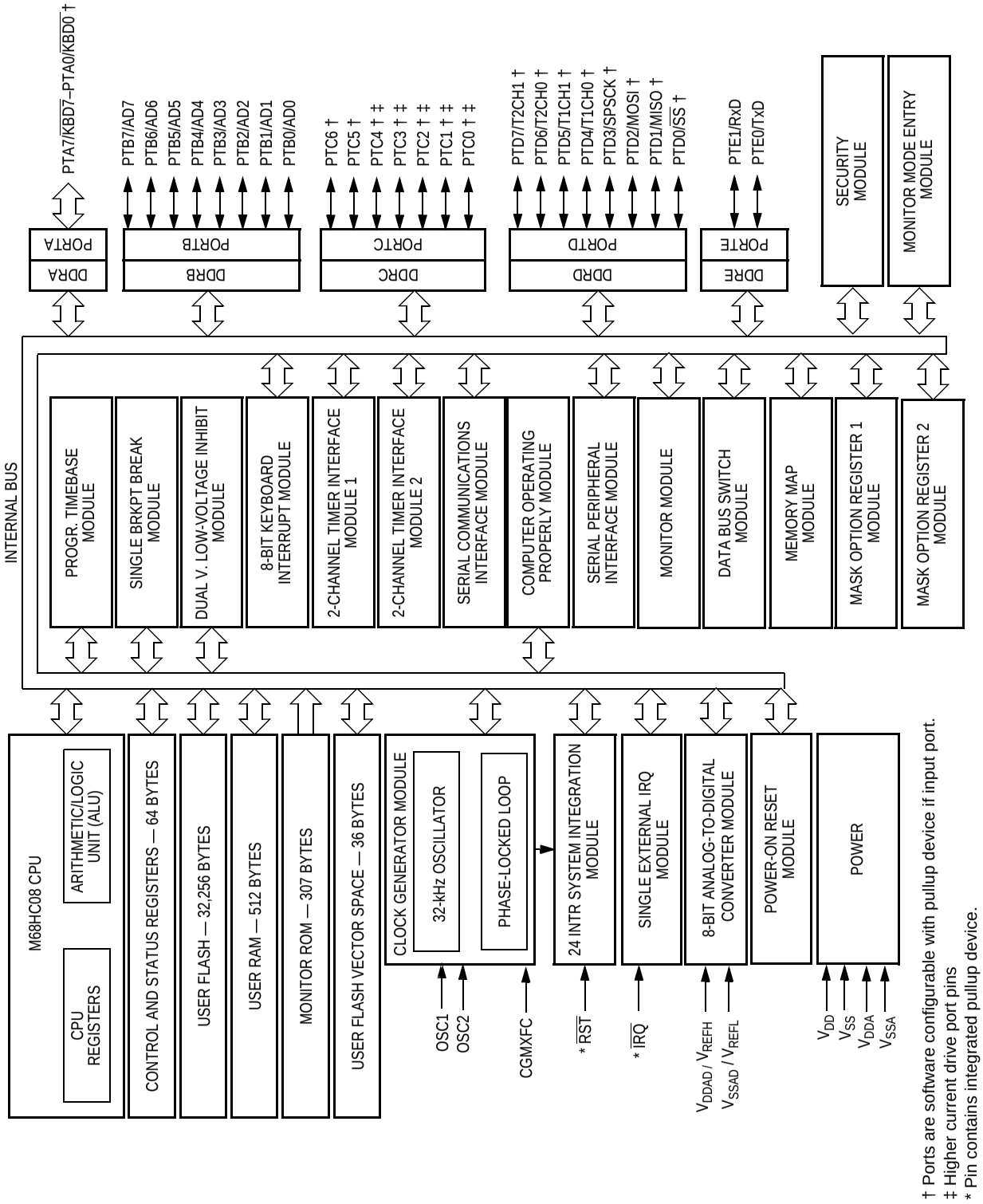
2.4 MC68HC908GP32

The MC68HC908GP32 is a fully integrated MCU created to make system design easier by eliminating external peripherals wherever possible. The 32-kHz PLL eliminates the need for expensive, high-speed crystals or noisy oscillators. The integrated second generation FLASH memory programs up to 100 times faster than prior FLASH solutions and offers in-application programming. See **Figure 2-2**.

Features include:

- Asynchronous serial peripheral interface (SPI)
- Asynchronous serial communications interface (SCI)
- Analog-to-digital converter (ADC)
- Auto wakeup from stop feature
- Low-voltage inhibit (LVI)
- COP watchdog timer

Table 2-4 provides an overview of the features and benefits of the MC68HC908GP32 broken down by major components.



† Ports are software configurable with pullup device if input port.
‡ Higher current drive port pins
* Pin contains integrated pullup device.

Figure 2-2. MC68HC908GP32 Block Diagram

Table 2-4. MC68HC908GP32 Features and Benefits

Features	Benefits
High-performance M68HC08 central processor unit (CPU) core	
• 8 MHz bus operation at 5 V operation for 125 ns minimum instruction cycle time • 4 MHz bus operation at 3 V for 250 ns minimum instruction cycle time • Efficient instruction set including multiply and divide • 16 flexible addressing modes including stack relative with 16-bit stack pointer • Fully static low-voltage, low-power design with wait and stop modes	• Object code compatible with the M68HC05 Family • Easy to learn and use architecture • C optimized architecture provides compact code
Integrated second generation FLASH memory	
• In-application re-programmable • Extremely fast programming, encoding 64 bytes in as fast as 2 ms • FLASH programming across the M68HC08's fully operating supply voltage with no extra programming voltage • 10 K write/erase cycles minimum over temperature • 100 K write/erase cycles typical • Flexible block protection and security	• Cost-effective programming changes and field software upgrades via in-application programmability and re-programmability • Reduces production programming costs through ultra-fast programming • Allows re-programmable battery-powered applications • Byte-writable for data as well as program memory • Protects code from unauthorized reading and to guard against unintentional erasing/writing of user-programmable segments of code
8-bit analog-to-digital converter (ADC)	
• Eight channels • Single conversion in 17 µs	• Fast, easy conversion from analog inputs like temperature, pressure, and fluid levels to digital values for CPU processing
Clock generation module with phase-locked loop (PLL)	
• Programmable clock frequency in integer multiples of external crystal reference • Crystal reference of 32 kHz to 100 kHz • External clock option with or without PLL	• Provides high performance using low-cost, low-frequency reference crystals • Reduces generated noise while still providing high performance (up to 32 MHz internal clock) • Fast, easy conversion from analog inputs like temperature, pressure, and fluid levels to digital values for CPU processing
Four programmable 16-bit timer channels	
• 125 ns resolution at 8 MHz bus • Free-running counter or modulo up-counter	• Each channel independently programmable for input capture, output compare, or unbuffered PWM • Pairing timer channels provides a buffered PWM function

Table 2-4. MC68HC908GP32 Features and Benefits (Continued)

Features	Benefits
Timebase module (TBM)	
• Eight user-selectable period real-time interrupts • Optionally operate in low-power stop mode	• Provides auto wakeup from low-power stop mode to maintain real-time clock or check external device status such as sensors
Serial communications interface (SCI)	
• UART asynchronous communications system • Flexible baud rate generator • Double buffered transmit and receive • Optional hardware parity checking and generation	• Enables synchronous serial communications with peripheral devices
Serial peripheral interface (SPI)	
• Full-duplex 3-wire synchronous transfers • Maximum master bit rate of 4 MHz for 8 MHz system clock	• High-speed synchronous communication between multiple MCUs or between MCU and serial peripherals • Cost-effective serial peripheral expansion to EEPROM, high-precision A/D and D/A converters, real-time clocks, etc.
Computer operating properly (COP) watchdog timer	
	• Issues reset in the event of runaway code
Selectable trip point low-voltage inhibit (LVI)	
	• Improves reliability by resetting the MCU when voltage drops below trip point • Two trip points allow optimum operation in both 5 V and 3 V nominal systems • Integration reduces system cost
Up to 33 bidirectional input/output (I/O) lines	
• 10 mA sink/source on all I/O pins • 15 mA sink capability on five I/O pins • Keyboard scan with selectable interrupts on eight I/O pins • Software programmable pullups on 23 I/O pins	• High-current I/O allows direct drive of LED and other circuits to eliminate external drivers and reduce system costs • Keyboard scan with programmable pullups eliminates external glue logic when interfacing to simple keypads

2.4.1 MC68HC908GP32 Application Notes and Engineering Bulletins

For additional information, these reference documents are available:

- *In-Circuit Programming of 68HC908GP32 FLASH Memory*, Motorola order number EB366/D
- *Creating Efficient C Code for the HC08*, Motorola order number AN2093/D
- *M68HC08 Integer Math Routines*, Motorola order number AN1219/D
- *HC05 to HC08 Optimization*, Motorola order number AN1218/D
- *Non-Volatile Memory Technology Review*, Motorola order number AN1837/D
- *Data Structures for 8-Bit MCUs*, Motorola order number AN1752/D
- *Noise Reduction Techniques for MCU-Based Systems*, Motorola order number AN1705/D

These documents and more are available on the World Wide Web at:
http://www.motorola.com/semiconductors/

2.4.2 MC68HC908GP32 Development Tool Kits

Part Number	Description
M68ICS08GP	68HC908GPxx programmer/in-circuit debug kit
KITMMEVS08GP	Cost-effective real-time in-circuit emulator kit
KITMMDS08GP	High-performance real-time in-circuit emulator kit

2.4.3 MC68HC908GP32 Individual Development Tool Components

Part Number	Description
M68MMD0508	High performance emulator
M68MMPFB0508	MMEVS platform board
M68EML08GP32	Emulation module daughter board
M68CBL05B	Low-noise flex cable
M68CBL05C	Low-noise flex cable
M68TB08GP32P40	40-pin DIP target head adapter
M68TC08GP32FB44	44-pin QFP target head adapter
M68TQS044SAG1	44-pin TQ socket with guides
M68TQP044SAMO1	44-pin TQPACK

2.4.4 MC68HC908GP32 Package Options

Table 2-5. MC68HC908GP32 Package Options

Part Number	Package	Temperature Range
MC68HC908GP32CP	40-pin dual in-line (DIP)	–40° C to +85° C
MC68HC908GP32CFB	44-pin quad flat pack (QFP)	–40° C to +85° C
MC68HC908GP32CB	42-pin shrink dual in-line (SDIP)	–40° C to +85° C

2.4.5 MC68HC908GP32 Sample Packs

Table 2-6. MC68HC908GP32 Sample Packs

Part Number	Package	Temperature Range
KMC908GP32CP	40-pin dual in-line (DIP)	–40° C to +85° C
KMC908GP32CFB	44-pin quad flat pack (QFP)	–40° C to +85° C
KMC908GP32CB	42-pin shrink dual in-line (SDIP)	–40° C to +85° C

Overview of Embedded FLASH MCUs Used

Section 3. Printed Circuit Board Structure

3.1 Contents

3.2 Introduction 37
3.3 Monitor Mode Interface (Area 1) 38
3.4 Controller (Area 2) 39
3.5 Application (Area 3) 39

3.2 Introduction

The printed circuit board (PCB) is:

- 200 mm x 100 mm
- Manufactured in four multilayers
- Needs a supply voltage of 9 to 18 Vac or Vdc

The printed circuit board has three functional areas which are discussed in this section (see **Figure 3-1**).

NOTE: *If necessary, the board can be broken apart at the break strips of these areas. Ribbon cables connect the areas.*

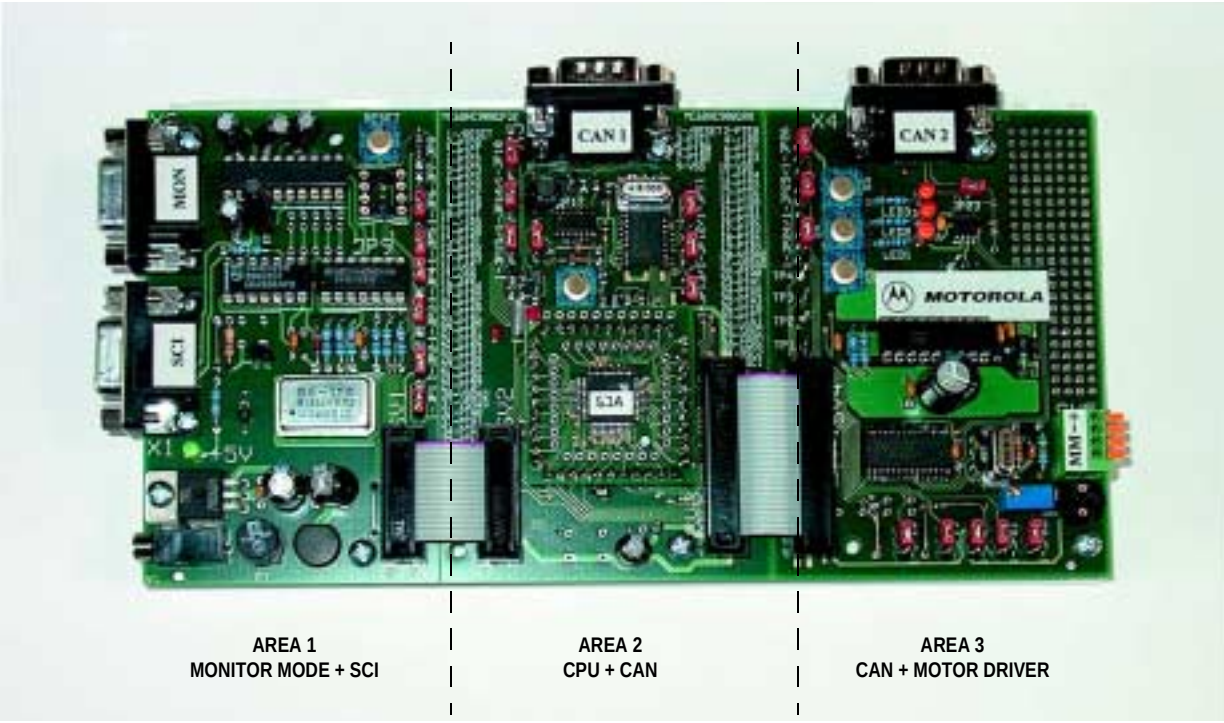


Figure 3-1. Motorola CAN Reference Design Board Modules

3.3 Monitor Mode Interface (Area 1)

Area 1 contains these functional groups:

- Power supply and level adapter on RS232 for the monitor interface and serial communications interface (SCI)
- Pin connections for monitor operation
- Reset module and reset push button
- Clock oscillator

Area 1 connections (plugs) are:

- Supply voltage (X3, two contacts)
- Monitor interface (X1, DSUB, nine contacts, female connector)
- SCI Interface (X2, DSUB,nine contacts, female connector)
- Connection to the monitor PCB (SV1, ribbon connector, 16 contacts)

3.4 Controller (Area 2)

Area 2 contains these functional groups:

- MC68HC908GR8 or MC68HC908GP32 microcontroller
- Oscillator with a 32-kHz crystal
- MCP2510 CAN controller
- MC33388 CAN transceiver

Area 2 connections (plugs) are:

- Connection to the monitor PCB (SV2, ribbon connector, 16 contacts)
- Connection to the application PCB (SV3, ribbon connector, 26 contacts)
- CAN 1 interface 1 (X5, DSUB, nine contacts, male connector)

3.5 Application (Area 3)

Area 3 contains these functional groups:

- SJA1000 CAN controller
- PCA82C250 CAN interface
- L6201, bridge for motor
- Push buttons T1, T2, and T3
- Light-emitting diodes (LED) 1, 2, and 3
- Potentiometer
- Wire wrap field

Area 3 connections are:

- Connection to the PCB controller (SV4, ribbon connector, 26 contacts)
- CAN 2 interface (X5, DSUB, nine contacts, male)
- Motor connection (SV5, four contacts)

Printed Circuit Board Structure

Section 4. Pin Assignments and Jumper Settings

4.1 Contents

4.2 Introduction 41
4.3 Supply Voltage X3 42
4.4 Monitor Mode X2 42
4.5 SCI X1 42
4.6 CAN1 X5 43
4.7 CAN2 X4 43
4.8 Motor SV5 44

4.2 Introduction

This section describes the pin assignments and jumper settings for the design reference. For clarification, these designations are defined as:

- NC (not connected)
  This connection is not forwarded on the printed circuit board (PCB).
- Reserved
  This connection goes to internal circuits. It is included for possible use in further options and should not be wired.

4.3 Supply Voltage X3

Jack connector, 3.5 mm diameter, polarity arbitrary

4.4 Monitor Mode X2

DSUB, nine contacts (see table), female connector

1	Reserved
2	TxD
3	RxD
4	NC
5	GND
6	Reserved
7	NC
8	Reserved
9	Reserved

4.5 SCI X1

DSUB, nine contacts (see table), female connector

1	Reserved
2	TxD
3	RxD
4	NC
5	GND
6	Reserved
7	NC
8	Reserved
9	Reserved

4.6 CAN1 X5

DSUB, nine contacts (see table), male connector

1	NC
2	CAN-L
3	GND
4	NC
5	NC
6	NC
7	CAN-H
8	NC
9	NC

Pins 5 and 6 can be ground by solder straps at the lower side of the DSUB connector.

4.7 CAN2 X4

DSUB, nine contacts (see table), male connector

1	NC
2	CAN-L
3	GND
4	NC
5	NC
6	NC
7	CAN-H
8	NC
9	NC

Pins 5 and 6 can be ground by solder straps at the lower side of the DSUB connector.

4.8 Motor SV5

Connector, four contacts (see table)

1	motor +
2	motor –
3	GND
4	$+V_{ext}$

CAUTION: *The supply voltage for the motor driver module L6202 can be switched from JP20 of X3 (reference design supply) or via contacts 3 and 4 of X6. The voltage should not exceed 18 Vdc!*

As shown in this table, the motor furnished can be operated straight forward by the power supply furnished.

Jumper	Functions 1 and 2	Functions 2 and 3	Default
20	Power for L6202 from SV5	Power for L6202 from X6	1-2

Section 5. Jumper Description

5.1 Contents

5.2 Introduction 45
5.3 Jumpers in the Monitor Mode Area. 45
5.4 Jumpers in the Controller Area 47
5.5 Jumpers in the Application Area 49
5.6 Motor Driver L6202 53

5.2 Introduction

Different jumpers are positioned on the reference design for flexible adaptation to the multiple possibilities. The jumpers are identified by component identification printing. For jumpers with two plug-in places pin 1 is identified.

Example: 1-JP1

1	2	3

5.3 Jumpers in the Monitor Mode Area

NOTE: *For this discussion refer to* ***Figure 1-2. Partitioning of the Reference Design Board****.*

A jumper field has been created to allow multiple connections of the reset conditions and, of course, the connection to the monitor mode. It has been designed with headroom to allow for future microcontroller derivatives to be used.

Table 5-1 and Table 5-2 show the usage of the reference design with the microcontroller derivatives MC68HC908GR8 and MC68HC908GP32. For further information, reference the monitor mode sections of:

- *MC68HC908GR8 Advance Information*, Motorola order number MC68HC908GR8/D
- *MC68HC908GP32 Technical Data*, Motorola order number MC68HC908GP32/H

Table 5-1. MC68HC908GR8 in Monitor Mode Block

Jumper	Functions 1 and 2	Functions 2 and 3	Default(1)
1	IRQ = GND	IRQ = V_{TST}	2-3
2	V_{TST} source from power supply	V_{TST} source from MC145407	1-2
3	PTA1 = +5 V	PTA1 = GND	2-3
4	PTB1 = +5 V	PTB1 = GND	2-3
5	Not used	Not used	1-2
6	PTB0 = +5 V	PTB0 = GND	1-2

1. Default — monitor mode is active, external oscillator, 9600 baud, see JP10 and JP14 in the microcontroller block of **Figure 1-1. Motorola CAN Reference Design Board**

Table 5-2. MC68HC908GP32 in Monitor Mode Block

Jumper	Functions 1 and 2	Functions 2 and 3	Default(1)
1	IRQ = GND	IRQ = V_{TST}	2-3
2	V_{TST} source from power supply	V_{TST} source from MC145407	1-2
3	PTA7 = +5 V	PTA7 = GND	2-3
4	PTC1 = +5 V	PTC1 = GND	2-3
5	PTC3 = +5 V	PTC3 = GND	1-2
6	PTC0 = +5 V	PTC0 = GND	1-2

1. Default — monitor mode is active, external oscillator, 9600 baud, see JP10 and JP14 in the microcontroller block of **Figure 1-1. Motorola CAN Reference Design Board**

5.4 Jumpers in the Controller Area

The reference design is either equipped with the MC68HC908GR8 or with the MC68HC908GP32 which has more configurations. **Table 5-3** and **Table 5-4** provide an overview of both parts.

Table 5-3. MC68HC908GR8 in Controller Area

Jumper	Functions 1 and 2	Functions 2 and 3	Default
10	External clock oscillator	32 kHz crystal on	1-2
11	PTC1 to EN MC33388	PTC1 to TX2RTS MCP2510	1-2
12	PTC0 to STB MC33388	PTC0 to TX1RTS MCP2510	1-2
13	PTA3 to NERR MC33388	PTA3 to TX0RTS MCP2510	1-2
14	32 kHz crystal off	32 kHz crystal on	1-2
15	Enable MCP2510	Disable MC2510	1-2
16	V_{PS} to BAT MC33388	+5 V to BAT MC33388	2-3

Note that:

- Jumpers JP10 and JP14 should both be connected in position 1-2 or 2-3 at the same time.
- Selection of clock generation is done by jumpers JP10 and JP14. The pluggable external oscillator module is placed in the monitor area. The result of this design is a very flexible adaptation. Refer to the monitor mode and clock generation sections of the MC68HC908GR8 and MC68HC908GP32 manuals and to **Figure 1-1. Motorola CAN Reference Design Board** for other plug combinations.
- With the MC68HC908GR8, jumper 15 should always stay in position 1-2 for CAN operation.
- With the MC68HC908GR8, the CAN operation is only possible with the MCP2510 due to available input/output (I/O).
- The selection of three I/O lines is done by the jumpers JP11, JP12, and JP13.
- In using plug position 1-2, the possibilities of the CAN transceiver MC33388 are available.

- Using plug positions 2-3 the possibilities of the CAN controller MCP2510 are better used. Here, start of a transmission in the transmitter buffer is done by hardware.
- Jumper 17 has no function.

Table 5-4. MC68HC908GP32 in Controller Area

Jumper	Functions 1 and 2	Functions 2 and 3	Default
10	External clock oscillator	32 kHz crystal on	1-2
11	PTC6 to EN MC33388	PTC6 to TX2RTS MCP2510	1-2
12	PTC5 to STB MC33388	PTC5 to TX1RTS MCP2510	1-2
13	PTA4 to NERR MC33388	PTA4 to TX0RTS MCP2510	1-2
14	32 kHz crystal off	32 kHz crystal on	1-2
15	Enable MCP2510	Disable MC2510	1-2
16	V_{PS} to BAT MC33388	+5 V to BAT MC33388	2-3

Note that:

- Jumpers JP10 and JP14 should both be in position 1-2 or 2-3. The selection of the possibilities of clock generation is performed with these two jumpers. The pluggable external oscillator module is located in the monitor area where a very flexible adaptation can be performed. Refer to the monitor mode and clock generation sections of the MC68HC908GR8 and MC68HC908GP32 manuals and to **Figure 1-1. Motorola CAN Reference Design Board** for other plug combinations.
- With jumper JP15 and JP22 (in the application block) the selection of the corresponding CAN controller is performed.
- Only one of the two CAN controllers should be enabled at the same time. Refer to **Table 5-5**.

Table 5-5. MC68HC908GP32 CAN Controller Selection

Jumper	Functions 1 and 2	Functions 2 and 3	Default
15	Enable MCP2510	Disable MC2510	1-2
22	Disable SJA1000	Enable SJA1000	1-2

- If MCP2510 is enabled (JP15 1-2 and JP22 1-2), the connection with jumpers JP11, JP12, and JP13, the selection of three I/O lines is performed.
- Using the plug positions 1-2, the possibilities of the CAN transceiver MC33388 are accessible.
- Using the plug positions 2-3 (JP11, JP12, and JP13), the possibilities of the CAN controller MCP2510 are performed better. Here, start of a transmission in the transmitter buffer is done by hardware.
- Jumper 17 has no function.

5.5 Jumpers in the Application Area

In the application area:

- By connecting to the I/O lines, selection of different available tasks is accomplished.
- By choosing jumpers, different functions can be made available.

Different functions can be made available at jumpers JP18, JP19, and JP21. Here once again a flexible solution is made possible. The number of jumpers and their possibilities is not as complicated as it may first look. In position 1-2, the signals at jumpers JP18 and JP19 are connected to a further jumper and there split up once again.

Figure 5-1 shows:

- The connection between jumpers JP18, JP19, and JP21 and jumpers JP24, JP25, and JP26
- The distribution to the light-emitting diodes (LED) or switches (S1, S2, and S3)

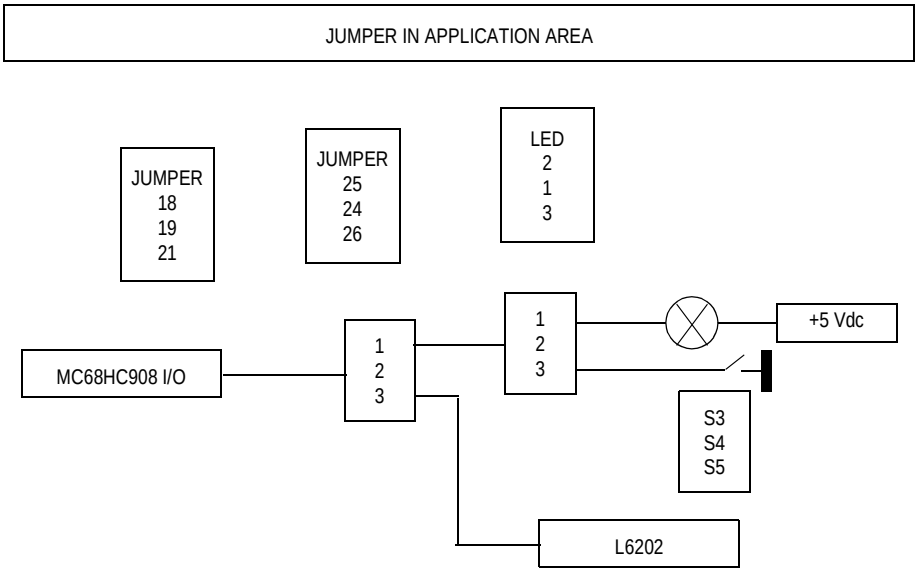


Figure 5-1. Jumper Connections

In the simplest case:

- With jumpers JP24, JP25, and JP26 in position 1-2, the special I/O line is defined as the output.
- With jumpers JP24, JP25, and JP26 in position 2-3, the special I/O line is the input.
- By dynamically reprogramming the standard definition output to input for only a short time, it is possible to use input and output at the same time.
- A special jumper that connects positions 1-2-3 at the same time is used.

NOTE: *If push button (S1, S2, S3) is pressed, the corresponding LED lights at the same time.*

- Jumpers JP18, JP19, and JP21 should be in position 2-3 to use the motor driver L6202.

Table 5-6. MC68HC908GR8 and MC68HC908GP32 in Application Area

Jumper	Functions 1 and 2	Functions 2 and 3	Default
18	PTD6 to JP25	PTD6 to PIN2 L6202	1-2
19	PTD5 to JP24	PTD5 to PIN 16 L6202	1-2
21	PTD4 to JP26	PTD4 to PIN 12 L6202	1-2
20	Power for L6202 from V_{CC}	Power for L6202 from SV5	1-2
22	Disable SJA1000	Enable SJA1000	1-2

With jumper JP20, the supply voltage for the motor driver L6202 is selected. In this there is the possibility to use a motor with higher supply voltage (12 Vdc power supply included in kit).

NOTE: *Jumper JP20 selects the supply voltage of the motor driver module L6202 from X3 (supply of reference design) or over contacts 3 and 4 of SV5.*

The supplied motor can get its supply voltage by the supplied power supply. See **5.6 Motor Driver L6202**.

CAUTION: *Observe limitations of current and voltage of the L6202 in surrounding temperature (U_{MAX} = 32 Vdc, I_{MAX} = 1.5 A). Additional capacitors and free-running diodes must possibly be used for other motors.*

Jumper 23 functions:

- Jumper 23 closed = CAN BUS termination with 120 Ω active
- Jumper 23 open = CAN BUS termination inactive

In a CAN bus system, terminated lines are used. So, at least at the beginning and at the end of the bus there should be a termination resistance of 120 Ω.

If every section of a bigger system is equipped with a terminal resistance it may lead to a greater bus load. Therefore, the terminal resistance can be switched on and off with jumper JP23 in the reference design.

In the application area, there are test points TP1 to TP4. Signals can be measured at these points. See **Table 5-7**.

Table 5-7. Table of Test Points (TP)

TP	MC68HC908GR8 Pinout	MC68HC908GP32 Pinout
1	PTB4 to POTI	PTB4 to POTI
2	PTB5 via R to L6202	PTB5 via R to L6202
3	NC	PTB1
4	NC	PTB0

At TP1 — The center tab of the potentiometer. The potentiometer is used for demonstration of an analog-to-digital (AD) transition. See **Figure 5-2**.

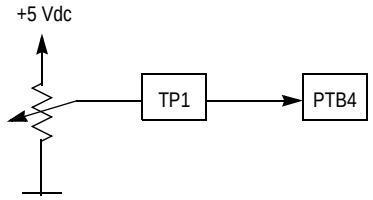


Figure 5-2. TP1 Example

At TP2 — **Figure 5-3** shows the possibility to operate the motor driver L6202 in a very optimal way.

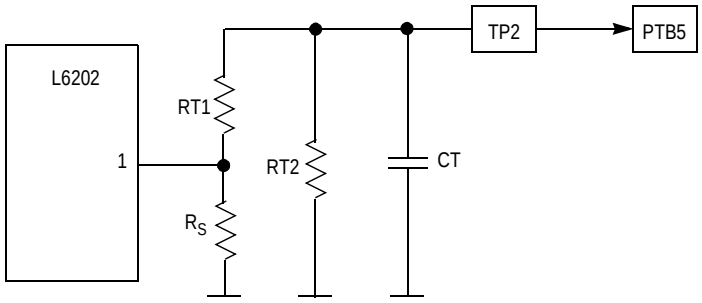


Figure 5-3. Motor Control Circuitry

At the sensor resistor (R_S) of the motor driver, a voltage is developed that is proportional to the motor current. This voltage is routed over the low pass filter (RT1 and CT) to the AD converter of the special controller. By control signals at the L6202, the current in the motor can be controlled using the sensed value.

5.6 Motor Driver L6202

This module is a bridge driver. It is fit-for-purpose to drive a brushed dc motor. The current inputs are TTL compatible, it uses a thermal protective circuit, and it can be operated with a clock frequency up to 100 kHz.

Generally, the bridge consists of four transistors that can be controlled by inputs. Short circuits in a vertical leg are disabled by an internal logic.

Table 5-8 shows the switching possibilities. The three conditions 1, 3, and 4 show the simple operation.

Table 5-8. Switching Possibilities

Condition	Enable	In2	In2	Output MOSFET
1	Low	X	X	All transistors off
2	High	Low	Low	Sink1, Sink2
3	High	Low	High	Sink1, Source2
4	High	High	Low	Source1, Sink2
5	High	High	High	Source1, Source2

In condition 1 no current flows through the motor. Without outer influences, it does not turn. In conditions 3 and 4, the current flows in different directions so that the reversal of direction is performed.

The motor current, with limitations on the driving speed, can be controlled by clocked switching at ENABLE. For more information, reference the documents listed in **2.3.1 MC68HC908GR8 Application Notes and Engineering Bulletins** and **2.4.1 MC68HC908GP32 Application Notes and Engineering Bulletins**.

Figure 5-4 shows the principal circuitry as demonstrated in the reference design.

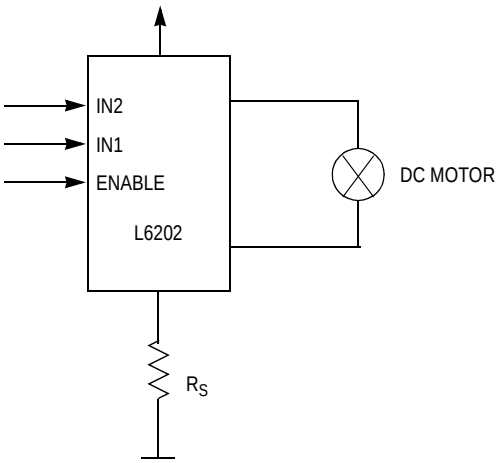


Figure 5-4. Motor Layout

Section 6. Working with the Reference Design

6.1 Contents

6.2 Introduction 55
6.3 Connecting the CPU to the CAN Controller 55

6.2 Introduction

This section describes how to work with the reference design.

6.3 Connecting the CPU to the CAN Controller

The reference design uses:

- MC68HC908GR8 with 21 input/output (I/O) lines
- MC68HC908GP32 with 33 I/O lines

Because of the different number of available I/O lines, there are two possibilities of coupling a standalone controller area network (CAN). The two standalone CAN controllers used are:

- MCP2510 (Microchip) — see **Figure 6-1**
- SJA1000 (Philips) — see **Figure 6-4**

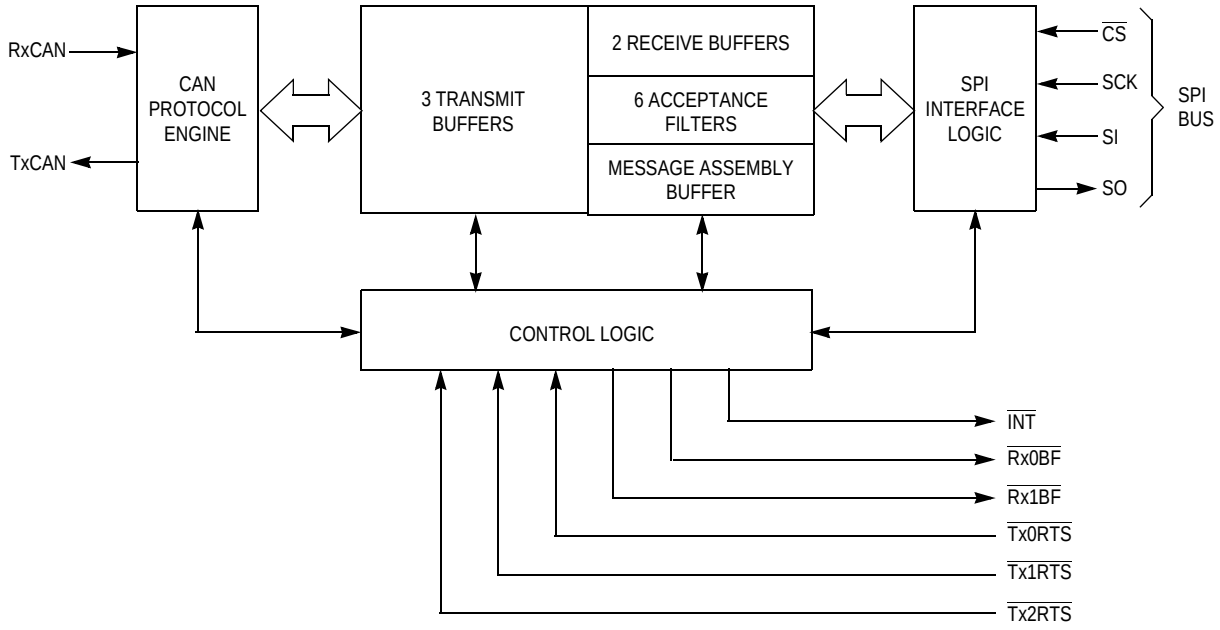


Figure 6-1. MCP2510 Block Diagram

The two microcontrollers (MCU) used, MC68HC908GR8 and MC68HC908GP32, can both drive the CAN controller using the serial peripheral interface (SPI) module. In this situation, the MC33388 is used as the CAN transceiver. See **Figure 6-2**.

NOTE: *The full specification for the MC33388 is reprinted in **Appendix D. MC33388 Motorola CAN Interface**.*

The block diagram in **Figure 6-3** shows the simpliest way of coupling the Motorola MCU to the MCP2510 with only four lines used. In this case, the MC68HC908GR8 and MC68HC908GP32 used a serial peripheral interface (SPI). It is possible to use an even less integrated MCU without an SPI by replicating this function in software. Three I/O lines of the microcontroller are used, controlled by jumpers, to choose the desired configuration and maximize the options of the MCP2510 and the MC33388. See **5.5 Jumpers in the Application Area**.

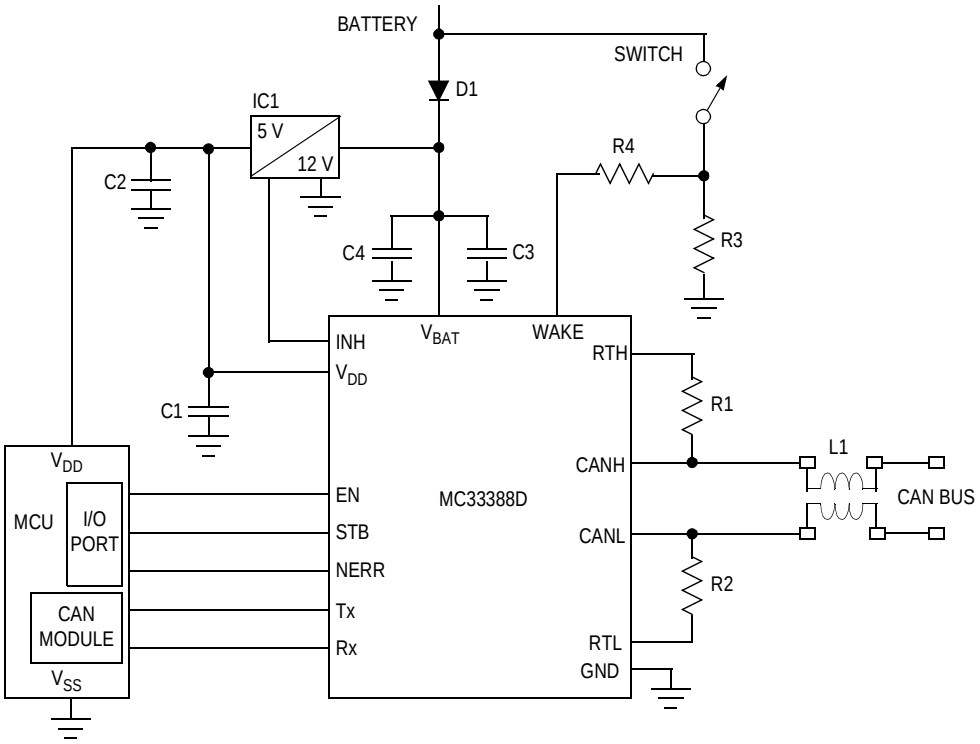


Figure 6-2. Block Diagram MC33388

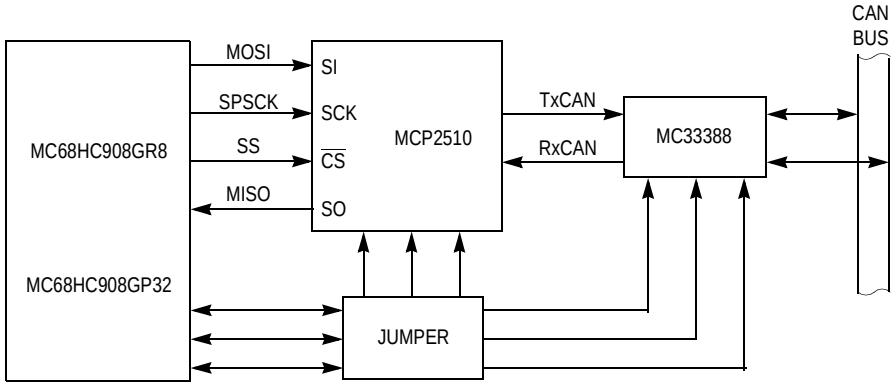


Figure 6-3. Block Diagram MC68HC908GR8 and MC68HC908GP32 + MCP2510 + MC33388

If the MC68HC908GP32 is used, the parallel Philips CAN controller SJA1000 can be used. See **Figure 6-4**.

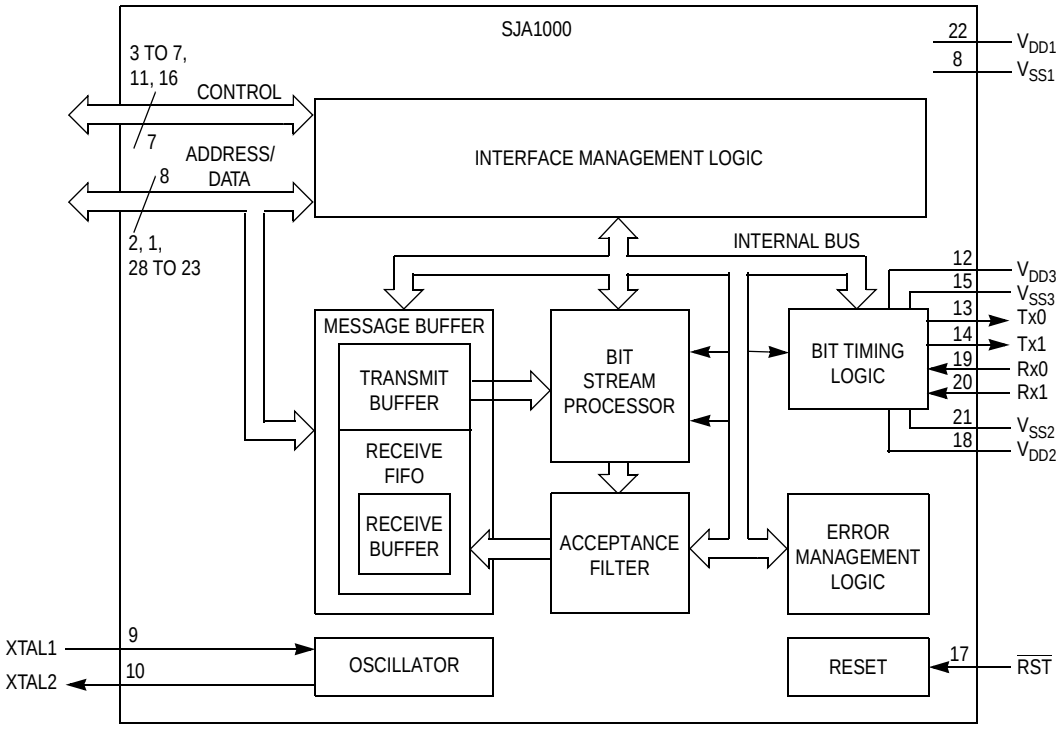


Figure 6-4. Block Diagram SJA1000

The connection between the MC68HC908GP32 and the SJA1000 is performed via a parallel interface. The SJA1000 has two different modes from the bus function:

- INTEL mode
- MOTOROLA mode

These modes can be set to the so called MOTOROLA mode by a MODE pin. In MOTOROLA mode the SJA1000 expects a multiplexed address and data bus and the control signals AS, WR, E, and CS. As the MC68HC980GP32 does not have these signals, they must be emulated by software and I/O pins.

The parallel interface to the SJA1000 is completed as shown in **Table 6-1** and **Table 6-2**.

Table 6-1. SJA1000 Control Bus

SJA1000 Signal	MC68HC908GP32 Signal	44-Pin QFP
AS	PTD0	Pin number 12
CS	PTD1	Pin number 13
E	PTD2	Pin number 14
WR	PTD2	Pin number 15

Table 6-2. SJA1000 Address and Data Bus

SJA1000 Signal	MC68HC908GP32 Signal	44-Pin QFP
AD0	PTA1	Pin number 33
AD1	PTA2	Pin number 34
AD2	PTA3	Pin number 35
AD3	PTA4	Pin number 36
AD4	PTA5	Pin number 37
AD5	PTA6	Pin number 38
AD6	PTB6	Pin number 28
AD7	PTB7	Pin number 29

The SJA1000 is controlled via an address and data bus (memory interface). Because the M68HC08 microcontrollers do not offer an address and data bus, the memory interface is emulated with I/O ports and the access control is done by software.

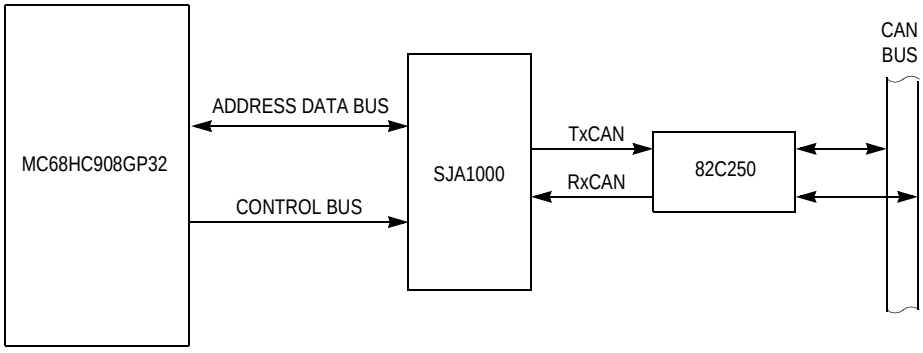


Figure 6-5. Block Diagram of M68HC908GP32 Connection to Philips SJA1000

Section 7. Circuit Description of the Monitor Mode Area

This section provides a circuit description of the monitor mode area.

- The reference design can be equipped with both the MC68HC908GR8 and the MC68HC908GP32.
- The input voltage for the reference design at X3 can be in the range of 8 to 18 Vac or Vdc.
- Capacitors C1 and C2 as well as C4 and C5 are used as filters for the input voltages.
- The light-emitting diode (LED) on top of voltage control (IC2) shows the presence of a +5 Vdc input voltage.
- The conversion from TTL to RS232 level for the monitor mode interface and SCI interface with RS232 performed by IC6 (MC145407).
- The switching of Rx and Tx signals at the monitor mode data interface is done by IC3 (74HC125).
- Jumpers 1, 3, 4, 5, and 6 define the necessary levels for entry into the monitor mode. In a later version by IC4 (not equipped) and IC1 (74HC4016), the use of static adjustments (JP 3, 4, 5, 6) of the monitor mode entry by the programmer will be realized.
- The signal levels defined by jumpers 3, 4, 5, and 6 are always connected to the microcontroller in the label version.
- A reset with IC4 will be possible by the monitor mode interface.
- A base socket is designated for operation of the microcontroller with an external crystal oscillator (supply version 9.8304 MHz).
- The controller can be put into a defined state by the reset push button.

- For jumper function, see **5.3 Jumpers in the Monitor Mode Area**
- All voltages and signals are available at connector SV1 (FB, 16 contacts).

For additional information regarding the monitor mode, refer to the appropriate section in these documents:

- *MC68HC908GR8 Advance Information*, Motorola order number MC68HC908GR8/D
- *MC68HC908GP32 Technical Data*, Motorola order number MC68HC908GP32/H

Section 8. Software Description

8.1 Contents

8.2 Introduction 63
8.3 CodeWarrior IDE 64
8.4 Module Description 65
8.4.1 MCU_Select.h 65
8.4.2 CAN Driver.C 68
8.4.3 MCU_Hardware_Description_HC08.h 69
8.4.4 MCU_Assembler_Instruction.h. 69
8.4.5 Hardware_HC08.c 69
8.4.6 IRQtab.c 69
8.4.7 CAN_Controller.h 70
8.4.8 CAN_Controller_SJA1000.C 70
8.4.9 CAN_Controller_MCP2510.C. 71
8.4.10 cptest.C 71

8.2 Introduction

The software offers the easiest possible introduction to a controller area network (CAN) bus environment. It was purposely arranged to be simple in order to easily keep the DIN overview. In a program loop in the module **`CPTEST10.C Application Main Function,`** both the CAN bus and the serial communications interface (SCI) are cyclically polled.

These functions are made available:

- Information from the CAN bus is sent over the SCI.
- Information from the SCI is sent over the CAN bus.
- In both cases, the message with a command interpreter is evaluated and additional functions are implemented.

The command interpreter knows these functions:

- Engine stop
- Engine run left
- Engine run right
- Send value of the analog-to-digital (AD) transducer over CAN

8.3 CodeWarrior IDE

The software (see **Appendix B. Flowcharts and Source Code Files**) was provided under CodeWarrior® and is divided into these modules:

- `MCU_Select.h`
- `CAN Driver.C`
- `MCU_Hardware_Descrition.h`
- `MCU_Hardware_Descrition_HC08.h`
- `MCU_Assembler_Instructions.h`
- `Hardware_HC08.c`
- `Hardware_HC08.h`
- `IRQtab.c`
- `CAN_Controller.h`
- `CAN_Controller_SJA1000.c`
- `CAN_Controller_MCP2510.c`
- `Cp10test.c`

NOTE: *The documentation differentiates between low speed and high speed. However, this does not refer to the actual speed on the CAN bus. Low speed is always used when the low-speed CAN transceiver MC33388 is used as the physical interface to the CAN bus. High speed is used when the CAN transceiver PCA82C250 is used to connect to the CAN bus.*

CodeWarrior is a registered trademark of Metrowerks, Inc., a wholly owned subsidiary of Motorola, Inc.

8.4 Module Description

This subsection describes the various modules.

8.4.1 MCU_Select.h

Three different programming models are possible depending on the hardware equipment being used. The selection takes place via a jumper in the hardware and the appropriate definitions in the **MCU_SELECT.H** module.

Possible functions to configure clock generation on the board are shown in **Table 8-1**.

Table 8-1. Clock Generation Configurations

Jumper	Functions 1 and 2	Functions 2 and 3	Default
1	IRQ = GND	IRQ = V_{TST}	1-2
10	External clock oscillator	32-kHz crystal on	2-3
14	32-kHz crystal off	32-kHz crystal on	2-3
16	V_{PS} to BAT MC33388	+5 V to BAT MC33388	2-3

The CAN reference design board can be used for software development in the monitor mode and as a standalone CAN node. The default jumper setting is standalone mode.

To configure the board to be used with monitor mode debugging:

1. Power down the board (no supply voltage)
2. Supply high voltage to the IRQ pin (set jumper 1 to position 2-3)
3. Select external clock oscillator (set jumper 10 to position 1-2)
4. Switch off the internal 32-kHz crystal (set jumper 14 to position 1-2)

To configure the board to be used as a standalone CAN node:

1. Power down the board (no supply voltage)
2. No test voltage on the IRQ pin (set jumper 1 to position 1-2)

3. Select internal 32-kHz crystal as the clock source (set jumper 10 to position 2-3)
4. Switch on the internal 32-kHz crystal (set jumper 14 to position 2-3)

The software can be complied for use with the MC68HC908GR8 or with the MC68HC908GP32 (both with the MCP2510 CAN controller). The compiling process is controlled via the define statements in the header file **MCU_SELECT.H**. See **Figure 8-1**.

To select the MC68HC908GR8 microcontroller, change these define statements in the **MCU_SELECT.H** header file:

```
#define GR8MCP TRUE     // Select MC68HC908GR8 and MCP2510
#define GP32MCP FALSE   // Select MC68HC908GP32 and MCP2510
#define GP32SJA TRUE    // Select MC68HC908GP32 and PCA82C250
```

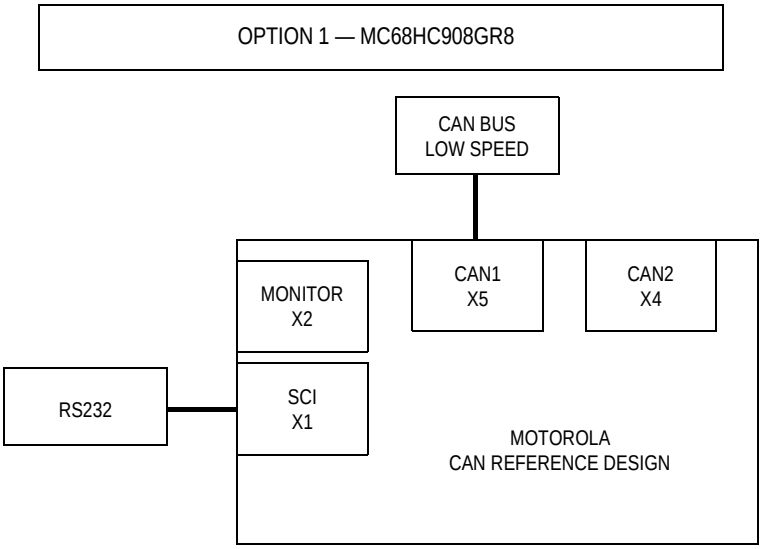


Figure 8-1. Option 1 — MC68HC908GR8 and MCP2510

Additionally, these jumpers are to be set:

- Jumper 15 in position 1-2 (enable MCP2510)
- Jumper 22 in position 1-2 (disable SJA1000)

The software can be complied for use with the MC68HC908GP32 and the MCP2510 CAN controller (see **Figure 8-2**). To select the MC68HC908GP32 microcontroller, change these define statements in the **MCU_SELECT.H** header file:

```
#define GR8MCP FALSE    // Select MC68HC908GR8 and MCP2510
#define GP32MCP TRUE    // Select MC68HC908GP32 and MCP2510
#define GP32SJA FALSE   // Select MC68HC908GP32 and PCA82C250
```

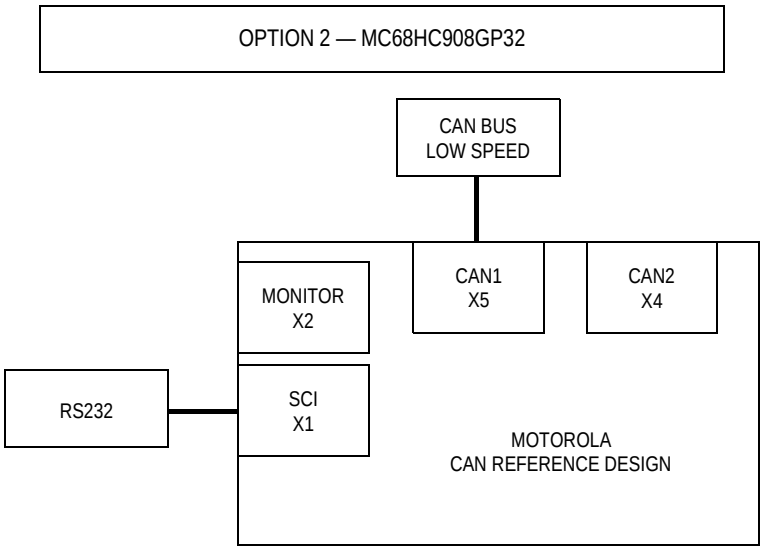


Figure 8-2. MC68HC908GP32 and MCP2510

Additionally, these jumpers are to be set:

- Jumper 15 in position 1-2 (enable MCP2510)
- Jumper 22 in position 1-2 (disable SJA1000)

The software can also be complied for use with the MC68HC908GP32 and the PCA82C250 CAN controller. See **Figure 8-3**.

NOTE: *It is not possible to use the MC68HC908GR8 with the PCA82C250.*

To select the MC68HC908GP32 microcontroller, change these define statements in the **MCU_SELECT.H** header file:

```
#define GR8MCP FALSE    // Select MC68HC908GR8 and MCP2510
#define GP32MCP FALSE   // Select MC68HC908GP32 and MCP2510
#define GP32SJA TRUE    // Select MC68HC908GP32 and PCA82C250
```

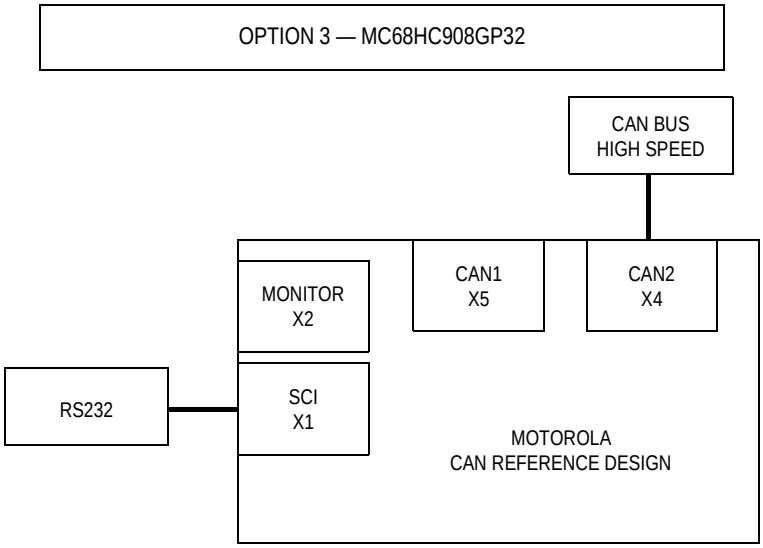


Figure 8-3. MC68HC908GP32 and SJA1000

Additionally, these jumpers are to be set:

- Jumper 15 in position 2-3 (disable MCP2510)
- Jumper 22 in position 2-3 (enable SJA1000)

8.4.2 CAN Driver.C

This file (**CAN_DRIVER.C CAN Driver Functions**) contains two functions:

- `Send_Message_Object()`
- `Receive_Message_Object()`

`Send_Message_Object()` sends a CAN message and `Receive_Message_Object()` receives a message over the CAN bus. Both functions use low-level driver functions which access the CAN controller.

8.4.3 MCU_Hardware_Description_HC08.h

This file (**MCU_Hardware_Description.H Hardware Description File**) describes the special-purpose registers of the MC68HC908GR8 and MC68HC908GP32. All addresses and bit definitions which are necessary to access the microcontroller can be found in this file.

8.4.4 MCU_Assembler_Instruction.h

It is a good programming practice not to mix high-level language C code and assembler instruction code. In this software, macros are used when programming in assembler. These macros are defined in this file (**MCU_ASSEMBLER_INSTRUCTIONS.H In-Line Assembler Commands**).

8.4.5 Hardware_HC08.c

In an embedded system, the microcontroller has to be configured before the application software is executed. All setups for the microcontroller like configuration (CONFIG), clock generation module (CGM and PLL), timer, synchronous and asynchronous serial interfaces (SPI and SCI), and analog-to-digital converter (ADC) are done in this file. See **HARDWARE_HC08.C Hardware and MCU Specific Routines**.

These functions are available to configure the MCU:

```
Ini_MCU()      // Call all setup functions
Ini_PLL()      // Set up the PLL
Ini_config()   // Set up the MCU configuration
                  (watchdog, LVI, etc.)
Ini_Timer()    // Initialize the timers
Ini_ADC()      // Initialize the ADC
Ini_SCI()      // Initialize the SCI
Ini_SPI()      // Initialize the SPI
                  (for use with the MCP2510 CAN controllers)
Ini_PORT()     // Specifies the control signals for the MC33388
                  (STP and EN)
```

8.4.6 IRQtab.c

In this file (**IRQTAB.C Interrupt Vectors**), the interrupt vectors for the microcontroller are defined.

8.4.7 CAN_Controller.h

In this file (**CAN_CONTROLLER.H CAN Controller Definitions**), all register definitions for the CAN controllers MCP2510 and SJA1000 are done.

8.4.8 CAN_Controller_SJA1000.C

The CAN controller SJA1000 is used only with the MC68HC908GP32. All routines related to the SJA1000 CAN controller are contained in this file (**CAN_CONTROLLER_SJA1000.C Software Routines for the CAN Controller SJA1000**)

These functions are available:

- `Initialize_CAN_Controller()`
- `Write_Byte` (address, DATA)
- `U08 Read_Byte` (address)
- `Bit_Set` (address, mask)
- `Bit_Clear` (address, mask)

The SJA1000 is a CAN controller which is controlled via a control and a multiplexed address and data bus. Ports and software on the MC68HC908GP32 microcontroller provide the interface to the bus. Therefore, specific ports must be configured. PORTD (bit 0 to bit 3) is used for the control bus of the SJA1000. PORTA (bit 1 to bit 6) and PORTB (bit 6 and bit 7) are used for the multiplexed address and data bus.

NOTE: *Because the reference design also offers debugging via the monitor mode not all pins of a port are available for general-purpose use. Therefore, to emulate the bus interface with port pins, a full 8-bit port cannot be used.*

To receive and send data to the CAN controller, a specific timing must be done. The CAN controller accepts commands from the MCU. These commands are transferred to the CAN controller with the functions `Write_Byte()` and `Read_Byte()`. The application software uses the driver supplied with the software and does not need to care about the handling of the CAN controller.

The function `Initialize_CAN_Controller()` initializes the SJA1000 for the basic CAN mode. Features of the basic CAN mode are:

- 11-bit identifier
- Baud rate of 20 Kbps with a 16-MHz crystal
- No message filter (all messages are received)

8.4.9 CAN_Controller_MCP2510.C

The CAN controller MCP2510.C (**CAN_CONTROLLER_MCP2510.C Software Routines for the CAN Controller MCP2510**) can be used either with the MC68HC908GP32 or with the MC68HC908GR8. These functions are available:

- `Initialize_CAN_Controller()`
- `SI_Write()`
- `SI_Read()`
- `SPI_BitMode()`
- `SPI_CAN_Start()`
- `SPI_read_status()`

The function `Initialize_CAN_Controller()` initializes the MCP2510 with these features:

- 11-bit identifier
- Baud rate of 20 Kbps with a 16-MHz crystal
- No message filtering (all messages are received)
- Only one receive (Rx) and transmit (Tx) buffer is used.

8.4.10 cptest.C

In this file, the main routine is placed. `Main()` is called after reset and all the necessary setup of the hardware and initialization of the peripherals is done here. A little demo program is implemented which sends messages over the CAN bus (identifier 0x7F and data “01234567”). All messages which are received on the SCI input are transmitted over the CAN bus.

Software Description

Appendix A. Bill of Materials and Schematic

This appendix provides:

- Bill of Materials — **Table A-1**
- Schematics for the Reference Design CAN:
  - Using MAX232 — **Figure A-1**
  - Using MC145487 — **Figure A-2**

NOTE: *New versions of the reference design hardware will use the MC145487 RS232 interface as shown in **Figure A-2**; however, some include the MAX232 RS232 interface as shown in **Figure A-1**.*

Table A-1. Bill of Materials (Sheet 1 of 6)

Position	Part	Device	Description	Package	Supplier
1	IC	74C4016		DIL14	
2	IC2	78L05	5V Voltage Regulator	T0220	
3	IC3	74HC125		DIL14	
4	IC4	IBD1	Delay Circuit	DIL8	
5	IC5	MC33064	Reset Controller	TO92	Motorola
6	IC6	MC145407	RS232 Driver	DIL20	Motorola
7	IC7	68HC08GR8	Microcontroller	QFP32	Motorola
8	IC8	MCP2510	CAN Controller	SIOC18	Microchip
9	IC9	68HC08GP32	Microcontroller	QFP44	Motorola
10	IC10	MC33388	CAN Transwithter	SO14	Motorola
11	IC11	L6201	Powerbridge	DIL18	ST
12	IC12	SJA1000	CAN Controller	SO28	Phillips
13	IC13	80C250	CAN Transwithter	SO8	Phillips
14	GL1	B80C1500	Rectifier		
15					
16	X1	DSUB9 Buchse	SCI Interface		
17	X2	DSUB9 Buchse	Debug-Interface		
18	X3	Buchse	Power Supply		
19	X4	DSUB9 Pine	CAN 2		
20	X5	DSUB9 Pine	CAN 1		
21	SV1	16pol Wanne			
22	SV2	16pol Wanne			
23	SV3	26pol Wanne			
24	SV4	26pol Wanne			
25	SV5	4 Pol Klemme			
26					
27	OSZI1	Sockel		DIL14	
28	OSZI1a	9,8304MHz	Crystal Oscillator	DIL14	
29	Q1	16MHz	Crystal	HC18	
30	Q2	32kHz	Crystal	L8mm D3mm	
31	Q3	16MHz	Crystal	HC18	

Table A-1. Bill of Materials (Sheet 2 of 6)

Position	Part	Device	Description	Package	Supplier
32	Q1S	2pol	Socket		
33	Q3S	2pol	Socket		
34	F1	1A T	Fuse		
35	F2	2A T	Fuse		
36	F1S	2pol	Socket	Rund RM5.08	
37	F2S	2pol	Socket	Rund RM5.08	
38	R1	10k	Resistor	1/4W	
39	R2	10k	Resistor	1/4W	
40	R3	10k	Resistor	1/4W	
41	R4	0	Wire Bridge		
42	R5	10k	Resistor	1/4W	
43	R6	10k	Resistor	1/4W	
44	R7	10k	Resistor	1/4W	
45	R8	10k	Resistor	1/4W	
46	R9	330k	Resistor	1/4W	
47	R10	0	Resistor	1/4W	
48	R11	1k	Resistor	1/4W	
49	R12	10k	Resistor	0805	
50	R13	10M	Resistor	1206	
51	R14	330k	Resistor	0805	
52	R15	22k	Resistor	0805	
53	R16	10k	Resistor	0805	
54	R17	120	Resistor	0805	
55	R18	1k	Resistor	0805	
56	R19	1k	Resistor	0805	
57	R20	0	Wire Bridge		
58	R21	0	Wire Bridge		
59	R22	0			
60	R23	22k	Resistor	1/4W	
61	R24	1k	Resistor	1/4W	
62	R25	10k	Resistor	1/4W	
63	R26	0	Wire Bridge		

Table A-1. Bill of Materials (Sheet 3 of 6)

Position	Part	Device	Description	Package	Supplier
64	R27	0,1	Resistor	1/4W	
65	R28	120	Resistor	1/4W	
66	R29	470	Resistor	1/4W	
67	R30	470	Resistor	1/4W	
68	R31	470	Resistor	1/4W	
69					
70					
71					
72	C1	220µF 25V	Electrolytic Capacitor D8 H11	RM3,5	
73	C2	100nF 40V	Ceramic Capacitor	RM2,5	
74	C3	100nF 40V	Ceramic Capacitor	RM2,5	
75	C4	220µF 25V	Electrolytic Capacitor D8 H11	RM3,5	
76	C5	100nF 40V	Ceramic Capacitor	RM2,5	
77	C6				
78	C7	10µF 25V	Electrolytic Capacitor D5 H11	RM3,5	
79	C8	10µF 25V	Electrolytic Capacitor D5 H11	RM3,5	
80	C9				
81	C10	10µF 25V	Electrolytic Capacitor D5 H11	RM3,5	
82	C11	10µF 25V	Electrolytic Capacitor D5 H11	RM3,5	
83	C12	10µF 25V	Electrolytic Capacitor D5 H11	RM3,5	
84	C13	100nF 40V	Ceramic Capacitor	0805	
85	C14	100nF 40V	Ceramic Capacitor	0805	
86	C15	10nF 40V	Ceramic Capacitor	0805	
87	C16	33nF 40V	Ceramic Capacitor	0805	
88	C17	22pF 40V	Ceramic Capacitor	0805	
89	C18	22pF 40V	Ceramic Capacitor	0805	
90	C19	22pF 40V	Ceramic Capacitor	0805	
91	C20	22pF 40V	Ceramic Capacitor	0805	
92	C21	100µF 25V	Electrolytic Capacitor D5 H11	RM3,5	
93	C22	100µF 25V	Ceramic Capacitor	0805	
94	C23				
95	C24	n.c			

Table A-1. Bill of Materials (Sheet 4 of 6)

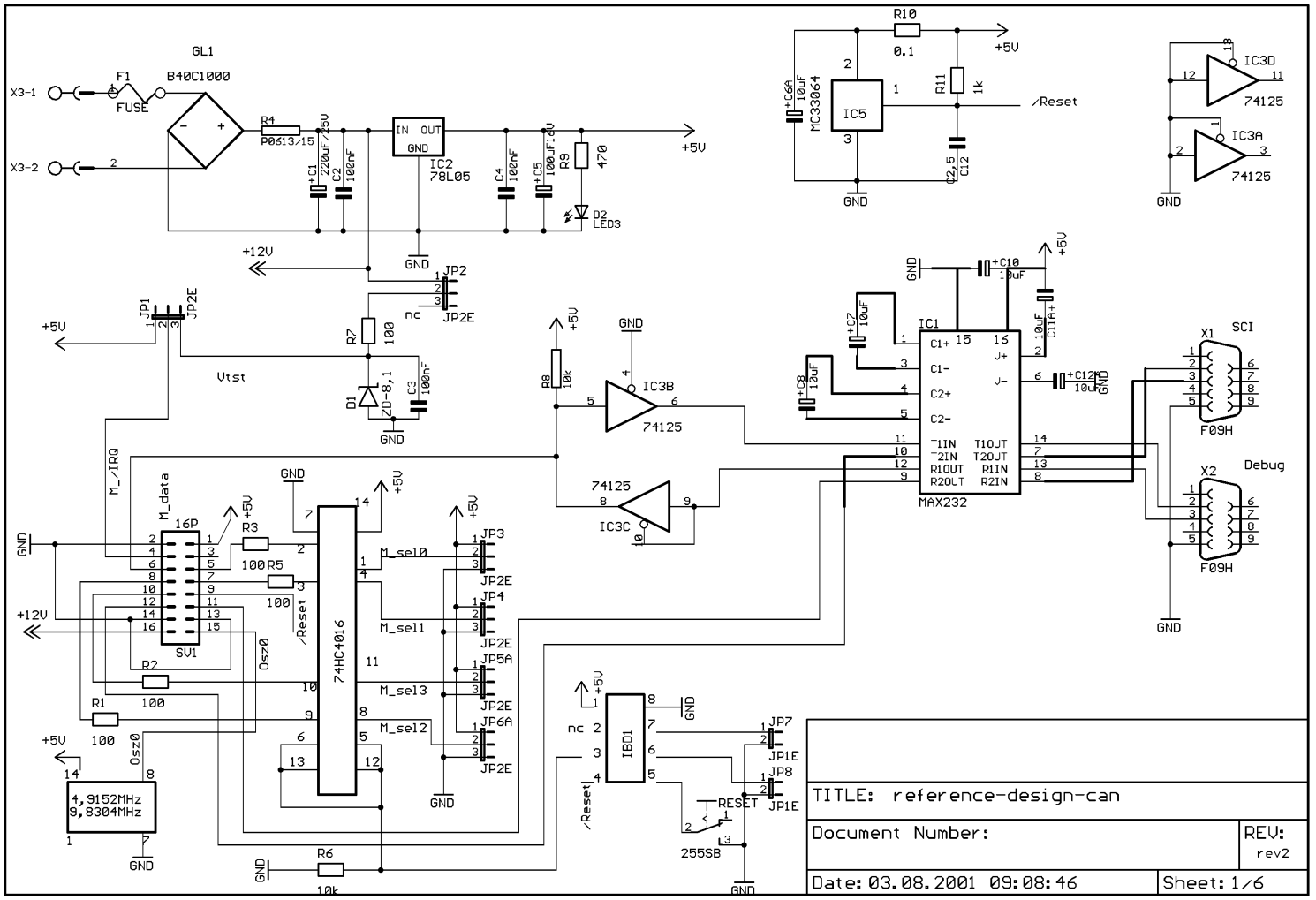
Position	Part	Device	Description	Package	Supplier
96	C25	100µF 16V	Ceramic Capacitor	RM2,5	
97	C26	0			
98	C27	0			
99	C28	330µF 35V	Electrolytic Capacitor D10 H14	RM5	
100	C29	100nF 40V	Ceramic Capacitor	RM2,5	
101	C30	10nF 40V	Ceramic Capacitor	RM2,5	
102	C31	0			
103	C32	10nF 40V	Ceramic Capacitor	RM2,5	
104	C33	10nF 40V	Ceramic Capacitor	RM2,5	
105	C34	22pF 40V	Ceramic Capacitor	RM2,5	
106	C35	22pF 40V	Ceramic Capacitor	RM2,5	
107	BC1	100nF 40V	Ceramic Capacitor	RM2,5	
108	BC2	100nF 40V	Ceramic Capacitor	RM2,5	
109	BC3	100nF 40V	Ceramic Capacitor	0805	
110	BC4	100nF 40V	Ceramic Capacitor	0805	
111	BC5	100nF 40V	Ceramic Capacitor	RM2,5	
112					
113					
114					
115	D1	ZD8,1	Zener Diode	RM10	
116	D2	LED	Green	D 3mm RM2,5	
117	D3	1N4148	Diode	Minimelf	
118	D4	1N4148	Diode	Minimelf	
119	D5	1N4004	Diode	RM10	
120	D6	1N4004	Diode	RM10	
121	D7	1N4004	Diode	RM10	
122	D8	LED	Red	D3 RM2,5	
123	D9	LED	Red	D3 RM2,5	
124	D10	LED	Red	D3 RM2,5	
125	P1	1k	Potentiometer		
126					
127	Kabel 1	16 AWG	Flat Cable with 2 Connectorn	RM1,27	

Table A-1. Bill of Materials (Sheet 5 of 6)

Position	Part	Device	Description	Package	Supplier
128	Kabel 2	26 AWG	Flat Cable with 2 Connectorn	RM1,27	
129					
130	Printed Circuit Board (PCB)	4 Layer	Printed Circuit Board (PCB), Solder Stop, Text	100*200mm	
131					
132	SV5		Phoenix Connector		
133					
134	S1		Taster		
135	S2		Taster		
136	S3		Taster		
137	S4		Taster		
138	S5		Taster		
139					
140	JP1	3pol	Pin		
141	JP2	3pol	Pin		
142	JP3	3pol	Pin		
143	JP4	3pol	Pin		
144	JP5	3pol	Pin		
145	JP6	3pol	Pin		
146	JP7	2pol	Pin		
147	JP8	2pol	Pin		
148	JP9	0			
149	JP10	3pol	Pin		
150	JP11	3pol	Pin		
151	JP12	3pol	Pin		
152	JP13	3pol	Pin		
153	JP14	3pol	Pin		
154	JP15	3pol	Pin		
155	JP16	3pol	Pin		
156	JP17	0			
157	JP18	3pol	Pin		
158	JP19	3pol	Pin		

Table A-1. Bill of Materials (Sheet 6 of 6)

Position	Part	Device	Description	Package	Supplier
159	JP20	3pol	Pin		
160	JP21	3pol	Pin		
161	JP22	3pol	Pin		
162	JP23	2pol	Pin		
163	JP24	3pol	Pin		
164	JP25	3pol	Pin		
165	JP26	3pol	Pin		
166	TP1	1pol	Pin		
167	TP2	1pol	Pin		
168	TP3	1pol	Pin		
169	TP4	1pol	Pin		
170	TP5	1pol	Pin		
171	TP6	1pol	Pin		
172	TP7	1pol	Pin		
173	X32a	8pol	Pin		
174	X32b	8pol	Pin		
175	X32c	8pol	Pin		
176	X32d	8pol	Pin		
177	XB32a	8pol	Connector		
178	XB32b	8pol	Connector		
179	XB32c	8pol	Connector		
180	XB32d	8pol	Connector		
181	X44a	11pol	Pin		
182	X44b	11pol	Pin		
183	X44c	11pol	Pin		
184	X44d	11pol	Pin		
185	XB44a	11pol	Connector		
186	XB44b	11pol	Connector		
187	XB44c	11pol	Connector		
188	XB44d	11pol	Connector		
189					
190	Jumper	25Stck			



Note:

Differences between this schematic and that shown in **Figure A-2** are confined to Sheet 1. Refer to **Figure A-2** for Sheets 2 through 6.

Figure A-1. Sheet 1 of CAN Reference Design Schematic with MAX232

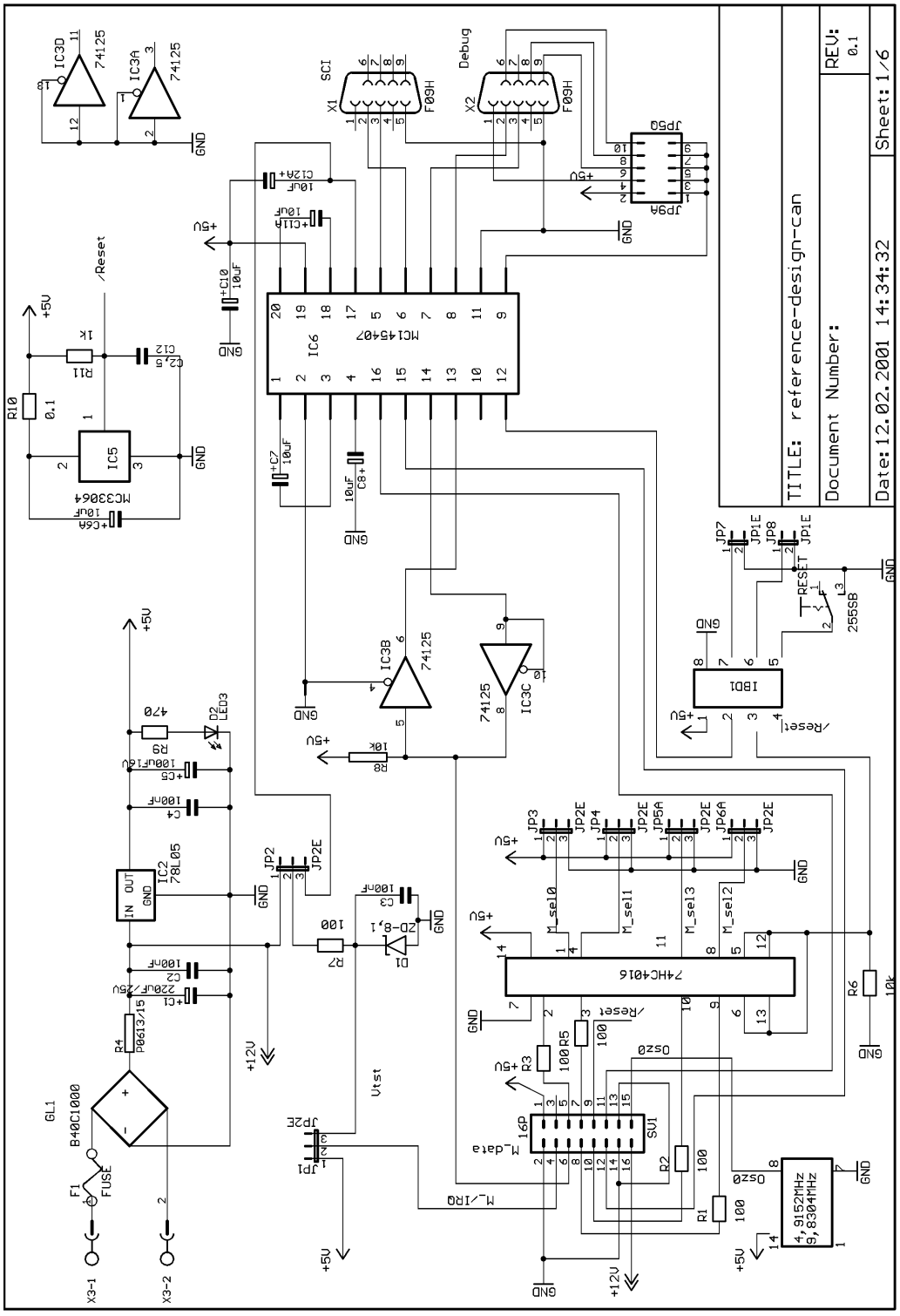


Figure A-2. CAN Reference Design Schematic with MC145487 (Sheet 1 of 6)

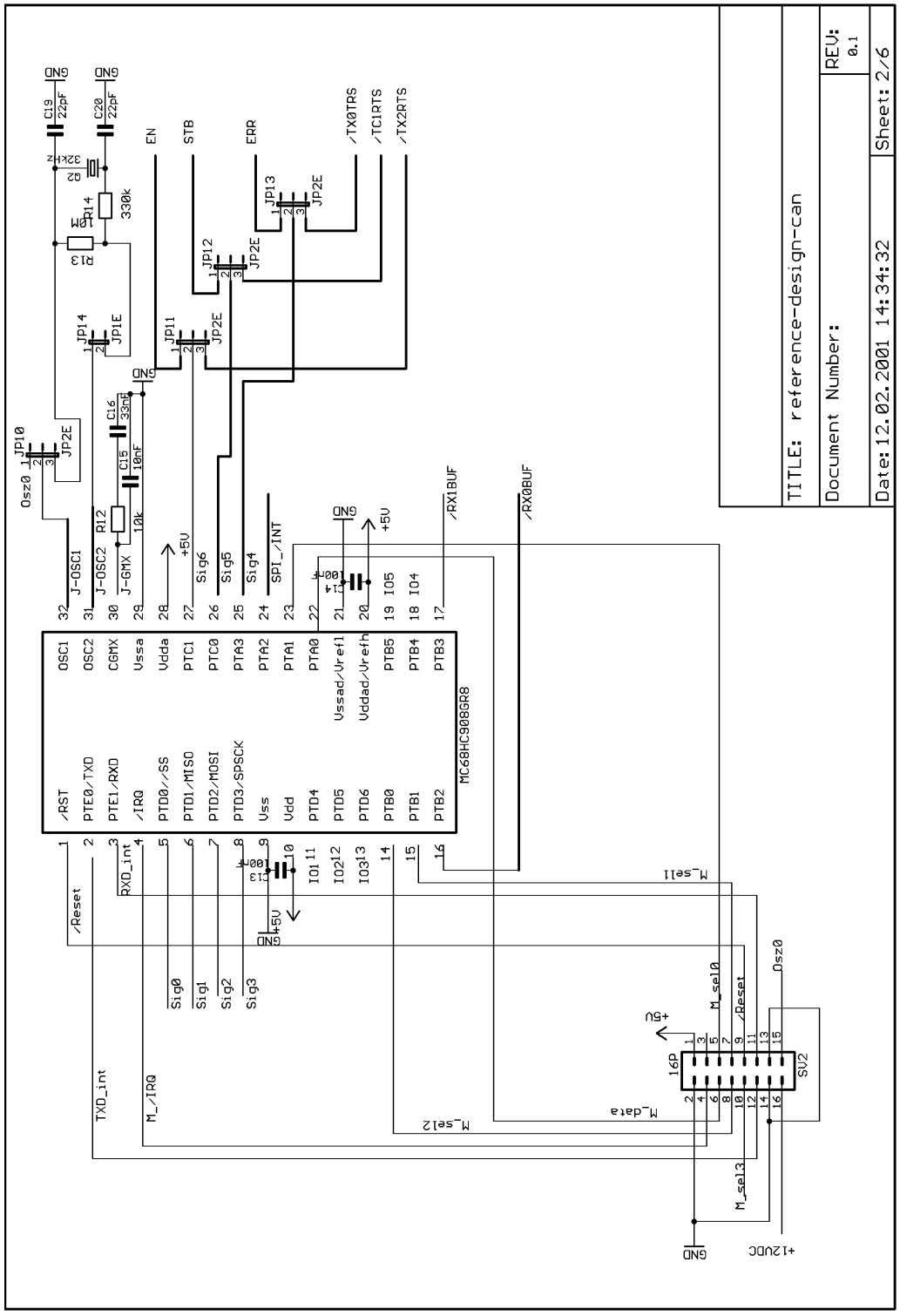


Figure A-1. CAN Reference Design Schematic with MC145487 (Sheet 2 of 6)

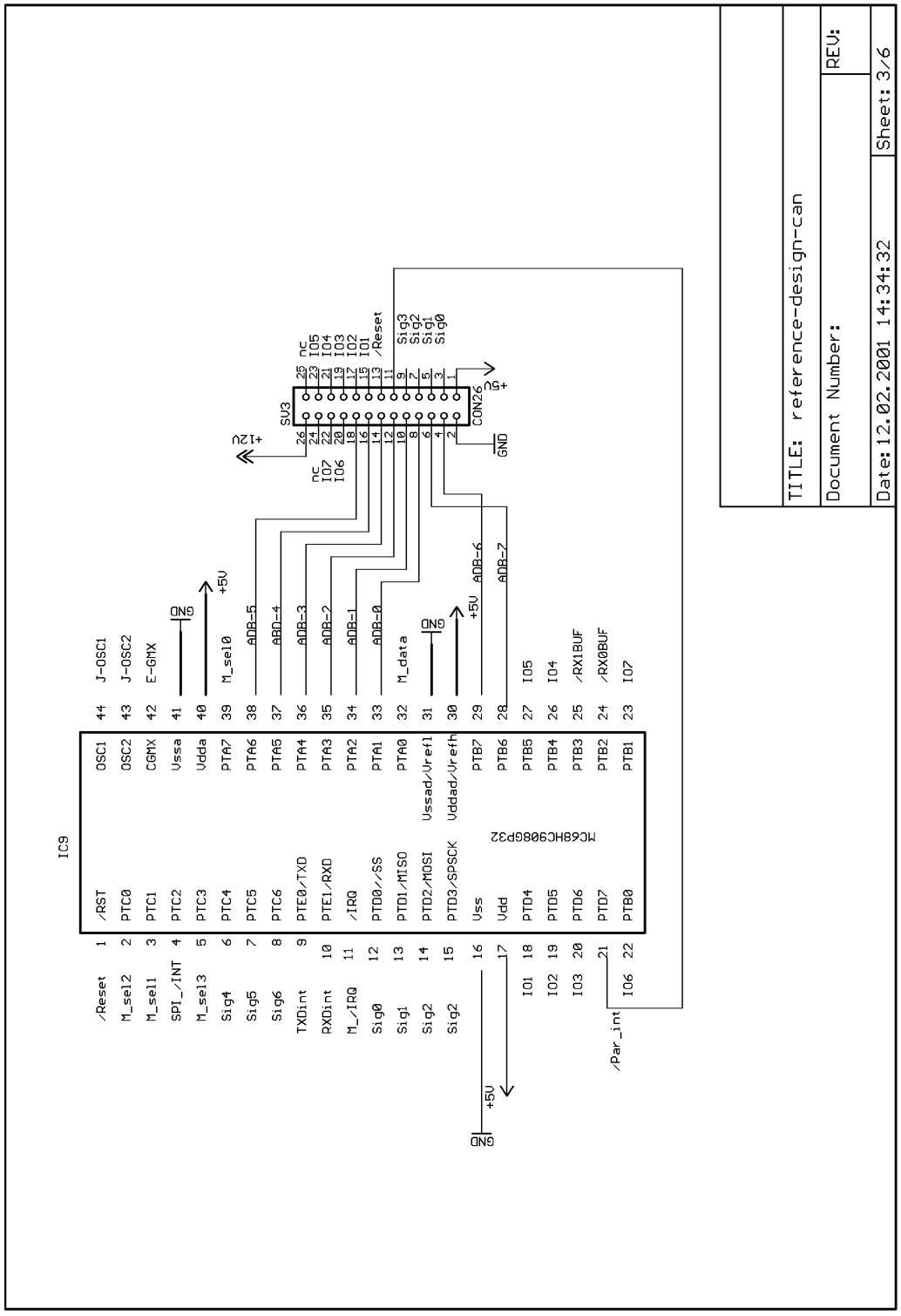


Figure A-1. CAN Reference Design Schematic with MC145487 (Sheet 3 of 6)

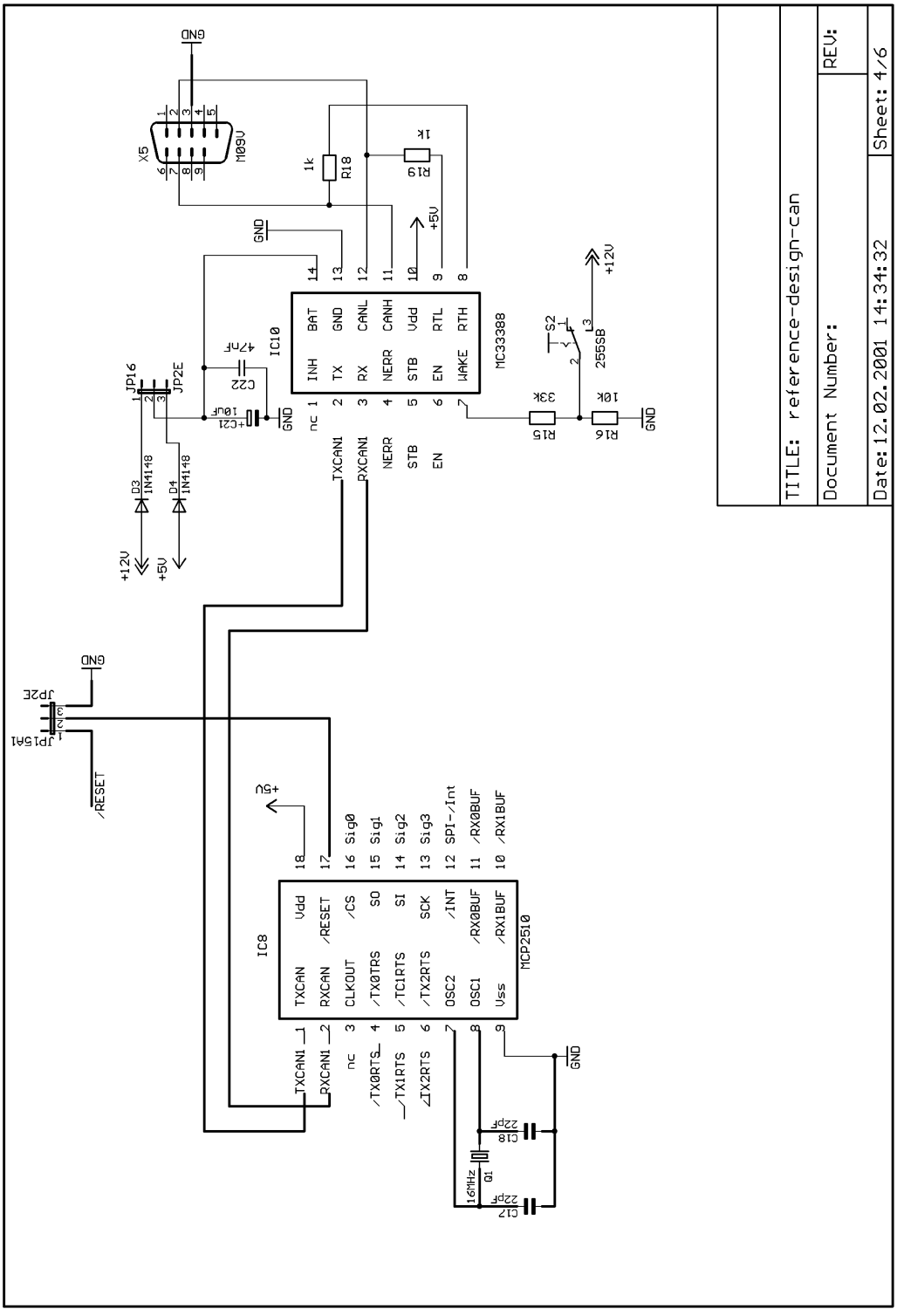


Figure A-1. CAN Reference Design Schematic with MC145487 (Sheet 4 of 6)

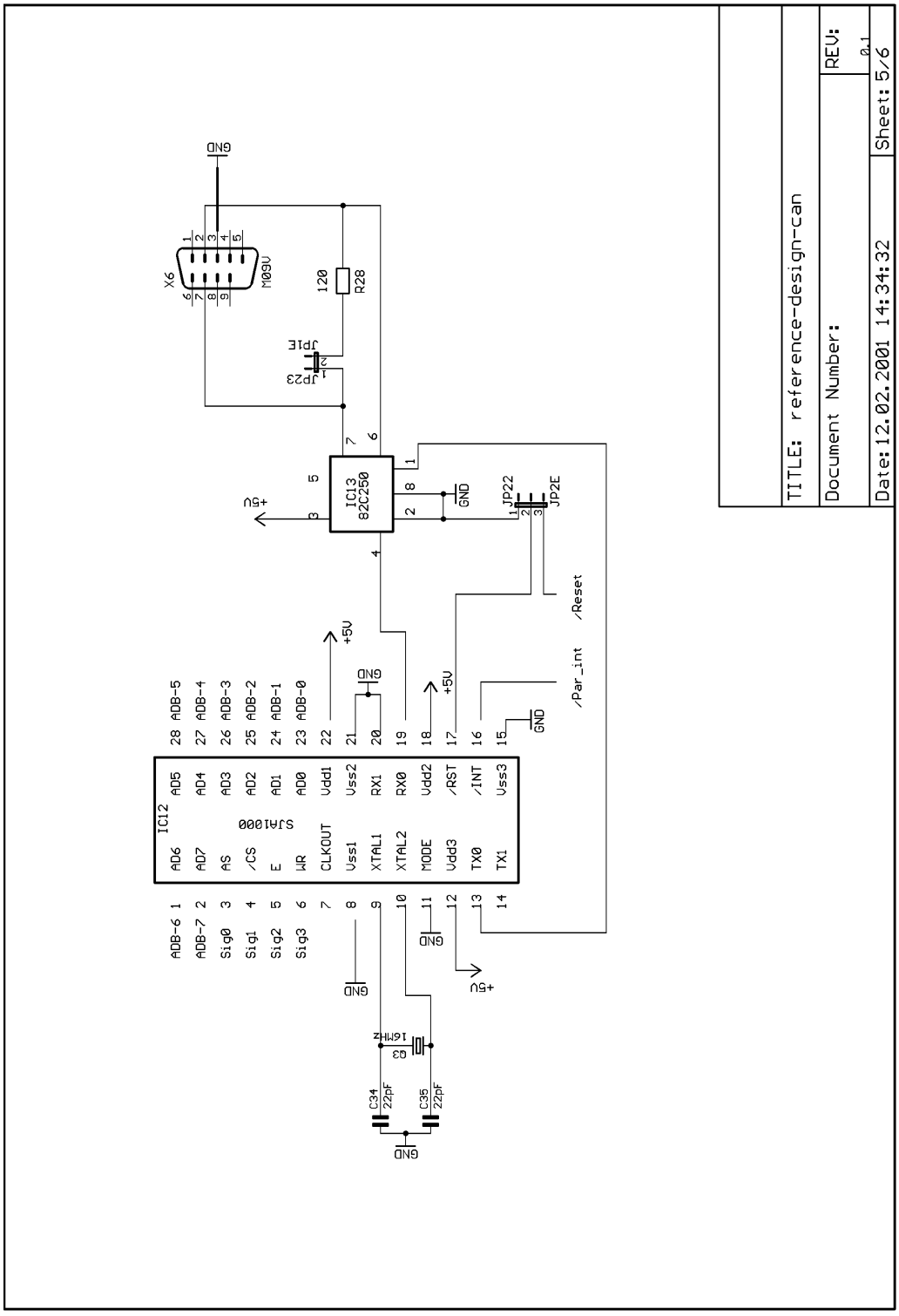


Figure A-1. CAN Reference Design Schematic with MC145487 (Sheet 5 of 6)

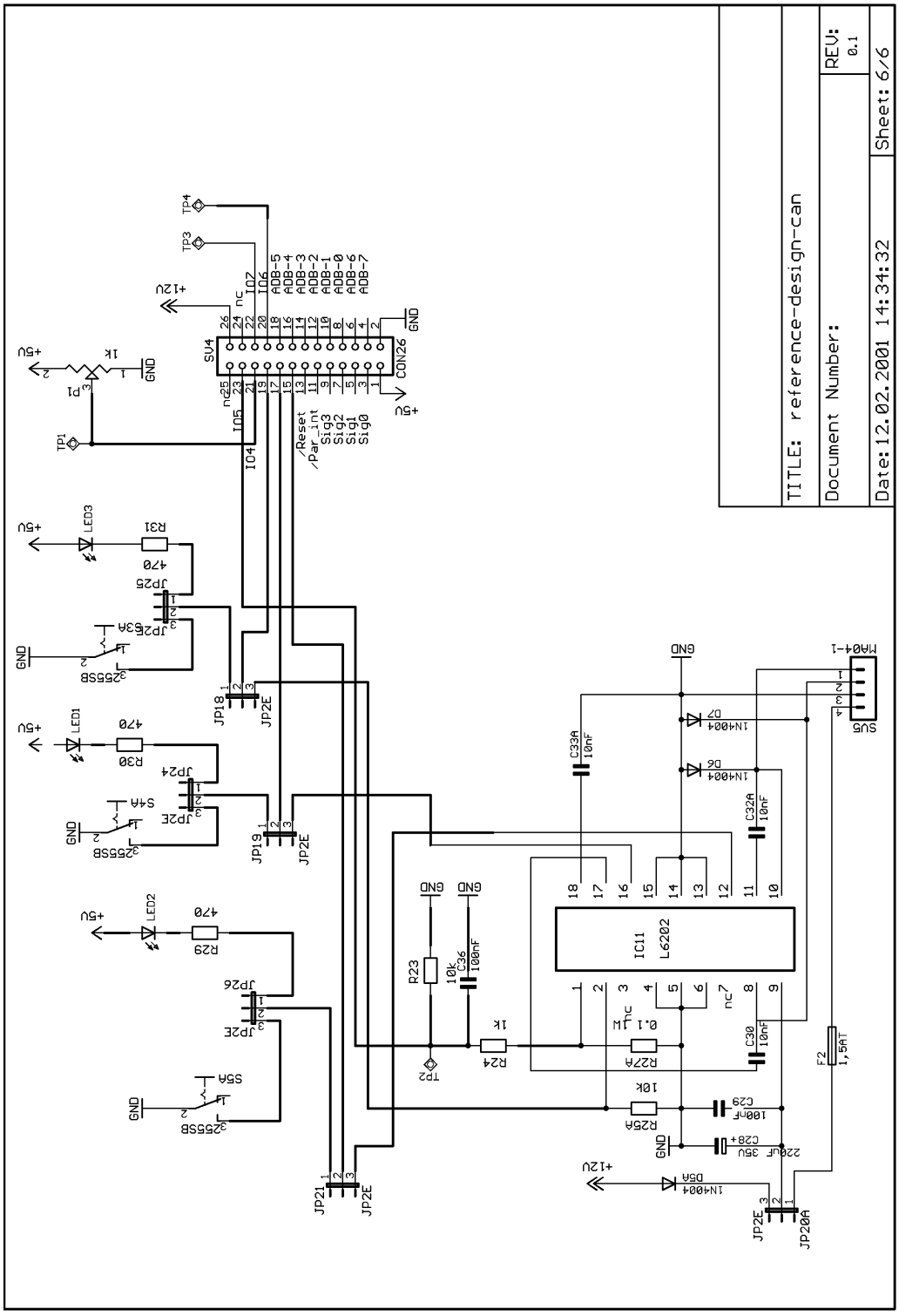


Figure A-1. CAN Reference Design Schematic with MC145487 (Sheet 6 of 6)

Appendix B. Flowcharts and Source Code Files

B.1 Contents

B.2 Introduction 87
B.3 Flowcharts 88
B.4 Source Code Files 91
CPTEST10.C — Application Main Function 91
HARDWARE_HC08.C — Hardware and MCU Specific Routines 95
CAN_DRIVER.C — CAN Driver Functions. 103
CAN_CONTROLLER_MCP2510.C — Software Routines for the CAN Controller MCP2510 107
CAN_CONTROLLER_SJA1000.C — Software Routines for the CAN Controller SJA1000 112
IRQTAB.C — Interrupt Vectors. 116
Include Files 118
MCU_SELECT.H 118
Select the Hardware Configuration. 118
HARDWARE_HC08.C — Register and Bit Definitions for the 68HC08 119
MCU_Hardware_Description.H — Hardware Description File 139
MCU_ASSEMBLER_INSTRUCTIONS.H — In-Line Assembler Commands 140
CAN_CONTROLLER.H — CAN Controller Definitions 141
mcp2510.h — Include File for MPC2510 Registers 147

B.2 Introduction

This section will provide both flowcharts for the CAN reference design (see **Figure B-1**) followed by the source code files.

B.3 Flowcharts

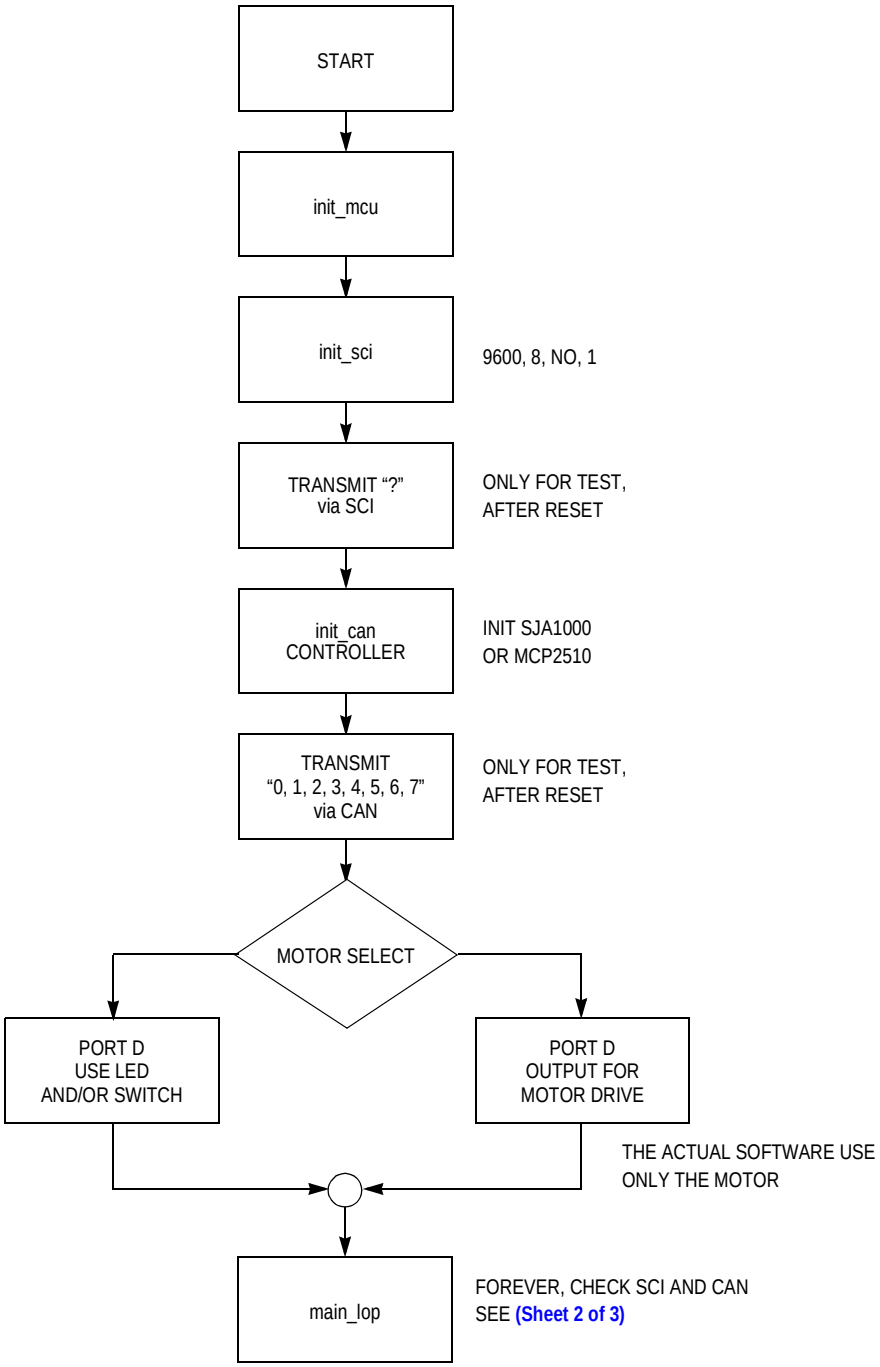


Figure B-1. CAN Reference Design Flowchart (Sheet 1 of 3)

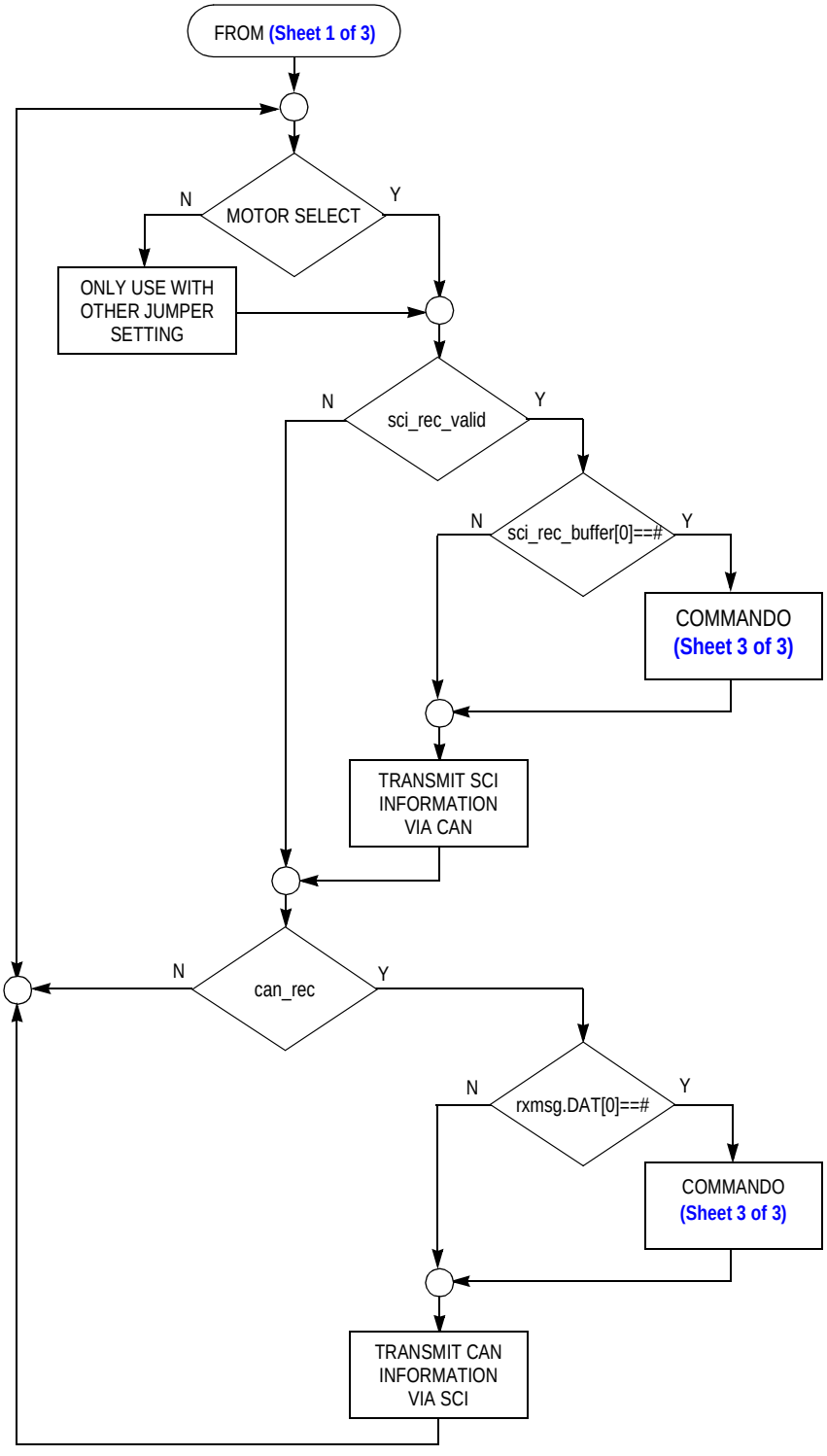


Figure B-1. CAN Reference Design Flowchart (Sheet 2 of 3)

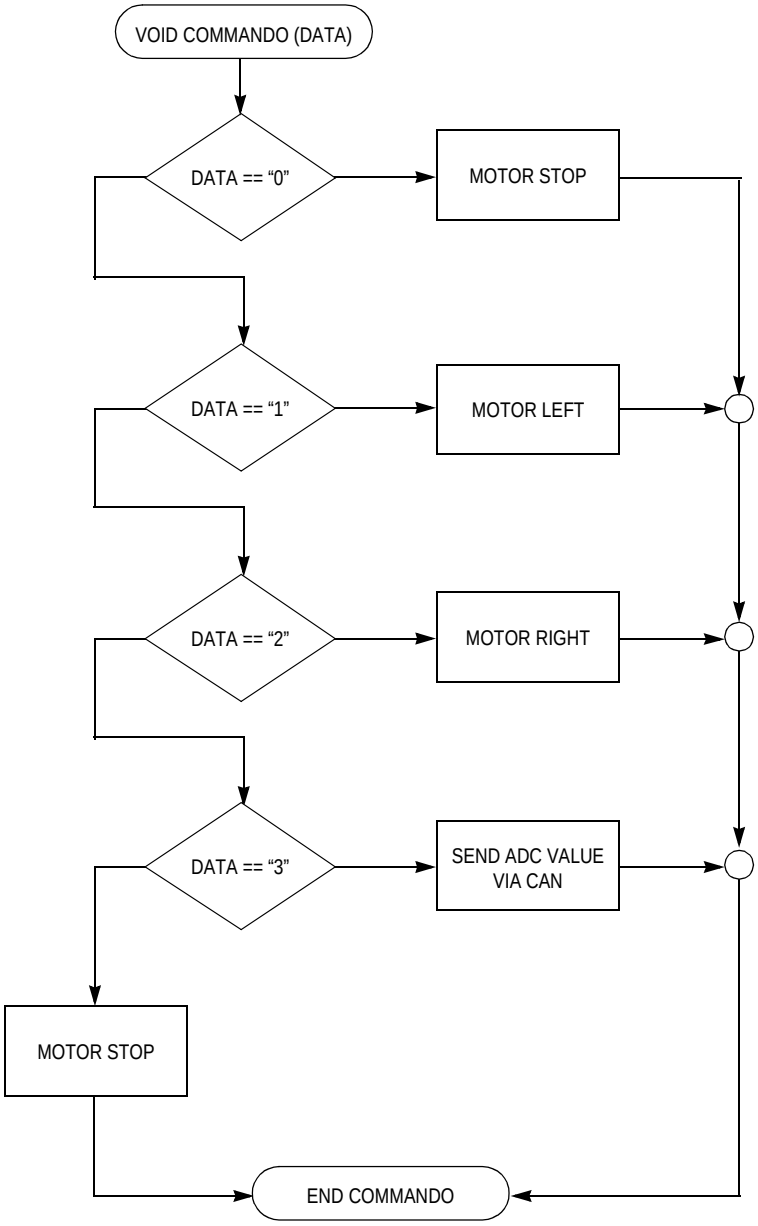


Figure B-1. CAN Reference Design Flowchart (Sheet 3 of 3)

B.4 Source Code Files

CPTEST10.C
Application Main Function

```
/*****************************************************************************/
/* File: cp_test10.c                                                         */
/*                                                                           */
/* Description: Main-Routines for CAN Reference design                       */
/* Author: Peter Dilger, ib.dilger@t-online.de                               */
/*    Harald Kreidl, Harald.Kreidl@motorola.com                              */
/*                                                                           */
/* ------------------------------------------------------------------------  */
/*                                                                           */
/* Version  Description                                        Date          */
/* ------------------------------------------------------------------------  */
/* 1.0      Initial version                                    20.02.2001    */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/* Project Includes                                                          */
/*****************************************************************************/

#include "cpmsg.h"
#include "MCU_Assembler_Instructions.h"
#include "MCU_Hardware_Description_HC08.h"
#include "MCU_Hardware_Description.h"
#include  "CAN_Controller.h"

//#include <stdlib.h>

TCANMsgtxmsg,rxmsg;

/*****************************************************************************/
/* Function void error( void )                                               */
/*****************************************************************************/

void error( void )
{}

/*****************************************************************************/
/* Function void transmit_ADC( void )                                        */
/*****************************************************************************/

void transmit_ADC( void )
{
   _U08  temp,tmp,i;
   _U16 tmp_id = 0;
   txmsg.DATA[0]= ADR;
   txmsg.LENGTH = 1;
   tmp_id = ADR;
   tmp_id &= 0x007f;
   txmsg.ID = tmp_id;
   temp = ADR;
```

```
   do                                // delay with ADC-value
   {  temp--;
      tmp = 0;
      for ( i=0;i<100; i++) tmp++;
   }
   while ( temp !=0);
   temp= Send_Message_Object( &txmsg );      // transmit ADC-value
}

/*****************************************************************************/
/* Function void commando( void )                                            */
/*****************************************************************************/

void commando ( _U08 motorfunction )
{  switch( motorfunction )
   {  case 0x30:
         PTD &= ~PTD6;                /* motor stop */
         break;
      case 0x31:
         PTD &= ~PTD5;
         PTD |= PTD4+PTD6;
         break;
      case 0x32:
         PTD &= ~PTD4;
         PTD |= PTD5+PTD6;
         break;
      case 0x33:
         transmit_ADC();
         break;
      default:
         PTD &= ~PTD6;                /* motor stop */
   }  /* switch */
}  /*commando*/

/*****************************************************************************/
/* Function void main( void )                                                */
/*****************************************************************************/

void main(void)
{  _U08  Status,temp, temp1,temp2;
   _U08  status;
   _U08  i=0;
   _U08  can_transbit=0;
   _U08  sci_tx_status=0;
   _U08  sci_tx_bit=0;
   _U08  sci_trans_status=0;
   _U08  *sci_tx_pointer;
   _U08  sci_tx_buffer[9];
   _U08  sci_test=0;
   _U08  led_status=FALSE;

/* for demo with dc-motor the motorselct is true, dont use the led and switch */
   _U08motor_select=TRUE;           // for demo=TRUE

   _U08  sci_rx_index=0;
   _U08  sci_rx_buffer[9];
   _U08sci_rec_valid = FALSE;
```

```
   txmsg.ID= 0x007f;
   txmsg.RTR= 0x00;
   txmsg.LENGTH= 8;
   txmsg.DATA[0]= 0x30;
   txmsg.DATA[1]= 0x31;
   txmsg.DATA[2]= 0x32;
   txmsg.DATA[3]= 0x33;
   txmsg.DATA[4]= 0x34;
   txmsg.DATA[5]= 0x35;
   txmsg.DATA[6]= 0x36;
   txmsg.DATA[7]= 0x37;

   //--- start the hardware initialization routine ---------------------
   Ini_mcu();        // Initialise the 908GP32 or GR8
   Ini_SCI();        // Initialise SCI
   temp = SCS1;
   temp = SCDR;
   temp = '?';       // test SCI
   SCDR = temp;

   Initialize_CAN_Controller();           // Initialise the SJA1000 or MCP2510

   temp2= Send_Message_Object( &txmsg );  // test messsage CAN

/* for demo with dc-motor the motorselct is 1, dont use the led and switch */
   if ( !motor_select )
   {  /* PTD 4  Out on led 3, PTD5 input S2,
      PTD6 input S1  */
      PTD |= PTD4;            /* led off        */
      DDRD |= DDRE4;          /* Led 3 on PTD4  */
      PTDPUE |= PTDPUE5 +PTDPUE6;
   }
   else  /*  motor active*/
   {  DDRD |= DDRE4+DDRE5+DDRE6;    /* PTD4,5,6 output for motordrive      */
   }
   while( 1 )  /* forever */
   {  if (!motor_select)              /* switch and leds                  */
      {  if (!( PTD & 0x20))          /* s3 mask                          */
         {  if (led_status)
            {  led_status =FALSE;
               PTD |= PTD4;           /* led off                          */
            }
            else
            {  led_status = TRUE;
               PTD &= ~PTD4; /* led on */
            }
            while (!( PTD & 0x20));    /* Release button                  */
         }                             /* Push Button S3                  */
         if (!( PTD & 0x40))           /* s4 mask                          */
         {  transmit_ADC();            /* Tramsmitt ADC value              */
         while (!( PTD & 0x40));       /* Remove S4                        */
      }                                /* Push button S4                        */
   }  /* End switch and leds */
/* Use the DC motor                                                        */
   if (sci_rec_valid)                 /* New message from SCI?            */
   {  if(sci_rx_buffer[0] == '#')     /* Check command                    */
      { commando( sci_rx_buffer[1] );
      }
```

```
/*  Alway transmit message via CAN                                          */
      i = 0;
      do
      {  txmsg.DATA[i] = sci_rx_buffer[i];
         i++;
      }  while ( (sci_rx_buffer[i] != 0x0d) && ( i<9 ));
      txmsg.LENGTH = i;
      txmsg.ID = 0x007f;
      sci_rec_valid = FALSE;
      temp2= Send_Message_Object(&txmsg);          // Send message via CAN
   }        /* end new message from SCI */
   temp2 = Receive_Message_Object( &rxmsg);        //check message from CAN

/* new message from CAN ?              */
   can_transbit = ( temp2 == 0);
   if (can_transbit)
   {  if(rxmsg.DATA[0] == '#')                     /* command ?               */
      {  commando(rxmsg.DATA[1]); /*select command                           */
      }
      sci_tx_bit = 1;
   }

/*  SCIi transmitt activ                                                    */
   if ( sci_tx_bit  && !sci_tx_status)
   {        // start sci transmitt
   for ( i=0;i<rxmsg.LENGTH; i++)
      sci_tx_buffer[i] = rxmsg.DATA[i];
   sci_tx_buffer[i] = 0x00; //string end
   sci_tx_status = 1;
   sci_tx_pointer = &sci_tx_buffer;
   sci_tx_bit = 0;
}
if ( sci_tx_status )
{  temp = (SCS1 & 0x80);
   if (temp ==  0x80 )  /*transmitt empty ? */
   {  temp = *sci_tx_pointer++;
      SCDR = temp;  /* transmitt on byte */
      if ( *sci_tx_pointer == 0x00) sci_tx_status = 0 ; /*string end ?*/
   }
}       /* end sci transmitt activ*/

/* start receive */
temp = (SCS1 & 0x20); /* receive state */
if (temp == 0x20 )/* receive full ?*/
{  temp = SCDR; /* get byte */
   sci_rx_buffer[sci_rx_index] = temp;
   sci_rx_index++;
   if (sci_rx_index > 8)
      sci_rx_index = 0;/* reset buffer*/
   if ( temp == 0x0d ) /* CR = end of message */
   {  if (sci_rx_buffer[0] != 0x0d ) sci_rec_valid = TRUE;
      sci_rx_index = 0;
   }
}     /* end receive    */
}     /*  while forever */

}  // main
```

HARDWARE_HC08.C
Hardware and MCU Specific Routines

```
/*****************************************************************************
 *
 *    DESCRIPTION: Hardware Interface Driver for 68HC08 Microcontroller
 *
 *    AUTHOR: Copyright (c) 1998 by Motorola GmbH, Harald Kreidl,
 *           Harald.Kreidl@Motorola.com
 *
 *    HISTORY:    Version 1.2 / 25-Nov-2000
 *
 *****************************************************************************/

/** include files **/

#include"MCU_Select.h"
#include"MCU_Hardware_Description.h"

/** Public functions: Interrupt Service Routines**/

voidADC_IRQ( void );       // ADC
voidKBD_IRQ( void );       // Keyboard
voidPLL_IRQ( void );       // PLL
voidIRQ1_IRQ( void );      // IRQ1
voidSWI_IRQ( void );       // SWI

voidIRQ_IRQ( void );       // IRQ1
voidIni_mcu( void );       // Initialsiation of the MCU
voidwait( unsigned int );

voidIni_config( void );    // Initialise the CONFIG Registers
voidIni_PLL( void );       // Initialisz the CGM
voidIni_Timer( void );
voidIni_SCI( void );
voidIni_SPI( void );
voidTERMIO_Init( void );
voidIni_ADC( void );
void Ini_PTC(void);

/* Initialise the MCU                    */
voidIni_mcu( void )
{  Ini_config();           // Initialise the CONFIG Registers
   Ini_PLL();              // Initialisz the CGM
                           // TERMIO_Init();
   Ini_Timer();
   //Ini_SCI();
   //Ini_SPI();
   Ini_ADC();
}
```

```
voidIni_config( void )
{  CONFIG2|=OSCSTOPENB+SCIBDSRC; // Oscillator Stop Mode Enable + SCI clocked by bus
   CONFIG1|=LVISTOP+LVI5OR3+STOP+COPD;
}

voidIni_PLL( void )
{  PCTL&=~PLLON;                 // Switch off the PLL
// fRCLK = 32.768 kHZ, fBus = 7.3728 MHz
/* PCTL|=PRE_0+VPR_2;            // P = 0, E = 2
   PMS=  0x384;                  // N = 384
   PMRS= 0xC0;                   // L = C0
   PMDS= 1;                      // R = 1
*/
// fRCLK = 32.768 kHZ, fBus = 4.9152 MHz
   PCTL|=PRE_0+VPR_2;            // P = 0, E = 2
   PMS=  0x258;                  // N = 384
   PMRS= 0x80;                   // L = C0
   PMDS= 1;                      // R = 1

   PCTL|=PLLON;                  // Switch on the PLL
   PBWC|=AUTO;                   // Automatic bandwidth control
//    while( (PBWC & LOCK) ) {}  // Wait to synchronize unlock the PLL
   PCTL|= BCS;                   // Select CGMVCLK/2 as CGMOUT

// TEST CODE FOR PLL SETUP
// Following tests the above PLL settings to see if the internal
// clock is set at the desired rate. Internal clock rate is 7x
// frequency sensed at bit 0 of port B.

// DDRB|=DDRB0;
// while( TRUE )
// {
// PTB|= PTB0;
// no_operation();
// no_operation();
// no_operation();
// TB &= ~PTB0;
// }
}

voidIni_Timer( void )
{
// Ini Timebase Module TBM for 1 ms interrupts
   TBCR&=~TBON;            /* TBM switch off*/
   TBCR|=TBM_1+TBIE;       /* 1 s interrupt, interrupt enable*/
   TBCR|=TBON;             /* TBM switch on*/
// Timer 1 Initialisation
   T1SC|=(TRST+TSTOP+IB1); // Reset, stop, prescaler = bus f / 1
   T1MOD=0xFFFF;
// Timer 1 Channel 1 Output Compare
   T1SC0=(OC_T+CH0IE);     // Channel 0 OC, interrupts enabled*/
   T1CH0=12;
// Timer 1 Channel 2 Input Capture
   T1SC1=(IC_R+CH0IE);
```

```
   T1SC&=~TSTOP;           /* Start timer 1*/
}

voidIni_SCI( void )
{  SCC1|= ENSCI+ILTY;
   SCBR|= SCR_8;           // 9600bd
   SCC2|= RE+TE ;
}

voidIni_ADC( void )        /* AD-Chanel 2 on PTB4 */
{  ADSCR=0;
   ADSCR|=ADCO + ADCH2;
   ADCLK|=ADICLK + ADIV1;

}

voidIni_SPI( void )
{  SPCR =0;                        // reset SPI
   SPCR|=SPMSTR;                   // SPI master
   SPSCR|=SPR0+SPR1;               // BD=128 slow for test
   PTD|= PTD0;                     // SS = CS = 1
   DDRD|= (DDRD0 + DDRD2 + DDRD3);  // Output
   SPCR|=SPE;                      // SPI enable
}

voidIni_Port( void )
{
 /*  MC33388 signal, select GR08 and GP32 */

#if HC08GP32 == TRUE
   PTC|= PTC5+PTC6;                /* PTC5= STB , PTC6=EN     */
   DDRC |= DDRC5+DDRC6;
#endif

#if HC08GR8 == TRUE
   PTC|= PTC0+PTC1;                /* PTC0=STB, PTC1=EN       */
   DDRC|= DDRC0+DDRC1;
#endif
}

/*
```

```
 ** ADC_IRQ *
 *  FILENAME:E:\Software\Cankit\Hardware.c
 *
 *  PARAMETERS:NONE
 *
 *  DESCRIPTION:ADC Interrupt Service Routine
 *
 *  RETURNS:NONE
 *
 */

#ifdef __HIWARE__
#pragma TRAP_PROC
void ADC_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function */
@interrupt voidADC_IRQ(void )
#endif
{}

/*
 ** KBD_IRQ
 *
 *  FILENAME: E:\Software\Cankit\Hardware.c
 *
 *  PARAMETERS:NONE
 *
 *  DESCRIPTION:KBD Interrupt Service Routine
 *
 *  RETURNS:NONE
 *
 */

#ifdef __HIWARE__
#pragma TRAP_PROC
void KBD_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function */
@interrupt voidKBD_IRQ( void )
#endif
{
}

/*
```

```
 ** PLL_IRQ
 *
 *  FILENAME: E:\Software\Cankit\Hardware.c
 *
 *  PARAMETERS:NONE
 *
 *  DESCRIPTION:PLL Interrupt Service Routine
 *
 *  RETURNS:NONE
 *
 */

#ifdef __HIWARE__
#pragma TRAP_PROC
void PLL_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function */
@interrupt voidPLL_IRQ( void )
#endif
{
}
/*
 ** IRQ1_IRQ
 *
 *  FILENAME: E:\Software\Cankit\Hardware.c
 *
 *  PARAMETERS:NONE
 *
 *  DESCRIPTION:IRQ1 Interrupt Service Routine
 *
 *  RETURNS:NONE
 *
 */

#ifdef __HIWARE__
#pragma TRAP_PROC
void IRQ1_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function */
@interrupt voidIRQ1_IRQ( void )
#endif
{  // ISCR|=ACK1;/* Acknowledge the interrupt requeset*/
#ifdef __HIWARE__
asm{
   JMP$0FF00
   }
#else
_asm("jmp $ff00\n");
#endif
}

#ifdef __HIWARE__
#pragma TRAP_PROC
void IRQ_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function*/
@interrupt voidIRQ_IRQ( void )
#endif
```

```
{  // ISCR|=ACK1;/* Acknowledge the interrupt requeset*/
#ifdef __HIWARE__
asm{
   JMP$0FF00
};
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function */
_asm("jmp $ff00\n");
#endif
}

/*
 ** SWI_IRQ
 *
 *  FILENAME: E:\Software\Cankit\Hardware.c
 *
 *  PARAMETERS:NONE
 *
 *  DESCRIPTION:SWI Interrupt Service Routine
 *
 *  RETURNS:NONE
 *
 */

#ifdef __HIWARE__
#pragma TRAP_PROC
void SWI_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function*/
@interrupt voidSWI_IRQ( void )
#endif
{
}

#ifdef __HIWARE__
#pragma TRAP_PROC
void SCI_Tx_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function*/
@interrupt voidSCI_Tx_IRQ( void )
#endif
{
}

#ifdef __HIWARE__
#pragma TRAP_PROC
void SCI_Rx_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function*/
@interrupt voidSCI_Rx_IRQ( void )
#endif
{
// Include Application Code Here
}
```

```
#ifdef __HIWARE__
#pragma TRAP_PROC
void SCI_Error_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function*/
@interrupt voidSCI_Error_IRQ( void )
#endif
{
}

#ifdef __HIWARE__
#pragma TRAP_PROC
void SPI_Tx_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function*/
@interrupt voidSPI_Tx_IRQ( void )
#endif
{
}

#ifdef __HIWARE__
#pragma TRAP_PROC
void SPI_Rx_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function*/
@interrupt voidSPI_Rx_IRQ( void )
#endif
{
}

#ifdef __HIWARE__
#pragma TRAP_PROC
void TIM2_Overflow_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function*/
@interrupt voidTIM2_Overflow_IRQ( void )
#endif
{
}

#ifdef __HIWARE__
#pragma TRAP_PROC
void  TIM2_Channel1_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function*/
@interrupt void TIM2_Channel1_IRQ( void )
#endif
{
}

#ifdef __HIWARE__
#pragma TRAP_PROC
void TIM2_Channel0_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function*/
@interrupt voidTIM2_Channel0_IRQ( void )
#endif
{
}
```

```
#ifdef __HIWARE__
#pragma TRAP_PROC
void TIM1_Overflow_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function*/
@interrupt voidTIM1_Overflow_IRQ( void )
#endif
{
}

#ifdef __HIWARE__
#pragma TRAP_PROC
void TIM1_Channel1_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function*/
@interrupt voidTIM1_Channel1_IRQ( void )
#endif
{
}

#ifdef __HIWARE__
#pragma TRAP_PROC
void  TIM1_Channel0_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function*/
@interrupt void TIM1_Channel0_IRQ( void )
#endif
{
}

#ifdef __HIWARE__
#pragma TRAP_PROC
void  Timebase_IRQ( void )
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function*/
@interrupt void Timebase_IRQ( void )
#endif
{
}
```

CAN_DRIVER.C
CAN Driver Functions

```
//***************************************************************************
// Project Includes
//***************************************************************************

#include "cpcore.h"
#include "MCU_Assembler_Instructions.h"
#include "CAN_Controller.h"
#include <stdlib.h>

/*****************************************************************************
* Function Name:Send_Message_Object()                                        *
* Description:Sends a message object of a maximum of 8 bytes through         *
*           the CAN interface                                                *
* Notes:    This routine copies the message object into the appropriate      *
*           registers in the stand-alone CAN controller and starts the       *
*           transmission. After successful transmission it returns a 1,      *
*           otherwise a 0.                                                   *
* Input:    char * objectPointer to the message object                       *
* Output:   1  Successful transmission                                       *
*           0  Transmission failed                                           *
*****************************************************************************
* Update    History:                                                         *
* Rev       Date        Author  Description of Change                        *
* ---       ---------   ------------------------------------------------*
* 0.1       08-Jan-01   Harald Kreidl Initial release,                       *
*                       Specification of this function                       *
* 1.0       Feb-01      P.Dilger relase                                      *
*****************************************************************************/

#if SJA1000 == TRUE

unsigned char Send_Message_Object( TCANMsg *atxmsg )
{  _U08 tmp,temp1;
   _U08 byte0,byte1,byte2,byte3,byte4,byte5,byte6,byte7;
   if ( ! (Read_Byte( StatusReg ) & 0x04) )// Check Transmit Buffer Access
   {  return CAN_ERR_XMTFULL;// transmit buffer locked
   }
   if ( Read_Byte( StatusReg )& 0x20 )
   {     return CAN_ERR_XMTFULL;
   }
   Write_Byte( TxBuffer1, (_U08)(atxmsg->ID >> 3) );// Fill transmit buffer
   tmp = ( ((_U08) atxmsg->ID & 0x07) << 5) | (atxmsg->LENGTH & 0x0f);// Length
   if ( atxmsg->RTR )
   {  Write_Byte( TxBuffer2, tmp | 0x10 ) ;// sende remote-request-Message
   }
   else
   {  /* write buffer to SJA*/
      Write_Byte( TxBuffer2, tmp ) ;
      Write_Byte(TxBuffer1+2, atxmsg->DATA[0]);
      Write_Byte(TxBuffer1+3, atxmsg->DATA[1]);
      Write_Byte(TxBuffer1+4, atxmsg->DATA[2]);
```

```
      Write_Byte(TxBuffer1+5, atxmsg->DATA[3]) ;
      Write_Byte(TxBuffer1+6, atxmsg->DATA[4]) ;
      Write_Byte(TxBuffer1+7, atxmsg->DATA[5]) ;
      Write_Byte(TxBuffer1+8, atxmsg->DATA[6]) ;
      Write_Byte(TxBuffer1+9, atxmsg->DATA[7]) ;
   }

/* Transmit data by setting of TRANSMISSION REQUEST                 */
   Write_Byte( CommandReg, TR_Bit);  /* Start transmition           */
   return( CAN_ERR_OK );
}


/*****************************************************************************
* Function Name:Receive_Message_Object()                 *
* Description:Receives a message object of a maximum of 8 bytes through the CAN*
*        interface                                              *
* Notes: This routine checks with the CAN stand-alone controller if there has been*
*   received a message. If there was a message this is copied into the*
*   RX_MESSAGE_OBJECT and a 1 is returned. If there was no new message object in*
*   the receive buffer of the controller a 0 is returned.*
* Input: none                                                        *
* Output:1    Successful receiption                          *
*        0    No message received                               *
*****************************************************************************
* UpdateHistory:                                                 *
* Rev Date        Author          Description of Change*
* --- ---------   ------          ---------------------------------------------*
* 0.1 08-Jan-01  Harald Kreidl   Initial release, Specification of this function*
*****************************************************************************/



unsigned char Receive_Message_Object(TCANMsg *arxmsg  )
{  _U08received ;
   _U08result = 0;
   _U08Identifier_high, Identifier_low, Length, Status;

/************************ receive *************************/
   received = 0 ;
   Status = Read_Byte( StatusReg );
   if ( Status & 0x01 )
   {  /* Receive buffer status == full ? */
      received = 1 ;
      Identifier_high = Read_Byte( RxBuffer1 );
      Identifier_low = Read_Byte( RxBuffer2 );
      Length = Identifier_low;
      arxmsg->ID = ((Identifier_high << 3) | (Identifier_low) >> 5);
      arxmsg->LENGTH = Length & 0x0f ;
      if( Length & 0x10 )
      {  arxmsg->RTR = 1 ;            /* RTR-Bit =1 ! */
      }
      else
      {  /* read buffer */
         arxmsg->RTR = 0 ;
```

```
         arxmsg->DATA[0] = Read_Byte( RxBuffer1+2 );
         arxmsg->DATA[1] = Read_Byte( RxBuffer1+3 );
         arxmsg->DATA[2] = Read_Byte( RxBuffer1+4 );
         arxmsg->DATA[3] = Read_Byte( RxBuffer1+5 );
         arxmsg->DATA[4] = Read_Byte( RxBuffer1+6 );
         arxmsg->DATA[5] = Read_Byte( RxBuffer1+7 );
         arxmsg->DATA[6] = Read_Byte( RxBuffer1+8 );
         arxmsg->DATA[7] = Read_Byte( RxBuffer1+9 );
      }
      /* release receive buffer */
      Bit_Set( CommandReg, RRB_Bit) ;
      /*  'Overrun'-reset */
      if ( Read_Byte( StatusReg ) & 0x02 )
      {  result |= CAN_ERR_OVERRUN ;
         Bit_Set( CommandReg, CDO_Bit);
      }
   }
/********************* errorhandling ***********************/
   if ( Status & 0x80 )
   {  /*** Bus-Off: Controller is NOT involved in bus activities */
      result |= CAN_ERR_BUSOFF ;
      /*** Reset request auf 'absent' */
      Bit_Clear( ModeControlReg, (!RM_RR_Bit)) ;
   }
   else
   {  /*** Bus-On: Controller is involved in bus activities **/
      if ( Status & 0x40 )
      {  /*  Errorcounter on limit  */
         result |= CAN_ERR_BUSERROR ;
      }
   }
   if ( ! received ) result |= CAN_ERR_RCVEMPTY ;
      return result ;
}  /* End of CAN_Read() */

#endif   // SJA mode
```

```
#ifMCP2510 == TRUE

unsigned char Send_Message_Object( TCANMsg *atxmsg )
{  _U08 tmp,temp1;
   _U16 tmp2;
   tmp2 = atxmsg->ID >> 3;
   Spi_Write(TXB0SIDH,((_U08) tmp2));/* ident hi */
   tmp2=( atxmsg->ID << 5);
   tmp2 = tmp2 & 0x00e0;
   Spi_Write(TXB0SIDL,(_U08)tmp2);/* ident low */
   Spi_Write(TXB0DCL,atxmsg->LENGTH);/* LENGTH */

   Spi_Write(TXB0D0,atxmsg->DATA[0]);
   Spi_Write(TXB0D0+1,atxmsg->DATA[1]);
   Spi_Write(TXB0D0+2,atxmsg->DATA[2]);
   Spi_Write(TXB0D0+3,atxmsg->DATA[3]);
   Spi_Write(TXB0D0+4,atxmsg->DATA[4]);
   Spi_Write(TXB0D0+5,atxmsg->DATA[5]);
   Spi_Write(TXB0D0+6,atxmsg->DATA[6]);
   Spi_Write(TXB0D0+7,atxmsg->DATA[7]);
   Spi_CAN_Start(); /*start transmitt*/
   return( CAN_ERR_OK );
}  /*end Send_Message_Object*/

unsigned char Receive_Message_Object(TCANMsg *arxmsg  )
{  _U08received ;
   _U08result = 0;
   _U08Identifier_high, Identifier_low, Length, Status;
   result = 1;
   Status = Spi_read_status();
   if( Status &0x01)
   {  /* read buffer */
      arxmsg->LENGTH =Spi_read(RXB0DLC);
      arxmsg->RTR = 0 ;
      arxmsg->DATA[0] = Spi_read( RXB0D0 );
      arxmsg->DATA[1] = Spi_read( RXB0D0+1 );
      arxmsg->DATA[2] = Spi_read( RXB0D0+2 );
      arxmsg->DATA[3] = Spi_read( RXB0D0+3 );
      arxmsg->DATA[4] = Spi_read( RXB0D0+4 );
      arxmsg->DATA[5] = Spi_read( RXB0D0+5 );
      arxmsg->DATA[6] = Spi_read( RXB0D0+6 );
      arxmsg->DATA[7] = Spi_read( RXB0D0+7 );
      Spi_BitMode( CANINTF,0x03,0x00 );/* reset info */
      result = 0;
   }
   return result ;
}  /* End Receive_Message_Object*/

#endif      /* MCP2510 mode*/
```

CAN_CONTROLLER_MCP2510.C
Software Routines for the CAN Controller MCP2510

```
#include"compiler.h"
#include"CAN_Controller.h"
#include"MCU_Hardware_Description.h"

#ifMCP2510 == TRUE

TX_MESSAGE_BUFFERCAN_Tx_Buffer0;
RX_MESSAGE_BUFFERCAN_Rx_Buffer0;
extern TCANMsg txmsg;
extern TCANMsg rxmsg;

voidBus_Input( void )      // adress/data-bus is output
{  DDRA= DDRA & 0x81;      // bit 1-6 for bus, input
   DDRB= DDRB & 0x3F;      // bit 6-7 for bus, input
}

voidBus_Output( void )     // adress/data-bus is output
{  DDRA= DDRA | 0x7E;      // bit 1-6 for bus, output
   DDRB= DDRB | 0xC0;      // bit 6-7 for bus, output
}

voidAddress_To_Bus( _U08 address )
{  Bus_Output();
   PTB = address;          // only bit 6-7
   address = address << 1;
   PTA = address;
   Bus_Output();
}

voidData_To_Bus( _U08 data )
{  Bus_Output();
   PTB = data;
   data= data << 1;
   PTA = data;
}

_U08Data_From_Bus( void )
{  _U08temp1, temp2;
   temp1= PTA;          //  only bit 0-5
   temp1= temp1 >> 1;
   temp2= PTB;
   temp2= temp2 & 0xC0; // only bit6-7
   return( temp1 | temp2 );
}

/* Read the register with the adress in sja_adress from sja in the sja_data */
_U08Read_Byte( _U08 address )
{  _U08PTD_Save;
   _U08byte;
```

```
   PTD_Save = PTD & 0xF0;                 // Bit 0 - 3 for signal
   byte= PTD_Save | read_cycle_0;
   PTD = byte;

   Bus_Output();
   Address_To_Bus( address );
   Bus_Output();

   PTD = PTD_Save | read_cycle_1;
   PTD = PTD_Save | read_cycle_2;
   Bus_Input();
   PTD = PTD_Save | read_cycle_3;
   byte= Data_From_Bus();
   PTD = PTD_Save | read_cycle_2;
   PTD = PTD_Save | read_cycle_1;
   return( byte );
}


voidWrite_Byte( _U08 address, _U08 data )    // write the sja_data into sja_adress
{  _U08PTD_Save;
   PTD_Save = PTD;
   PTD_Save = PTD_Save & 0xF0;               // bit0-3 for signal
   PTD = PTD_Save | write_cycle_0;
   Bus_Output();
   Address_To_Bus( address );
   Bus_Output();
   PTD = PTD_Save | write_cycle_1;
   PTD = PTD_Save | write_cycle_2;
   PTD = PTD_Save | write_cycle_3;
   Data_To_Bus( data );
   PTD = PTD_Save | write_cycle_2;
   PTD = PTD_Save | write_cycle_1;
   PTD = PTD_Save | write_cycle_0;
   Bus_Input();
}

voidBit_Set( _U08 address, _U08 mask )
{  _U08temp;
   temp= Read_Byte (address);
   temp= temp | mask;
   Write_Byte ( address, temp);
}

voidBit_Clear( _U08 address, _U08 mask )
{  _U08temp;
   temp= Read_Byte (address);
   temp= temp & mask;
   Write_Byte ( address, temp);
}
void Initialize_CAN_Controller( void )
{  _U08 temp;
   _U08 temp2;
   PTD = write_cycle_1;                        // Control signal for SJA1000
   DDRD|= (DDRD0 + DDRD1 + DDRD2 + DDRD3);     // Output
```

```
   Write_Byte( ModeControlReg, RM_RR_Bit );     // CAN controller in reset mode
   Write_Byte( CommandReg, (CDO_Bit + RRB_Bit + AT_Bit) );
   Write_Byte( AcceptCodeReg, 0 );              // All messages are received
   Write_Byte( AcceptMaskReg, 0xFF );
   Write_Byte( BusTiming0Reg, can_baud0_20k );  // 20 kBaud
   Write_Byte( BusTiming1Reg, can_baud1_20k );
   Write_Byte( ClockDivideReg, (DivBy1+CBP_Bit) );
   Write_Byte( OutControlReg, (Tx0PshPull + NormalMode) );
   Write_Byte( ModeControlReg, 0 );             // CAN controller out of reset mode
}  /* END Initialize_CAN_Controller */

/*****************************************************************************
*  CAN_Write()
*  Send message via CAN bus.
*  The message is taken from 'PCANMsg->' uebernehmen.
*  Input:
*     PCANMsg: Pointer to message buffer
*  Output:
*     Error Bit field mask
*     0: Message was successfully sent
*     CAN_ERR_XMTFULL:Transmission active, Message was not sent
*/

//_U08CAN_Write( TCANMsg *PCANMsg )
_U08CAN_Write( void )
{  _U08 tmp;
   // test test
   _U08 byte1, byte2, byte3, byte4, byte5, byte6, byte7, byte8,test1,test2;
   if ( ! (Read_Byte( StatusReg ) & 0x04) )     // Check 'Transmit Buffer Access
   {  return CAN_ERR_XMTFULL;// transmit buffer locked
   }
   if ( Read_Byte( StatusReg )& 0x20 )
   {  return CAN_ERR_XMTFULL;
   }

   Write_Byte( TxBuffer1, (_U08)(txmsg.ID >> 3) ); // Transmit Puffer fuellen
   tmp = ( ((_U08) txmsg.ID & 0x07) << 5) | (txmsg.LENGTH & 0x0f);   // Length
   if ( txmsg.RTR )
   {  Write_Byte( TxBuffer2, tmp | 0x10 ) ;// sende remote-request-Message
   }
   else
   {     /* Send data bytes      */
      Write_Byte( TxBuffer2, tmp ) ;
      /* den ganzen Sendepuffer in den Controller bewegen */
      Write_Byte(TxBuffer1+2, txmsg.DATA[0]) ;
      Write_Byte(TxBuffer1+3, txmsg.DATA[1]) ;
      Write_Byte(TxBuffer1+4, txmsg.DATA[2]) ;
      Write_Byte(TxBuffer1+5, txmsg.DATA[3]) ;
      Write_Byte(TxBuffer1+6, txmsg.DATA[4]) ;
      Write_Byte(TxBuffer1+7, txmsg.DATA[5]) ;
      Write_Byte(TxBuffer1+8, txmsg.DATA[6]) ;
      Write_Byte(TxBuffer1+9, txmsg.DATA[7]) ;
   }
```

```
// test test
/*    byte1 = Read_Byte(TxBuffer1+0 );
      byte2 = Read_Byte(TxBuffer1+1 );
      byte3 = Read_Byte(TxBuffer1+2) ;
      byte4 = Read_Byte(TxBuffer1+3) ;
      byte5 = Read_Byte(TxBuffer1+4) ;
      byte6 = Read_Byte(TxBuffer1+5) ;
      byte7 = Read_Byte(TxBuffer1+6) ;
      byte8 = Read_Byte(TxBuffer1+7) ;
      byte8 = Read_Byte(TxBuffer1+8) ;
      byte8 = Read_Byte(TxBuffer1+9) ;
      test1 = Read_Status();
*/
/* Transmit data by setting of TRANSMISSION REQUEST */
   Write_Byte( CommandReg, TR_Bit);
   // test2 = Read_Status();
   return( CAN_ERR_OK );
}

/*****************************************************************************
*  CAN_Read( void )
*  Polling Routine:
*  -  The received message is transfered int the buffer 'PCANMsg->'
*  -  Check the bus status and if necessary reset the CAN controller
*  Input:
*     PCANMsg: Pointer to message buffer of the received message
*  Output:
*     Error bit field mask
*     0 : Receiption completed, new message in 'PCANMsg->'
*     CAN_ERR_RCVEMPTY: no message was received
*     CAN_ERR_OVERRUN : next message came before the previous message could be read
*     CAN_ERR_BUSERROR: Bus errors happened
*     CAN_ERR_BUSOFF:      CAN controller was in bus off state
*                          and was activated afterwards
*/

_U08  CAN_Read( void )
{  _U08received ;
   _U08result = 0;
   _U08Identifier_high, Identifier_low, Length, Status;

/************************ Receive data ************************/
   received = 0 ;
   Status = Read_Byte( StatusReg );
   if ( Status & 0x01 )
   {  /* Receive buffer status == full ? */
      /*** Receive message                      */
      received = 1 ;
      Identifier_high = Read_Byte( RxBuffer1 );
      Identifier_low = Read_Byte( RxBuffer2 );
      Length = Identifier_low;
      rxmsg.ID = ((Identifier_high << 3) | (Identifier_low) >> 5);
      rxmsg.LENGTH = Length & 0x0f ;
      if( Length & 0x10 )
      {  rxmsg.RTR = 1 ;           /* Set RTR bit       */
```

```
      }
      else
      {  /* Copy whole receive buffer to CAN_RWBUFF      */
         rxmsg.RTR = 0;
         rxmsg.DATA[0] = Read_Byte( RxBuffer1+2 );
         rxmsg.DATA[1] = Read_Byte( RxBuffer1+3 );
         rxmsg.DATA[2] = Read_Byte( RxBuffer1+4 );
         rxmsg.DATA[3] = Read_Byte( RxBuffer1+5 );
         rxmsg.DATA[4] = Read_Byte( RxBuffer1+6 );
         rxmsg.DATA[5] = Read_Byte( RxBuffer1+7 );
         rxmsg.DATA[6] = Read_Byte( RxBuffer1+8 );
         rxmsg.DATA[7] = Read_Byte( RxBuffer1+9 );
      }
      /* release receive buffer           */
      Bit_Set( CommandReg, RRB_Bit) ;
      /* Clear 'Overrun' condition        */
      if ( Read_Byte( StatusReg ) & 0x02 )
      {  result |= CAN_ERR_OVERRUN ;
         Bit_Set( CommandReg, CDO_Bit);
      }
   }
/********************* Error processing ***********************/
   Status = Read_Byte( StatusReg );
   if ( Status & 0x80 )
   {  /*** Bus-Off: Controller is NOT involved in bus activities */
      result |= CAN_ERR_BUSOFF ;
      /*** Reset request auf 'absent' */
      Bit_Clear( ModeControlReg, (!RM_RR_Bit)) ;
      /* CAN controller in bus on state after the receiption of 128 * 11
         consecutive recessive bits                                    */
   }
   else
   {  /*** Bus-On: Controller is involved in bus activities **/
      if ( Status & 0x40 )
      {  /*** Error counter reached limit                          */
         result |= CAN_ERR_BUSERROR ;
      }
   }
   if ( ! received ) result |= CAN_ERR_RCVEMPTY ;
   return result ;
}  /* End of CAN_Read() */

_U08Read_Status( void )
{  return( Read_Byte( StatusReg ) );
}

#endif // MCP2510
```

CAN_CONTROLLER_SJA1000.C
Software Routines for the CAN Controller SJA1000

```
#include    "compiler.h"
#include    "CAN_Controller.h"
#include    "MCU_Hardware_Description.h"

#ifSJA1000 == TRUE

extern TCANMsg txmsg;
extern TCANMsg rxmsg;

voidBus_Input( void )            // adress/data-bus is output
{  DDRA= DDRA & 0x81;            // bit 1-6 for bus, input
   DDRB= DDRB & 0x3F;            // bit 6-7 for bus, input
}

voidBus_Output( void )           // adress/data-bus is output
{  DDRA= DDRA | 0x7E;            // bit 1-6 for bus, output
   DDRB= DDRB | 0xC0;            // bit 6-7 for bus, output
}

voidAddress_To_Bus( _U08 address )
{  Bus_Output();
   PTB= address;                 // only bit 6-7
   address = address << 1;
   PTA= address;
   Bus_Output();
}

voidData_To_Bus( _U08 data )
{  Bus_Output();
   PTB = data;
   data= data << 1;
   PTA= data;
}

_U08Data_From_Bus( void )
{  _U08temp1, temp2;
   temp1= PTA;                   //  only bit 0-5
   temp1= temp1 >> 1;
   temp2= PTB;
   temp2= temp2 & 0xC0;          // only bit6-7
   return( temp1 | temp2 );
}

_U08Read_Byte( _U08 address )    // read the register with the adress
                                 // in sja_adress from sja in the sja_data
{  _U08PTD_Save;
   _U08byte;

   PTD_Save= PTD & 0xF0;         // Bit 0 - 3 for signal
   byte= PTD_Save | read_cycle_0;
   PTD= byte;
```

```
   Bus_Output();
   Address_To_Bus( address );
   Bus_Output();

   PTD= PTD_Save | read_cycle_1;
   PTD= PTD_Save | read_cycle_2;
   Bus_Input();
   PTD= PTD_Save | read_cycle_3;
   byte= Data_From_Bus();
   PTD= PTD_Save | read_cycle_2;
   PTD= PTD_Save | read_cycle_1;
   return( byte );
}

voidWrite_Byte( _U08 address, _U08 data )    // write the sja_data into sja_adress
{  _U08PTD_Save;
   PTD_Save = PTD;
   PTD_Save = PTD_Save & 0xF0;               // bit0-3 for signal
   PTD= PTD_Save | write_cycle_0;
   Bus_Output();
   Address_To_Bus( address );
   Bus_Output();
   PTD= PTD_Save | write_cycle_1;
   PTD= PTD_Save | write_cycle_2;
   PTD= PTD_Save | write_cycle_3;
   Data_To_Bus( data );
   PTD= PTD_Save | write_cycle_2;
   PTD= PTD_Save | write_cycle_1;
   PTD= PTD_Save | write_cycle_0;
   Bus_Input();
}

voidBit_Set( _U08 address, _U08 mask )
{  _U08temp;
   temp= Read_Byte (address);
   temp= temp | mask;
   Write_Byte ( address, temp);
}

voidBit_Clear( _U08 address, _U08 mask )
{  _U08temp;
   temp= Read_Byte (address);
   temp= temp & mask;
   Write_Byte ( address, temp);
}

voidInitialize_CAN_Controller( void )
{  _U08 temp;
   _U08 temp2;
   PTD= write_cycle_1;                       // Control signal for SJA1000
   DDRD|= (DDRD0 + DDRD1 + DDRD2 + DDRD3);   // Output
   Write_Byte( ModeControlReg, RM_RR_Bit );  // CAN controller in reset mode
   Write_Byte( CommandReg, (CDO_Bit + RRB_Bit + AT_Bit) );
   Write_Byte( AcceptCodeReg, 0 );           // All messages are received
```

```
   Write_Byte( AcceptMaskReg, 0xFF );
   Write_Byte( BusTiming0Reg, can_baud0_20k );// 20 kBaud
   Write_Byte( BusTiming1Reg, can_baud1_20k );
   Write_Byte( ClockDivideReg, (DivBy1+CBP_Bit) );
   Write_Byte( OutControlReg, (Tx0PshPull + NormalMode) );
   Write_Byte( ModeControlReg, 0 );          // CAN controller out of reset mode
}  /* END Initialize_CAN_Controller */



_U08CAN_Write( void )
{  _U08 tmp;
   // test test
   _U08 byte1, byte2, byte3, byte4, byte5, byte6, byte7, byte8,test1,test2;
   if ( ! (Read_Byte( StatusReg ) & 0x04) )        // Check 'Transmit Buffer Access
   {  return CAN_ERR_XMTFULL;                      // transmit buffer locked
   }
   if ( Read_Byte( StatusReg )& 0x20 )
   {  return CAN_ERR_XMTFULL;
   }

   Write_Byte( TxBuffer1, (_U08)(txmsg.ID >> 3) ); // Transmit Puffer fuellen
   tmp = ( ((_U08) txmsg.ID & 0x07) << 5) | (txmsg.LENGTH & 0x0f);   // Length
  if ( txmsg.RTR )
  {   Write_Byte( TxBuffer2, tmp | 0x10 ) ;// sende remote-request-Message
  }
  else
  {   /* sende data bytes                                              */
      Write_Byte( TxBuffer2, tmp ) ;
      /* Copy send buffer into the CAN controller                     */
      Write_Byte(TxBuffer1+2, txmsg.DATA[0]);
      Write_Byte(TxBuffer1+3, txmsg.DATA[1]);
      Write_Byte(TxBuffer1+4, txmsg.DATA[2]);
      Write_Byte(TxBuffer1+5, txmsg.DATA[3]);
      Write_Byte(TxBuffer1+6, txmsg.DATA[4]);
      Write_Byte(TxBuffer1+7, txmsg.DATA[5]);
      Write_Byte(TxBuffer1+8, txmsg.DATA[6]);
      Write_Byte(TxBuffer1+9,txmsg.DATA[7]);
   }
   /* transmit data by setting of TRANSMISSION REQUEST */
   Write_Byte( CommandReg, TR_Bit);
   return( CAN_ERR_OK );
}

_U08CAN_Read( void )
{  _U08received ;
   _U08result = 0;
   _U08Identifier_high, Identifier_low, Length, Status;

   received = 0 ;
   Status = Read_Byte( StatusReg );
   if ( Status & 0x01 )
   {  /* Receive buffer status == full ? */
      received = 1 ;
      Identifier_high = Read_Byte( RxBuffer1 );
```

```
      Identifier_low = Read_Byte( RxBuffer2 );
      Length= Identifier_low;
      rxmsg.ID = ((Identifier_high << 3) | (Identifier_low) >> 5);
      rxmsg.LENGTH = Length & 0x0f ;
      if( Length & 0x10 )
      {  rxmsg.RTR = 1 ; /* RTR-Bit gesetzt! */
      }
      else
      {  rxmsg.RTR = 0 ;
         rxmsg.DATA[0] = Read_Byte( RxBuffer1+2 );
         rxmsg.DATA[1] = Read_Byte( RxBuffer1+3 );
         rxmsg.DATA[2] = Read_Byte( RxBuffer1+4 );
         rxmsg.DATA[3] = Read_Byte( RxBuffer1+5 );
         rxmsg.DATA[4] = Read_Byte( RxBuffer1+6 );
         rxmsg.DATA[5] = Read_Byte( RxBuffer1+7 );
         rxmsg.DATA[6] = Read_Byte( RxBuffer1+8 );
         rxmsg.DATA[7] = Read_Byte( RxBuffer1+9 );
      }
      /* release receive buffer        */
      Bit_Set( CommandReg, RRB_Bit) ;
      /*  'Overrun'-reset              */
      if ( Read_Byte( StatusReg ) & 0x02 )
      {  result |= CAN_ERR_OVERRUN ;
         Bit_Set( CommandReg, CDO_Bit);
      }
   }

/********************** error handling ***********************/
   Status = Read_Byte( StatusReg );
   if ( Status & 0x80 )
   {  /*** Bus-Off: Controller is NOT involved in bus activities        */
      result |= CAN_ERR_BUSOFF ;
      /*** Reset request 'absent'                                       */
      Bit_Clear( ModeControlReg, (!RM_RR_Bit)) ;
   }
   else
   {  /*** Bus-On: Controller is involved in bus activities **/
      if ( Status & 0x40 )
      {  /*** Ein Errorcounter on Limit  */
         result |= CAN_ERR_BUSERROR ;
      }
   }
   if ( ! received ) result |= CAN_ERR_RCVEMPTY ;
   return result ;
}  /* End of CAN_Read() */

_U08Read_Status( void )
{  return( Read_Byte( StatusReg ) );
}

#endif/* SJA1000 */
```

IRQTAB.C
Interrupt Vectors

```
/*****************************************************************
 *
 * DESCRIPTION:   Interrupt Vector Tables for the HC08 and HC12 Microcontrollers
 *
 * AUTHOR:        Copyright (c) 1998 by Motorola GmbH, Harald Kreidl,
 *                Harald.Kreidl@Motorola.com
 *
 * HISTORY:       Version 1.1 / December 1998
 *                Version 1.2 / December 1999
 *
 *****************************************************************/

/** include files **/
#include            "MCU_Select.h"
#include            "MCU_Hardware_Description.h"

/*
 **
 *
 *  FILENAME:       IRQtab.c
 *
 *  PARAMETERS:     NONE
 *
 *  DESCRIPTION:    Interrupt Vector Table
 *
 *  RETURNS:        NONE
 *
 */

// IRQ Functions are external
extern void         Timebase_IRQ();         /* Timebase                      */
extern void         ADC_IRQ();              /* ADC                           */
extern void         KBD_IRQ();              /* Keyboard                      */
extern void         SCI_Tx_IRQ();           /* SCI transmit                  */
extern void         SCI_Rx_IRQ();           /* SCI receive                   */
extern void         SCI_Error_IRQ();        /* SCI error                     */
extern void         SPI_Tx_IRQ();           /* SPI transmit                  */
extern void         SPI_Rx_IRQ();           /* SPI receive                   */
extern void         TIM2_Overflow_IRQ();    /* TIM2 overflow                 */
extern void         TIM2_Channel1_IRQ();    /* TIM2 channel 1                */
extern void         TIM2_Channel0_IRQ();    /* TIM2 channel 0                */
extern void         TIM1_Overflow_IRQ();    /* TIM1 overflow                 */
extern void         TIM1_Channel1_IRQ();    /* TIM1 channel                 1*/
extern void         TIM1_Channel0_IRQ();    /* TIM1 channel                 0*/
extern void         PLL_IRQ();              /* PLL                           */
extern void         IRQ_IRQ();              /* IRQ1                          */
extern void         SWI_IRQ();              /* SWI                           */
extern void         _stext();               /* RESET                         */
```

```
void (* const _vectab[])() =
{       Timebase_IRQ,                   /* Timebase                          */
        ADC_IRQ,                        /* Analog-to-Digital Converter       */
        KBD_IRQ,                        /* Keyboard                          */
        SCI_Tx_IRQ,                     /* SCI transmit                      */
        SCI_Rx_IRQ,                     /* SCI receive                       */
        SCI_Error_IRQ,                  /* SCI error                         */
        SPI_Tx_IRQ,                     /* SPI transmit                      */
        SPI_Rx_IRQ,                     /* SPI receive                       */
        TIM2_Overflow_IRQ,              /* TIM2 overflow                     */
        TIM2_Channel1_IRQ,              /* TIM2 channel 1                    */
        TIM2_Channel0_IRQ,              /* TIM2 channel 0                    */
        TIM1_Overflow_IRQ,              /* TIM1 overflow                     */
        TIM1_Channel1_IRQ,              /* TIM1 channel                     1*/
        TIM1_Channel0_IRQ,              /* TIM1 channel                     0*/
        PLL_IRQ,                        /* PLL                               */
        IRQ_IRQ,                        /* IRQ1                              */
        SWI_IRQ,                        /* SWI                               */
        _stext,                         /* RESET                             */
};
```

Include Files

MCU_SELECT.H
Select the Hardware Configuration

```
#define TRUE1
#define FALSE0

/* These options should be changed from the user*/
/* Only one of the options can be TRUE!*/

// Option 1: MC68HC908GR8 + MCP2510
#define GR8MCPTRUE//FALSE

// Option 2: MC68HC908GP32 + MCP2510
#define GP32MCPFALSE

// Option 3: MC68HC908GP32 + SJA1000
#define GP32SJAFALSE//TRUE


/* END OF USER OPTIONS              */

/* Do not change following defines!!!!*/

#if GP32SJA == TRUE
#define HC08GP32TRUE// Select the 908GP32 as MCU
#define HC08GR8FALSE// No 908GR8
#define SJA1000TRUE// Select the SJA1000
#define MCP2510FALSE// No MCP2510
#endif

#if GP32MCP == TRUE
#define HC08GP32TRUE// Select the 908GP32 as MCU
#define HC08GR8FALSE// No 908GR8
#define MCP2510TRUE// Select the MCP2510
#define SJA1000FALSE// No SJA1000
#endif

#if GR8MCP == TRUE
#define HC08GR8TRUE// Select the 908GR8 as MCU
#define HC08GP32FALSE// No 908GP32
#define MCP2510TRUE// Select the MCP2510
#define SJA1000FALSE// No SJA1000
#endif
```

HARDWARE_HC08.C
Register and Bit Definitions for the 68HC08

```
/*****************************************************************************
 *
 * DESCRIPTION:       I/O Definitions for Motorola 68HC08 MCU's:
 *
 * FILE NAME:         MCU_Hardware_Description_HC08.h
 *
 * AUTHOR:            Harald Kreidl, Motorola GmbH, Munich,
 *                    Email: harald.kreidl@motorola.com
 *
 * HISTORY:
 *   Version 1.0, 23. September 1998 (created the file)
 *   Version 1.2, 18. August 1999 (added the HC08GP family)
 *   Version 2.0, 17. November 1999 (restructured file system)
 *
 *****************************************************************************/

//#pragma space [] @tiny

/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      I/O Port Declarations for the HC08 Family*/
/*      Ports A to H                                          */
/*      It is not checked if the ports are available at a specific HC08 MCU!*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */

#if (HC08AZ || HC08GP) == TRUE

#define PTA (*(volatile char*)(0x00))/* port A */
// @tiny volatile char PTA @0x00;
#define PTA0BIT0_SET /* Port A data register, Bit 0*/
#define PTA1BIT1_SET /* Port A data register, Bit 1*/
#define PTA2BIT2_SET /* Port A data register, Bit 2*/
#define PTA3BIT3_SET /* Port A data register, Bit 3*/
#define PTA4BIT4_SET /* Port A data register, Bit 4*/
#define PTA5BIT5_SET /* Port A data register, Bit 5*/
#define PTA6BIT6_SET /* Port A data register, Bit 6*/
#define PTA7BIT7_SET /* Port A data register, Bit 7*/

#define PTB (*(volatile char*)(0x01))/* port B */
// @tiny volatile char PTB @0x01;
#define PTB0BIT0_SET /* Port B data register, Bit 0*/
#define PTB1BIT1_SET /* Port B data register, Bit 1*/
#define PTB2BIT2_SET /* Port B data register, Bit 2*/
#define PTB3BIT3_SET /* Port B data register, Bit 3*/
#define PTB4BIT4_SET /* Port B data register, Bit 4*/
#define PTB5BIT5_SET /* Port B data register, Bit 5*/
#define PTB6BIT6_SET /* Port B data register, Bit 6*/
#define PTB7BIT7_SET /* Port B data register, Bit 7*/
```

```
#define PTC (*(volatile char*)(0x02))/* port C*/
// @tiny volatile  char PTC @0x02;
#define PTC0BIT0_SET/* Port C data register, Bit 0*/
#define PTC1BIT1_SET /* Port C data register, Bit 1*/
#define PTC2BIT2_SET /* Port C data register, Bit 2*/
#define PTC3BIT3_SET /* Port C data register, Bit 3*/
#define PTC4BIT4_SET /* Port C data register, Bit 4*/
#define PTC5BIT5_SET /* Port C data register, Bit 5*/
#define PTC6BIT6_SET /* Port C data register, Bit 6*/
#define PTC7BIT7_SET /* Port C data register, Bit 7*/

#define PTD (*(volatile char*)(0x03))/* port D */
// @tiny volatile  char PTD @0x03;
#define PTD0BIT0_SET /* Port D data register, Bit 0*/
#define PTD1BIT1_SET /* Port D data register, Bit 1*/
#define PTD2BIT2_SET /* Port D data register, Bit 2*/
#define PTD3BIT3_SET /* Port D data register, Bit 3*/
#define PTD4BIT4_SET /* Port D data register, Bit 4*/
#define PTD5BIT5_SET /* Port D data register, Bit 5*/
#define PTD6BIT6_SET /* Port D data register, Bit 6*/
#define PTD7BIT7_SET /* Port D data register, Bit 7*/

#define DDRA (*(volatile char*)(0x04))/* data direction port A */
// @tiny volatile char DDRA @0x04;
#define DDRA0BIT0_SET /* Port A data direction register, Bit 0*/
#define DDRA1BIT1_SET /* Port A data direction register, Bit 1*/
#define DDRA2BIT2_SET /* Port A data direction register, Bit 2*/
#define DDRA3BIT3_SET /* Port A data direction register, Bit 3*/
#define DDRA4BIT4_SET /* Port A data direction register, Bit 4*/
#define DDRA5BIT5_SET /* Port A data direction register, Bit 5*/
#define DDRA6BIT6_SET /* Port A data direction register, Bit 6*/
#define DDRA7BIT7_SET /* Port A data direction register, Bit 7*/

#define DDRB (*(volatile char*)(0x05))/* Data direction port B */
// @tiny volatile char DDRB @0x05;
#define DDRB0BIT0_SET /* Port B data direction register, Bit 0*/
#define DDRB1BIT1_SET /* Port B data direction register, Bit 1*/
#define DDRB2BIT2_SET/* Port B data direction register, Bit 2*/
#define DDRB3BIT3_SET /* Port B data direction register, Bit 3*/
#define DDRB4BIT4_SET /* Port B data direction register, Bit 4*/
#define DDRB5BIT5_SET /* Port B data direction register, Bit 5*/
#define DDRB6BIT6_SET /* Port B data direction register, Bit 6*/
#define DDRB7BIT7_SET /* Port B data direction register, Bit 7*/

#define DDRC (*(volatile char*)(0x06))/* data direction port C */
// @tiny volatile char DDRC @0x06;
#define DDRC0BIT0_SET /* Port C data direction register, Bit 0*/
#define DDRC1BIT1_SET /* Port C data direction register, Bit 1*/
#define DDRC2BIT2_SET /* Port C data direction register, Bit 2*/
#define DDRC3BIT3_SET /* Port C data direction register, Bit 3*/
#define DDRC4BIT4_SET /* Port C data direction register, Bit 4*/
#define DDRC5BIT5_SET /* Port C data direction register, Bit 5*/
#define DDRC6BIT6_SET /* Port C data direction register, Bit 6*/
#define DDRC7BIT7_SET /* Port C data direction register, Bit 7*/
```

```
#define DDRD (*(volatile char*)(0x04))
// @tiny volatile char DDRD @0x07;/* data direction port D */
#define DDRD0 BIT0_SET/* Port D data direction register, Bit 0*/
#define DDRD1 BIT1_SET/* Port D data direction register, Bit 1*/
#define DDRD2 BIT2_SET/* Port D data direction register, Bit 2*/
#define DDRD3BIT3_SET/* Port D data direction register, Bit 3*/
#define DDRD4BIT4_SET /* Port D data direction register, Bit 4*/
#define DDRD5BIT5_SET/* Port D data direction register, Bit 5*/
#define DDRD6BIT6_SET/* Port D data direction register, Bit 6*/
#define DDRD7BIT7_SET/* Port D data direction register, Bit 7*/

#define PTE (*(volatile char*)(0x08))/* port E */
// @tiny volatile char PTE @0x08;
#define PTE0BIT0_SET/* Port E data register, Bit 0*/
#define PTE1BIT1_SET/* Port E data register, Bit 1*/
#define PTE2BIT2_SET/* Port E data register, Bit 2*/
#define PTE3BIT3_SET /* Port E data register, Bit 3*/
#define PTE4BIT4_SET/* Port E data register, Bit 4*/
#define PTE5BIT5_SET/* Port E data register, Bit 5*/
#define PTE6BIT6_SET/* Port E data register, Bit 6*/
#define PTE7BIT7_SET/* Port E data register, Bit 7*/

#define PTF (*(volatile char*)(0x09))/* port F */
// @tiny volatile char PTF @0x09;
#define PTF0BIT0_SET/* Port F data register, Bit 0*/
#define PTF1BIT1_SET /* Port F data register, Bit 1*/
#define PTF2BIT2_SET/* Port F data register, Bit 2*/
#define PTF3BIT3_SET/* Port F data register, Bit 3*/
#define PTF4BIT4_SET/* Port F data register, Bit 4*/
#define PTF5BIT5_SET/* Port F data register, Bit 5*/
#define PTF6BIT6_SET/* Port F data register, Bit 6*/
#define PTF7BIT7_SET/* Port F data register, Bit 7*/

#define PTG (*(volatile char*)(0x0A))/* port G */
// @tiny volatile char PTG @0x0A;
#define PTG0BIT0_SET/* Port G data register, Bit 0*/
#define PTG1BIT1_SET/* Port G data register, Bit 1*/
#define PTG2BIT2_SET/* Port G data register, Bit 2*/
#define PTG3BIT3_SET/* Port G data register, Bit 3*/
#define PTG4BIT4_SET/* Port G data register, Bit 4*/
#define PTG5BIT5_SET/* Port G data register, Bit 5*/
#define PTG6BIT6_SET/* Port G data register, Bit 6*/
#define PTG7BIT7_SET/* Port G data register, Bit 7*/

#define PTH (*(volatile char*)(0x0B))/* port H */
// @tiny volatile  char PTH @0x0B;
#define PTH0BIT0_SET/* Port H data register, Bit 0*/
#define PTH1BIT1_SET/* Port H data register, Bit 1*/
#define PTH2BIT2_SET/* Port H data register, Bit 2*/
#define PTH3BIT3_SET/* Port H data register, Bit 3*/
#define PTH4BIT4_SET/* Port H data register, Bit 4*/
#define PTH5BIT5_SET/* Port H data register, Bit 5*/
#define PTH6BIT6_SET/* Port H data register, Bit 6*/
#define PTH7BIT7_SET/* Port H data register, Bit 7*/
```

```
#define DDRE (*(volatile char*)(0x0C))/* data direction port E */
// @tiny volatile char DDRE @0x0C;
#define DDRE0BIT0_SET/* Port E data direction register, Bit 0*/
#define DDRE1BIT1_SET/* Port E data direction register, Bit 1*/
#define DDRE2BIT2_SET/* Port E data direction register, Bit 2*/
#define DDRE3BIT3_SET/* Port E data direction register, Bit 3*/
#define DDRE4BIT4_SET/* Port E data direction register, Bit 4*/
#define DDRE5BIT5_SET/* Port E data direction register, Bit 5*/
#define DDRE6BIT6_SET/* Port E data direction register, Bit 6*/
#define DDRE7BIT7_SET/* Port E data direction register, Bit 7*/

#endif

#if HC08AZ == TRUE

#define DDRF (*(volatile char*)(0x0D))
// @tiny volatile char DDRF @0x0D;/* data direction port F */
#define DDRF0BIT0_SET/* Port F data direction register, Bit 0*/
#define DDRF1BIT1_SET/* Port F data direction register, Bit 1*/
#define DDRF2BIT2_SET/* Port F data direction register, Bit 2*/
#define DDRF3BIT3_SET/* Port F data direction register, Bit 3*/
#define DDRF4BIT4_SET/* Port F data direction register, Bit 4*/
#define DDRF5BIT5_SET/* Port F data direction register, Bit 5*/
#define DDRF6BIT6_SET/* Port F data direction register, Bit 6*/
#define DDRF7BIT7_SET/* Port F data direction register, Bit 7*/

#define DDRG (*(volatile char*)(0x0E))
// @tiny volatile char DDRG @0x0E;/* data direction port G */
#define DDRG0BIT0_SET/* Port G data direction register, Bit 0*/
#define DDRG1BIT1_SET /* Port G data direction register, Bit 1*/
#define DDRG2BIT2_SET/* Port G data direction register, Bit 2*/
#define DDRG3BIT3_SET/* Port G data direction register, Bit 3*/
#define DDRG4BIT4_SET/* Port G data direction register, Bit 4*/
#define DDRG5BIT5_SET/* Port G data direction register, Bit 5*/
#define DDRG6BIT6_SET/* Port G data direction register, Bit 6*/
#define DDRG7BIT7_SET/* Port G data direction register, Bit 7*/

#define DDRH (*(volatile char*)(0x0F))
// @tiny volatile char DDRH @0x0F;/* data direction port H */
#define DDRH0BIT0_SET/* Port H data direction register, Bit 0*/
#define DDRH1BIT1_SET/* Port H data direction register, Bit 1*/
#define DDRH2BIT2_SET/* Port H data direction register, Bit 2*/
#define DDRH3BIT3_SET/* Port H data direction register, Bit 3*/
#define DDRH4BIT4_SET/* Port H data direction register, Bit 4*/
#define DDRH5BIT5_SET /* Port H data direction register, Bit 5*/
#define DDRH6BIT6_SET/* Port H data direction register, Bit 6*/
#define DDRH7BIT7_SET/* Port H data direction register, Bit 7*/

#endif
```

```
#if HC08GP == TRUE

#define PTAPUE                    (*(volatile char*)(0x0D))
// @tiny volatile char PTAPUE @0x0D; /* Port A input pullup enable                */
#define PTAPUE0     BIT0_SET      /* Port A input pullup enable register, Bit 0   */
#define PTAPUE1     BIT1_SET      /* Port A input pullup enable register, Bit 1   */
#define PTAPUE2     BIT2_SET      /* Port A input pullup enable register, Bit 2   */
#define PTAPUE3     BIT3_SET      /* Port A input pullup enable register, Bit 3   */
#define PTAPUE4     BIT4_SET      /* Port A input pullup enable register, Bit 4   */
#define PTAPUE5     BIT5_SET      /* Port A input pullup enable register, Bit 5   */
#define PTAPUE6     BIT6_SET      /* Port A input pullup enable register, Bit 6   */
#define PTAPUE7     BIT7_SET      /* Port A input pullup enable register, Bit 7   */

#define PTCPUE                    (*(volatile char*)(0x0E))
// @tiny volatile char PTCPUE @0x0E; /* Port A input pullup enable                */
#define PTCPUE0     BIT0_SET      /* Port C input pullup enable register, Bit 0   */
#define PTCPUE1     BIT1_SET      /* Port C input pullup enable register, Bit 1   */
#define PTCPUE2     BIT2_SET      /* Port C input pullup enable register, Bit 2   */
#define PTCPUE3     BIT3_SET      /* Port C input pullup enable register, Bit 3   */
#define PTCPUE4     BIT4_SET      /* Port C input pullup enable register, Bit 4   */
#define PTCPUE5     BIT5_SET      /* Port C input pullup enable register, Bit 5   */
#define PTCPUE6     BIT6_SET      /* Port C input pullup enable register, Bit 6   */
#define PTCPUE7     BIT7_SET      /* Port C input pullup enable register, Bit 7   */

#endif

/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      END of I/O Port Declarations for the HC08 Family                          */
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */

/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      SPI Declarations for the HC08 Family                                      */
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */

#if (HC08AZ || HC08GP) == TRUE

#define SPCR                      (*(volatile char*)(0x10))
// @tiny volatile char SPCR @0x10;/* SPI control register                        */
#define SPTIE       BIT0_SET      /* SPI:                                        */
#define SPE         BIT1_SET      /* SPI:                                        */
#define SPWOM       BIT2_SET      /* SPI:                                        */
#define CPHA        BIT3_SET      /* SPI:                                        */
#define CPOL        BIT4_SET      /* SPI:                                        */
#define SPMSTR      BIT5_SET      /* SPI:                                        */
#define DMAS        BIT6_SET      /* SPI:                                        */
#define SPRIE       BIT7_SET      /* SPI:                                        */

#define SPSCR                     (*(volatile char*)(0x11))
// @tiny volatile char SPSCR @0x11; /* SPI control/status register               */
#define SPR0        BIT0_SET      /* SPI:                                        */
#define SPR1        BIT1_SET      /* SPI:                                        */
#define MODFEN      BIT2_SET      /* SPI:                                        */
#define SPTE        BIT3_SET      /* SPI:                                        */
#define MODF        BIT4_SET      /* SPI:                                        */
#define OVFR        BIT5_SET      /* SPI:                                        */
#define ERRIE       BIT6_SET      /* SPI:                                        */
#define SPRF        BIT7_SET      /* SPI:
              */
#define SPDR                      (*(volatile char*)(0x12))
// @tiny volatile char SPDR @0x12;/* SPI data register                           */
#endif
```

```
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      END of SPI Declarations for the HC08 Family*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */

/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      SCI Declarations for the HC08 Family*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */

#if (HC08AZ || HC08GP) == TRUE

#define SCC1                        (*(volatile char*)(0x13))
// @tiny volatile char SCC1 @0x13;/* SCI control register 1                       */
#define PTY      BIT0_SET          // 0 = even parity, 1 = odd parity
#define PEN      BIT1_SET          // 0 = parity disabled, 1 = parity enabled
#define ILTY     BIT2_SET          // Idle character bit count begins after start (0)
or stop (1) bit
#define WAKE     BIT0_SET+BIT3_SET// 0 = idle line wakeup, 1 = address mark wakeup
#define M        BIT4_SET          // 0 = 8-bit character, 1 = 9-bit character
#define TXINV    BIT0_SET+BIT5_SET// Transmitter output inverted (1) or not inverted
(0)
#define ENSCI    BIT6_SET          // 0 = SCI disabled, 1 = SCI enabled
#define LOOPS    BIT7_SET          // 0 = Normal operation, 1 = loop mode enabled

#define SCC2                        (*(volatile char*)(0x14))
// @tiny volatile char SCC2 @0x14;/* SCI control register 2                       */
#define SBK      BIT0_SET          // 0 = No break characters being transmitted,
                                      1 = transmit break charakters
#define RWU      BIT1_SET          // 0 = Normal operation, 1 = standby state
#define RE       BIT2_SET          // 0 = Receiver disabled, 1 = receiver enabled
#define TE       BIT3_SET          // 0 = Transmitter disabled, 1 = transmitter
                                      enabled
#define ILIE     BIT4_SET          // IDLE not (0) enabled to generate CPU
                                      interrupt requests
#define SCRIE    BIT5_SET          // SCRF not (0) enabled to generate CPU
                                      interrupt
#define TCIE     BIT6_SET          // TC not (0) enabled to generate CPU
                                      interrupt requests
#define SCTIE    BIT7_SET          // SCTE not (0) enabled to generate CPU
                                      interrupt
#define SCC3                        (*(volatile char*)(0x15))
// @tiny volatile char SCC3 @0x15;/* SCI control register 3                       */
#define PEIE     BIT0_SET          /* SCI:                                        */
#define FEIE     BIT1_SET          /* SCI:                                        */
#define NEIE     BIT2_SET          /* SCI:                                        */
#define ORIE     BIT3_SET          /* SCI:                                        */
#define DMATE    BIT4_SET          /* SCI:                                        */
#define DMARE    BIT5_SET          /* SCI:                                        */
#define T8       BIT6_SET          /* SCI:                                        */
#define R8       BIT7_SET          /* SCI:                                        */
```

```
#define SCS1                        (*(volatile char*)(0x16))
// @tiny volatile char SCS1 @0x16;/* SCI status register 1                          */
#define PE       BIT0_SET           /* SCI:                                        */
#define FE       BIT1_SET           /* SCI:                                        */
#define NF       BIT2_SET           /* SCI:                                        */
#define OR       BIT3_SET           /* SCI:                                        */
#define IDLE     BIT4_SET           /* SCI:                                        */
#define SCRF     BIT5_SET           /* SCI: Received data available (1)
                                       or not available (0) in SCDR*/
#define TC       BIT6_SET           /* SCI:                                        */
#define SCTE     BIT7_SET           /* SCI: SCDR data not (0) transferred
                                       to transmit shift register*/

#define SCS2                        (*(volatile char*)(0x17))
// @tiny volatile char SCS2 @0x17;/* SCI status register 2                          */
#define RPF      BIT0_SET           /* SCI:                                        */
#define BKF      BIT1_SET           /* SCI:                                        */

#define SCDR                        (*(volatile char*)(0x18))
// @tiny volatile char SCDR @0x18;/* SCI data register                              */
#define SCBR                        (*(volatile char*)(0x19))
// @tiny volatile char SCBR @0x19;/* SCI baud rate                                  */
#define SCP_1    0x00               /* SCI baud rate prescaling: 1                 */
#define SCP_3    BIT4_SET           /* SCI baud rate prescaling: 3                 */
#define SCP_4    BIT5_SET           /* SCI baud rate prescaling: 4                 */
#define SCP_13   BIT4_SET+BIT5_SET/* SCI baud rate prescaling: 13                   */
#define SCR_1    0x00               /* SCI baud rate selection: 1                  */
#define SCR_2    BIT0_SET           /* SCI baud rate selection: 2                  */
#define SCR_4    BIT1_SET           /* SCI baud rate selection: 4                  */
#define SCR_8    BIT0_SET+BIT1_SET/* SCI baud rate selection: 8                    */
#define SCR_16   BIT2_SET           /* SCI baud rate selection: 16                 */
#define SCR_32   BIT0_SET+BIT2_SET/* SCI baud rate selection: 32                   */
#define SCR_64   BIT1_SET+BIT2_SET/* SCI baud rate selection: 64                   */
#define SCR_128  BIT0_SET+BIT1_SET+BIT2_SET /* SCI baud rate selection: 128       */

#if F_BUS == 49
#if BAUDRATE == 9600
#define  SCBR_Register 0x21
#endif                              // BAUDRATE
#endif                              // B_BUS

#endif
```

```
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      END of SCI Declarations for the HC08 Family*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */


/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      IRQ, Keyboard, Time Base Module (only HC08GP-Family)*/
/*      Declarations for the HC08 Family*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */

#if HC08AZ == TRUE

#define ISCR   (*(volatile char*)(0x1A))
// @tiny volatile char ISCR @0x1A;/* IRQ control/status register */
#define MODE1BIT0_SET /* IRQ1 Edge/Level select bit  */
#define IMASK1BIT1_SET/* Masks all external interrupts*/
#define ACK1BIT2_SET/* IRQ Acknowledge          */
#define IRQ1FBIT3_SET/* IRQ1 Flag bit        */

#define KBSCR (*(volatile char*)(0x1B))
// @tiny volatile char KBSCR @0x1B;/* Keyboard control/status reg */
#define KBICR (*(volatile char*)(0x21))
// @tiny volatile char KBICR  @0x21;/* Keyboard interrupt register */
#endif

#if HC08GP == TRUE

#define INTKBSCR (*(volatile char*)(0x1A))
// @tiny volatile char INTKBSCR @0x1A;/* Keyboard status and control register*/
#define MODEKBIT0_SET      */
#define IMASKKBIT1_SET      */
#define ACKKBIT2_SET      */

#define INTKBIER (*(volatile char*)(0x1B))
// @tiny volatile char INTKBIER @0x1B;/* Keyboard interrupt enable register*/
#define TBCR (*(volatile char*)(0x1C))
// @tiny volatile char TBCR @0x1C;/* Time base module control register*/
#define TBTSTBIT0_SET/* PTM test mode*/
#define TBONBIT1_SET/* Timebase enabled (= 1)*/
#define TBIEBIT2_SET/* Timebase interrupt enabled (= 1)*/
#define TACKBIT3_SET/* Timebase ACKnowlege
                 (1 =clear TB IRQ flag*/
#define TBM_10x00/* Timebase Rate Selection: 1 Hz = 1000 ms*/
#define TBM_4BIT4_SET/* Timebase Rate Selection: 4 Hz = 250 ms*/
#define TBM_16BIT5_SET/* Timebase Rate Selection: 16 Hz = 62.5 ms*/
#define TBM_256BIT4_SET+BIT5_SET/* Timebase Rate Selection: 256 Hz = 3.9 ms*/
#define TBM_512BIT6_SET/* Timebase Rate Selection: 512 Hz = 2 ms*/
#define TBM_1024 BIT4_SET+BIT6_SET/* Timebase Rate Selection: 1024 Hz = 1 ms*/
#define TBM_2048 BIT4_SET+BIT5_SET+BIT6_SET/* Timebase Rate Selection:
                       4096 Hz = 0.24 ms           */
#define TBIFBIT7_SET/* Timebase Interrupt Flag (1 = pending)*/

#define INTSCR (*(volatile char*)(0x1D))
// @tiny volatile char INTSCR @0x1D;/* IRQ status and control register*/

#endif
```

```
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      END of IRQ, Keyboard, Time Base Module (only HC08GP-Family)*/
/*      Declarations for the HC08 Family*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */

/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      PLL Declarations for the HC08 Family*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */

#if HC08AZ == TRUE

#define PCTL   (*(volatile char*)(0x1C))
// @tiny volatile char PCTL @0x1C;/* PLL control register */
#define PBWC   (*(volatile char*)(0x1D))
// @tiny volatile char PBWC @0x1D;/* PLL bandwidth register */
#define PPG    (*(volatile char*)(0x1E))
// @tiny volatile char PPG  @0x1E;/* PLL programming register */

#endif

#if HC08GP == TRUE

#define PCTL   (*(volatile char*)(0x36))
#define VPR_00x00/* VPR0, VPR1: E = 0, VCO power-of-two
                 range multiplier = 1    */
#define VPR_1BIT0_SET/* VPR0, VPR1: E = 1, VCO power-of-two
                 range multiplier = 2    */
#define PR_2BIT1_SET/* VPR0, VPR1: E = 2, VCO power-of-two
                 range multiplier = 4    */
#define PRE_00x00/* PRE0, PRE1: P = 0,
                 Prescaler Multiplier = 1*/
#define PRE_1BIT0_SET/* PRE0, PRE1: P = 1,
                 Prescaler Multiplier = 2*/
#define PRE_2BIT1_SET/* PRE0, PRE1: P = 2,
                 Prescaler Multiplier = 4*/
#define PRE_3BIT0_SET+BIT1_SET/* PRE0, PRE1: P = 3,
                 Prescaler Multiplier = 8         */
#define BCS  BIT4_SET/* Base clock select bit*/
               /* 1 = CGMVCLK divided by 2 drives CGMOUT*/
               /* 0 = CGMXCLK divided by 2 drives CGMOUT*/
#define PLLONBIT5_SET/* PLL on bit (1 = on)*/
#define PLLFBIT6_SET/* PLL Interrupt flag bit
                 (1 = change in lock condition)*/
#define PLLIEBIT7_SET/* PLL Interrupt enable bit
                 (1 = enabled)        */

#define PBWC   (*(volatile char*)(0x37))
// @tiny volatile char PBWC @0x37;/* PLL bandwidth register */
#define ACQ  BIT5_SET/* Acquisition mode bit*/
               /* 1 = Tracking mode, 0 = Acquisition mode*/
#define LOCKBIT6_SET/* Lock indicator bit, 1 = locked*/
#define AUTOBIT7_SET/* Automatic bandwidth control bit*/
               /* 1 = automatic, 0 = manual*/
```

```
#define PMS    (*(volatile int*)(0x38))
//@tiny volatile  int  PMS  @0x38;/* PLL multiplier select register */
#define PMSH   (*(volatile char*)(0x38))
// @tiny volatile char PMSH @0x38;/* PLL multiplier select register high-byte*/
#define PMSL   (*(volatile char*)(0x39))
//@tiny volatile  char PMSL @0x39;/* PLL multiplier select register low-byte */
#define PMRS   (*(volatile char*)(0x3A))
// @tiny volatile char PMRS @0x3A;/* PLL VCO select range register */
#define PMDS   (*(volatile char*)(0x3B))
// @tiny volatile char PMDS @0x3B;/* PLL reference divider select register */

#if F_BUS == 2
#define N_FACTOR0xF5
#define L_FACTOR0xD1
#define R_FACTOR0x01
#endif         // F_BUS 2.0 MHz

#if F_BUS == 24
#define N_FACTOR0x12C
#define L_FACTOR0x80
#define R_FACTOR0x01
#endif         // F_BUS 2.4576 MHz

#if F_BUS == 25
#define N_FACTOR0x132
#define L_FACTOR0x83
#define R_FACTOR0x01
#endif         // F_BUS 2.5 MHz

#if F_BUS == 4
#define N_FACTOR0x1E9
#define L_FACTOR0xD1
#define R_FACTOR0x01
#endif         // F_BUS 4.0 MHz

#if F_BUS == 49
#define N_FACTOR0x258
#define L_FACTOR0x80
#define R_FACTOR0x01
#endif         // F_BUS 4.9152 MHz

#if F_BUS == 5
#define N_FACTOR0x263
#define L_FACTOR0x82
#define R_FACTOR0x01
#endif         // F_BUS 5.0 MHz

#if F_BUS == 73
#define N_FACTOR0x384
#define L_FACTOR0xC0
#define R_FACTOR0x01
#endif         // F_BUS 7.3728 MHz
```

```
#if F_BUS == 8
#define N_FACTOR0x3D1
#define L_FACTOR0xD0
#define R_FACTOR0x01
#endif         // F_BUS 8.0 MHz

#endif         // HC08GP

/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      END of PLL Declarations for the HC08 Family*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */

/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      Option/Config Declarations for the HC08 Family*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */

#if HC08AZ == TRUE

#define MORA   (*(volatile char*)(0x1F))
// @tiny volatile char MORA @0x1f;/* mask option register 1 08AZxx,
                  not avail. on 08AZ0!          */
#define CONFIG1 (*(volatile char*)(0x1F))
// @tiny volatile char CONFIG1 @0x1f;/* Configuration register 1, only 908AZ60*/
#define COPD BIT0_SET/* 0 = COP enable, 1 = COP disable*/
#define STOP BIT1_SET/* 0 = STOP instruction treated as
                  illegal opcode & 1 = enabled*/
#define COPRSBIT2_SET/* 0 = COP timeout period is
                  0 = 2**13-2**4 & 1 = 2**18-2**4*/
#define COPL BIT2_SET/* 0 = COP timeout period is
                  0 = 2**13-2**4 & 1 = 2**18-2**4*/
#define SSRECBIT3_SET/* STOP mode recovery after
                  1 = 4096 & 0 = 32 CGMXCLK cycles*/
#define LVIPWRDBIT4_SET/* LVI power 0 = disabled & 1 = enabled*/
#define LVIPWRBIT4_SET/* LVI module 0 = disabled & 1 = enabled*/
#define LVIRSTDBIT5_SET/* LVI resets 0 = disabled & 1 = enabled*/
#define LVIRSTBIT5_SET/* LVI resets 0 = disabled & 1 = enabled*/
#define SEC  BIT6_SET/* ROM security 0 = disabled & 1 = enabled*/
#define LVISTOPBIT7_SET/* LVI 0 = disabled & 1 = enabled
                  during STOP mode     */

#define CONFIG2 (*(volatile char*)(0xFE09))
               /* Configuration register 1, only 908AZ60*/
#define AZxx BIT0_SET/* 0 = HC08AS, 1 = HC08AZ*/
#define MSCANDBIT1_SET/* 0 = MSCAN enable, 1 = MSCAN disable*/

#define MORB   (*(volatile char*)(0x3F))
// @tiny volatile char MORB @0x3f;/* mask option register B */
#endif
```

```
#if HC08GP == TRUE
#define CONFIG2 (*(volatile char*)(0x1E))
// @tiny volatile char CONFIG2 @0x1E;/* Configuration register 2*/
#define SCIBDSRCBIT0_SET*/
#define OSCSTOPENBBIT1_SET             */
#define PMPSGVLVENBIT2_SET             */

#define CONFIG1 (*(volatile char*)(0x1F))
// @tiny volatile char CONFIG1 @0x1F;/* Configuration register 1, only 908AZ60*/
#define COPDBIT0_SET/* 0 = COP enable, 1 = COP disable*/
#define STOPBIT1_SET/* 0 = STOP instruction treated as illegal
                  opcode & 1 = enabled       */
#define SSRECBIT2_SET                */
#define LVI5OR3BIT3_SET                 */
#define LVIPWRDBIT4_SET                 */
#define LVIRSTDBIT5_SET                 */
#define LVISTOPBIT6_SET                 */
#define COPRSBIT7_SET                   */
#endif

/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      END of Option/Config Declarations for the HC08 Family*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */

/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      Timer A/1 Declarations for the HC08 Family*/
/*      Timer A is HC08AZ nomenclature, Timer 1 is HC08GP nomenclature*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */

#if (HC08AZ || HC08GP) == TRUE
#define TASC   (*(volatile char*)(0x20))
// @tiny volatile  char TASC   @0x20;/* Timer A status/ctrl register*/
#define TACNT  (*(volatile char*)(0x21))
// @tiny volatile  int  TACNT  @0x21;/* Timer A counter register */
#define TACNTH (*(volatile char*)(0x21))
// @tiny volatile  char TACNTH @0x21;/* Timer A counter high */
#define TACNTL (*(volatile char*)(0x22))
// @tiny volatile  char TACNTL @0x22;/* Timer A counter low */
#define TAMOD  (*(volatile char*)(0x23))
// @tiny volatile  int  TAMOD  @0x23;/* Timer A modulo register */
#define TAMODH (*(volatile char*)(0x23))
// @tiny volatile  char TAMODH @0x23;/* Timer A modulo high */
#define TAMODL (*(volatile char*)(0x24))
// @tiny volatile  char TAMODL @0x24;/* Timer A modulo low */
#define TASC0  (*(volatile char*)(0x25))
// @tiny volatile  char TASC0  @0x25;/* Timer A chan 0 status/ctrl */
#define TACH0  (*(volatile char*)(0x26))
// @tiny volatile  int  TACH0  @0x26;/* Timer A chan 0 register */
#define TACH0H (*(volatile char*)(0x26))
// @tiny volatile  char TACH0H @0x26;/* Timer A chan 0 high */
#define TACH0L (*(volatile char*)(0x27))
// @tiny volatile  char TACH0L @0x27;/* Timer A chan 0 low */
#define TASC1  (*(volatile char*)(0x28))
// @tiny volatile  char TASC1  @0x28;/* Timer A chan 1 status/ctrl */
```

```
#define TACH1  (*(volatile char*)(0x29))
// @tiny volatile int  TACH1 @0x29;/* Timer A chan 1 register */
#define TACH1H (*(volatile char*)(0x29))
// @tiny volatile char TACH1H @0x29;/* Timer A chan 1 high */
#define TACH1L (*(volatile char*)(0x2A))
// @tiny volatile char TACH1L @0x2A;/* Timer A chan 1 low */

#define T1SC   (*(volatile char*)(0x20))
// @tiny volatile char T1SC @0x20;/* Timer 1 status/ctrl register*/
#define IB1  0x00/* PS0-PS2: Internal bus / 1*/
#define IB2  0x01/* PS0-PS2: Internal bus / 2*/
#define IB4  0x02/* PS0-PS2: Internal bus / 4*/
#define IB8  0x03/* PS0-PS2: Internal bus / 8*/
#define IB16 0x04/* PS0-PS2: Internal bus / 16*/
#define IB32 0x05/* PS0-PS2: Internal bus / 32*/
#define IB64 0x06/* PS0-PS2: Internal bus / 64*/
#define TRST BIT4_SET/* Timer reset bit*/
#define TSTOPBIT5_SET/* Timer stop bit*/
#define TOIE BIT6_SET/* Timer overflow interrupt enable */
#define TOF  BIT7_SET/* Timer Overflow flag bit*/

#define T1CNT  (*(volatile int*)(0x21))
// @tiny volatile int  T1CNT @0x21;/* Timer 1 counter register */
#define T1CNTH (*(volatile char*)(0x21))
// @tiny volatile char T1CNTH @0x21;/* Timer 1 counter high */
#define T1CNTL (*(volatile char*)(0x22))
// @tiny volatile char T1CNTL @0x22;/* Timer 1 counter low */
#define T1MOD  (*(volatile int*)(0x23))
// @tiny volatile int  T1MOD @0x23;/* Timer 1 modulo register */
#define T1MODH (*(volatile char*)(0x20))
// @tiny volatile char T1MODH @0x23;/* Timer 1 modulo high */
#define T1MODL (*(volatile char*)(0x24))
// @tiny volatile char T1MODL @0x24;/* Timer 1 modulo low */

#define T1SC0  (*(volatile char*)(0x25))
// @tiny volatile char T1SC0 @0x25;/* Timer 1 chan 0 status/ctrl */
#define CH0MAXBIT0_SET/* Timer 1, Channel 0 Maximum duty
                  cycle bit*/
#define TOV0BIT1_SET/* Timer 1, Channel 0 Toggle on
                  overflow bit      */
#define CH0IEBIT6_SET/* Timer 1, Channel 0 interrupt
                  enable bit        */
#define CH0FBIT7_SET/* Timer 1, Channel 0 flag bit*/
#define IC_R0x04// Input Capture, Rising edge
#define IC_F0x08// Input Capture, Falling edge
#define IC_RF0x0C// Input Capture, Rising or Falling edge
#define OC_T0x14// Output Compare, Toggle
#define OC_C0x18// Output Compare, Clear
#define OC_S0x1C// Output Compare, Set
#define BOC_T0x24// Buffered, Toggle
#define BOC_C0x28// Buffered, Clear
#define BOC_S0x2C// Buffered, Set
```

```
#define T1CH0                                (*(volatile int*)(0x26))
// @tiny volatile int  T1CH0 @0x26;          /* Timer 1 chan 0 register          */
#define T1CH0H                               (*(volatile char*)(0x26))
// @tiny volatile char T1CH0H @0x26;         /* Timer 1 chan 0 high              */
#define T1CH0L                               (*(volatile char*)(0x27))
// @tiny volatile char T1CH0L @0x27;         /* Timer 1 chan 0 low               */

#define T1SC1                                (*(volatile char*)(0x28))
// @tiny volatile char T1SC1 @0x28;          /* Timer 1 chan 1 status/ctrl       */
#define CH1MAX      BIT0_SET                 /* Timer 1, Channel 0 Maximum duty
                                                cycle bit                        */
#define TOV1        BIT1_SET                 /* Timer 1, Channel 0 Toggle on
                                                overflow bit                     */
#define CH1IE       BIT6_SET                 /* Timer 1, Channel 0 interrupt
                                                enable bit                       */
#define CH1F        BIT7_SET                 /* Timer 1, Channel 0 flag bit      */

#define T1CH1                                (*(volatile int*)(0x29))
// @tiny volatile int  T1CH1 @0x29;          /* Timer 1 chan 1 register          */
#define T1CH1H                               (*(volatile char*)(0x29))
// @tiny volatile char T1CH1H @0x29;         /* Timer 1 chan 1 high              */
#define T1CH1L                               (*(volatile char*)(0x2A))
// @tiny volatile char T1CH1L @0x2A;         /* Timer 1 chan 1 low               */
#endif

#if HC08AZ == TRUE
#define TASC2                                (*(volatile char*)(0x2C))
//@tiny volatile  char TASC2  @0x2c;         /* Timer A chan 2 status/ctrl       */
#define TACH2                                (*(volatile char*)(0x2D))
// @tiny volatile int  TACH2  @0x2d;         /* Timer A chan 2 register          */
#define TACH2H                               (*(volatile char*)(0x2D))
// @tiny volatile char TACH2H @0x2d;         /* Timer A chan 2 high              */
#define TACH2L                               (*(volatile char*)(0x2E))
// @tiny volatile char TACH2L @0x2e;         /* Timer A chan 2 low               */
#define TASC3                                (*(volatile char*)(0x2F))
// @tiny volatile char TASC3  @0x2f;         /* Timer A chan 3 status/ctrl       */
#define TACH3                                (*(volatile char*)(0x30))
// @tiny volatile int  TACH3  @0x30;         /* Timer A chan 3 register          */
#define TACH3H                               (*(volatile char*)(0x30))
// @tiny volatile char TACH3H @0x30;         /* Timer A chan 3 high              */
#define TACH3L                               (*(volatile char*)(0x31))
// @tiny volatile char TACH3L @0x31;         /* Timer A chan 3 low               */
#endif
```

```
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      END of Timer A/1 Declarations for the HC08 Family*/
/*      Timer A is HC08AZ nomenclature, Timer 1 is HC08GP nomenclature*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */

/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      Timer B/2 Declarations for the HC08 Family*/
/*      Timer B is HC08AZ nomenclature, Timer 2 is HC08GP nomenclature*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */

#if HC08AZ == TRUE
#define TBSC   (*(volatile char*)(0x40))
// @tiny volatile  char TBSC   @0x40;/* timer B status/ctrl register */
#define TBCNT  (*(volatile char*)(0x41))
// @tiny volatile  int  TBCNT  @0x41;/* timer B counter register */
#define TBCNTH (*(volatile char*)(0x41))
// @tiny volatile  char TBCNTH @0x41;/* timer B counter high */
#define TBCNTL (*(volatile char*)(0x42))
// @tiny volatile  char TBCNTL @0x42;/* timer B counter low */
#define TBMOD  (*(volatile char*)(0x43))
// @tiny volatile  int  TBMOD  @0x43;/* timer B modulo register */
#define TBMODH (*(volatile char*)(0x43))
// @tiny volatile  char TBMODH @0x43;/* timer B modulo high */
#define TBMODL (*(volatile char*)(0x44))
// @tiny volatile  char TBMODL @0x44;/* timer B modulo low */
#define TBSC0  (*(volatile char*)(0x45))
// @tiny volatile  char TBSC0  @0x45;/* timer B chan 0 status/ctrl*/
#define TBCH0  (*(volatile char*)(0x46))
// @tiny volatile  int  TBCH0  @0x46;/* timer B chan 0 register */
#define TBCH0H (*(volatile char*)(0x46))
// @tiny volatile  char TBCH0H @0x46;/* timer B chan 0 high */
#define TBCH0L (*(volatile char*)(0x47))
// @tiny volatile  char TBCH0L @0x47;/* timer B chan 0 low */
#define TBSC1  (*(volatile char*)(0x48))
// @tiny volatile  char TBSC1  @0x48;/* timer B chan 1 status/ctrl */
#define TBCH1  (*(volatile char*)(0x49))
// @tiny volatile  int  TBCH1  @0x49;/* timer B chan 1 register*/
#define TBCH1H (*(volatile char*)(0x49))
// @tiny volatile  char TBCH1H @0x49;/* timer B chan 1 high */
#define TBCH1L (*(volatile char*)(0x4A))
//@tiny volatile   char TBCH1L @0x4a;/* timer B chan 1 low */
#endif

#if HC08GP == TRUE
#define T2SC   (*(volatile char*)(0x2B))
// @tiny volatile  char T2SC   @0x2B;  /* Timer 2 status/ctrl register*/
#define T2CNT  (*(volatile char*)(0x2C))
// @tiny volatile  int  T2CNT  @0x2C;  /* Timer 2 counter register */
#define T2CNTH (*(volatile char*)(0x2C))
// @tiny volatile  char T2CNTH @0x2C;  /* Timer 2 counter high */
#define T2CNTL (*(volatile char*)(0x2D))
// @tiny volatile  char T2CNTL @0x2D;  /* Timer 2 counter low */
#define T2MOD  (*(volatile char*)(0x2E))
// @tiny volatile  int  T2MOD  @0x2E;  /* Timer 2 modulo register */
#define T2MODH (*(volatile char*)(0x2E))
```

```
// @tiny volatile  char T2MODH @0x2E;  /* Timer 2 modulo high */
#define T2MODL (*(volatile char*)(0x2F))
// @tiny volatile  char T2MODL @0x2F;  /* Timer 2 modulo low */
#define T2SC0  (*(volatile char*)(0x30))
// @tiny volatile  char T2SC0  @0x30;  /* Timer 2 chan 0 status/ctrl*/
#define T2CH0  (*(volatile char*)(0x31))
// @tiny volatile  int  T2CH0  @0x31;  /* Timer 2 chan 0 register */
#define T2CH0H (*(volatile char*)(0x31))
// @tiny volatile  char T2CH0H @0x31;  /* Timer 2 chan 0 high */
#define T2CH0L (*(volatile char*)(0x32))
// @tiny volatile  char T2CH0L @0x32;  /* Timer 2 chan 0 low */
#define T2SC1  (*(volatile char*)(0x33))
// @tiny volatile  char T2SC1  @0x33;  /* Timer 2 chan 1 status/ctrl */
#define T2CH1  (*(volatile char*)(0x34))
// @tiny volatile  int  T2CH1  @0x34;  /* Timer 2 chan 1 register*/
#define T2CH1H (*(volatile char*)(0x34))
// @tiny volatile  char T2CH1H @0x34;  /* Timer 2 chan 1 high */
#define T2CH1L (*(volatile char*)(0x35))
// @tiny volatile  char T2CH1L @0x35;  /* Timer 2 chan 1 low */
#endif

/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      END of Timer B/2 Declarations for the HC08 Family*/
/*      Timer B is HC08AZ nomenclature, Timer 2 is HC08GP nomenclature*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */


/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      Analog to Digital Converter Declarations for the HC08 Family*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */

#if HC08AZ == TRUE
#define ADSCR  (*(volatile char*)(0x38))
               // ADC status and control register
#define ADCO BIT5_SET// ADC continuous conversion bit
#define AIEN BIT6_SET// ADC interrupt enable bit
#define COCOBIT7_SET// Conversion complete flag

#define ADR    (*(volatile char*)(0x39))
// @tiny volatile  char ADR    @0x39;  /* A/D data register */
#define ADCLKR (*(volatile char*)(0x3A))
// @tiny volatile  char ADCLKR @0x3a;  /* A/D clock register */
#define EBIC   (*(volatile char*)(0x3B))
// @tiny volatile  char EBIC   @0x3b;  /* EBI control register */
#endif

#if HC08GP == TRUE
#define ADSCR  (*(volatile char*)(0x3C))
               // ADC status and control register
#define ADCO BIT5_SET// ADC continuous conversion bit
#define AIEN BIT6_SET// ADC interrupt enable bit
#define COCOBIT7_SET// Conversion complete flag
```

```
#define ADR    (*(volatile char*)(0x3D))
#define ADCLK  (*(volatile char*)(0x3E))
#define ADC_Power_Off0x00// ADC Channel 0 select
#define ADC_Channel_10x01// ADC Channel 1 select
#define DC_Channel_20x02// ADC Channel 2 select
#define ADC_Channel_30x03// ADC Channel 3 select
#define ADC_Channel_40x04// ADC Channel 4 select
#define ADC_Channel_50x05// ADC Channel 5 select
#define ADC_Channel_60x06// ADC Channel 6 select
#define ADC_Channel_70x07// ADC Channel 7 select
#define ADC_VREFH0x1D// ADC VREFH select
#define ADC_VREFL0x0E// ADC VREFL select

#if F_BUS == 2
#define ADIV0x30// ADC input clock : 2
#endif      // 2.0 MHz

#if F_BUS == 24
#define ADIV0x30// ADC input clock : 2
#endif      // 2.4576 MHz

#if F_BUS == 25
#define ADIV0x30// ADC input clock : 2
#endif      // 2.5 MHz

#if F_BUS == 4
#define ADIV0x50// ADC input clock : 4
#endif      // 4.0 MHz

#if F_BUS == 49
#define ADIV0x50// ADC input clock : 4
#endif      // 4.9152 MHz

#if F_BUS == 5
#define ADIV0x50// ADC input clock : 4
#endif      // 5 MHz

#if F_BUS == 73
#define ADIV0x50// ADC input clock : 4
#endif      // 73728 MHz

#if F_BUS == 8
#define ADIV0x70// ADC input clock : 8
#endif      // 8.0 MHz

#endif
```

```
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      END of Analog to Digital Converter Declarations for the HC08 Family*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */


/*  PIT                                                 */

#if HC08AZ == TRUE
#define PSC    (*(volatile char*)(0x4B))
// @tiny volatile  char PSC    @0x4b;  /* PIT status/ctrl register */
#define PCNT   (*(volatile char*)(0x4C))
// @tiny volatile  int  PCNT   @0x4c;  /* PIT counter register */
#define PCNTH  (*(volatile char*)(0x4C))
// @tiny volatile  char PCNTH  @0x4c;  /* PIT counter high */
#define PCNTL  (*(volatile char*)(0x4D))
// @tiny volatile  char PCNTL  @0x4d;  /* PIT counter low */
#define PMOD   (*(volatile char*)(0x4E))
// @tiny volatile  int  PMOD   @0x4e;  /* PIT modulo register */
#define PMODH  (*(volatile char*)(0x4E))
// @tiny volatile  char PMODH  @0x4e;  /* PIT modulo high */
#define PMODL  (*(volatile char*)(0x4F))
// @tiny volatile  char PMODL  @0x4f;  /* PIT modulo low */
#endif

//#pragma space []

/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      SIM/FLASH/BREAK Declarations for the HC08 Family*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */

#if HC08AZ == TRUE
#define SBSR   (*(volatile char*)(0xFE00))
// volatile  char SBSR   @0xfe00;   /* SIM break status register*/

#define SRSR   (*(volatile char*)(0xFE01))
// volatile  char SRSR   @0xfe01;   /* SIM reset status register*/

#define SBFCR  (*(volatile char*)(0xFE03))
// volatile  char SBFCR  @0xfe03;   /* SIM break control register*/

#define BRK    (*(volatile char*)(0xFE0C))
// volatile  int  BRK    @0xfe0c;   /* BREAK address register */

#define BRKH   (*(volatile char*)(0xFE0C))
// volatile  char BRKH   @0xfe0c;   /* BREAK address register low*/

#define BRKL   (*(volatile char*)(0xFE0D))
// volatile  char BRKL   @0xfe0d;   /* BREAK address register high*/

#define BRKSCR (*(volatile char*)(0xFE0E))
// volatile  char BRKSCR @0xfe0e;   /* BREAK status/ctrl register*/

#define LVISR  (*(volatile char*)(0xFE0F))
// volatile  char LVISR  @0xfe0f;   /* LVI status register */
```

```
#define EENVR  (*(volatile char*)(0xFE1C))
// volatile  char EENVR  @0xfe1c;   /* EEPROM non-volatile register*/

#define EECR   (*(volatile char*)(0xFE1D))
// volatile  char EECR   @0xfe1d;   /* EEPROM control register*/

#define EEACR  (*(volatile char*)(0xFE1F))
// volatile  char EEACR  @0xfe1f;   /* EEPROM array ctrl register*/

#define COPCTL (*(volatile char*)(0xFFFF))
// volatile  char COPCTL @0xffff;   /* COP control register */
#endif

#if HC08GP == TRUE

#define SBSR   (*(volatile char*)(0xFE00))
// volatile  char SBSR   @0xfe00;   /* SIM break status register*/

#define SRSR   (*(volatile char*)(0xFE01))
// volatile  char SRSR   @0xfe01;   /* SIM reset status register*/

#define SUBAR  (*(volatile char*)(0xFE02))
// volatile  char SUBAR  @0xfe02;   /* SIM upper byte address register*/

#define SBFCR  (*(volatile char*)(0xFE03))
// volatile  char SBFCR  @0xfe03;   /* SIM break control register*/

#define INT1   (*(volatile char*)(0xFE04))
// volatile  char INT1   @0xfe04;   /* SIM interrupt status register 1*/
#define I1  0x04// IRQ
#define I2  0x08// PLL
#define I3  0x10// TIM1, CH0
#define I4  0x20// TIM1, CH1
#define I5  0x40// TIM1, OVL
#define I6  0x80// TIM2, CH0

#define INT2   (*(volatile char*)(0xFE05))
// volatile  char INT2   @0xfe05;   /* SIM interrupt status register 2*/

#define INT3   (*(volatile char*)(0xFE06))
// volatile  char INT3   @0xfe06;   /* SIM interrupt status register 3*/

#define FLTCR  (*(volatile char*)(0xFE07))
// volatile  char FLTCR  @0xfe07;   /* FLASH test control register*/

#define FLCR   (*(volatile char*)(0xFE08))
// volatile  char FLCR   @0xfe08;   /* FLASH control register*/
#define PGM BIT0_SET/* Program control bit (1 = on)*/
#define ERASEBIT1_SET/* Erase control bit (1 = on)*/
#define MASSBIT2_SET/* Mass erase control bit (1 = on)*/
#define HVENBIT3_SET/* High-voltage enbable bit (1 = on)*/

#define BRK    (*(volatile int*)(0xFE09))
// volatile  int  BRK    @0xfe09;   /* BREAK address register */
```

```
#define BRKH   (*(volatile char*)(0xFE09))
// volatile  char BRKH   @0xfe09;   /* BREAK address register low*/
#define BRKL   (*(volatile char*)(0xFE0A))
// volatile  char BRKL   @0xfe0A;   /* BREAK address register high*/

#define BRKSCR (*(volatile char*)(0xFE0B))
// volatile  char BRKSCR @0xfe0B;   /* BREAK status/ctrl register*/

#define LVISR  (*(volatile char*)(0xFE0C))
// volatile  char LVISR  @0xfe0C;   /* LVI status register */

#define COPCTL (*(volatile char*)(0xFFFF))
// volatile  char COPCTL @0xFFFF;   /* COP control register */
#endif

#if     HC08GP20
#define FLBPR  (*(volatile char*)(0xFF80))
// volatile char FLBPR  @0xFF80;/* FLASH block protect register*/
#endif

#if     HC08GP32
#define FLBPR  (*(volatile char*)(0xFF7E))
// volatile char FLBPR  @0xFF7E;/* FLASH block protect register*/
#endif

/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
/*      END of SIM/FLASH/BREAK Declarations for the HC08 Family*/
/* HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08 * HC08   */
```

MCU_Hardware_Description.H
Hardware Description File

```
#include       "MCU_General_Statements.h"
#include       "MCU_Assembler_Instructions.h"
#include       "MCU_Hardware_Description_HC08.h"

/*****************************************************************************
 *
 * DESCRIPTION:     I/O Definitions for Motorola MCU's:
 * 68HC(9)08:       MCU_Hardware_Description_HC08.h
 *
 * FILE NAME:       MCU_Hardware_Description.h
 *
 * AUTHOR:          Harald Kreidl, Motorola GmbH, Munich,
 *                  Email: harald.kreidl@motorola.com
 *
 * HISTORY:
 *   Version 1.0, 23. September 1998 (created the file)
 *   Version 1.2, 18. August 1999 (added the HC08GP family)
 *   Version 2.0, 17. November 1999 (added MPC555, restructured file system
 *
 * Some Compiler dependent explanations:
 *
 *   1. Placing Registers in the Zero Page:
 *
 * COSMIC:
 *   @tiny places data objects selectivly in the zeropage (COSMIC C).
 *   But this statement is not understood by the Hiware Compiler.
 *   Therefor I chose:
 *      #define PTA (*(volatile char*)(0x00)).
 *  This definition is understood by both compilers but you have to force the COSMIC
 *  compiler into the small memory modell. Otherwise the zeropage instructions are
 *  not used. The Hiware Compiler puts the data automatically in the zero page.
 *  If tiny is not used force the tiny memory model (set the +zpg switch).
 *
 *  HIWARE:
 *    It is not necessary to force the tiny memory model. The Compiler does it
 *    by itself.

 *      #define PTA (*(volatile char*)(0x00))
 *      // @tiny volatile  char PTA@   0x00;
 *      #define PTA0BIT                0_SET
 *      #define PTA1BIT                1_SET
 *      #define PTA2BIT                2_SET
 *
 *      Set a bit:                     PTA         |=     PTA0;
 *      Clear a bit:                   PTA         &=     ~PTA0;
 *      Set multiple bits in a byte:   PTA         |=     PTA0+PTA1;
 *      Clear multiple bits in a byte: PTA         &=     ~(PTA0+PTA1);
 *
 *
 *****************************************************************************/
```

MCU_ASSEMBLER_INSTRUCTIONS.H
In-Line Assembler Commands

```
/*****************************************************************************
 *
 * DESCRIPTION:     Assembler Instruction Macros for Motorola MCU's:
 * 68HC(7,8)05:     MCU_Hardware_Description_HC05.h
 *
 * FILE NAME:       MCU_Assembler_Instructions.h
 *
 * AUTHOR:          Harald Kreidl, Motorola GmbH, Munich,
 *                  Email: harald.kreidl@motorola.com
 *
 * HISTORY:
 *   Version 1.0, 23. September 1998 (created the file)
 *   Version 2.0, 17. November 1999 (restructured file system)
 *
 *
 *
 *****************************************************************************/

#ifdef __HIWARE__                                       // Hiware
#define disable_interrupts()          asm SEI;
#define enable_interrupts()           asm CLI;
#define stop_MCU()                    asm STOP;
#define no_operation()                asm NOP;
#else /* COSMIC C-Compiler-vendor non-ANSI declaration of interrupt function      */
#define disable_interrupts()          _asm("sei\n")
#define enable_interrupts()           _asm("cli\n")
#define stop_MCU()                    _asm("stop\n")
#define no_operation()                _asm("nop\n")
#endif                                                  // Hiware
```

CAN_CONTROLLER.H
CAN Controller Definitions

```
#include       "MCU_Select.h"

/*****************************************************************************/
/*  Structures for the CAN Tx and Rx Buffer                                  */
/*****************************************************************************/
typedef struct
{       volatile unsigned char IDR0;
        volatile unsigned char IDR1;
        volatile unsigned char IDR2;
        volatile unsigned char IDR3;
        volatile unsigned char DSR[8];
        volatile unsigned char DLR;
} TX_MESSAGE_BUFFER;

/* Eine CAN-Message */
typedef struct {
        _U16   ID ;         /* 11 Bit-Kennung                                */
        _U08   RTR ;        /* TRUE, wenn remote request                     */
        _U08   LENGTH ;     /* Anzahl der gueltigen Datenbytes (0..8)        */
        _U08   DATA[8] ;    /* Datenbytes 0..7                               */
} TCANMsg ;

typedef struct
{
        volatile unsigned char IDR0;
        volatile unsigned char IDR1;
        volatile unsigned char IDR2;
        volatile unsigned char IDR3;
        volatile unsigned char DSR[8];
        volatile unsigned char DLR;
} RX_MESSAGE_BUFFER;


// Function Prototypes
        void   Bus_Output( void );
        void   Bus_Input( void );
        void   Address_To_Bus( _U08 );
        void   Data_To_Bus( _U08 );
        _U08   Data_From_Bus( void );
        _U08   Read_Byte( _U08 );
        void   Write_Byte( _U08, _U08 );
        void   Bit_Set( _U08, _U08 );
        void   Bit_Clear( _U08, _U08 );
        void   Initialize_CAN_Controller( void );
        _U08   CAN_Write( void );
        _U08   CAN_Read( void );
        _U08   Read_Status( void );
```

```
/*** Error Mask Bits ***/
#define CAN_ERR_OK         0x00
#define CAN_ERR_XMTFULL    0x01   /* Send buffer full in the CAN controller       */
#define CAN_ERR_RCVEMPTY   0x02   /* No new message was received                  */
#define CAN_ERR_OVERRUN    0x04   /* CAN controller message overrun               */
#define CAN_ERR_BUSERROR   0x08   /* Bus error: error counter reached limit       */
#define CAN_ERR_BUSOFF     0x10   /* Busf error: CAN controller bus off           */
#define CAN_ERR_REGTEST    0x20   /* Register test at CAN controller failed       */

#if     SJA1000            == TRUE

/************************************************************
; with sja1000 the SPI-signal are used as controll bus
************************************************************
;
; sja1000  signal write define      Bit 0    1   2   3
*/

#define        write_cycle_0      0x03    // AS=1 CS=1 E=0 WR=0
#define        write_cycle_1      0x02    // AS=0 CS=1 E=0 WR=0
#define        write_cycle_2      0x00    // AS=0 CS=0 E=0 WR=0
#define        write_cycle_3      0x04    // AS=0 CS=0 E=1 WR=0

// sja signal read define          Bit 0    1   2   3
#define        read_cycle_0       0x0B    // AS=1 CS=1 E=0 WR=1
#define        read_cycle_1       0x02    // AS=0 CS=1 E=0 WR=0
#define        read_cycle_2       0x00    // AS=0 CS=0 E=0 WR=0

#define        read_cycle_3       0x0C    // AS=0 CS=0 E=1 WR=1
#define        read_cycle_4       0x04    // AS=0 CS=1 E=0 WR=1

// define can parameter bezogen auf 16MHz Quarz, 1 sample point
#define        can_baud0_20k      0xF1
#define        can_baud1_20k      0x23    //; 62%

#define        can_baud0_50k      0x9f
#define        can_baud1_50k      0x11    // 60%
#define        can_baud0_125k     0xc3
#define        can_baud1_125k     0x58    // 62%

#define        can_baud0_250k     0xc1
#define        can_baud1_250k     0x58    // 62%

#define        can_baud0_500k     0xc0
#define        can_baud1_500k     0x58    // 62%

#define        can_baud0_1000k    0xc0
#define        can_baud1_1000k    0x23    // 62%

/* address and bit definitions for the Mode & Control Register                 */
#define ModeControlReg 0x00
#define RM_RR_Bit                 0x01    /* reset mode (request) bit          */
#define STM_Bit                   0x04    /* self test mode bit                */
```

```
#if defined (PeliCANMode)
#define LOM_Bit            0x02   /* listen only mode bit                        */
#define STM_Bit            0x04   /* self test mode bit                          */
#define AFM_Bit            0x08   /* acceptance filter mode bit                  */
#define SM_Bit             0x10   /* enter sleep mode bit                        */
#endif

/* address and bit definitions for the Interrupt Enable & Control Register      */
#if defined (PeliCANMode)
#define InterruptEnReg     0x03   /* PeliCAN mode                                */
#define RIE_Bit            0x01   /* receive interrupt enable bit                */
#define TIE_Bit            0x02   /* transmit interrupt enable bit               */
#define EIE_Bit            0x04   /* error warning interrupt enable bit          */
#define DOIE_Bit           0x08   /* data overrun interrupt enable bit           */
#define WUIE_Bit           0x10   /* wake-up interrupt enable bit                */
#define EPIE_Bit           0x20   /* error passive interrupt enable bit          */
#define ALIE_Bit           0x40   /* arbitration lost interr. enable bit         */
#define BEIE_Bit           0x80   /* bus error interrupt enable bit              */

#else /* BasicCAN mode                                                          */

#define InterruptEnReg     0x00   /* Control Register                            */
#define RIE_Bit            0x02   /* Receive Interrupt enable bit                */
#define TIE_Bit            0x04   /* Transmit Interrupt enable bit               */
#define EIE_Bit            0x08   /* Error Interrupt enable bit                  */
#define DOIE_Bit           0x10   /* Overrun Interrupt enable bit                */
#endif

/* address and bit definitions for the Command Register                         */
#define CommandReg         0x01
#define TR_Bit             0x01   /* transmission request bit                    */
#define AT_Bit             0x02   /* abort transmission bit                      */
#define RRB_Bit            0x04   /* release receive buffer bit                  */
#define CDO_Bit            0x08   /* clear data overrun bit                      */
#if defined (PeliCANMode)
#define SRR_Bit            0x10   /* self reception request bit                  */

#else                             /* BasicCAN mode                               */
#define GTS_Bit            0x10   /* goto sleep bit (BasicCAN mode)              */
#endif

/* address and bit definitions for the Status Register                          */
#define StatusReg          0x02
#define RBS_Bit            0x01   /* receive buffer status bit                   */
#define DOS_Bit            0x02   /* data overrun status bit                     */
#define TBS_Bit            0x04   /* transmit buffer status bit                  */
#define TCS_Bit            0x08   /* transmission complete status bit            */
#define RS_Bit             0x10   /* receive status bit                          */
#define TS_Bit             0x20   /* transmit status bit                         */
#define ES_Bit             0x40   /* error status bit                            */
#define BS_Bit             0x80   /* bus status bit                              */
```

```
/* address and bit definitions for the Interrupt Register                    */
#define InterruptReg       0x03
#define RI_Bit             0x01   /* receive interrupt bit                   */
#define TI_Bit             0x02   /* transmit interrupt bit                  */
#define EI_Bit             0x04   /* error warning interrupt bit             */
#define DOI_Bit            0x08   /* data overrun interrupt bit              */
#define WUI_Bit            0x10   /* wake-up interrupt bit                   */
#if defined (PeliCANMode)
#define EPI_Bit            0x20   /* error passive interrupt bit             */
#define ALI_Bit            0x40   /* arbitration lost interrupt bit          */
#define BEI_Bit            0x80   /* bus error interrupt bit                 */
#endif

/* address and bit definitions for the Bus Timing Registers                  */
#define BusTiming0Reg      0x06
#define BusTiming1Reg      0x07
#define SAM_Bit            0x80   /* sample mode bit
1 == the bus is sampled 3 times
0 == the bus is sampled once                                                */

/* address and bit definitions for the Output Control Register               */
#define OutControlReg      0x08
/* OCMODE1, OCMODE0                                                         */
#define BiPhaseMode        0x00   /* bi-phase output mode                    */
#define NormalMode         0x02   /* normal output mode                      */
#define ClkOutMode         0x03   /* clock output mode                       */
/* output pin configuration for TX1                                         */
#define OCPOL1_Bit         0x20   /* output polarity control bit             */
#define Tx1Float           0x00   /* configured as float                     */
#define Tx1PullDn          0x40   /* configured as pull-down                 */
#define Tx1PullUp          0x80   /* configured as pull-up                   */
#define Tx1PshPull         0xC0   /* configured as push/pull                 */
/* output pin configuration for TX0 */
#define OCPOL0_Bit         0x04   /* output polarity control bit             */
#define Tx0Float           0x00   /* configured as float                     */
#define Tx0PullDn          0x08   /* configured as pull-down                 */
#define Tx0PullUp          0x10   /* configured as pull-up                   */
#define Tx0PshPull         0x18   /* configured as push/pull                 */

/* address definitions of Acceptance Code & Mask Registers                   */
#if defined (PeliCANMode)
#define AcceptCode0Reg     0x16
#define AcceptCode1Reg     0x08
#define AcceptCode2Reg     0x08
#define AcceptCode3Reg     0x08
#define AcceptMask0Reg     0x08
#define AcceptMask1Reg     0x08
#define AcceptMask2Reg     22
#define AcceptMask3Reg     23
#else                              /* BasicCAN mode                         */
#define AcceptCodeReg      4
#define AcceptMaskReg      5
#endif
```

```
/* address definitions of the Rx-Buffer                                    */
#if defined (PeliCANMode)
#define RxFrameInfo        16
#define RxBuffer1          17
#define RxBuffer2          18
#define RxBuffer3          19
#define RxBuffer4          20
#define RxBuffer5          21
#define RxBuffer6          22
#define RxBuffer7          23
#define RxBuffer8          24
#define RxBuffer9          25
#define RxBuffer10         26
#define RxBuffer11         27
#define RxBuffer12         28
#else /* BasicCAN mode                                                     */
#define RxBuffer1          20
#define RxBuffer2          21
#define RxBuffer3          22
#define RxBuffer4          23
#define RxBuffer5          24
#define RxBuffer6          25
#define RxBuffer7          26
#define RxBuffer8          27
#define RxBuffer9          28
#define RxBuffer10         29
#endif
/* address definitions of the Tx-Buffer                                    */
#if defined (PeliCANMode)
/* write only addresses                                                    */
#define TxFrameInfo        16
#define TxBuffer1          17
#define TxBuffer2          18
#define TxBuffer3          19
#define TxBuffer4          20
#define TxBuffer5          21
#define TxBuffer6          22
#define TxBuffer7          23
#define TxBuffer8          24
#define TxBuffer9          25

#define TxBuffer10         26
#define TxBuffer11         27
#define TxBuffer12         28
/* read only addresses                                                     */
#define TxFrameInfoRd      96
#define TxBufferRd1        97
#define TxBufferRd2        98
#define TxBufferRd3        99
#define TxBufferRd4        100
#define TxBufferRd5        101
#define TxBufferRd6        102
#define TxBufferRd7        103
#define TxBufferRd8        104
#define TxBufferRd9        105
```

```
#define TxBufferRd10       106
#define TxBufferRd11       107
#define TxBufferRd12       108
#else /* BasicCAN mode                                                    */
#define TxBuffer1          10
#define TxBuffer2          11
#define TxBuffer3          12
#define TxBuffer4          13
#define TxBuffer5          14
#define TxBuffer6          15
#define TxBuffer7          16
#define TxBuffer8          17
#define TxBuffer9          18
#define TxBuffer10         19
#endif
/* address definitions of Other Registers                                */
#if defined (PeliCANMode)
#define ArbLostCapReg      11
#define ErrCodeCapReg      12
#define ErrWarnLimitReg    13
#define RxErrCountReg      14
#define TxErrCountReg      15
#define RxMsgCountReg      29
#define RxBufStartAdr      30
#endif
/* address and bit definitions for the Clock Divider Register            */
#define ClockDivideReg     31
#define DivBy1             0x07   /* CLKOUT = oscillator frequency       */
#define DivBy2             0x00   /* CLKOUT = 1/2 oscillator frequency   */
#define ClkOff_Bit         0x08   /* clock off bit,
control of the CLK OUT pin                                               */
#define RXINTEN_Bit        0x20   /* pin TX1 used for receive interrupt  */
#define CBP_Bit            0x40   /* CAN comparator bypass control bit   */
#define CANMode_Bit        0x80   /* CAN mode definition bit             */

#endif

#if     MCP2510            ==     TRUE
/*
// Function Prototypes

*/

void Spi_Write(_U08,_U08 );
void Spi_CAN_Start(void);
void Spi_trans( _U08 );
void enable_MCP( void);
void disable_MCP( void);
_U08 Spi_read(_U08 );
void Spi_BitMode(_U08,_U08,_U08);
void Initialize_CAN_Controller( void );
_U08 Spi_read_status(void);
```

mcp2510.h
Include File for MPC2510 Registers

```
//#ifndef _MCP2510_H_
//#define _MCP2510_H_

/*
** Register offsets into the transmit buffers.
*/
#define TXBnCTRL              0
#define TXBnSIDH              1
#define TXBnSIDL              2
#define TXBnEID8              3
#define TXBnEID0              4
#define TXBnDLC               5
#define TXBnD0                6
#define TXBnD1                7
#define TXBnD2                8
#define TXBnD3                9
#define TXBnD4                10
#define TXBnD5                11
#define TXBnD6                12
#define TXBnD7                13
#define CANSTAT               14
#define CANCTRL               15

#define SIDH                  0
#define SIDL                  1
#define EID8                  2
#define EID0                  3

/*
** Register offsets into the receive buffers.
*/
#define RXBnCTRL              0
#define RXBnSIDH              1
#define RXBnSIDL              2
#define RXBnEID8              3
#define RXBnEID0              4
#define RXBnDLC               5
#define RXBnD0                6
#define RXBnD1                7
#define RXBnD2                8
#define RXBnD3                9
#define RXBnD4                10
#define RXBnD5                11
#define RXBnD6                12
#define RXBnD7                13


#define RXB0DLC               0x65
#define RXB0D0                0x66
```

```
/*
** Bits in the TXBnCTRL registers.
*/
#define TXB_TXBUFE_M          0x80
#define TXB_ABTF_M            0x40
#define TXB_MLOA_M            0x20
#define TXB_TXERR_M           0x10
#define TXB_TXREQ_M           0x08
#define TXB_TXIE_M            0x04
#define TXB_TXP10_M           0x03

#define DLC_MASK              0x0F
#define RTR_MASK              0x40


#define TXB0CTRL              0x30
#define TXB0SIDH              0x31
#define TXB0SIDL              0x32
#define TXB0DCL               0x35
#define TXB0D0                0x36



#define TXB1CTRL              0x40
#define TXB1SIDH              0x41

#define TXB2CTRL              0x50
#define TXB2SIDH              0x51

#define TXPRIOHIGH            0x03
#define TXPRIOHIGHLOW         0x02
#define TXPRIOLOWHIGH         0x01
#define TXPRIOLOW             0x00

#define TXB_EXIDE_M           0x08                    // In TXBnSIDL
#define TXB_RTR_M             0x40                    // In TXBnDLC

#define RXB_IDE_M             0x08                    // In RXBnSIDL
#define RXB_RTR_M             0x40                    // In RXBnDLC

#define BFPCTRL               0x0C

#define B2RTS                 0x20
#define B1RTS                 0x10
#define B0RTS                 0x08
#define B2RTSM                0x04
#define B1RTSM                0x02
#define B0RTSM                0x01

#define TEC                   0x1C
#define REC                   0x1D
#define CLKCTRL               CANCTRL

#define RXF0SIDH              0
#define RXF0SIDL              1
```

```
#define RXF0EID8            2
#define RXF0EID0            3
#define RXF1SIDH            4
#define RXF1SIDL            5
#define RXF1EID8            6
#define RXF1EID0            7
#define RXF2SIDH            8
#define RXF2SIDL            9
#define RXF2EID8            10
#define RXF2EID0            11

#define RXF3SIDH            16
#define RXF3SIDL            17
#define RXF3EID8            18
#define RXF3EID0            19
#define RXF4SIDH            20
#define RXF4SIDL            21
#define RXF4EID8            22
#define RXF4EID0            23
#define RXF5SIDH            24
#define RXF5SIDL            25
#define RXF5EID8            26
#define RXF5EID0            27

#define RXF_EXIDE_M         0x08

#define RXM0SIDH            0x20
#define RXM0SIDL            0x21
#define RXM1SIDH            0x24
#define RXM1SIDL            0x25
#define CNF3                0x28
#define CNF2                0x29
#define CNF1                0x2A
#define CANINTE             0x2B
#define CANINTF             0x2C
#define EFLG                0x2D
#define TXRTSCTRL           0x0D

#define EFLG_RX1OVR         0x80
#define EFLG_RX0OVR         0x40
#define EFLG_TXBO           0x20
#define EFLG_TXEP           0x10
#define EFLG_RXEP           0x08
#define EFLG_TXWAR          0x04
#define EFLG_RXWAR          0x02
#define EFLG_EWARN          0x01

#define SJW1                0x00
#define SJW2                0x40
#define SJW3                0x80
#define SJW4                0xC0

#define BTLMODE_CNF3        0x80

#define SAMP1               0x00
```

```
#define SAMP3                   0x40

#define SEG1                    0x00
#define SEG2                    0x01
#define SEG3                    0x02
#define SEG4                    0x03
#define SEG5                    0x04
#define SEG6                    0x05
#define SEG7                    0x06
#define SEG8                    0x07

#define BRP1                    0x00
#define BRP2                    0x01
#define BRP3                    0x02
#define BRP4                    0x03
#define BRP5                    0x04
#define BRP6                    0x05
#define BRP7                    0x06
#define BRP8                    0x07

#define IVRIE                   0x80
#define WAKIE                   0x40
/* !!!!!!!!!!! second define !!!!
#define ERRIE                   0x20
*/
#define TX2IE                   0x10
#define TX1IE                   0x08
#define TX0IE                   0x04
#define RX1IE                   0x02
#define RX0IE                   0x01
#define NO_IE                   0x00

#define IVRINT                  0x80
#define WAKINT                  0x40
#define ERRINT                  0x20
#define TX2INT                  0x10
#define TX1INT                  0x08
#define TX0INT                  0x04
#define RX1INT                  0x02
#define RX0INT                  0x01
#define NO_INT                  0x00

#define RXB0CTRL                0x60
#define RXB1CTRL                0x70

#define RXB_RXRDY               0x80
#define RXB_RXM1                0x40
#define RXB_RXM0                0x20
#define RXB_RX_ANY              0x60
#define RXB_RX_EXT              0x40
#define RXB_RX_STD              0x20
#define RXB_RX_STDEXT           0x00
#define RXB_RXMx_M              0x60
// #define RXB_RXIE_M           0x10
#define RXB_RXRTR               0x08                    // In RXBnCTRL
```

```
#define RXB_BUKT              0x04
#define RXB_BUKT_RO           0x02

#define RXB_FILHIT            0x01
#define RXB_FILHIT2           0x04
#define RXB_FILHIT1           0x02
#define RXB_FILHIT_M          0x07
#define RXB_RXF5              0x05
#define RXB_RXF4              0x04
#define RXB_RXF3              0x03
#define RXB_RXF2              0x02
#define RXB_RXF1              0x01
#define RXB_RXF0              0x00

#define CLKEN                 0x04

#define CLK1                  0x00
#define CLK2                  0x01
#define CLK4                  0x02
#define CLK8                  0x03

#define MODE_NORMAL           0x00
#define MODE_SLEEP            0x20
#define MODE_LOOPBACK         0x40
#define MODE_LISTENONLY       0x60
#define MODE_CONFIG           0xE0
#define ABORT                 0x10

#define SPI_RESET             0xC0
#define SPI_READ              0x03
#define SPI_WRITE             0x02
#define SPI_RTS               0x80
#define SPI_STATUS            0xA0
#define SPI_BITMOD            0x05

#define RECEIVE_BUFFER(x)     (0x60 + 0x10*(x))
#define TRANSMIT_BUFFER(x)    (0x30 + 0x10*(x))

//extern void mcp_read_can ( uchar buffer, uchar* ext, unsigned long* can_id,
//                           uchar* dlc, uchar* rtr, uchar* data );

//extern void mcp_write_can_id ( uchar mcp_addr, uchar ext, unsigned long can_id );
//extern void mcp_read_can_id ( uchar mcp_addr, uchar* ext, unsigned long* can_id );

//extern void mcp_reset(void);
//extern void mcp_read ( uchar MCPaddr, uchar* readdata, uchar length );
//extern void mcp_read_all ( void );
//extern void mcp_write_tbuf ( unsigned char value );
//extern void mcp_write_can (uchar buffer, uchar ext, unsigned long can_id,
//                           uchar dlc, uchar rtr, const uchar* data );
//extern void mcp_transmit( uchar buffer );
//extern void mcp_init(void);
//extern void mcp_write ( uchar MCPaddr, const uchar* writedata, uchar length );

#endif
```

Flowcharts and Source Code Files

Appendix C. Motorola Embedded CAN MCU Selector Guide

See **Table C-1** for the Motorola Embedded CAN MCU Selector Guide. Please check with your local Motorola distributor for availability.

Table C-1. Embedded CAN Microcontroller Selector Guide

Device[1]	ROM (Bytes)	RAM (Bytes)	FLASH or OTP (Bytes)	Device Inte-gration	EEPROM (Bytes)	Timer	I/O	Serial	MUX	A/D	PWM	COP	Pkg Option	Oper. Voltage (V)	Oper. Freq. (MHz)	Temp	FLASH or OTP	Comments	Documentation
MC68HC705X4	—	176	4K	—	—	16-bit, 1 I/C, 1 O/C	16	—	CAN	—	—	Y	28 SOIC (DW)	5.0	2.1	C	—	2-MHz bus speed only	MC68HC05X4/D
MC68HC05X4	4K	176	—	—	—	16-bit, 1 I/C, 1 O/C, MFI, RTI	16	—	CAN	—	—	Y	28 SOIC (DW)	5.0	2.1	C	706X4 (limited)	2-MHz bus speed only	MC68HC05X4/D
MC68HC908AZ60A	—	2K	60K FLASH	—	1K	8-CH+2-CH, 16-bit, I/C, O/C, or PWM	50	SCI, SPI	CAN 2.0a/2.0b	15-CH 8-bit	See timer	Y	64 QFP (FU)	5.0	8.0 Max	C, V, M	—		MC68HC908AZ60A/D
MC68HC08AZ32A	32K	1K	—	—	512	4-CH+4-CH, 16-bit, I/C, O/C, or PWM	48	SCI, SPI	CAN 2.0a/2.0b	15-CH 8-bit	See timer	Y	64 QFP (FU)	5.0	8.0 Max	C, V, M	908AZ60A		MC68HC08AZ32A/D
MC68HC08AZ60	60K	2K	—	—	1K	6-CH+2-CH, 16-bit, I/C, O/C, or PWM	48	SCI, SPI	CAN 2.0a/2.0b	15-CH 8-bit	See timer	Y	64 QFP (FU)	5.0	8.0 Max	C, V, M	908AZ60A		MC68HC08AZ60/D
MC9S12DP256	—	12K	256K FLASH	—	4K	8-CH, 16-bit	Up to 45	2 SCI, 1 SPI	Up to 5 CAN and 1 x J1850	2x8-CH, 10-bit	8-CH, 8-bit or 4-CH, 16-bit	Y	112 LQFP 80 QFP	5.0	25.0	C, V, M[2]	—		MC9S12DP256/D
MC68HC912BC32	—	1K	32K FLASH	—	768	8-CH, 16-bit I/C or O/C, RTI, pulse accumulator	Up to 63	SCI, SPI	CAN J1850	8-CH, 10-bit	4-CH, 8-bit or 2-CH, 16-bit	Y	80 QFP (FU)	5.0	8.0 Max	C, V, M	—	msCAN CAN 2.0a and 2.0b, BDM	MC68HC912B32TS/D
MC68HC912D60A	—	2K	60K FLASH	—	1K	8-CH, 16-bit enhanced capture timer (ECT)	Up to 66, plus up to 18 input-only lines	2 SCI, 1 SPI	CAN 2.0a/2.0b	2 x 8-CH, 10-bit	4-CH, 8-bit or 2-CH, 16-bit	Y	112 QFP	5.0	8.0	C, V, M[2]	—	0.5 µ technology, 5 V FLASH, MC plan Q2 2001	MC68HC912D60/D
MC68HC912DG128A	—	8K	128K FLASH	—	2K	8-CH, 16-bit buffered input captures	Up to 67, plus up to 18 input-only lines	2 SCI, 1 SPI	2 x CAN 2.0a/2.0b	2 x 8-CH, 10-bit	4-CH, 8-bit or 2-CH,16-bit	Y	112 LQFP	5.0	8.0	C, V, M[2]	—	0.5µ technology, 5 V FLASH, ideal for gateway applications. Replacement for XC68HCDG128	MC68HC912DT128A/D
MC68HC912DT128A	—	8K	128K FLASH	—	2K	8-CH, 16-bit buffered input captures	Up to 67, plus up to 18 input-only lines	2 SCI, 1 SPI	3 x CAN 2.0a/2.0b	2 x 8-CH, 10-bit	4-CH, 8-bit or 2-CH,16-bit	Y	112 LQFP	5.0	8.0	C, V, M[2]	—	0.5µ technology, 5 V FLASH, ideal for Gateway applications	MC68HC912DT128A/D
MPC555/6	0	26K + 6K for TPU	448K	USIU	—	50-CH timer system: 2 TPU3 + MIOS1	—	QSMCM (2 SCI + QSPI) + 2 TOUCAN	2 x TOUCAN	2 QADC64 (10-bit A/D with 64 result registers each)	8 x PWM	—	272 PBGA	3.3 Vdc for core, 5.0 Vdc for FLASH	40.0	M	—	MPC556 offers code compression support	MPC555UM/AD TPURM/AD RCPURM/AD

1. Please check with your local Motorola distributor for availability.
2. M temperature range limited to single-chip mode.

Appendix D. MC33388 Motorola CAN Interface

D.1 Contents

D.2 Introduction 156

D.3 Features 157

D.4 Pin Connections and Ordering Information. 158

D.5 Electrical Characteristics 159
D.5.1 Maximum Ratings. 159
D.5.2 Thermal Ratings 159
D.5.3 DC Characteristics 160
D.5.4 AC Characteristics 163

D.6 Device Description 165
D.6.1 Packaging. 165
D.6.2 Transmitter Function 165
D.6.3 Receiver Function 166
D.6.4 Noise Filtering 166
D.6.5 Device Operation Modes 166
D.6.6 Operation Modes 168
D.6.7 System Power On 169
D.6.8 V_{DD} Reset Function 170
D.6.9 Battery Fail Flag 170
D.6.10 Bus Failure Detection 170
D.6.11 TX Permanent Dominant Detection 171
D.6.12 Behavior Under Faults Condition 171
D.6.13 Detailed Description of Error Detections 171
D.6.14 Wakeup Events 173
D.6.15 Fault Operation Table. 173

D.7 Pin Function Description. 174
D.7.1 V_{BAT} (Input) 174
D.7.2 V_{DD} (Input) 174

D.7.3 CANH 174
D.7.4 RTH 175
D.7.5 CANL 175
D.7.6 RTL 175
D.7.7 STB and EN 176
D.7.8 INH 176
D.7.9 WAKE 176
D.7.10 TX 177
D.7.11 RX 177
D.7.12 NERR 177

D.8 Application 178

D.9 Electromagnetic Compatibility 180
D.9.1 EMI Noise 180
D.9.2 EMC Susceptibility Performances 181
D.9.3 Susceptibility Evaluation with BCI 182
D.9.4 Results 182

D.10 Case Outline Dimensions 183

D.2 Introduction

The MC33388 is a controller area network (CAN) physical interface device dedicated to automotive body electronics multiplexing applications. It operates in differential mode, allowing ground shifts up to 1.5 V, reducing radio frequency interference (RFI) disturbances. It offers very low standby current in sleep mode and standby mode operation and supports communication speeds up to 125 kbaud.

The MC33388 is fully protected against harsh automotive environments and the driver is able to detect fault conditions and automatically switch into appropriate default mode. Under default conditions, it continuously monitors bus failures and switches back to normal bus operation as soon as faults disappear.

See **Figure D-1** for a simplified block diagram.

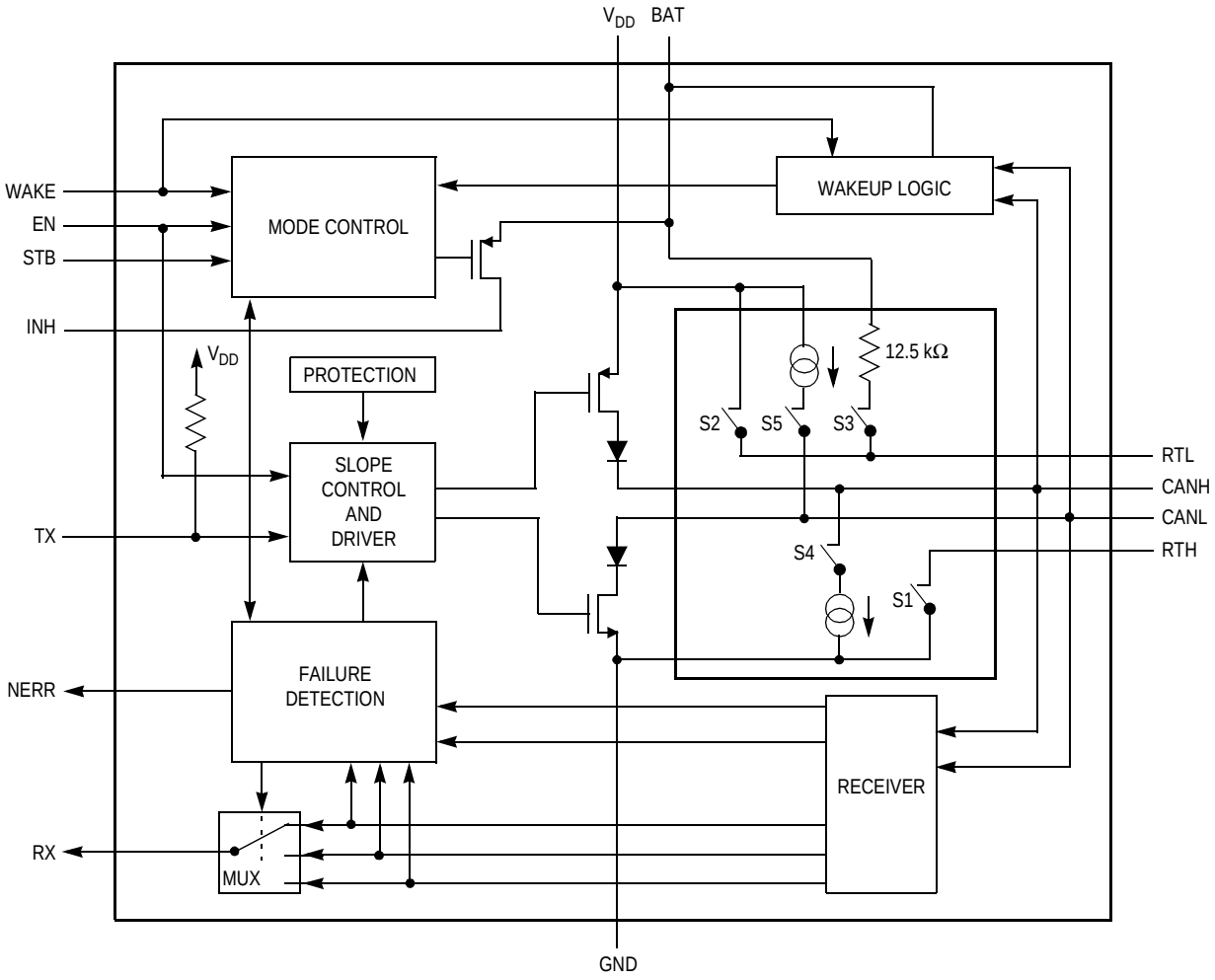


Figure D-1. Simplified Block Diagram

D.3 Features

Features include:

- Very low sleep/standby current (15 μA/typical)
- Baud rate from 10 kbaud up to 125 kbaud
- Automatic switching to single-wire mode in case of bus failures and return to differential mode if bus failures disappear
- Supports 1-wire transmission modes with ground offset up to 1.5 V
- Internal bus driver slope control function to minimize RFI

Features continued:

- Bus line short circuit protected to battery, V_{DD}, and ground
- Bus line protected against automotive transients
- Thermal protection of bus line drivers
- Supports unshielded twisted pair bus
- An unpowered node does not disturb the bus lines
- Wakeup capability triggered from bus message and wakeup input pin
- Wakeup pin with dual edge sensitivity
- Battery fail flag reported on NERR output
- Ambient temperature range from –40° C to 125° C

D.4 Pin Connections and Ordering Information

Figure D-2 shows the pin connections for the MC33388 with ordering information given in **Table D-1**. See **D.10 Case Outline Dimensions**.

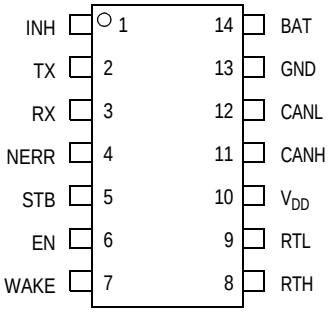


Figure D-2. Pin Connections

Table D-1. Ordering Information

Device	Operating Temperature Range	Package
MC33388D	T_A = –40° C to 125° C	SO-14

D.5 Electrical Characteristics

D.5.1 Maximum Ratings

Ratings	Symbol	Min	Max	Unit
DC supply voltage pin 10	V_{DD}	–0.3	6	V
DC voltage on pins 2, 3, 4, 5, 6, and 7	V_{DD}	–0.3	V_{DD} + 0.3	V
DC voltage on pins 11 and 12	V_{BUS}	–20	+27	V
Transient voltage at pins 11 and 12 $0 < V_{DD} < 5.5$ V; $V_{BAT} \geq 0$; T < 500 ms	V_{CANH}/V_{CANL}	–40	40	V
Transient voltage on pins 11 and 12 (coupled through 1 nF capacitor)	V_{TR}	–150	100	V
DC voltage on pin 7	V_{Wake}	—	V_{BAT} + 0.3	V
Current in pin 7	I_{Wake}	–15	3	mA
DC voltage on pin 1	V_{INH}	–0.3	V_{BAT} + 0.3	V
DC voltage on pins 8 and 9	V_{RTL}, V_{RTH}	–0.3	40	V
DC voltage on pin 14	V_{BAT}	–0.3	27	V
Voltage on pin 14 (load dump, 500 ms)	V_{BAT}	—	40	V
ESD voltage on any pins (HBM.100 pF; 1.5 kΩ)	V_{ESD}	–3.0	3.0	kV
ESD voltage on any pins (MM.200 pF; 0 Ω)	V_{ESD}	–200	200	V
Junction temperature	T_J	–40	150	°C
Storage temperature	T_{STG}	–55	150	°C
RTH and RTL termination resistance	R_T	500	16,000	Ω

D.5.2 Thermal Ratings

Rating	Symbol	Value	Unit
Thermal resistance from junction to ambient	$R_{THJ/A}$	120	°C/W

D.5.3 DC Characteristics

Conditions[1]	Symbol	Min	Typ	Max	Unit
SUPPLY					
V_{DD} supply current (normal mode) TX = V_{DD}, recessive state	I_{VDD}	—	2.3	3	mA
V_{DD} supply current (normal mode) TX = 0 V, no load, dominant state	I_{VDD}	—	3.3	5	mA
V_{BAT} supply current (normal mode) TX = V_{DD}	I_{BAT}	—	150	300	µA
Total supply current (receive-only mode) V_{DD} = 5 V; V_{BAT} = 12 V	$I_{VDD} + I_{BAT}$	—	0.85	1.2	mA
Total supply current (V_{BAT} standby mode) V_{DD} = 5 V; V_{BAT} = 12 V	$I_{VDD} + I_{BAT}$	—	20	40	µA
V_{BAT} supply current (sleep mode) V_{DD} = 0 V; V_{BAT} = 12 V	I_{BAT}	—	15	25	µA
STB, EN, and TX Pins					
High-level input voltage	V_{IH}	0.7*V_{DD}	—	V_{DD} +0.3	V
Low-level input voltage	V_{IL}	–0.3	—	0.3*V_{DD}	V
High-level input current (STB, EN) (V_I = 4 V)	I_{IH}	—	20	40	µA
Low-level input current (STB, EN) (V_I = 1 V)	I_{IL}	10	20	—	µA
TX high-level input current (V_I = 4 V)	I_{TX}	–25	–80	–200	µA
TX low-level input current (V_I = 1 V)	I_{TX}	–100	–320	–800	µA
Forced V_{BAT} standby mode (fail safe) threshold	V_{DD}	3	4	4.7	V
Battery voltage for setting power on flag	V_{BAT}	1.5	3	4	V
RX and NERR Pins					
High-level output voltage NERR (I_O = –100 µA)	V_{OH}	V_{DD} –0.9	—	V_{DD}	V
High-level output voltage RX (I_O = –250 µA)	V_{OH}	V_{DD} –0.9	—	V_{DD}	V
Low-level output voltage (I_O = 1.5 mA)	V_{OL}	0	—	0.9	V
WAKE Pin (must be connected to GND or BAT if not used)					
Typical wakeup threshold (V_{STB} = 0 V), high-to-low transition, V_{BAT} = 6 V to 18 V[2]	$Wu_{threshl}$	—	0.44 V_{BAT}	—	V

Continued on next page

Conditions[1]	Symbol	Min	Typ	Max	Unit
Typical wakeup threshold (V_{STB} = 0 V), low-to-high transition, V_{BAT} = 6 V to 18 V[2]	$Wu_{threslh}$	—	0.57 V_{BAT}	—	V
Wakeup threshold hysteresis	Wu_{hyst}	500	—	—	mV
Wakeup threshold, high-to-low transition at V_{BAT} = 12 V	Wu_{hi}	3.6	—	6.5	V
Wakeup threshold, low-to-high transition at V_{BAT} = 12 V	Wu_{lh}	6.2	—	7.5	V
INH Pin					
High-level voltage drop (I_{INH} = –0.2 mA, INH high)	V_{Drop}	0	—	0.8	V
Leakage current (sleep mode; V_{INH} = 0 V)	I_{LINH}	0	—	5	µA
CANH and CANL Pins					
Differential receiver, recessive-to-dominant threshold (by definition, $V_{DIFF} = V_{CANH} - V_{CANL}$)	V_{DIFF1}	–3.2	—	–2.5	V
Differential receiver, dominant-to-recessive threshold (bus failures 1, 2, and 5)	V_{DIFF2}	–3.2	—	–2.5	V
CANH recessive output voltage TX = V_{DD}; $R_{(RTH)}$ < 4 k	V_{CANH}	—	—	0.2	V
CANL recessive output voltage TX = V_{DD}; $R_{(RTL)}$ < 4 k	V_{CANL}	V_{DD} –0.2	—	—	V
CANH output voltage, dominant TX = 0 V; I_{CANH} = –40 mA; normal operating mode	V_{CANH}	V_{DD} –1.4	—	—	V
CANL output voltage, dominant TX = 0 V; I_{CANL} = 40 mA, normal operating mode	V_{CANL}	—	—	1.4	V
CANH output compare (V_{CANH} = 0; TX = 0	I_{CANH}	50	75	100	mA
CANL output current (V_{CANL} = 14 V; TX = 0)	I_{CANL}	50	90	130	mA
Detection threshold for short-circuit to battery voltage (normal mode)	V_{CANH}, V_{CANL}	7.3	7.9	8.9	V
Detection threshold for short-circuit to battery voltage (standby/sleep mode)	V_{CANH}	$V_{BAT}/2 + 3$	—	$V_{BAT}/2 + 5$	V
CANH output current (sleep mode; V_{CANH} = 12 V, failure 3)	—	—	5	10	µA

Continued on next page

Conditions[1]	Symbol	Min	Typ	Max	Unit
CANL output current (sleep mode; V_{CANL} = 0 V; V_{BAT} = 12 V, failure 4)	I_{CANL}	—	0	2	µA
CANL wakeup voltage threshold	V_{WakeL}	2.5	3.3	3.9	V
CANH wakeup voltage threshold	V_{WakeH}	1.2	2	2.7	V
Wakeup threshold difference (hysteresis)	V_{WakeL}– V_{WakeH}	0.2	—	—	V
CANH single-ended receiver threshold (failures 4, 6, and 7)	$V_{SE,CANH}$	1.5	1.85	2.15	V
CANL single-ended receiver threshold (failures 3 and 8)	$V_{SE,CANL}$	2.8	3.05	3.4	V
CANL pullup current (normal mode, failures 4, 6,and 7)	$I_{CANL,PU}$	45	75	90	µA
CANH pulldown current (normal mode, failure 3)	$I_{CANH.PD}$	45	75	90	µA
Receiver differential input impedance CANH/CANL	R_{DIFF}	100	—	180	kΩ
Differential receiver common mode voltage range	V_{COM}	–10	—	10	V
CANH-to-ground capacitance	C_{CANH}	—	—	50	pF
CANL-to-ground capacitance	C_{CANL}	—	—	50	pF
C_{CANL}-to-C_{CANH} capacitance difference (absolute value)	DC_{CAN}	—	—	10	pF
RTH and RTL Pins					
RTL-to-V_{DD} switch on resistance (I_{Out} < –10 mA; normal operating mode)	R_{RTL}	10	30	50	Ω
RTL-to-BAT switch series resistance (V_{BAT} standby mode or sleep mode)	R_{RTL}	8	12.5	20	kΩ
RTH-to-ground switch on resistance (I_{Out} < 10 mA; normal operating mode)	R_{RTH}	10	25	50	Ω
Thermal Shutdown					
Shutdown temperature	T_{SD}	—	165	—	°C

1. V_{DD} = 4.75 to 5.25; V_{BAT} = 6 to 27 V; T_{AMB} = –40 to 125° C, unless otherwise noted
2. When V_{BAT} is greater than 18 V, the wakeup thresholds remain identical to the wakeup thresholds at 18 V.

D.5.4 AC Characteristics

Conditions[1]	Symbol	Min	Typ	Max	Unit
CANL and CANH slew rate (10% to 90%) rising or falling edges[2]	t_{SL}	3.5	5	10	V/µs
Propagation delay TX-to-RX low[2]	t_{PDLow}	—	1	2	µs
Propagation delay TX-to-RX high[2]	t_{PDHigh}	—	1	2	µs
Minimum dominant time for wakeup on CANL or CANH (V_{BAT} standby and sleep modes; V_{BAT} = 12 V)	t_{Wake}	8	16	30	µs
Minimum WAKE time for wakeup (V_{BAT} standby and sleep modes; V_{BAT} = 12 V)	t_{Wake}	6	15	30	µs
Failure 3 detection time (normal mode)	t_{DF3}	10	—	60	µs
Failure 6 detection time (normal mode)	t_{DF6}	50	—	400	µs
Failure 3 recovery time (normal mode)	t_{DR3}	10	—	60	µs
Failure 6 recovery time (normal mode)	t_{DR6}	150	—	1000	µs
Failure 4, 7, and 8 detection time (normal mode)	t_{DR478}	0.75	—	4	ms
Failure 4, 7, and 8 recovery time (normal mode)	t_{DR478}	10	—	60	µs
Failure 3, 4, 7, and 8 detection time V_{BAT} standby and sleep modes; V_{BAT} = 12 V)	t_{DF347}	0.8	—	8	ms
Failure 3, 4, 7, and 8 recovery time (V_{BAT} standby and sleep modes; V_{BAT} = 12 V)	t_{DR347}	—	2.5	—	ms
Minimum hold time for “go-to-sleep” command	t_{GTS}	4	—	38	µs
Edge count difference between CANH and CANL for failures 1, 2, and 5 detection (NERR becomes low), (normal mode)	E_{CDF}	—	3	—	
Edge count difference between CANH and CANL for failures 1, 2, and 5 recovery (normal mode)	E_{CDR}	—	3	—	
TX permanent dominant timer disable time (normal mode and failure mode)	$t_{TX,d}$	0.75	—	4	ms
TX permanent dominant timer enable time (normal mode and failure mode)	$t_{TX,e}$	10	—	60	µs

1. V_{DD} = 4.75 to 5.25; V_{BAT} = 6 to 27 V; T_{AMB} = –40 to 125° C, unless otherwise noted
2. AC characteristics measured according to the schematic shown in **Figure D-4**.

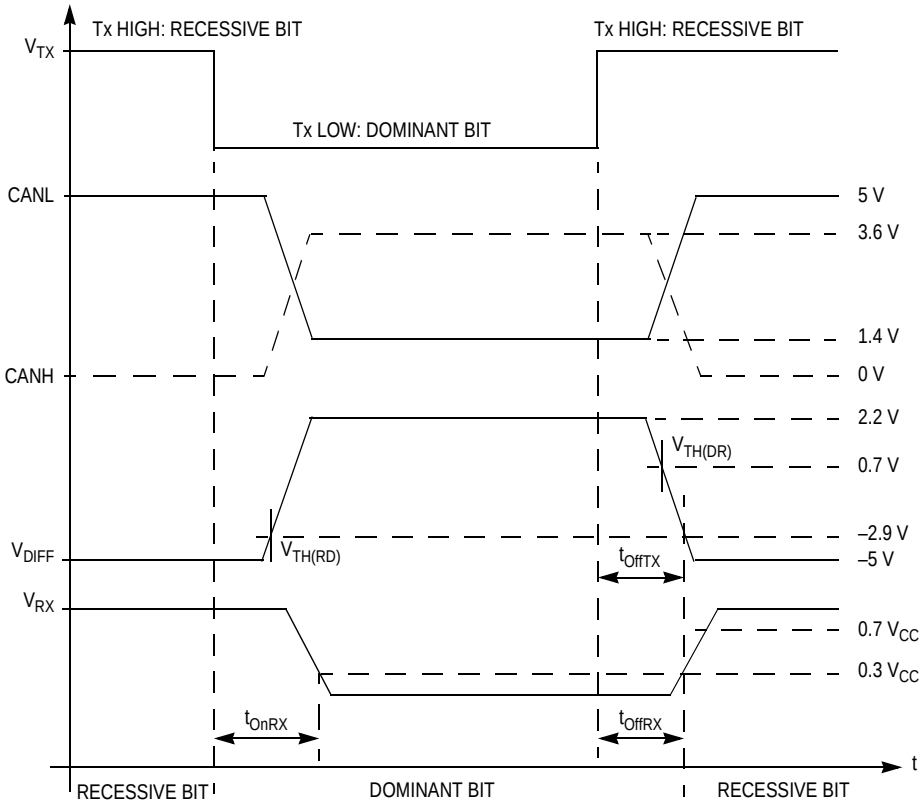


Figure D-3. Device Signal Waveforms

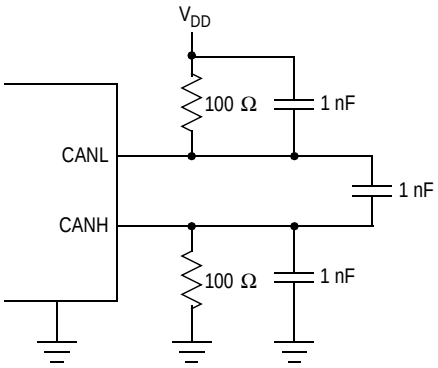


Figure D-4. Test Circuit for AC Characteristics

D.6 Device Description

The MC33388 is a low-speed CAN fault tolerant physical interface designed for automotive multiplexed electronic systems. The MC33388 addresses the low-speed body electronics application, in which the speed of communications is between 10 kbaud and 125 kbaud, on two wire bus configurations. It is designed to operate in the harsh automotive environment.

The MC33388 can control the external voltage regulator of the system through the dedicated INH pin. It allows the application to be switched into low-power mode. Wakeup can be done either from bus activity or local wakeup switch. The MC33388 is tolerant to faults occurring at the CAN bus in normal operating mode and low-power mode.

D.6.1 Packaging

The device is assembled in an SO-14 narrow body package. Thermal performances allow the device to operate in the automotive ambient temperature range from –40° C to 125° C.

D.6.2 Transmitter Function

CAN bus levels are called dominant and recessive and correspond respectively to the low and high states of the TX input pin. The recessive state is a weak state, where bus lines are driven through pullup and pulldown resistors. Recessive state can be overwritten by any other node forcing a dominant state, in which bus lines are driven through active switches.

The bus is terminated by pullup and pulldown resistors which are connected to GND, V_{DD}, or V_{BAT} through dedicated RTL and RTH pins and internal circuitry.

The bus line slew rates are controlled to minimize the RFI allowing use of unshielded cables for the bus.

D.6.3 Receiver Function

In normal operation (no bus failures), RX is the image of the differential bus voltage. The differential receiver inputs are connected to CANH and CANL through integrated filters. The filtered input signals are also used for the single-wire receivers.

The device incorporates comparators connected to CANH and CANL in order to monitor and report the bus state to the microcontroller as well as to detect bus failures. Failures are reported to the microcontroller through the NERR pin.

In normal operation when no failure is present, the differential comparator is active. Under a fault condition, one of the two CANH or CANL pins can be non-operational. Then, the single-ended comparator of either CANH or CANL is activated and continues to report bus states to the microcontroller. The MC33388 permanently monitors bus failure and recovery, and as soon as fault disappears, it automatically switches back to differential operation.

D.6.4 Noise Filtering

The device is optimized for dual-wire operation. During all single-wire transmissions, the electromagnetic compliance (EMC) performance in both immunity and emission are worse than in differential mode. Integrated receiver filters suppress any high frequency (HF) noise induced into the bus wires. The cut-off frequency of these filters is a compromise between propagation delay and HF suppression. In single-wire mode, low frequency noise can not be distinguished from the active signal at the bus line.

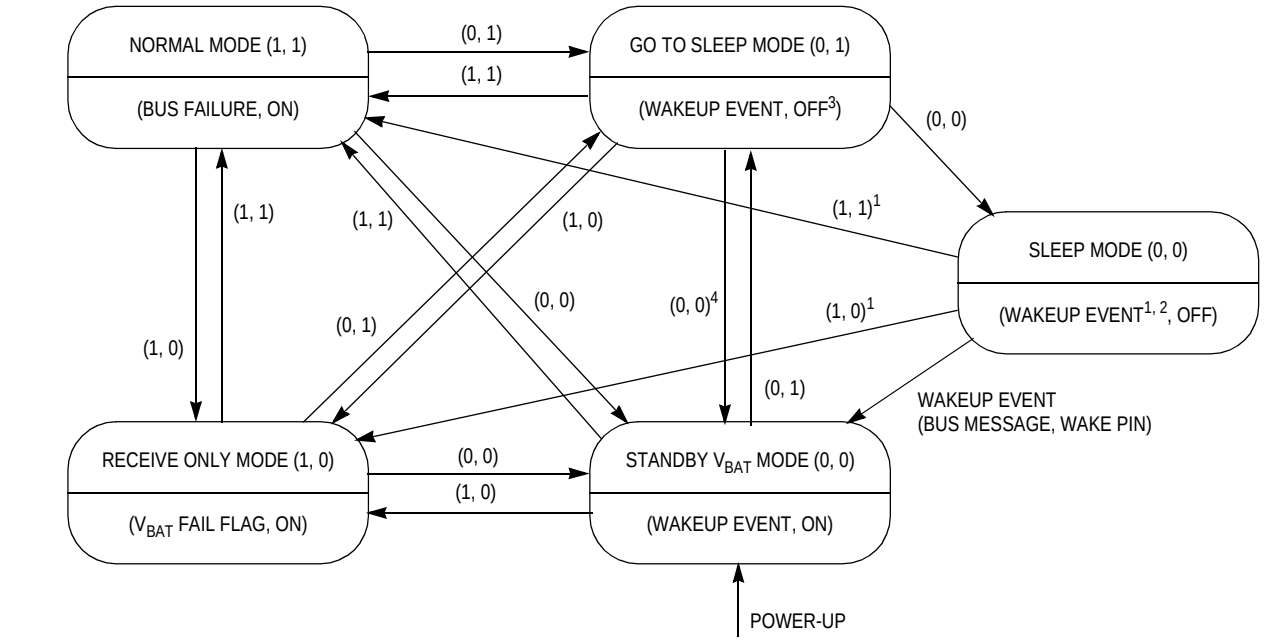
D.6.5 Device Operation Modes

The MC33388 has four operation modes:

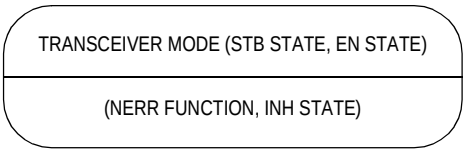
- Normal
- Receive only
- Standby V_{BAT}
- Sleep

Each of these modes is controlled by the state of the EN and STB pins.

The state machine shown in **Figure D-5** and the truth table in **Table D-2** indicate how to configure the device into each mode and the pin functions in each operation mode.



LEGEND:



NERR:
- BUS — BUS ERROR IS FLAGGED.
- WAKE — WAKEUP EVENT IS FLAGGED.
- BAT — BAT FAIL IS FLAGGED.

INH:
- ON — INH IS SWITCHED ON.
- OFF — INH IS FLOATING.

Notes:
1. With permanent V_{DD}
2. V_{DD} is still switched on.
3. If no wakeup occurs after 25 µs
4. As long as INH is on

Figure D-5. State Machine and Operation Modes

Table D-2. Truth Table

STB	EN	Mode	INH	NERR	RX	RTL	I_{DD}(1)	I_{BAT}(1)
0	0	V_{BAT} standby(2)	High	Active low: wakeup interrupt signal (if V_{DD} is present)		Switched to V_{BAT}	5 µA	Typical 15 µA
0	0	Sleep(3)	Floating			Switched to V_{BAT}	Not applicable	Typical 15 µA
0	1	Go-to-sleep command	Floating			Switched to V_{BAT}	—	—
1	0	Receive only(4)	High	Active low: V_{BAT} power-on flag	High: recessive state Low: dominant state	Switched to V_{DD}	800 µA	—
1	1	Normal	High	Active low: error flag		Switched to V_{DD}	2.3 mA(5) 3.3 mA(6)	Typical 150 µA

1. Values are typical, without bus load current.
2. Wakeup interrupts are released when entering normal operating mode.
3. If the go-to-sleep command was used before EN, it may turn low as V_{DD} drops without affecting internal functions because of the fail safe functionality.
4. V_{BAT} power-on flag will be reset when entering normal operation mode.
5. In recessive state
6. In dominant state, value with no load at bus

D.6.6 Operation Modes

Normal Mode

In this mode, all functions are available and the NERR pin reports bus failure.

Receive-Only Mode

In this mode, the transmitter path is disabled, so the device does not drive the bus. It maintains CANL and CANH in recessive state. The receiver function operates normally. As the device can not drive the bus, an incoming CAN message could not be acknowledged by the node. NERR output signals the V_{BAT} power-on flag and RX reports bus state. Failure detection and management are the same as in normal mode.

Sleep Mode

In this mode, the transmitter and receiver functions are disabled. The CANL pin is connected to V_{BAT} through the RTL resistor and an internal pullup resistor of 12.5 kΩ. The INH pin is switched in high-impedance state. The external voltage regulator connected to INH will be switched off and no V_{DD} is supplied to the MC33388. In this mode the device is still supplied by the V_{BAT}. Supply current from V_{BAT} is 15 µA typical. The MC33388 monitors the bus activity and the state of WAKE pin and V_{BAT} level. If wakeup conditions are encountered, the device wakes up to standby V_{BAT} mode and INH is switched on.

Standby V_{BAT} Mode

This mode is similar to sleep mode, but the INH pin is in the high state to maintain the external 5-V regulator activated. Wakeup events are directly reported to NERR and RX thanks to the 5 V available at V_{DD}. CANL is the same configuration as in sleep mode.

NOTE: *Standby and sleep modes are active when STB and EN are low. Selection of standby or sleep mode is done through the activation sequence of the EN and STB pins. Sleep mode is entered through an intermediate step (go to sleep) where STB and EN are 0, 1. See* ***Table D-2****.*

D.6.7 System Power On

When the supply is first applied to the system, V_{BAT} and V_{DD} rise from 0 up to their nominal value and the device automatically enters into V_{BAT} standby mode. At this time, INH is switched in high state to activate the external voltage regulator and an internal flag is set (batt fail flag). EN and STB pins are internally forced in low state to maintain the device into V_{BAT} standby mode.

The V_{DD} “forced V_{BAT} standby mode (fail safe)” circuit will maintain the device in V_{BAT} standby mode until V_{DD} is higher than 3 V, whatever the external state of EN and STB.

As soon as V_{DD} reaches the “forced V_{BAT} standby mode (fail safe)” threshold, the device can enter into other modes, depending on EN and STB state.

D.6.8 V_{DD} Reset Function

If during operation V_{DD} drops below “forced V_{BAT} standby mode (fail safe)” threshold, the device is automatically switched into V_{BAT} standby mode to provide fail safe functionality.

D.6.9 Battery Fail Flag

When the V_{BAT} supply drops below “battery voltage for setting power-on flag” threshold, this information is internally latched. This means that the system power supply has been lost (disconnected and reconnected for instance).

This flag can be read by the microcontroller by switching the device into receive-only mode, where the NERR pin reports the V_{BAT} power-on flag. This flag is reset by entering into the normal mode.

D.6.10 Bus Failure Detection

The device permanently monitors the bus lines and detects faults in normal and receive-only modes. Failures detected at the bus level are listed here:

1. CANH wire interruption
2. CANL wire interruption
3. CANH short-circuit to battery
4. CANL short-circuit to ground
5. CANH short-circuit to ground
6. CANL short-circuit to battery
7. CANL and CANH mutually shorted
8. CANH short-circuit to V_{DD}

D.6.11 TX Permanent Dominant Detection

In addition to the failures previously listed, the MC33388 detects a permanent low state at TX input which results in a permanent dominant bus state. The MC33388 detects if TX is low for more than 2 ms typical and then disables the bus output driver in order to switch into recessive state. This avoids blocked communication between other nodes of the network.

D.6.12 Behavior Under Faults Condition

When a fault is detected, the device automatically takes appropriate action to minimize the system current consumption and to allow communication on the network. Depending on the type of fault, the mode of operation and the fault detected, the device automatically switches off one or more of these functions:

- CANL or CANH line driver
- RTL or RTH pullup or pulldown resistors or internal switches

These actions are detailed in **Table D-3**.

The device permanently monitors the faults and in case of fault recovery, it automatically switches back to normal operation and reconnects the open functions. Fault detection and recovery circuitry have internal filters and delay timing detailed in **D.5.4 AC Characteristics**.

D.6.13 Detailed Description of Error Detections

The differential receiver threshold is set at –2.8 V. This ensures a proper reception in the normal operating modes and in case of failures 1, 2, and 5 (**Table D-3**) noise margin as high as possible. These failures or their recovery do not destroy ongoing transmissions.

Failures 3 and 6 (**Table D-3**) are detected by comparators respectively connected to CANH and CANL. If the comparator threshold is exceeded for a certain time, the device is switched to single-wire mode. This time is needed to avoid false triggering by external RF fields. Recovery from these failures is detected automatically after a certain timeout (filtering) and no transmission is lost.

Table D-3. Detail Fault Operation Table

Failure Number	Description	Mode	State of S1, S2, and S3 Internal Switches and CANL CANH Output Drivers[1] (Refers to Normal Mode)
No failure	—	Normal and receive only	S1 and S2 closed, CANL and CANH drivers enabled
No failure	—	V_{BAT} standby	S3 closed, CANL and CANH drivers disabled
No failure	—	Sleep mode	S3 closed, CANL and CANH drivers disabled
1	CANH wire interrupted	Normal and receive only	S1 and S2 closed, CANL and CANH drivers enabled
2	CANL wire interrupted	Normal and receive only	S1 and S2 closed, CANL and CANH drivers enabled
3	CANH short to BAT	All	S1 open, CANH driver disabled
4	CANL short to GND	All	S2 and S3 open, CANL driver disabled
5	CANH short to GND	Normal and receive only	S1 and S2 closed, CANL and CANH drivers enabled
6	CANL short to BAT	Normal and receive only	S2 and S3 open, CANL driver disabled
7	CANL short to CANH	All	S2 and S3 open, CANL driver disabled
8	CANH short to V_{DD}	All	S1 open, CANH driver disabled

1. S4 and S5 behaviors are only dependent on the device operating mode.
S4 and S5 switches are closed in normal and receive-only modes and are open in sleep and standby modes.

Failures 4, 7, and 8 (**Table D-3**) initially result in a permanent dominant level at the internal comparator outputs. If failures 4 and 7 appear, the CANL driver and the RTL pin are switched off after a timeout, and only a weak pullup at RTL remains. Reception continues by switching to single-wire mode through CANH. When the failures 4 or 7 are removed, the recessive bus levels are restored. If the receiver voltages remain in the recessive state for a certain time, reception and transmission switch back to the differential mode.

If failure 8 (**Table D-3**) is recognized, the CANH driver is switched off after a timeout and the reception is switched to single-wire mode through CANL. If the receiver voltages remain in the recessive state for a certain time, reception and transmission switch back automatically to the differential mode.

If any of the eight wiring failures occur, the output NERR will be switched low. When the error recovers, NERR will be switched back to the high state.

D.6.14 Wakeup Events

Wakeup requests are recognized by the MC33388 in sleep and V_{BAT} standby modes, either when a dominant state is detected on CANL or CANH bus lines (remote wakeup) or if the WAKE pin changes state (local wakeup).

Under power-up conditions when V_{BAT} is higher than 5 V, the state voltage on the WAKE pin is considered the reference state for the wakeup function. On leaving normal mode, the current WAKE pin state becomes the new reference state.

In sleep mode, on a wakeup request the transceiver sets the INH output high to activate the external voltage regulator used for V_{DD} supplied. In V_{BAT} standby, INH is already set high. When V_{DD} is set, the wakeup request can be read on the NERR or RX outputs by the microcontroller.

To prevent false wakeup due to transients or RF fields, wakeup threshold levels have to be maintained for a certain time. In the low-power modes, failure detection circuit remains partly active to prevent increased power consumption in cases of errors 3, 4, 7, and 8.

D.6.15 Fault Operation Table

Table D-3 shows the device operation in normal and low-power modes and the internal actions happening under fault conditions. Refer to **Figure D-1** for device internal switch references.

D.7 Pin Function Description

D.7.1 V_{BAT} (Input)

This is the supply line of the device. It can be directly connected to the battery line, can operate up to 27 Vdc, and sustain up to 40 V during load dump condition. The supply current is dependent on the device operation mode. In low-power mode, it is as low as 12 μA typical. A battery fail flag circuitry is associated to this pin.

The V_{BAT} pin must be protected by an external component against reverse battery and negative transient voltages.

D.7.2 V_{DD} (Input)

This pin is the 5-V input supply voltage for the device. In normal mode, the current is up to mA typical and does not exceed mA in V_{BAT} standby mode. An undervoltage function is associated to V_{DD} and resets the device to V_{BAT} standby mode when V_{DD} falls below 3 V.

D.7.3 CANH

CANH is the bus output driver pin. CANH is a high-side switch structure connected to V_{DD} supply. In recessive state, the high-side switch is off and the bus CANH line is biased through the RTH pin circuitry and external RTH resistor. In dominant state, the high-side switch is on and the CANH line is switched to V_{DD}. Output voltage is above 3.6 V. CANH output is protected against short-circuit to ground by internal current limitation. A thermal shutdown with hysteresis switches off the CANH driver if the temperature rises above 150° C. CANH output is protected against short-circuit to higher voltage, such as V_{BAT}, by an internal serial diode in series with the switching component. CANH has a pulldown current source, typically 75 μA, which can be activated under some fault conditions.

D.7.4 RTH

RTH is the connection to the CANH bus terminal resistors. RTH output structure is a low-side switch, turned on under normal conditions and automatically deactivated under some fault conditions. In the application, an external resistor is connected between RTH and CANH pins. In recessive state when CANH driver is off, CANH line is tied to ground through the external RTH resistor.

D.7.5 CANL

CANL is the bus output driver pin. CANL is a low-side switch structure to the ground. In recessive state, the low-side switch is off, and bus CANL line is biased through the RTL circuitry and external RTL resistor. In dominant state, low-side switch is on and CANL line is switched to ground. Output voltage is below 1.4 V. CANL output is protected against short-circuit to V_{BAT} by internal current limitation. A thermal shutdown with hysteresis is integrated and switches off CANL driver if temperature rises above 150° C. CANL is protected against negative voltage, by an internal serial diode in series with the switching component. CANL has a pullup current source, typically 75 μA, activated under some fault condition.

D.7.6 RTL

RTL is the connection to the CANL bus terminal resistors. RTL output structure is a low-side switch, turned on under normal condition and automatically deactivated under some fault conditions. In the application, an external resistor is connected between RTL and CANL pins. In recessive state when the CANL driver is off, CANL line is tied to V_{DD} through external RTL resistor.

In V_{BAT} standby and sleep modes, the RTL pin is connected to the V_{BAT} line through an internal switch and a 12.5-kΩ resistor. This means that in these modes CANL bus line is biased to V_{BAT}. Wakeup from these modes is detected by CANL line going from V_{BAT} level to CANL wakeup threshold level.

D.7.7 STB and EN

STB and EN are input pins used to configure the device into a desired mode. These pins are CMOS (complementary metal-oxide semiconductor) compatible and are connected to the microcontroller of the application.

D.7.8 INH

This is an output of the MC33388 used to control an external voltage regulator having an inhibit input. INH is a high-side switch structure, active when the MC33388 is in normal, receive-only, or standby V_{BAT} modes. INH is switched off in sleep mode to switch off the voltage regulator of the application. INH has no pulldown structure.

D.7.9 WAKE

This is a high voltage input used to wake up the device from sleep and V_{BAT} standby modes. WAKE is usually connected to an external switch in the application. The typical WAKE thresholds are $V_{BAT}/2$.

The WAKE pin has a special design structure and allows wakeup from both high-to-low or low-to-high transitions. When entering into sleep or V_{BAT} standby mode, the MC33388 monitors the state of the WAKE pin and stores it as a reference state. The opposite state of this reference state will be the wakeup event used by the device to enter again into normal mode.

An internal filter is implemented with an 8-µs to 38-µs filtering time delay. The WAKE pin input structure exhibits a high impedance, with extremely low input current. A serial resistor should be inserted to limit the input current mainly during transient pulses.

CAUTION: *The WAKE pin should not be left open. If the wakeup function is not used, WAKE should be connected to GND to avoid false wakeup.*

D.7.10 TX

TX is the transmitter input pin to control bus state. It is CMOS compatible and usually is directly connected to the TX output of a microcontroller. When TX is at high state, CANH and CANL are in recessive state. When TX is low, CANL and CANH are in dominant state. Special fault handling is provided to this input to detect permanent dominant state (TX low) which would result in the CAN bus being permanently locked in dominant state and would not allow communication. In the case where TX is low for more than 2 ms typical, the device automatically switches the bus lines to recessive. TX has an internal pullup resistor to V_{DD}. See **Table D-4**.

D.7.11 RX

RX is the output receiver connection to the microcontroller. It reports the bus state to the RX input pin of the microcontroller. RX is high when the bus is recessive and low in dominant state. In sleep and standby V_{BAT} modes, RX reports wakeup events. See **Table D-4**.

Table D-4. Truth Table of RX and TX CAN Bus States

TX	RX	Bus State	Comment
Low	Low	Dominant	
High	High	Recessive	
High	Low	Dominant	Bus driven by other node

D.7.12 NERR

NERR is the error output pin which reports errors encountered by the device. When NERR is high, no error is reported; when NERR is low, an error has been detected. In standby V_{BAT} mode, NERR reports a wakeup event. In receive-only mode, NERR reports V_{BAT} power-on flag.

D.8 Application

Figure D-6 is a typical application schematic. All MC33388 capabilities are utilized:

- CAN interface
- Normal and low-power modes
- Wakeup source from CAN bus or wakeup switch

The MC33388 V_{DD} is supplied through an external voltage regulator having an inhibit input pin. In addition to TX and RX connections to the microcontroller CAN module, the MC33388 requires three additional connections to a standard microcontroller input/output (I/O) port for the EN, STB, and NERR pins.

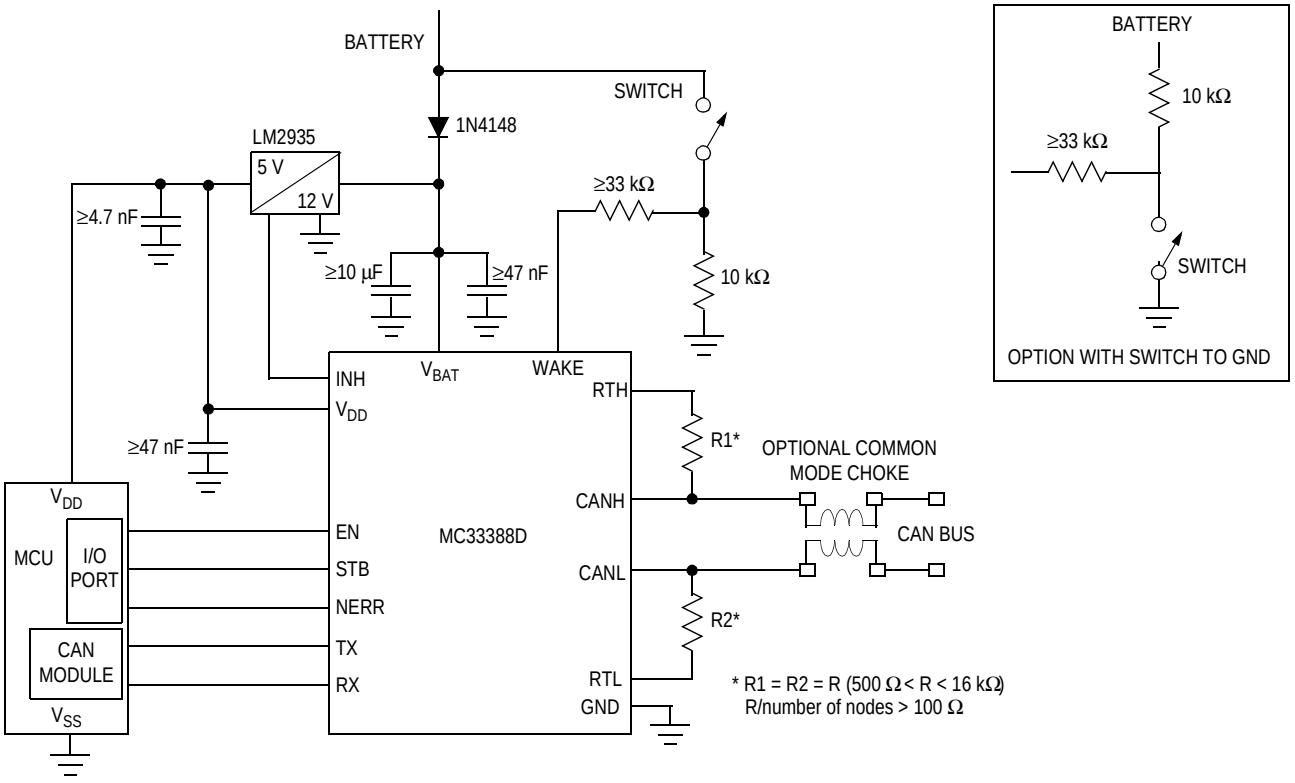


Figure D-6. Typical Application Schematic

The MC33388 WAKE pin is connected to an external signal switch. MC33388 allows the single switch configuration to be connected either to V_{BAT} or to ground through pullup or pulldown resistor. Configuration with switch to GND is indicated as an option in the application schematic (**Figure D-6**). A resistor must be inserted in series with the WAKE pin to limit the input current during positive and negative transient pulses.

Decoupling capacitors are recommended on the MC33388 V_{BAT} and V_{DD} lines. Those capacitors might be shared with other devices from the same printed circuit board, depending on its configuration.

R1 and R2 are the network termination resistors. For proper operation, they must have identical values (R1 equal to R2). Their value is determined by the total network termination resistor and the number of nodes.

The total network termination resistor value must be higher than 100 Ω. If a 500-Ω termination resistor is chosen, with a system composed of 32 nodes, each R1 or R2 will be 16 kΩ. In addition, R1 and R2 values should be chosen between 500 Ω and 16 kΩ at each node.

The CANH and CANL pins can be directly connected to the CAN bus. A serial common mode inductance can be inserted in order to improve electromagnetic compatibility performances both in emission and susceptibility.

The minimum device configuration is shown in **Figure D-7**. The device is used as a CAN transceiver only and other features are not used. The device EN and STB input pins must be connected to 5 V to set the device in normal mode. INH and NERR can be left open.

CAUTION: *WAKE should not be left open and must be conected to a known state, for example GND.*

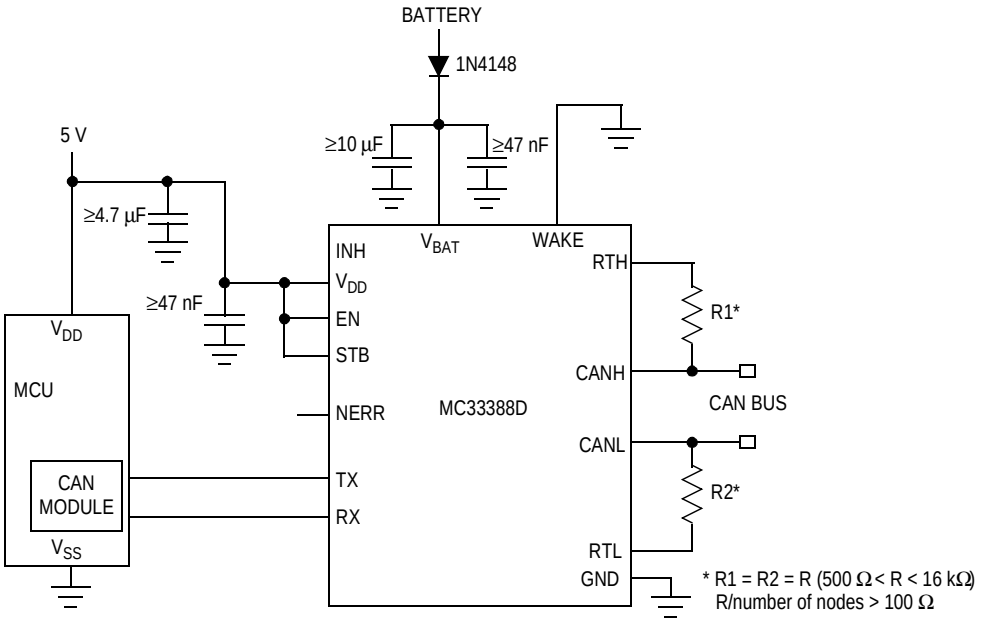


Figure D-7. Minimum Application Schematic

D.9 Electromagnetic Compatibility

The device is designed for optimized noise emission (EMI) and high susceptibility performances (EMC). The source for both disturbance and susceptibility is primarily coming from the bus line wires. They are by far the longest connections compared to the printed circuit board of the application receiving the MC33388, the microcontroller, and the other components.

D.9.1 EMI Noise

To minimize the HF noise generated by the complete application, the MC33388 minimizes the common mode voltage and current glitches occurring at each bus transition: from dominant to recessive and from recessive to dominant. This is achieved by excellent matching in signal transition CANL and CANH. The common mode voltage and current glitches are defined as:

- $CmV = (V_{CANH} + V_{CANL})/2$
- $CmI = (I_{CANH} = (I_{CANL})/2$

The device is optimized for dual-wire operations. For instance, under the fault condition where one CAN bus connection is shorted to fixed voltage (for example, GND) the common mode will be considerably degraded.

Figure D-8 shows the typical signals for common mode voltage measured at the CANL and CANH pins.

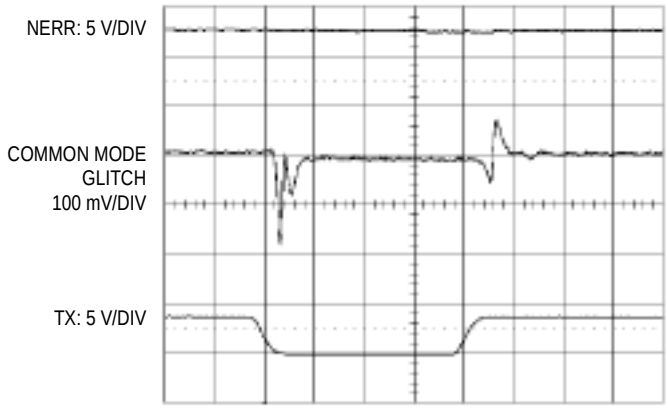


Figure D-8. Typical Common Mode Glitch Measured at CANL and CANH

CAUTION: *The common mode voltage characteristics are dependant on the immediate device environment, such as bus capacitor loading and bus wire length and type, etc. In addition, the symmetry of the CANL and CANH bus lines is a key parameter to optimize common mode glitches. For instance, un-symmetry could result in different parasitic capacitor values between CANL to GND and CANH to GND and will increase common mode glitches and degrade overall system performance.*

D.9.2 EMC Susceptibility Performances

The MC33388 is optimized for high immunity from external field disturbances. The bus lines are by far the primary antenna for external field coupling to the CANL, CANH, RTL, and RTH connections. The device performances are characterized using the bulk current injection (BCI) test method, derived from specification ISO 11452-4.

D.9.3 Susceptibility Evaluation with BCI

The component is configured according to the electrical schematic very close to the typical application schematic shown in **Figure D-6**. The main difference is that the microcontroller is replaced by an external generator and analyzer connected to RX, TX, and NERR through an optical link. A network composed of two nodes equipped with MC33388, one in emitter and one in receiver, is evaluated. The disturbance is applied to both the CANL and CANH twisted pair bus lines with an appropriate coupling clamp.

During test sequences, received bits are compared to transmit bits. When received bits are different from transmit bits, the device is considered to have failed for the particular frequency.

D.9.4 Results

Figure D-9 and **Figure D-10** describe the device susceptibility performances in the frequency range of 1 MHz to 400 MHz with injected current target of 200 mA and 316 mA.

When the target current is reached and when no susceptibility is observed, the next frequency point is analyzed, until reaching the maximum frequency of 400 MHz. If a susceptibility is observed for a particular frequency, the free point is marked.

- **Figure D-9** shows results with a target susceptibility level of 316 mA or 49 dBmA with a 1 kHz, 80 percent modulation added to the injected current.
- **Figure D-10** shows the susceptibility levels with a target susceptibility level of 200 mA or 46 dBmA without modulation added to the injected current.

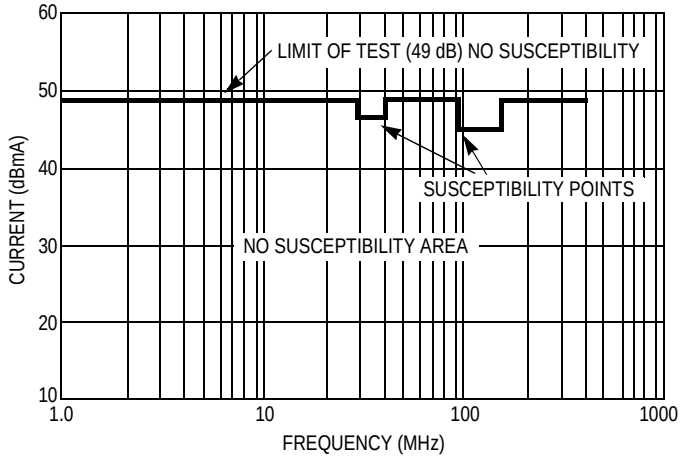


Figure D-9. Minimum Susceptibility Level with Modulation

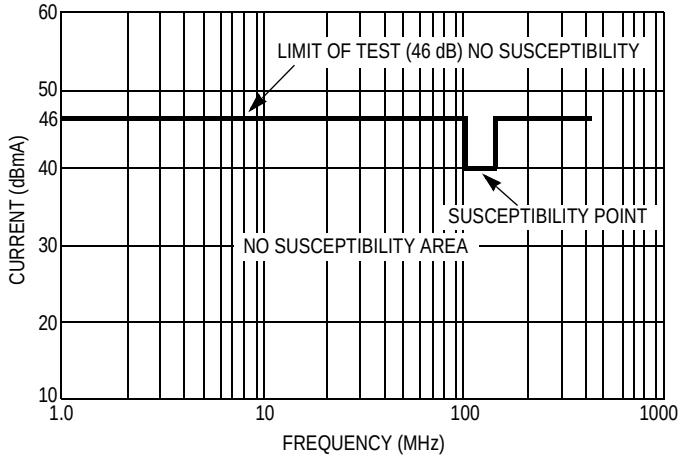
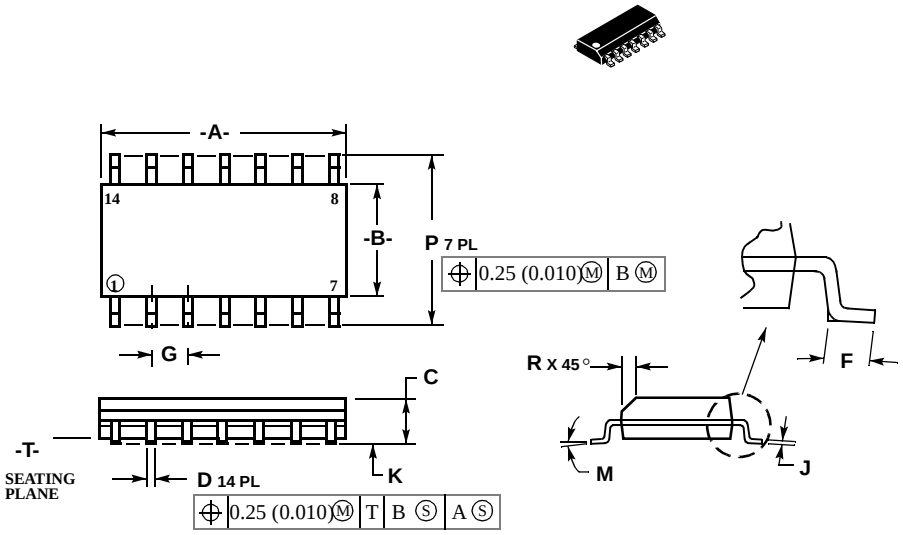


Figure D-10. Minimum Susceptibility Level without Modulation

D.10 Case Outline Dimensions

The MC33388 is packaged in a 14-pin plastic small outline dual in-line package. Dimensions for this package are shown in **Figure D-11**.



NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: MILLIMETER.
3. DIMENSIONS A AND B DO NOT INCLUDE MOLD PROTRUSION.
4. MAXIMUM MOLD PROTRUSION 0.15 (0.006) PER SIDE.
5. DIMENSION D DOES NOT INCLUDE DAMBAR PROTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.127 (0.005) TOTAL IN EXCESS OF THE D DIMENSION AT MAXIMUM MATERIAL CONDITION.

	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	8.55	8.75	0.337	0.344
B	3.80	4.00	0.150	0.157
C	1.35	1.75	0.054	0.068
D	0.35	0.49	0.014	0.019
F	0.40	1.25	0.016	0.049
G	1.27 BSC		0.050 BSC	
J	0.19	0.25	0.008	0.009
K	0.10	0.25	0.004	0.009
M	0°	7°	0°	7°
P	5.80	6.20	0.228	0.244
R	0.25	0.50	0.010	0.019

Figure D-11. Outline Dimensions for Case 751A-03

How to Reach Us:

USA/EUROPE/LOCATIONS NOT LISTED:
Motorola Literature Distribution
P.O. Box 5405
Denver, Colorado 80217
1-303-675-2140
1-800-441-2447

TECHNICAL INFORMATION CENTER:
1-800-521-6274

JAPAN:
Motorola Japan Ltd.
SPS, Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:
Motorola Semiconductors H.K. Ltd.
Silicon Harbour Centre
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
852-26668334

HOME PAGE:
http://www.motorola.com/semiconductors/

MOTOROLA

DRM003/D