Taximeter Reference
Design Using the
MC68HC908JL3
Designer Reference
Manual

M68HC08
Microcontrollers

MOTOROLA.COM/SEMICONDUCTORS

For More Information On This Product,
Go to: www.freescale.com
Taximeter Reference Design Using the MC68HC908JL3
Reference Design

By: Mauricio Capistrán Garza Jirash
Rebeca Delgado
Motorola Semiconductor Products Sector
Mexico

PCB Designed by Raul Hernandez-Arthur
Motorola Semiconductor Products Sector
Mexico

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 summarizes changes contained in this document. For your convenience, the page number designators have been linked to the appropriate location.
<table>
<thead>
<tr>
<th>Date</th>
<th>Revision Level</th>
<th>Description</th>
<th>Page Number(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>November, 2003</td>
<td>N/A</td>
<td>Initial release</td>
<td>N/A</td>
</tr>
</tbody>
</table>
# Table of Contents

## Section 1. General Description
1.1 Introduction .................................................... 11  
1.2 Taximeter Overview ........................................ 11  
1.3 MC68HC908JL3 Microcontroller .................................. 12  

## Section 2. System Requirements
2.1 Introduction .................................................... 13  
2.2 General Requirements ........................................ 13  
2.3 Functional Requirements ...................................... 13  
2.4 Dimensional Requirements .................................... 14  
2.5 Memory Requirements ......................................... 14  
2.6 Display Requirements ......................................... 14  
2.7 Wheel-Turn Indicator Requirements ........................... 14  

## Section 3. Hardware Design
3.1 Introduction .................................................... 15  
3.2 Hardware Design Goals ....................................... 15  
3.3 Block Diagram .................................................. 15  
3.4 Block Description ............................................. 15  
3.5 Schematics ...................................................... 15  
3.5.1 Display Circuit ........................................... 19  
3.5.2 IRQ Circuit ................................................ 20  
3.6 Bill of Materials .............................................. 21  
3.7 Layout ............................................................ 22  

## Section 4. Print Circuit Board (PCB) Design
4.1 Introduction .................................................... 23  
4.2 PCB Layout ...................................................... 23  
4.2.1 Top Layer .................................................. 23  
4.2.2 Bottom Layer ............................................... 24
### Table of Contents

4.3 Mechanical Characteristics ........................................ 24
  4.3.1 Conductor Widths .................................................. 24
  4.3.2 Trace Angles ........................................................ 25
  4.3.3 Trace Distances ..................................................... 25
  4.3.4 Silk Screen Content ............................................... 25
  4.3.5 Layout ............................................................... 25

4.4 Noise-Reduction Characteristics ................................... 25
  4.4.1 EMI-Reduction Measures .......................................... 25
  4.4.2 Oscillator Circuit, MCU, and Power Supply ...................... 26
  4.4.3 Separated Ground Traces ......................................... 27

5.1 Introduction ........................................................... 29

5.2 Programmer and Testing Board Design ............................... 29
  5.2.1 Schematic .......................................................... 29
  5.2.2 Layout ............................................................. 30
  5.2.3 Bill of Materials ................................................... 31
  5.2.4 PCB ................................................................. 32

6.1 Introduction ........................................................... 33

6.2 Software Design Goals ................................................ 33

6.3 General Software Architecture ....................................... 33

6.4 State Machine Description .......................................... 34

6.5 State Machine Diagram ............................................... 36

6.6 Push Button Actions ................................................... 37

6.7 Memory Map ............................................................ 37
  6.7.1 FLASH Memory ...................................................... 37
  6.7.2 Random-Access Memory (RAM) ................................... 38

6.8 File Structure .......................................................... 39

6.9 Naming Conventions .................................................... 40

6.10 Taximeter ISRs Description ......................................... 40
  6.10.1 IRQ ISR ............................................................. 40
  6.10.2 Timer ISR .......................................................... 41

6.11 Taximeter API Description .......................................... 41
  6.11.1 Delay ............................................................... 41
  6.11.2 WaitForButtonsRelease ......................................... 41
  6.11.3 DisplayMsg ........................................................ 42
  6.11.4 DisplayNum ........................................................ 42
# Table of Contents

[For More Information On This Product, Go to: www.freescale.com]
### List of Figures and Tables

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Common Taximeter Add-Ons</td>
<td>12</td>
</tr>
<tr>
<td>3-1</td>
<td>Taximeter Hardware Block Diagram</td>
<td>16</td>
</tr>
<tr>
<td>3-2</td>
<td>Taximeter Schematic</td>
<td>17</td>
</tr>
<tr>
<td>3-3</td>
<td>Display Circuit</td>
<td>19</td>
</tr>
<tr>
<td>3-4</td>
<td>IRQ Circuit</td>
<td>20</td>
</tr>
<tr>
<td>3-5</td>
<td>Taximeter Layout</td>
<td>22</td>
</tr>
<tr>
<td>4-1</td>
<td>Taximeter PCB (Top Layer)</td>
<td>23</td>
</tr>
<tr>
<td>4-2</td>
<td>Taximeter PCB Bottom Layer</td>
<td>24</td>
</tr>
<tr>
<td>4-3</td>
<td>Taximeter Oscillator Circuit and MCU</td>
<td>26</td>
</tr>
<tr>
<td>4-4</td>
<td>Ground Net Underneath the LED Displays</td>
<td>27</td>
</tr>
<tr>
<td>5-1</td>
<td>Programmer and Testing Board Schematic</td>
<td>30</td>
</tr>
<tr>
<td>5-2</td>
<td>Programmer and Testing Board Layout</td>
<td>30</td>
</tr>
<tr>
<td>5-3</td>
<td>Programmer and Testing PCB</td>
<td>32</td>
</tr>
<tr>
<td>6-1</td>
<td>Software Architecture Layers</td>
<td>33</td>
</tr>
<tr>
<td>6-2</td>
<td>State Machine Diagram</td>
<td>36</td>
</tr>
<tr>
<td>6-3</td>
<td>FLASH Memory Sections</td>
<td>38</td>
</tr>
<tr>
<td>6-4</td>
<td>RAM Sections</td>
<td>39</td>
</tr>
<tr>
<td>7-1</td>
<td>Information Exchange</td>
<td>49</td>
</tr>
</tbody>
</table>
# List of Figures and Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>Block Description</td>
<td>16</td>
</tr>
<tr>
<td>3-2</td>
<td>Taximeter Bill of Materials</td>
<td>21</td>
</tr>
<tr>
<td>5-1</td>
<td>Programmer and Testing Board Bill of Materials</td>
<td>31</td>
</tr>
<tr>
<td>6-1</td>
<td>Description of States</td>
<td>35</td>
</tr>
<tr>
<td>6-2</td>
<td>Push Button Actions</td>
<td>37</td>
</tr>
<tr>
<td>6-3</td>
<td>Software File Structure</td>
<td>39</td>
</tr>
<tr>
<td>6-4</td>
<td>Naming Conventions</td>
<td>40</td>
</tr>
<tr>
<td>7-1</td>
<td>FLASH Tables and Variables</td>
<td>47</td>
</tr>
</tbody>
</table>
Section 1. General Description

1.1 Introduction

Motorola is a leader in automotive electronics. It has stated its leadership by being the number one seller of microcontrollers for automotive applications. Now, Motorola has developed this reference design for a low-cost high-performance ultra-thin taximeter.

1.2 Taximeter Overview

The taximeter layout is compact and thin. The physical proportions allow it to be manufactured as a very portable electronic device. The taximeter hardware is a low-cost microcontroller-based system which controls:

- Six displays
- Five push buttons
- A wheel turn indicator
- A programming interface

This reference design follows the NOM-007-SCFI-1997 regulation, which defines the requirements for taximeters in Mexico. This regulation is similar to regulations all over Latin America and can be found at:

http://www.se.gob.mx

The taximeter software was developed in "C" programming language using Metrowerks CodeWarrior; therefore, it is very flexible and easy to manipulate. The modular architecture makes further development easy to integrate. The reference design includes an API that has been developed to facilitate and optimize the creation of new modules. For further information about CodeWarrior visit:

http://www.metrowerks.com

This reference design can be used as a development board because it includes:

- MON08 connector for development purposes
- Programming software design (PC, Windows based)
- Emulation and testing board design
General Description

The taximeter also has an 8-bit expansion port which can be used in the MCU for customer specific applications. The most common applications are:

- Thermo-printer
- RF communication module (Tango-Romeo)
- GPS

![Figure 1-1. Common Taximeter Add-Ons](image)

1.3 MC68HC908JL3 Microcontroller

The MC68HC908JL3 microcontroller has the following characteristics that are relevant to this design:

- Pin count: 28 pins
- FLASH memory size: 4096 bytes
- In-system FLASH programming
- 128 bytes of on-chip random-access-memory (RAM)
- Timer interface module
- 23 general-purpose I/O ports
- Efficient C language support
Section 2. System Requirements

2.1 Introduction

Most of the requirements for this reference design are stated by Mexican regulations. Other requirements were added in order to improve the functionality and resistance, and to reduce the cost and design time.

The Mexican regulation can be found at: http://www.se.gob.mx

2.2 General Requirements

The taximeter has the following general requirements:

- The user interface must be simple
- The contact area of push buttons should be at least 50 mm²
- The energy source must be provided by a car
- It must be able to show its serial number and the license plates
- The cost of production should be low
- The software architecture must be modular
- The design should include the possibility of being used as a development board for new modules

2.3 Functional Requirements

The taximeter has the following functional requirements:

- It must be able to handle different fares
- It must be able to change the number of fares and the cost of them in less than 15 minutes
- It must be able to perform the following functions:
  - Change from FREE → IN SERVICE → PAY with a single button
  - Show taximeter information
  - Keep track of accumulators
2.4 Dimensional Requirements

The outer box should be at least 14 cm long, 5 cm high, and 2 cm deep. The outer box shouldn't be bigger than 18 cm long, 9 cm high, 16 cm deep.

2.5 Memory Requirements

Read-only memories should be able to hold data for five years (even if the device is not used).

The taximeter must store in memory:

- Total distance traveled by the automobile
- Total distance traveled in-service
- Total number of travels in-service
- Total number of increments of the amount to pay
- Total income

2.6 Display Requirements

Display requirements include:

- It must have at least five digits
- It must have one display that shows the active fare at all times
- The digits must be at least 12.7 mm high

2.7 Wheel-Turn Indicator Requirements

The wheel-turn indicator must work in the speed range of 0 Km/h to 120 Km/h. For this reference design, the assumption was made that the wheel's perimeter is 1 meter. This assumption affects how the speed of the car is calculated. However, it doesn't affect how the accumulators (distance in-service distance traveled) are kept, since what the taximeter accumulates is the number of wheel revolutions.

**NOTE:** The actual perimeter of the wheel may vary from one car to another.
Section 3. Hardware Design

3.1 Introduction

A taximeter is a device that indicates the amount to be charged for use of a taxi. As discussed in this section, the taximeter is controlled by five buttons; with these buttons the user may select between many different functions. The taximeter must also have visual feedback (displays).

3.2 Hardware Design Goals

This hardware design was developed with the following goals:
- Compliance with the Mexican regulation NOM-007-SCFI-1997. This regulation establishes many requirements: mechanical, dimensional, interface, external conditions, etc. Refer to Section 2. System Requirements for further information.
- EMI proof — since it will be surrounded by automotive systems, the taximeter must be able to work even in a noisy environment
- Small — as small as possible to save on cost of materials
- Without glue logic — to make it cost-competitive and easily upgradeable
- Serve as development board — the design also provides a development board for software.

3.3 Block Diagram

Figure 3-1 shows the taximeter hardware block diagram.

3.4 Block Description

Table 3-1 provides a description of each block.

3.5 Schematics

Figure 3-2 shows the taximeter schematics. Also, the schematics are available for download at:
Motorola (http://www.motorola.com) > Semiconductors > Microcontrollers > Reference Designs
Table 3-1. Block Description

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller</td>
<td>The taximeter hardware is based on the Motorola microcontroller MC68HC908JL3.</td>
</tr>
<tr>
<td>Wheel turn indicator</td>
<td>Every time the wheel turns, a pulse is sent to the the IRQ pin. To avoid high-frequency noise that could cause multiple IRQ requests within one revolution, a passive low-pass filter is included.</td>
</tr>
<tr>
<td>MON08</td>
<td>Connector MON08 is included. Through this connector the microcontroller can be programmed. Therefore, this reference design can also be used as a development board</td>
</tr>
<tr>
<td>Display enables</td>
<td>Port A is designated to perform two tasks. The first task is to enable the six displays at given times. Since multiplexing is taking place, a minimum current of 200 mA is needed in order to reach the 350uCD (needed luminosity) per segment. Since Port A is not capable of supplying that amount of current, NPN transistors are included.</td>
</tr>
<tr>
<td>Push buttons</td>
<td>Port A’s second task is to read five push buttons. The push buttons are connected to port A before the transistors that lead to the common anode of the displays. The push buttons have pull-down resistors, so the microcontroller must enable its internal pull-up resistors while reading the push buttons.</td>
</tr>
<tr>
<td>Display segments</td>
<td>Port B is designated to send data to the displays’ segments. However, port B is not capable of draining the needed current since through each segment flows from 20 to 25 mA. Therefore, PNP transistors are also included.</td>
</tr>
<tr>
<td>Programming interface</td>
<td>Pin 0 of port B has a dual purpose. It turns on or off a segment of the displays and it is also the programming interface for the taximeter.</td>
</tr>
<tr>
<td>Expansion port</td>
<td>Port D is free for custom applications</td>
</tr>
</tbody>
</table>
Figure 3-2. Taximeter Schematic (Sheet 2 of 2)
3.5.1 Display Circuit

The displays are multiplexed in time. Since the microcontroller can not provide enough current to power the displays, it serves only as a trigger for the transistors that power them. Each segment of the displays must flow around 25 mA (to be able to achieve the 3500 µcd – needed luminosity). Afterwards, port B (bits 0–7) must drain the current of each segment (around 25 mA). Since port B can not drain that amount of current, transistors (triggered by port B) are the drainers.

Figure 3-3 shows the display circuit.

Figure 3-3. Display Circuit
3.5.2 IRQ Circuit

Pulses are received through the IRQ pin. These pulses are generated every time the wheel turns. However, to avoid noise that causes IRQ requests, a passive low-pass filter was added. It is important to note that the filter diminishes the amplitude of the signal (the higher the frequency the more it is diminished). Therefore, the filter must let through the signal produced by the taxi going at maximum speed. The cut frequency for this filter is around 106 Hz. Assuming that the perimeter of the wheel is 1 meter, the filter allows the car to go up to 380 Km/h.

Figure 3-4 shows the IRQ circuit.

---

**Figure 3-4** IRQ Circuit
### Table 3-2. Taximeter Bill of Materials

<table>
<thead>
<tr>
<th>Qty</th>
<th>Value</th>
<th>Device</th>
<th>Package</th>
<th>Description</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td>7SEG-CA</td>
<td>7SEG-13</td>
<td>7-segment display</td>
<td>DISP1 – DISP6</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>JP1E</td>
<td></td>
<td>Jumper</td>
<td>JP1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>JP2E</td>
<td></td>
<td>Jumper</td>
<td>JP4</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>JP5Q</td>
<td></td>
<td>Jumper</td>
<td>JP3_EXT, INTERFACE</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>JP8Q</td>
<td></td>
<td>Jumper</td>
<td>JP2</td>
</tr>
<tr>
<td>1</td>
<td>2.2 K</td>
<td>R-US_M1206</td>
<td>M1206</td>
<td>Resistor</td>
<td>R18</td>
</tr>
<tr>
<td>1</td>
<td>3 K</td>
<td>R-US_R1206</td>
<td>M1206</td>
<td>Resistor</td>
<td>R3</td>
</tr>
<tr>
<td>1</td>
<td>4.9152 MHz</td>
<td>MM39SL</td>
<td>MM39SL</td>
<td>Crystal</td>
<td>Y1</td>
</tr>
<tr>
<td>5</td>
<td>10 K</td>
<td>R-US_M1206</td>
<td>M1206</td>
<td>Resistor</td>
<td>R13 – R17</td>
</tr>
<tr>
<td>1</td>
<td>10 uF</td>
<td>C-USC2012</td>
<td>C2012</td>
<td>Capacitor</td>
<td>C5</td>
</tr>
<tr>
<td>1</td>
<td>15 K</td>
<td>R-US_M1206</td>
<td>M1206</td>
<td>Resistor</td>
<td>R4</td>
</tr>
<tr>
<td>2</td>
<td>22 pF</td>
<td>C-USC2012</td>
<td>C2012</td>
<td>Capacitor</td>
<td>C1,C2</td>
</tr>
<tr>
<td>1</td>
<td>100 K</td>
<td>R-US_M1206</td>
<td>M1206</td>
<td>Resistor</td>
<td>R2</td>
</tr>
<tr>
<td>1</td>
<td>100 nF</td>
<td>C-USC2012</td>
<td>C2012</td>
<td>Capacitor</td>
<td>C3</td>
</tr>
<tr>
<td>8</td>
<td>150</td>
<td>R-US_M1206</td>
<td>M1206</td>
<td>Resistor</td>
<td>R5 – R12</td>
</tr>
<tr>
<td>1</td>
<td>470 K</td>
<td>R-US_M1206</td>
<td>M1206</td>
<td>Resistor</td>
<td>R1</td>
</tr>
<tr>
<td>1</td>
<td>470 nF</td>
<td>C-USC2012</td>
<td>C2012</td>
<td>Capacitor</td>
<td>C6</td>
</tr>
<tr>
<td>1</td>
<td>470 uF</td>
<td>C-USC2012</td>
<td>C2012</td>
<td>Capacitor</td>
<td>C4</td>
</tr>
<tr>
<td>1</td>
<td>7805T</td>
<td></td>
<td>TO220H</td>
<td>Positive voltage regulator</td>
<td>IC2</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>HC908JL3DW</td>
<td>HC908JL3_SOIC8</td>
<td>Microcontroller</td>
<td>DW1</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>NPNSOT23</td>
<td>SOT-23</td>
<td>NPN transistor</td>
<td>Q0 – Q5</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>BCX71SMD</td>
<td>SOT23</td>
<td>PNP transistor</td>
<td>Q6 – Q13</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>PMLL4150</td>
<td>SOD80C</td>
<td>Diode</td>
<td>D1</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>TL1100E</td>
<td></td>
<td>Push button</td>
<td>SW1 – SW5</td>
</tr>
</tbody>
</table>
3.7 Layout

Figure 3-5 shows the taximeter layout.

![Taximeter Layout Diagram]
Section 4. Print Circuit Board (PCB) Design

4.1 Introduction

Since one of the hardware design goals for the taximeter was to be EMI proof, special attention was taken for the PCB design. Electronic devices carried in an automobile have to deal with many interference sources, such as radio frequency interference, electrostatic discharges, power line electricity, and magnetic fields, among many others. Therefore, automotive electronics must be designed in a careful way so they comply with such demands.

The taximeter PCB design follows IPC (Institute for Interconnecting and Packaging Electronic Circuits) regulations. Moreover, the design was enhanced to achieve better results.

4.2 PCB Layout

The top and bottom layers of the PCB are shown in this subsection.

4.2.1 Top Layer

Figure 4-1 shows the top layer of the taximeter PCB. The gerber files are available for download at:

Motorola (http://www.motorola.com) > Semiconductors > Microcontrollers > Reference Designs

Figure 4-1. Taximeter PCB (Top Layer)
4.2.2 Bottom Layer

Figure 4-2 shows the bottom layer of the taximeter PCB. The gerber files are available for download at:
Motorola (http://www.motorola.com) > Semiconductors > Microcontrollers > Reference Designs

![Figure 4-2. Taximeter PCB Bottom Layer](image)

4.3 Mechanical Characteristics

This subsection discusses the mechanical characteristics.

4.3.1 Conductor Widths

There are two classes of nets in this PCB design: the power type and the default type. The power type is used exclusively for supply lines (GND and V_{DD}) while the default is used for buses and signals in general. The characteristics are as follow:

<table>
<thead>
<tr>
<th>Class</th>
<th>Width</th>
<th>Clearance</th>
<th>Drill</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>10 mil</td>
<td>10 mil</td>
<td>28 mil</td>
</tr>
<tr>
<td>Power</td>
<td>20 mil</td>
<td>10 mil</td>
<td>28 mil</td>
</tr>
</tbody>
</table>

The line widths were calculated based on the values of current and temperature to be supported by the lines. The clearance field describes the minimum distance allowed between two traces. The drill measure indicates the drill’s diameter to be used when transferring a track from one layer to the other (vias or plated through-holes). These measures were based on standards by the IPC and were chosen in order to ease the manufacturer of the board by giving regular sizes.
4.3.2 Trace Angles

A source of RFI is an abrupt change of direction of a PCB track, which effectively looks like impedance discontinuities and will radiate accordingly. For HCMOS designs it is important to ensure that 90-degree track-direction changes do not occur. Also, from the mechanical point of view, a 90° angle is more likely to be detached from the board.

4.3.3 Trace Distances

Distances between lines and components were taken from IPC guidelines. Some measures include the following:

<table>
<thead>
<tr>
<th>PCB Parts</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copper/Dimension</td>
<td>30 mil</td>
</tr>
<tr>
<td>Drill/Hole/Copper</td>
<td>10 mil</td>
</tr>
<tr>
<td>Mask</td>
<td>1 mil</td>
</tr>
</tbody>
</table>

The distances between elements are very important because they protect signals in each trace. Spacing between traces in each layer should be maximized whenever possible.

4.3.4 Silk Screen Content

Following IPC specifications, it is recommended that all parts, pins, and connectors are properly labeled.

4.3.5 Layout

In order to make the production of the board faster, all elements were placed parallel or perpendicular to the border. This also helps optimize the flow of cooling air and presents an orderly appearance. The layout also shows six holes around the push buttons, these were placed there to support the extra pressure obtained when pressing the buttons. There are also tiny mounting holes, this was done to ease the orientations at assembly.

4.4 Noise-Reduction Characteristics

This subsection discusses the noise-reduction characteristics.

4.4.1 EMI-Reduction Measures

The current flow induces electromagnetic fields that can be the source of EMI. In order to decrease this source of noise, $V_{DD}$ and GND are traced parallel to each other. This type of tracing reduces (by cancellation) the EMI caused by the current.
4.4.2 Oscillator Circuit, MCU, and Power Supply

The most delicate parts of the PCB design are the oscillator circuit, the microcontroller, and the power supply. Different factors were taken into account for this part of the board. Important layout considerations include:

1. A balanced \( V_{DD} \) and GND directly underneath the microcontroller decreases noise that could affect the microcontroller.

2. The distance between the oscillator circuit and the microcontroller should be minimized. This reduces the risk of interference in the OSC tracks.
   a. Special attention is needed to have the lines from the crystal pins go directly to the OSC pins in the microcontroller without switching layers. Vias or through-holes, affect the frequency of the crystal.
   b. The oscillator circuit is one of the most sensitive parts of the design; therefore, isolation of the crystal and capacitors from the rest of the circuit is needed. No other lines or signals must flow near the OSC circuit.

3. A GND trace encircling the oscillator is needed. According to studies realized by Motorola Italy, a GND trace encircling the oscillator circuit minimizes the susceptibility to radiated interference.

4. Add a simple decoupling circuit made from a capacitor between \( V_{DD} \) and \( V_{SS} \), which can help reduce the risk of power perturbations near the microcontroller. That is why there is a capacitor right next to the power supply pins.

5. Supply the microcontroller with isolated traces for \( V_{DD} \) and \( V_{SS} \). This ensures that no current is taken from, or at the microcontroller by external sources. Thus, protecting the microcontroller from noise generated in other parts of the board.

Figure 4-3 shows the oscillator circuit and MCU.

![Figure 4-3. Taximeter Oscillator Circuit and MCU](image_url)
4.4.3 Separated Ground Traces

Separated GND traces protect different parts of the board. Basically, there are three different traces:

- One exclusively for the microcontroller and the oscillator circuit
- One for the LED displays
- The general one

The separations help control the noise in each sector. For instance, there is a net underneath the displays that absorbs the noise generated from the switching LEDs and that is drained by a single trace of GND with no contact with any other. This separation allows the draining to take place without disturbing the rest of the system.

Figure 4-4 shows the ground net underneath the LED displays.

![Figure 4-4. Ground Net Underneath the LED Displays](image)

The voltage regulator has a special area in the bottom layer. This area has two principal reasons:

- To help dissipate the heat produced by the regulator
- To absorb the noise created by the power supply

In the bottom layer, the traces were all done parallel to each other. This helps to cancel EMF created by the electron flow.
5.1 Introduction

The taximeter’s programmer allows the PC to communicate with the taximeter so that it’s data tables can be modified.

**NOTE:** This programmer DOES NOT allow overwriting of the software running on the microcontroller. It is used only to modified the data tables.

The microcontroller has one bidirectional serial communication port (pin PTB0). This port is used for both sending and receiving. The taximeter’s programmer is used to couple a PC’s serial port to the bidirectional serial communication port in the microcontroller.

The testing board allows the designer of taximeters to:

- Emulate the input received from the car
- Test the taximeter as if it was operating in a taxi

This board generates a square signal (frequency can be modified through a variable resistor) which emulates the pulses received each time the wheel turns.

5.2 Programmer and Testing Board Design

Since both boards (programmer and testing) are very simple boards, this reference design has merged them into a single board. The programmer section uses a MAX232 converter and 3-state non-inverting buffer. This design is the same as the one described in the monitor ROM (MON) section of the *MC68HC908JL3 Technical Data* (Motorola order number MC68HC908JL3/H). The testing section uses an a stable timer to emulate the pulses received from the wheel turns.

5.2.1 Schematic

*Figure 5-1* shows the programmer and testing board schematic. The schematic is available for download at:

Motorola ([http://www.motorola.com](http://www.motorola.com)) > Semiconductors > Microcontrollers > Reference Designs
5.2.2 Layout

Figure 5-2 shows the programmer and testing board layout.
5.2.3 Bill of Materials

Table 5-1 shows the programmer and testing board bill of materials.

<table>
<thead>
<tr>
<th>Qty.</th>
<th>Value</th>
<th>Device Description</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1N7776</td>
<td>Diode</td>
<td>D1</td>
</tr>
<tr>
<td>1</td>
<td>JP1Q</td>
<td>Jumper</td>
<td>JP1</td>
</tr>
<tr>
<td>1</td>
<td>JP2E</td>
<td>Jumper</td>
<td>JP2</td>
</tr>
<tr>
<td>1</td>
<td>M90HP</td>
<td>DB9 connector</td>
<td>DB9</td>
</tr>
<tr>
<td>5</td>
<td>.1 µF</td>
<td>C-EU025-024X044</td>
<td>C1-C5</td>
</tr>
<tr>
<td>1</td>
<td>1 µF</td>
<td>CPOL-USTAP5-50</td>
<td>C8</td>
</tr>
<tr>
<td>2</td>
<td>10 K</td>
<td>R-US_0207/12</td>
<td>R1-R2</td>
</tr>
<tr>
<td>1</td>
<td>10 nF</td>
<td>C-EU025-024X044</td>
<td>C7</td>
</tr>
<tr>
<td>1</td>
<td>10 µF</td>
<td>CPOL-USTAP5-50</td>
<td>C11</td>
</tr>
<tr>
<td>1</td>
<td>100 K</td>
<td>POTPOT</td>
<td>Pot</td>
</tr>
<tr>
<td>1</td>
<td>470 µF</td>
<td>CPOL-USTAP5-50</td>
<td>C10</td>
</tr>
<tr>
<td>1</td>
<td>7805T</td>
<td>Voltage Regulator</td>
<td>IC3</td>
</tr>
<tr>
<td>1</td>
<td>74125</td>
<td>3-state buffer</td>
<td>V1</td>
</tr>
<tr>
<td>1</td>
<td>LM555N</td>
<td>A stable timer</td>
<td>IC2</td>
</tr>
<tr>
<td>1</td>
<td>MAX232</td>
<td></td>
<td>IC1</td>
</tr>
</tbody>
</table>
5.2.4 PCB

**Figure 5-3** shows the programmer and testing PCB. The gerber file is available for download at:
Motorola ([http://www.motorola.com](http://www.motorola.com)) > Semiconductors > Microcontrollers > Reference Designs

![Figure 5-3. Programmer and Testing PCB](image)
Section 6. Software Design

6.1 Introduction
The purpose of a taximeter is to indicate the amount to be charged for the use of a taxi. This amount increases depending on several factors, such as time and distance. Moreover, the taximeter must be able to perform many other functions, such as displaying information, keeping statistical records, etc.

6.2 Software Design Goals
This software design pursues the following goals:

- Compliance — Compliance with the Mexican regulation NOM-007-SCFI-1997. This regulation establishes many functional and procedural requirements. See Section 2. System Requirements for further information.

- Modularity — The software must be completely modular and with as little cohesion as possible. This modularity should be reflected by the ease in making changes and adding new functionality.

- Small size — The limited amount of FLASH memory must be taken into account. In less than 4 Kb of memory, the software must be able to be fully functional.

- High-level programming — As much as possible, the software must be programmed in ‘C’, making future improvements easy and reducing development time.

6.3 General Software Architecture
In general, the software is divided into layers (see Figure 6-1) which are described following the figure.

<table>
<thead>
<tr>
<th>STATE MACHINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH PROGRAMMING API</td>
</tr>
<tr>
<td>HARDWARE</td>
</tr>
</tbody>
</table>

Figure 6-1. Software Architecture Layers
Taximeter API
Interacts with the hardware and is in charge of timing, ISR, and managing the hardware (displays, push buttons). It serves all upper layers. *(taxi_api.c)*

FLASH Programming API
Facilitates the FLASH tables programming. It interacts with the taximeter API and serves the state machine. *(taxi_flash.c)*

External Modules
Specific routines needed to control external devices through the expansion port, i.e., printers, GPS modules, RF modules, etc. *(external_ [name_of_module].c)*

State Machine
Event-driven functions that perform the high-level routines that give functionality. *(taxi_states.c)*.

### 6.4 State Machine Description

Each state has a specific function. The change from one state to another is triggered by push buttons or by time. The 11 states that build the state machine are:

1. FREE
2. IN-SERVICE
3. PAY
4. PROGRAM
5. EXTRAS
6. FARES
7. SPEEDOMETER
8. PULSES
9. CLOCK
10. INFO
11. OFF

*Table 6-1* describes what is done at each state of the state machine.
### Table 6-1. Description of States

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREE</td>
<td>Idle state</td>
</tr>
</tbody>
</table>
| IN-SERVICE | Enables any other function  
|          | Backups the distance traveled not in service  
|          | Displays the fare type selected  
|          | The initial charge and the extra charges are added to the trip cost  
|          | Displays the trip cost  
|          | The cost of the trip is incremented every given time or every given distance (whichever happens first)  |
| PAY     | Backups the distance traveled in service, total income, number of trips and number of cost increments  
|         | Toggles the display of the label "PAY" and the trip cost  |
| EXTRAS  | Displays label “EXTRA” for a given time  
|         | Lets choose the type of extra charge  |
| FARES   | Displays label “FARE” for a given time  
|         | Allows to choose the fare type  |
| CLOCK   | Displays the time  |
| SPEEDOMETER | Displays the speed  |
| PULSES  | Counts the number of wheel turns (used to verify that the wheel-turn indicator is working properly)  |
| INFO    | Display important information about the taximeter and its accumulators. The accumulators are displayed in the following sequence:  
|         | Total distance traveled  
|         | Total distance traveled in service  
|         | Total number of travels  
|         | Total number of cost increments  
|         | Total income  |
| PROGRAM | Displays label “PROGR” for a given time  
|         | Programs FLASH memory in order to change fares, labels, etc.  |
| OFF     | Displays label “OFF” for a given time  
|         | Turns displays off  |
6.5 State Machine Diagram

Figure 6-2 shows the state machine diagram

![State Machine Diagram]

- B1: Button 1
- B2: Button 2
- B3: Button 3
- B4: Button 4
- B5: Button 5

**Figure 6-2. State Machine Diagram**
6.6 Push Button Actions

Each of the five push buttons triggers a different action depending on the active state of the program. **Table 6-2** shows the actions triggered by each push button on each state.

<table>
<thead>
<tr>
<th>State</th>
<th>Button 1</th>
<th>Button 2</th>
<th>Button 3</th>
<th>Button 4</th>
<th>Button 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREE</td>
<td>SERVICE</td>
<td>FARES</td>
<td>CLOCK</td>
<td>INFO</td>
<td>PROGRAM</td>
</tr>
<tr>
<td>SERVICE</td>
<td>PAY</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>PAY</td>
<td>FREE</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>EXTRAS</td>
<td>—</td>
<td>FARES</td>
<td>extra ++</td>
<td>—</td>
<td>Select extra; FREE</td>
</tr>
<tr>
<td>FARES</td>
<td>—</td>
<td>FREE</td>
<td>fare++</td>
<td>—</td>
<td>Select fare; FREE</td>
</tr>
<tr>
<td>CLOCK</td>
<td>hours ++</td>
<td>minutes ++</td>
<td>SPEEDOMETER</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SPEEDOMETER</td>
<td>—</td>
<td>—</td>
<td>PULSES</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>PULSES</td>
<td>reset</td>
<td>freeze</td>
<td>FREE</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>INFO</td>
<td>—</td>
<td>—</td>
<td>Info++</td>
<td>FREE</td>
<td>—</td>
</tr>
<tr>
<td>PROGRAM</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>OFF</td>
<td>FREE</td>
<td>FREE</td>
<td>FREE</td>
<td>FREE</td>
<td>—</td>
</tr>
</tbody>
</table>

6.7 Memory Map

This subsection provides memory map information for both the FLASH memory and the random-access memory (RAM).

6.7.1 FLASH Memory

FLASH memory is divided into three basic sections:

1. Tables section — where modifiable data are stored.
2. Backup section — this area is a blank area (64 bytes long) used for global variables backup (stored in RAM) while FLASH reprogramming. This section is needed since the RAM area is used for FLASH reprogramming purposes.
3. Code section — area where all code is stored.

The *tables section* is further divided into three small subdivisions: FLASH tables, FLASH variables, and FLASH integrity-verification byte. The FLASH integrity-verification byte is checked at the beginning of the program to ensure integrity of FLASH tables and variables. When FLASH tables and variables are erased, this byte is automatically erased (and must be written to its original value).
For the name, content, and exact location of each FLASH table and FLASH variable refer to file **A.4 taxi_tables.c**.

**Figure 6-3** shows the FLASH memory sections.

![FLASH Memory Sections](image)

**6.7.2 Random-Access Memory (RAM)**

RAM is divided into two sections: global variables section, and stack section. All local variables are stored in the stack. Global variables section has two areas: the area used during FLASH programming and the area not used during FLASH programming. FLASH programming needs to use the Global variables section from address 0x0088 to 0x00AB (refer to **FlashManager() documentation in taxi_flash.c**) . All data within this addresses is overwritten (but can be backed up previously). However, when FLASH programming is not taking place, this area is used for global variables. The area not used for FLASH programming is reserved to allocate variables that need to be read during FLASH programming (and that indicate the values that must be programmed).

**Figure 6-4** shows RAM sections.

For the name and type of each variable refer to file **A.12 taxi_variables.h**.
6.8 File Structure

The software file structure is explained in Table 6-3.

Table 6-3. Software File Structure

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>taximeter.c</td>
<td>Defines the main() function and the ISRs</td>
</tr>
<tr>
<td>taximeter.h</td>
<td>taximeter.c header file</td>
</tr>
<tr>
<td>taxi_tables.c</td>
<td>Initializes FLASH tables and variables</td>
</tr>
<tr>
<td>taxi_tables.h</td>
<td>Declares FLASH tables and variables</td>
</tr>
<tr>
<td>taxi_api.c</td>
<td>Defines the taximeter API</td>
</tr>
<tr>
<td>taxi_api.h</td>
<td>taxi_api.c header file</td>
</tr>
<tr>
<td>taxi_FLASH.c</td>
<td>Defines the FLASH-programming API</td>
</tr>
<tr>
<td>taxi_FLASH.h</td>
<td>taxi_FLASH.c header file</td>
</tr>
<tr>
<td>taxi{[module_name].c</td>
<td>Defines the specific external module API</td>
</tr>
<tr>
<td>taxi{[module_name].h</td>
<td>taxi{[external_module_name].c header file</td>
</tr>
<tr>
<td>taxi_states.c</td>
<td>Defines the State Machine states</td>
</tr>
<tr>
<td>taxi_states.h</td>
<td>taxi_states.c header file</td>
</tr>
<tr>
<td>taxi_variables.h</td>
<td>Declares all global variables stored in RAM</td>
</tr>
<tr>
<td>taxi_tables.h</td>
<td>Declares FLASH tables and FLASH variables</td>
</tr>
<tr>
<td>taxi_messages.h</td>
<td>Defines all the alphanumeric messages that are to be displayed</td>
</tr>
<tr>
<td>taxi_config.h</td>
<td>Defines the taximeter global configuration</td>
</tr>
<tr>
<td>taximeter.prm</td>
<td>Defines the sections into which the RAM and FLASH are divided</td>
</tr>
</tbody>
</table>
6.9 Naming Conventions

The project follows the naming conventions given in Table 6-4.

Table 6-4. Naming Conventions

<table>
<thead>
<tr>
<th>Structure</th>
<th>Convention</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macros</td>
<td>All macros are written in all UPPER CASE.</td>
<td>#Define DAY 0</td>
</tr>
<tr>
<td>Functions</td>
<td>The first letter of each word of the function’s name is capitalized.</td>
<td>void CleanUp (void)</td>
</tr>
<tr>
<td>Global variables</td>
<td>All global variables are written in lower case. If the variable has more</td>
<td>Byte current_state</td>
</tr>
<tr>
<td></td>
<td>than one word, they are separated with underscores.</td>
<td></td>
</tr>
<tr>
<td>Variables and tables</td>
<td>All variables and tables located in FLASH are written in all upper case.</td>
<td>const BYTE INFO_TABLE[45]</td>
</tr>
<tr>
<td>located in FLASH</td>
<td>If the variable has more than one word, they are separated with underscores.</td>
<td></td>
</tr>
<tr>
<td>Local variables</td>
<td>All local variables are written the same way as global variables but</td>
<td>Byte _cancel_buttons.</td>
</tr>
<tr>
<td></td>
<td>preceded by an underscore.</td>
<td></td>
</tr>
<tr>
<td>Assembly labels</td>
<td>All assembly labels are written in all capital letters preceded by an</td>
<td>_RECEIVE_BYTE:</td>
</tr>
<tr>
<td></td>
<td>underscore.</td>
<td></td>
</tr>
</tbody>
</table>

6.10 Taximeter ISRs Description

This subsection describes the taximeter ISRs.

6.10.1 IRQ ISR

The IRQ is activated every time the wheel turns (in some cars it might be twice for each wheel turn). This interrupt is used to count the distance traveled while in service and the total distance traveled. When the taxi is in service, it also updates the amount to charge (every n number of interrupts). Finally this interrupt also increments a general-purpose IRQ counter.
6.10.2 Timer ISR

The timer interrupts every 1 millisecond (this is the frequency needed to correctly multiplex the displays). The timer interrupt service routine performs the following tasks:

1. When the taxi is in service, it updates the amount to charge (every \( n \) number of interrupts)
2. Increments a general-purpose time counter (used by taximeter API)
3. Every 30 interrupts it scans the push buttons
4. Enables one out-of-six displays and writes the data
5. Manages the clock

6.11 Taximeter API Description

The taximeter API provides service to upper layers. It interfaces with the taximeter hardware (displays and buttons) as well as with the microcontroller modules (TIM).

6.11.1 Delay

The delay function creates a pause in the software. This pause can be configured to be interruptible by the push buttons.

Prototype: \( \text{void Delay(Word } _{ms}, \text{ Byte } _{cancel\_buttons} ) \)

Parameters: 
- \(_{ms}\) — number of milliseconds the pause lasts.
- \(_{cancel\_buttons}\) — buttons that might cause the pause to end even if time has not ran out

Remarks: The taximeter displays are refreshed even when the software is paused.

6.11.2 WaitForButtonsRelease

The WaitForButtonsRelease function pauses the software until all buttons are released.

Prototype: \( \text{void WaitForButtonsRelease(void) } \)

Parameters: none

Remarks: Even when this function is a simple variation of \( \text{Delay} \), it is profitable to have because it appears many times throughout the code and it uses less stack and less overhead in the call.
6.11.3 DisplayMsg

Sends display strings to the taximeter. This function is used to display alphanumeric messages to the user.

Prototype: void DisplayMsg (char * _str)
Parameters: _str - pointer to the string to be displayed.
Remarks: Only the first five characters of * _str will be displayed. Since this function uses tables to convert form ASCII to 7-segment-display format, only capital letters can be used as parameters.

6.11.4 DisplayNum

Sends number displays to the taximeter. This function is used to display numeric messages (or amounts) to the user.

Prototype: void DisplayNum (ulong _num)
Parameters: _num - number to be displayed.
Remarks: _num must be a 5 digit number, where the last two digits represent decimal values. For example: if _num = 12345, the number displayed will be 123.45.
If the number is larger than 5 digits long, it might not be displayed correctly.
The use of this function needs to be carefully determined. Due to the large amount of stack it uses it is not recommended to use it from more than two functions deep; it may overflow the stack.

6.12 Software Modification Procedures

Before making any modifications to the software there are certain general issues that must be kept in mind.

1. **RAM free space for global variables** — Only 3 bytes that can be used for declaring global variables (at addresses 0xAA – 0xAC). As described in 6.7 Memory Map, from address 0xAD to 0xAF is used for specific variables needed in FLASH programming. If more global variables are declared it exists the risk that the stack overwrites them.

2. **Stack size** — Care with the stack size should be taken when adding new functions. It is important to remember that stack area is shared with RAM. If stack is over-used, RAM might be overwritten.

3. **FLASH free space** — Even when the code may be far from reaching 4 Kb of memory, it is important to remember that the 4 Kb is not used for code alone. For FLASH-programming purposes (refer to 6.7 Memory Map) 256 bytes of FLASH are reserved.
If more stack, RAM and/or FLASH memory are needed, the microcontroller can be replaced by the MC68HCHC908JL8. The MC68HCHC908JL8 is pin-to-pin compatible with the MC68HC908JL3 but in has 8 KB of FLASH and 256 bytes of RAM as opposed to 4KB of FLASH and 128 bytes of RAM.

6.12.1 How to Add an External Module

Adding an external module implies two basic actions:
1. Create an API or driver to handle the external module (if needed)
2. Create a new state in the state machine

6.12.2 How to Create a New State

To create a new state follow this procedure:
1. Define a name for the new state. Example, PRINT.
2. Define the previous and next states (which state is before the new state and which is next). Example, previous state = SERVICE; next state = PAY.
3. Define the action that will trigger the change to the new state. Example, Button 1.
4. Add a case statement in main function at taxieter.c. Example, CASE PRINT: Print();
5. Modify the previous state so that when the appropriate action happens it leads to the new state. Example, Inside SERVICE – current_state = PRINT;
6. Create the body of the new state. Example, void Print () {…}
7. At the end of the new state, modify variable current_state so it leads to the next state(s). Example, current_state = PAY.

6.12.3 How to Display Scrolling Messages

Refer to state Info() in the code (taxieter.c — A.10 taxieter.c).
6.12.4 How to Add FLASH Tables and/or Variables

FLASH tables are declared in `taxi_tables.h`. In order to add a FLASH table open this file and declare a new table. There are two types of tables: tables used for static values (such as table `TRANSLATE_NUMBER`) and tables used for dynamic values that can be changed through FLASH reprogramming (such as `EXTRA_TABLE`). If the new table is static no further action is needed; otherwise, the following actions must be taken:

1. Verify that the new table doesn’t exceed the 128 bytes reserved for FLASH tables. If the space is not enough, the `taximeter.prm` file should be modified. Within this file, the section FLASH_VARIABLES can be removed and FLASH_TABLES incremented in size. If this action is taken, the FLASH variables must be relocated (`taxi_tables.c`) and the functions that deal with them must be updated (such as `BackupAccumulators`).

2. Verify that the new table doesn’t overlap with FLASH variables. If it does, change the location of FLASH variables and mirror this change in `BackupAccumulators()` function. Refer to `BackupAccumulators()` information in file `taxi_flash.h`.

3. Declare the new table after all existing tables.

To add a FLASH variable just declare it in `taxi_tables.h` inside the subdivision: FLASH_VARIABLES.

6.12.5 How to Change FLASH Programming Routines

FLASH programming routines are very delicate. A complete understanding of programming resident ROM routines is needed (please read the application note entitled Using MC68HC908On-Chip FLASH Programming Routines, Motorola document order number AN1831/D).

FLASH programming routines use part of the RAM as a buffer to store the data that is to be programmed into FLASH. Therefore, this section in RAM must be backed up (and restored later) or it will be overwritten (refer to 6.7 Memory Map). The backing up (and restoring) procedure has two phases:

1. Manually backing up the first four bytes into stack
2. Automatically backing up the rest into FLASH

After setting up RAM, the FLASH programming routine might be used for anything that is wanted. If FLASH programming is used to modified FLASH tables and/or FLASH variables there are certain considerations to take:

1. Verify the location of tables and variables.
2. If needed, mirror their location in RAM to their location in FLASH.
3. Remember that interrupts are disabled after using FLASH programming routines.
Before the FLASH programming routine is finished, the FLASH integrity-verification byte must be set and RAM must be restored from backup.

6.12.6 How to Translate the Taximeter to Another Language

All alphanumeric messages are stored in the file `taxi_messages.h`. To translate the taximeter to another language (it currently supports Spanish, English, French, and Portuguese) open `taxi_messages.h` and define the new language. Then, open file `taxi_config.h` and define the new language as the active language.

6.12.7 How to Program the Entire FLASH

Connector MON08 is included in hardware because any programmer interface with a MON08 connector can program the taximeter. For example, an in-circuit programmer can be connected to the taximeter through the MON08 connector. Once this is done, the MCU is programmed as if it were on the in-circuit programmer.

This procedure is done when the code of the taximeter must be changed. If only the data tables need to be updated, the programming and testing board must be used. Refer to Section 4, Print Circuit Board (PCB) Design.
Section 7. PC Interface

7.1 Introduction

A PC Interface is what re-programs the taximeter data tables. This interface must follow the communication protocol and must compile with the taximeter specifications for transferring data.

7.2 Communication Protocol

The PC communicates serially with the taximeter. The communication speed is 4800 bps. The protocol used is NRZ with 8 data bits, no parity, and one stop bit (4800 8-N-1). The communication happens as if the MCU was in monitor mode.

7.3 Specifications

The taximeter has five re-programmable tables and eight re-programmable variables (refer to file A.4 taxi_tables.c for detailed information). The tables and variables are shown in Table 7-1. The PC interface must send all this information in order starting with address 0xEC00.

<table>
<thead>
<tr>
<th>Name</th>
<th>Size in Bytes</th>
<th>Address Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>FARE_MESSAGE_TABLE</td>
<td>45</td>
<td>0xEC00–0xEC2C</td>
</tr>
<tr>
<td>INL_CH_TABLE</td>
<td>18</td>
<td>0xEC2D–0xEC3E</td>
</tr>
<tr>
<td>FARE_TABLE</td>
<td>18</td>
<td>0xEC3F–0xEC50</td>
</tr>
<tr>
<td>EXTRA_TABLE</td>
<td>18</td>
<td>0xEC51–0xEC62</td>
</tr>
<tr>
<td>INFO_TABLE</td>
<td>24</td>
<td>0xEC63–0xEC7A</td>
</tr>
<tr>
<td>empty space</td>
<td>6</td>
<td>0xEC7B–0xEC80</td>
</tr>
<tr>
<td>TOTAL_DISTANCE_TRAVELED</td>
<td>4</td>
<td>0xEC81–0xEC84</td>
</tr>
<tr>
<td>TOTAL_DISTANCE_IN_SERVICE</td>
<td>4</td>
<td>0xEC85–0xEC88</td>
</tr>
<tr>
<td>TOTAL_NUMBER_TRAVELS</td>
<td>4</td>
<td>0xEC89–0xEC8C</td>
</tr>
<tr>
<td>TOTAL_NUMBER_INCREMENTS</td>
<td>4</td>
<td>0xEC8D–0xEC90</td>
</tr>
<tr>
<td>TOTAL_INCOME</td>
<td>5</td>
<td>0xEC91–0xEC94</td>
</tr>
<tr>
<td>empty space</td>
<td>14</td>
<td>0xEC95–0xEC9D</td>
</tr>
<tr>
<td>FARES_NUMBER</td>
<td>1</td>
<td>0xEC9E</td>
</tr>
<tr>
<td>EXTRAS_NUMBER</td>
<td>1</td>
<td>0xEC9F</td>
</tr>
<tr>
<td>empty space</td>
<td>31</td>
<td>0xECA0–0xECBE</td>
</tr>
<tr>
<td>FLASH_PROGRAM</td>
<td>1</td>
<td>0xECBF</td>
</tr>
</tbody>
</table>
If the programmer and testing board is being used, it is important to note that the board connects the serial connector transmit and receive (TD and RD). Therefore, whatever the PC sends to the taximeter will also reach the PC.

The communication with the taximeter uses frames. Each frame is 32 bytes long. Since 192 bytes (from 0xEC00 to 0xECBF) must be programmed, six frames are needed.

The taximeter performs the tables re-programming at state PROGRAM. When state PROGRAM is called (by pressing button 5 at state FREE) it shows the label “PROGR” for 1 second. After that, all the displays are turned off. The turning off of the displays announces the taximeter is ready to receive data.

To start communicating the PC must send one byte to the taximeter (this is the only time a byte isn’t part of a frame). The taximeter will stay on hold until this byte has been sent. This first byte must be 0xAA, which indicates that the communication has begun. After receiving 0xAA, the taximeter will send a frame (32 bytes) back to the PC. This frame contains the bytes stored from address 0xEC81 to 0xEC9F (the accumulators). The accumulators are sent to the PC interface before being erased so the PC decides whether it sends them back to the taximeter (to keep accumulating) or it resets them. It is important to remember that if these accumulators exceed 99,999 counts, they won’t be displayed correctly by the taximeter (refer to 6.11 Taximeter API Description); however, they can hold up to 4,294'967,295 before overflowing.

**NOTE:** The accumulators are stored as the value minus one. For example, if five travels have been done, NUMBER_TRAVELS will be 0x00000004; if no travels have been done, NUMBER_TRAVELS will be 0xFFFFFFFF.

After sending the accumulators, the taximeter expects to receive the first information frame (for addresses 0xEC00-0xEC1F). Once received, the first frame is programmed in FLASH and echoed back to the PC for verification purposes. Then the PC sends the second information frame and so on with the 192 bytes (from address 0xEC00 to address 0xECBF). After each frame is received, it is programmed in FLASH and echoed back. The last byte sent to the taximeter is the FLASH_PROGRAM. This byte indicates if the FLASH is properly programmed or not. This byte must be set to 0xAA; otherwise, the taximeter will display “ERROR” and it will have to be programmed again.

**Figure 7-1** shows how the information exchange takes place between the PC and the taximeter.

**Appendix B. PC Interface Source Code** shows software written in C which implements a PC interface for the taximeter.
Figure 7-1. Information Exchange
Appendix A. Taximeter Source Code

A.1 Introduction

This appendix contains the following source code:

- taxi.c
- taxi.h
- taxi_tables.c
- taxi_tables.h
- taxi_api.c
- taxi_api.h
- taxi_flash.c
- taxi_flash.h
- taxi_states.c
- taxi_states.h
- taxi_variables.h
- taxi_messages.h
- taxi_config.h
- taxi.prm
A.2 taximeter.c

/*================================================================*/
/**
* Copyright (c) 2002, Motorola Inc.
* Motorola Reference Design
* File name : taximeter.c
* Project name : Taximeter Reference Design
* Author : Mauricio Capistran-Garza
* Department : Guadalajara - SPS
* Description : This is the core of the state machine. Each
* state, before ending, modifies the global
* variable current_state. Then, in this file
* according to current_state, the next state is
* selected and executed.
* The ISRs are also defined in this file.
* *
* History :
*/
/*================================================================*/
#define __TAXIMETER_C__
#include "taximeter.h"

/***********************************************************************/
INTERRUPT SERVICE ROUTINES - ISRs
/***********************************************************************/
/**
* Irq: + Every wheel turn the external interrupt is requested.
* It increases the distance accumulator (accumulator_pulses)
* and, if the Taxi is in service and a given distance
* (inc_distance) has been traveled, the amount to charge
* is updated.
* + It also increments a general purpose pulse counter.
* + Debouncing of the IRQ might be needed.
* *
* Return: void
*/
interrupt 2 void Irq (void) { // Every wheel turn.
    ISCR |= ACK1; // Acknowledge the interrupt
    if (flag_charge == CHARGE) { // If it's in service...
        accumulator_pulses++;
        if (accumulator_pulses >= inc_distance) { // Compare the distance traveled so far
            accumulator_pulses = 0; // distance before an increment
            accumulator_time = 0; // Reset accumulators
            accumulator_trip += fare_active; // Update the amount to charge
            accumulator_inc++;
        }
    }
}
accumulator_distance_traveled++;  

counter_pulses++; // General purpose pulse counter
}

/**
 * Timer: + Every 1 ms a timer interrupt is requested.
 * + It increases a time accumulator (accumulator_time)
 * and, if the Taxi is in service and a given
time (inc_time) has gone by, the amount to charge is updated.
 * + It increments a general purpose time counter.
 * + Every 30 interrupts (30 ms) the buttons are scanned
 * and stored into pushed_buttons. No debouncing is needed, since the 30 ms interval is enough.
 * + Enables one out-of-six displays and sends its data. (Multiplexing).
 * + Updates the variables needed by the clock.
 * Return: void
 */
interrupt 6 void Timer(void) {
    /* Every 1 ms one display is turned on. Every 30 ms the buttons are read. */
    TASC=TASC; // Clean interrupt flag
    // Lectura de TASC para limpiar la bandera de overflow del timer
    TASC &= TOF;
    PTB = DISPLAY_OFF; // Turn off the display currently on
    arr_display_index++; // Select the next display
    display_en = (display_en << 1); // Enable the next display
    if(arr_display_index >= 6) {
        read_buttons_time++;
        if(read_buttons_time == 5) { // Read buttons every 30 ms (6 displays * 5
read_buttons_time)
            read_buttons_time = 0;
            PTAPUE = 0x3F; // Enable internal pull-up resistors in port A
            DDRA = 0x00; // Configure port A as input
            pushed_buttons = PTA;
            pushed_buttons &= 0x1F;
            DDRA = 0xFF; // Configure port A as outputs (pull-up resistors are automatically disabled)
        }
        arr_display_index = 0; // Turn on display 0
        display_en = 0x01;
    }
    PTA = display_en;
    PTB = arr_display[arr_display_index];
    if (flag_charge == CHARGE) { // Compare the time elapsed so far with the max.
        accumulator_time++; // time before an increment to the total charge
        if (accumulator_time >= inc_time) {
            accumulator_time = 0;
            accumulator_pulses = 0;
            accumulator_trip += fare_active;
            accumulator_inc++; // There has been an increment
        }
    }
}
counter_ms++;                     // General purpose time counter

clock_ms ++;                      // Clock counter updates
if (clock_ms == 60000)            // 60,000 ms = 1 min.
    clock_ms = 0;
    clock_minutes++;
    if (clock_minutes == 60) {
        clock_minutes = 0;
        clock_hours++;
        if (clock_hours == 24) {
            clock_hours = 0;
        }
    }

/************************************************************
** MAIN **
************************************************************/

void main (void){
    _asm rsp;
    Initialize();                      // Configure system

    while(-1)                           // State Machine
    {
        switch(current_state)          // current_state is changed in each state
        {
            case FREE:        Free();
                break;
            case SERVICE:     Service();
                break;
            case PAY:         Pay();
                break;
            case EXTRAS:      Extras();
                break;
            case FARES:       Fares();
                break;
            case CLOCK:       Clock();
                break;
            case SPEEDOMETER: Speedometer();
                break;
            case PULSES:      Pulses();
                break;
            case INFO:        Info();
                break;
            case PROGRAM:     Program();
                break;
            case OFF:         Off();
                break;
            default:          current_state = FREE;
                break;
        }
    }
}
A.3 taximeter.h

;/*----------------------------------------------------------------------*/
/**
 * Copyright (c) 2002, Motorola Inc.
 * Motorola Reference Design
 * File name : taximeter.h
 * Project name : Taximeter Reference Design
 * Author : Mauricio Capistran-Garza
 * Department : Guadalajara - SPS
 * Description : main header file. It includes all other header
 *                files needed.
 * History :
 */
;/*----------------------------------------------------------------------*/

#ifndef __TAXIMETER_H__
#define __TAXIMETER_H__
#define __TAXIMETER_C__
#endif

#include <hidef.h>
#include "jl3.h"
#include "taxi_config.h"
#include "taxi_tables.h"
#include "taxi_messages.h"
#include "taxi_variables.h"
#include "taxi_api.h"
#include "taxi_flash.h"
#include "taxi_states.h"
#endif // __TAXIMETER_H__
A.4 taxi_tables.c

/*================================================================*/
/**
* Copyright (c) 2002, Motorola Inc.
* Motorola Reference Design
*
* File name    : taxi_tables.h
* Project name : Taximeter Reference Design
* Author       : Mauricio Capistran-Garza
* Department   : Guadalajara - SPS
*
* Description : Declares the tables allocated in flash. Some of
*                these tables are re-programmed in state Program.
*
* History      :
*/
/*================================================================*/

#include "taximeter.h"

//====================================================
//====================   TABLES   ====================
//====================================================

const Byte TRANSLATE_NUMBER[10] =    {    // Translates a number to its 8-segment-display
  // numbers from 0 to 9
  // PTB = G F A B P C D E
  // PTB = 7 6 5 4 3 2 1 0
  SEG_A & SEG_B & SEG_C & SEG_D & SEG_E & SEG_F,              // 0
  SEG_B & SEG_C,                                            // 1
  SEG_A & SEG_B & SEG_D & SEG_E & SEG_G,                      // 2
  SEG_A & SEG_B & SEG_C & SEG_D & SEG_G,                      // 3
  SEG_B & SEG_C & SEG_F & SEG_G,                            // 4
  SEG_A & SEG_C & SEG_D & SEG_F & SEG_G,                       // 5
  SEG_A & SEG_B & SEG_C & SEG_D & SEG_E & SEG_F & SEG_G,      // 6
  SEG_A & SEG_B & SEG_C,                                      // 7
  SEG_A & SEG_B & SEG_C & SEG_D & SEG_E & SEG_F & SEG_G,      // 8
  SEG_A & SEG_B & SEG_C,                                       // 9
};

const Byte TRANSLATE_LETTER[59] = {
  // Valid letters: A,b,Cc,d,E,F,G,Hh,Ii,Jj,Ll,n,Oo,P,q,r,S,Uu,0,1,2,3,4,5,6,7,8,9,-,. 
  // PTB = G F A B P C D E
  // PTB = 7 6 5 4 3 2 1 0
  DISPLAY_OFF,      // space         32
  DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF, // 33-37
  DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF, // 38-42
  DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF, // 43-44
  SEG_G,           // -            45
  SEG_P,           // .            46
  DISPLAY_OFF,DISPLAY_OFF, // 47
  SEG_A & SEG_B & SEG_C & SEG_D & SEG_E & SEG_F, // 0 48
  SEG_B & SEG_C, // 1 49
};
Taximeter Reference Design Using the MC68HC908JL3

```
SEG_A & SEG_B & SEG_G & SEG_F & SEG_D,        // 2        50
SEG_A & SEG_B & SEG_G & SEG_C & SEG_D,        // 3        51
SEG_F & SEG_G & SEG_B & SEG_C,                // 4        52
SEG_A & SEG_F & SEG_E & SEG_C & SEG_D & SEG_G, // 5        53
SEG_A & SEG_B & SEG_C,                       // 6        54
SEG_A & SEG_B & SEG_C & SEG_D & SEG_E & SEG_F & SEG_G, // 7        55
SEG_A & SEG_B & SEG_C & SEG_D & SEG_E & SEG_F & SEG_G, // 8        56
SEG_A & SEG_B & SEG_C & SEG_D & SEG_E & SEG_F & SEG_G, // 9        56
DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,    //    57-61
SEG_A & SEG_B & SEG_C,                                    // 7        55
SEG_A & SEG_B & SEG_C & SEG_D & SEG_E & SEG_F & SEG_G,    // 8        56
SEG_A & SEG_B & SEG_C & SEG_D & SEG_E & SEG_F & SEG_G,    // 9        56
DISPLAY_OFF,DISPLAY_OFF, // 57-61
SEG_F & SEG_E & SEG_D,                                    // 8        56
SEG_F & SEG_E & SEG_D & SEG_C & SEG_G,                     // G        70
SEG_F & SEG_E & SEG_B & SEG_C & SEG_G,                      // H        71
SEG_B & SEG_C,                                            // I        72
SEG_B & SEG_C & SEG_G & SEG_E,                                      // J        73
DISPLAY_OFF,                                              // K        74
SEG_F & SEG_E & SEG_D,                                    // L        75
SEG_E & SEG_A & SEG_C & SEG_B & SEG_F,                      // M        76
SEG_E & SEG_G & SEG_C,                                   // N        77
SEG_A & SEG_B & SEG_C & SEG_D & SEG_E & SEG_F,              // O        78
SEG_E & SEG_F & SEG_A & SEG_B & SEG_G,                      // P        79
SEG_A & SEG_B & SEG_G & SEG_F & SEG_C,                      // Q        80
SEG_E & SEG_G,                                            // R        81
SEG_A & SEG_F & SEG_G & SEG_C & SEG_D,                      // S        82
SEG_A & SEG_B & SEG_C,                                   // T        83
SEG_F & SEG_E & SEG_D & SEG_C & SEG_B,                      // U        84
SEG_F & SEG_E & SEG_D & SEG_C & SEG_B,                      // V        85
DISPLAY_OFF, DISPLAY_OFF, // W-X        86-87,
SEG_B & SEG_C & SEG_F & SEG_G,                                      // Y        88
SEG_A & SEG_B & SEG_D & SEG_E & SEG_G,                      // Z        89
};

//====================================================
//=============  REPROGRAMMABLE TABLES  ==============
//====================================================
#pragma CONST_SEG FLASH_TABLES

volatile const Byte FARE_MESSAGE_TABLE[45] = {
  ' ', 'D', 'A', 'Y', ' ',
  'N', 'I', 'G', 'H', 'T',
  'S', 'U', 'N', 'D', 'A',
  'T', 'I', 'M', 'E', ' ',
  'F', 'A', 'R', 'E', '5',
  'F', 'A', 'R', 'E', '6',
  'F', 'A', 'R', 'E', '7',
  'F', 'A', 'R', 'E', '8',
  'F', 'A', 'R', 'E', '9',
};
```

---

Freescale Semiconductor, Inc.

For More Information On This Product, Go to: www.freescale.com

---

MOTOROLA

Taximeter Source Code

Taximeter Reference Design Using the MC68HC908JL3

DRM053
volatile const Word INI_CH_TABLE[9] = { // Price of initial charges
600,
1200,
1000,
3500,
500,
600,
700,
800,
900,
};

volatile const Word FARE_TABLE[9] = { // Price of every fare
150,
300,
200,
100,
50,
60,
70,
80,
90,
};

volatile const Word EXTRA_TABLE[9] = { // Price of extra charges
1000,
2000,
3000,
4000,
5000,
6000,
7000,
8000,
9000,
};

volatile const Byte INFO_TABLE[24] = {
'S','E','R','I','A','L',' ',' ', // Serial number
'P','L','A','T','E','S',' ',' ', // License plate
'M','O','T','O','R','O','L','A', // Advertiser
};

//====================================================
//================  FLASH VARIABLES  =================
//====================================================
#pragma CONST_SEG FLASH_VARIABLES

/*
 * Accumulators in flash.
 * These accumulators are transmitted to the programmer at the
 * beginning of the programming process and are erased. If you
 * wish to keep these in the taximeter, the programmer should
 * send them back so they are re-programmed. All accumulators
 * are incremented in the state PAY, but the TOTAL_DISTANCE_
 * TRAVELED is also incremented in the state OFF.
 */
* To be modified, these accumulators are read into RAM
  * <BackupAccumulator()> starting at address 0x008C. There
  * are global variables defined in that space. Those global
  * variables are incremented and the they are re-programmed
  * into flash.
  */

ulong TOTAL_DISTANCE_TRAVELED @0xEC81;
ulong TOTAL_DISTANCE_IN_SERVICE @0xEC85;
ulong TOTAL_NUMBER_TRAVELS @0xEC89;
ulong TOTAL_NUMBER_INCREMENTS @0xEC8D;
ulong TOTAL_INCOME @0xEC91;

volatile const Byte FARES_NUMBER @ 0xEC9E = 2; // Number of active fares
volatile const Byte EXTRAS_NUMBER @ 0xEC9F = 5; // Number of active extras

#pragma CONST_SEG FLASH_VERIFY
volatile const Byte FLASH_PROGRAM = UNCORRUPT_FLASH; // Key to verify flash programation
#pragma DEFAULT_SEG
A.5 taxi_tables.h

/*================================================================*/
/**
* Copyright (c) 2002, Motorola Inc.
* Motorola Reference Design
*
* File name : taxi_tables.h
* Project name : Taximeter Reference Design
* Author : Mauricio Capistran-Garza
* Department : Guadalajara - SPS
*
* Description : Declares the tables allocated in flash. Some of
* these tables are re-programmed in state Program.
*
* History :
*/
/*================================================================*/
#ifndef _TAXI_TABLES_H__
#define _TAXI_TABLES_H__

//====================================================
//====================   TABLES   ====================
//====================================================
extern const Byte TRANSLATE_NUMBER[];
extern const Byte TRANSLATE_LETTER[];

//====================================================
//=============  REPROGRAMMABLE TABLES  ==============
//====================================================
extern volatile const Byte FARE_MESSAGE_TABLE[];
extern volatile const Word INI_CH_TABLE[];
extern volatile const Word FARE_TABLE[];
extern volatile const Word EXTRA_TABLE[];
extern volatile const Byte INFO_TABLE[];

//====================================================
//================  FLASH VARIABLES  =================
//====================================================

/*
* Accumulators in flash.
* These accumulators are transmitted to the programmer at the
* beginning of the programming process and are erased. If you
* wish to keep these in the taximeter, the programmer should
* send them back so they are re-programmed. All accumulators
* are incremented in the state PAY, but the TOTAL_DISTANCE_
* TRAVELED is also incremented in the state OFF.
* To be modified, these accumulators are read into RAM
* <BackupAccumulator()> starting at address 0x008C. There
* are global variables defined in that space. Those global
* variables are incremented and they are re-programmed
* into flash.
*/

extern ulong TOTAL_DISTANCE_TRAVELED @0xEC81;
extern ulong TOTAL_DISTANCE_IN_SERVICE @0xEC85;
extern ulong TOTAL_NUMBER_TRAVELS @0xEC89;
extern ulong TOTAL_NUMBER_INCREMENTS @0xEC8D;
extern ulong TOTAL_INCOME @0xEC91;

extern volatile const Byte FARES_NUMBER @ 0xEC9E; // Number of active fares
extern volatile const Byte EXTRAS_NUMBER @ 0xEC9F; // Number of active extras

extern volatile const Byte FLASHPROGRAM; // Key to verify flash programation

#else // _TAXI_TABLES_H__
A.6 taxi_api.c

/*================================================================----------*/
/**
* Copyright (c) 2002, Motorola Inc.
* Motorola Reference Design
*
* File name : taxi_API.c
* Project name : Taximeter Reference Design
* Author : Mauricio Capistran-Garza
* Department : Guadalajara - SPS
*
* Description : This is the definition of the taximeter API
*
* History :
*/
/*================================================================----------*/

#include "taximeter.h"

/******************************
API - Initialize
*******************************/

/**
* Initialize: set all global variables, set up timers and
* irq, configure system.
*
* Parameters: none.
*
* Variables read: none.
*
* Variables modified: none
*
* Subfunctions: none.
*
* Return: void
*/

void Initialize () {

  current_state = FREE;          // Start in FREE state (Idle State)

  /* Displays related variables */
  PTA = 0x00;                    // Display enabler (none enabled)
  DDRA = 0xFF;                   // Port A as output
  PTB = DISPLAY_OFF;             // Displays off
  DDRB = 0xFF;                   // Port B as output

  read_buttons_time = 0;         // Counter that indicates when to read buttons
  arr_display_index = 0;         // Display 0 selected; pointer to arr_diplay[]
  display_en = 0x01;             // Enable display 0
  pushed_buttons = NO_BUTTON;    // Button(s) pushed

  /* Fares related variables */
  fare_type = DAY;               // Default fare = DAY
  fare_active = FARE_TABLE[FARE_DAY]; // Active fare (according to FARE_TABLE)
  arr_display[0] = TRANSLATE_NUMBER[DAY+1]; // Active fares goes to Display 0
  ini_ch_active = INI_CH_TABLE[INI_CH_DAY]; // Initial charge (according to INI_CH_TABLE)
}
/* Counters & Accumulators */
  counter_ms = 0;                // Reset counters
  counter_pulses = 0;
  accumulator_pulses = 0;        // Reset accumulators
  accumulator_time = 0;
  accumulator_trip = 0;
  accumulator_inc = 0;
  accumulator_distance_traveled = 0;
  accumulator_extras = 0;
  clock_ms = 0;
  clock_minutes = 0;
  clock_hours = 12;              // Start at 12:00

/* Incremental-charge related variables */
  flag_charge = NO_CHARGE;
  inc_time = 5000;   // Default time for incrementing accumulator_trip = 30 segs
  inc_distance = 45; // Default distance for incrementing accumulator_trip = 100 mts

/* System configuration */
  CONFIG2 = 0x80;    //    10000000b;
                   //    ||||||||_________ Reserved
                   //    |||||____________ LVIT
                   //    ||| _____________ Reserved
                   //    |________________ IRQ Internal Pull-Up disconnected
  CONFIG1 = 0x11;    //    00010001b;
                   //    ||||||||_________ COP disabled
                   //    |||||||__________ STOP instruction treated as illegal opcode
                   //    ||||||___________ Short Stop Recovery Bit
                   //    |||||____________ Reserved
                   //    ||||_____________ Low Voltage Inhibit enabled
                   //    |||______________ Reserved
                   //    |________________ COP reset period
  ISCR = 0x00;       //    00000000b;
                   //    ||||||||_________ MODE1 (falling edges only)
                   //    |||||||__________ IMASK (Interrupt Enabled)
                   //    ||||||___________ IRQ Acknowledge (Write Only)
                   //    |||______________ IRQ Flag (Read Only)
                   //    ||_______________ Unimplemented
  TAMODH = 0x04;     // 0x04CC = 1228 counts/ms; Interrupt every 1 ms
  TAMODL = 0xCD;
  TASC = 0x40;       //    01000000b;
                   //    |||||___________ Prescaler: Internal Bus Clock 1.2288 MHz /
                   //    |||||___________ 1 = 1229 counts; error = 0.58 segs / hour.
                   //    |||______________ Unimplemented
                   //    |________________ TIM Reset Bit = 0 -> No effect
                   //    |________________ TIM Stop Bit: Counter active
                   //    |________________ TIM Overflow interrupts enabled
                   //    |________________ TIM Overflow Flag

EnableInterrupts;
return;
}
/*******************************************************
API - CleanUp
*******************************************************/
/**
* CleanUp: Reset variables needed to return to idle state
*
* Parameters: none.
*
* Variables read: none.
*
* Variables modified: accumulator_extras.
* accumulator_trip.
* accumulator_pulses.
* accumulator_time.
* accumulator_distance_traveled.
* accumulator_inc.
*
* Subfunctions: none.
*
* Return: void
*/
void CleanUp(void) {
    accumulator_extras = 0;
    accumulator_trip = 0;
    accumulator_pulses = 0;
    accumulator_time = 0;
    accumulator_distance_traveled = 0;
    accumulator_inc = 0;
    return;
}

/************************************************************
API - Delay
*******************************************************/
/**
* Delay: Waits _ms milliseconds or until one of
*        _cancel_buttons is pressed.
*
* Parameters: _ms - number of milliseconds to
*              wait before ending function.
*              _cancel_buttons - buttons that might cause
*              the function to end even if
*              time has not finished.
*
* Variables read: pushed_buttons.
*
* Variables modified: counter_ms.
*
* Subfunctions: none.
*
* Return: void
*/
void Delay(Word _ms, Byte _cancel_buttons) {
    counter_ms = 0;
    while ( (counter_ms <= _ms) && ((pushed_buttons | _cancel_buttons) == NO_BUTTON) )
        {} // Empty Body
    return;
}
API - WaitForButtonsRelease

/**
 * WaitForButtonsRelease: Waits until all buttons have been released.
 * Parameters: none.
 * Variables read: pushed_buttons.
 * Variables modified: none.
 * Subfunctions: none.
 * Return: void
 */
void WaitForButtonsRelease() {
    while (pushed_buttons != NO_BUTTON) // Wait until the user releases any pushed button(s)
        {;} // Empty Body
    return;
}

API - DisplayMsg

/**
 * DisplayMsg: Sends to the displays the string pointed by _str.
 * The string must be all upper case characters.
 * Parameters: *_str - pointer to the string to be displayed.
 * Only the first five characters of *_str will be displayed.
 * Variables read: none.
 * Variables modified: arr_display[1-5].
 * Subfunctions: none.
 * Return: void
 */
void DisplayMsg(const Byte *_str) {
    Byte _i;
    Byte _temp;
    Byte _selector_display;
    for(_i = 0, _selector_display = 1 ; _i<5 ; _i++, _selector_display++) {
        _temp = *( _str + _i);
        arr_display[ _selector_display] = TRANSLATE_LETTER[ (_temp - 32)];
    }
    return;
}
/*******************************************************************************
API - DisplayNum
*******************************************************************************
/**
* DisplayNum: Sends to the displays the number _num
*
* Parameters:         _num - number to be displayed.
*                          It must be a 5 digit number, where the
*                          last 2 digits represent decimal values.
*                          For example:
*                          if _num = 12345, the number displayed
*                          will be 123.45
*                          If the number is larger than 5 digits
*                          long, it might not be displayed
*                          correctly
*
* Variables read: none.
*
* Variables modified: arr_display[1-5].
*
* Subfunctions: none.
*
* Return: void
*/
void DisplayNum(ulong _num) {
    Byte _i;                             // general purpose counter
    Word _short_num;                     // the coefficient of _num/10
                                             // if _num > 6 digits long
                                             // _short_num is overflowed
    ulong _temp;

    for (_i = 5; _i > 0; _i--) {         // Display 5 numbers.
        _temp = (ulong)(_num / 10);     // backup the coefficient
        _short_num = _temp;            // get number without last digit
        _temp *= 10;                   // get less significant digit
        _num = _short_num;             // restart process with coefficient
        arr_display[_i] = TRANSLATE_NUMBER[(Byte)_temp];
        if (_i == 3) {
            arr_display[_i] &= SEG_P; // turn on P segment
        }
    }
    return;
}

/* End of tax_API.c */
A.7 taxi_api.h

/*================================================================*/
/**
 * Copyright (c) 2002, Motorola Inc.
 * Motorola Reference Design
 *
 * File name : tax_API.h
 * Project name : TaxAPI Reference Design
 * Author : Mauricio Capistran-Garza
 * Department : Guadalajara - SPS
 *
 * Description : Declares the function prototypes for taxi_api.c
 *
 * History :
 */
/*================================================================*/

#ifndef __TAX_API_H__
#define __TAX_API_H__
/*
#include <stdtypes.h>
#include "taxi_config.h"
#include "taxi_variables.h"
*/

/***************************************************************
 FUNCTION PROTOTYPES
***************************************************************/
/**
 * Initialize: set all global variables, set up timers and
 * irq, configure system.
 *
 * Parameters: none.
 *
 * Variables read: none.
 *
 * Subfunctions: none.
 *
 * Return: void
 */
void Initialize (void);

/**
 * CleanUp: Reset variables needed to return to idle state
 *
 * Parameters: none.
 *
 * Variables read: none.
 */
Taximeter Source Code

* Variables modified: accumulator_extras.
  * accumulator_trip.
  * accumulator_pulses.
  * accumulator_time.
  * accumulator_distance_traveled.
  * accumulator_inc.

* Subfunctions: none.

* Return: void

/**
* CleanUp: Frees all resources.
* Parameters: none.
* Variables read: none.
* Variables modified: none.
* Subfunctions: none.
* Return: void
*/
void CleanUp(void);

/**
* Delay: Waits _ms milliseconds or until one of
* _cancel_buttons is pressed.
* Parameters: _ms - number of milliseconds to
* wait before ending function.
* _cancel_buttons - buttons that might cause
* the function to end even if
* time has not finished.
* Variables read: pushed_buttons.
* Variables modified: counter_ms.
* Subfunctions: none.
* Return: void
*/
void Delay(Word _ms, Byte _cancel_buttons);

/**
* WaitForButtonsRelease: Waits until all buttons have been
* released.
* Parameters: none.
* Variables read: pushed_buttons.
* Variables modified: none.
* Subfunctions: none.
* Return: void
*/
void WaitForButtonsRelease(void);

/**
* DisplayMsg: Sends to the displays the string pointed by _str.
* The string must be all upper case characters.
* Parameters: _str.
* Variables read: pushed_buttons.
* Variables modified: none.
* Subfunctions: none.
* Return: void
*/
void DisplayMsg(Byte _str);

For More Information On This Product,
Go to: www.freescale.com
* Parameters: *str - pointer to the string to be displayed.
* Only the first five characters of *str
* will be displayed.
*
* Variables read: none.
*
* Variables modified: arr_display[1-5].
*
* Subfunctions: none.
*
* Return: void
*/
void DisplayMsg(const Byte * _str);

/**
 * DisplayNum: Sends to the displays the number _num
 *
 * Parameters: _num - number to be displayed.
 * It must be a 5 digit number, where the
 * last 2 digits represent decimal values.
 * For example:
 * if _num = 12345, the number displayed
 * will be 123.45
 * If the number is larger than 5 digits
 * long, it might not be displayed
 * correctly
 *
 * Variables read: none.
 *
 * Variables modified: arr_display[1-5].
 *
 * Subfunctions: none.
 *
 * Return: void
 */
void DisplayNum(ulong _number);

#endif // __TAX_API_H__
A.8 taxi_flash.c

/*================================================================*/
/**
* Copyright (c) 2002, Motorola Inc.
* Motorola Reference Design
*
* File name : taxi_flash.c
* Project name : Taximeter Reference Design
* Author : Mauricio Capistran-Garza
* Department : Guadalajara - SPS
*
* Description : API for flash management. With this API the user
* can erase, program and verify flash. It can also
* be used to send and receive data from programmer.
*
* History :
*/
/*================================================================*/

#include "taximeter.h"

/*******************************
FLASH PROGRAMMING API - FlashManager
*******************************
/*================================================================*/
/**
* FlashManager: It manages most ROM-Resident Routines for
* flash reprogramming. There are four basic
* ROM-Resident Routines:
*   + Program Range (PROG): it programs a range
*     of flash memory. The starting address to
*     be programmed must be defined in registers
*     H:X, and the ending address in the virtual
*     registers LADDRH:LADDRL (0x8A:0x8B). This
*     range is programmed with the RAM content
*     starting in address 0x8C.
*   + Erase Page (ERASE): it erases a page (64
*     bytes) of flash memory. To specify the
*     page to be erased, you must load H:X with
*     any address within that page.
*     Calling this routine automatically disables
*     all interrupts.
*   + Read and Verify Range (COPY | SEND): it
*     reads flash memory and compares it against
*     RAM content (starting at address 0x8C).
*     The first address to be compared must be
*     loaded in registers H:X and the last
*     address in virtual registers LADDRH:LADDRL
*     (0x8A:0x8B).
*     There are two variants of this routine
*     depending on the value of the Acc. If
*     the Acc is cleared, after comparing each
*     byte, the RAM will be overwritten. So this
*     routine can be used to COPY a range of flash
*     and place it into RAM. If the Acc is set,
* after comparing each byte, this byte will be sent serially through the communication port (PTB0). So this routine can be used to SEND data to the taximeter programmer.
* + Read Byte: this ROM-Resident routine is not managed by FlashManager.

There are also other virtual registers that all ROM-Resident routines use:
+ CTRLBYT - 0x0088 (RAM)
+ CPUSPD - 0x0089 (RAM)
The contents of these virtual registers and LADDRH:LADDRL should be backed up before calling FlashManager.

For more information on ROM-Resident Routines read AN1831.

* Entry Coditions: Back up virtual registers or they'll be overwritten.
* Exit Conditions: PTB0 configured as input.
If ERASE is called, interrupts are disabled.
* Parameters: _type - What you want to do with the flash: PROG, ERASE, COPY, SEND
_start_offset - memory offset from DATA_START at which you want to start the process chosen in _type. DATA_START is defined at the beginning of a 64-bytes page.
* Implicit Parameters: _end_offset - any process chosen in _type (excepting ERASE) begins in _start_offset and ends 32 bytes later.
_accumulator - routines for COPY and SEND require special values of Acc. This are set implicitly.

* Variables read: none.
* Variables modified: Any variable located between 0x88 and 0x8B.
* Subfunctions: PRGRNGE(); Resident ROM routine
ERARNGE(); Resident ROM routine
RDVRNG(); Resident ROM routine
* Return: void
*
void FlashManager(Byte _type, Byte _start_offset) {
    Word _temp;
    FLBPR = 0xFF; // Enable flash reading and writing
    CPUSPD = OSC; // Set the CPUSPD
    CTRLBYT &= 0xBF; // Clear bit 6 to page erase mode.
    _temp = DATA_START + _start_offset; // Calculate starting absolute address
    LADDRH = DATA_START >> 8; // Set Last Address High
}
LADDRL = (_start_offset + ROW_LIMIT); // Set Last Address Low
DDRB &= 0xFE; // Set PTB0 as input (Entry Condition)
__asm ldhx _temp; // Load in H:X starting address.
__asm lda _type; // make a switch on variable _type
__asm dbnza _NEXT1; // if _type != PROG, check next
PRGRNGE(); // Program Range
__asm bra _END_FLASH_MANAGER;
__asm _NEXT1: dbnza _NEXT2; // if _type != ERASE, check next
ERARNGE(); // Erase Page
__asm bra _END_FLASH_MANAGER;
__asm _NEXT2: deca; // Decrement Acc to set it up for
RDVRRNG(); // "Read/Verify Range (read above).
__asm _END_FLASH_MANAGER: ;
FLBPR = 0x00; // Disable flahs reading and writing
return;

FLASH PROGRAMMING API - BackupAccumulators
**********************************************************************************/
/**
* BackupAccumulators: it backs up in flash the following
* accumulators: TOTAL_DISTANCE_TRAVELED
*             TOTAL_DISTANCE_IN_SERVICE
*             TOTAL_NUMBER_TRAVELS
*             TOTAL_NUMBER_INCREMENT
*             TOTAL_INCOME
* * Parameters: none.
* *
* Variables read: accumulator_distance_traveled.
* "accumulator_trip.
* "accumulator_inc.
* *
* Variables modified: total_distance_traveled.
* total_distance_traveled_in_service.
* total_number_travels.
* total_number_increments.
* total_income.
* *
* Subfunctions: FlashManager().
* *
* Return: void
*/
void BackupAccumulators() {
    DisableInterrupts;
/* Backup RAM data; form 0x88 to 0x8B in the stack and from 0x8C to 0xAB in flash */
    __asm LDHX 0x0088;
    __asm PSHH;
    __asm PSHX;
__asm LDHX 0x008A;
__asm PSHH;
__asm PSHX;
FlashManager(ERASE, BACKUP_FLASH_OFFSET); // Erase FLASH backup space.
FlashManager(PROG, BACKUP_FLASH_OFFSET); // Backup RAM data

/* Bring all accumulators into RAM so they can be modified */
FlashManager(COPY, PAGE2_OFFSET);

total_distance_traveled += accumulator_distance_traveled; // Update values
if (accumulator_trip > 0) {
    total_income += accumulator_trip;
    total_distance_in_service += accumulator_distance_traveled;
    total_number_increments += accumulator_inc;
    total_number_travels ++;
}

/* Erase Flash Accumulators and program new values */
FlashManager(ERASE, PAGE2_OFFSET); // erase old values of Accumulators
FlashManager(PROG, PAGE2_OFFSET); // program new values

/* Set flash-verification byte to indicate that flash programming was successful */
FlashManager(COPY, PAGE2_OFFSET + ROW); // Copy into RAM flash-verification byte
__asm LDA #0xAA;
__asm STA 0xAB; // Set flash-verification byte in RAM
FlashManager(PROG, PAGE2_OFFSET + ROW); // Program flash-verification byte.

/* Restore RAM data */
FlashManager(COPY, BACKUP_FLASH_OFFSET); // Restore RAM data from flash
__asm PULX;
__asm PULH;
__asm STHX 0x008A;
__asm PULX;
__asm PULH;
__asm STHX 0x0088;

DDRB = 0xFF;
EnableInterrupts;
return;
A.9 taxi_flash.h

/*================================================================*/
/**
* Copyright (c) 2002, Motorola Inc.
* Motorola Reference Design
*
* File name : taxi_flash.h
* Project name : Taximeter Reference Design
* Author : Mauricio Capistran-Garza
* Department : Guadalajara - SPS
*
* Description : Flash programming header file.
*
* History :
*/
/*================================================================*/

#ifndef _TAXI_FLASH_H__
#define _TAXI_FLASH_H__

#ifndef TAXI_FLASH_H__
#define TAXI_FLASH_H__

/*************************************************************************************************
DEFINES
***************************************************************************************************/

#define OSC 0x05            // Bus Operation Frequency * 4

#define CTRLBYT     (*(volatile unsigned char*)(0x88))
#define CPUSPD      (*(volatile unsigned char*)(0x89))
#define LADDRH      (*(volatile unsigned char*)(0x8A))
#define LADDRL      (*(volatile unsigned char*)(0x8B))
#define RAM_START   (*(volatile unsigned char*)(0x8C))

#define GETBYTE()   {__asm jsr 0xFC00;}
#define RDVRRNG()   {__asm jsr 0xFC03;}
#define ERARNGE()   {__asm jsr 0xFC06;}
#define PRGRNGE()   {__asm jsr 0xFC09;}
#define GET_BIT()   {__asm jsr 0xFF00;}

#define RESIDENT_ROM_ROUTINES 0xFC00
#define PROG        0x01
#define ERASE       0x02
#define SEND        0x03
#define COPY        0x04

#define DATA_START               0xEC00  // to 0xECBE  -> 191 bytes
#define PAGE0_OFFSET             0x00
#define PAGE0_OFFSET_END         0x3F
#define PAGE1_OFFSET             0x40
#define PAGE1_OFFSET_END         0x3F
#define PAGE0_OFFSET            0x00
#define PAGE1_OFFSET            0x3E
#define PAGE1_OFFSET_END 0x7F
#define PAGE2_OFFSET 0x80
#define PAGE2_OFFSET_END 0xBF
#define ACCUMULATORS_OFFSET 0x80
#define ACCUMULATORS_OFFSET_END 0xBF
#define BACKUP_FLASH_OFFSET 0xECC0 // to 0xECFF -> 64 bytes
#define BACKUP_FLASH_OFFSET 0xC0
#define BACKUP_FLASH_OFFSET_END 0xFF
#define PAGE_SIZE 64 // number of bytes in one page of flash; flash gets erased on a page basis
#define ROW 32 // number of bytes in one row
#define ROW_LIMIT 31 // ending limit of a row

FUNCTION PROTOTYPES

FlashManager: It manages most ROM-Resident Routines for flash reprogramming. There are four basic ROM-Resident Routines:

+ Program Range (PROG): it programs a range of flash memory. The starting address to be programmed must be defined in registers H:X, and the ending address in the virtual registers LADDRH:LADDRL (0x8A:0x8B). This range is programmed with the RAM content starting in address 0x8C.

+ Erase Page (ERASE): it erases a page (64 bytes) of flash memory. To specify the page to be erased, you must load H:X with any address within that page. Calling this routine automatically disables all interrupts.

+ Read and Verify Range (COPY | SEND): it reads flash memory and compares it against RAM content (starting at address 0x8C). The first address to be compared must be loaded in virtual registers LADDRH:LADDRL (0x8A:0x8B). There are two variants of this routine depending on the value of the Acc. If the Acc is cleared, after comparing each byte, the RAM will be overwritten. So this routine can be used to COPY a range of flash and place it into RAM. If the Acc is set, after comparing each byte, this byte will be sent serially through the communication port (PTB0). So this routine can be used to SEND data to the taximeter programmer.

+ Read Byte: this ROM-Resident routine is not managed by FlashManager.

There are also other virtual registers that all ROM-Resident routines use:
Taximeter Reference Design Using the MC68HC908JL3

Taximeter Source Code

* + CTRLBYT - 0x0088 (RAM)
* + CPUSPD - 0x0089 (RAM)
* The contents of these virtual registers
* and LADDRH:LADDRL should be backed up before
* calling FlashManager.
* *
* For more information on ROM-Resident Routines
* read AN1831.
* *
* Entry Conditions: Back up virtual registers or they'll be
* overwritten.
* *
* Exit Conditions: PTB0 configured as input.
* If ERASE is called, interrupts are disabled.
* *
* Parameters: _type - What you want to do with the flash:
* PROG, ERASE, COPY, SEND
* _start_offset - memory offset from DATA_START
* at which you want to start the process
* chosen in _type. DATA_START is defined
* at the beginning of a 64-bytes page.
* *
* Implicit Parameters: _end_offset - any process chosen in
* _type (excepting ERASE) begins in
* _start_offset and ends 32 bytes later.
* _accumulator - routines for COPY and SEND
* require special values of Acc. This
* are set implicitly.
* *
* Variables read: none.
* *
* Variables modified: Any variable located between 0x88 and 0x8B.
* *
* Subfunctions: PRGRNGE(); Resident ROM routine
* ERARNGE(); Resident ROM routine
* RDVRRNG(); Resident ROM routine
* *
* Return: void
*/
void FlashManager(Byte _type, Byte _start_offset);

/**
* BackupAccumulators: it backs up in flash the following
* accumulators: TOTAL_DISTANCE_TRAVELED
*                          TOTAL_DISTANCE_IN_SERVICE
*                          TOTAL_NUMBER_TRAVELS
*                          TOTAL_NUMBER_INCREMENTS
*                          TOTAL_INCOME
*
* Parameters: none.
*
* Variables read: accumulator_distance_traveled.
* accumulator_trip.
* accumulator_inc.
* Variables modified: total_distance_traveled.
*                     total_distance_traveled_in_service.
*                     total_number_travels.
*                     total_number_increments.
*                     total_income.
*
* Subfunctions:       FlashManager().
* *
* Return: void
* /
void BackupAccumulators(void);

#endif  // _TAXI_FLASH_H__
A.10 taxi_states.c

/*================================================================*/
/**
* Copyright (c) 2002, Motorola Inc.
* Motorola Reference Design
*
* File name    : taxi_states.c
* Project name : Taximeter Reference Design
* Author       : Mauricio Capistran-Garza
* Department   : Guadalajara - SPS
*
* Description : This is where all the states that make up the
taximeter state machine are defined.
* The states are: Free, In Service, Pay, Extras,
* Fares, Clock, Speedometer, Pulses, Information,
* Off, and Program.
*
* History : 
*/
/*================================================================*/

#include "taximeter.h"

/******************************************
State - Free
******************************************/
/**
* Free: Idle state. It shows in the displayes the label
"FREE" and constantly scans the value of the buttons.
*
* Parameters: none.
*
* Variables read: pushed_buttons.
*
* Variables modified: current_state.
*
* Subfunctions: DisplayMsg().
*
* Return: void
*/

void Free(void){
    WaitForButtonsRelease();
    DisplayMsg(FREE_STATE_MSG);  // Display "FREE"
    if (FLASH_PROGRAM != UNCORRUPT_FLASH)  // Verify if Flash is not corrupt
        DisplayMsg(ERROR_MSG);
    while(pushed_buttons != BUTTON5)  // Empty Body
        ;
    while(-1) {
        // Stay here until a button is pressed
        switch(pushed_buttons) {
            case BUTTON1:  current_state = SERVICE;
                           return;
        }
    }
}
case BUTTON2: current_state = EXTRAS;
    return;

case BUTTON3: current_state = CLOCK;
    return;

case BUTTON4: current_state = INFO;
    return;

case BUTTON5: current_state = PROGRAM;
    return;
}
}

/***************************************************************************/
State - In Service
/***************************************************************************/
/**
* Service: Starts charging.
* According to time and distance, it increments
* the value of the total amount to pay by the
* costumer and displays it.
* 
* Parameters: none.
* 
* Variables read: pushed_buttons.
* accumulator_extras.
* ini_ch_active.
* 
* Variables modified: accumulator_trip.
* flag_charge.
* current_state.
* 
* Subfunctions: BackupAccumulators().
* CleanUp().
* DisplayMsg().
* DisplayNum().
* Delay().
* 
* Return: void
*/
void Service(void) {
    char *_ptrMsg;

    BackupAccumulators(); // Backup the distance traveled not in service
    CleanUp();
    _ptrMsg = &FARE_MESSAGE_TABLE[(fare_type * 5)];
    DisplayMsg(_ptrMsg); // Display the selected fare for one second
    WaitForButtonsRelease();
    Delay(1000, BUTTON1); // Wait one second
    accumulator_trip = accumulator_extras + ini_ch_active;
    WaitForButtonsRelease();
    flag_charge = CHARGE; // Start charging
    while (pushed_buttons != BUTTON1) { // Keep refreshing the amount to pay so far
        until the user
DisplayNum(accumulator_trip); // pushed button 1. The accumulator_trip is incremented
y en el IRQ.
}                                   // in the interrupt service routines.
current_state = PAY;               // Go to PAY state.
DisplayMsg(PAY_STATE_MSG);         // Display message "PAY"
return;
}

/***********************************************************/
State - Pay
**************************************************************************/
/**
* Pay: Stops charging.
* Displays (toggling) the label " PAY " and the total
* amount to pay.
*
* Parameters: none.
*
* Variables read: pushed_buttons.
* accumulator_trip.
*
* Variables modified: flag_charge.
* current_state.
*
* Subfunctions: DisplayMsg().
* DisplayNum().
* Delay().
*
* Return: void
*/
void Pay(void) {
    flag_charge = NO_CHARGE;            // Stop charging
    BackupAccumulators();              // Backup accumulators
    WaitForButtonsRelease();
    while (pushed_buttons != BUTTON1) { // Toggle the message "PAY" and the total
        Delay(2000, BUTTON1);          // to pay until the user presses button
        DisplayMsg(PAY_STATE_MSG);     // 1
        Display(2000, BUTTON1);
        DisplayNum(accumulator_trip);
        Delay(2000, BUTTON1);
    }
    current_state = FREE;               // Go to FREE state.
    CleanUp();                         // Reinitialize accumulators.
    return;
}

/***********************************************************/
State - Extras
**************************************************************************/
/**
* Extras: Adds one or more extra charges to the extras'
* accumulator. It can also clear this accumulator.
*
Parameters: none.

Variables read: pushed_buttons.

Variables modified: accumulator_extras.

Subfunctions: DisplayMsg().

Return: void

```c
void Extras(void) {
    Byte _extra_ch_temp = 1;  // The extra charges go from 1 to 9
    DisplayMsg(EXTRA_STATE_MSG1);  // Display "PLUS" leyd for one second
    WaitForButtonsRelease();
    Delay(1000, BUTTON2&BUTTON3);
    DisplayMsg(EXTRA_STATE_MSG2);
    WaitForButtonsRelease();
    while (-1) {
        Delay(10000, BUTTON2&BUTTON3&BUTTON5);  // If in 10 seconds the user doesn't do
        // anything, then go back to FREE state
        if (pushed_buttons != NO_BUTTON) {  // If the user pressed the button number
            if (pushed_buttons == BUTTON2) {  // 2 -> Go to FARES state
                current_state = FARES;
                return;
            }
            if (pushed_buttons == BUTTON5) {  // 5 -> Handle the
                accumulator_extras
                if (_extra_ch_temp == 0) {
                    accumulator_extras = 0;  // Erase accumulator_extras
                } else {
                    accumulator_extras += EXTRA_TABLE[_extra_ch_temp-1]; // Add an extra
                }
                current_state = FREE;  // Go to FREE state.
                return;
            }
            if (pushed_buttons == BUTTON3) {  // 3 -> Go to the next extra charge
                _extra_ch_temp++;
                if (_extra_ch_temp > EXTRAS_NUMBER) {
                    _extra_ch_temp = 0;
                }
            } else {
                DisplayMsg(EXTRA_STATE_MSG_ERASE);
                arr_display[5] = TRANSLATE_NUMBER[_extra_ch_temp];
            }
            WaitForButtonsRelease();
        } else {  // 10 seconds went by and no buttons were pressed
```
current_state = FREE; // Go to FREE state
return;
}
}

/***************************************************************
State - Fares
***************************************************************
/**
* Fares: Selects the fare type to use as well as the
*        initial charge corresponding to that fare.
* *
* Parameters: none.
* *
* Variables read: pushed_buttons.
* *
* Variables modified: fare_active.
*                     ini_ch_active.
*                     current_state.
*                     arr_display[0].
*                     fare_type.
* *
* Subfunctions: DisplayMsg().
*               Delay().
* *
* Return: void
*/
void Fares(void) {
  Byte *_tarMsg;
  _tarMsg = &_FARE_MESSAGE_TABLE[(fare_type * 5)]; // Display "FARE " for 1 second
  DisplayMsg(FARE_STATE_MSG);                     // Display "FARE " for 1 second
  WaitForButtonsRelease();
  Delay(500, BUTTON2&BUTTON3);                   // Wait one second
  DisplayMsg(_tarMsg);                           // Display the active-fare message
  WaitForButtonsRelease();
  while (-1) {
    Delay(10000, BUTTON2&BUTTON3&BUTTON5);      // If in 10 seconds the user doesn't
      // do    // anything, then go back to
    FREE state
      if (pushed_buttons != NO_BUTTON) {       // If the user presses the button num-
        if (pushed_buttons == BUTTON2) {     // 2 -> Go to FREE state
          current_state = FREE;
          return;
        }
        if (pushed_buttons == BUTTON5) {     // 5 -> Change the active fare
          fare_active = FARE_TABLE[fare_type];
          ini_ch_active = INI_CH_TABLE[fare_type];
          arr_display[0] = TRANSLATE_NUMBER[fare_type+1];
          current_state = FREE;               // Go to FREE state
          return;
        }
      }
  }
}
if (pushed_buttons == BUTTON3) {       // 3 -> Go to the next fare
    fare_type++;                        
    if (fare_type >= FARES_NUMBER) {    
        fare_type = 0;                   
    }
    _tarMsg = &FARE_MESSAGE_TABLE[(fare_type * 5)];
    DisplayMsg(_tarMsg);
    WaitForButtonsRelease();
} else {                          // 10 seconds went by and no buttons were pressed
    current_state = FREE;         // Go to FREE state
    return;
}
}

/************************************************************
State - Clock
************************************************************/
/**
* Clock: Display the time.
* The hour and minute counters are managed in the
* Timer ISR
* *
* Parameters: none.
* *
* Variables read: pushed_buttons.
* clock_hours.
* clock_minutes.
* *
* Variables modified: clock_hours.
* clock_minutes.
* current_state.
* *
* Subfunctions: DisplayMsg().
* DisplayNum().
* Delay().
* *
* Return: void
*
*/
void Clock(void) {
    unsigned long _time;

    DisplayMsg(CLOCK_STATE_MSG);                  // Display "TIME " for 1 second
    WaitForButtonsRelease();
    Delay(1000, BUTTON3);
    while(pushed_buttons != BUTTON3) {            // Display the time until button 3 is
      _time = (clock_hours * 100) + clock_minutes;
      DisplayNum(_time);
      if (pushed_buttons == BUTTON1) {          // Hours ++
        clock_hours++;
        if (clock_hours >= 24) {
          clock_hours = 0;
        }
    }
Taximeter Source Code

```c
void Speedometer(void) {
    Byte _current = 0;                      // number of pulses in current time interval
    Byte _previous = 0;                     // number of pulses in previous time interval
    unsigned long _spd;                     // velocidad = 0
    const int _time = 1500;                 // time of each sample
    const int _factor = 240;                // to translate to Km/h

    DisplayMsg(SPEED_STATE_MSG);            // Display "SPEED" for 1 second
    WaitForButtonsRelease();
    Delay(1000, BUTTON3);
    Delay(1000, BUTTON3);
    counter_ms = 0;                         // reset general purpose counters
    counter_pulses = 0;                     // for distance and time
```

State - Speedometer

* Speedometer: Shows the speed at which the car is traveling.
* The speed is calculated as following:
* `#of-wheel-turns / time`
* It is assumed that the perimeter of the wheel
* is one meter.
* The speedometer has a resolution of 2.4 Km/h,
* this resolution can be improved by increasing
* the _time and re-calculate the _factor.

* Parameters: none.
* Variables read: pushed_buttons.
* counter_ms.
* counter_pulses.
* Variables modified: counter_ms.
* counter_pulses.
* current_state.
* Subfunctions: DisplayMsg().
* DisplayNum().
* Delay().
* Return: void
*/
while(pushed_buttons != BUTTON3) {  // Display speed until button 3 is pressed
  _spd = (unsigned long) ((_current + _previous) / 2) * _factor;
  // Speed = #pulses * 240; 240 is a factor to go
  DisplayNum(_spd);
  if(counter_ms >= _time) {
    _previous = _current;
    _current = (Byte) counter_pulses;
    counter_pulses = 0;
    counter_ms = 0;
  }
}
current_state = PULSES;  // Go to PULSES state

/******************************************************************************
 State - Pules
*******************************************************************************/
/**
 * Pules: Verifies that the wheel-turn transductor is working properly. It counts
 * the number of pulses caused by the turn of the wheels and it displays it. It
 * can also reset or freeze the counter.
 *
 * Parameters: none.
 *
 * Variables read: pushed_buttons.
 * counter_pulses.
 *
 * Variables modified: counter_pulses.
 * current_state.
 *
 * Subfunctions: DisplayNum().
 * Delay().
 *
 * Return: void
 */
void Pulses(void) {
  DisplayMsg(PULSE_STATE_MSG);  // Display lenyed "PULSE" for 1 second
  WaitForButtonsRelease();
  Delay(1000,BUTTON3);  // Wait 1 second
  WaitForButtonsRelease();
  counter_pulses = 0;
  while (pushed_buttons != BUTTON3) {  // Display the number of pulses it it
    DisplayNum(counter_pulses);  // receiving until the user press button
    if (pushed_buttons == BUTTON1) {  // Button 1 = reset
      counter_pulses = 0;
    }
    if (pushed_buttons == BUTTON2) {  // Button 2 = freeze
      Delay(20000,BUTTON1);
    }
  }
current_state = FREE;  // Go to FREE state
/*****************************************
State - Information
******************************************/

/**
* Info: Displays the desired information and the accumulators.
* The information to be displayed is stored in
* INFO_TABLE. It can display 3 messages (8 character
* long each message). It also displays the accumulators
* stored in flash. The accumulators are displayed in
* the following order:
* TOTAL_DISTANCE_TRAVELED
* TOTAL_DISTANCE_IN_SERVICE
* TOTAL_NUMBR_TRAVELS
* TOTAL_NUMBER_INCREMENTs
* TOTAL_INCOME
*
* Parameters: none.
*
* Variables read: pushed_buttons.
*
* Variables modified: current_state.
*
* Subfunctions: DisplayMsg().
* DisplayNum().
* Delay().
*
* Return: void
*/

void Info(void) {
  Byte _char_offset=0;
  Byte _message_offset=0;
  ulong *_acc_table;

  _acc_table = &TOTAL_DISTANCE_TRAVELED;
  DisplayMsg(INFO_STATE_MSG); // Display leyend "INFOR" for 1 second
  WaitForButtonsRelease();
  Delay(1000,BUTTON4); // Wait 1 second
  WaitForButtonsRelease();

  while (pushed_buttons != BUTTON4) {
    if(_message_offset < 24){ // Show informaiton until the user
      DisplayMsg(&INFO_TABLE[_message_offset + _char_offset]);
      _char_offset++;
    }
    if (_char_offset > 3) { // Since the messages are 8 char long
      _char_offset = 0; // it has to scroll 3 times.
    } else {
      DisplayNum(*_acc_table + 1); // Accumulators are initialized in -1
    }
  }
  Delay (800, BUTTON4&BUTTON3);

  if (pushed_buttons == BUTTON3) { // Button 3 -> Next message
    WaitForButtonsRelease();
    _message_offset += LENGTH_MESSAGE;
    _char_offset = 0;
    if (_message_offset > LENGTH_3_MESSAGES) 

```c
void Program (void) {
    Byte _counter;                          // aux. counter to received bytes
    /* The user must press BUTTON5 during 6 seconds in order to start programming */
    DisplayMsg(OFF_STATE_MSG_START);
    for (_counter = 0; _counter < 200; _counter++) {
        if (pushed_buttons != BUTTON5) {
            current_state = OFF;
        } else {
            DisplayMsg(ADMINISTRATOR_MSG_OFF_STATE_PROGRAMMING);
            ERASE_ROW Đề Command:
            if (_counter < 20) {
                current_state = FREE;
            } else {
                current_state = PROGRAMMING;
            }
        }
    }
}
```
return;
}
DisplayMsg(PROG_STATE_MSG_START);
Delay(1000, NO_BUTTON);
WaitForButtonsRelease();
DisplayMsg(EMPTY_MSG);
Delay(50, NO_BUTTON);

/* Handsake: Wait to receive 0xAA and the send accumulators */
DisableInterrupts;

DDRB &= 0xFE;                           // GETBYTE Entry condition; PB0 configured as input
PTB &= 0xFE;                            // GETBYTE Entry condition; PB0 = 0
GETBYTE();
__asm cmp #0xAA;                        // Must receive a 0xAA confirmation to start programming
__asm beq _CONTINUE;
EnableInterrupts;
DisplayMsg(PROG_STATE_NO_COMM);
while(-1);
__asm _CONTINUE: ;
FlashManager(SEND,
            ACCUMULATORS_OFFSET);
/* Erase Flash Tables and FLASH_PROGRAM variable */
FlashManager(ERASE,
            PAGE0_OFFSET);
FlashManager(ERASE,
            PAGE1_OFFSET);
FlashManager(ERASE,
            PAGE2_OFFSET);

/* Receive new Tables from PC, if a byte is not received, the software will enter into an infinite loop. */
_counter = 0;
__asm _GET32:    ldhx #0x008C; // Where the data structure in RAM begins
__asm _RECEIVE_BYTE: jsr $0FC00; // GETBYTE();
__asm sta ,X;    // save the received byte in location pointed by H:X
__asm incx;     // point to next location in RAM
__asm cpx #0x00AC; // if H:X has incremented 32 times, exit loop
__asm bne _RECEIVE_BYTE; // else, keep receiving
FlashManager(PROG,
            _counter);
FlashManager(SEND,
            _counter);
_counter+= ROW;
if (_counter < 192) {
    __asm bra _GET32; // get the next 32 bytes
}
DDRB = 0xFF;
EnableInterrupts;
DisplayMsg(PROG_STATE_MSG_END);
while(-1)
    {;} // Empty body

/***************************************************************************/
State - Off
/***************************************************************************/
/**
 * Off: Add up to accumulators and store in flash the Km
 *      traveled since the last time it was stored.
 *      Turns displays off.
 *      Exit this state with any button pushed.
 *      
 * Parameters:         none.
 *                    *
 * Variables read:     pushed_buttons.
 *                    *
 * Variables modified: current_state.
 *                    *
 * Subfunctions:       DisplayMsg().
 *                    *
 * Return: void
 */
void Off(void) {

    if (FLASH_PROGRAM != UNCORRUPT_FLASH) { // If Flash is not corrupt
        BackupAccumulators();
        CleanUp();
    }
    DisplayMsg(EMPTY_MSG);
    arr_display[0] = 0xFF;
    WaitForButtonsRelease();
    while ((pushed_buttons == NO BUTTON) || pushed_buttons == BUTTON5) // Turn on displays with any button press
        {;} // Empty Body
    arr_display[0] = TRANSLATE_NUMBER[fare_type+1];
    current_state = FREE;
}
A.11 taxi_states.h

/***************************************************************
  DEFINES
***************************************************************
#define LENGTH_MESSAGE 8
#define LENGTH_3_MESSAGES 24
#define LENGTH_8_MESSAGES 64

/***************************************************************
  FUNCTION PROTOTYPES
***************************************************************
/**
 * Free: Idle state. It shows in the displayes the label
 * "FREE" and constantly scans the value of the buttons.
 * Parameters: none.
 * Variables read: pushed_buttons.
 * Variables modified: current_state.
 * Subfunctions: DisplayMsg().
 * Return: void
 */
void Free(void);

/**
 * Service: Starts charging.
 * According to time and distance, it increments
 * the value of the total amount to pay by the
 * customer and displays it.
 */
**Parameters:** none.

**Variables read:** pushed_buttons.

* accumulator_extras.
* ini_ch_active.

**Variables modified:** accumulator_trip.

* flag_charge.
* current_state.

**Subfunctions:** DisplayMsg().

* DisplayNum().
* Delay().

**Return:** void

*/

void Service(void);

/**
* Pay: Stops charging.
* Displays (toggling) the label "PAY" and the total
* amount to pay.
*
* **Parameters:** none.
*
* **Variables read:** pushed_buttons.
* accumulator_trip.
*
* **Variables modified:** flag_charge.
* current_state.
*
* **Subfunctions:** DisplayMsg().

* DisplayNum().
* Delay().
*
* **Return:** void

*/

void Pay(void);

/**
* Extras: Adds one or more extra charges to the extras'
* accumulator. It can also clear this accumulator.
*
* **Parameters:** none.
*
* **Variables read:** pushed_buttons.
*
* **Variables modified:** accumulator_extras.
* arr_display[5]
* current_state.
*
* **Subfunctions:** DisplayMsg().

* Delay().
*
* Return: void
*/
void Extras(void);

/**
 * Fares: Selects the fare type to use as well as the
 * initial charge corresponding to that fare.
 *
 * Parameters: none.
 *
 * Variables read: pushed_buttons.
 *
 * Variables modified: fare_active.
 * ini_ch_active.
 * current_state.
 * arr_display[0].
 * fare_type.
 *
 * Subfunctions: DisplayMsg().
 * Delay().
 *
 * Return: void
*/
void Fares(void);

/**
 * Clock: Display the time.
 * The hour and minute counters are managed in the
 * Timer ISR
 *
 * Parameters: none.
 *
 * Variables read: pushed_buttons.
 * clock_hours.
 * clock_minutes.
 *
 * Variables modified: clock_hours.
 * clock_minutes.
 * current_state.
 *
 * Subfunctions: DisplayMsg().
 * DisplayNum().
 * Delay().
 *
 * Return: void
*/
void Clock(void);

/**
 * Speedometer: Shows the speed at which the car is traveling.
 * The speed is calculated as following:
 * #of-wheel-turns / time
 * It is assumed that the perimeter of the wheel
 * is one meter.
 * The speedometer has a resolution of 2.4 Km/h,
 */
/*
   this resolution can be improved by increasing
   the _time and re-calculate the _factor.
*/

/**
 * Parameters: none.
 *
 * Variables read: pushed_buttons.
 * counter_ms.
 * counter_pulses.
 *
 * Variables modified: current_state.
 * counter_ms.
 * counter_pulses.
 *
 * Subfunctions: DisplayMsg().
 * DisplayNum().
 * Delay().
 *
 * Return: void
 */

void Speedometer(void);

/**
 * Pulses: Verifies that the wheel-turn transductor is working
 * properly. It counts the number of pulses caused
 * by the turn of the wheels and it displays it. It
 * can also reset or freeze the counting.
 *
 * Parameters: none.
 *
 * Variables read: pushed_buttons.
 * counter_pulses.
 *
 * Variables modified: current_state.
 * counter_pulses.
 *
 * Subfunctions: DisplayNum().
 * Delay().
 *
 * Return: void
 */

void Pulses(void);

/**
 * Info: Displays the desired information and the accumulators.
 * The information to be displayed is stored in
 * INFO_TABLE. It can display 3 messages (8 character
 * long each message). It also displays the accumulators
 * stored in flash. The accumulators are displayed in
 * the following order:
 * TOTAL_DISTANCE_TRAVELED
 * TOTAL_DISTANCE_IN_SERVICE
 * TOTAL_NUMBER_TRAVELS
 * TOTAL_NUMBER_INCREMENTST
 * TOTAL_INCOME
 */
void Info(void);

/**
 * Program: Changes reference values with new ones, such as:
 *        fares, initial charges, extra charges, active
 *        fares, active extra charges, information table.
 *        All interrupts are disabled while communication
 *        is in process.
 *        It waits indefinitely to receive a byte. If
 *        the byte is never received (there's no connection)
 *        the taximeter must be restarted.
 *        After programation is done, the taximeter must be
 *        restarted. There's an infinite loop (this way
 *        the RAM data that was overwritten doesn't need to
 *        be previously backed up.
 *        
 * Parameters: none.
 */
void Program (void);

/**
 * Off: Add up to accumulator and store in flash the Km
 *        traveled since the last time it was stored.
 *        Turns displays off.
 *        Exit this state with any button pushed.
 *        
 * Parameters: none.
 */
* Variables read: pushed_buttons.
* Variables modified: current_state.
* Subfunctions: DisplayMsg().
* Return: void
*/
void Off(void);

#endif  // _TAXI_STATES_H__
A.12 taxi_variables.h

/***************************/
/**
* Copyright (c) 2002, Motorola Inc.
* Motorola Reference Design
*
* File name : taxi_variables.h
* Project name : Taximeter Reference Design
* Author : Mauricio Capistran-Garza
* Department : Guadalajara - SPS
*
* Description : All global variables used in proyect Taximeter
* are defined in this file.
*
* History :
*/
/***************************/

ifndef __GLOB_VAR_H__
define __GLOB_VAR_H__

/**************************************************************************
* These variables are allocated from address 0x88 to 0xAC. During
* flash management these variables are backe up in BACKUP_FLASH
* area.
*/

/* Display related variables */
EXT Byte display_en; // Used for the multiplexing of the displays.
EXT Byte arr_display_index; // Indicates which display is selected
EXT Byte arr_display[6]; // Array containing the data that is to
// be shown in the diplayes.

/* Push button related variables */
EXT Byte read_buttons_time; // To read the buttons every 30 ms.
EXT Byte pushed_buttons; // Byte indicating which buttons are pressed.

/* State-machine related variables */
EXT Byte current_state; // To select a state in the state machine.

/* Variables used in state FARES */
EXT Byte fare_type; // Indicates which fare is being selected.
EXT Word fare_active; // Indicates which fare is active.
EXT Word ini_ch_active; // Indicates which initial charge is active.

/* Variables used for timing and counting*/
EXT Word counter_ms; // General purpose counter; Incremented every
  // timer overflow: - 1ms
EXT Word counter_pulses; // General purpose counter; Incremented every
  // IRQ - wheel turn.
EXT Byte clock_hours; // Incremented every hour
EXT Byte clock_minutes; // Incremented every minute
EXT Word clock_ms; // Incremented every ms.

/* Variables used for accumulating while in service */
EXT Word accumulator_pulses; // Number of pulses since last increment.
EXT Word accumulator_time; // Number of ms since last increment.
EXT Word accumulator_extras; // Accumulator of all extra charges applied.

/* Variables used for charging */
EXT Word inc_time; // Time for an increment in accumulator_trip.
EXT Word inc_distance; // Distance for an increment in accumulator_trip.

/***************************************************************
* These variables are allocated from address 0x80 to 0x87 and from
* 0xAD to 0xAF. These variables must not overlap with any other
* global variable. If global variables are added, you must verify
* (using the .map file) that these variables do not overlap with any
* others. It is also important to keep these variables in the
* location they are placed, because they are read <BackupAccumulators>
* when flash memory is being manipulated. Manipulation of flash
* memory requires to use RAM area from 0x88 to 0xAC, so these variables
* must be outside that area.
*/

EXT ulong accumulator_distance_traveled @ 0x80; // Distance traveled since
    // last time it was
    // backed up in flash
EXT ulong accumulator_trip @ 0x84; // Accumulator of the trip
    // income.
EXT Word accumulator_inc @ 0xAD; // Number of increments
    // in accumulator_trip
EXT Byte flag_charge @ 0xAF; // Indicates whether the
    // trip is being charged
    // or not. 1 = Charging.
    // 0 = No Charging.

/***************************************************************
* Variables only used when reprogramming accumulators.
* These variables overlap with some of the previously defined
* variables. However, these variables are used only in specific
* places <BackupAccumulators()> and overlapping variables are
* backed up before they are overwritten and they are restored
* later. The position of these variables matches the position
* of variables: TOTAL_DISTANCE_TRAVELED, TOTAL_DISTANCE_IN_SERVICE,
* TOTAL_NUMBER_TRAVERS, TOTAL_NUMBER_INCREMENT, and TOTAL_INCOME
* (The previous variables are stored starting at 0xEC01 = page
* address 0xEC80 + offset 0x0001. When they are read into RAM, they
* are placed starting at address 0x008C. Therefore 0x008D has
* an offset of 0x0001).
*/

EXT ulong total_distance_traveled @ 0x8D;
EXT ulong total_distance_in_service @ 0x91;
EXT ulong total_number_travels @ 0x95;
EXT ulong total_number_increments @ 0x99;
EXT ulong total_income @ 0x9D;

#endif // ___GLOB_VAR_H__
A.13 taxi_messages.h

/*================================================================*/
/**
* Copyright (c) 2002, Motorola Inc.
* Motorola Reference Design
* File name : taxi_messages.h
* Project name : Taximeter Reference Design
* Author : Mauricio Capistran-Garza
* Department : Guadalajara - SPS
* Description : Defines all the messages that the taxi displays.
* History :
*/
/*================================================================*/
#ifndef __TAXI_MESSAGES_H__
#define __TAXI_MESSAGES_H__
/*================================================================*/
/*=================    Definition of messages   ==================*/
/*================================================================*/
#define EMPTY_MSG                   "     "
/* By default, the messages are in spahanish. */
#ifdef LANGUAGE_SPANISH
#define FREE_STATE_MSG          "LIBRE"
#define PAY_STATE_MSG           "PAGAR"
#define EXTRA_STATE_MSG1        " MAS "
#define EXTRA_STATE_MSG2        "MAS-1"
#define EXTRA_STATE_MSG_ERASE   "BORRA"
#define FARE_STATE_MSG          "TARIF"
#define PULSE_STATE_MSG         "PULSO"
#define INFO_STATE_MSG          "INFOR"
#define CLOCK_STATE_MSG         "RELOJ"
#define SPEED_STATE_MSG         "VELOC"
#define PROG_STATE_MSG_START    "PROGR"
#define PROG_STATE_MSG_END      "FIN"
#define PROG_STATE_NO_COMM      "E-COM"
#define OFF_STATE_MSG_START     "APAGA"
#define ERROR_MSG               "ERROR"
#endif
/* English */
#ifdef LANGUAGE_ENGLISH
#define FREE_STATE_MSG          "FREE 
#define PAY_STATE_MSG           " PAY"
#define EXTRA_STATE_MSG1        "PLUS 
#define EXTRA_STATE_MSG2        "PL-01"
#define EXTRA_STATE_MSG_ERASE   "ERASE"
#define FARE_STATE_MSG          "FARES"
#endif
/*================================================================*/
#define PULSE_STATE_MSG "PULSE"
#define INFO_STATE_MSG "INFOR"
#define CLOCK_STATE_MSG "TIME"
#define SPEED_STATE_MSG "SPEED"
#define PROG_STATE_MSG_START "PROGR"
#define PROG_STATE_MSG_END "DONE"
#define PROG_STATE_NO_COMM "E-COM"
#define OFF_STATE_MSG_START "OFF"
#define ERROR_MSG "ERROR"
#endif

/* French */
#ifndef LANGUAGE_FRENCH
#define FREE_STATE_MSG "LIBRE"
#define PAY_STATE_MSG "PAYER"
#define EXTRA_STATE_MSG1 "PLUS"
#define EXTRA_STATE_MSG2 "PL-01"
#define EXTRA_STATE_MSG_ERASE "EFFAC"
#define FARE_STATE_MSG "TARIF"
#define PULSE_STATE_MSG "POULS"
#define INFO_STATE_MSG "INFOR"
#define CLOCK_STATE_MSG "HORLO"
#define SPEED_STATE_MSG "VITES"
#define PROG_STATE_MSG_START "PROGR"
#define PROG_STATE_MSG_END "FIN"
#define PROG_STATE_NO_COMM "E-COM"
#define OFF_STATE_MSG_START "ETEIN"
#define ERROR_MSG "ERREU"
#endif

/* Portuguese */
#ifndef LANGUAGE_PORTUGUESE
#define FREE_STATE_MSG "LIBRE"
#define PAY_STATE_MSG "PAYER"
#define EXTRA_STATE_MSG1 "PLUS"
#define EXTRA_STATE_MSG2 "PL-01"
#define EXTRA_STATE_MSG_ERASE "EFFAC"
#define FARE_STATE_MSG "TARIF"
#define PULSE_STATE_MSG "POULS"
#define INFO_STATE_MSG "INFOR"
#define CLOCK_STATE_MSG "HORLO"
#define SPEED_STATE_MSG "VITES"
#define PROG_STATE_MSG_START "PROGR"
#define PROG_STATE_MSG_END "FIN"
#define PROG_STATE_NO_COMM "E-COM"
#define OFF_STATE_MSG_START "APAGA"
#define ERROR_MSG "ERREU"
#endif

#endif // __TAXI_MESSAGES_H
A.14 taxi_config.h

/*================================================================*/
/**
* Copyright (c) 2002, Motorola Inc.
* Motorola Reference Design
*
* File name : taxi_config.h
* Project name : Taximeter Reference Design
* Author : Mauricio Capistran-Garza
* Department : Guadalajara - SPS
*
* Description : This file contains all the taximeter´s settings.
* This file can be used to choose the language
* of the taximeter, choose between debug and regular
* mode, etc.
*
* History :
*/
/*================================================================*/
/*================================================================*/
/*===================    Decide the language   ===================*/
/*================================================================*/
#elif define LANGUAGE_SPANISH
#elif define LANGUAGE_ENGLISH
#elif define LANGUAGE_FRENCH
#elif define LANGUAGE_PORTUGUESE
/*================================================================*/
/*================================================================*/
/*=================      Mode of compliation     =================*/
/*================================================================*/
#elif define DEBUG_MODE
#elif define REGULAR_MODE
/*================================================================*/
/*================================================================*/
//====================   DEFINES   =======================
/*================================================================*/
// FLASH ADDRESSES
#define FLASH_START 0xEC00

// STATES (options in taximeter.c switch)
#define FREE 0
#define SERVICE 1
#define PAY 2
#define EXTRAS 3
#define FARES 4
#define CLOCK 5
#define SPEEDOMETER 6
#define PULSES 7
#define INFO 8
#define PROGRAM 9
#define OFF 10
// BUTTONS: every button is a bit within a byte (negative logic)
#define NO_BUTTON   0x1F
#define BUTTON1     0x1E
#define BUTTON2     0x1D
#define BUTTON3     0x1B
#define BUTTON4     0x17
#define BUTTON5     0x0F

// FARE AND INITIAL_CHARGES (INI_CH) TYPES
#define DAY         0
#define NIGHT       1
#define SUNDAY      2
#define TIME        3
#define FARE5       4
#define FARE6       5
#define FARE7       6
#define FARE8       7
#define FARE9       8

// FARES (index within the FARES_TABLE[9] array. Every fare is a WORD)
#define FARE_DAY    0
#define FARE_NIGHT  1
#define FARE_SUNDAY 2
#define FARE_TIME   3
#define FARE_5      4
#define FARE_6      5
#define FARE_7      6
#define FARE_8      7
#define FARE_9      8

// INITIAL_CHARGES (INI_CH): (index within the
//    INI_CH_TABLE[9] array. Every INI_CH is a WORD)
#define INI_CH_DAY  0
#define INI_CH_NIGHT 1
#define INI_CH_SUNDAY 2
#define INI_CH_TIME  3
#define INI_CH_5   4
#define INI_CH_6   5
#define INI_CH_7   6
#define INI_CH_8   7
#define INI_CH_9   8

// EXTRAS TYPES
#define EXTRA1      0
#define EXTRA2      1
#define EXTRA3      2
#define EXTRA4      3
#define EXTRA5      4
#define EXTRA6      5
#define EXTRA7      6
#define EXTRA8      7
#define EXTRA9      8
// FLAG CHARGE
#define NO_CHARGE 0x00
#define CHARGE 0x01

// DISPLAY'S SEGMENTS
#define SEG_A ~0x20
#define SEG_B ~0x10
#define SEG_C ~0x04
#define SEG_D ~0x02
#define SEG_E ~0x01
#define SEG_F ~0x40
#define SEG_G ~0x80
#define SEG_P ~0x08
#define DISPLAY_OFF 0xFF

// FLASH PROGRAM VERIFICATION
#define CORRUPT_FLASH 0xFF
#define UNCORRUPT_FLASH 0xAA

A.15  taximeter.prm

NAMES END

SEGMENTS
RAM   = READ_WRITE 0x0088 TO 0x00FF;
FLASH_T = READ_ONLY 0xEC00 TO 0xEC7F;  // 128 bytes
FLASH_V = READ_ONLY 0xEC80 TO 0xECBE;  // 63 bytes
FLASH_C = READ_ONLY 0xECBF TO 0xECBF;  // 1 byte
BACKUP_T = READ_ONLY 0xEC00 TO 0xECFF;  // 64 bytes
ROM   = READ_ONLY 0xED00 TO 0xFBFF;
END

PLACEMENT
MY_ZEROPAGE, DEFAULT_RAM INTO RAM;
DEFAULT_ROM, ROM_VAR, STRINGS INTO ROM;
FLASH_TABLES INTO FLASH_T;
FLASH_VERIFY INTO FLASH_C;
FLASH_VARIABLES INTO FLASH_V;
BACKUP_TABLES INTO BACKUP_T;

END

STACKSIZE 0x50

VECTOR 0 main
Appendix B. PC Interface Source Code

B.1 Introduction

This appendix contains the PC interface source code.

B.2 PC Interface Source Code

/*================================================================*/
/**
* Copyright (c) 2002, Motorola Inc.
* Motorola Reference Design
* File name : taxi_pc_interface.c
* Project name : Taximeter Reference Design
* Author : Mauricio Capistran-Garza
* Department : Guadalajara - SPS
* Description : This is the PC interface used to program the
* taximeter flash tables. Read section "PC Interface" in the Taximeter Reference Design
* for further explanation.
* It is assumed that the Programmer and Testing
* board is used for the communication. This
* implies that every byte sent to the taximeter
* will also be received by the PC.
* History :
* /
/*================================================================*/

#include <dos.h>
#include <stdio.h>
/********** PC Interface Source Code **********/
/#define UART 0x03F8 // Where data is received and transmitted
/#define IER (UART+1) // Interrupt Enable Register 0x03F9
/#define IIR (UART+2) // Interrupt Identification Register (read only) 0x03FA
/#define FCR (UART+2) // FIFO Control Register (write only)
/#define BRLSB (UART) // Baud rate divisors LSB, if LCR bit 7 is set
/#define BRMSB (UART+1) // Baud rate divisor MSB, if LCR bit 7 is set
/#define LCR (UART+3) // Line Control Register 0x03FB
/#define BITS8 0x03 // 8-bits-per-character mask
/#define STOP1 0x00 // 1-stop-bit mask
/#define STOP2 0x04 // 2-stop-bits mask
/#define PARITY 0x38 // Enable parity and set it to SPACE parity (always 0)
/#define DLAB 0x80 // Set UART and IER as baud rate divisors
/#define MCR (UART+4) // Modem Control Register 0x03FC
/#define LSR (UART+5) // Line Status Register (read only) 0x03FD
/#define DR 0x01 // Data ready
/#define OE 0x02 // Overrun error mask
/#define FE 0x08 // Framing error mask
/#define PE 0x04 // Parity error mask
/#define THRE 0x20 // Transmitter holding register empty
/#define TSRE 0x40 // Transmitter shift register empty
/#define MSR (UART+6) // Modem Status Register (read only) 0x03FE
/#define DSR 0x20 // Data Set Ready
/#define BR4800 0x0018 // Baud Rate divisor for 4800 bps
/#define BR9600 0x000C // Baud Rate divisor for 9600 bps
/#define NO_BYTE 0x00

typedef unsigned char Byte;

/#define UART 0x03F8
/#define IER 0x03F9
/#define IIR 0x03FA
/#define FCR 0x03FA
/#define BRLSB (UART)
/#define BRMSB (UART+1)
/#define LCR (UART+3)
/#define BITS8 0x03
/#define STOP1 0x00
/#define STOP2 0x04
/#define PARITY 0x38
/#define DLAB 0x80
/#define MCR (UART+4)
/#define LSR (UART+5)
/#define DR 0x01
/#define OE 0x02
/#define FE 0x08
/#define PE 0x04
/#define THRE 0x20
/#define TSRE 0x40
/#define MSR (UART+6)
/#define DSR 0x20
/#define BR4800 0x0018
/#define BR9600 0x000C
/#define NO_BYTE 0x00

typedef unsigned char Byte;

void SetComPort(void);
Byte Receive(void);
void Send(Byte data);
/*================================================================*/
/*============================   MAIN   ==========================*/
/*================================================================*/

void main () {
  Byte i,j,temp;
  Byte tables_back[192]; // In this array is where echoed data will be stored
  Byte tables[192] = { // This array contains the data to be programmed.
/* FARE_MESSAGE_TABLE */
    ' ', 'D', 'A', 'Y', '1',
    'F', 'A', 'R', 'E', '2',
    'F', 'A', 'R', 'E', '3',
    'F', 'A', 'R', 'E', '4',
    'F', 'A', 'R', 'E', '5',
    'F', 'A', 'R', 'E', '6',
    'F', 'A', 'R', 'E', '7',
    'F', 'A', 'R', 'E', '8',
    'F', 'A', 'R', 'E', '9',

/* INI_CH_TABLE */
    0x00, 0x01,
    0x00, 0x02,
    0x00, 0x03,
    0x00, 0x04,
    0x00, 0x05,
    0x00, 0x06,
    0x00, 0x07,
    0x00, 0x08,
    0x00, 0x09,

/* FARE_TABLE */
    0x00, 0x02,
    0x00, 0x04,
    0x00, 0x06,
    0x00, 0x08,
    0x00, 0x0A,
    0x00, 0x0C,
    0x00, 0x0E,
    0x00, 0x10,
    0x00, 0x12,

/* EXTRA_TABLE */
    0x00, 0x0A, //10
    0x00, 0x14, //20
    0x00, 0x1E, //30
    0x00, 0x28, //40
    0x00, 0x32, //50
    0x00, 0x3C, //60
    0x00, 0x46, //70
    0x00, 0x50, //80
    0x00, 0x5A, //90

/* INFO_TABLE */
    'M', 'E', 'S', 'S', 'A', ' ', '1', ' ',
    'M', 'E', 'S', 'S', 'A', ' ', '2', ' ',
    'M', 'E', 'S', 'S', 'A', ' ', '3', ' ',

/* STUFFING_BYTES - to complete first 128 bytes */
0xFF,0xFF,0xFF,0xFF,
/* FLASH VARIABLES */
0xFF, // Empty
0x00,0x00,0x00,0x0A, // TOTAL_DISTANCE_TRAVELED
0x00,0x00,0x00,0x14, // TOTAL_DISTANCE_IN_SERVICE
0x00,0x00,0x00,0x1E, // TOTAL_NUMBER_TRAVELS
0x00,0x00,0x00,0x28, // TOTAL_NUMBER_INCREMENTs
0x00,0x00,0x00,0x32, // TOTAL_INCOME
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF, // Empty
0x03, // FARES_NUMBER
0x04, // EXTRAS_NUMBER
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF, // Empty
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF, // Empty
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF, // Empty
0xAA // FLASH_PROGRAM
};
Byte accumulators[32]; // This array is where the accumulators are received
Byte empty_line = 0xB0;
Byte fill_line = 0xDB;
Byte carriage_return = 0x0D;
SetComPort(); // Set serial port to 4800bps 8-N-1
delay(1000);
/* Clean receiving buffer */
for (i=0;i<100;i++) {
    temp = Receive();
}

/* Send Handshake */
printf("Handshake: ");
Send (0xAA); // Handshake
while ( (inportb(LSR)&DR) == 0 ) {
    /* Empty Body: Waiting to receive a byte */
}
temp = Receive(); // Receive the 0xAA

/* Receive the accumulators */
printf("sent... Waiting confirmation: ");
for (i=0,j=0; i<32; i++,j++) {
    while ( (inportb(LSR)&DR) == 0 ) {
        /* Empty Body: Waiting to receive a byte */
    }
    if (i==0) {
        printf("CONFIRMED.");
        printf("\n\nReceiving Accumulators: ");
    }
    accumulators[i] = Receive();
}
printf("RECEIVED.");

/* Send the information frames: the tables to be programmed */
printf("\n\nSending Data Tables:   %c%c%c%c%c%c%c%c%c%c%c%c
", empty_line, empty_line, empty_line, empty_line, empty_line, empty_line, empty_line, empty_line, empty_line, empty_line, empty_line, empty_line);
printf("\n\nSending Data Tables:   ", carriage_return);
for (i=0; i<6; i++) { // Send 6 ROWS of data
    /* Send a ROW of data */
    for (j=0; j<32; j++) { // Each ROW is 32 bytes long
        Send(tables[(i*32)+j]); // Send a byte
        while ((inportb(LSR)&DR) == 0) { // Wait to receive the same byte
            /* Empty body */
            }
        tables_back[(i*32)+j] = Receive();
        if (tables_back[(i*32)+j] != tables[(i*32)+j]) { // If what was sent is dif-
            ferent from what is received
            printf("\n\n        There is a problem in the communication\n");
            getch();
            exit(-1);
        }
        }
    /* Receive the ROW just sent (for verification) */
    for (j=0; j<32; j++) { // Receive the ROW just sent
        while ((inportb(LSR)&DR) == 0) { // Wait to receive the same byte
            /* Empty body */
            }
        tables_back[(i*32)+j] = Receive();
        if (tables_back[(i*32)+j] != tables[(i*32)+j]) { // If what was sent is dif-
            ferent from what is received
            printf("\n\n        There is a problem in the communication\n");
            getch();
            exit(-1);
        }
    }
    printf("%c%c", fill_line, fill_line);
}
printf("\n\nOperation was completed successfully.\n");

/*================================================================*/
/*=========================   FUNCTIONS   ========================*/
/*================================================================*/
/**
* SetComPort: Sets the communication port to the needed values:
*  4800 bps, 8 data bits, no parity, 1 stop bit
*Parameters: none.
*Return: void
*/
void SetComPort (void) {
outportb (IER, 0x00); // Clear interrupts
outportb (FCR, 0x01);
outportb (FCR, 0xFF); // Reset receiver and transmitter
outportb (LCR, inportb(LCR)|DLAB); // To configure baud rate divisor
}
outportb (BRMSB, 0x00);        // MSB
outportb (BRLSB, BR4800);     // LSB
outportb (LCR, (inportb(LCR)&(~DLAB)));  // End configuration of baud rate
outportb (LCR, (BITS8|STOP1));  // Configure 8 data-bits, 2 stop-bits, parity space
outportb (FCR, 0xC7);          // Configure 14 bytes RCVR FIFO
outportb (MCR, 0x00);  // No handshake
}

/**
 * Receive: It receives a byte in the serial port and returns it.
 * If no byte is ready, it returns NO_BYTE
 *
 * Parameters: none.
 *
 * Return: the byte received
 */
Byte Receive(void) {
  Byte _rcv;
  Byte _err;
  Byte temp;
  if ((inportb(LSR)&DR)!=0) {
    _err = inportb(LSR);       // Acquire status (before receiving data)
    _rcv = inportb(UART);      // Acquire data received
    /*
     * If (_err & FE) == FE) {   // Verify if framing error has occurred
     *   printf("FE ");        // _rcv |= 0x01;
     * }
     * if ((_err & PE) == PE) {   // Verify if a parity error has occurred
     *   printf("PE ");
     * }
     * if ((_err & OE) == OE) {   // Verify if an overrun error has occurred
     *   printf("OE ");
     * }
     */
    return _rcv;
  }
  return NO_BYTE;
}

/**
 * Send: Sends a byte out the serial port.
 *
 * Parameters: _data: the byte to be sent.
 *
 * Return: void
 */
void Send(Byte _data) {
  while ((inportb(LSR)&THRE) == 0) {
    /* Empty Body; Waiting for Transmitter Holding Register to be empty */
  }
  outportb(UART, _data);
}
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

Information in this document is provided solely to enable system and software implementers to use Motorola products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

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 that 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 and the Stylized M Logo are registered in the US Patent and Trademark Office. All other product or service names are the property of their respective owners. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

© Motorola Inc. 2003