MOTOROLA
intelligence everywhere™

digital dna™

Cluster for Motorbikes Using the MC68HC908LJ12 and MC33970

Designer Reference Manual

M68HC08
Microcontrollers

DRM059/D
Rev. 0
3/2004

MOTOROLA.COM/SEMICONDUCTORS

Cluster for Motorbikes Using the MC68HC908LJ12 and MC33970

Designer Reference Manual

by: Jaromir Chocholac
TU682
Czech Systems Laboratories
e-mail: jaromir.chocholac@motorola.com

To provide the most up-to-date information, the revision of our documents on the World Wide Web will be the most current. Your printed copy may be an earlier revision. To verify you have the latest information available, refer to:
http://motorola.com/semiconductors

The following revision history table is provided to summarize any changes contained in future revisions of this document. For your convenience, the page number designators will be linked to the appropriate location.

Revision History

Revision History

Date	Revision Level	Description	Page Number(s)
March, 2004	N/A	Initial release	N/A

Table of Contents

Table of Contents

Section 4. Software Description

Appendix A. Bill of Materials and Schematics

Appendix B. Glossary

Table of Contents

List of Figures and Tables

List of Figures and Tables

Section 1. Introduction

1.1 Application Intended Functionality

This reference design of the Cluster for Motorbikes provides an example of the speedometer, odometer, tachometer, and fuel gauge functionality in the Cluster. The reference design demonstrates the application of the Gauge Driver Integrated Circuit (GDIC) together with an M68HC08 Microcontroller Unit (MCU).

The reference design is based on:

- MC33970 GDIC
- MC68HC908LJ12 MCU

1.2 Benefits of Our Solution

The Cluster for Motorbikes uses a modular concept. The base board with the microcontroller and GDIC is able to perform all of the cluster functionality. The application functionality can be easily changed by the hardware configuration on the PCB.

In addition, the Cluster for Motorbikes can be used as a hardware platform for software development. For this purpose, the board is equipped with an interface for reprogramming, and the MCU is programmed with the Developer's Serial Bootloader for M68HC08 devices. This tool allows the MCU memory to be reprogrammed in-circuit, using the standard serial asynchronous port.

The module is designed to be housed in a standard 3 3/8-inch case.

Introduction

Section 2. Quick Start

2.1 Introduction

This section describes the main procedures required to set up and start the Cluster for Motorbikes Demo Kit. The demo is designed to show the basic functionality of the Cluster for Motorbikes. The document also describes the specific steps and provides additional reference information.

The Cluster for Motorbikes Demo Kit can operate in two modes: Local or Remote. In Remote mode, the application is controlled from a user-friendly graphical environment, running on a PC.

The Cluster for Motorbikes Demo Kit is distributed with the following components:

- Cluster demo module
- CD ROM
- 12-V power supply
- Parallel port cable

The Demo System layout is shown in **Figure 2-1**

2.2 System Requirements

The Cluster module is distributed with embedded application software. No additional software is needed to run the demonstration in Local mode. To run the demonstration in Remote mode, installation of the Cluster Control Panel application is required.

The Cluster Control panel application can run on any computer using Microsoft Windows XP, Windows NT, 98, or 95 operating systems with Internet Explorer V4.0 or higher installed.

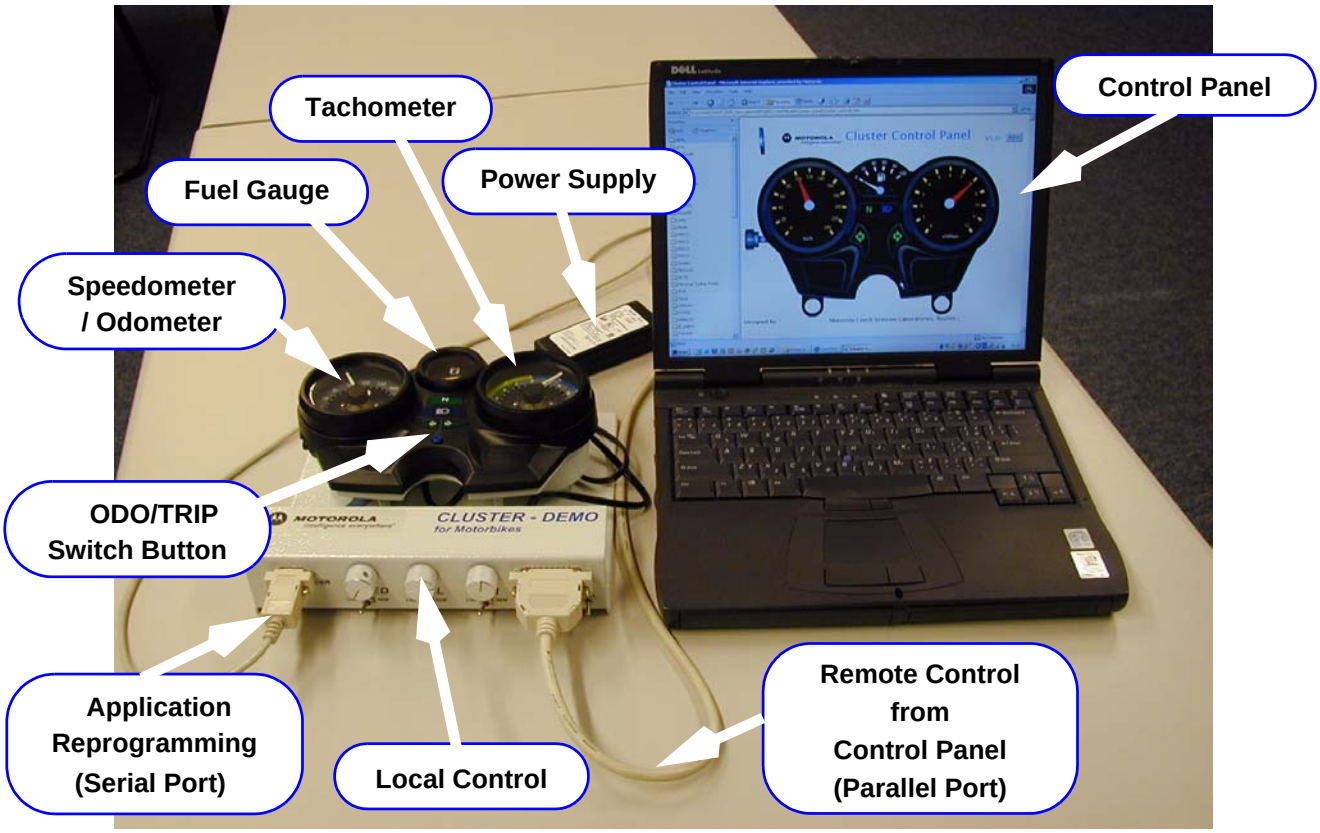

Figure 2-1. Demo System Layout

2.3 Cluster Control Panel Installation

The Cluster Control Panel application is distributed on a CD ROM. To install the Cluster Control Panel application, follow this step-by-step procedure:

1. Insert the CD into your CD-ROM drive.
2. Click on the CD-ROM drive; click on the Cluster_Panel folder.
3. Double click on **cluster_panel.exe** file.
4. Follow the on-screen instructions and answer the prompted questions.
5. Specify the folder to unzip the files to.
6. Press the Unzip button.

2.4 Demo System Setup

The Cluster application software has the following embedded functions:

- Speedometer and odometer
- Tachometer
- Fuel gauge

Each function can run in either Local or Remote control mode. The Mode selection is made through switches as shown in **Figure 2-2**

Figure 2-2. Front Panel

2.4.1 Local Mode

Setting up the demo to the Local mode does not require any special action. Just connect the power supply and set all switches to the “LOC” position. All cluster functions can be controlled by the appropriate knob.

2.4.2 Remote Mode

To run the demonstration in the Remote mode, installation of the Cluster Control Panel application must be done. Refer to **2.3 Cluster Control Panel Installation** for step-by-step instructions on how install the Cluster Control Panel.

Once you have installed the Cluster Control Panel, refer to the following for step-by-step instructions on how to start the Cluster Demo.

NOTE: *Stopping all other applications running on your PC is strongly recommended when you run the Cluster Control Panel. This can avoid instability in the PC generated frequency.*

Step 1
Switch OFF the Cluster Demo module PWR switch. See **Figure 2-2**

CAUTION: *To avoid possible damage to the PC parallel port, the power to the Cluster Demo module must be switched OFF before connecting or disconnecting a straight-through parallel cable from the PC to the Cluster Demo module.*

Step 2
Set up all switches to the “REM” position.

Step 3
Connect a straight-through parallel cable from the PC to the Cluster Demo module “REMOTE CONTROL” connector.

Step 4
Start the Cluster Control Panel.

Step 5
Switch ON the Cluster Demo module PWR switch.

Step 6
Control the application remotely from the Cluster Control Panel (see **Figure 2-3**) by simply clicking on the selected needle and dragging it to a new position.

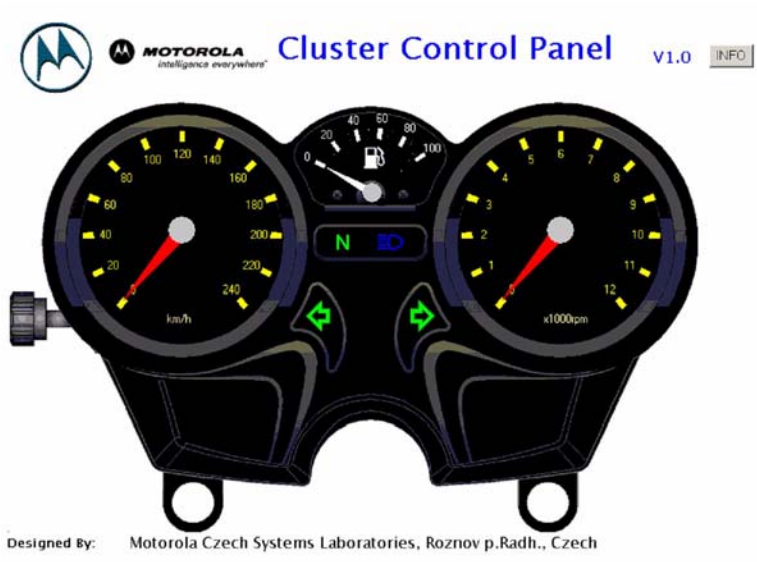

Figure 2-3. Cluster Control Panel

2.5 Remote Control Connector Description

Table 2-1 provides a description of the remote control connectors.

Table 2-1. Remote Control Connectors

Pin #	Name	Description
1	NC	Not connected
2	Speed_IN	TTL signal 0 to 400 Hz
3	Rpm_IN	TTL signal 0 to 400 Hz
4	GND	Connection to ground
5	Fuel_full	TTL log 1 → indication activated(1)
6	Fuel_3/4	TTL log 1 → indication activated(1)
7	Fuel_1/2	TTL log 1 → indication activated(1)
8	Fuel_1/4	TTL log 1 → indication activated(1)
9–17	NC	Not connected
18–25	GND	Connection to ground

1. Only one input can be activated.

CAUTION: *To avoid possible damage to the PC parallel port, the power to the Cluster Demo module must be switched OFF before connecting or disconnecting a straight-through parallel cable from the PC to the Cluster Demo module.*

2.6 Application Reprogramming

For additional changes in features or the parameters of the application, the user has an option to reprogram the Cluster application. The MCU of the Cluster application is programmed with the Developer's Serial Bootloader for the M68HC08 MCU Family, which means the MCU memory can be reprogrammed in-circuit using the standard serial asynchronous port.

To reprogram the embedded application using the bootloader, you need to have master software (**hc08sprg.exe**) installed on the host computer. To use this software, copy the appropriate "exe" file to your folder only, as there is no need for installation.

The following gives a step-by-step procedure for reprogramming the application.

Step 1
Switch OFF the Cluster Demo module Power supply.

CAUTION: *To avoid possible damage to the PC serial port, the power to the Cluster Demo module must be switched OFF before connecting or disconnecting a straight-through serial cable from the PC to the Cluster Demo module.*

Step 2

Connect a straight-through serial cable from the PC to the Cluster Demo module “BOOTLOADER” connector.

Step 3

Start the **cmd.exe** file from Windows.

Step 4

Start the **hc08sprg.exe** file with the required parameters from the Command window. The command line for the **hc08sprg.exe** file has the following syntax:

hc08sprg port[:][S|D|?] [speed] file

port:D .....dual wire mode <default>

port:S .....single wire mode

port:?.....detect single/dual wire mode

speed .....speed in kbps <9600 default>

file .....S19 file

After execution of the **hc08sprg.exe** program, you will get a message “Waiting for HC08 reset ACK ...” (see **Figure 2-4**). In this example, the COM1 port is selected.

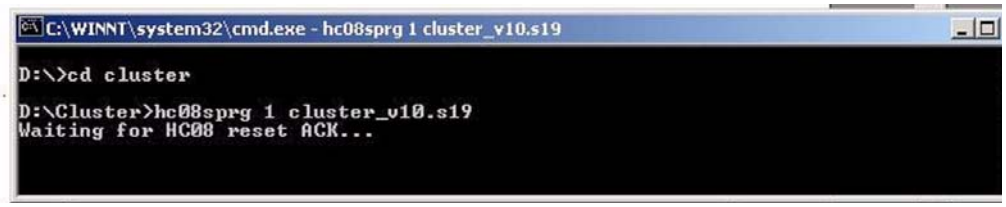

Figure 2-4. Command Window Prompt

Step 5

Switch ON the Cluster Demo module Power supply. The program performs some actions and will prompt a message:

“Are you sure to program part? [y/N]:” (see **Figure 2-5**).

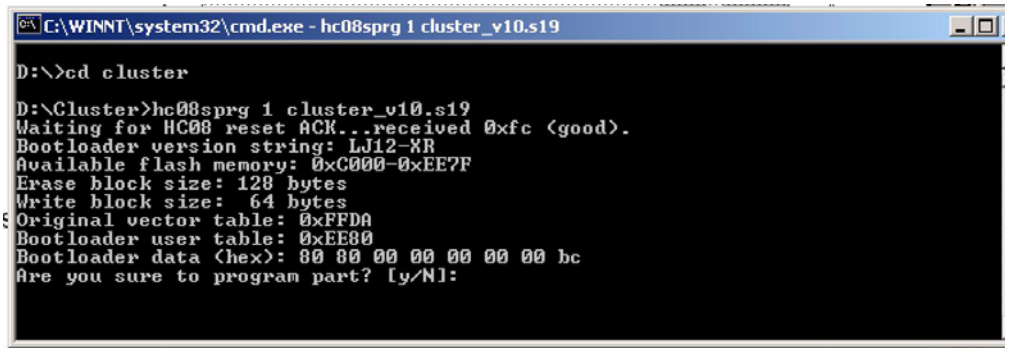

Figure 2-5. Command Window after Power ON

Step 6

To program the device, answer “y” (see **Figure 2-6**).

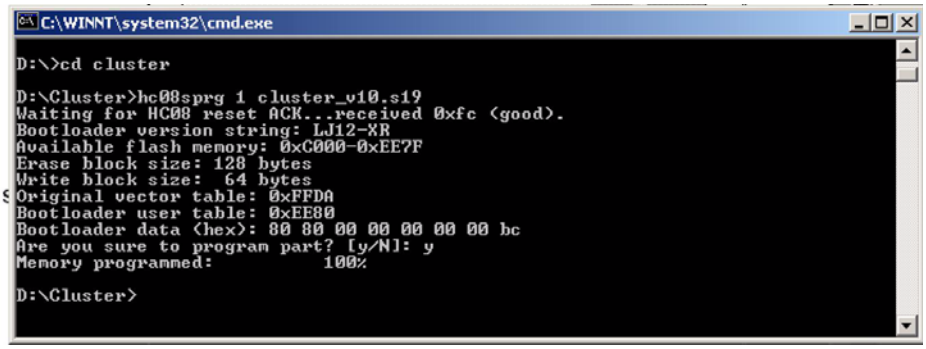

Figure 2-6. Command Window when Programming

After the programming is finished, the embedded program starts to run.

2.7 Bootloader Connector Description

Table 2-2 provides a description of the bootloader connectors.

Table 2-2. Bootloader Connectors

Pin #	Name	Description
1	NC	Not connected
2	Rx	Output signal from Cluster Demo module
3	Tx	Input signal to Cluster Demo module
4	NC	Not connected
5	NC	Not connected
6	NC	Not connected
7	NC	Not connected
8	NC	Not connected
9	NC	Not connected

CAUTION: *To avoid possible damage to the PC serial port, the power to the Cluster Demo module must be switched OFF before connecting or disconnecting a straight-through serial cable from the PC to the Cluster Demo module.*

2.8 ODO/TRIP Button Functions

The Cluster can operate in either odometer or tripmeter mode. The odometer mode is indicated by “ODO” label, and tripmeter mode is indicated by “TRIP” label on the LCD display. The basic function of the ODO/TRIP button is to switch between these two modes of operation.

ODO/TRIP toggle

Each push of the switch button for a time shorter then 1s will toggle between the ODO or TRIP modes.

TRIP data nulling

Each push of the switch button for a time longer then 3s resets the TRIP data in the memory.

Software version indication

Holding the switch button pressed down, when powering up the demo, will display the embedded software version on the LCD display.

Section 3. Hardware Description

3.1 Introduction

This reference design of the Cluster for Motorbikes provides these basic modules: speedometer, odometer, tachometer, and fuel gauge for the motorbike's cluster. In addition to this, the Cluster for Motorbikes can be used as a hardware platform for the software development. It also enables the implementation and testing of user's software. For this purpose, the board is equipped with an interface for reprogramming.

Refer to **Figure 3-1** for a block diagram of the module.

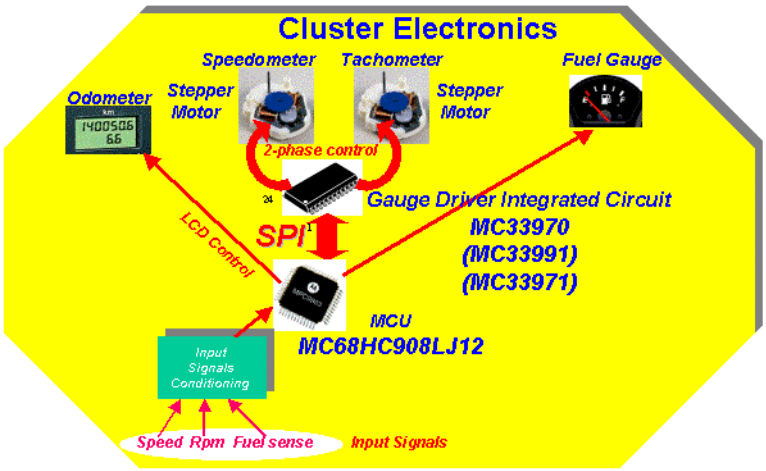

Figure 3-1. Cluster for Motorbikes Block Diagram

The reference design is based on the Gauge Driver Integrated Circuit (GDIC) MC33970, or MC33991, or Single Gauge Driver Integrated Circuit (SGDIC) MC33971, which are analogue products controlling one (MC33971) or two 2-phase instrumentation stepper motors. The application is supported by HC08LJ12 MCU, which is a part of the 8-bit MCU Family.

3.2 How Instruments Work

This section provides a short overview of how the cluster instruments work.

3.2.1 Speedometer and Odometer

The speedometer used on motorbikes indicates the speed of the motorbike and records the distance the motorbike has travelled. Speedometers are calibrated in kilometres and/or in miles per hour. The instrument also records the traveled distance, recorded in kilometres or miles. This part of the instrument is known as the odometer. Most odometers record the total distance travelled. Some also record the distance of individual trips. These can be reset to zero.

The digital speedometer is operated by a speed sensor that outputs electrical pulses to be processed by the microcontroller. The motorbike has magnets attached to one of the wheels, and a pickup attached to the frame. Once per wheel revolution, each magnet passes the pickup generating a voltage in the pickup. The microcontroller counts these voltage spikes, or pulses, and uses them to calculate the speed and distance travelled. The microcontroller drives both speed and odometer indicators. The important information for calculation is the number of pulses per kilometre or mile.

3.2.2 Tachometer

A tachometer is designed to give the speed of a rotating part, in revolutions per minute (rpm). For motorbike use, the tachometer is there to measure the speed of the engine. The digital tachometer is operated by a revolution sensor or engine control unit, that outputs electrical pulses to be processed by the microcontroller. The microcontroller drives the tachometer indicator. The important information for calculation is the number of pulses per revolution.

3.2.3 Fuel Gauge

In the fuel tank, a float moves a simple arm which rubs against a variable resistor. A small current flows through the resistor, just enough to give a voltage across the wire — zero volts at the earth and maximum volts at the other end. As the arm moves across the resistor according to fuel level, it sends a differing voltage to the fuel gauge. The microcontroller measures the sensing voltage and drives the fuel gauge indicator.

3.3 Technical Data

The following list provides technical data for the Cluster for Motorbikes itself, as well as for individual Motorola devices used in the Cluster.

- Speedometer
  - Input Ranges:
    8 input pulses per turn of the wheel → 120 km/h
    4 input pulses per turn of the wheel → 240 km/h
  - Max Input frequency:
    400 Hz
  - Resolution:
    ± 2 Hz on the maximum frequency 400 Hz
- Odometer
  - 6 digits LCD display
  - Function:
    Switchable ODO/TRIP
  - Resolution:
    1km — ODO function
    100m — TRIP function
- Tachometer
  - Input Ranges:
    200 Hz <12000 rpm/1 pulse per turn>
    400 Hz <8000 rpm/3 pulses per turn>
  - Resolution:
    ± 2 Hz on the maximum frequency 400 Hz
- Fuel Gauge
  - Indication:
    LED bar
  - Fuel level sensor:
    Resistor <maximum 100 Ohm>
  - Input Ranges — Indication
  - <41 Ohm — Tank full
  - <44–61> Ohm — Tank 3/4 full
  - <65–74> Ohm — Tank 1/2 full
  - <76–96> Ohm — Tank 1/3 full
  - >96 Ohm — Tank empty
- Power Supply
  - 12 V/190 mA

3.3.1 MC68HC908LJ12 Processor

The control unit of the Cluster for Motorbikes is the MC68HC908LJ12 microcontroller unit (MCU). The MCU uses an 8-bit enhanced central processor unit (CPU08) and a variety of peripheral modules.

Features:

- High-performance M68HC08 architecture
- Maximum internal bus frequency
  - 8 MHz at 5 V operating voltage
- 32-kHz crystal oscilator clock input with 32-MHz internal phase-lock-loop
- Memory:
  - 12K byte FLASH memory
  - 512 bytes of RAM
  - Resident routines for in-circuit programming and EEPROM emulation
- 6-channel Analog-to-Digital Converters (ADC)
  - 10-bit resolution
- Two 16-bit, 2-channel timer interface modules with selectable input capture, output compare, and PWM capability on each channel
- Real time clock with clock, calendar, alarm, and chronograph functions.
- Serial interfaces:
  - Asynchronous Serial Communications Interface (SCI) with infrared encoder/decoder
  - Synchronous Serial Peripheral Interfaces (SPI) module
- 8-bit keyboard wakeup port with programmable pullup
- 32 general-purpose input/output (GPIO) pins
- 4/3 backplanes and static with a maximum of 27 frontplanes liquid crystal display driver
  - CRG (low current oscillator, PLL, reset, clocks, COP watchdog, real time interrupt, clock monitor)
  - MEBI (Multiplexed External Bus Interface)
  - MMC (Module Mapping Control)
  - INT (Interrupt control)
  - BKP (Breakpoints)
  - BDM (Background Debug Mode)
- 64-Pin QFP package, or 64-Pin LQFP package, or 52-Pin LQFP package

3.3.2 MC33970 Gauge Driver

The MC33970 device is a single packaged, Serial Peripheral Interface (SPI) controlled, dual stepper motor Gauge Driver Integrated Circuit (GDIC). This monolithic IC consists of four dual output H-Bridge coil drivers and the associated control logic. Each pair of H-Bridge drivers is used to automatically control the speed, direction, and magnitude of current through the two coils of a 2-phase instrumentation stepper motor, similar to an MMT licensed AFIC 6405.

Features:

- MMT-licensed two-phase stepper motor compatible
- Minimal processor overhead required
- Fully integrated pointer movement and position state machine core movement emulation
- 4096 possible steady state pointer positions
- 340 degree maximum pointer sweep
- Maximum pointer acceleration 4500 deg/s^2
- Maximum pointer velocity of 400 deg/s
- Analog micro stepping (12 steps/degree of pointer movement)
- Pointer calibration and return to zero
- SPI controlled 16-bit word
- Calibratable internal clock
- Low Sleep mode current

3.4 Functionality

The Cluster for Motorbikes is dedicated for use in the Mid-End Motorbike market.

3.5 Architecture

Schematics of the Cluster for Motorbikes are provided in **Appendix A. Bill of Materials and Schematics**. The Cluster for Motorbikes schematic can be seen in **A.2 Cluster for Motorbikes Schematics**.

The Cluster for Motorbikes is a modular system, designed to demonstrate the performance of a Motorola gauge driver device. The basic design is made with full functionality of the cluster. The modularity is provided for a different assembly of the PCB.

The Cluster for Motorbikes can be logically divided into the following four basic blocks:

- Microcontroller
- Gauge driver
- Input signal conditioning
- Power supply

Data transfer between the Microcontroller and Gauge driver is ensured by the SPI protocol.

3.5.1 Microcontroller

The main function of this part of the Cluster for Motorbikes is to control the application. A Motorola 8-bit MC68HC908LJ12 high-performance microcontroller unit (MCU) was selected to control the application. The MCU (U7) uses enhanced central processor unit and embedded peripheral modules.

The application occupies the following peripheral modules:

- Two 16-bit Timer Interface Modules (TIM1, TIM2) in the input capture mode.
- Serial Peripheral Interface module (SPI)
- Liguid Crystal Display driver (LCD)
- General-purpose Input/Output pins (I/O)
- Keyboard wakeup port (KBI)
- 10-bit successive approximation Analog-to-Digital Converter (ADC)
- Serial Communication Interface module (SCI)

The MCU controls the GDIC device through the SPI channel. The SPI module allows full-duplex, synchronous, and serial communication between the MCU and peripheral devices. The SPI shares four I/O pins with four parallel I/O ports. When the SPI system is enabled, the four associated SPI port pins are dedicated to the SPI function as:

- Serial clock (SPSCK)
- Master out/slave in (MOSI)
- Master in/slave out (MISO)
- Slave select ($\overline{SS}$) — see NOTE

NOTE: *During an SPI transmission, data is transmitted (shifted out serially) and received (shifted in serially) simultaneously. The serial clock (SPSCK) synchronizes shifting and sampling of the information on the two serial data lines (MOSI, MISO). A slave select ($\overline{SS}$) line allows selection of an individual*

slave SPI device; slave devices that are not selected do not interfere with SPI bus activities.

The SPI can be configured to operate as a master or as a slave. The master mode must be selected for the SPI module to control the GDIC, because only a master SPI module can initiate transmissions.

The Cluster for Motorbikes uses one channel (ADC3) of the Analog-to-Digital Converter (ADC) to perform fuel level sensing. The resolution of the ADC is 10 bits, but only 8 bits are used by the application.

A simple switchable current source of 20 mA (Q3, D4, D6, R17, R13) supplies the external fuel level sensing resistor (maximum value is 100 Ohm). The voltage corresponding to the resistor value is measured by the ADC.

The board provides a 4-position DIP switch (SW1), for configuring the application parameters.

The general-purpose I/O (PTA1 to PTA5) is used to drive the fuel gauge indicator. A simple LED bar was used as a fuel gauge indicator for the demo purpose.

The general-purpose I/O (PTC5 to PTC7) emulates the I^2C interface to control the external EEPROM. This external EEPROM is used to store Odometer and Tripmeter data.

The embedded LCD driver module can drive a maximum of 27 frontplanes and 4 backplanes of an LCD display. The application uses an LCD display with 11 frontplanes and 4 backplanes. Because of the 4 backplanes, a 1/4 duty of the output waveform is set up. When the LCD driver module is enabled, the backplane waveforms for the selected duty are driven out through the backplane pins. The backplane waveforms are periodic.

The Keyboard Interrupt module (KBI) provides eight independently maskable external interrupts. The KBI pins are shared with standard port I/O pins. The application uses two of them:

- KBI0 for the ODO/TRIP button service
- KBI4 for the power down service

The standard SCI interface, together with the bootloader embedded code, is used to communicate with the PC for reprogramming the application.

NOTE: *The bootloader can be used only for reprogramming, not for in-circuit debugging. The bootloader is a low-cost, in-circuit programming solution.*

3.5.1.1 External EEPROM

An external EEPROM (M24C04) is used to store Odometer and Tripmeter data. The device is compatible with the I^2C memory protocol. The device carries a built-in 4-bit Device Type Identifier code (1010) in accordance with the I^2C bus definition. The device behaves as a slave in the I^2C protocol, with all memory operations synchronized by the serial clock. Read and Write operations are initiated by a Start condition generated by the bus master. The Start condition is followed by a Device Select Code and R/W bit, terminated by an acknowledgement bit. When writing data to the memory, the device inserts an acknowledgement bit during the 9th bit time, following the bus master's 8-bit transmission. When data is read by the bus master, the bus master acknowledges the receipt of the data byte in the same way. Data transfers are terminated by a Stop condition after an Ack for Write, and after a NoAck for Read. See **Figure 3-2**.

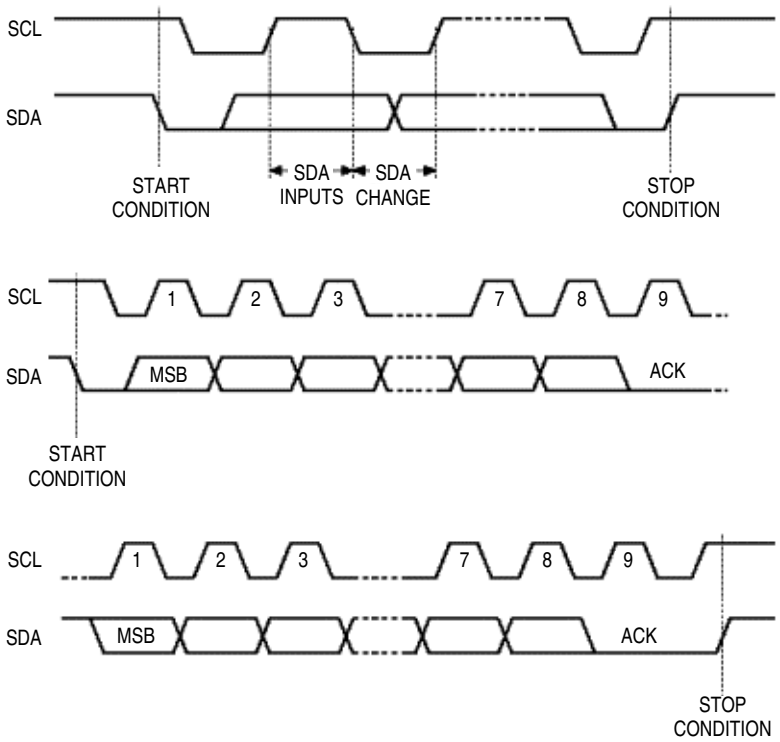

Figure 3-2. I^2C Communication

Write Control ($\overline{WC}$) input signal is useful for protecting the entire memory content from inadvertent write operations. Write operations are disabled to the entire memory array when $\overline{WC}$ is driven high.

3.5.2 Gauge Driver

The GDIC MC33970 is able to control two instrumentation stepper motors. One of the stepper motors is a part of the speedometer board (U4), another is connected through the connector J1. See **Figure 3-3**.

The microcontroller controls the GDIC through the SPI channel. $\overline{RST}$ signal is controlled from the microcontroller and resets the device, or places the device into a sleep state, if driven to a logic 0. The RTZ output signal gives information about a Return to Zero event to the microcontroller.

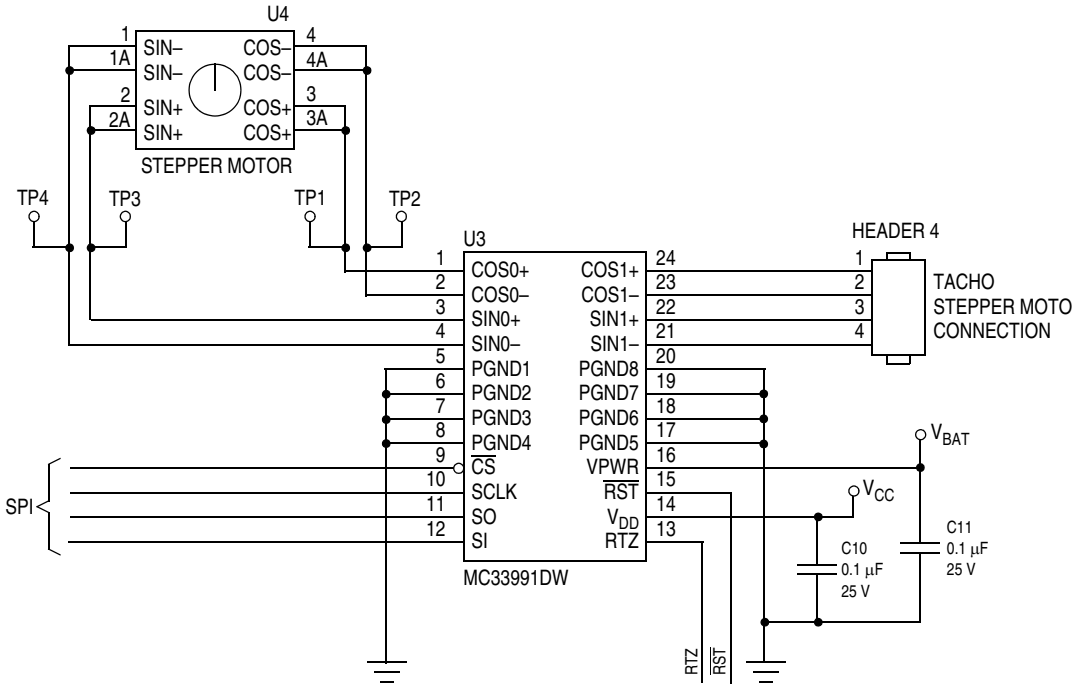

Figure 3-3. Gauge Driver

3.5.3 Input Signals Conditioning

The digital speedometer is operated by a speed sensor that outputs electrical pulses, as well as a revolution sensor, outputs pulses to be processed by the microcontroller. The pulses are in the frequency range 0 to 400 Hz. The microcontroller manipulates these signals by Timer Interface Modules (TIM1, TIM2) in the input capture mode. To improve the signal shapes, some signal conditioning circuitry was designed. For the speed sensor signal conditioning, see **Figure 3-4** For the revolution sensor signal conditioning, see **Figure 3-5**

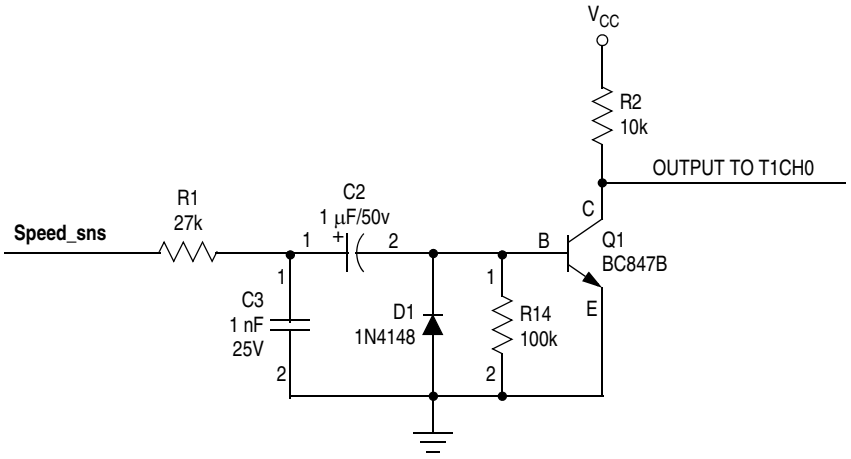

Figure 3-4. Speed Sensor Signal Conditioning

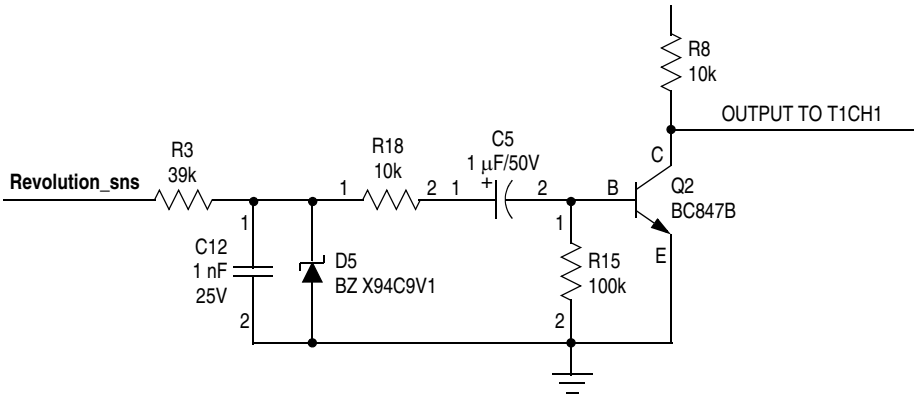

Figure 3-5. Revolution Sensor Signal Conditioning

3.5.4 The Power Supply

The Cluster is supplied from the 12-V motorbike battery. A simple linear voltage regulator MC7805 is used to provide a 5-V power supply for the Cluster devices.

The schematic of the power supply can be seen in **Figure 3-6**

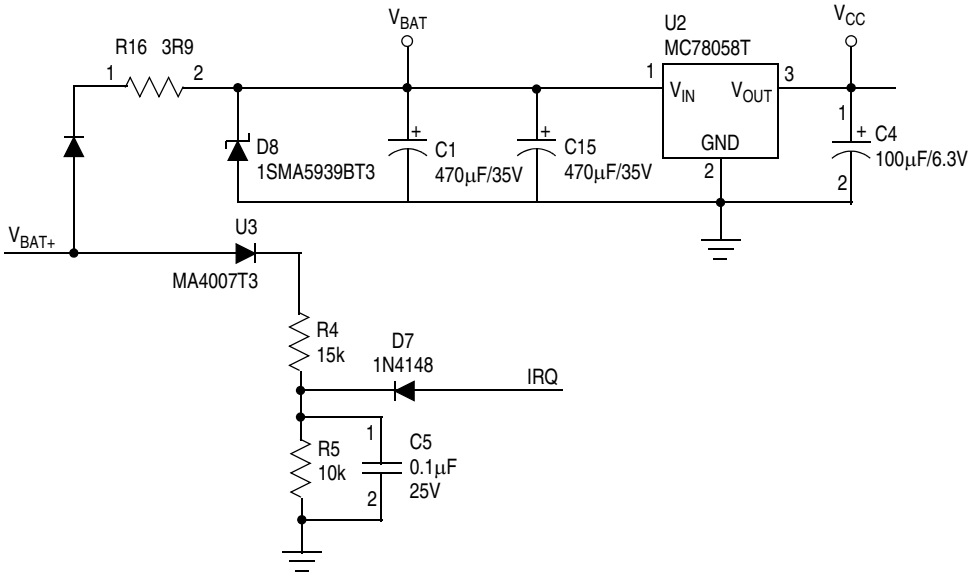

Figure 3-6. Power Supply

It is necessary to save Odometer and Tripmeter data if the power supply from the battery is switched off. The resistor dividers R4 and R5 monitor the battery power supply and generate an interrupt signal in case a power off occurs. The capacitors C1 and C15 hold a voltage long enough for the microcontroller to perform an interrupt service routine to save the data.

3.6 Speedometer Board Layout

Detailed layout plans of the Cluster for Motorbikes boards with the names of all components are shown in **Figure 3-7** and **Figure 3-8**.

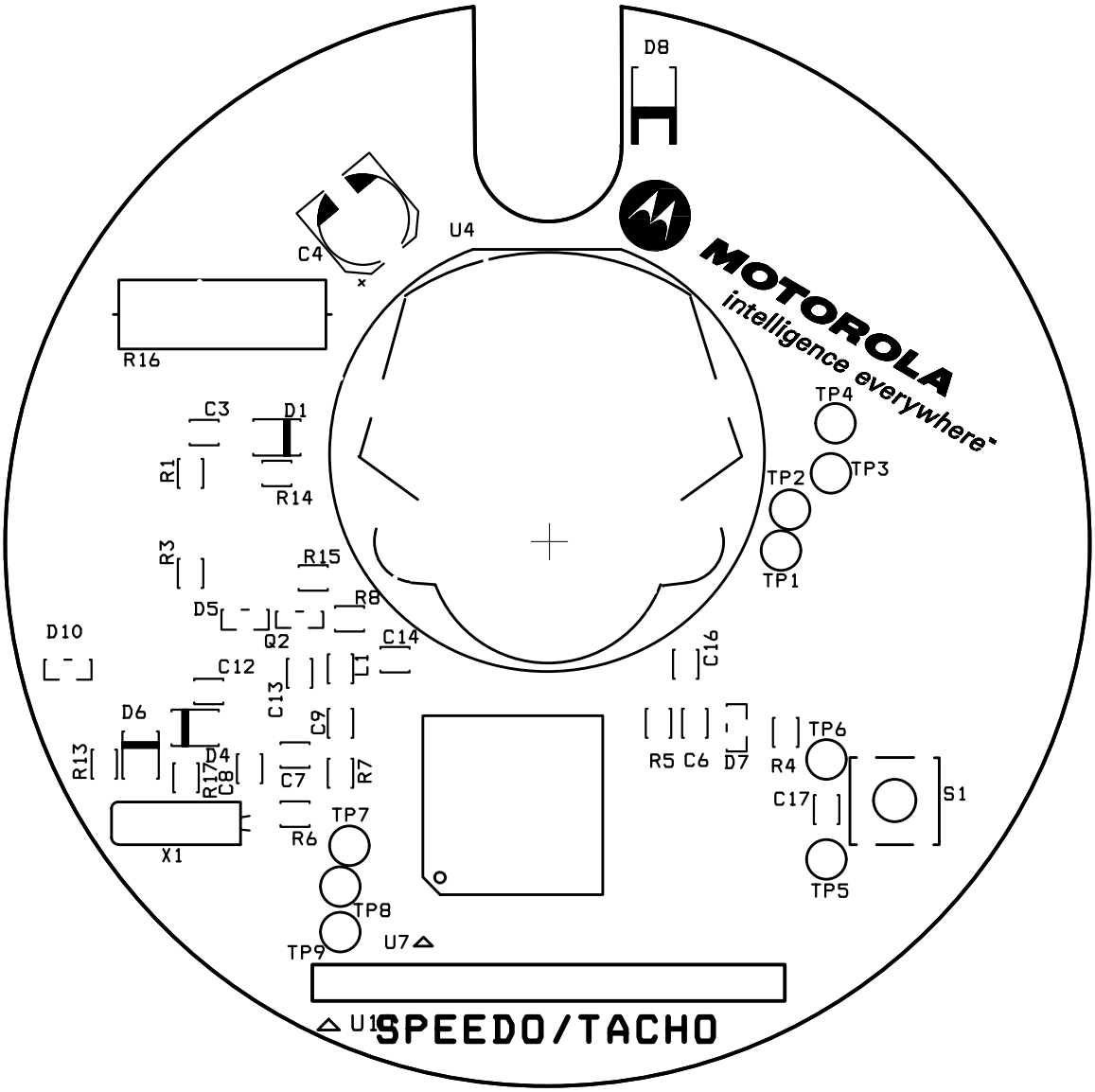

Figure 3-7. Speedometer Board Component Side Layout

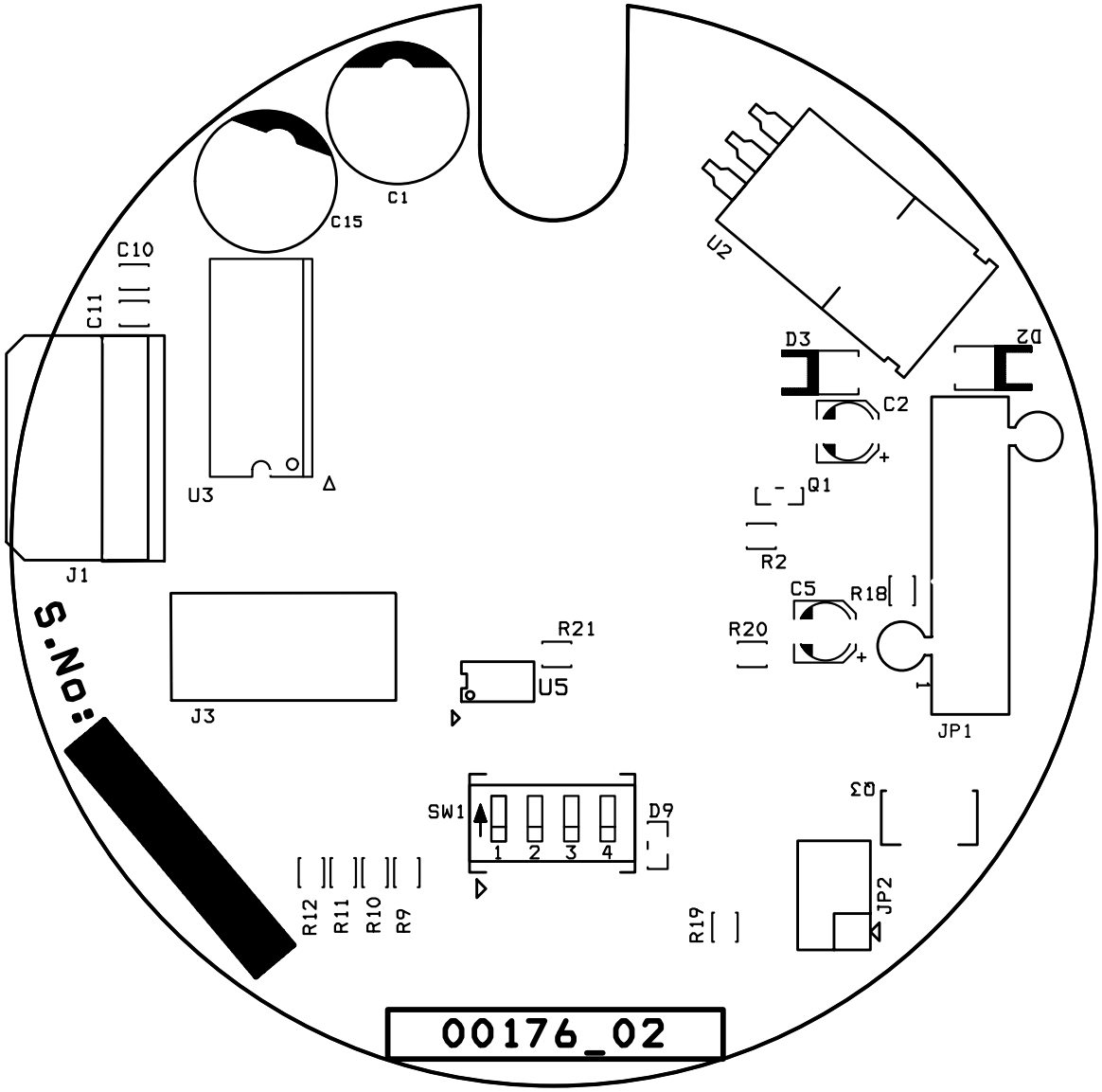

Figure 3-8. Speedometer Board Solder Side Layout

3.7 Speedometer Board Connectors and Dip Switch

Table 3-1 through Table 3-4 describe the speedometer board connector pin assignments and their meanings. Table 3-5 shows the dip switch settings.

Table 3-1. Main Connector (JP1)

Pin #	Name	Description
1	VBAT–	Minus battery voltage
2	RPM_SNS	Revolution sensor signal
3	FUEL_SNS	Fuel level sensor signal
4	SPEED_SNS	Speed sensor signal
5	VBAT+	Plus battery voltage

Table 3-2. Tacho Stepper Motor Connector (J1)

Pin #	Name	Description
1	COS1+	Cosine coil driving
2	COS1–	Cosine coil driving
3	SIN1+	Sine coil driving
4	SIN1–	Sine coil driving

Table 3-3. Fuel Indicator Connector (J3)

Description	Name	Pin #		Name	Description
+5 V power supply	V_{CC}	1	2	GND	Ground
Tank full indication	TANK_FULL	3	4	GND	Ground
Tank empty indication	TANK_EMPTY	5	6	TANK_3/4FULL	Tank 3/4 full indication
Tank 1/3 full indication	TANK_1/3FULL	7	8	TANK_1/2FULL	Tank 1/2 full indication

Table 3-4. Programming Connector (JP2)

Description	Name	Pin #		Name	Description
+5 V power supply	V_{CC}	1	2	RxD	SCI input
Not Connected/KEY	N.C.	3	4	TxD	SCI output
Ground	GND	5	6	GND	Ground

Table 3-5. Dip Switch (SW1) Settings

Switch #	Meaning	
	Off	On
1	TACHO OFF	TACHO ON
2	SPEEDO OFF	SPEEDO ON
3	4 PULSES	8 PULSES
4	200 Hz	400 Hz

3.8 Memory Map

The MC68HC908LJ12 device memory map is shown in **Table 3-6**

Table 3-6. MC68HC908LJ12 Memory Map

From	To	Size	Content
0x0000	0x005F	96 bytes	I/O registers
0x0060	0x025F	512 bytes	RAM
0x0260	0xBFFF	48 k	Unimplemented
0xC000	0xEFFF	12 k	FLASH
0xF000	0xFBFF	3 k	Unimplemented
0xFC00	0xFDFF	512 bytes	Monitor ROM1
0xFE00	0xFE0F	16 bytes	Status and control registers
0xFE10	0xFFCF	448 bytes	Monitor ROM2
0xFFD0	0xFFFF	48 bytes	FLASH vectors

For a detailed description of the MC68HC908LJ12 memory map, refer to the *MC68HC908LJ12 Technical Data* (Motorola document order number MC68HC908LJ12/D).

Hardware Description

Section 4. Software Description

4.1 Introduction

This section of the reference design provides a complete documentation of the Cluster for Motorbikes software.

4.1.1 Software Basics

All embedded software of this project was written using the CodeWarrior for Motorola 8- and 16-bit MCU, CW08 V2.1, by Metrowerks Corporation refer to:
http://www.metrowerks.com

Software content of the project can be divided into two basic groups. In the first group, there are all the initialization routines necessary to configure both the MCU peripherals and all of the system the sub-modules, while the second part consists of routines for the Cluster for Motorbikes application. Therefore, separate descriptions will be given for the initialization routines and for the application. As mentioned in previous chapters, the Cluster for Motorbikes is a modular system with implemented functions of a speedometer, odometer, tachometer, and fuel gauge.

4.1.2 Initialization Routines Basics

Here is a list of routines for the first group, responsible for initialization and configuration of the Cluster for Motorbikes Using the MC68HC908LJ12 and MC33970. Details of each item on the list will be given in the **4.3 Software Implementation**.

- PLL module initialization
- LCD module initialization
- Timer module initialization
- Analog-to-Digital converter (ATD) module initialization
- Keyboard module initialization
- Serial Peripheral Interface (SPI) periphery module initialization for communication with GDIC device
- Initialization and configuration of the GDIC via the SPI channel
- I^2C communication channel emulator initialization

4.1.3 Demo Application Basics

A detailed introduction describing software installation, demo setup, and configuration of the application is given in **Section 2. Quick Start**. Hence **4.3 Software Implementation** will primarily provide information of the software implementation of the Cluster for Motorbikes.

The aim of the demo application is to show the main features of the Cluster for Motorbikes.

4.2 Project Introduction

This section gives an introduction and description of the software implementation of the Cluster for Motorbikes project.

4.2.1 List of the Project Files

The project was written using the Metrowerks CodeWarrior for Motorola 8- and 16-bit MCU CW08 V2.1. In this subsection, a list of all source code files of the CodeWarrior project can be found.

4.2.1.1 Project Source Codes

Definitions of the project source codes are:

- **cluster.mcp** is a Metrowerks CodeWarrior project file
- **speedo.c** is a central file of the project, containing the complete initialization, global variables declaration, and the *main()* routine
- **speedo.h** is the header file of the **speedo.c**; it contains the entire set of application-related symbolic constants, as well as the structure definitions
- **spi.c** and **spi.h** consist of Serial Peripheral Interface (SPI) based routines
- **timer.c** and **timer.h** contain all Timer related routines
- **lcd.c** and **lcd.h** include all LCD related routines used in the project
- **GDIC.c** and **GDIC970.h** files contain all GDIC MC33970 device related routines used in the project
- **keyboard.c** and **keyboard.h** include all keyboard related routines used in the project
- **I2C.h** is a header file of **I2C.asm** containing all routines related to the emulation of I2C communication
- **hc08lj12.h** is a file for periphery module allocation within MCU memory
- **common.h** is a header file for common definitions
- **HC08Lj12.prm** is a device parameter file for FLASH configuration

4.2.1.2 Utilized MCU Peripherals

All MCU peripheral components used in the project are briefly described here. It gives an overall summary picture of the necessary MCU resources.

Usage of the Timer Interface Modules is given in **Table 4-1** and **Table 4-2**.

Table 4-1. TIM1 Module

MCU Pin	Symbolic Name of the Signal	Purpose	ISR Function
T1CH0	N.A.	Speedo signal time interval data conversion	Int_Timer1CH0
T1CH1	N.A.	Tacho signal time interval data conversion	Int_Timer1CH1
N.C.	N.A.	long time support for speedo and tacho signals conversion	Int_Timer1OVF

Table 4-2. TIM2 Module

MCU Pin	Symbolic Name of the Signal	Purpose	ISR Function
N.C.	N.A.	Internal timer — speedo watch dog	Int_Timer2CH0
N.C.	N.A.	Internal timer — tacho watch dog	Int_Timer2CH1

A brief description of the Keyboard Interface (KBI) module usage is given in **Table 4-3**.

Table 4-3. KBI Module

MCU Pin	Symbolic Name of the Signal	Purpose	ISR Function
KBI0	N.A.	ODO/TRIP button service	Int_Keyboard
KBI4	POWER_SENSE	Power down sensor service	Int_Keyboard

A description of the General Purpose I/O (GPIO) usage is given in Table 4-4.

Table 4-4. GPIO Module

MCU Pin	Symbolic Name of the Signal	Purpose	ISR Function
PTC0	TACHO_ON_SW	Input for TACHO_ENABLE Dip switch	—
PTC1	SPEEDO_ON_SW	Input for SPEEDO_ENABLE Dip switch	—
PTC2	SPEEDO_PULS_SW	Input for PULSES Dip switch	—
PTC3	TACHO_FREQ_SW	Input for FREQ Dip switch	—
PTA1	FUEL_1_2FULL_IND	Output for Fuel 1/2 full tank indicator	—
PTA2	FUEL_1_4FULL_IND	Output for Fuel 1/4 full tank indicator	—
PTA3	FUEL_3_4FULL_IND	Output for Fuel 3/4 full tank indicator	—
PTA4	FUEL_EMPTY_IND	Output for Fuel empty tank indicator	—
PTA5	FUEL_FULL_IND	Output for Fuel full tank indicator	—
PTA6	FUEL_IND_DIS	Output to control fuel sensor feed current source	—
PTB4	N.A.	User defined I/O pin	—
PTB5	N.A.	User defined I/O pin	—
PTB6	N.A.	Input to sense Return to zero event of GDIC	—
PTB7	GDIC_RESET	Output to control GDIC Reset pin	—

Usage of the Analog to Digital (ATD) converter module is given in Table 4-5. Only ADC3 channel is used.

Table 4-5. ATD Module

MCU Pin	Symbolic Name of the Signal	Purpose	ISR Function
ADC3	FUEL	Data conversion of FUEL signal	—

A brief description of the SPI module usage is given in Table 4-6.

Table 4-6. SPI Module Usage

SPI	Purpose	ISR Function
SPI	Communication with MC33970 devices	—

A description of the SCI module usage is given in Table 4-7.

Table 4-7. SCI Module Usage

SCI	Purpose	ISR Function
SCI	Bootloader Interface communication	—

NOTE: *SCI peripheral module is only used with bootloader.*

4.2.2 Utilized Interrupts

All interrupts used within the Cluster for Motorbikes project are briefly listed in Table 4-8.

Table 4-8. Interrupts

Symbolic Name of Periphery	ISR Function	Type of Interrupt	Note
TIM1	Int_Timer1CH0	Input capture	Speedo signal time interval data conversion
TIM1	Int_Timer1CH1	Input capture	Tacho signal time interval data conversion
TIM1	Int_Timer1OVF	Timer1 overflow	Long time support for speedo and tacho signals conversion
TIM2	Int_Timer2CH0	Output compare	Internal timer — speedo watch dog
TIM2	Int_Timer2CH1	Output compare	Internal timer — tacho watch dog
KBI	Int_Keyboard	Keyboard interrupt	ODO/TRIP button service and power down service

4.2.3 Project Variables and Flags

In this section a brief description of main project variables and flags is given. These variables are related to the basic cluster functions.

4.2.3.1 Speedometer Function

In addition to the following, there are also a couple of symbolic constants (defined in **speedo.h**), controlling the behavior and configuration of the application.

```
volatile tU32 speed_delta_time;         /* Input capture delta time related to the speed
                Speed_delta_time = Speed_second_edge_time - Speed_first_edge_time */
volatile tU16 speed_first_edge_time;    /* Time value of the first edge input capture related
                                           to the speed */
volatile tU16 speed_second_edge_time;   /* Time value of the second edge input capture related
                                           to the speed */
```

4.2.3.2 Tachometer Function

```
volatile tU32 rpm_delta_time;           /* Input capture delta time related to the RPM
                Rpm_delta_time = Rpm_second_edge_time - Rpm_first_edge_time */
volatile tU16 rpm_first_edge_time;      /* Time value of the first edge input capture related
                                           to the RPM */
volatile tU16 rpm_second_edge_time;     /* Time value of the second edge input capture related
                                           to the RPM */
```

4.2.3.3 Odometer Function

```
volatile tU32 odometer_value;           /* Odometer variable */
volatile tU16 trip_value;               /* Trip variable */
volatile tU16 odobase;                  /* Odometer base variable */
```

4.2.3.4 Software Timer Function

```
volatile tU08 ticks;                    /* Number of ticks (overflows of the timer1) */
```

4.2.3.5 Cluster Flags

```
volatile tU08 speed_second_edge_flag; /* Flag indicating that second edge input capture event
                                         related to speed will follow */
volatile tU08 speed_capture_done;     /* Flag indicating that input capture of the speed was
                                         done */
volatile tU08 rpm_second_edge_flag;   /* Flag indicating that second edge input capture event
                                         related to RPM will follow */
volatile tU08 rpm_capture_done;       /* Flag indicating that input capture of the RPM
                                         was done */
volatile tU08 trip_flag;              /* TRIP value indication flag */
volatile tU08 fuel_empty_flag;        /* Fuel empty indication flag */

volatile tU08 sw_timer_flag;          /* Indication that software timer was set */
```

4.2.3.6 Speedometer Constants

```
#define SAMPLE_PERIOD        8       /* Sample period in [us] for the input capture impulses */
#define MAX_STEPS       3200L      /* Max # of steps of the stepper motor for max angle
                                      indication */
#define MAX_SPEED         240L      /* Max speed in [km/ho] to be indicated by speedometer */
#define WHEEL_PERIMETER   133      /* Wheel perimeter in [cm] */
#define PULSES_TURN         4      /* # of pulses per wheel revolution */
#define SPEEDO_CONST
(((3600*MAX_STEPS)/(PULSES_TURN*SAMPLE_PERIOD*(MAX_SPEED-10)))*10)*WHEEL_PERIMETER
                                   /* Speedometer constant */
#define ODOMETER_CONST       (10000L*PULSES_TURN)/WHEEL_PERIMETER
                                   /* # of pulses per 100m */

#define SPEEDO_SCALE_CORR   122    /* Speedometer scale correction constant 1 */

#define SPEEDO_WATCH_PERIOD 65535  /* Max. period in [#*8us] for the speedo input watching */
```

4.2.3.7 Tachometer Constants

```
#define MAX_STEPS_RPM      3114L   /* Max # of steps of the stepper motor for max angle
                                      indication */
#define MAX_RPM              12L   /* Max RPM in 1000*[rev/min] to be indicated by
                                      tachometer */
#define PULSES_REV_RPM     1       /* # of pulses per motor revolution */
#define RPM_CONST
(((6000*MAX_STEPS_RPM)/(PULSES_REV_RPM*SAMPLE_PERIOD*((2*MAX_RPM)-1)))*20)
                                    /* RPM constant */

#define RPM_SCALE_CORR1  130       /* Tachometer scale correction constant 1 */

#define RPM_WATCH_PERIOD 65535     /* Max. period in [#*8us] for the Rpm input watching */
```

4.2.4 Memory Usage

Table 4-9 shows the Cluster for Motorbikes software memory usage.

Table 4-9. Memory Usage

Type of Memory	Total Size (B)	Used Memory (B)
Program FLASH	$C000_h$	$D05_h$
Z_RAM	60_h	$2D_h$
RAM	100_h	7_h
RAM — STACK	108_h	50_h

4.3 Software Implementation

In this section a complete description of the key software modules for the reference design is given.

4.3.1 Application

This subsection summarizes the Cluster for Motorbikes application for the reference design.

4.3.1.1 Application Introduction

The application provides all the following tasks:

- Hardware initialization
- Hardware functionality presentation
- Speedometer task
- Odometer task
- Tachometer task
- Fuel Gauge task

The application main routine (**speedo.c**) can be divided from the functionality point of view in two parts: introductory part and main loop. The introductory part is executed just once after the program start and the main loop is an endless one. The introductory part performs hardware initialization and hardware functionality presentation tasks. The main loop performs the other tasks. The flow chart of the application can be seen in Figure 4-1.

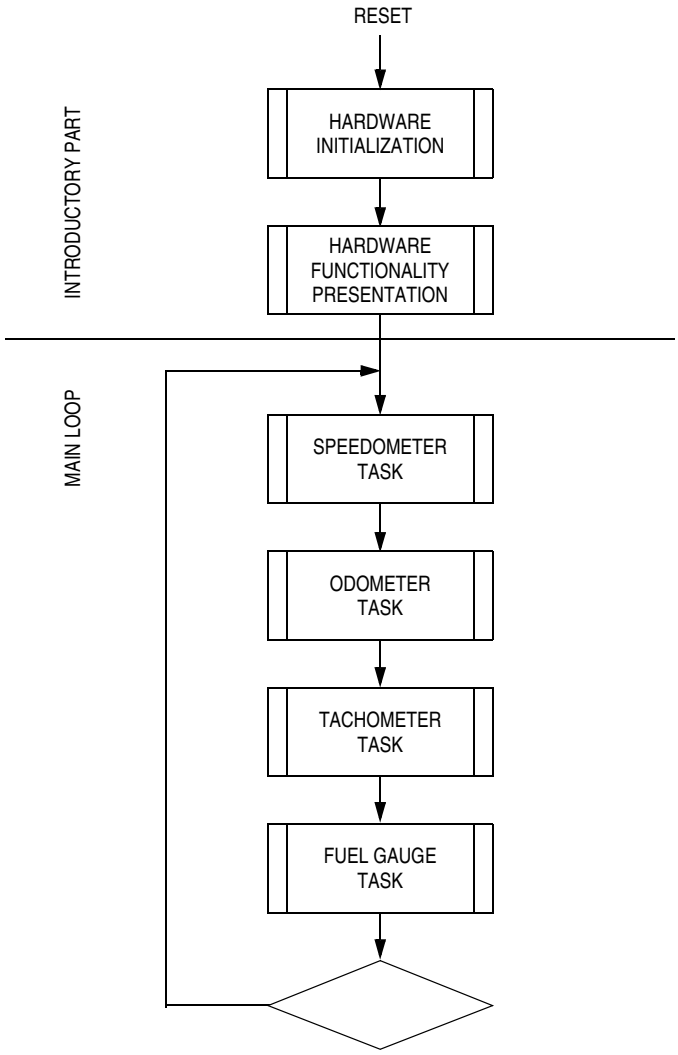

Figure 4-1. Application Flow Chart

4.3.1.2 Hardware Initialization

The hardware initialization task is an introductory part of the main application routine. It initializes all hardware parts used by the application.

4.3.1.3 Hardware Functionality Presentation

This task demonstrates the functionality of the application in the introductory part. The speedometer and tachometer pointers are moved to their maximum scale positions and then back to zero.

4.3.1.4 Speedometer Task

The speedometer indicates the speed of the motorbike in kilometres and/or in miles per hour. The digital speedometer used in application is operated by a speed sensor that outputs electrical pulses to be processed by the microcontroller. The microcontroller measures the frequency or period of the pulses, and uses them to calculate the speed. The speed is converted into the number of steps to be sent to the stepper motor driving the speed pointer. The functional diagram of the speedometer task can be seen in **Figure 4-2**.

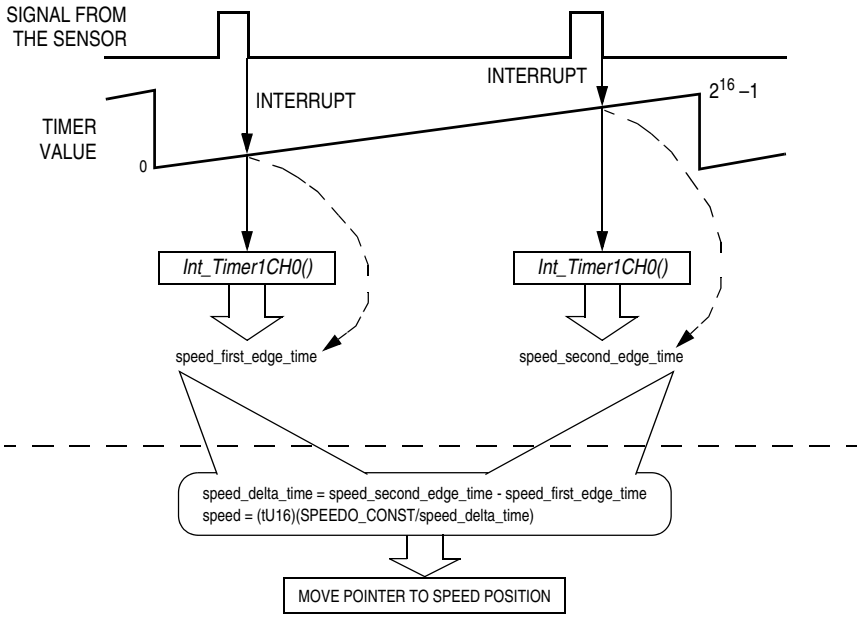

Figure 4-2. Speedometer Functional Diagram

Apart from speed, there are other factors that have an impact on the period of the pulses. They are:

- number of magnets attached to the wheel (number of pulses per wheel revolution) — (PULSES_TURN)
- the circumference of the wheel (WHEEL_PERIMETER)

Both factors are taken into consideration in the “SPEEDO_CONST” constant defined in the **speedo.h** file. See the code listing below.

```
/*****************************************************************************/
/*          S P E E D O M E T E R   D E F I N E S                            */
/*****************************************************************************/
/* public defines for user's reconfiguration */
#define SAMPLE_PERIOD   8     /* Sample period in [us] for the input capture impulses */
//#define MAX_DELTA     65535 /* Max delta of input capture pulses */
#define MAX_STEPS        3200L /* Max # of steps of the stepper motor for max angle indication */
#define MAX_SPEED         240L /* Max speed in [km/hod] to be indicated by speedometer */
#define WHEEL_PERIMETER  133  /* Wheel perimeter in [cm] */
#define PULSES_TURN      4    /* # of pulses per wheel revolution */
#define SPEEDO_CONST
(((3600*MAX_STEPS)/(PULSES_TURN*SAMPLE_PERIOD*(MAX_SPEED-10)))*10)*WHEEL_PERIMETER       /*
Speedometer constant */
#define ODOMETER_CONST      (10000L*PULSES_TURN)/WHEEL_PERIMETER     /* # of pulses per 100m */
```

The “SPEEDO_CONST” is a complex constant that also takes into consideration:

- Sample period for the input capture impulses (SAMPLE_PERIOD)
- Maximum speed to be indicated by speedometer (MAX_SPEED)
- Maximum number of the pointer steps driven by the stepper motor, for max angle indication (MAX_STEPS)

The speed, in the number of pointer steps driven by the stepper motor, is calculated by the following equation:

Speed = (SPEEDO_CONST/speed_delta_time)

The calculated speed pointer position is sent to the GDIC with the following macro command:

POS0R_CMD(speed)

For more details about the macro command refer to **4.3.3 MC33970 Device Control**.

4.3.1.5 Odometer Task

The odometer records the distance that the motorbike has travelled; it is recorded in kilometres or miles. The odometer in the application records the total distance travelled, and also the distance of individual trips. These can be reset to zero.

The number of pulses per kilometre or mile is important information. “ODOMETER_CONST” is used by the application, representing the number of pulses per 100 metres. The number of pulses per wheel revolution (PULSES_TURN), and the circumference of the wheel (WHEEL_PERIMETER) have an influence on this constant. The constant is defined in the **speedo.h** file.

The odometer task is served by the *Odometer_service ()* routine. See below.

```
/*****************************************************************************
 * Function:        Odometer_service (void)
 *
 * Description:     Odometer service function
 *
 * Returns:         none
 *
 * Notes:           This function calculates data on the distance traveled and displays it.
 *                  It uses global variables:
 *                  - Odobase
 *
 *                  execution time 760us if TRIP display
 *                  execution time 828us if ODO display
 *                  executed every 1,5 sec
 *****************************************************************************/

void Odometer_service (void)
  {
      odobase--;                                    /* Decrement Odometer base variable */
      if (odobase == 0)                             /* If loopy distance is 100 meters */
          {
              odometer_value++;                     /* Increment ODOMETER variable */
              trip_value++;                         /* Increment TRIP variable */
              if (trip_flag == 0)
                  {
                  Lcd_update((odometer_value/10)); /* Update LCD display */
                  ODO =1;                           /* Display ODO label on LCD */
                  }
              else
                  {
                  Lcd_update(trip_value);           /* Update LCD display */
                  TRIP =1;                          /* Display TRIP label on LCD */
                  DOT = 1;                          /* Display DOT before last digit */
                  }
              odobase = ODOMETER_CONST;            /* Odometer base variable initialization */
          }
  }
```

4.3.1.6 Tachometer Task

The tachometer gives the engine speed in revolutions per minute (rpm). The digital tachometer is operated by a revolution sensor or engine control unit that outputs electrical pulses to be processed by the microcontroller. The microcontroller measures the frequency or period of the pulses and uses them to calculate the rpm. The rpm is converted into the number of steps to be sent to the stepper motor driving the rpm pointer. The functional diagram of the tachometer task can be seen in **Figure 4-3**.

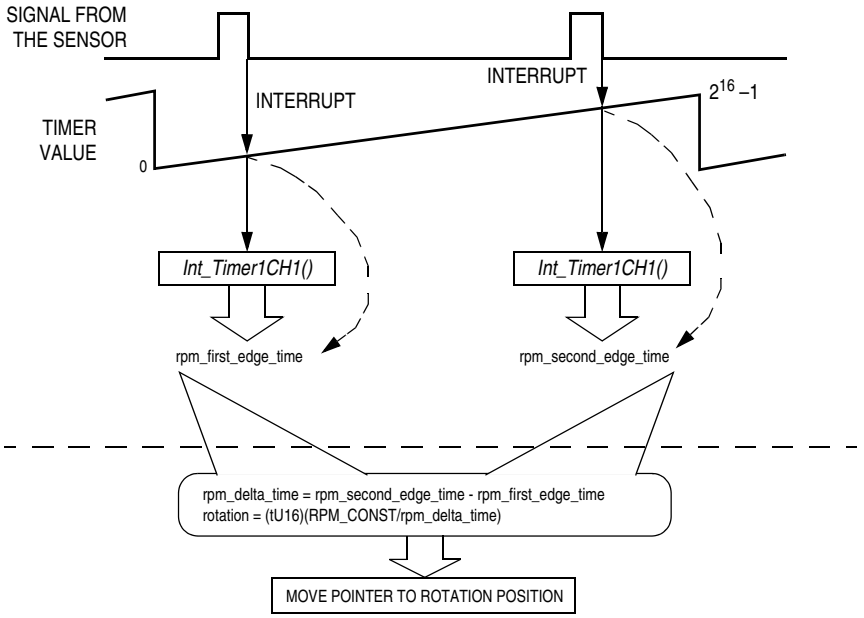

Figure 4-3. Tachometer Functional Diagram

The period of the pulses is affected by the number of pulses per motor revolution (PULSES_REV_RPM). This factor is taken into consideration in the “RPM_CONST” constant defined in the **speedo.h** file. See the code listing below.

```
/*****************************************************************************/
/*                    R P M   D E F I N E S                                  */
/*****************************************************************************/

#define MAX_STEPS_RPM    3114L      /* Max # of steps of the stepper motor for max angle
indication */
#define MAX_RPM          12L       /* Max RPM in 1000*[rev/min] to be indicated by tachometer */
#define PULSES_REV_RPM   1          /* # of pulses per motor revolution */
#define RPM_CONST
(((6000*MAX_STEPS_RPM)/(PULSES_REV_RPM*SAMPLE_PERIOD*((2*MAX_RPM)-1)))*20)  /* RPM constant */
```

The “RPM_CONST” is a complex constant that also takes into consideration:

- Sample period for the input capture impulses (SAMPLE_PERIOD)
- Maximum rpm to be indicated by tachometer (MAX_RPM)
- Maximum number of the pointer steps driven by the stepper motor, for max angle indication (MAX_STEPS_RPM)

The rpm, in the number of pointer steps driven by the stepper motor, is calculated by the following equation:

rotation= (RPM_CONST/rpm_delta_time)

The calculated rpm pointer position is sent to the GDIC with the following macro command:

POS1R_CMD(rotation)

For more details about the macro command refer **4.3.3 MC33970 Device Control**.

4.3.1.7 Fuel Gauge Task

In the fuel tank, a float moves a simple arm which rubs against a variable resistor according to fuel level. The microcontroller measures the sensing voltage across the resistor and drives the fuel gauge indicator. The *Fuel_ind_service ()* routine serves the fuel gauge task.

4.3.2 SPI Communication

The configuration of the Gauge Driver Integrated Circuit — MC33970 device, is done through the SPI channel. A description of the SPI module initialization is presented in **4.3.2.1 SPI Periphery Module Initialization**.

4.3.2.1 SPI Periphery Module Initialization

The initialization is implemented in the *SPI_init()* routine (**spi.c**).

There are a couple of key settings in the SPI format. First, it is important to set the *Master mode of the SPI device* equal to one, because only a master SPI device can initiate a transmission with peripherals. The SPI baud rate setting (value of the SPI serial clock, called SCLK) has to be set to a value suitable for the connected device. SCLK frequency equal to 1 MHz was chosen.

For the SPI communication, the polling approach was chosen so that the SPI interrupt is disabled.

A complete listing of the *SPI_init()* can be found on the next page.

NOTE: *Some parts of the listing are discussed further within this section.*

```
/*************************************************************************
 * Function:        SPI_Init(void)
 *
 * Description:     SPI initialization
 *
 *
 * Returns:         none
 *
 * Notes:
 *
 *************************************************************************/

void SPI_Init(void)
{

    SPCR_SPMSTR  = 1;               /* Master mode selected */
    SPCR_CPHA  = 1;                 /* SPI clock phase bit */
                                    /* first SCLK edge issued at the beginning
                                       of the 8-cycle transfer operation */
    SPCR_CPOL  = 0;                 /* clock polarity bit */
                                    /* serial clock (SCK) active in high, SCK idles low */
    SPCR_SPTIE = 0;                 /* transmit interrupt disabled, SPI
                                       communication done in polling style */
    SPCR_SPE   = 1;                 /* enable SPI, SPI port pins are dedicated
                                       to SPI module */
    SPCR_SPRIE  = 0;                /* SPI Receiver interrupt disabled */

    SPSCR_MODFEN  = 0;              /* disable the MODF error */


    /* divider is set to 8, so SCLK is 1MHz for 8MHz Module Clk */
    SPSCR_SPR0  = 0;                /* baud rate selection */
    SPSCR_SPR1  = 0;                /* baud rate selection */
}
```

There are a couple of different SPI channel settings applicable for devices. The first of them is the SPI format; for the GDIC-MC33970, the most significant bit is transferred first. The second divergence is in the *SPI Clock phase bit* settings; for the MC33970 device, this value has to be equal to one (the first SCLK edge is issued at the beginning of the 8-cycle transfer operation). The device uses a 16-bit long SPI format of communication.

4.3.2.2 SPI Communication Routine

For the SPI communication, the following function is implemented:

- *tU16 SPI_SendRecv16(tU16 data)* is used for the 16-bit long SPI communication suitable for the MC33970 device

```
/*****************************************************************************
 * Function:        SPI_SendRecv16(tU16 data)
 *
 * Description:     SPI send and receive data
 *
 *
 * Returns:         SPI received data
 *
 * Notes:
 *
 *****************************************************************************/

tU16 SPI_SendRecv16(tU16 data)
{
    tU16 spi_ret;

    SPI_CSB = 0;                          /*  SPI chip select active now*/
        while (SPSCR_SPTE == 0);         /* when Tx ready (should be immediately) */
    SPDR = data >> 8;                     /* send first byte */
    while (SPSCR_SPRF == 0);             /* wait for Rxflag (first byte) */
    spi_ret = SPDR << 8;                  /* read data */
        while (SPSCR_SPTE == 0);         /* when Tx ready (should be immediately) */
    SPDR = (tU08)data;                    /* send second byte */
    while (SPSCR_SPRF == 0);             /* wait for Rxflag (second byte) */
    spi_ret += SPDR;                      /* read data */
    SPI_CSB = 1;                          /* SPI chip select inactive */
    return spi_ret;
}
```

4.3.3 MC33970 Device Control

The MC33970 device is controlled from the microcontroller via the 16-bit SPI protocol and reports back the status information. The MC33970 uses six registers to control the device. The registers are addressed via D15:D13 of the SPI word. See **Table 4-10**. For more details refer to the *MC33970 Data Sheet* (Motorola document order number MC33970/D).

Table 4-10. MC33970 Registers

Address [D15:D13]	Name	Use
000	PECCR	Power, Enable, Calibration and Configuration Register
001	VELR	Maximum Velocity Register
010	POS0R	Gauge 0 Position Register
011	POS1R	Gauge 1 Position Register
100	RTZR	Return to Zero Register
101	RTZCR	Return to Zero Configuration Register
110	N.A.	Not Used
111	N.A.	Reserved for Test

The control of the device is done by macros defined in the **GDIC970.h** file. The macro creates a 16-bit SPI data word and calls *SPI_SendRecv16(tU16 data)* function. The symbolic constants for the parameters of the macros are defined in **GDIC970.h.** A short description of all macro syntaxes is made in the following subsections.

4.3.3.1 PECCR_CMD Macro

This macro controls the “Power, Enable, Calibration and Configuration Register”. Syntax of the macro is as follows:

PECCR_CMD(nullcmd,statusselect,gaugepos,rtzloc,aircore,clkcalsel,clkcal,oscadj,gauge1,gauge0)

- nullcmd
  - Enables or disables the Null Command for the device Status read.
    NULL_CMD_ENA
    NULL_CMD_DIS
- statusselect
  - Selects the information that is clocked out of the MISO bit
    STATUS_OUT
    ACCUM_OUT
    POS_OUT
    SPEED_OUT

- gaugepos
  - Selects the gauge for which the zero position will be determined by “rtzloc” parameter
    GAUGE1_POSITION
    GAUGE0_POSITION
- rtzloc
  - Determines the zero position for the gauge selected by “gaugepos” parameter
    CW_ZERO_POSITION
    CCW_ZERO_POSITION
- aircore
  - Enables or disables the Air Core Motor emulation
    AIR_CORE_EMUL_DIS
    AIR_CORE_EMUL_ENA
- clkcalsel
  - Selects the clock calibration frequency
    NOMINAL_F
    MAXIMUM_F
- clkcal
  - Enables or disables the clock calibration
    CLK_CAL_ENA
    CLK_CAL_DIS
- oscadj
  - Adjusts the oscillator to “0,66xTosc” or “Tosc”
    TOSC066
    TOSC
- gauge1
  - Enable or disable the output driver of Gauge 1
    GAUGE1_ENA
    GAUGE1_DIS
- gauge0
  - Enable or disable the output driver of Gauge 0
    GAUGE0_ENA
    GAUGE0_DIS

4.3.3.2 VELR_CMD Macro

This macro controls “Maximum Velocity Register”. Syntax of the macro is as follows:

VELR_CMD(gauge1vel,gauge0vel,maxvel)

- gauge1vel
  - Specifies whether the maximum velocity specified in the “maxvel” parameter will apply to gauge 1 or not.
    VEL_NOTAPPLY_G1
    VEL_APPLY_G1
- gauge0vel
  - Specifies whether the maximum velocity specified in the “maxvel” parameter will apply to gauge 0 or not.
    VEL_NOTAPPLY_G0
    VEL_APPLY_G0
- maxvel
  - Specifies the maximum velocity position from the acceleration table. Velocities can range from position 1 to position 255. For more details refer to the datasheet.

4.3.3.3 POS0R_CMD Macro

This macro controls “Gauge 0 Position Register”. Syntax of the macro is as follows:

POS0R_CMD(position)

- position
  - Determines the desired pointer position of the gauge. Pointer position can range from 0 to position 4095.

4.3.3.4 POS1R_CMD Macro

This macro controls “Gauge 1 Position Register”. Syntax of the macro is as follows:

POS1R_CMD(position)

- position
  - Determines the desired pointer position of the gauge. Pointer position can range from 0 to position 4095.

4.3.3.5 RTZR_CMD Macro

This macro controls “Gauge Return to Zero Register”. Syntax of the macro is the following:

RTZR_CMD(rtzevent,rtzdir,rtzenable,gaugesel)

- rtzevent
  - This parameter selects between Unconditional or Automatic return to zero events
    RTZ_AUTO
    RTZ_UNCON
- rtzdir
  - Determines if return to zero event will occur in the CW or CCW direction
    CW_RTZ
    CCW_RTZ
- rtzenable
  - Enable or disable RTZ
    RTZ_DIS
    RTZ_ENA
- gaugesel
  - Select the gauge to be commanded
    GAUGE0
    GAUGE1

4.3.3.6 RTZCR_CMD Macro

This macro controls “Gauge Return to Zero Configuration Register”. This register modifies the step time, at which the pointer moves during the RTZ event. The full step time is generated using the following equation:

FullStep(t) = Delta(t) * M + blanking(t)

Syntax of the macro is as follows:

RTZCR_CMD(fsmul,preloadval,rtzblnk,rtzfullstp)

- fsmul
  - This parameter determines the multiplier “M” of the equation .
    FS_MUL1
    FS_MUL2
    FS_MUL4
    FS_MUL8
- preloadval
  - Determines the value that is used for the calculation of the preloaded value into the RTZ integration accumulator, to adjust the detection threshold. Value ranges from 0 to 63.
- rtzblnk
  - This parameter determines the RTZ blanking time “blanking(t)” of the equation .
    RTZBLNK512
    RTZBLNK768
- rtzfullstp
  - Determines the time variable “Delta(t)” of the equation .
    FST0
    FST4096
    FST8192
    FST12288
    FST16384
    FST20480
    FST24576
    FST28672
    FST32768
    FST36864
    FST40960
    FST45056
    FST49152
    FST53248
    FST57344
    FST61440

4.3.3.7 GDIC Device Initialization

The initialization is implemented in *Gdic_init()* routine (**GDIC.c**).

There is a couple of key actions to initialize the gauge device:

- Apply RESET to GDIC
- Disable gauges before calibration
- Clock calibration
- Enable both gauges
- Read status
- Wait 40 ms, allowing power to be applied to the motor coils
- Set zero position to gauge 0
- Set zero position to gauge 1

4.3.4 LCD Control

An LCD display with 4 backplanes and 12 frontplanes is used within the application, see **Figure 4-4**. The MCU is able to control the display with up to 4 backplanes and 26 frontplanes by use of the LCD data registers. Each LCD data register controls two frontplanes. The LCD driver module uses an LCD base clock derived from the MCU oscillator. The configuration is done in the LCD clock register. Due to four backplanes of the LCD display, a 1/4 LCD output waveform duty is required.

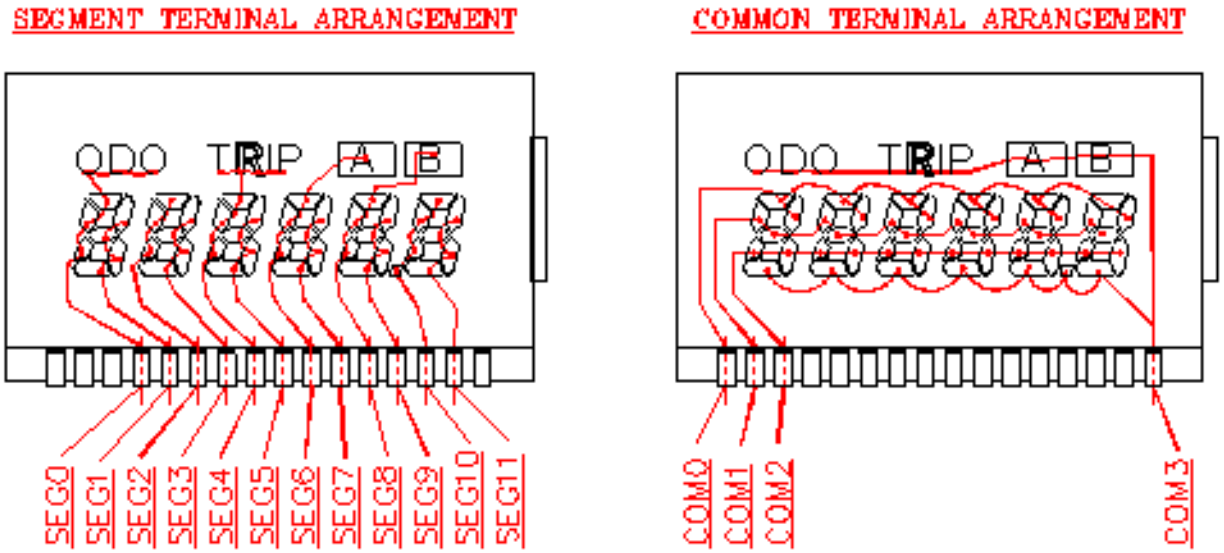

Figure 4-4. LCD Display Arrangement

4.3.4.1 LCD Symbols Coding

The LCD display used in the application contains six 7-segment digits and four labels; ODO, TRIP, A, B. Symbolic labels for each segment of the 7-segment digit can be seen in **Figure 4-5**

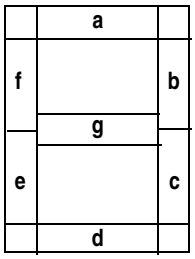

Figure 4-5. Display Segments Labels

The LCD symbols coding is done through the coding table. See **Figure 4-6**.

Input Code	Indicated Symbol	LCD Data Register D7	D6	D5	D4	D3	D2	D1	D0	HEX
		Segment label d	c	g	b	lab	e	f	a	
0	0	1	1	0	1	0	1	1	1	0xD7
1	1	0	1	0	1	0	0	0	0	0x50
2	2	1	0	1	1	0	1	0	1	0xB5
3	3	1	1	1	1	0	0	0	1	0xF1
4	4	0	1	1	1	0	0	1	0	0x72
5	5	1	1	1	0	0	0	1	1	0xE3
6	6	1	1	1	0	0	1	1	1	0xE7
7	7	0	1	0	1	0	0	0	1	0x51
8	8	1	1	1	1	0	1	1	1	0xF7
9	9	1	0	1	1	0	1	1	1	0xB7
10	C	1	0	0	0	0	1	1	1	0x87
11	A	0	1	1	1	0	1	1	1	0x77
12	L	1	0	0	0	0	1	1	0	0x86
13	S	1	1	1	0	0	0	1	1	0xE3
14	r	0	0	1	0	0	1	0	0	0x24
15	E	1	0	1	0	0	1	1	1	0xA7
16	P	0	0	1	1	0	1	1	1	0x37
17	d	1	1	1	1	0	1	0	0	0xF4
18	o	1	1	1	0	0	1	0	0	0xE4
19	b	1	1	1	0	0	1	1	0	0xE6
20	n	0	1	1	0	0	1	0	0	0x64
21	blank	0	0	0	0	0	0	0	0	0x00
22	U	1	1	0	1	0	1	1	0	0xD6

Figure 4-6. LCD Coding Table

The “Input code” in the table represents a parameter (number) of the *Lcd_seg(x)_display (number)* routine.

NOTE: *x in the routine name represents an identifier of the LCD digit.*

The “Indicated symbol” in the table represents a symbol to be displayed on the selected digit of the LCD display. The next columns represent the segment assignments for each bit in the LCD data registers.

The coding table is realized by the vector *lcdconv[]* (**lcd.c**).

```
const unsigned char lcdconv[]=
{0xD7, 0x50, 0xB5, 0xF1, 0x72, 0xE3, 0xE7, 0x51, 0xF7, 0xF3, 0x87, 0x77, 0x86, 0xE3, 0x24, 0xA7,
0x37, 0xF4, 0xE4, 0xE6, 0x64, 0x00, 0xD6};
/*  0     1     2     3     4     5     6     7     8     9  10->C 11->A 12->L 13->S 14->r 15->E
16->P 17->d 18->o 19->b 20->n 21->blanc 22->U */
```

4.3.4.2 LCD Initialization

The initialization is implemented in the *Lcd_init()* routine (**lcd.c**).

Here is complete listing of the *Lcd_init().*

```
/*****************************************************************************
 * Function:        Lcd_init(void)
 *
 * Description:     LCD initialization
 *
 *
 * Returns:         none
 *
 * Notes:
 *
 *****************************************************************************/

void Lcd_init(void)
{
    LCDCLK = LCD_CTL_INIT;
    LDAT1 = 0;
    LDAT2 = 0;
    LDAT3 = 0;
    LDAT4 = 0;
    LDAT5 = 0;
    LDAT6 = 0;
    LDAT7 = 0;
    LCDCR_LCDE = 1;         /* LCD enable */
}
```

4.3.5 Keyboard Control

The MCU has an embedded Keyboard Interrupt module (KBI) that can provide eight independently maskable external interrupts. When a port pin is enabled for keyboard interrupt function, an internal 30 kOhm pull-up device is enabled on the pin. The application uses two pins only: KBI0, and KBI4. The KBI0 pin serves the “ODO/TRIP” switch, and the KBI4 serves power down sensing function.

4.3.5.1 KBI Initialization

The initialization is implemented in the *Keyboard_init()* routine (**keyboard.c**).

Here is complete listing of the *Keyboard_init().*

```
/*****************************************************************************
 * Function:        Keyboard_init(void)
 *
 * Description:     keyboard initialization
 *
 *
 * Returns:         none
 *
 * Notes:
 *
 *****************************************************************************/

 void Keyboard_init(void)
    {
    KBSCR_IMASKK = 1;           /* Mask keyboard interrupts */
        KBIER_KBIE0 = 1;            /* Enable KBIE0 pin */
        KBIER_KBIE4 = 1;            /* Enable KBIE4 pin */
    KBSCR_ACKK = 1;             /* Clear any keyboard interrupts */
    KBSCR_IMASKK = 0;           /* NOT Mask keyboard interrupts */
        }
```

4.3.6 Timer Control

The MCU provides Timer Interface Module (TIM), which is a two-channel timer that provides a timing reference with input capture, output compare, and pulse-width modulation functions. The application uses both timers. Timer 1 is used in the input capture mode to measure periods of the pulses from the speed and revolution sensors. Timer 2 is used in the output compare mode to perform watchdog periods for the speedometer and tachometer functions.

4.3.6.1 TIM Initialization

The initialization is implemented in *Timer1_init()*, and *Timer2_init()* routines (**timer.c**). The *Timer1_init() routine* initialize both channels of the TIM1 module for the input capture function.

Here is complete listing of the *Timer1_init().*

```
/*************************************************************************
 * Function:        Timer1_init(void)
 *
 * Description:     timer initialization
 *
 *
 * Returns:         none
 *
 * Notes:
 *
 *************************************************************************/

 void Timer1_init(void)
    {
        T1SC = TIMER_PRESC_32;       /* Setup Timer for bus clock 4MHz/32 = 8 usec/count */
    T1SC0 = TIMER_INP_CAP_FALLING;    /* Capture on falling edge of ch0 */
    T1SC1 = TIMER_INP_CAP_FALLING;    /* Capture on falling edge of ch1 */
        }
```

The *Timer2_init() routine* initializes both channels of the TIM2 module for the output compare function.

Here is complete listing of the *Timer2_init().*

```
/*************************************************************************
 * Function:        Timer2_init(void)
 *
 * Description:     timer initialization
 *
 *
 * Returns:         none
 *
 * Notes:
 *
 *************************************************************************/

 void Timer2_init(void)
    {
        T2SC = TIMER_PRESC_32;       /* Setup Timer for bus clock 4MHz/32 = 8 usec/count */
    T2SC0 = TIMER_OUT_PRESET0;
    T2CH0H = 0;
    T2CH0L = 0;
    T2SC1 = TIMER_OUT_PRESET0;
    T2CH1H = 0;
    T2CH1L = 0;
        }
```

4.3.7 External EEPROM Control

An external EEPROM is used to store Odometer and Tripmeter data. The device is compatible with the I^2C memory protocol and behaves as a slave. MCU emulates the I^2C protocol by software. The I^2C communication channel is initiated by the *I2Cinit()* routine. Data is stored by the *SaveData(data)*, and read by the *ReadData(data)* routines.

4.3.8 ATD Module Control

The Analog to Digital Converter (ADC) measures the signal from the fuel level sensor. The ADC has a 6-channel 10-bit linear successive approximation. The application uses only one channel of the ADC with 8-bit resolution.

The *Ad_convert(chan) routine initializes ADC, makes the conversion, and returns the 8-bit value.*

Here is complete listing of the *Ad_convert(chan).*

```
/*************************************************************************
 * Function:        unsigned char Ad_convert(unsigned char chan)
 *
 * Description:     services A/D conversion in a 8-bit truncation mode
 *
 * Returns:         unsigned char - content of the ADRL register
 *
 * Notes:
 *
 *
 *
 *************************************************************************/

unsigned char Ad_convert(unsigned char chan)
{
    ADCLK = ADCLK_PRESET;
    ADSCR = chan;           // start conversion
    while (!ADSCR_COCO);    // & wait until finished
    return ADRL;
}
```

4.3.9 SCI Module Control

The SCI module is not utilized within the application, it is used by the Developer's Serial Bootloader for M68HC08. It allows an in-circuit reprogramming of Motorola's M68HC08 FLASH devices using standard communication media (e.g., a serial asynchronous port). For further information, refer to application note entitled *Developer's Serial Bootloader for M68HC08* (Motorola document order number AN2295). The serial bootloader offers a zero-cost solution for applications already equipped with a serial interface and that have SCI pins available on a connector.

Software Description

Appendix A. Bill of Materials and Schematics

A.1 Speedometer Bill of Materials

Item	Quantity	Reference	Part		Dstributor Number
1	2	C1,C15	470μF/35V		Farnell 320-1600
2	2	C5,C2	1μF/50V		Farnell 556-312
3	2	C12,C3	1nF	C0805	
4	1	C4	100μF/6.3V		Farnell 556-130
5	7	C6,C9,C10,C11,C13, C14,C16	100nF	C0805	Farnell 499-687
6	2	C7,C8	27pF	C0805	
7	2	C17,C18 C19	10nF 33nF	C0805 C0805	
8	1	D2,D3	MRA4007T3	ON Semiconductor	
9	4	D1,D4,D6,D7	1N4148		
10	2	D2,D3	MRA4007T3	ON Semiconductor	
11	1	D5	BZX84C9V1	ON Semiconductor	
12	2	D9,D10	BZX84C4V3	ON Semiconductor	
13	1	D8	1SMA5939BT3	ON Semiconductor	
14	1	JP1	HDR 5X1	Molex	Molex 39-30-2050
15	1	JP2	HDR 3X2		Farnell 511-780
16	1	J1	HEADER 4	Molex	Molex 43650-0401
17	1	J3	HDR8	Molex	Molex 43045-0812
18	1	L1	BEAD	Steward	
19	2	Q1,Q2	BC847B		Farnell 932-980
20	1	Q3	BCP52		Farnell 932-954
21	1	R1	27k	R0805	
22	10	R2,R5,R8,R9,R10,R11, R12,R13,R18,R22	10k	R0805	Farnell 911-975
23	1	R3	39k	R0805	

Table continued on the next page

Item	Quantity	Reference	Part		Dstributor Number
24	1	R4	15k	R0805	
25	1	R6	330k	R0805	
26	1	R7	10M	R0805	
27	2	R14,R15	100k	R0805	
28	1	R16	3R9		
29	1	R17	33R	R0805	
30	1	R19	1k5	R0805	
31	2	R20,R21	4k7	R0805	
32	1	SW1	SWITCH-4	Omron	Farnell 328-1899
33	1	S1	KSC241J		
34	9	TP1,TP2,TP3,TP4,TP5, TP6,TP7,TP8,TP9	TERMINAL	Not populated	
35	1	U2	MC7805BT	ON Semiconductor	
36	1	U3	MC33991DW	Motorola	
37	1	U4	STEPPER MOTOR	Sonceboz	
38	1	U5	M24C04-MN6		Farnell 302-5263
39	1	U7	MC68HC908LJ12CPB	Motorola	
40	1	U10	LCD		
41	1	X1	XT/C 0.032TC		Farnell 571-672

A.2 Cluster for Motorbikes Schematics

Refer to **Figure A-1**, **Figure A-2**, and **Figure A-3**.

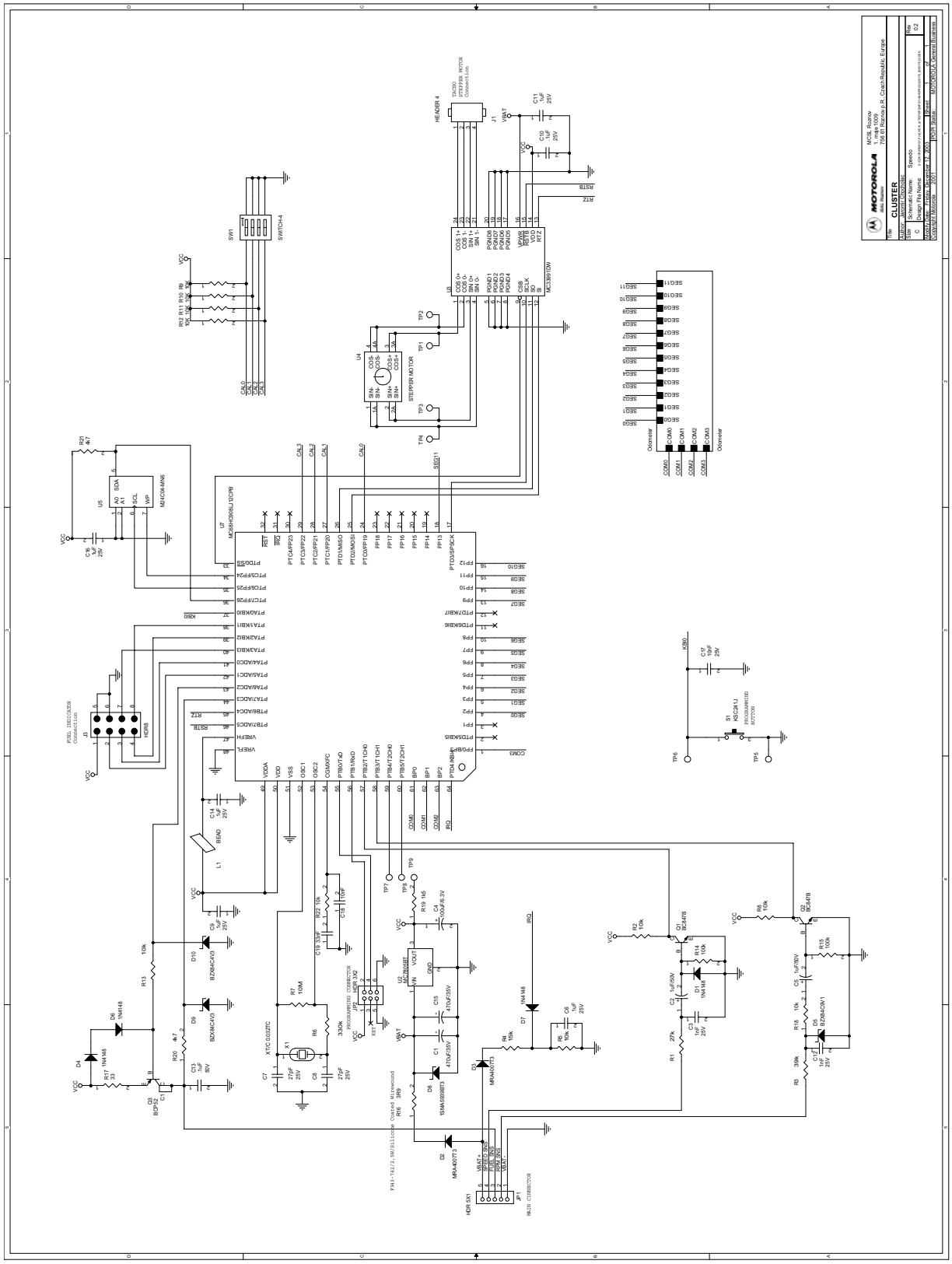

Figure A-1. Speedometer Schematic

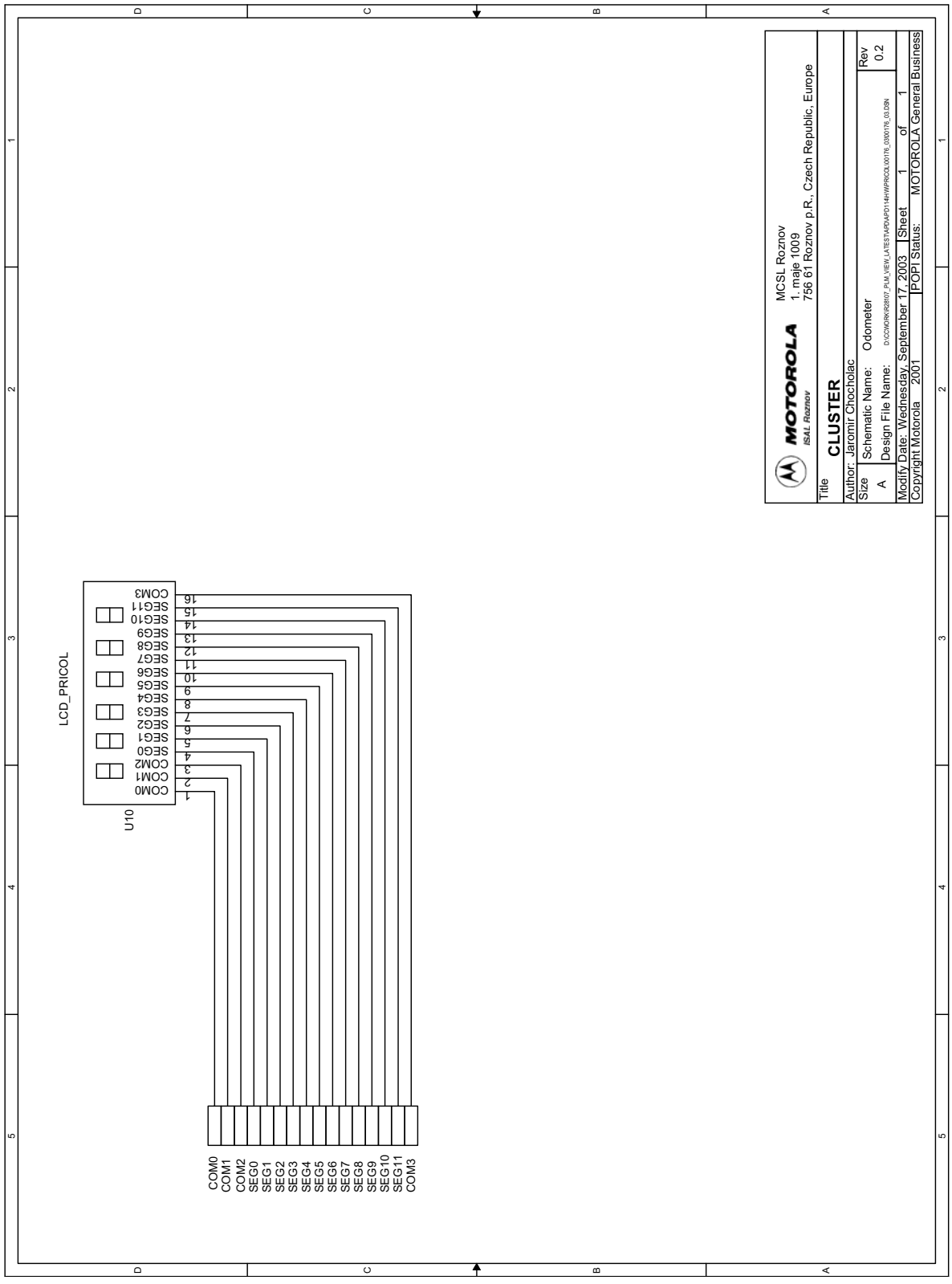

Figure A-2. Odometer Schematic

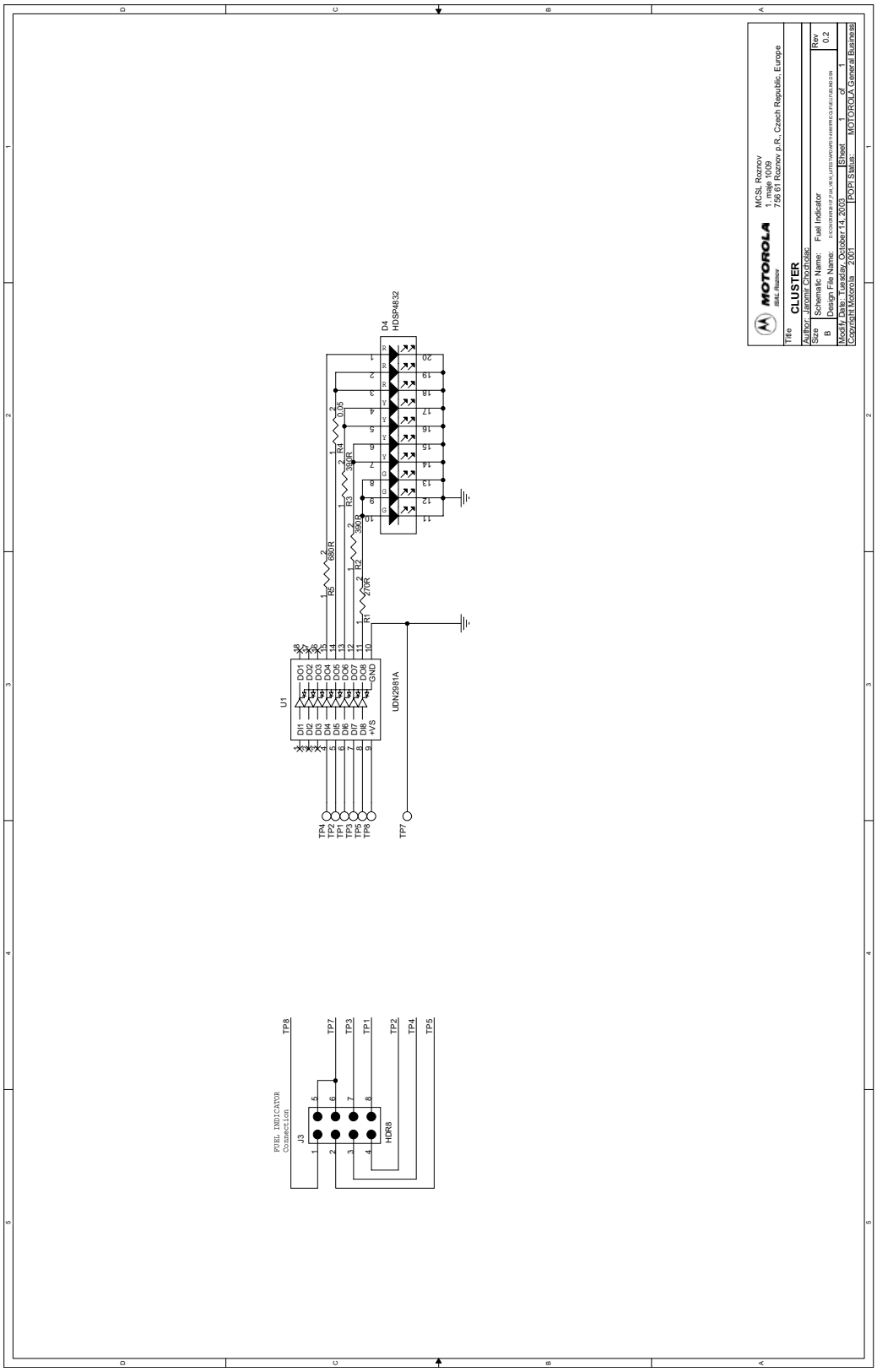

Figure A-3. Demo Fuel Indicator

Bill of Materials and Schematics

Glossary

A — See “accumulators (A and B or D).”

accumulators (A and B or D) — Two 8-bit (A and B) or one 16-bit (D) general-purpose registers in the CPU. The CPU uses the accumulators to hold operands and results of arithmetic and logic operations.

acquisition mode — A mode of PLL operation with large loop bandwidth. Also see ’tracking mode’.

address bus — The set of wires that the CPU or DMA uses to read and write memory locations.

addressing mode — The way that the CPU determines the operand address for an instruction. The M68HC12 CPU has 15 addressing modes.

ALU — See “arithmetic logic unit (ALU).”

analogue-to-digital converter (ATD) — The ATD module is an 8-channel, multiplexed-input successive-approximation analog-to-digital converter.

arithmetic logic unit (ALU) — The portion of the CPU that contains the logic circuitry to perform arithmetic, logic, and manipulation operations on operands.

asynchronous — Refers to logic circuits and operations that are not synchronized by a common reference signal.

ATD — See “analogue-to-digital converter”.

B — See “accumulators (A and B or D).”

baud rate — The total number of bits transmitted per unit of time.

BCD — See “binary-coded decimal (BCD).”

binary — Relating to the base 2 number system.

binary number system — The base 2 number system, having two digits, 0 and 1. Binary arithmetic is convenient in digital circuit design because digital circuits have two permissible voltage levels, low and high. The binary digits 0 and 1 can be interpreted to correspond to the two digital voltage levels.

binary-coded decimal (BCD) — A notation that uses 4-bit binary numbers to represent the 10 decimal digits and that retains the same positional structure of a decimal number. For example,

234 (decimal) = 0010 0011 0100 (BCD)

Glossary

bit — A binary digit. A bit has a value of either logic 0 or logic 1.

branch instruction — An instruction that causes the CPU to continue processing at a memory location other than the next sequential address.

break module — The break module allows software to halt program execution at a programmable point in order to enter a background routine.

breakpoint — A number written into the break address registers of the break module. When a number appears on the internal address bus that is the same as the number in the break address registers, the CPU executes the software interrupt instruction (SWI).

break interrupt — A software interrupt caused by the appearance on the internal address bus of the same value that is written in the break address registers.

bus — A set of wires that transfers logic signals.

bus clock — See "CPU clock".

byte — A set of eight bits.

CAN — See "Motorola scalable CAN."

CCR — See “condition code register.”

central processor unit (CPU) — The primary functioning unit of any computer system. The CPU controls the execution of instructions.

CGM — See “clock generator module (CGM).”

clear — To change a bit from logic 1 to logic 0; the opposite of set.

clock — A square wave signal used to synchronize events in a computer.

clock generator module (CGM) — The CGM module generates a base clock signal from which the system clocks are derived. The CGM may include a crystal oscillator circuit and/or phase-locked loop (PLL) circuit.

comparator — A device that compares the magnitude of two inputs. A digital comparator defines the equality or relative differences between two binary numbers.

computer operating properly module (COP) — A counter module that resets the MCU if allowed to overflow.

condition code register (CCR) — An 8-bit register in the CPU that contains the interrupt mask bit and five bits that indicate the results of the instruction just executed.

control bit — One bit of a register manipulated by software to control the operation of the module.

control unit — One of two major units of the CPU. The control unit contains logic functions that synchronize the machine and direct various operations. The control unit decodes instructions and generates the internal control signals that perform the requested operations. The outputs of the control unit drive the execution unit, which contains the arithmetic logic unit (ALU), CPU registers, and bus interface.

COP — See "computer operating properly module (COP)."

CPU — See “central processor unit (CPU).”

CPU12 — The CPU of the MC68HC12 Family.

CPU clock — Bus clock select bits BCSP and BCSS in the clock select register (CLKSEL) determine which clock drives SYSCLK for the main system, including the CPU and buses. When EXTALi drives the SYSCLK, the CPU or bus clock frequency (f_o) is equal to the EXTALi frequency divided by 2.

CPU cycles — A CPU cycle is one period of the internal bus clock, normally derived by dividing a crystal oscillator source by two or more so the high and low times will be equal. The length of time required to execute an instruction is measured in CPU clock cycles.

CPU registers — Memory locations that are wired directly into the CPU logic instead of being part of the addressable memory map. The CPU always has direct access to the information in these registers. The CPU registers in an M68HC12 are:

A (8-bit accumulator)

B (8-bit accumulator)

D (16-bit accumulator formed by concatenation of accumulators A and B)

IX (16-bit index register)

IY (16-bit index register)

SP (16-bit stack pointer)

PC (16-bit program counter)

CCR (8-bit condition code register)

cycle time — The period of the operating frequency: $t_{CYC} = 1/f_{OP}$.

D — See “accumulators (A and B or D).”

decimal number system — Base 10 numbering system that uses the digits zero through nine.

duty cycle — A ratio of the amount of time the signal is on versus the time it is off. Duty cycle is usually represented by a percentage.

ECT — See “enhanced capture timer.”

EEPROM — Electrically erasable, programmable, read-only memory. A nonvolatile type of memory that can be electrically erased and reprogrammed.

EPROM — Erasable, programmable, read-only memory. A nonvolatile type of memory that can be erased by exposure to an ultraviolet light source and then reprogrammed.

Glossary

enhanced capture timer (ECT) — The HC12 Enhanced Capture Timer module has the features of the HC12 Standard Timer module enhanced by additional features in order to enlarge the field of applications.

exception — An event such as an interrupt or a reset that stops the sequential execution of the instructions in the main program.

fetch — To copy data from a memory location into the accumulator.

firmware — Instructions and data programmed into nonvolatile memory.

free-running counter — A device that counts from zero to a predetermined number, then rolls over to zero and begins counting again.

full-duplex transmission — Communication on a channel in which data can be sent and received simultaneously.

hexadecimal — Base 16 numbering system that uses the digits 0 through 9 and the letters A through F.

high byte — The most significant eight bits of a word.

illegal address — An address not within the memory map

illegal opcode — A nonexistent opcode.

index registers (IX and IY) — Two 16-bit registers in the CPU. In the indexed addressing modes, the CPU uses the contents of IX or IY to determine the effective address of the operand. IX and IY can also serve as a temporary data storage locations.

input/output (I/O) — Input/output interfaces between a computer system and the external world. A CPU reads an input to sense the level of an external signal and writes to an output to change the level on an external signal.

instructions — Operations that a CPU can perform. Instructions are expressed by programmers as assembly language mnemonics. A CPU interprets an opcode and its associated operand(s) and instruction.

inter-IC bus (I^2C) — A two-wire, bidirectional serial bus that provides a simple, efficient method of data exchange between devices.

interrupt — A temporary break in the sequential execution of a program to respond to signals from peripheral devices by executing a subroutine.

interrupt request — A signal from a peripheral to the CPU intended to cause the CPU to execute a subroutine.

I/O — See “input/output (I/0).”

jitter — Short-term signal instability.

latch — A circuit that retains the voltage level (logic 1 or logic 0) written to it for as long as power is applied to the circuit.

latency — The time lag between instruction completion and data movement.

least significant bit (LSB) — The rightmost digit of a binary number.

logic 1 — A voltage level approximately equal to the input power voltage (V_{DD}).

logic 0 — A voltage level approximately equal to the ground voltage (V_{SS}).

low byte — The least significant eight bits of a word.

M68HC12 — A Motorola family of 16-bit MCUs.

mark/space — The logic 1/logic 0 convention used in formatting data in serial communication.

mask — 1. A logic circuit that forces a bit or group of bits to a desired state. 2. A photomask used in integrated circuit fabrication to transfer an image onto silicon.

MCU — Microcontroller unit. See “microcontroller.”

memory location — Each M68HC12 memory location holds one byte of data and has a unique address. To store information in a memory location, the CPU places the address of the location on the address bus, the data information on the data bus, and asserts the write signal. To read information from a memory location, the CPU places the address of the location on the address bus and asserts the read signal. In response to the read signal, the selected memory location places its data onto the data bus.

memory map — A pictorial representation of all memory locations in a computer system.

MI-Bus — See "Motorola interconnect bus".

microcontroller — Microcontroller unit (MCU). A complete computer system, including a CPU, memory, a clock oscillator, and input/output (I/O) on a single integrated circuit.

modulo counter — A counter that can be programmed to count to any number from zero to its maximum possible modulus.

most significant bit (MSB) — The leftmost digit of a binary number.

Motorola interconnect bus (MI-Bus) — The Motorola Interconnect Bus (MI Bus) is a serial communications protocol which supports distributed real-time control efficiently and with a high degree of noise immunity.

Motorola scalable CAN (msCAN) — The Motorola scalable controller area network is a serial communications protocol that efficiently supports distributed real-time control with a very high level of data integrity.

msCAN — See "Motorola scalable CAN".

MSI — See "multiple serial interface".

multiple serial interface — A module consisting of multiple independent serial I/O sub-systems, e.g. two SCI and one SPI.

Glossary

multiplexer — A device that can select one of a number of inputs and pass the logic level of that input on to the output.

nibble — A set of four bits (half of a byte).

object code — The output from an assembler or compiler that is itself executable machine code, or is suitable for processing to produce executable machine code.

opcode — A binary code that instructs the CPU to perform an operation.

open-drain — An output that has no pullup transistor. An external pullup device can be connected to the power supply to provide the logic 1 output voltage.

operand — Data on which an operation is performed. Usually a statement consists of an operator and an operand. For example, the operator may be an add instruction, and the operand may be the quantity to be added.

oscillator — A circuit that produces a constant frequency square wave that is used by the computer as a timing and sequencing reference.

OTPROM — One-time programmable read-only memory. A nonvolatile type of memory that cannot be reprogrammed.

overflow — A quantity that is too large to be contained in one byte or one word.

page zero — The first 256 bytes of memory (addresses $0000–$00FF).

parity — An error-checking scheme that counts the number of logic 1s in each byte transmitted. In a system that uses odd parity, every byte is expected to have an odd number of logic 1s. In an even parity system, every byte should have an even number of logic 1s. In the transmitter, a parity generator appends an extra bit to each byte to make the number of logic 1s odd for odd parity or even for even parity. A parity checker in the receiver counts the number of logic 1s in each byte. The parity checker generates an error signal if it finds a byte with an incorrect number of logic 1s.

PC — See “program counter (PC).”

peripheral — A circuit not under direct CPU control.

phase-locked loop (PLL) — A clock generator circuit in which a voltage controlled oscillator produces an oscillation which is synchronized to a reference signal.

PLL — See "phase-locked loop (PLL)."

pointer — Pointer register. An index register is sometimes called a pointer register because its contents are used in the calculation of the address of an operand, and therefore points to the operand.

polarity — The two opposite logic levels, logic 1 and logic 0, which correspond to two different voltage levels, V_{DD} and V_{SS}.

polling — Periodically reading a status bit to monitor the condition of a peripheral device.

port — A set of wires for communicating with off-chip devices.

prescaler — A circuit that generates an output signal related to the input signal by a fractional scale factor such as 1/2, 1/8, 1/10 etc.

program — A set of computer instructions that cause a computer to perform a desired operation or operations.

program counter (PC) — A 16-bit register in the CPU. The PC register holds the address of the next instruction or operand that the CPU will use.

pull — An instruction that copies into the accumulator the contents of a stack RAM location. The stack RAM address is in the stack pointer.

pullup — A transistor in the output of a logic gate that connects the output to the logic 1 voltage of the power supply.

pulse-width — The amount of time a signal is on as opposed to being in its off state.

pulse-width modulation (PWM) — Controlled variation (modulation) of the pulse width of a signal with a constant frequency.

push — An instruction that copies the contents of the accumulator to the stack RAM. The stack RAM address is in the stack pointer.

PWM period — The time required for one complete cycle of a PWM waveform.

RAM — Random access memory. All RAM locations can be read or written by the CPU. The contents of a RAM memory location remain valid until the CPU writes a different value or until power is turned off.

RC circuit — A circuit consisting of capacitors and resistors having a defined time constant.

read — To copy the contents of a memory location to the accumulator.

register — A circuit that stores a group of bits.

reserved memory location — A memory location that is used only in special factory test modes. Writing to a reserved location has no effect. Reading a reserved location returns an unpredictable value.

reset — To force a device to a known condition.

SCI — See "serial communication interface module (SCI)."

serial — Pertaining to sequential transmission over a single line.

serial communications interface module (SCI) — A module that supports asynchronous communication.

serial peripheral interface module (SPI) — A module that supports synchronous communication.

set — To change a bit from logic 0 to logic 1; opposite of clear.

shift register — A chain of circuits that can retain the logic levels (logic 1 or logic 0) written to them and that can shift the logic levels to the right or left through adjacent circuits in the chain.

Glossary

signed — A binary number notation that accommodates both positive and negative numbers. The most significant bit is used to indicate whether the number is positive or negative, normally logic 0 for positive and logic 1 for negative. The other seven bits indicate the magnitude of the number.

software — Instructions and data that control the operation of a microcontroller.

software interrupt (SWI) — An instruction that causes an interrupt and its associated vector fetch.

SPI — See "serial peripheral interface module (SPI)."

stack — A portion of RAM reserved for storage of CPU register contents and subroutine return addresses.

stack pointer (SP) — A 16-bit register in the CPU containing the address of the next available storage location on the stack.

start bit — A bit that signals the beginning of an asynchronous serial transmission.

status bit — A register bit that indicates the condition of a device.

stop bit — A bit that signals the end of an asynchronous serial transmission.

subroutine — A sequence of instructions to be used more than once in the course of a program. The last instruction in a subroutine is a return from subroutine (RTS) instruction. At each place in the main program where the subroutine instructions are needed, a jump or branch to subroutine (JSR or BSR) instruction is used to call the subroutine. The CPU leaves the flow of the main program to execute the instructions in the subroutine. When the RTS instruction is executed, the CPU returns to the main program where it left off.

synchronous — Refers to logic circuits and operations that are synchronized by a common reference signal.

timer — A module used to relate events in a system to a point in time.

toggle — To change the state of an output from a logic 0 to a logic 1 or from a logic 1 to a logic 0.

tracking mode — A mode of PLL operation with narrow loop bandwidth. Also see ‘acquisition mode.’

two’s complement — A means of performing binary subtraction using addition techniques. The most significant bit of a two’s complement number indicates the sign of the number (1 indicates negative). The two’s complement negative of a number is obtained by inverting each bit in the number and then adding 1 to the result.

unbuffered — Utilizes only one register for data; new data overwrites current data.

unimplemented memory location — A memory location that is not used. Writing to an unimplemented location has no effect. Reading an unimplemented location returns an unpredictable value.

variable — A value that changes during the course of program execution.

VCO — See "voltage-controlled oscillator."

vector — A memory location that contains the address of the beginning of a subroutine written to service an interrupt or reset.

voltage-controlled oscillator (VCO) — A circuit that produces an oscillating output signal of a frequency that is controlled by a dc voltage applied to a control input.

waveform — A graphical representation in which the amplitude of a wave is plotted against time.

wired-OR — Connection of circuit outputs so that if any output is high, the connection point is high.

word — A set of two bytes (16 bits).

write — The transfer of a byte of data from the CPU to a memory location.

Glossary

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:
Motorola Literature Distribution
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 480-768-2130

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://motorola.com/semiconductors

DRM059/D
Rev. 0
3/2004

Freescale Semiconductor, Inc.