87C196CB Supplement to 8XC196NT User’s Manual
87C196CB Supplement to 8XC196NT User’s Manual

August 2004

Order Number: 272787-003
Information in this document is provided in connection with Intel products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel’s Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The 87C196CB and 8XC196NT microprocessors may contain design defects or errors known as errata which may cause the products to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website at http://www.intel.com.

Copyright © Intel Corporation, 1998, 2004

*Third-party brands and names are the property of their respective owners.
CONTENTS

CHAPTER 1
GUIDE TO THIS MANUAL
1.1 MANUAL CONTENTS ................................................................................................... 1-1
1.2 RELATED DOCUMENTS .............................................................................................. 1-2

CHAPTER 2
ARCHITECTURAL OVERVIEW
2.1 DEVICE FEATURES ..................................................................................................... 2-1
2.2 BLOCK DIAGRAM ......................................................................................................... 2-2
2.3 INTERNAL TIMING........................................................................................................ 2-2

CHAPTER 3
MEMORY PARTITIONS
3.1 MEMORY MAP, SPECIAL-FUNCTION REGISTERS, AND WINDOWING .................. 3-1

CHAPTER 4
STANDARD AND PTS INTERRUPTS
4.1 INTERRUPT SOURCES, VECTORS, AND PRIORITIES ............................................. 4-1

CHAPTER 5
I/O PORTS
5.1 PORT 0 AND EPORT .................................................................................................... 5-1

CHAPTER 6
ANALOG-TO-DIGITAL (A/D) CONVERTER
6.1 ADDITIONAL A/D INPUT CHANNELS ........................................................................ 6-1

CHAPTER 7
CAN SERIAL COMMUNICATIONS CONTROLLER
7.1 CAN FUNCTIONAL OVERVIEW .................................................................................. 7-1
7.2 CAN CONTROLLER SIGNALS AND REGISTERS ..................................................... 7-3
7.3 CAN CONTROLLER OPERATION ................................................................................ 7-4
7.3.1 Address Map .......................................................................................................... 7-5
7.3.2 Message Objects .................................................................................................... 7-5
7.3.2.1 Receive and Transmit Priorities ....................................................................... 7-6
7.3.2.2 Message Acceptance Filtering ......................................................................... 7-6
7.3.3 Message Frames .................................................................................................... 7-7
7.3.4 Error Detection and Management Logic ............................................................... 7-9
7.3.5 Bit Timing ................................................................................................................ 7-10
7.3.5.1 Bit Timing Equations ....................................................................................... 7-12
7.4 CONFIGURING THE CAN CONTROLLER ................................................................. 7-13
7.4.1 Programming the CAN Control (CAN_CON) Register ......................................... 7-13
7.4.2 Programming the Bit Timing 0 (CAN_BTIME0) Register .................................... 7-15
7.4.3 Programming the Bit Timing 1 (CAN_BTIME1) Register .................................... 7-16
### FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>87C196CB Block Diagram</td>
</tr>
<tr>
<td>2-2</td>
<td>Clock Circuitry</td>
</tr>
<tr>
<td>2-3</td>
<td>Internal Clock Phases</td>
</tr>
<tr>
<td>2-4</td>
<td>Effect of Clock Mode on CLKOUT Frequency</td>
</tr>
<tr>
<td>4-1</td>
<td>Interrupt Mask 1 (INT_MASK1) Register</td>
</tr>
<tr>
<td>4-2</td>
<td>interrupt Pending 1 (INT_PEND1) Register</td>
</tr>
<tr>
<td>5-1</td>
<td>Port x Pin Input (Px_PIN) Register</td>
</tr>
<tr>
<td>5-2</td>
<td>Extended Port I/O Direction (EP_DIP) Register</td>
</tr>
<tr>
<td>5-3</td>
<td>Extended Port Mode (EP_MODE) Register</td>
</tr>
<tr>
<td>5-4</td>
<td>Extended Port Input (EP_PIN) Register</td>
</tr>
<tr>
<td>5-5</td>
<td>Extended Port Data Output (EP_REG) Register</td>
</tr>
<tr>
<td>6-1</td>
<td>A/D Command (AD_COMMAND) Register</td>
</tr>
<tr>
<td>6-2</td>
<td>A/D Result (AD_RESULT) Register — Read Format</td>
</tr>
<tr>
<td>7-1</td>
<td>A System Using CAN Controllers</td>
</tr>
<tr>
<td>7-2</td>
<td>CAN Controller Block Diagram</td>
</tr>
<tr>
<td>7-3</td>
<td>CAN Message Frames</td>
</tr>
<tr>
<td>7-4</td>
<td>A Bit Time as Specified by the CAN Protocol</td>
</tr>
<tr>
<td>7-5</td>
<td>A Bit Time as Implemented in the CAN Controller</td>
</tr>
<tr>
<td>7-6</td>
<td>CAN Control (CAN_CON) Register</td>
</tr>
<tr>
<td>7-7</td>
<td>CAN Bit Timing 0 (CAN_BTIME0) Register</td>
</tr>
<tr>
<td>7-8</td>
<td>CAN Bit Timing 1 (CAN_BTIME1) Register</td>
</tr>
<tr>
<td>7-9</td>
<td>CAN Standard Global Mask (CAN_SGMSK) Register</td>
</tr>
<tr>
<td>7-10</td>
<td>CAN Extended Global Mask (CAN_EGMSK) Register</td>
</tr>
<tr>
<td>7-11</td>
<td>CAN Message 15 Mask (CAN_MSK15) Register</td>
</tr>
<tr>
<td>7-12</td>
<td>CAN Message Object x Configuration (CAN_MSGxCFG) Register</td>
</tr>
<tr>
<td>7-13</td>
<td>CAN Message Object x Identifier (CAN_MSGxID0–3) Register</td>
</tr>
<tr>
<td>7-14</td>
<td>CAN Message Object x Control 0 (CAN_MSGxCON0) Register</td>
</tr>
<tr>
<td>7-15</td>
<td>CAN Message Object x Control 1 (CAN_MSGxCON1) Register</td>
</tr>
<tr>
<td>7-16</td>
<td>CAN Message Object Data (CAN_MSGxDATA0–7) Registers</td>
</tr>
<tr>
<td>7-17</td>
<td>CAN Control (CAN_CON) Register</td>
</tr>
<tr>
<td>7-18</td>
<td>CAN Message Object x Control 0 (CAN_MSGxCON0) Register</td>
</tr>
<tr>
<td>7-19</td>
<td>CAN Interrupt Pending (CAN_INT) Register</td>
</tr>
<tr>
<td>7-20</td>
<td>CAN Status (CAN_STAT) Register</td>
</tr>
<tr>
<td>7-21</td>
<td>CAN Message Object x Control 0 (CAN_MSGxCON0) Register</td>
</tr>
<tr>
<td>7-22</td>
<td>Receiving a Message for Message Objects 1–14 — CPU Flow</td>
</tr>
<tr>
<td>7-23</td>
<td>Receiving a Message for Message Object 15 — CPU Flow</td>
</tr>
<tr>
<td>7-24</td>
<td>Receiving a Message — CAN Controller Flow</td>
</tr>
<tr>
<td>7-25</td>
<td>Transmitting a Message — CPU Flow</td>
</tr>
<tr>
<td>7-26</td>
<td>Transmitting a Message — CAN Controller Flow</td>
</tr>
<tr>
<td>8-1</td>
<td>Clock Circuitry</td>
</tr>
<tr>
<td>9-1</td>
<td>Modes 0 and 3 Timings</td>
</tr>
<tr>
<td>9-2</td>
<td>Chip Configuration 1 (CCR1) Register</td>
</tr>
<tr>
<td>10-1</td>
<td>Auto Programming Circuit</td>
</tr>
<tr>
<td>A-1</td>
<td>87C196CB 84-pin PLCC Package</td>
</tr>
</tbody>
</table>
## FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-2</td>
<td>87C196CB 100-pin QFP Package</td>
<td>A-3</td>
</tr>
</tbody>
</table>
Guide to This Manual
CHAPTER 1
GUIDE TO THIS MANUAL

This document is a supplement to the 8XC196NT Microcontroller User’s Manual. It describes the differences between the 87C196CB and the 8XC196NT. For information not found in this supplement, please consult the 8XC196NT Microcontroller User’s Manual (order number 272317) or the 87C196CB datasheet (87C196CA/87C196CB 20 MHz Advanced 16-Bit CHMOS Microcontroller with Integrated CAN 2.0, order number 272405).

1.1 MANUAL CONTENTS

This supplement contains several chapters, an appendix, a glossary, and an index. This chapter, Chapter 1, provides an overview of the supplement. This section summarizes the contents of the remaining chapters and appendixes. The remainder of this chapter provides references to related documentation.

Chapter 2 — Architectural Overview — compares the features of the 87C196CB with those of the 8XC196NT and describes the 87C196CB’s internal clock circuitry.

Chapter 3 — Memory Partitions — describes the addressable memory space of the 84-pin and 100-pin 87C196CB, lists the peripheral special-function registers (SFRs), and provides tables of WSR values for windowing higher memory into the lower register file for direct access.

Chapter 4 — Standard and PTS Interrupts — describes the additional interrupts for the CAN (controller area network) peripheral and the SFRs that support those interrupts.

Chapter 5 — I/O Ports — describes the port 0 and EPORT differences for the 100-pin 87C196CB. Both port 0 and the EPORT are implemented as eight-bit ports on the 100-pin 87C196CB, but as four-bit ports (like the 8XC196NT) on the 84-pin 87C196CB.

Chapter 6 — Analog-to-digital (A/D) Converter — illustrates the SFRs that are affected by the implementation of port 0 as an eight-bit port.

Chapter 7 — CAN Serial Communications Controller — describes the 87C196CB’s integrated CAN controller and explains how to configure it. This integrated peripheral is similar to Intel’s standalone 82527 CAN serial communications controller, supporting both the standard and extended message frames specified by the CAN 2.0 protocol parts A and B.

Chapter 8 — Special Operating Modes — illustrates the clock control circuitry of the 87C196CB.
Chapter 9 — Interfacing with External Memory — discusses differences in the bus timing modes supported by the 8XC196NT and the 87C196CB.

Chapter 10 — Programming the Nonvolatile Memory — describes the memory maps and recommended circuits to support programming of the 87C196CB’s 56 Kbytes of OTPROM.

Appendix A — Signal Descriptions — describes the additional signals implemented on the 87C196CB.

Glossary — defines terms with special meaning used throughout this supplement.

Index — lists key topics with page number references.

1.2 RELATED DOCUMENTS

Table 1-1 lists additional documents that you may find useful in designing systems incorporating the 87C196CB microcontroller.

<table>
<thead>
<tr>
<th>Title and Description</th>
<th>Order Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>8XC196NT Microcontroller User’s Manual</td>
<td>272317</td>
</tr>
<tr>
<td>Automotive Products handbook</td>
<td>231792</td>
</tr>
<tr>
<td>87C196CB 20 MHz Advanced 16-Bit CMOS Microcontroller with Integrated CAN 2.0 (datasheet)</td>
<td>272405</td>
</tr>
</tbody>
</table>
Architectural Overview
CHAPTER 2
ARCHITECTURAL OVERVIEW

This chapter describes architectural differences between the 87C196CB and the 8XC196NT. Both the 8XC196NT and the 87C196CB are designed for high-speed calculations and fast I/O. With the addition of the CAN (controller area network) peripheral, the 87C196CB reduces point-to-point wiring requirements, making it well-suited to automotive and factory automation applications.

The 87C196CB is available in either an 84-pin or a 100-pin package. The 84-pin 87C196CB, like the 8XC196NT, has up to 20 external address lines, enabling access to 1 Mbyte of linear address space. The 100-pin 87C196CB has four additional pins available for external address lines. With all 24 external address lines connected, the 100-pin 87C196CB can access 16 Mbytes of linear address space.

2.1 DEVICE FEATURES

Table 2-1 lists the features of the 8XC196NT and the 87C196CB. The 87C196CB implements more OTPROM, more register RAM, four additional A/D channels, and the CAN peripheral. The 100-pin 87C196CB also implements four additional EPORT pins.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8XC196NT</td>
<td>68</td>
<td>0 or 32 K</td>
<td>1 K</td>
<td>512</td>
<td>56</td>
<td>10</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>87C196CB</td>
<td>84</td>
<td>56 K</td>
<td>1.5 K</td>
<td>512</td>
<td>56</td>
<td>10</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>87C196CB</td>
<td>100</td>
<td>56 K</td>
<td>1.5 K</td>
<td>512</td>
<td>60</td>
<td>10</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

* Register RAM amount includes the 24 bytes allocated to the core SFRs and stack pointer.
2.2 BLOCK DIAGRAM

Figure 2-1 shows the major blocks within the device. The 8XC196NT and 87C196CB have the same peripheral set with the exception of the CAN (controller area network) peripheral, which is unique to the 87C196CB. The CAN peripheral manages communications between multiple network nodes. This integrated peripheral is similar to Intel’s standalone 82527 CAN serial communications controller, supporting both the standard and extended message frames specified by the CAN 2.0 protocol parts A and B.

![Figure 2-1. 87C196CB Block Diagram](image)

2.3 INTERNAL TIMING

The 87C196CB’s clock circuitry (Figure 2-2) implements phase-locked loop and clock multiplier circuitry, which can substantially increase the CPU clock rate while using a lower-frequency input clock. The clock circuitry accepts an input clock signal on XTAL1 provided by an external crystal or oscillator. Depending on the value of the PLLEN pin, this frequency is routed either through the phase-locked loop and multiplier or directly to the divide-by-two circuit. The multiplier circuitry can quadruple the input frequency ($F_{XTAL1}$) before the frequency ($f$) reaches the divide-by-two circuitry. The clock generators accept the divided input frequency ($f/2$) from the divide-by-two circuit and produce two nonoverlapping internal timing signals, PH1 and PH2. These signals are active when high.

**NOTE**

This manual uses lowercase “$f$” to represent the internal clock frequency. For the 87C196CB, $f$ is equal to either $F_{XTAL1}$ or $4F_{XTAL1}$, depending on the clock multiplier mode, which is controlled by the PLLEN input pin.
The rising edges of PH1 and PH2 generate the internal CLKOUT signal (Figure 2-3). The clock circuitry routes separate internal clock signals to the CPU and the peripherals to provide flexibility in power management. It also outputs the CLKOUT signal on the CLKOUT pin. Because of the complex logic in the clock circuitry, the signal on the CLKOUT pin is a delayed version of the internal CLKOUT signal. This delay varies with temperature and voltage.
Figure 2-3. Internal Clock Phases

The combined period of phase 1 and phase 2 of the internal CLKOUT signal defines the basic time unit known as a state time or state. Table 2-2 lists state time durations at various frequencies.

Table 2-2. State Times at Various Frequencies

<table>
<thead>
<tr>
<th>f (Frequency Input to the Divide-by-two Circuit)</th>
<th>State Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 MHz</td>
<td>250 ns</td>
</tr>
<tr>
<td>12 MHz</td>
<td>167 ns</td>
</tr>
<tr>
<td>16 MHz</td>
<td>125 ns</td>
</tr>
<tr>
<td>20 MHz</td>
<td>100 ns</td>
</tr>
</tbody>
</table>

The following formulas calculate the frequency of PH1 and PH2, the duration of a state time, and the duration of a clock period (t).

\[
PH1 \text{ (in MHz)} = \frac{f}{2} = PH2 \\
\text{State Time (in µs)} = \frac{2}{f} \\
t = \frac{1}{f}
\]

Because the device can operate at many frequencies, this manual defines time requirements (such as instruction execution times) in terms of state times rather than specific measurements. Datasheets list AC characteristics in terms of clock periods (t; sometimes called T\text{osc}).

Figure 2-4 illustrates the timing relationships between the input frequency (F\text{XTAL1}), the operating frequency (f), and the CLKOUT signal with each PLLEN pin configuration. Table 2-3 details the relationships between the input frequency (F\text{XTAL1}), the PLLEN pin, the operating frequency (f), the clock period (t), and state times.
Table 2-3. Relationships Between Input Frequency, Clock Multiplier, and State Times

<table>
<thead>
<tr>
<th>$F_{XTAL}$ (Frequency on XTAL1)</th>
<th>PLLEN</th>
<th>Multiplier</th>
<th>( f ) (Input Frequency to the Divide-by-two Circuit)</th>
<th>( t ) (Clock Period)</th>
<th>State Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 MHz</td>
<td>0</td>
<td>1</td>
<td>4 MHz</td>
<td>250 ns</td>
<td>500 ns</td>
</tr>
<tr>
<td>5 MHz</td>
<td>0</td>
<td>1</td>
<td>5 MHz</td>
<td>200 ns</td>
<td>400 ns</td>
</tr>
<tr>
<td>8 MHz</td>
<td>0</td>
<td>1</td>
<td>8 MHz</td>
<td>125 ns</td>
<td>250 ns</td>
</tr>
<tr>
<td>12 MHz</td>
<td>0</td>
<td>1</td>
<td>12 MHz</td>
<td>83.5 ns</td>
<td>167 ns</td>
</tr>
<tr>
<td>16 MHz</td>
<td>0</td>
<td>1</td>
<td>16 MHz</td>
<td>62.5 ns</td>
<td>125 ns</td>
</tr>
<tr>
<td>20 MHz</td>
<td>0</td>
<td>1</td>
<td>20 MHz</td>
<td>50 ns</td>
<td>100 ns</td>
</tr>
<tr>
<td>4 MHz</td>
<td>1</td>
<td>4</td>
<td>16 MHz</td>
<td>62.5 ns</td>
<td>125 ns</td>
</tr>
<tr>
<td>5 MHz</td>
<td>1</td>
<td>4</td>
<td>20 MHz</td>
<td>50 ns</td>
<td>100 ns</td>
</tr>
</tbody>
</table>
Memory Partitions
CHAPTER 3
MEMORY PARTITIONS

This chapter describes the differences in the address space of the 87C196CB from that of the 8XC196NT. The 87C196CB has 56 Kbytes of one-time-programmable read-only memory (OTPROM), while the 8XC196NT is available with 32 Kbytes. The 87C196CB also has an additional 512 bytes of register RAM.

The 87C196CB is available in either an 84-pin or a 100-pin package. The 84-pin 87C196CB, like the 8XC196NT, has up to 20 external address lines, enabling access to 1 Mbyte of linear address space. The 100-pin 87C196CB has four additional pins available for external address lines. With all 24 external address lines connected (A23:16 and AD15:0), the 100-pin 87C196CB can access 16 Mbytes of linear address space.

3.1 MEMORY MAP, SPECIAL-FUNCTION REGISTERS, AND WINDOWING

Table 3-1 compares the register file addresses of the 8XC196NT and 87C196CB. Table 3-2 is a memory map of the 87C196CB. Table 3-3 lists the 87C196CB’s peripheral SFRs (these are the same as those of the 8XC196NT). Table 3-4 lists the CAN peripheral SFRs, which are unique to the 87C196CB. Tables 3-5 through 3-9 provide the information necessary to window higher memory into the lower register file for direct access.

<table>
<thead>
<tr>
<th>Device and Hex Address Range</th>
<th>Description</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CB 1DFF</td>
<td>Register RAM</td>
<td>Indirect, indexed, or windowed direct</td>
</tr>
<tr>
<td>1C00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03FF 03FF</td>
<td>Upper register file (register RAM)</td>
<td>Indirect, indexed, or windowed direct</td>
</tr>
<tr>
<td>0100 0100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00FF 00FF</td>
<td>Upper register file (register RAM)</td>
<td>Indirect, indirect, or indexed</td>
</tr>
<tr>
<td>001A 001A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0019 0018</td>
<td>Lower register file (stack pointer)</td>
<td>Direct, indirect, or indexed</td>
</tr>
<tr>
<td>0000 0000</td>
<td>Lower register file (CPU SFRs)</td>
<td>Direct, indirect, or indexed</td>
</tr>
</tbody>
</table>
Table 3-2. 87C196CB Memory Map

<table>
<thead>
<tr>
<th>Hex Address</th>
<th>Description</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFFFFF</td>
<td>Program memory (After a device reset, the first instruction fetch is from FF2080H) †</td>
<td>Indirect, indexed, extended</td>
</tr>
<tr>
<td>FF207F</td>
<td>Special purpose memory †</td>
<td>Indirect, indexed, extended</td>
</tr>
<tr>
<td>FF1FF</td>
<td>External device (memory or I/O) connected to address/data bus</td>
<td>Indirect, indexed, extended</td>
</tr>
<tr>
<td>FF05FF</td>
<td>Internal code and data RAM (mapped identically into pages FFH and 00H)</td>
<td>Indirect, indexed, extended</td>
</tr>
<tr>
<td>FF03FF</td>
<td>External device (memory or I/O) connected to address/data bus</td>
<td>Indirect, indexed, extended</td>
</tr>
<tr>
<td>FF00FF</td>
<td>Reserved ††</td>
<td>Indirect, indexed, extended</td>
</tr>
<tr>
<td>FEFFFF</td>
<td>100-pin 87C196CB: External device (memory or I/O) 84-pin 87C196CB: Overlaid memory ††</td>
<td>Indirect, indexed, extended</td>
</tr>
<tr>
<td>0EFFFF</td>
<td>External device (memory or I/O) connected to address/data bus</td>
<td>Indirect, indexed, extended</td>
</tr>
<tr>
<td>00FFFF</td>
<td>External device or remapped OTPROM †††</td>
<td>Indirect, indexed, extended</td>
</tr>
<tr>
<td>0011FF</td>
<td>Memory-mapped SFRs</td>
<td>Indirect, indexed, extended</td>
</tr>
<tr>
<td>001DFD</td>
<td>Peripheral SFRs</td>
<td>Indirect, indexed, extended, windowed direct</td>
</tr>
<tr>
<td>001EFF</td>
<td>CAN SFRs</td>
<td>Indirect, indexed, extended</td>
</tr>
<tr>
<td>001DFF</td>
<td>Internal register RAM</td>
<td>Indirect, indexed, windowed direct</td>
</tr>
<tr>
<td>001BFF</td>
<td>External device (memory or I/O) connected to address/data bus; future SFR expansion</td>
<td>Indirect, indexed, extended</td>
</tr>
<tr>
<td>0005FF</td>
<td>Internal code and data RAM (mapped identically into pages 00H and FFH)</td>
<td>Indirect, indexed, extended</td>
</tr>
<tr>
<td>0003FF</td>
<td>Upper register file (register RAM)</td>
<td>Indirect, indexed, windowed direct</td>
</tr>
<tr>
<td>0000FF</td>
<td>Lower register file (register RAM, stack pointer, CPU SFRs)</td>
<td>Direct, indirect, indexed</td>
</tr>
</tbody>
</table>

† For the 87C196CB, the program and special-purpose memory locations (FF2000-FFFFFFH) can reside either in external memory or in internal OTPROM.

†† Locations xF0000-xF00FFH are reserved for in-circuit emulators. Do not use these locations except to initialize them. Except as otherwise noted, initialize unused program memory locations and reserved memory locations to FFH.

††† These locations can be either external memory (CCB2.2=0) or a copy of the OTPROM (CCB2.2=1).
### Table 3-3. 87C196CB Peripheral SFRs

<table>
<thead>
<tr>
<th>Address</th>
<th>High (Odd) Byte</th>
<th>Low (Even) Byte</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1FDEH</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDC0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD4H</td>
<td>P6_REG</td>
<td>P1_REG</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>P6_PIN</td>
<td>P1_PIN</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD9H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDAH</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDAAH</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDBH</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDC0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD6H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FD8H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
<tr>
<td>1FDB0H</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Ports 0, 1, 2, and 6 SFRs</td>
</tr>
</tbody>
</table>
Table 3-4. CAN Peripheral SFRs

<table>
<thead>
<tr>
<th>Message 15</th>
<th>Addr</th>
<th>High (Odd) Byte</th>
<th>Low (Even) Byte</th>
<th>Message 11</th>
<th>Addr</th>
<th>High (Odd) Byte</th>
<th>Low (Even) Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1EF6H</td>
<td>Reserved</td>
<td>CAN_MSG15DATA7</td>
<td></td>
<td>1EB6H</td>
<td>Reserved</td>
<td>CAN_MSG11DATA7</td>
</tr>
<tr>
<td></td>
<td>1EFCH</td>
<td>CAN_MSG15DATA6</td>
<td>CAN_MSG15DATA5</td>
<td></td>
<td>1EBCH</td>
<td>CAN_MSG11DATA6</td>
<td>CAN_MSG11DATA5</td>
</tr>
<tr>
<td></td>
<td>1EFAH</td>
<td>CAN_MSG15DATA4</td>
<td>CAN_MSG15DATA3</td>
<td></td>
<td>1EBAH</td>
<td>CAN_MSG11DATA4</td>
<td>CAN_MSG11DATA3</td>
</tr>
<tr>
<td></td>
<td>1EF8H</td>
<td>CAN_MSG15DATA2</td>
<td>CAN_MSG15DATA1</td>
<td></td>
<td>1EB8H</td>
<td>CAN_MSG11DATA2</td>
<td>CAN_MSG11DATA1</td>
</tr>
<tr>
<td></td>
<td>1EF6H</td>
<td>CAN_MSG15DATA0</td>
<td>CAN_MSG15CFG</td>
<td></td>
<td>1EB6H</td>
<td>CAN_MSG11DATA0</td>
<td>CAN_MSG11CFG</td>
</tr>
<tr>
<td></td>
<td>1EF4H</td>
<td>CAN_MSG15D3</td>
<td>CAN_MSG15ID2</td>
<td></td>
<td>1EB4H</td>
<td>CAN_MSG11D3</td>
<td>CAN_MSG11ID2</td>
</tr>
<tr>
<td></td>
<td>1EC2H</td>
<td>CAN_MSG15D1</td>
<td>CAN_MSG15ID0</td>
<td></td>
<td>1EB2H</td>
<td>CAN_MSG11D1</td>
<td>CAN_MSG11ID0</td>
</tr>
<tr>
<td></td>
<td>1E0H</td>
<td>CAN_MSG15CON1</td>
<td>CAN_MSG15CON0</td>
<td></td>
<td>1E0H</td>
<td>CAN_MSG11CON1</td>
<td>CAN_MSG11CON0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Message 14</th>
<th>Addr</th>
<th>High (Odd) Byte</th>
<th>Low (Even) Byte</th>
<th>Message 10</th>
<th>Addr</th>
<th>High (Odd) Byte</th>
<th>Low (Even) Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1EEEH</td>
<td>Reserved</td>
<td>CAN_MSG14DATA7</td>
<td></td>
<td>1EAEH</td>
<td>Reserved</td>
<td>CAN_MSG10DATA7</td>
</tr>
<tr>
<td></td>
<td>1EECH</td>
<td>CAN_MSG14DATA6</td>
<td>CAN_MSG14DATA5</td>
<td></td>
<td>1EACH</td>
<td>CAN_MSG10DATA6</td>
<td>CAN_MSG10DATA5</td>
</tr>
<tr>
<td></td>
<td>1EAAH</td>
<td>CAN_MSG14DATA4</td>
<td>CAN_MSG14DATA3</td>
<td></td>
<td>1EAAH</td>
<td>CAN_MSG10DATA4</td>
<td>CAN_MSG10DATA3</td>
</tr>
<tr>
<td></td>
<td>1EE8H</td>
<td>CAN_MSG14DATA2</td>
<td>CAN_MSG14DATA1</td>
<td></td>
<td>1EE8H</td>
<td>CAN_MSG10DATA2</td>
<td>CAN_MSG10DATA1</td>
</tr>
<tr>
<td></td>
<td>1EE6H</td>
<td>CAN_MSG14DATA0</td>
<td>CAN_MSG14CFG</td>
<td></td>
<td>1EE6H</td>
<td>CAN_MSG10DATA0</td>
<td>CAN_MSG10CFG</td>
</tr>
<tr>
<td></td>
<td>1EE2H</td>
<td>CAN_MSG14D1</td>
<td>CAN_MSG14ID0</td>
<td></td>
<td>1EE2H</td>
<td>CAN_MSG10D1</td>
<td>CAN_MSG10ID0</td>
</tr>
<tr>
<td></td>
<td>1EOH</td>
<td>CAN_MSG14CON1</td>
<td>CAN_MSG14CON0</td>
<td></td>
<td>1EOH</td>
<td>CAN_MSG10CON1</td>
<td>CAN_MSG10CON0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Message 13</th>
<th>Addr</th>
<th>High (Odd) Byte</th>
<th>Low (Even) Byte</th>
<th>Message 9</th>
<th>Addr</th>
<th>High (Odd) Byte</th>
<th>Low (Even) Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1DEEH</td>
<td>Reserved</td>
<td>CAN_MSG13DATA7</td>
<td></td>
<td>1E9EH</td>
<td>Reserved</td>
<td>CAN_MSG9DATA7</td>
</tr>
<tr>
<td></td>
<td>1DCH</td>
<td>CAN_MSG13DATA6</td>
<td>CAN_MSG13DATA5</td>
<td></td>
<td>1E9CH</td>
<td>CAN_MSG9DATA6</td>
<td>CAN_MSG9DATA5</td>
</tr>
<tr>
<td></td>
<td>1DAH</td>
<td>CAN_MSG13DATA4</td>
<td>CAN_MSG13DATA3</td>
<td></td>
<td>1E9AH</td>
<td>CAN_MSG9DATA4</td>
<td>CAN_MSG9DATA3</td>
</tr>
<tr>
<td></td>
<td>1D8H</td>
<td>CAN_MSG13DATA2</td>
<td>CAN_MSG13DATA1</td>
<td></td>
<td>1E98H</td>
<td>CAN_MSG9DATA2</td>
<td>CAN_MSG9DATA1</td>
</tr>
<tr>
<td></td>
<td>1D6H</td>
<td>CAN_MSG13DATA0</td>
<td>CAN_MSG13CFG</td>
<td></td>
<td>1E96H</td>
<td>CAN_MSG9DATA0</td>
<td>CAN_MSG9CFG</td>
</tr>
<tr>
<td></td>
<td>1D4H</td>
<td>CAN_MSG13D3</td>
<td>CAN_MSG13D2</td>
<td></td>
<td>1E94H</td>
<td>CAN_MSG9D3</td>
<td>CAN_MSG9D2</td>
</tr>
<tr>
<td></td>
<td>1D2H</td>
<td>CAN_MSG13D1</td>
<td>CAN_MSG13D0</td>
<td></td>
<td>1E92H</td>
<td>CAN_MSG9D1</td>
<td>CAN_MSG9D0</td>
</tr>
<tr>
<td></td>
<td>1D0H</td>
<td>CAN_MSG13CON1</td>
<td>CAN_MSG13CON0</td>
<td></td>
<td>1E90H</td>
<td>CAN_MSG9CON1</td>
<td>CAN_MSG9CON0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Message 12</th>
<th>Addr</th>
<th>High (Odd) Byte</th>
<th>Low (Even) Byte</th>
<th>Message 8</th>
<th>Addr</th>
<th>High (Odd) Byte</th>
<th>Low (Even) Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1CEEH</td>
<td>Reserved</td>
<td>CAN_MSG12DATA7</td>
<td></td>
<td>1E8EH</td>
<td>Reserved</td>
<td>CAN_MSG8DATA7</td>
</tr>
<tr>
<td></td>
<td>1CECH</td>
<td>CAN_MSG12DATA6</td>
<td>CAN_MSG12DATA5</td>
<td></td>
<td>1E8CH</td>
<td>CAN_MSG8DATA6</td>
<td>CAN_MSG8DATA5</td>
</tr>
<tr>
<td></td>
<td>1CAH</td>
<td>CAN_MSG12DATA4</td>
<td>CAN_MSG12DATA3</td>
<td></td>
<td>1E8AH</td>
<td>CAN_MSG8DATA4</td>
<td>CAN_MSG8DATA3</td>
</tr>
<tr>
<td></td>
<td>1C8H</td>
<td>CAN_MSG12DATA2</td>
<td>CAN_MSG12DATA1</td>
<td></td>
<td>1E88H</td>
<td>CAN_MSG8DATA2</td>
<td>CAN_MSG8DATA1</td>
</tr>
<tr>
<td></td>
<td>1C6H</td>
<td>CAN_MSG12DATA0</td>
<td>CAN_MSG12CFG</td>
<td></td>
<td>1E86H</td>
<td>CAN_MSG8DATA0</td>
<td>CAN_MSG8CFG</td>
</tr>
<tr>
<td></td>
<td>1C4H</td>
<td>CAN_MSG12D3</td>
<td>CAN_MSG12D2</td>
<td></td>
<td>1E84H</td>
<td>CAN_MSG8D3</td>
<td>CAN_MSG8D2</td>
</tr>
<tr>
<td></td>
<td>1C2H</td>
<td>CAN_MSG12D1</td>
<td>CAN_MSG12D0</td>
<td></td>
<td>1E82H</td>
<td>CAN_MSG8D1</td>
<td>CAN_MSG8D0</td>
</tr>
<tr>
<td></td>
<td>1COH</td>
<td>CAN_MSG12CON1</td>
<td>CAN_MSG12CON0</td>
<td></td>
<td>1E80H</td>
<td>CAN_MSG8CON1</td>
<td>CAN_MSG8CON0</td>
</tr>
</tbody>
</table>
### Table 3-4. CAN Peripheral SFRs (Continued)

#### Message 7

<table>
<thead>
<tr>
<th>Addr</th>
<th>High (Odd) Byte</th>
<th>Low (Even) Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1E7EH</td>
<td>Reserved</td>
<td>CAN_MSG7DATA7</td>
</tr>
<tr>
<td>1E7CH</td>
<td>CAN_MSG7DATA6</td>
<td>CAN_MSG7DATA5</td>
</tr>
<tr>
<td>1E7AH</td>
<td>CAN_MSG7DATA4</td>
<td>CAN_MSG7DATA3</td>
</tr>
<tr>
<td>1E78H</td>
<td>CAN_MSG7DATA2</td>
<td>CAN_MSG7DATA1</td>
</tr>
<tr>
<td>1E76H</td>
<td>CAN_MSG7DATA0</td>
<td>CAN_MSG7CFG</td>
</tr>
<tr>
<td>1E74H</td>
<td>CAN_MSG7ID3</td>
<td>CAN_MSG7ID2</td>
</tr>
<tr>
<td>1E72H</td>
<td>CAN_MSG7ID1</td>
<td>CAN_MSG7ID0</td>
</tr>
<tr>
<td>1E70H</td>
<td>CAN_MSG7CON1</td>
<td>CAN_MSG7CON0</td>
</tr>
</tbody>
</table>

#### Message 3 and Bit Timing 0

<table>
<thead>
<tr>
<th>Addr</th>
<th>High (Odd) Byte</th>
<th>Low (Even) Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1E3EH</td>
<td>CAN_BTIME0†</td>
<td>CAN_MSG3DATA7</td>
</tr>
<tr>
<td>1E3CH</td>
<td>CAN_MSG3DATA6</td>
<td>CAN_MSG3DATA5</td>
</tr>
<tr>
<td>1E3AH</td>
<td>CAN_MSG3DATA4</td>
<td>CAN_MSG3DATA3</td>
</tr>
<tr>
<td>1E38H</td>
<td>CAN_MSG3DATA2</td>
<td>CAN_MSG3DATA1</td>
</tr>
<tr>
<td>1E36H</td>
<td>CAN_MSG3DATA0</td>
<td>CAN_MSG3CFG</td>
</tr>
<tr>
<td>1E34H</td>
<td>CAN_MSG3ID3</td>
<td>CAN_MSG3ID2</td>
</tr>
<tr>
<td>1E32H</td>
<td>CAN_MSG3ID1</td>
<td>CAN_MSG3ID0</td>
</tr>
<tr>
<td>1E30H</td>
<td>CAN_MSG3CON1</td>
<td>CAN_MSG3CON0</td>
</tr>
</tbody>
</table>

#### Message 6

<table>
<thead>
<tr>
<th>Addr</th>
<th>High (Odd) Byte</th>
<th>Low (Even) Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1E6EH</td>
<td>Reserved</td>
<td>CAN_MSG6DATA7</td>
</tr>
<tr>
<td>1E6CH</td>
<td>CAN_MSG6DATA6</td>
<td>CAN_MSG6DATA5</td>
</tr>
<tr>
<td>1E6AH</td>
<td>CAN_MSG6DATA4</td>
<td>CAN_MSG6DATA3</td>
</tr>
<tr>
<td>1E68H</td>
<td>CAN_MSG6DATA2</td>
<td>CAN_MSG6DATA1</td>
</tr>
<tr>
<td>1E66H</td>
<td>CAN_MSG6DATA0</td>
<td>CAN_MSG6CFG</td>
</tr>
<tr>
<td>1E64H</td>
<td>CAN_MSG6ID3</td>
<td>CAN_MSG6ID2</td>
</tr>
<tr>
<td>1E62H</td>
<td>CAN_MSG6ID1</td>
<td>CAN_MSG6ID0</td>
</tr>
<tr>
<td>1E60H</td>
<td>CAN_MSG6CON1</td>
<td>CAN_MSG6CON0</td>
</tr>
</tbody>
</table>

#### Message 2

<table>
<thead>
<tr>
<th>Addr</th>
<th>High (Odd) Byte</th>
<th>Low (Even) Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1E2EH</td>
<td>Reserved</td>
<td>CAN_MSG2DATA7</td>
</tr>
<tr>
<td>1E2CH</td>
<td>CAN_MSG2DATA6</td>
<td>CAN_MSG2DATA5</td>
</tr>
<tr>
<td>1E2AH</td>
<td>CAN_MSG2DATA4</td>
<td>CAN_MSG2DATA3</td>
</tr>
<tr>
<td>1E28H</td>
<td>CAN_MSG2DATA2</td>
<td>CAN_MSG2DATA1</td>
</tr>
<tr>
<td>1E26H</td>
<td>CAN_MSG2DATA0</td>
<td>CAN_MSG2CFG</td>
</tr>
<tr>
<td>1E24H</td>
<td>CAN_MSG2ID3</td>
<td>CAN_MSG2ID2</td>
</tr>
<tr>
<td>1E22H</td>
<td>CAN_MSG2ID1</td>
<td>CAN_MSG2ID0</td>
</tr>
<tr>
<td>1E20H</td>
<td>CAN_MSG2CON1</td>
<td>CAN_MSG2CON0</td>
</tr>
</tbody>
</table>

#### Message 5 and Interrupts

<table>
<thead>
<tr>
<th>Addr</th>
<th>High (Odd) Byte</th>
<th>Low (Even) Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1E5EH</td>
<td>CAN_INT</td>
<td>CAN_MSG5DATA7</td>
</tr>
<tr>
<td>1E5CH</td>
<td>CAN_MSG5DATA6</td>
<td>CAN_MSG5DATA5</td>
</tr>
<tr>
<td>1E5AH</td>
<td>CAN_MSG5DATA4</td>
<td>CAN_MSG5DATA3</td>
</tr>
<tr>
<td>1E58H</td>
<td>CAN_MSG5DATA2</td>
<td>CAN_MSG5DATA1</td>
</tr>
<tr>
<td>1E56H</td>
<td>CAN_MSG5DATA0</td>
<td>CAN_MSG5CFG</td>
</tr>
<tr>
<td>1E54H</td>
<td>CAN_MSG5ID3</td>
<td>CAN_MSG5ID2</td>
</tr>
<tr>
<td>1E52H</td>
<td>CAN_MSG5ID1</td>
<td>CAN_MSG5ID0</td>
</tr>
<tr>
<td>1E50H</td>
<td>CAN_MSG5CON1</td>
<td>CAN_MSG5CON0</td>
</tr>
</tbody>
</table>

#### Message 4 and Bit Timing 1

<table>
<thead>
<tr>
<th>Addr</th>
<th>High (Odd) Byte</th>
<th>Low (Even) Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1E4EH</td>
<td>CAN_BTIME1†</td>
<td>CAN_MSG4DATA7</td>
</tr>
<tr>
<td>1E4CH</td>
<td>CAN_MSG4DATA6</td>
<td>CAN_MSG4DATA5</td>
</tr>
<tr>
<td>1E4AH</td>
<td>CAN_MSG4DATA4</td>
<td>CAN_MSG4DATA3</td>
</tr>
<tr>
<td>1E48H</td>
<td>CAN_MSG4DATA2</td>
<td>CAN_MSG4DATA1</td>
</tr>
<tr>
<td>1E46H</td>
<td>CAN_MSG4DATA0</td>
<td>CAN_MSG4CFG</td>
</tr>
<tr>
<td>1E44H</td>
<td>CAN_MSG4ID3</td>
<td>CAN_MSG4ID2</td>
</tr>
<tr>
<td>1E42H</td>
<td>CAN_MSG4ID1</td>
<td>CAN_MSG4ID0</td>
</tr>
<tr>
<td>1E40H</td>
<td>CAN_MSG4CON1</td>
<td>CAN_MSG4CON0</td>
</tr>
</tbody>
</table>

### Mask, Control, and Status

<table>
<thead>
<tr>
<th>Addr</th>
<th>High (Odd) Byte</th>
<th>Low (Even) Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1E0EH</td>
<td>CAN_MSK15</td>
<td>CAN_MSK15</td>
</tr>
<tr>
<td>1E0CH</td>
<td>CAN_MSK15</td>
<td>CAN_MSK15</td>
</tr>
<tr>
<td>1E0AH</td>
<td>CAN_EGMSK</td>
<td>CAN_EGMSK</td>
</tr>
<tr>
<td>1E08H</td>
<td>CAN_EGMSK</td>
<td>CAN_EGMSK</td>
</tr>
<tr>
<td>1E06H</td>
<td>CAN_SGMSK</td>
<td>CAN_SGMSK</td>
</tr>
<tr>
<td>1E04H</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>1E02H</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>1E00H</td>
<td>CAN_STAT</td>
<td>CAN_CON†</td>
</tr>
</tbody>
</table>

† The CCE bit in the control register (CAN_CON) must be set to enable write access to the bit timing registers (CAN_BTIME0 and CAN_BTIME1).
Table 3-5. Selecting a Window of Peripheral SFRs

<table>
<thead>
<tr>
<th>Peripheral</th>
<th>WSR Value for 32-byte Window (00E0–00FFH)</th>
<th>WSR Value for 64-byte Window (00C0–00FFH)</th>
<th>WSR Value for 128-byte Window (0080–00FFH)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ports 0, 1, 2, 6</td>
<td>7EH</td>
<td>3FH</td>
<td>1FH</td>
</tr>
<tr>
<td>A/D converter, EPA interrupts</td>
<td>7DH</td>
<td>3EH</td>
<td></td>
</tr>
<tr>
<td>EPA compare 0–1, capture/compare 8–9, timers</td>
<td>7CH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPA capture/compare 0–7</td>
<td>7BH</td>
<td>3DH</td>
<td>1EH</td>
</tr>
<tr>
<td>CAN messages 14–15</td>
<td>77H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN messages 12–13</td>
<td>76H</td>
<td>3BH</td>
<td>1DH</td>
</tr>
<tr>
<td>CAN messages 10–11</td>
<td>75H</td>
<td>3AH</td>
<td></td>
</tr>
<tr>
<td>CAN messages 8–9</td>
<td>74H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN messages 6–7</td>
<td>73H</td>
<td>39H</td>
<td>1CH</td>
</tr>
<tr>
<td>CAN messages 4–5, bit timing 1, interrupts</td>
<td>72H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN messages 2–3, bit timing 0</td>
<td>71H</td>
<td>38H</td>
<td></td>
</tr>
<tr>
<td>CAN message 1, control, status, mask</td>
<td>70H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Table 3-6. Selecting a Window of the Upper Register File

<table>
<thead>
<tr>
<th>Register RAM Locations</th>
<th>WSR Value for 32-byte Window (00E0–00FFH)</th>
<th>WSR Value for 64-byte Window (00C0–00FFH)</th>
<th>WSR Value for 128-byte Window (0080–00FFH)</th>
</tr>
</thead>
<tbody>
<tr>
<td>03E0–03FFH</td>
<td>5FH</td>
<td>2FH</td>
<td>17H</td>
</tr>
<tr>
<td>03C0–03DFH</td>
<td>5EH</td>
<td>2EH</td>
<td></td>
</tr>
<tr>
<td>03A0–03BFH</td>
<td>5DH</td>
<td>2EH</td>
<td></td>
</tr>
<tr>
<td>0380–039FH</td>
<td>5CH</td>
<td>2EH</td>
<td></td>
</tr>
<tr>
<td>0360–037FH</td>
<td>5BH</td>
<td>2DH</td>
<td>16H</td>
</tr>
<tr>
<td>0340–035FH</td>
<td>5AH</td>
<td>2CH</td>
<td></td>
</tr>
<tr>
<td>0320–033FH</td>
<td>59H</td>
<td>2CH</td>
<td></td>
</tr>
<tr>
<td>0300–031FH</td>
<td>58H</td>
<td>2CH</td>
<td></td>
</tr>
<tr>
<td>02E0–02FFH</td>
<td>57H</td>
<td>2CH</td>
<td></td>
</tr>
<tr>
<td>02C0–02DFH</td>
<td>56H</td>
<td>2CH</td>
<td></td>
</tr>
<tr>
<td>02A0–02BFH</td>
<td>55H</td>
<td>2CH</td>
<td></td>
</tr>
<tr>
<td>0280–029FH</td>
<td>54H</td>
<td>2AH</td>
<td>15H</td>
</tr>
<tr>
<td>0260–027FH</td>
<td>53H</td>
<td>2AH</td>
<td></td>
</tr>
<tr>
<td>0240–025FH</td>
<td>52H</td>
<td>2AH</td>
<td></td>
</tr>
<tr>
<td>0220–023FH</td>
<td>51H</td>
<td>2AH</td>
<td></td>
</tr>
<tr>
<td>0200–021FH</td>
<td>50H</td>
<td>2AH</td>
<td></td>
</tr>
<tr>
<td>01E0–01FFH</td>
<td>4FH</td>
<td>2AH</td>
<td></td>
</tr>
<tr>
<td>01C0–01DFH</td>
<td>4EH</td>
<td>2AH</td>
<td>13H</td>
</tr>
<tr>
<td>01A0–01BFH</td>
<td>4DH</td>
<td>2AH</td>
<td></td>
</tr>
<tr>
<td>0180–019FH</td>
<td>4CH</td>
<td>2AH</td>
<td></td>
</tr>
<tr>
<td>0160–017FH</td>
<td>4BH</td>
<td>2AH</td>
<td></td>
</tr>
<tr>
<td>0140–015FH</td>
<td>4AH</td>
<td>2AH</td>
<td>12H</td>
</tr>
<tr>
<td>0120–013FH</td>
<td>49H</td>
<td>2AH</td>
<td></td>
</tr>
<tr>
<td>0100–011FH</td>
<td>48H</td>
<td>2AH</td>
<td></td>
</tr>
<tr>
<td>Register RAM Locations</td>
<td>WSR Value for 32-byte Window (00E0–00FFH)</td>
<td>WSR Value for 64-byte Window (00C0–00FFH)</td>
<td>WSR Value for 128-byte Window (0080–00FFH)</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------</td>
<td>------------------------------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>0DE0–0DFFH</td>
<td>6FH</td>
<td>37H</td>
<td>1BH</td>
</tr>
<tr>
<td>0DC0–0DFFH</td>
<td>6EH</td>
<td>36H</td>
<td>1AH</td>
</tr>
<tr>
<td>0DA0–0DBFH</td>
<td>6DH</td>
<td>35H</td>
<td>19H</td>
</tr>
<tr>
<td>0D80–0D9FH</td>
<td>6CH</td>
<td>34H</td>
<td>18H</td>
</tr>
<tr>
<td>0D60–0D7FH</td>
<td>6BH</td>
<td>33H</td>
<td></td>
</tr>
<tr>
<td>0D40–0D5FH</td>
<td>6AH</td>
<td>32H</td>
<td></td>
</tr>
<tr>
<td>0D20–0D3FH</td>
<td>69H</td>
<td>31H</td>
<td></td>
</tr>
<tr>
<td>0D00–0D1FH</td>
<td>68H</td>
<td>30H</td>
<td></td>
</tr>
<tr>
<td>0CE0–0CFFH</td>
<td>67H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0CC0–0CDFH</td>
<td>66H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0CA0–0CBFH</td>
<td>65H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0C80–0C9FH</td>
<td>64H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0C60–0C7FH</td>
<td>63H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0C40–0C5FH</td>
<td>62H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0C20–0C3FH</td>
<td>61H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0C00–0C1FH</td>
<td>60H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 3-8. Windows

<table>
<thead>
<tr>
<th>Base Address</th>
<th>WSR Value for 32-byte Window (00E0–00FFH)</th>
<th>WSR Value for 64-byte Window (00C0–00FFH)</th>
<th>WSR Value for 128-byte Window (0080–00FFH)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peripheral SFRs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1FE0H</td>
<td>7FH †</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1FC0H</td>
<td>7EH</td>
<td>3FH †</td>
<td></td>
</tr>
<tr>
<td>1FA0H</td>
<td>7DH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1F80H</td>
<td>7CH</td>
<td>3EH</td>
<td></td>
</tr>
<tr>
<td>1F60H</td>
<td>7BH</td>
<td></td>
<td>1FH †</td>
</tr>
<tr>
<td>1F40H</td>
<td>7AH</td>
<td>3DH</td>
<td></td>
</tr>
<tr>
<td>1F20H</td>
<td>79H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1F00H</td>
<td>78H</td>
<td>3CH</td>
<td>1EH</td>
</tr>
<tr>
<td>CAN Peripheral SFRs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1EE0H</td>
<td>77H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1EC0H</td>
<td>76H</td>
<td>3BH</td>
<td></td>
</tr>
<tr>
<td>1EA0H</td>
<td>75H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E80H</td>
<td>74H</td>
<td>3AH</td>
<td>1DH</td>
</tr>
<tr>
<td>1E60H</td>
<td>73H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E40H</td>
<td>72H</td>
<td>39H</td>
<td></td>
</tr>
<tr>
<td>1E20H</td>
<td>71H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E00H</td>
<td>70H</td>
<td>38H</td>
<td>1CH</td>
</tr>
<tr>
<td>Register RAM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1DE0H</td>
<td>6FH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1DC0H</td>
<td>6EH</td>
<td>37H</td>
<td></td>
</tr>
<tr>
<td>1DA0H</td>
<td>6DH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1D80H</td>
<td>6CH</td>
<td>36H</td>
<td>18H</td>
</tr>
<tr>
<td>1D60H</td>
<td>6BH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1D40H</td>
<td>6AH</td>
<td>35H</td>
<td></td>
</tr>
<tr>
<td>1D20H</td>
<td>69H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1D00H</td>
<td>68H</td>
<td>34H</td>
<td>1AH</td>
</tr>
<tr>
<td>1CE0H</td>
<td>67H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1CC0H</td>
<td>66H</td>
<td>33H</td>
<td></td>
</tr>
<tr>
<td>1CA0H</td>
<td>65H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1C80H</td>
<td>64H</td>
<td>32H</td>
<td>19H</td>
</tr>
<tr>
<td>1C60H</td>
<td>63H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1C40H</td>
<td>62H</td>
<td>31H</td>
<td></td>
</tr>
<tr>
<td>1C20H</td>
<td>61H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1C00H</td>
<td>60H</td>
<td>30H</td>
<td>18H</td>
</tr>
</tbody>
</table>

† Locations 1FE0–1FFFH contain memory-mapped SFRs that cannot be accessed through a window. Reading these locations through a window returns FFH; writing these locations through a window has no effect.
### Table 3-8. Windows (Continued)

<table>
<thead>
<tr>
<th>Base Address</th>
<th>WSR Value for 32-byte Window (00E0–00FFH)</th>
<th>WSR Value for 64-byte Window (00C0–00FFH)</th>
<th>WSR Value for 128-byte Window (0080–00FFH)</th>
</tr>
</thead>
<tbody>
<tr>
<td>03E0H</td>
<td>5FH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03C0H</td>
<td>5EH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03A0H</td>
<td>5DH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0380H</td>
<td>5CH</td>
<td>2EH</td>
<td>17H</td>
</tr>
<tr>
<td>0360H</td>
<td>5BH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0340H</td>
<td>5AH</td>
<td>2DH</td>
<td></td>
</tr>
<tr>
<td>0320H</td>
<td>59H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0300H</td>
<td>58H</td>
<td>2CH</td>
<td>16H</td>
</tr>
<tr>
<td>02E0H</td>
<td>57H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02C0H</td>
<td>56H</td>
<td>2BH</td>
<td></td>
</tr>
<tr>
<td>02A0H</td>
<td>55H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0280H</td>
<td>54H</td>
<td>2AH</td>
<td>15H</td>
</tr>
<tr>
<td>0260H</td>
<td>53H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0240H</td>
<td>52H</td>
<td>29H</td>
<td></td>
</tr>
<tr>
<td>0220H</td>
<td>51H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0200H</td>
<td>50H</td>
<td>28H</td>
<td>14H</td>
</tr>
<tr>
<td>01E0H</td>
<td>4FH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01C0H</td>
<td>4EH</td>
<td>27H</td>
<td></td>
</tr>
<tr>
<td>01A0H</td>
<td>4DH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0180H</td>
<td>4CH</td>
<td>26H</td>
<td>13H</td>
</tr>
<tr>
<td>0160H</td>
<td>4BH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0140H</td>
<td>4AH</td>
<td>25H</td>
<td></td>
</tr>
<tr>
<td>0120H</td>
<td>49H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100H</td>
<td>48H</td>
<td>24H</td>
<td>12H</td>
</tr>
</tbody>
</table>

† Locations 1FE0–1FFFH contain memory-mapped SFRs that cannot be accessed through a window. Reading these locations through a window returns FFH; writing these locations through a window has no effect.
### Table 3-9. WSR Settings and Direct Addresses for Windowable SFRs

<table>
<thead>
<tr>
<th>Register Mnemonic</th>
<th>Memory Location</th>
<th>32-byte Windows (00E0–00FFH)</th>
<th>64-byte Windows (00C0–00FFH)</th>
<th>128-byte Windows (0080–00FFH)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>WSR</td>
<td>Direct Address</td>
<td>WSR</td>
</tr>
<tr>
<td>AD_COMMAND</td>
<td>1FACH</td>
<td>7DH</td>
<td>00ECH</td>
<td>3EH</td>
</tr>
<tr>
<td>AD_RESULT</td>
<td>1FAAH</td>
<td>7DH</td>
<td>00EAH</td>
<td>3EH</td>
</tr>
<tr>
<td>AD_TEST</td>
<td>1FAEH</td>
<td>7DH</td>
<td>00EEH</td>
<td>3EH</td>
</tr>
<tr>
<td>AD_TIME</td>
<td>1FAPH</td>
<td>7DH</td>
<td>00EFH</td>
<td>3EH</td>
</tr>
<tr>
<td>CAN_BT1TIME0</td>
<td>1E3FH</td>
<td>71H</td>
<td>00FFF</td>
<td>3EH</td>
</tr>
<tr>
<td>CAN_BT1TIME1</td>
<td>1E4FH</td>
<td>72H</td>
<td>00EFH</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_CON</td>
<td>1E00H</td>
<td>70H</td>
<td>00E0H</td>
<td>38H</td>
</tr>
<tr>
<td>CAN_EGMSK</td>
<td>1E08H</td>
<td>70H</td>
<td>00E8H</td>
<td>38H</td>
</tr>
<tr>
<td>CAN_INT</td>
<td>1E5FH</td>
<td>72H</td>
<td>00FFH</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG1CFG</td>
<td>1E16H</td>
<td>70H</td>
<td>00F6H</td>
<td>38H</td>
</tr>
<tr>
<td>CAN_MSG2CFG</td>
<td>1E26H</td>
<td>71H</td>
<td>00E6H</td>
<td>38H</td>
</tr>
<tr>
<td>CAN_MSG3CFG</td>
<td>1E36H</td>
<td>71H</td>
<td>00F6H</td>
<td>38H</td>
</tr>
<tr>
<td>CAN_MSG4CFG</td>
<td>1E46H</td>
<td>72H</td>
<td>00E6H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG5CFG</td>
<td>1E56H</td>
<td>72H</td>
<td>00F6H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG6CFG</td>
<td>1E66H</td>
<td>73H</td>
<td>00E6H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG7CFG</td>
<td>1E76H</td>
<td>73H</td>
<td>00F6H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG8CFG</td>
<td>1E86H</td>
<td>74H</td>
<td>00E6H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG9CFG</td>
<td>1E96H</td>
<td>74H</td>
<td>00F6H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG10CFG</td>
<td>1EA6H</td>
<td>75H</td>
<td>00E6H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG11CFG</td>
<td>1EB6H</td>
<td>75H</td>
<td>00F6H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG12CFG</td>
<td>1EC6H</td>
<td>76H</td>
<td>00E6H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG13CFG</td>
<td>1ED6H</td>
<td>76H</td>
<td>00F6H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG14CFG</td>
<td>1EE6H</td>
<td>77H</td>
<td>00E6H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG15CFG</td>
<td>1EF6H</td>
<td>77H</td>
<td>00F6H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG1CON0</td>
<td>1E10H</td>
<td>70H</td>
<td>00F0H</td>
<td>38H</td>
</tr>
<tr>
<td>CAN_MSG2CON0</td>
<td>1E20H</td>
<td>71H</td>
<td>00E0H</td>
<td>38H</td>
</tr>
<tr>
<td>CAN_MSG3CON0</td>
<td>1E30H</td>
<td>71H</td>
<td>00F0H</td>
<td>38H</td>
</tr>
<tr>
<td>CAN_MSG4CON0</td>
<td>1E40H</td>
<td>72H</td>
<td>00E0H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG5CON0</td>
<td>1E50H</td>
<td>72H</td>
<td>00F0H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG6CON0</td>
<td>1E60H</td>
<td>73H</td>
<td>00E0H</td>
<td>39H</td>
</tr>
</tbody>
</table>

† Must be addressed as a word.
**Table 3-9. WSR Settings and Direct Addresses for Windowable SFRs (Continued)**

<table>
<thead>
<tr>
<th>Register Mnemonic</th>
<th>Memory Location</th>
<th>32-byte Windows (00E0–00FFH)</th>
<th>64-byte Windows (00C0–00FFH)</th>
<th>128-byte Windows (0080–00FFH)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WSR</td>
<td>Direct Address</td>
<td>WSR</td>
<td>Direct Address</td>
</tr>
<tr>
<td>CAN_MSG7CON0</td>
<td>1E70H</td>
<td>73H</td>
<td>00F0H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG8CON0</td>
<td>1E80H</td>
<td>74H</td>
<td>00E0H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG9CON0</td>
<td>1E90H</td>
<td>74H</td>
<td>00F0H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG10CON0</td>
<td>1EA0H</td>
<td>75H</td>
<td>00E0H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG11CON0</td>
<td>1EB0H</td>
<td>75H</td>
<td>00F0H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG12CON0</td>
<td>1EC0H</td>
<td>76H</td>
<td>00E0H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG13CON0</td>
<td>1ED0H</td>
<td>76H</td>
<td>00F0H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG14CON0</td>
<td>1EE0H</td>
<td>77H</td>
<td>00E0H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG15CON0</td>
<td>1EF0H</td>
<td>77H</td>
<td>00F0H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG1CON1</td>
<td>1E11H</td>
<td>70H</td>
<td>00F1H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG2CON1</td>
<td>1E21H</td>
<td>71H</td>
<td>00E1H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG3CON1</td>
<td>1E31H</td>
<td>71H</td>
<td>00F1H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG4CON1</td>
<td>1E41H</td>
<td>72H</td>
<td>00E1H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG5CON1</td>
<td>1E51H</td>
<td>72H</td>
<td>00F1H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG6CON1</td>
<td>1E61H</td>
<td>73H</td>
<td>00E1H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG7CON1</td>
<td>1E71H</td>
<td>73H</td>
<td>00F1H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG8CON1</td>
<td>1E81H</td>
<td>74H</td>
<td>00E1H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG9CON1</td>
<td>1E91H</td>
<td>74H</td>
<td>00F1H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG10CON1</td>
<td>1EA1H</td>
<td>75H</td>
<td>00E1H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG11CON1</td>
<td>1EB1H</td>
<td>75H</td>
<td>00F1H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG12CON1</td>
<td>1EC1H</td>
<td>76H</td>
<td>00E1H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG13CON1</td>
<td>1ED1H</td>
<td>76H</td>
<td>00F1H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG14CON1</td>
<td>1EE1H</td>
<td>77H</td>
<td>00E1H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG15CON1</td>
<td>1EF1H</td>
<td>77H</td>
<td>00F1H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG1DATA0</td>
<td>1E17H</td>
<td>70H</td>
<td>00F7H</td>
<td>38H</td>
</tr>
<tr>
<td>CAN_MSG2DATA0</td>
<td>1E27H</td>
<td>71H</td>
<td>00E7H</td>
<td>38H</td>
</tr>
<tr>
<td>CAN_MSG3DATA0</td>
<td>1E37H</td>
<td>71H</td>
<td>00F7H</td>
<td>38H</td>
</tr>
<tr>
<td>CAN_MSG4DATA0</td>
<td>1E47H</td>
<td>72H</td>
<td>00E7H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG5DATA0</td>
<td>1E57H</td>
<td>72H</td>
<td>00F7H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG6DATA0</td>
<td>1E67H</td>
<td>73H</td>
<td>00E7H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG7DATA0</td>
<td>1E77H</td>
<td>73H</td>
<td>00F7H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG8DATA0</td>
<td>1E87H</td>
<td>74H</td>
<td>00E7H</td>
<td>3AH</td>
</tr>
</tbody>
</table>

† Must be addressed as a word.
### Table 3-9. WSR Settings and Direct Addresses for Windowable SFRs (Continued)

<table>
<thead>
<tr>
<th>Register Mnemonic</th>
<th>Memory Location</th>
<th>32-byte Windows (00E0–00FFH)</th>
<th>64-byte Windows (00C0–00FFH)</th>
<th>128-byte Windows (0080–00FFH)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>WSR</td>
<td>Direct Address</td>
<td>WSR</td>
</tr>
<tr>
<td>CAN_MSG9DATA0</td>
<td>1E97H</td>
<td>74H</td>
<td>00F7H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG10DATA0</td>
<td>1E17H</td>
<td>75H</td>
<td>00E7H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG11DATA0</td>
<td>1EB7H</td>
<td>75H</td>
<td>00F7H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG12DATA0</td>
<td>1EC7H</td>
<td>76H</td>
<td>00E7H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG13DATA0</td>
<td>1ED7H</td>
<td>76H</td>
<td>00F7H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG14DATA0</td>
<td>1EE7H</td>
<td>77H</td>
<td>00E7H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG15DATA0</td>
<td>1EF7H</td>
<td>77H</td>
<td>00F7H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG1DATA1</td>
<td>1E18H</td>
<td>70H</td>
<td>00F8H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG2DATA1</td>
<td>1E28H</td>
<td>71H</td>
<td>00E8H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG3DATA1</td>
<td>1E38H</td>
<td>71H</td>
<td>00F8H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG4DATA1</td>
<td>1E48H</td>
<td>72H</td>
<td>00E8H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG5DATA1</td>
<td>1E58H</td>
<td>72H</td>
<td>00F8H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG6DATA1</td>
<td>1E68H</td>
<td>73H</td>
<td>00E8H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG7DATA1</td>
<td>1E78H</td>
<td>73H</td>
<td>00F8H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG8DATA1</td>
<td>1E88H</td>
<td>74H</td>
<td>00E8H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG9DATA1</td>
<td>1E98H</td>
<td>74H</td>
<td>00F8H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG10DATA1</td>
<td>1EA8H</td>
<td>75H</td>
<td>00E8H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG11DATA1</td>
<td>1EB8H</td>
<td>75H</td>
<td>00F8H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG12DATA1</td>
<td>1EC8H</td>
<td>76H</td>
<td>00E8H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG13DATA1</td>
<td>1ED8H</td>
<td>76H</td>
<td>00F8H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG14DATA1</td>
<td>1EE8H</td>
<td>77H</td>
<td>00E8H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG15DATA1</td>
<td>1EF8H</td>
<td>77H</td>
<td>00F8H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG1DATA2</td>
<td>1E19H</td>
<td>70H</td>
<td>00F9H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG2DATA2</td>
<td>1E29H</td>
<td>71H</td>
<td>00E9H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG3DATA2</td>
<td>1E39H</td>
<td>71H</td>
<td>00F9H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG4DATA2</td>
<td>1E49H</td>
<td>72H</td>
<td>00E9H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG5DATA2</td>
<td>1E59H</td>
<td>72H</td>
<td>00F9H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG6DATA2</td>
<td>1E69H</td>
<td>73H</td>
<td>00E9H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG7DATA2</td>
<td>1E79H</td>
<td>73H</td>
<td>00F9H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG8DATA2</td>
<td>1E89H</td>
<td>74H</td>
<td>00E9H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG9DATA2</td>
<td>1E99H</td>
<td>74H</td>
<td>00F9H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG10DATA2</td>
<td>1EA9H</td>
<td>75H</td>
<td>00E9H</td>
<td>3AH</td>
</tr>
</tbody>
</table>

† Must be addressed as a word.
### Table 3-9. WSR Settings and Direct Addresses for Windowable SFRs (Continued)

<table>
<thead>
<tr>
<th>Register Mnemonic</th>
<th>Memory Location</th>
<th>32-byte Windows (00E0–00FFH)</th>
<th>64-byte Windows (00C0–00FFH)</th>
<th>128-byte Windows (0080–00FFH)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>WSR Direct Address</td>
<td>WSR Direct Address</td>
<td>WSR Direct Address</td>
</tr>
<tr>
<td>CAN_MSG11DATA2</td>
<td>1EB9H</td>
<td>75H 00F9H 3AH 00F9H 1DH 00B9H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG12DATA2</td>
<td>1EC9H</td>
<td>76H 00E9H 3BH 00C9H 1DH 00C9H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG13DATA2</td>
<td>1ED9H</td>
<td>76H 00F9H 3BH 00D9H 1DH 00D9H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG14DATA2</td>
<td>1EE9H</td>
<td>77H 00E9H 3BH 00E9H 1DH 00E9H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG15DATA2</td>
<td>1EF9H</td>
<td>77H 00F9H 3BH 00F9H 1DH 00F9H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG1DATA3</td>
<td>1E1AH</td>
<td>70H 00FAH 3BH 00DAH 1CH 009AH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG2DATA3</td>
<td>1E2AH</td>
<td>71H 00EAH 3BH 00EAH 1CH 00AAH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG3DATA3</td>
<td>1E3AH</td>
<td>71H 00FAH 3BH 00FAH 1CH 00BAH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG4DATA3</td>
<td>1E4AH</td>
<td>72H 00EAH 3BH 00CAH 1CH 00CAH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG5DATA3</td>
<td>1E5AH</td>
<td>72H 00FAH 3BH 00DAH 1CH 00DAH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG6DATA3</td>
<td>1E6AH</td>
<td>73H 00EAH 3BH 00EAH 1CH 00EAH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG7DATA3</td>
<td>1E7AH</td>
<td>73H 00FAH 3BH 00FAH 1CH 00FAH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG8DATA3</td>
<td>1E8AH</td>
<td>74H 00EAH 3AH 00CAH 1DH 008AH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG9DATA3</td>
<td>1E9AH</td>
<td>74H 00FAH 3AH 00DAH 1DH 009AH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG10DATA3</td>
<td>1EAAH</td>
<td>75H 00EAH 3AH 00EAH 1DH 00AAH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG11DATA3</td>
<td>1EBAH</td>
<td>75H 00FAH 3AH 00FAH 1DH 00BAH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG12DATA3</td>
<td>1ECAH</td>
<td>76H 00EAH 3BH 00CAH 1DH 00CAH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG13DATA3</td>
<td>1EDAH</td>
<td>76H 00FAH 3BH 00DAH 1DH 00DAH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG14DATA3</td>
<td>1EEAH</td>
<td>77H 00EAH 3BH 00EAH 1DH 00EAH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG15DATA3</td>
<td>1EFAH</td>
<td>77H 00FAH 3BH 00FAH 1DH 00FAH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG1DATA4</td>
<td>1E1BH</td>
<td>70H 00FBH 3BH 00DBH 1CH 009BH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG2DATA4</td>
<td>1E2BH</td>
<td>71H 00EBH 3BH 00EBH 1CH 00ABH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG3DATA4</td>
<td>1E3BH</td>
<td>71H 00FBH 3BH 00FBH 1CH 00BBH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG4DATA4</td>
<td>1E4BH</td>
<td>72H 00EBH 3BH 00CAH 1CH 00CBH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG5DATA4</td>
<td>1E5BH</td>
<td>72H 00FBH 3BH 00DBH 1CH 00DBH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG6DATA4</td>
<td>1E6BH</td>
<td>73H 00EBH 3BH 00EBH 1CH 00EBH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG7DATA4</td>
<td>1E7BH</td>
<td>73H 00FBH 3BH 00FBH 1CH 00FBH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG8DATA4</td>
<td>1E8BH</td>
<td>74H 00EBH 3AH 00CBH 1CH 00CBH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG9DATA4</td>
<td>1E9BH</td>
<td>74H 00FBH 3AH 00DBH 1CH 00DBH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG10DATA4</td>
<td>1EABH</td>
<td>75H 00EBH 3AH 00EBH 1DH 00ABH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG11DATA4</td>
<td>1EBBH</td>
<td>75H 00FBH 3AH 00FBH 1DH 00BBH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN_MSG12DATA4</td>
<td>1ECBH</td>
<td>76H 00EBH 3BH 00CBH 1DH 00CBH</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

† Must be addressed as a word.
<table>
<thead>
<tr>
<th>Register Mnemonic</th>
<th>Memory Location</th>
<th>32-byte Windows (00E0–00FFH)</th>
<th>64-byte Windows (00C0–00FFH)</th>
<th>128-byte Windows (0080–00FFH)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WSR</td>
<td>Direct Address</td>
<td>WSR</td>
<td>Direct Address</td>
</tr>
<tr>
<td>CAN_MSG13DATA4</td>
<td>1EDBH</td>
<td>76H  00FBH</td>
<td>3BH  00DBH</td>
<td>1DH  00DBH</td>
</tr>
<tr>
<td>CAN_MSG14DATA4</td>
<td>1EEBH</td>
<td>77H  00EBH</td>
<td>3BH  00EBH</td>
<td>1DH  00EBH</td>
</tr>
<tr>
<td>CAN_MSG15DATA4</td>
<td>1EFBH</td>
<td>77H  00FBH</td>
<td>3BH  00FBH</td>
<td>1DH  00FBH</td>
</tr>
<tr>
<td>CAN_MSG1DATA5</td>
<td>1E1CH</td>
<td>70H  00FCH</td>
<td>3BH  00FCH</td>
<td>1CH  009CH</td>
</tr>
<tr>
<td>CAN_MSG2DATA5</td>
<td>1E2CH</td>
<td>71H  00ECH</td>
<td>3BH  00ECH</td>
<td>1CH  00ACH</td>
</tr>
<tr>
<td>CAN_MSG3DATA5</td>
<td>1E3CH</td>
<td>71H  00FCH</td>
<td>3BH  00FCH</td>
<td>1CH  00BCH</td>
</tr>
<tr>
<td>CAN_MSG4DATA5</td>
<td>1E4CH</td>
<td>72H  00ECH</td>
<td>3BH  00ECH</td>
<td>1CH  00CCH</td>
</tr>
<tr>
<td>CAN_MSG5DATA5</td>
<td>1E5CH</td>
<td>72H  00FCH</td>
<td>3BH  00FCH</td>
<td>1CH  00DCH</td>
</tr>
<tr>
<td>CAN_MSG6DATA5</td>
<td>1E6CH</td>
<td>73H  00ECH</td>
<td>3BH  00ECH</td>
<td>1CH  00ECH</td>
</tr>
<tr>
<td>CAN_MSG7DATA5</td>
<td>1E7CH</td>
<td>73H  00FCH</td>
<td>3BH  00FCH</td>
<td>1CH  00FCH</td>
</tr>
<tr>
<td>CAN_MSG8DATA5</td>
<td>1E8CH</td>
<td>74H  00ECH</td>
<td>3AH  00CCH</td>
<td>1DH  008CH</td>
</tr>
<tr>
<td>CAN_MSG9DATA5</td>
<td>1E9CH</td>
<td>74H  00FCH</td>
<td>3AH  00DCH</td>
<td>1DH  009CH</td>
</tr>
<tr>
<td>CAN_MSG10DATA5</td>
<td>1EACH</td>
<td>75H  00ECH</td>
<td>3AH  00ECH</td>
<td>1DH  00ACH</td>
</tr>
<tr>
<td>CAN_MSG11DATA5</td>
<td>1EBCH</td>
<td>75H  00FCH</td>
<td>3AH  00FCH</td>
<td>1DH  00BCH</td>
</tr>
<tr>
<td>CAN_MSG12DATA5</td>
<td>1ECCCH</td>
<td>76H  00ECH</td>
<td>3BH  00ECH</td>
<td>1DH  00CCH</td>
</tr>
<tr>
<td>CAN_MSG13DATA5</td>
<td>1EDCH</td>
<td>76H  00FCH</td>
<td>3BH  00FCH</td>
<td>1DH  00DCH</td>
</tr>
<tr>
<td>CAN_MSG14DATA5</td>
<td>1EECH</td>
<td>77H  00ECH</td>
<td>3BH  00ECH</td>
<td>1DH  00ECH</td>
</tr>
<tr>
<td>CAN_MSG15DATA5</td>
<td>1EFCH</td>
<td>77H  00FCH</td>
<td>3BH  00FCH</td>
<td>1DH  00FCH</td>
</tr>
<tr>
<td>CAN_MSG1DATA6</td>
<td>1E1DH</td>
<td>70H  00FDH</td>
<td>3BH  00DDH</td>
<td>1CH  009DH</td>
</tr>
<tr>
<td>CAN_MSG2DATA6</td>
<td>1E2DH</td>
<td>71H  00EDH</td>
<td>3BH  00EDH</td>
<td>1CH  00ADH</td>
</tr>
<tr>
<td>CAN_MSG3DATA6</td>
<td>1E3DH</td>
<td>71H  00FDH</td>
<td>3BH  00FDH</td>
<td>1CH  00BDH</td>
</tr>
<tr>
<td>CAN_MSG4DATA6</td>
<td>1E4DH</td>
<td>72H  00EDH</td>
<td>3BH  00CDH</td>
<td>1CH  00CDH</td>
</tr>
<tr>
<td>CAN_MSG5DATA6</td>
<td>1E5DH</td>
<td>72H  00FDH</td>
<td>3BH  00DDH</td>
<td>1CH  00DDH</td>
</tr>
<tr>
<td>CAN_MSG6DATA6</td>
<td>1E6DH</td>
<td>73H  00EDH</td>
<td>3BH  00EDH</td>
<td>1CH  00DDH</td>
</tr>
<tr>
<td>CAN_MSG7DATA6</td>
<td>1E7DH</td>
<td>73H  00FDH</td>
<td>3BH  00FDH</td>
<td>1CH  00FDH</td>
</tr>
<tr>
<td>CAN_MSG8DATA6</td>
<td>1E8DH</td>
<td>74H  00EDH</td>
<td>3AH  00CDH</td>
<td>1DH  008DH</td>
</tr>
<tr>
<td>CAN_MSG9DATA6</td>
<td>1E9DH</td>
<td>74H  00FDH</td>
<td>3AH  00DDH</td>
<td>1DH  009DH</td>
</tr>
<tr>
<td>CAN_MSG10DATA6</td>
<td>1EADH</td>
<td>75H  00EDH</td>
<td>3AH  00EDH</td>
<td>1DH  00ADH</td>
</tr>
<tr>
<td>CAN_MSG11DATA6</td>
<td>1EBDH</td>
<td>75H  00FDH</td>
<td>3AH  00FDH</td>
<td>1DH  00BDH</td>
</tr>
<tr>
<td>CAN_MSG12DATA6</td>
<td>1ECDH</td>
<td>76H  00EDH</td>
<td>3BH  00CDH</td>
<td>1DH  00CDH</td>
</tr>
<tr>
<td>CAN_MSG13DATA6</td>
<td>1EDDH</td>
<td>76H  00FDH</td>
<td>3BH  00DDH</td>
<td>1DH  00DDH</td>
</tr>
<tr>
<td>CAN_MSG14DATA6</td>
<td>1EEDH</td>
<td>77H  00EDH</td>
<td>3BH  00EDH</td>
<td>1DH  00EDH</td>
</tr>
</tbody>
</table>

† Must be addressed as a word.
### Table 3-9. WSR Settings and Direct Addresses for Windowable SFRs (Continued)

<table>
<thead>
<tr>
<th>Register Mnemonic</th>
<th>Memory Location</th>
<th>32-byte Windows (00E0–00FFH)</th>
<th>64-byte Windows (00C0–00FFH)</th>
<th>128-byte Windows (0080–00FFH)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WSR</td>
<td>Direct Address</td>
<td>WSR</td>
<td>Direct Address</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WSR</td>
<td>Direct Address</td>
</tr>
<tr>
<td>CAN.MSG15DATA6</td>
<td>1EFDH</td>
<td>77H 00FDH</td>
<td>3BH 00FDH</td>
<td>1DH 00FDH</td>
</tr>
<tr>
<td>CAN.MSG11DATA7</td>
<td>1E12H</td>
<td>70H 00FEH</td>
<td>3BH 00DEH</td>
<td>1CH 00E9H</td>
</tr>
<tr>
<td>CAN.MSG2DATA7</td>
<td>1E2EH</td>
<td>71H 00EEH</td>
<td>3BH 00EEH</td>
<td>1CH 00AEH</td>
</tr>
<tr>
<td>CAN.MSG3DATA7</td>
<td>1E3EH</td>
<td>71H 00FEH</td>
<td>3BH 00FEH</td>
<td>1CH 00E9H</td>
</tr>
<tr>
<td>CAN.MSG4DATA7</td>
<td>1E4EH</td>
<td>72H 00EEH</td>
<td>3BH 00FEH</td>
<td>1CH 00CEH</td>
</tr>
<tr>
<td>CAN.MSG5DATA7</td>
<td>1E5EH</td>
<td>72H 00FEH</td>
<td>3BH 00DEH</td>
<td>1CH 00E9H</td>
</tr>
<tr>
<td>CAN.MSG6DATA7</td>
<td>1E6EH</td>
<td>73H 00EEH</td>
<td>3BH 00EEH</td>
<td>1CH 00E9H</td>
</tr>
<tr>
<td>CAN.MSG7DATA7</td>
<td>1E7EH</td>
<td>73H 00FEH</td>
<td>3BH 00FEH</td>
<td>1CH 00FEH</td>
</tr>
<tr>
<td>CAN.MSG8DATA7</td>
<td>1E8EH</td>
<td>74H 00EEH</td>
<td>3BH 00CEH</td>
<td>1CH 00AEH</td>
</tr>
<tr>
<td>CAN.MSG9DATA7</td>
<td>1E9EH</td>
<td>74H 00FEH</td>
<td>3BH 00DEH</td>
<td>1CH 00AEH</td>
</tr>
<tr>
<td>CAN.MSG10DATA7</td>
<td>1EAEH</td>
<td>75H 00EEH</td>
<td>3BH 00EEH</td>
<td>1CH 00E9H</td>
</tr>
<tr>
<td>CAN.MSG11DATA7</td>
<td>1EBEH</td>
<td>75H 00FEH</td>
<td>3BH 00FEH</td>
<td>1CH 00E9H</td>
</tr>
<tr>
<td>CAN.MSG12DATA7</td>
<td>1ECEH</td>
<td>76H 00EEH</td>
<td>3BH 00CEH</td>
<td>1CH 00E9H</td>
</tr>
<tr>
<td>CAN.MSG13DATA7</td>
<td>1EDEH</td>
<td>76H 00FEH</td>
<td>3BH 00CEH</td>
<td>1CH 00E9H</td>
</tr>
<tr>
<td>CAN.MSG14DATA7</td>
<td>1EEEH</td>
<td>77H 00EEH</td>
<td>3BH 00CEH</td>
<td>1CH 00E9H</td>
</tr>
<tr>
<td>CAN.MSG15DATA7</td>
<td>1EFEH</td>
<td>77H 00FEH</td>
<td>3BH 00CEH</td>
<td>1CH 00E9H</td>
</tr>
<tr>
<td>CAN.MSG11D0</td>
<td>1E12H</td>
<td>70H 00F2H</td>
<td>3BH 00D2H</td>
<td>1CH 0092H</td>
</tr>
<tr>
<td>CAN.MSG21D0</td>
<td>1E22H</td>
<td>71H 00F2H</td>
<td>3BH 00E2H</td>
<td>1CH 00A2H</td>
</tr>
<tr>
<td>CAN.MSG31D0</td>
<td>1E32H</td>
<td>71H 00F2H</td>
<td>3BH 00F2H</td>
<td>1CH 00B2H</td>
</tr>
<tr>
<td>CAN.MSG41D0</td>
<td>1E42H</td>
<td>72H 00E2H</td>
<td>3BH 00C2H</td>
<td>1CH 00C2H</td>
</tr>
<tr>
<td>CAN.MSG51D0</td>
<td>1E52H</td>
<td>72H 00F2H</td>
<td>3BH 00D2H</td>
<td>1CH 00D2H</td>
</tr>
<tr>
<td>CAN.MSG61D0</td>
<td>1E62H</td>
<td>73H 00E2H</td>
<td>3BH 00E2H</td>
<td>1CH 00E2H</td>
</tr>
<tr>
<td>CAN.MSG71D0</td>
<td>1E72H</td>
<td>73H 00F2H</td>
<td>3BH 00F2H</td>
<td>1CH 00F2H</td>
</tr>
<tr>
<td>CAN.MSG81D0</td>
<td>1E82H</td>
<td>74H 00E2H</td>
<td>3AH 00C2H</td>
<td>1CH 00B2H</td>
</tr>
<tr>
<td>CAN.MSG91D0</td>
<td>1E92H</td>
<td>74H 00F2H</td>
<td>3AH 00D2H</td>
<td>1CH 00D2H</td>
</tr>
<tr>
<td>CAN.MSG101D0</td>
<td>1EA2H</td>
<td>75H 00E2H</td>
<td>3AH 00E2H</td>
<td>1CH 00E2H</td>
</tr>
<tr>
<td>CAN.MSG111D0</td>
<td>1EB2H</td>
<td>75H 00F2H</td>
<td>3AH 00F2H</td>
<td>1CH 00E2H</td>
</tr>
<tr>
<td>CAN.MSG121D0</td>
<td>1EC2H</td>
<td>76H 00E2H</td>
<td>3BH 00C2H</td>
<td>1CH 00C2H</td>
</tr>
<tr>
<td>CAN.MSG131D0</td>
<td>1ED2H</td>
<td>76H 00F2H</td>
<td>3BH 00D2H</td>
<td>1CH 00D2H</td>
</tr>
<tr>
<td>CAN.MSG141D0</td>
<td>1EE2H</td>
<td>77H 00E2H</td>
<td>3BH 00E2H</td>
<td>1CH 00E2H</td>
</tr>
<tr>
<td>CAN.MSG151D0</td>
<td>1EF2H</td>
<td>77H 00F2H</td>
<td>3BH 00F2H</td>
<td>1CH 00F2H</td>
</tr>
<tr>
<td>CAN.MSG11D1</td>
<td>1E13H</td>
<td>70H 00F3H</td>
<td>3BH 00D3H</td>
<td>1CH 0093H</td>
</tr>
</tbody>
</table>

† Must be addressed as a word.
<table>
<thead>
<tr>
<th>Register Mnemonic</th>
<th>Memory Location</th>
<th>32-byte Windows (00E0–00FFH)</th>
<th>64-byte Windows (00C0–00FFH)</th>
<th>128-byte Windows (0080–00FFH)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>WSR Direct Address</td>
<td>WSR Direct Address</td>
<td>WSR Direct Address</td>
</tr>
<tr>
<td>CAN_MSG2ID1</td>
<td>1E23H</td>
<td>71H 00E3H</td>
<td>38H 00E3H</td>
<td>1CH 00A3H</td>
</tr>
<tr>
<td>CAN_MSG3ID1</td>
<td>1E33H</td>
<td>71H 00F3H</td>
<td>38H 00F3H</td>
<td>1CH 00B3H</td>
</tr>
<tr>
<td>CAN_MSG4ID1</td>
<td>1E43H</td>
<td>72H 00E3H</td>
<td>39H 00C3H</td>
<td>1CH 00C3H</td>
</tr>
<tr>
<td>CAN_MSG5ID1</td>
<td>1E53H</td>
<td>72H 00F3H</td>
<td>39H 00D3H</td>
<td>1CH 00D3H</td>
</tr>
<tr>
<td>CAN_MSG6ID1</td>
<td>1E63H</td>
<td>73H 00E3H</td>
<td>39H 00E3H</td>
<td>1CH 00E3H</td>
</tr>
<tr>
<td>CAN_MSG7ID1</td>
<td>1E73H</td>
<td>73H 00F3H</td>
<td>39H 00F3H</td>
<td>1CH 00F3H</td>
</tr>
<tr>
<td>CAN_MSG8ID1</td>
<td>1E83H</td>
<td>74H 00E3H</td>
<td>3AH 00C3H</td>
<td>1DH 0083H</td>
</tr>
<tr>
<td>CAN_MSG9ID1</td>
<td>1E93H</td>
<td>74H 00F3H</td>
<td>3AH 00D3H</td>
<td>1DH 0093H</td>
</tr>
<tr>
<td>CAN_MSG10ID1</td>
<td>1EA3H</td>
<td>75H 00E3H</td>
<td>3AH 00E3H</td>
<td>1DH 00A3H</td>
</tr>
<tr>
<td>CAN_MSG11ID1</td>
<td>1EB3H</td>
<td>75H 00F3H</td>
<td>3AH 00F3H</td>
<td>1DH 00B3H</td>
</tr>
<tr>
<td>CAN_MSG12ID1</td>
<td>1EC3H</td>
<td>76H 00E3H</td>
<td>3BH 00C3H</td>
<td>1DH 00C3H</td>
</tr>
<tr>
<td>CAN_MSG13ID1</td>
<td>1ED3H</td>
<td>76H 00F3H</td>
<td>3BH 00D3H</td>
<td>1DH 00D3H</td>
</tr>
<tr>
<td>CAN_MSG14ID1</td>
<td>1EE3H</td>
<td>77H 00E3H</td>
<td>3BH 00E3H</td>
<td>1DH 00E3H</td>
</tr>
<tr>
<td>CAN_MSG15ID1</td>
<td>1EF3H</td>
<td>77H 00F3H</td>
<td>3BH 00F3H</td>
<td>1DH 00F3H</td>
</tr>
<tr>
<td>CAN_MSG11D2</td>
<td>1E14H</td>
<td>70H 00E4H</td>
<td>3AH 00C4H</td>
<td>1CH 0094H</td>
</tr>
<tr>
<td>CAN_MSG21D2</td>
<td>1E24H</td>
<td>71H 00E4H</td>
<td>3AH 00E4H</td>
<td>1CH 00A4H</td>
</tr>
<tr>
<td>CAN_MSG31D2</td>
<td>1E34H</td>
<td>71H 00F4H</td>
<td>3AH 00F4H</td>
<td>1CH 00B4H</td>
</tr>
<tr>
<td>CAN_MSG41D2</td>
<td>1E44H</td>
<td>72H 00E4H</td>
<td>39H 00C4H</td>
<td>1CH 00C4H</td>
</tr>
<tr>
<td>CAN_MSG51D2</td>
<td>1E54H</td>
<td>72H 00F4H</td>
<td>39H 00D4H</td>
<td>1CH 00D4H</td>
</tr>
<tr>
<td>CAN_MSG61D2</td>
<td>1E64H</td>
<td>73H 00E4H</td>
<td>39H 00E4H</td>
<td>1CH 00E4H</td>
</tr>
<tr>
<td>CAN_MSG71D2</td>
<td>1E74H</td>
<td>73H 00F4H</td>
<td>39H 00F4H</td>
<td>1CH 00F4H</td>
</tr>
<tr>
<td>CAN_MSG81D2</td>
<td>1E84H</td>
<td>74H 00E4H</td>
<td>3AH 00C4H</td>
<td>1DH 0084H</td>
</tr>
<tr>
<td>CAN_MSG91D2</td>
<td>1E94H</td>
<td>74H 00F4H</td>
<td>3AH 00D4H</td>
<td>1DH 0094H</td>
</tr>
<tr>
<td>CAN_MSG101D2</td>
<td>1EA4H</td>
<td>75H 00E4H</td>
<td>3AH 00E4H</td>
<td>1DH 00A4H</td>
</tr>
<tr>
<td>CAN_MSG111D2</td>
<td>1EB4H</td>
<td>75H 00F4H</td>
<td>3AH 00F4H</td>
<td>1DH 00B4H</td>
</tr>
<tr>
<td>CAN_MSG121D2</td>
<td>1EC4H</td>
<td>76H 00E4H</td>
<td>3BH 00C4H</td>
<td>1DH 00C4H</td>
</tr>
<tr>
<td>CAN_MSG131D2</td>
<td>1ED4H</td>
<td>76H 00F4H</td>
<td>3BH 00D4H</td>
<td>1DH 00D4H</td>
</tr>
<tr>
<td>CAN_MSG141D2</td>
<td>1EE4H</td>
<td>77H 00E4H</td>
<td>3BH 00E4H</td>
<td>1DH 00E4H</td>
</tr>
<tr>
<td>CAN_MSG151D2</td>
<td>1EF4H</td>
<td>77H 00F4H</td>
<td>3BH 00F4H</td>
<td>1DH 00F4H</td>
</tr>
<tr>
<td>CAN_MSG11D3</td>
<td>1E15H</td>
<td>70H 00F5H</td>
<td>38H 00D5H</td>
<td>1CH 0095H</td>
</tr>
<tr>
<td>CAN_MSG21D3</td>
<td>1E25H</td>
<td>71H 00E5H</td>
<td>38H 00E5H</td>
<td>1CH 00A5H</td>
</tr>
<tr>
<td>CAN_MSG31D3</td>
<td>1E35H</td>
<td>71H 00F5H</td>
<td>38H 00F5H</td>
<td>1CH 00B5H</td>
</tr>
</tbody>
</table>

† Must be addressed as a word.
### Table 3-9. WSR Settings and Direct Addresses for Windowable SFRs (Continued)

<table>
<thead>
<tr>
<th>Register Mnemonic</th>
<th>Memory Location</th>
<th>32-byte Windows (00E0–00FFH)</th>
<th>64-byte Windows (00C0–00FFH)</th>
<th>128-byte Windows (0080–00FFH)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>WSR</td>
<td>Direct Address</td>
<td>WSR</td>
</tr>
<tr>
<td>CAN_MSG4ID3</td>
<td>1E45H</td>
<td>72H</td>
<td>00E5H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG5ID3</td>
<td>1E55H</td>
<td>72H</td>
<td>00F5H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG6ID3</td>
<td>1E65H</td>
<td>73H</td>
<td>00E5H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG7ID3</td>
<td>1E75H</td>
<td>73H</td>
<td>00F5H</td>
<td>39H</td>
</tr>
<tr>
<td>CAN_MSG8ID3</td>
<td>1E85H</td>
<td>74H</td>
<td>00E5H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG9ID3</td>
<td>1E95H</td>
<td>74H</td>
<td>00F5H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG10ID3</td>
<td>1EA5H</td>
<td>75H</td>
<td>00E5H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG11ID3</td>
<td>1EB5H</td>
<td>75H</td>
<td>00F5H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_MSG12ID3</td>
<td>1EC5H</td>
<td>76H</td>
<td>00E5H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG13ID3</td>
<td>1ED5H</td>
<td>76H</td>
<td>00F5H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG14ID3</td>
<td>1EE5H</td>
<td>77H</td>
<td>00E5H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSG15ID3</td>
<td>1EF5H</td>
<td>77H</td>
<td>00F5H</td>
<td>3BH</td>
</tr>
<tr>
<td>CAN_MSK15</td>
<td>1E0CH</td>
<td>70H</td>
<td>00E6H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_SGMSK</td>
<td>1E06H</td>
<td>70H</td>
<td>00E6H</td>
<td>3AH</td>
</tr>
<tr>
<td>CAN_STAT</td>
<td>1E01H</td>
<td>70H</td>
<td>00E1H</td>
<td>3BH</td>
</tr>
<tr>
<td>COMP0_CON</td>
<td>1F88H</td>
<td>7CH</td>
<td>00E8H</td>
<td>3EH</td>
</tr>
<tr>
<td>COMP1_CON</td>
<td>1F8CH</td>
<td>7CH</td>
<td>00E8H</td>
<td>3EH</td>
</tr>
<tr>
<td>COMP0_TIME†</td>
<td>1F8AH</td>
<td>7CH</td>
<td>00EAH</td>
<td>3EH</td>
</tr>
<tr>
<td>COMP1_TIME†</td>
<td>1F8EH</td>
<td>7CH</td>
<td>00E8H</td>
<td>3EH</td>
</tr>
<tr>
<td>EPA_Mask†</td>
<td>1FA0H</td>
<td>7DH</td>
<td>00E0H</td>
<td>3EH</td>
</tr>
<tr>
<td>EPA_Mask1</td>
<td>1FA4H</td>
<td>7DH</td>
<td>00E4H</td>
<td>3EH</td>
</tr>
<tr>
<td>EPA_Pend†</td>
<td>1FA2H</td>
<td>7DH</td>
<td>00E2H</td>
<td>3EH</td>
</tr>
<tr>
<td>EPA_Pend1</td>
<td>1FA6H</td>
<td>7DH</td>
<td>00E6H</td>
<td>3EH</td>
</tr>
<tr>
<td>EPA0_CON</td>
<td>1F60H</td>
<td>7BH</td>
<td>00E0H</td>
<td>3DH</td>
</tr>
<tr>
<td>EPA1_CON†</td>
<td>1F64H</td>
<td>7BH</td>
<td>00E4H</td>
<td>3DH</td>
</tr>
<tr>
<td>EPA2_CON</td>
<td>1F68H</td>
<td>7BH</td>
<td>00E8H</td>
<td>3DH</td>
</tr>
<tr>
<td>EPA3_CON†</td>
<td>1F6CH</td>
<td>7BH</td>
<td>00ECH</td>
<td>3DH</td>
</tr>
<tr>
<td>EPA8_CON</td>
<td>1F80H</td>
<td>7CH</td>
<td>00E0H</td>
<td>3EH</td>
</tr>
<tr>
<td>EPA9_CON</td>
<td>1F84H</td>
<td>7CH</td>
<td>00E4H</td>
<td>3EH</td>
</tr>
<tr>
<td>EPA9_Time†</td>
<td>1F86H</td>
<td>7CH</td>
<td>00E6H</td>
<td>3EH</td>
</tr>
<tr>
<td>EPA0_Time†</td>
<td>1F62H</td>
<td>7BH</td>
<td>00E2H</td>
<td>3DH</td>
</tr>
<tr>
<td>EPA1_Time†</td>
<td>1F66H</td>
<td>7BH</td>
<td>00ECH</td>
<td>3DH</td>
</tr>
</tbody>
</table>

† Must be addressed as a word.
### Table 3-9. WSR Settings and Direct Addresses for Windowable SFRs (Continued)

<table>
<thead>
<tr>
<th>Register Mnemonic</th>
<th>Memory Location</th>
<th>32-byte Windows (00E0–00FFH)</th>
<th></th>
<th>64-byte Windows (00C0–00FFH)</th>
<th></th>
<th>128-byte Windows (0080–00FFH)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>WSR</td>
<td>Direct Address</td>
<td>WSR</td>
<td>Direct Address</td>
<td>WSR</td>
<td>Direct Address</td>
</tr>
<tr>
<td>WSR Direct Address</td>
<td>WSR Direct Address</td>
<td>WSR Direct Address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPA2_TIME†</td>
<td>1F6AH</td>
<td>7BH</td>
<td>00EAH</td>
<td>3DH</td>
<td>00EAH</td>
<td>1EH</td>
<td>00EAH</td>
</tr>
<tr>
<td>EPA3_TIME†</td>
<td>1F6EH</td>
<td>7BH</td>
<td>00EEH</td>
<td>3DH</td>
<td>00EEH</td>
<td>1EH</td>
<td>00EEH</td>
</tr>
<tr>
<td>EPA8_TIME†</td>
<td>1F82H</td>
<td>7CH</td>
<td>00E2H</td>
<td>3EH</td>
<td>00C2H</td>
<td>1FH</td>
<td>0082H</td>
</tr>
<tr>
<td>EPA9_TIME†</td>
<td>1F86H</td>
<td>7CH</td>
<td>00E6H</td>
<td>3EH</td>
<td>00C6H</td>
<td>1FH</td>
<td>0086H</td>
</tr>
<tr>
<td>EPAIPV</td>
<td>1FA8H</td>
<td>7DH</td>
<td>00E8H</td>
<td>3EH</td>
<td>00E8H</td>
<td>1FH</td>
<td>00A8H</td>
</tr>
<tr>
<td>P1_DIR</td>
<td>1FD2H</td>
<td>7EH</td>
<td>00F2H</td>
<td>3FH</td>
<td>00D2H</td>
<td>1FH</td>
<td>00D2H</td>
</tr>
<tr>
<td>P2_DIR</td>
<td>1FCBH</td>
<td>7EH</td>
<td>00EBH</td>
<td>3FH</td>
<td>00CBH</td>
<td>1FH</td>
<td>00CBH</td>
</tr>
<tr>
<td>P6_DIR</td>
<td>1FD3H</td>
<td>7EH</td>
<td>00F3H</td>
<td>3FH</td>
<td>00D3H</td>
<td>1FH</td>
<td>00D3H</td>
</tr>
<tr>
<td>P1_MODE</td>
<td>1FD0H</td>
<td>7EH</td>
<td>00F0H</td>
<td>3FH</td>
<td>00D0H</td>
<td>1FH</td>
<td>00D0H</td>
</tr>
<tr>
<td>P2_MODE</td>
<td>1FC9H</td>
<td>7EH</td>
<td>00E9H</td>
<td>3FH</td>
<td>00C9H</td>
<td>1FH</td>
<td>00C9H</td>
</tr>
<tr>
<td>P6_MODE</td>
<td>1FD1H</td>
<td>7EH</td>
<td>00F1H</td>
<td>3FH</td>
<td>00D1H</td>
<td>1FH</td>
<td>00D1H</td>
</tr>
<tr>
<td>P0_PIN</td>
<td>1FDAH</td>
<td>7EH</td>
<td>00FAH</td>
<td>3FH</td>
<td>00DAH</td>
<td>1FH</td>
<td>00DAH</td>
</tr>
<tr>
<td>P1_PIN</td>
<td>1FDBH</td>
<td>7EH</td>
<td>00F6H</td>
<td>3FH</td>
<td>00D6H</td>
<td>1FH</td>
<td>00D6H</td>
</tr>
<tr>
<td>P2_PIN</td>
<td>1FCFH</td>
<td>7EH</td>
<td>00EFH</td>
<td>3FH</td>
<td>00CFH</td>
<td>1FH</td>
<td>00CFH</td>
</tr>
<tr>
<td>P6_PIN</td>
<td>1FD7H</td>
<td>7EH</td>
<td>00F7H</td>
<td>3FH</td>
<td>00D7H</td>
<td>1FH</td>
<td>00D7H</td>
</tr>
<tr>
<td>P1_REG</td>
<td>1FDDH</td>
<td>7EH</td>
<td>00F4H</td>
<td>3FH</td>
<td>00D4H</td>
<td>1FH</td>
<td>00D4H</td>
</tr>
<tr>
<td>P2_REG</td>
<td>1FCDH</td>
<td>7EH</td>
<td>00EDH</td>
<td>3FH</td>
<td>00CDH</td>
<td>1FH</td>
<td>00CDH</td>
</tr>
<tr>
<td>P6_REG</td>
<td>1FD5H</td>
<td>7EH</td>
<td>00F5H</td>
<td>3FH</td>
<td>00D5H</td>
<td>1FH</td>
<td>00D5H</td>
</tr>
<tr>
<td>SBUF_RX</td>
<td>1FB8H</td>
<td>7DH</td>
<td>00F8H</td>
<td>3EH</td>
<td>00F8H</td>
<td>1FH</td>
<td>00B8H</td>
</tr>
<tr>
<td>SBUF_TX</td>
<td>1FBAH</td>
<td>7DH</td>
<td>00FAH</td>
<td>3EH</td>
<td>00FAH</td>
<td>1FH</td>
<td>00BAH</td>
</tr>
<tr>
<td>SP_BAUD†</td>
<td>1FBCCH</td>
<td>7DH</td>
<td>00FCH</td>
<td>3EH</td>
<td>00FCH</td>
<td>1FH</td>
<td>00BCH</td>
</tr>
<tr>
<td>SP_CON</td>
<td>1FBBH</td>
<td>7DH</td>
<td>00FBH</td>
<td>3EH</td>
<td>00FBH</td>
<td>1FH</td>
<td>00BBH</td>
</tr>
<tr>
<td>SP_STATUS</td>
<td>1FB9H</td>
<td>7DH</td>
<td>00F9H</td>
<td>3EH</td>
<td>00F9H</td>
<td>1FH</td>
<td>00B9H</td>
</tr>
<tr>
<td>SSIO0_BAUD</td>
<td>1FB4H</td>
<td>7DH</td>
<td>00F4H</td>
<td>3EH</td>
<td>00F4H</td>
<td>1FH</td>
<td>00B4H</td>
</tr>
<tr>
<td>SSIO0_BUF</td>
<td>1FB0H</td>
<td>7DH</td>
<td>00F0H</td>
<td>3EH</td>
<td>00F0H</td>
<td>1FH</td>
<td>00B0H</td>
</tr>
<tr>
<td>SSIO1_BUF</td>
<td>1FB2H</td>
<td>7DH</td>
<td>00F2H</td>
<td>3EH</td>
<td>00F2H</td>
<td>1FH</td>
<td>00B2H</td>
</tr>
<tr>
<td>SSIO0_CON</td>
<td>1FB1H</td>
<td>7DH</td>
<td>00F1H</td>
<td>3EH</td>
<td>00F1H</td>
<td>1FH</td>
<td>00B1H</td>
</tr>
<tr>
<td>SSIO1_CON</td>
<td>1FB3H</td>
<td>7DH</td>
<td>00F3H</td>
<td>3EH</td>
<td>00F3H</td>
<td>1FH</td>
<td>00B3H</td>
</tr>
<tr>
<td>T1CONTROL</td>
<td>1F98H</td>
<td>7CH</td>
<td>00F8H</td>
<td>3EH</td>
<td>00D8H</td>
<td>1FH</td>
<td>0098H</td>
</tr>
<tr>
<td>T2CONTROL</td>
<td>1F9CH</td>
<td>7CH</td>
<td>00FCH</td>
<td>3EH</td>
<td>00DCH</td>
<td>1FH</td>
<td>009CH</td>
</tr>
<tr>
<td>TIMER1†</td>
<td>1F9AH</td>
<td>7CH</td>
<td>00FAH</td>
<td>3EH</td>
<td>00DAH</td>
<td>1FH</td>
<td>009AH</td>
</tr>
<tr>
<td>TIMER2†</td>
<td>1F9EH</td>
<td>7CH</td>
<td>00FEH</td>
<td>3EH</td>
<td>00DEH</td>
<td>1FH</td>
<td>009EH</td>
</tr>
</tbody>
</table>

† Must be addressed as a word.
Standard and PTS Interrupts
CHAPTER 4
STANDARD AND PTS INTERRUPTS

4.1 INTERRUPT SOURCES, VECTORS, AND PRIORITIES

The interrupt structure of the 87C196CB is the same as that of the 8XC196NT. The only difference is that INT13, which was reserved on the 8XC196NT, supports the CAN peripheral.

Table 4-1 lists the 87C196CB’s interrupts sources, default priorities (30 is highest and 0 is lowest), and vector addresses. Figures 4-1 and 4-2 illustrate the interrupt mask and pending registers.

<table>
<thead>
<tr>
<th>Interrupt Source</th>
<th>Mnemonic</th>
<th>Interrupt Controller Service</th>
<th>PTS Service</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Name</td>
<td>Vector</td>
</tr>
<tr>
<td>Nonmaskable Interrupt</td>
<td>NMI</td>
<td>INT15</td>
<td>FF203EH</td>
</tr>
<tr>
<td>EXTINT Pin</td>
<td>EXTINT</td>
<td>INT14</td>
<td>FF203CH</td>
</tr>
<tr>
<td>CAN</td>
<td>CAN</td>
<td>INT13</td>
<td>FF203AH</td>
</tr>
<tr>
<td>SIO Receive</td>
<td>RI</td>
<td>INT12</td>
<td>FF2038H</td>
</tr>
<tr>
<td>SIO Transmit</td>
<td>TI</td>
<td>INT11</td>
<td>FF2036H</td>
</tr>
<tr>
<td>SSIO Channel 1 Transfer</td>
<td>SSIO1</td>
<td>INT10</td>
<td>FF2034H</td>
</tr>
<tr>
<td>SSIO Channel 0 Transfer</td>
<td>SSIO0</td>
<td>INT09</td>
<td>FF2032H</td>
</tr>
<tr>
<td>Slave Port Command Buff Full</td>
<td>CBF</td>
<td>INT08</td>
<td>FF2030H</td>
</tr>
<tr>
<td>Unimplemented Opcode</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Software TRAP Instruction</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Slave Port Input Buff Full</td>
<td>IBF</td>
<td>INT07</td>
<td>FF200EH</td>
</tr>
<tr>
<td>Slave Port Output Buff Empty</td>
<td>OBE</td>
<td>INT06</td>
<td>FF200CH</td>
</tr>
<tr>
<td>A/D Conversion Complete</td>
<td>AD_DONE</td>
<td>INT05</td>
<td>FF200AH</td>
</tr>
<tr>
<td>EPA Capture/Compare 0</td>
<td>EPA0</td>
<td>INT04</td>
<td>FF2008H</td>
</tr>
<tr>
<td>EPA Capture/Compare 1</td>
<td>EPA1</td>
<td>INT03</td>
<td>FF2006H</td>
</tr>
<tr>
<td>EPA Capture/Compare 2</td>
<td>EPA2</td>
<td>INT02</td>
<td>FF2004H</td>
</tr>
<tr>
<td>EPA Capture/Compare 3</td>
<td>EPA3</td>
<td>INT01</td>
<td>FF2002H</td>
</tr>
<tr>
<td>EPA Capture/Compare 4–9, EPA 0–9 Overrun, EPA Compare 0–1, Timer 1 Overflow, Timer 2 Overflow</td>
<td>EPAx</td>
<td>INT00</td>
<td>FF2000H</td>
</tr>
</tbody>
</table>

† PTS service is not recommended because the PTS cannot determine the source of shared interrupts.
The interrupt mask 1 (INT_MASK1) register enables or disables (masks) individual interrupt requests. (The EI and DI instructions enable and disable servicing of all maskable interrupts.) INT_MASK1 can be read from or written to as a byte register. PUSHA saves this register on the stack and POPA restores it.

<table>
<thead>
<tr>
<th>7:0 Setting a bit enables the corresponding interrupt.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bit Mnemonic</strong></td>
</tr>
<tr>
<td>NMI†</td>
</tr>
<tr>
<td>EXTINT</td>
</tr>
<tr>
<td>CAN</td>
</tr>
<tr>
<td>RI</td>
</tr>
<tr>
<td>TI</td>
</tr>
<tr>
<td>SSIO1</td>
</tr>
<tr>
<td>SSIO0</td>
</tr>
<tr>
<td>CBF</td>
</tr>
</tbody>
</table>

† NMI is always enabled. This nonfunctional mask bit exists for design symmetry with the INT_PEND1 register. Always write zero to this bit.

When hardware detects a pending interrupt, it sets the corresponding bit in the interrupt pending (INT_PEND or INT_PEND1) registers. When the vector is taken, the hardware clears the pending bit. Software can generate an interrupt by setting the corresponding interrupt pending bit.

<table>
<thead>
<tr>
<th>7:0 Any set bit indicates that the corresponding interrupt is pending. The interrupt bit is cleared when processing transfers to the corresponding interrupt vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bit Mnemonic</strong></td>
</tr>
<tr>
<td>NMI</td>
</tr>
<tr>
<td>EXTINT</td>
</tr>
<tr>
<td>CAN†</td>
</tr>
<tr>
<td>RI</td>
</tr>
<tr>
<td>TI</td>
</tr>
<tr>
<td>SSIO1</td>
</tr>
<tr>
<td>SSIO0</td>
</tr>
<tr>
<td>CBF</td>
</tr>
</tbody>
</table>

† NMI is always enabled. This nonfunctional mask bit exists for design symmetry with the INT_PEND1 register. Always write zero to this bit.
I/O Ports
CHAPTER 5
I/O PORTS

5.1 PORT 0 AND EPORT

The I/O ports of the 87C196CB are functionally identically to those of the 8XC196NT. However, the 87C196CB implements all eight pins of port 0, and the 100-pin 87C196CB also implements all eight pins of the EPORT. The associated registers have been modified to include bits corresponding to the upper nibble of the ports. Table 5-1 provides an overview of the 8XC196CB’s I/O ports. Figure 5-1 illustrates the port 0 pin state register, and Figures 5-2 through 5-5 illustrate the EPORT registers.

<table>
<thead>
<tr>
<th>Port</th>
<th>Bits</th>
<th>Type</th>
<th>Direction</th>
<th>Associated Peripheral(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port 0</td>
<td>8</td>
<td>Standard</td>
<td>Input-only</td>
<td>A/D converter</td>
</tr>
<tr>
<td>Port 1</td>
<td>8</td>
<td>Standard</td>
<td>Bidirectional</td>
<td>EPA and timers</td>
</tr>
<tr>
<td>Port 2</td>
<td>8</td>
<td>Standard</td>
<td>Bidirectional</td>
<td>SIO, interrupts, bus control, clock gen.</td>
</tr>
<tr>
<td>Port 3</td>
<td>8</td>
<td>Memory-mapped</td>
<td>Bidirectional</td>
<td>Address/data bus</td>
</tr>
<tr>
<td>Port 4</td>
<td>8</td>
<td>Memory-mapped</td>
<td>Bidirectional</td>
<td>Address/data bus</td>
</tr>
<tr>
<td>Port 5</td>
<td>8</td>
<td>Memory-mapped</td>
<td>Bidirectional</td>
<td>Bus control, slave port</td>
</tr>
<tr>
<td>Port 6</td>
<td>8</td>
<td>Standard</td>
<td>Bidirectional</td>
<td>EPA, SSIO</td>
</tr>
<tr>
<td>EPORT</td>
<td>4 (84-pin CB)</td>
<td>Memory mapped</td>
<td>Bidirectional</td>
<td>Extended address lines</td>
</tr>
<tr>
<td></td>
<td>8 (100-pin CB)</td>
<td>Memory mapped</td>
<td>Bidirectional</td>
<td></td>
</tr>
</tbody>
</table>

Each bit of the port 0 pin input (P0_PIN) register reflects the current state of the corresponding pin, regardless of the pin configuration.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>PIN7:0</td>
<td>Port 0 Pin x Input Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This bit contains the current state of P0.x.</td>
</tr>
</tbody>
</table>

Figure 5-1. Port x Pin Input (Px_PIN) Register
In I/O mode, each bit of the extended port I/O direction (EP_DIR) register controls the direction of the corresponding pin. Clearing a bit configures a pin as a complementary output; setting a bit configures a pin as either an input or an open-drain output. (Open-drain outputs require external pull-ups).

Any pin that is configured for its extended-address function is forced to the complementary output mode except during reset, hold, idle, and powerdown.

### Figure 5-2. Extended Port I/O Direction (EP_DIR) Register

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>PIN7:0</td>
<td>Extended Address Port Pin x Direction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This bit configures EPORT.x as a complementary output or an input/open-drain output.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = complementary output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = input or an open-drain output</td>
</tr>
</tbody>
</table>

### Figure 5-3. Extended Port Mode (EP_MODE) Register

Each bit of the extended port mode (EP_MODE) register controls whether the corresponding pin functions as a standard I/O port pin or as an extended-address signal. Setting a bit configures a pin as an extended-address signal; clearing a bit configures a pin as a standard I/O port pin.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>PIN7:0</td>
<td>Extended Address Port Pin x Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This bit determines the mode of EPORT.x:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = standard I/O port pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = extended-address signal</td>
</tr>
</tbody>
</table>
Each bit of the extended port input (EP_PIN) register reflects the current state of the corresponding pin, regardless of the pin configuration.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
</table>
| 7:0        | PIN7:0       | Extended Address Port Pin x Input
This bit contains the current state of EPORT.x.

Figure 5-4. Extended Port Input (EP_PIN) Register

Each bit of the extended port data output (EP_REG) register contains data to be driven out by the corresponding pin. When a pin is configured as standard I/O (EP_MODE.x = 0), the result of a CPU write to EP_REG is immediately visible on the pin.

During nonextended data accesses, EP_REG contains the value of the memory page that is to be accessed. For compatibility with software tools, clear the EP_REG bit for any EPORT pin that is configured as an extended-address signal (EP_MODE.x set).

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
</table>
| 7:0        | PIN7:0       | Extended Address Port Pin x Output
If EPORT.x is to be used as an output, write the data that it is to drive out.
If EPORT.x is to be used as an input, set this bit.
If EPORT.x is to be used as an address line, write the correct value for the memory page to be accessed by nonextended instructions.

Figure 5-5. Extended Port Data Output (EP_REG) Register
Analog-to-digital (A/D) Converter
6.1 ADDITIONAL A/D INPUT CHANNELS

The 87C196CB’s A/D converter is functionally identical to that of the 8XC196NT, but it has eight analog input channels instead of four. Table 6-1 lists the A/D signals. Figure 6-1 describes the command register and Figure 6-2 describes the result register.

<table>
<thead>
<tr>
<th>Port Pin</th>
<th>A/D Signal</th>
<th>A/D Signal Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0.7:0</td>
<td>ACH7:0</td>
<td>I</td>
<td>Analog inputs. See the “Voltage on Analog Input Pin” specification in the datasheet.</td>
</tr>
<tr>
<td>—</td>
<td>ANGND</td>
<td>GND</td>
<td>Reference Ground Must be connected for A/D converter and port operation.</td>
</tr>
<tr>
<td>—</td>
<td>$V_{REF}$</td>
<td>PWR</td>
<td>Reference Voltage Must be connected for A/D converter and port operation.</td>
</tr>
</tbody>
</table>
The **AD_COMMAND** register selects the A/D channel number to be converted, controls whether the A/D converter starts immediately or with an EPA command, and selects the conversion mode.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:6</td>
<td>—</td>
<td>Reserved; for compatibility with future devices, write zeros to these bits.</td>
</tr>
<tr>
<td>5:4</td>
<td>M1:0</td>
<td>A/D Mode†</td>
</tr>
<tr>
<td></td>
<td></td>
<td>These bits determine the A/D mode.</td>
</tr>
<tr>
<td></td>
<td>M1 M0</td>
<td>Mode</td>
</tr>
<tr>
<td></td>
<td>0 0</td>
<td>10-bit conversion</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>8-bit conversion</td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>threshold detect high</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>threshold detect low</td>
</tr>
<tr>
<td>3</td>
<td>GO</td>
<td>A/D Conversion Trigger††</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Writing this bit arms the A/D converter. The value that you write to it determines at what point a conversion is to start.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = EPA initiates conversion</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = start immediately</td>
</tr>
<tr>
<td>2:0</td>
<td>ACH2:0</td>
<td>A/D Channel Selection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Write the A/D conversion channel number to these bits. The 87C196CB has eight A/D channel inputs, numbered 0–7.</td>
</tr>
</tbody>
</table>

† While a threshold-detection mode is selected for an analog input pin, no other conversion can be started. If another value is loaded into **AD_COMMAND**, the threshold-detection mode is disabled and the new command is executed.

†† It is the act of writing to the GO bit, rather than its value, that starts a conversion. Even if the GO bit has the desired value, you must set it again to start a conversion immediately or clear it again to arm it for an EPA-initiated conversion.

---

**Figure 6-1. A/D Command (AD_COMMAND) Register**
The A/D result (AD_RESULT) register consists of two bytes. The high byte contains the eight most-significant bits from the A/D converter. The low byte contains the two least-significant bits from a ten-bit A/D conversion, indicates the A/D channel number that was used for the conversion, and indicates whether a conversion is currently in progress.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
</table>
| 15:6       | ADRLT9:0     | A/D Result  
These bits contain the A/D conversion result. |
| 5:4        | —            | Reserved. These bits are undefined. |
| 3          | STATUS       | A/D Status  
Indicates the status of the A/D converter. Up to 8 state times are required to set this bit following a start command. When testing this bit, wait at least the 8 state times.  
0 = A/D is idle  
1 = A/D conversion is in progress |
| 2:0        | ACH2:0       | A/D Channel Number  
These bits indicate the A/D channel number that was used for the conversion. The 87C196CB has eight A/D channel inputs, numbered 0–7 |

Figure 6-2. A/D Result (AD_RESULT) Register — Read Format
CAN Serial Communications Controller
CHAPTER 7
CAN SERIAL COMMUNICATIONS CONTROLLER

The 87C196CB has a peripheral not found in the 8XC196NT — the CAN (controller area network) peripheral. The CAN serial communications controller manages communications between multiple network nodes. This integrated peripheral is similar to Intel’s standalone 82527 CAN serial communications controller. It supports both the standard and the extended message frames specified by CAN 2.0 protocol parts A and B developed by Robert Bosch, GmbH. This chapter describes the integrated CAN controller and explains how to configure it. Consult Appendix A, “Signal Descriptions,” for detailed descriptions of the signals discussed in this chapter.

7.1 CAN FUNCTIONAL OVERVIEW

The integrated CAN controller transfers messages between network nodes according to the CAN protocol. The CAN protocol uses a multiple-master, contention-based bus configuration, which is also called CSMA/CR (carrier sense, multiple access, with collision resolution). Each CAN controller’s input and output pins are connected to a two-line CAN bus through which all communication takes place (Figure 7-1).

Figure 7-1. A System Using CAN Controllers
This bus configuration reduces point-to-point wiring requirements, making the CAN controller well suited to automotive and factory automation applications. In addition, it relieves the CPU of much of the communications burden while providing a high level of data integrity through error management logic.

The CAN controller (Figure 7-2) has one input pin, one output pin, control and status registers, and error detection and management logic.

Figure 7-2. CAN Controller Block Diagram
7.2 CAN CONTROLLER SIGNALS AND REGISTERS

Table 7-1 describes the CAN controller’s pins, and Table 7-2 describes the control and status registers.

**Table 7-1. CAN Controller Signals**

<table>
<thead>
<tr>
<th>Signal</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RXCAN</td>
<td>I</td>
<td>Receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This signal carries messages from other nodes on the CAN bus to the CAN controller.</td>
</tr>
<tr>
<td>TXCAN</td>
<td>O</td>
<td>Transmit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This signal carries messages from the CAN controller to other nodes on the CAN bus.</td>
</tr>
</tbody>
</table>

**Table 7-2. Control and Status Registers**

<table>
<thead>
<tr>
<th>Register Mnemonic</th>
<th>Register Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN_BTIME0††</td>
<td>1E3FH</td>
<td>Bit Timing 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Program this register to define the length of one time quantum and the maximum number of time quanta by which a bit time can be modified for resynchronization.</td>
</tr>
<tr>
<td>CAN_BTIME1††</td>
<td>1E4FH</td>
<td>Bit Timing 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Program this register to define the sample time and mode.</td>
</tr>
<tr>
<td>CAN_CON††</td>
<td>1E00H</td>
<td>Control</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Program this register to prevent transfers to and from the CAN bus, to enable and disable CAN interrupts, and to control write access to the bit timing registers.</td>
</tr>
<tr>
<td>CAN_EGMSK</td>
<td>1E08H, 1E09H,</td>
<td>Extended Global Mask</td>
</tr>
<tr>
<td></td>
<td>1E0AH, 1E0BH</td>
<td>Program this register to mask (“don’t care”) specific message identifier bits for extended message objects.</td>
</tr>
<tr>
<td>CAN_INT</td>
<td>1E5FH</td>
<td>CAN Interrupt Pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This read-only register indicates the source of the highest-priority pending interrupt.</td>
</tr>
<tr>
<td>CAN_MSGxCFG</td>
<td>1Ey6H</td>
<td>Message Object x Configuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Program this register to specify a message object’s data length, transfer direction, and identifier type.</td>
</tr>
<tr>
<td>CAN_MSGxCON0</td>
<td>1Ey0H</td>
<td>Message Object x Control 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Program this register to enable or disable the message object’s successful transmission (TX) and reception (RX) interrupts. Read this register to determine whether a message object is ready to transmit and whether an interrupt is pending.</td>
</tr>
</tbody>
</table>

††The CCE bit in CAN_CON must be set to enable write access to the bit timing registers.

††In register names, x = 1–15; in addresses, y = 1–F.
7.3 CAN CONTROLLER OPERATION

This section describes the address map, message objects, message frames (which contain message objects), error detection and management logic, and bit timing for CAN transmissions and receptions.
7.3.1 Address Map

The CAN controller has 256 bytes of RAM, containing 15 message objects and control and status registers at fixed addresses. Each message object occupies 15 consecutive bytes beginning at a base address that is a multiple of 16 bytes. The byte above each message object is reserved (indicated by a dash (—) character) or occupied by a control register. The lowest 16 bytes of RAM contain the remaining control and status registers (Table 7-3). This 256-byte section of memory can be windowed for register-direct access.

<table>
<thead>
<tr>
<th>Hex Address</th>
<th>Description</th>
<th>Hex Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1EFF</td>
<td>—</td>
<td>1E6F</td>
<td>—</td>
</tr>
<tr>
<td>1E0F–1EFE</td>
<td>Message Object 15</td>
<td>1E60–1E6E</td>
<td>Message Object 6</td>
</tr>
<tr>
<td>1EEF</td>
<td>—</td>
<td>1E5F</td>
<td>Interrupt Register</td>
</tr>
<tr>
<td>1E00–1EEE</td>
<td>Message Object 14</td>
<td>1E50–1E5E</td>
<td>Message Object 5</td>
</tr>
<tr>
<td>1EDF</td>
<td>—</td>
<td>1E4F</td>
<td>Bit Timing Register 1†</td>
</tr>
<tr>
<td>1E00–1EDE</td>
<td>Message Object 13</td>
<td>1E40–1E4E</td>
<td>Message Object 4</td>
</tr>
<tr>
<td>1ECF</td>
<td>—</td>
<td>1E3F</td>
<td>Bit Timing Register 0†</td>
</tr>
<tr>
<td>1E00–1ECE</td>
<td>Message Object 12</td>
<td>1E30–1E3E</td>
<td>Message Object 3</td>
</tr>
<tr>
<td>1E8F</td>
<td>—</td>
<td>1E2F</td>
<td>—</td>
</tr>
<tr>
<td>1E00–1EBE</td>
<td>Message Object 11</td>
<td>1E20–1E2E</td>
<td>Message Object 2</td>
</tr>
<tr>
<td>1EAF</td>
<td>—</td>
<td>1E1F</td>
<td>—</td>
</tr>
<tr>
<td>1E00–1EAE</td>
<td>Message Object 10</td>
<td>1E10–1E1E</td>
<td>Message Object 1</td>
</tr>
<tr>
<td>1E9F</td>
<td>—</td>
<td>1E0C–1E0F</td>
<td>Message 15 Mask Register</td>
</tr>
<tr>
<td>1E00–1E9E</td>
<td>Message Object 9</td>
<td>1E08–1E0B</td>
<td>Extended Global Mask Register</td>
</tr>
<tr>
<td>1E8F</td>
<td>—</td>
<td>1E06–1E07</td>
<td>Standard Global Mask Register</td>
</tr>
<tr>
<td>1E00–1E8E</td>
<td>Message Object 8</td>
<td>1E02–1E05</td>
<td>—</td>
</tr>
<tr>
<td>1E7F</td>
<td>—</td>
<td>1E01</td>
<td>Status Register</td>
</tr>
<tr>
<td>1E00–1E7E</td>
<td>Message Object 7</td>
<td>1E00</td>
<td>Control Register†</td>
</tr>
</tbody>
</table>

†The control register’s CCE bit must be set to enable write access to the bit timing registers.

7.3.2 Message Objects

The CAN controller includes 15 message objects, each of which occupies 15 bytes of RAM (Table 7-4). Message objects 1–14 can be configured to either transmit or receive messages, while message object 15 can only receive messages. Message objects 1–14 have only a single buffer, so if a second message is received before the CPU reads the first, the first message is overwritten. Message object 15 has two alternating buffers, so it can receive a second message while the first is being processed. However, if a third message is received while the CPU is reading the first, the second message is overwritten.
7.3.2.1 Receive and Transmit Priorities

The lowest-numbered message object always has the highest priority, regardless of the message identifier. When multiple messages are ready to transmit, the CAN controller transmits the message from the lowest-numbered message object first. When multiple message objects are capable of receiving the same message, the lowest-numbered message object receives it. For example, if all identifier bits are masked, message object 1 receives all messages.

7.3.2.2 Message Acceptance Filtering

The mask registers provide a method for developing an acceptance filtering strategy for a specific system. Software can program the mask registers to require an exact match on specific identifier bits while masking (“don’t care”) the remaining bits. Without a masking strategy, a message object could accept only those messages with an identical message identifier. With a masking strategy in place, a message object can accept messages whose identifiers are not identical.

The CAN controller filters messages by comparing an incoming message’s identifier with that of an enabled internal message object. The standard global mask register applies to messages with standard (11-bit) identifiers, while the extended global mask register applies to those with extended (29-bit) identifiers. The CAN controller applies the appropriate global mask to each incoming message identifier and checks for an acceptance match in message objects 1–14. If no match exists, it then applies the message 15 mask and checks for a match on message object 15. The message 15 mask is ANDed with the global mask, so any bit that is masked by the global mask is automatically masked for message 15.

The CAN controller accepts an incoming data message if the message’s identifier matches that of any enabled receive message object. It accepts an incoming remote message (request for data transmission) if the message’s identifier matches that of any enabled transmit message object. The remote message’s identifier is stored in the transmit message object, overwriting any masked bits. Table 7-5 shows an example.
7.3.3 Message Frames

A message object is contained within a *message frame* that adds control and error-detection bits to the content of the message object. The frame for an extended message differs slightly from that for a standard message, but they contain similar information. A *data frame* contains a message object with data to be transmitted; a *remote frame* is a request for another node to transmit a data frame, so it contains no data.

Figure 7-3 illustrates standard and extended message frames. Table 7-6 and Table 7-7 describe their contents and summarize the minimum message lengths. Actual message lengths may differ because the CAN controller adds bits during transmission (see “Error Detection and Management Logic” on page 7-9). After each message frame, an intermission field consisting of three recessive (1) bits separates messages. This intermission may be followed by a bus idle time.

<table>
<thead>
<tr>
<th>Transmit message object ID</th>
<th>110000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mask (0 = don’t care; 1 = must match)</td>
<td>00000000011</td>
</tr>
<tr>
<td>Received remote message object ID</td>
<td>00111111100</td>
</tr>
<tr>
<td>Resulting message object ID</td>
<td>00111111100</td>
</tr>
</tbody>
</table>

**Table 7-5. Effect of Masking on Message Identifiers**

**Figure 7-3. CAN Message Frames**
Table 7-6. Standard Message Frame

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Bit Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOF</td>
<td>Start-of-frame. A dominant (0) bit marks the beginning of a message frame.</td>
<td>1</td>
</tr>
<tr>
<td>Arbitration</td>
<td>11-bit message identifier.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RTR. Remote transmission request. Dominant (0) for data frames; recessive (1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>for remote frames.</td>
<td>12</td>
</tr>
<tr>
<td>Control</td>
<td>IDE. Identifier extension bit; always dominant (0).</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>r0. Reserved bit; always dominant (0).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DLC. Data length code. A 4-bit code indicating the number of data bytes (0–8).</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>Data. 1 to 8 bytes for data frames; 0 bytes for remote frames.</td>
<td>0–64</td>
</tr>
<tr>
<td>CRC</td>
<td>CRC code. A 15-bit CRC code plus a recessive (1) delimiter bit.</td>
<td>16</td>
</tr>
<tr>
<td>Ack</td>
<td>Acknowledgment. A dominant (0) bit sent by nodes receiving the frame plus a</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>recessive (1) delimiter bit.</td>
<td></td>
</tr>
<tr>
<td>End of frame</td>
<td>7 recessive (1) bits mark the end of a frame.</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Minimum standard message frame length (bits)</td>
<td>44–108</td>
</tr>
</tbody>
</table>

Table 7-7. Extended Message Frame

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Bit Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOF</td>
<td>Start-of-frame. A dominant (0) bit marks the beginning of a message frame.</td>
<td>1</td>
</tr>
<tr>
<td>Arbitration</td>
<td>11 bits of the 29-bit message identifier.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SRR. Substitute remote transmission request; always recessive (1).</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>IDE. Identifier extension bit; always recessive (1).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>18 bits of the 29-bit message identifier.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RTR. Remote transmission request; always recessive (1).</td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>r0. Reserved bit; always dominant (0).</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>r1. Reserved bit; always dominant (0).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DLC. Data length code. A 4-bit code indicating the number of data bytes (0–8).</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>Data. 1 to 8 bytes for data frames; 0 bytes for remote frames.</td>
<td>0–64</td>
</tr>
<tr>
<td>CRC</td>
<td>CRC code. A 15-bit CRC code plus a recessive (1) delimiter bit.</td>
<td>16</td>
</tr>
<tr>
<td>Ack</td>
<td>Acknowledgment. A dominant (0) bit sent by nodes receiving the frame plus a</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>recessive (1) delimiter bit.</td>
<td></td>
</tr>
<tr>
<td>End of frame</td>
<td>7 recessive (1) bits mark the end of a frame.</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Minimum extended message frame length (bits)</td>
<td>64–128</td>
</tr>
</tbody>
</table>
7.3.4 Error Detection and Management Logic

The CAN controller has several error detection mechanisms, including cyclical redundancy checking (CRC) and bit coding rules (stuffing and destuffing). The CAN controller generates a CRC code for transmitted messages and checks the CRC code of incoming messages. The CRC polynomial has been optimized for control applications with short messages.

After five consecutive bits of equal value are transmitted, a bit with the opposite polarity is added to the bit stream. This bit is called a stuff bit; by adding a transition, a stuff bit aids in synchronization. All message fields are stuffed except the CRC delimiter, the acknowledgment field, and the end-of-frame field.

Receiving nodes reject data from any message that is corrupted during transmission and send an error message via the CAN bus. Transmitting nodes monitor the CAN bus for error messages and automatically repeat a transmission if an error occurs. The following error types are detected:

- stuff error — more than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed
- form error — the fixed-format part of a received frame has the wrong format (for example, a reserved bit has the wrong value)
- acknowledgment error — this device transmitted a message, but it was not acknowledged by another node on the CAN bus. (The transmit error counter stops incrementing after 128 acknowledgment errors, so this error type does not cause a bus-off state.)
- bit 1 error — the CAN controller tried to send a recessive (logic 1) bit as part of a transmitted message (with the exception of the arbitration field), but the monitored CAN bus value was dominant (logic 0)
- bit 0 error — the CAN controller tried to send a dominant (logic 0) bit as part of a transmitted message (with the exception of the arbitration field), but the monitored CAN bus value was recessive (logic 1)
- CRC error — the CRC checksum received for an incoming message does not match the CRC value that the CAN controller calculated for the received data

The CAN status register indicates the type of the first transmission error that occurred on the CAN bus and whether an abnormal number of errors have occurred. Two counters (a receive error counter and a transmit error counter) track the number of errors. The status register’s warning bit is set when the receive or transmit error counter reaches 96; the bus-off bit is set when either counter reaches 256. If this occurs, the CAN controller isolates itself from the CAN bus (floats the TX pin). Software must clear the INIT bit in the control register (Figure 7-6 on page 7-13) to begin a bus-off recovery sequence.
7.3.5 Bit Timing

A message object consists of a series of bits transmitted in consecutive bit times. The CAN protocol specifies a bit time composed of four separate, nonoverlapping time segments: a synchronization delay segment, a propagation delay segment, and two phase delay segments (Figure 7-4 and Table 7-8). The CAN controller implements a bit time as three segments, combining PROP_SEG and PHASE_SEG1 into $t_{SEG1}$ (Figure 7-5 and Table 7-9). This implementation is identical to that of the 82527 CAN peripheral.

![Figure 7-4. A Bit Time as Specified by the CAN Protocol](image)

### Table 7-8. CAN Protocol Bit Time Segments

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNC_SEG</td>
<td>The synchronization delay segment allows for synchronization of the various nodes on the bus. An edge is expected to lie within this segment.</td>
</tr>
<tr>
<td>PROP_SEG</td>
<td>The propagation delay segment compensates for the physical delay times within the network. It is twice the sum of the signal’s propagation time on the bus line, the input comparator delay, and the output driver delay. The factor of two accounts for the requirement that all nodes monitor all bus transmissions for errors.</td>
</tr>
<tr>
<td>PHASE_SEG1</td>
<td>This segment compensates for edge phase errors. It can be lengthened or shortened by resynchronization.</td>
</tr>
<tr>
<td>PHASE_SEG2</td>
<td>This segment compensates for edge phase errors. It can be lengthened or shortened by resynchronization.</td>
</tr>
</tbody>
</table>
Table 7-9. CAN Controller Bit Time Segments

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>tSYNC_SEG</td>
<td>This time segment is equivalent to SYNC_SEG in the CAN protocol. Its length is one time quantum.</td>
</tr>
<tr>
<td>tTSEG1</td>
<td>This time segment is equivalent to the sum of PROP_SEG and PHASE_SEG1 in the CAN protocol. Its length is specified by the TSEG1 field in bit timing register 1. To allow for resynchronization, the sample point can be moved (tTSEG1 or tTSEG2 can be shortened and the other lengthened) by 1 to 4 time quanta, depending on the programmed value of the SJW field in bit timing register 0. The CAN controller samples the bus once or three times, depending on the value of the sampling mode (SPL) bit in bit timing register 0. In three-sample mode, the hardware lengthens tTSEG1 by 2 time quanta to allow time for the additional two bus samples. In this case, the &quot;sample point&quot; shown in Figure 7-5 is the time of the third sample; the first and second samples occur 2 and 1 time quanta earlier, respectively.</td>
</tr>
<tr>
<td>tTSEG2</td>
<td>This time segment is equivalent to PHASE_SEG2 in the CAN protocol. Its length is specified by the TSEG2 field in bit timing register 1. To allow for resynchronization, the sample point can be moved (tTSEG1 or tTSEG2 can be shortened and the other lengthened) by 1 to 4 time quanta, depending on the programmed value of the SJW field in bit timing register 0.</td>
</tr>
</tbody>
</table>
7.3.5.1 Bit Timing Equations

The bit timing equations of the integrated CAN controller are equivalent to those for the 82527 CAN peripheral with the DSC bit in the CPU interface register set (system clock divided by two). The following equations show the timing calculations for the integrated CAN controller and the 82527 CAN peripheral, respectively.

CAN Controller CAN bus frequency = \[
\frac{F_{osc}}{2 \times (BRP + 1) \times (3 + TSEG1 + TSEG2)}
\]

82527 CAN bus frequency = \[
\frac{F_{osc}}{(DSC + 1) \times (BRP + 1) \times (3 + TSEG1 + TSEG2)}
\]

where:

- \(F_{osc}\) = the input clock frequency on the XTAL1 pin, in MHz
- \(BRP\) = the value of the BRP bit in bit timing register 0
- \(TSEG1\) = the value of the TSEG1 field in bit timing register 0
- \(TSEG2\) = the value of the TSEG1 field in bit timing register 1

Table 7-10 defines the bit timing relationships of the CAN controller.

Table 7-10. Bit Timing Relationships

<table>
<thead>
<tr>
<th>Timing Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_{BITTIME})</td>
<td>(t_{SYNC_SEG} + t_{TSEG1} + t_{TSEG2})</td>
</tr>
<tr>
<td>(t_{XTAL1})</td>
<td>input clock period on XTAL1 (50 ns at 20 MHz operation)</td>
</tr>
<tr>
<td>(t_{q})</td>
<td>(2t_{XTAL1} \times (BRP + 1)), where BRP is a field in bit timing register 0 (valid values are 0–63)</td>
</tr>
<tr>
<td>(t_{SYNC_SEG})</td>
<td>(1t_{q})</td>
</tr>
<tr>
<td>(t_{TSEG1})</td>
<td>((TSEG1 + 1) \times t_{q}), where TSEG1 is a field in bit timing register 1 (valid values are 2–15)</td>
</tr>
<tr>
<td>(t_{TSEG2})</td>
<td>((TSEG2 + 1) \times t_{q}), where TSEG2 is a field in bit timing register 1 (valid values are 1–7)</td>
</tr>
<tr>
<td>(t_{SJW})</td>
<td>((SJW + 1) \times t_{q}), where SJW is a field in bit timing register 0 (valid values are 0–3)</td>
</tr>
<tr>
<td>(t_{PROP})</td>
<td>The portion of (t_{TSEG1}) that is equivalent to PROP_SEG as defined by the CAN protocol. Twice the maximum sum of the physical bus delay, input comparator delay, and output driver delay, rounded up to the nearest multiple of (t_{q}).</td>
</tr>
</tbody>
</table>
7.4 CONFIGURING THE CAN CONTROLLER

This section explains how to configure the CAN controller. Several registers combine to control the configuration: the CAN control register, the two bit timing registers, and the three mask registers.

7.4.1 Programming the CAN Control (CAN_CON) Register

The CAN control register (Figure 7-6) controls write access to the bit timing registers, enables and disables global interrupt sources (error, status change, and individual message object), and controls access to the CAN bus.

Program the CAN control (CAN_CON) register to control write access to the bit timing registers, to enable and disable CAN interrupts, and to control access to the CAN bus.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>—</td>
<td>Reserved; for compatibility with future devices, write zero to this bit.</td>
</tr>
</tbody>
</table>
| 6          | CCE          | Change Configuration Enable  
This bit controls whether software can write to the bit timing registers.  
0 = prohibit write access  
1 = allow write access |
| 5:4        | —            | Reserved; for compatibility with future devices, write zeros to these bits. |
| 3          | EIE          | Error Interrupt Enable  
This bit enables and disables the bus-off and warn interrupts.  
0 = disable bus-off and warn interrupts  
1 = enable bus-off and warn interrupts |
| 2          | SIE          | Status-change Interrupt Enable  
This bit enables and disables the successful reception (RXOK), successful transmission (TXOK), and error code change (LEC2:0) interrupts.  
0 = disable status-change interrupt  
1 = enable status-change interrupt  
When the SIE bit is set, the CAN controller generates a successful reception (RXOK) interrupt request each time it receives a valid message, even if no message object accepts it. |
Program the CAN control (CAN_CON) register to control write access to the bit timing registers, to enable and disable CAN interrupts, and to control access to the CAN bus.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IE</td>
<td>Interrupt Enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This bit globally enables and disables interrupts (error, status-change, and message object transmit and receive interrupts).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = disable interrupts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = enable interrupts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When the IE bit is set, an interrupt is generated only if the corresponding interrupt source’s enable bit (EIE or SIE in CAN_CON; TXIE or RXIE in CAN_MSGx_CON0) is also set. If the IE bit is clear, an interrupt request updates the CAN interrupt pending register, but does not generate an interrupt.</td>
</tr>
<tr>
<td>0</td>
<td>INIT</td>
<td>Software Initialization Enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Setting this bit isolates the CAN bus from the system. (If a transfer is in progress, it completes, but no additional transfers are allowed.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = software initialization disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = software initialization enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A hardware reset sets this bit, enabling you to configure the RAM without allowing any CAN bus activity. After a hardware reset or software initialization, clearing this bit completes the initialization. The CAN peripheral waits for a bus idle state (11 consecutive recessive bits) before participating in bus activities.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Software can set this bit to stop all receptions and transmissions on the CAN bus. (To prevent transmission of a specific message object while its contents are being updated, set the CPUUPD bit in the individual message object’s control register 1. See “Configuring Message Objects” on page 7-20.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Entering powerdown mode stops an in-progress CAN transmission immediately. To avoid stopping a CAN transmission while it is sending a dominant bit on the CAN bus, set the INIT bit before executing the IDLPD instruction.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The CAN peripheral also sets this bit to isolate the CAN bus when an error counter reaches 256. This isolation is called a bus-off condition. After a bus-off condition, clearing this bit initiates a bus-off recovery sequence, which clears the error counters. The CAN peripheral waits for 128 bus idle states (128 packets of 11 consecutive recessive bits), then resumes normal operation. (See “Bus-off State” on page 7-41.)</td>
</tr>
</tbody>
</table>

Figure 7-6. CAN Control (CAN_CON) Register (Continued)
7.4.2 Programming the Bit Timing 0 (CAN_BTIME0) Register

Bit timing register 0 (Figure 7-7) defines the length of one time quantum and the maximum amount by which the sample point can be moved (tTSEG1 or tTSEG2 can be shortened and the other lengthened) to compensate for resynchronization.

Program the CAN bit timing 0 (CAN_BTIME0) register to define the length of one time quantum and the maximum number of time quanta by which a bit time can be modified for resynchronization.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:6</td>
<td>SJW1:0</td>
<td>Synchronization Jump Width</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This field defines the maximum number of time quanta by which a resynchronization can modify tTSEG1 and tTSEG2. Valid programmed values are 0–3. The hardware adds 1 to the programmed value, so a “1” value causes the CAN peripheral to add or subtract 2 time quanta, for example. This adjustment has no effect on the total bit time; if tTSEG1 is increased by 2 tq, tTSEG2 is decreased by 2 tq, and vice versa.</td>
</tr>
<tr>
<td>5:0</td>
<td>BRP5:0</td>
<td>Baud-rate Prescaler</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This field defines the length of one time quantum (tq), using the following formula, where tXTAL1 is the input clock period on XTAL1. Valid programmed values are 0–63.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tq = 2tXTAL1 × (BRP + 1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example, at 20 MHz operation, the system clock period is 50 ns. Writing 3 to BRP achieves a time quanta of 400 ns; writing 1 to BRP achieves a time quanta of 200 ns.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tq = (2 × 50) × (3 + 1) = 400 ns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tq = (2 × 50) × (1 + 1) = 200 ns</td>
</tr>
</tbody>
</table>

† The CCE bit (CAN_CON.6) must be set to enable write access to this register.

Figure 7-7. CAN Bit Timing 0 (CAN_BTIME0) Register
7.4.3 Programming the Bit Timing 1 (CAN_BTIME1) Register

Bit timing register 1 (Figure 7-8) controls the time at which the bus is sampled and the number of samples taken. In single-sample mode, the bus is sampled once and the value of that sample is considered valid. In three-sample mode, the bus is sampled three times and the value of the majority of those samples is considered valid. Single-sample mode may achieve a faster transmission rate, but it is more susceptible to errors caused by noise on the CAN bus. Three-sample mode is less susceptible to noise-related errors, but it may be slower. If you specify three-sample mode, the hardware adds two time quanta to the TSEG1 value to allow time for two additional samples during $t_{TSEG1}$.

Program the CAN bit timing 1 (CAN_BTIME1) register to define the sample time and the sample mode. The CAN controller samples the bus during the last one (in single-sample mode) or three (in three-sample mode) time quanta of $t_{TSEG1}$, and initiates a transmission at the end of $t_{TSEG2}$. Therefore, specifying the lengths of $t_{TSEG1}$ and $t_{TSEG2}$ defines both the sample point and the transmission point.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
</table>
| 7          | SPL          | Sampling Mode  
This bit determines how many samples are taken to determine a valid bit value.  
0 = 1 sample  
1 = 3 samples, using majority logic |
| 6:4        | TSEG2††      | Time Segment 2  
This field determines the length of time that follows the sample point within a bit time. Valid programmed values are 1–7; the hardware adds 1 to this value. |
| 3:0        | TSEG1††      | Time Segment 1  
This field defines the length of time that precedes the sample point within a bit time. Valid programmed values are 2–15; the hardware adds 1 to this value. In three-sample mode, the hardware adds 2 time quanta to allow time for the two additional samples. |

† The CCE bit (CAN_CON.6) must be set to enable write access to this register.
†† For correct operation according to the CAN protocol, the total bit time must be at least 8 time quanta, so the sum of the programmed values of TSEG1 and TSEG2 must be at least 5. (The total bit time is the sum of $t_{SYNC} + t_{TSEG1} + t_{TSEG2}$. The length of $t_{SYNC}$ is 1 time quanta, and the hardware adds 1 to both TSEG1 and TSEG2. Therefore, if TSEG1 + TSEG2 = 5, the total bit length will be equal to 8 ($1+5+1+1$)). Table 7-11 lists additional conditions that must be met to maintain synchronization.

Figure 7-8. CAN Bit Timing 1 (CAN_BTIME1) Register
### Table 7-11. Bit Timing Requirements for Synchronization

<table>
<thead>
<tr>
<th>Bit Time Segment</th>
<th>Requirement</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{TSEG1}$</td>
<td>$\geq 3tq$</td>
<td>minimum tolerance with $1tq$ propagation delay allowance</td>
</tr>
<tr>
<td></td>
<td>$\geq t_{SJW} + t_{PROP}$</td>
<td>for single-sample mode</td>
</tr>
<tr>
<td></td>
<td>$\geq t_{SJW} + t_{PROP} + 2tq$</td>
<td>for three-sample mode</td>
</tr>
<tr>
<td>$t_{TSEG2}$</td>
<td>$\geq 2tq$</td>
<td>minimum tolerance</td>
</tr>
<tr>
<td></td>
<td>$\geq t_{SJW}$</td>
<td>if $t_{SJW} &gt; t_{TSEG2}$, sampling may occur after the bit time</td>
</tr>
</tbody>
</table>

### 7.4.4 Programming a Message Acceptance Filter

The mask registers provide a method for developing an acceptance filtering strategy. Without a filtering strategy, a message object could accept an incoming message only if their identifiers were identical. The mask registers allow a message object to ignore one or more bits of incoming message identifiers, so it can accept a range of message identifiers.

The standard global mask register (Figure 7-9) applies to messages with standard (11-bit) message identifiers, while the extended global mask register (Figure 7-10) applies to messages with extended (29-bit) identifiers. The message 15 mask register (Figure 7-11) provides an additional filter for message object 15, to allow it to accept a greater range of message identifiers than message objects 1–14 can. Clear a mask bit to accept either a zero or a one in that position.

The CAN controller applies the appropriate global mask to each incoming message identifier and checks for an acceptance match on message objects 1–14. If no match exists, it then applies the message 15 mask and checks for a match on message object 15.
Program the CAN standard global mask (CAN_SGMSK) register to mask ("don’t care") specific message identifier bits for standard message objects.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:13</td>
<td>MSK20:18</td>
<td>ID Mask</td>
</tr>
<tr>
<td></td>
<td></td>
<td>These bits individually mask incoming message identifier (ID) bits.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = mask the ID bit (accept either &quot;0&quot; or &quot;1&quot;)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = accept only an exact match</td>
</tr>
<tr>
<td>12:8</td>
<td>—</td>
<td>Reserved; for compatibility with future devices, write zeros to these bits.</td>
</tr>
<tr>
<td>7:0</td>
<td>MSK28:21</td>
<td>ID Mask</td>
</tr>
<tr>
<td></td>
<td></td>
<td>These bits individually mask incoming message identifier (ID) bits.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = mask the ID bit (accept either &quot;0&quot; or &quot;1&quot;)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = accept only an exact match</td>
</tr>
</tbody>
</table>
Program the CAN extended global mask (CAN_EGMSK) register to mask ("don’t care") specific message identifier bits for extended message objects.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:27</td>
<td>MSK4:0</td>
<td>ID Mask</td>
</tr>
<tr>
<td></td>
<td></td>
<td>These bits individually mask incoming message identifier (ID) bits.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = mask the ID bit (accept either “0” or “1”)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = accept only an exact match</td>
</tr>
<tr>
<td>26:24</td>
<td>—</td>
<td>Reserved; for compatibility with future devices, write zeros to these bits.</td>
</tr>
<tr>
<td>23:16, 15:8, 7:0</td>
<td>MSK12:5, MSK20:13, MSK28:21</td>
<td>ID Mask</td>
</tr>
<tr>
<td></td>
<td></td>
<td>These bits individually mask incoming message identifier (ID) bits.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = mask the ID bit (accept either “0” or “1”)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = accept only an exact match</td>
</tr>
</tbody>
</table>

Figure 7-10. CAN Extended Global Mask (CAN_EGMSK) Register
7.5 CONFIGURING MESSAGE OBJECTS

Each message object consists of a configuration register, a message identifier, control registers, and data registers (from zero to eight bytes of data). This section explains how to configure message objects and determine their status.

Program the CAN message 15 mask (CAN_MSK15) register to mask ("don't care") specific message identifier bits for message 15 in addition to those bits masked by a global mask (CAN_EGMSK or CAN_SGMSK).

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:27</td>
<td>ID Mask</td>
</tr>
<tr>
<td>26:24</td>
<td>Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.</td>
</tr>
<tr>
<td>23:16</td>
<td>ID Mask</td>
</tr>
<tr>
<td>15:8</td>
<td>These bits individually mask incoming message identifier (ID) bits.</td>
</tr>
<tr>
<td>7:0</td>
<td>0 = mask the ID bit (accept either &quot;0&quot; or &quot;1&quot;)</td>
</tr>
<tr>
<td></td>
<td>1 = accept only an exact match</td>
</tr>
</tbody>
</table>

Setting a CAN_MSK15 bit in any position that is cleared in the global mask register has no effect. The message 15 mask is ANDed with the global mask, so any "don't care" bits defined in a global mask are also "don't care" bits for message 15.

Figure 7-11. CAN Message 15 Mask (CAN_MSK15) Register
7.5.1 Specifying a Message Object’s Configuration

Each message object configuration register (Figure 7-12) specifies a message identifier type (standard or extended), transfer direction (transmit or receive), and data length (in bytes).

Set the XTD bit for a message object with an extended identifier; clear it for a message with a standard identifier. If you accidentally clear the XTD bit for a message that has an extended identifier, the CAN controller will clear the extended bits in the identification register. If you set the XTD bit for a message object, that message object cannot receive message objects with standard identifiers.

For a transmit message, set the DIR bit and write the number of programmed data bytes (0–8) to the DLC field. For a receive message, clear the DIR bit. The CAN controller stores the data length from the received message in the DLC field.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:4</td>
<td>DLC3:0</td>
<td>Data Length Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specify the number of data bytes this message object contains. Valid values are 0–8. The CAN controller updates a receive message object’s data length code after each reception to reflect the number of data bytes in the current message.</td>
</tr>
<tr>
<td>3</td>
<td>DIR</td>
<td>Direction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specify whether this message object is to be transmitted or is to receive a message object from a remote node.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = transmit</td>
</tr>
<tr>
<td>2</td>
<td>XTD</td>
<td>Extended Identifier Used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specify whether this message object’s identification registers contain an extended (29-bit) or a standard (11-bit) identifier.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = standard identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = extended identifier</td>
</tr>
<tr>
<td>1:0</td>
<td>—</td>
<td>Reserved; for compatibility with future devices, write zeros to these bits.</td>
</tr>
</tbody>
</table>

Figure 7-12. CAN Message Object x Configuration (CAN_MSGxCFG) Register
7.5.2 Programming the Message Object Identifier

Each message identifier register (Figure 7-13) specifies the message’s identifier. For messages with extended identifiers, write the identifier to bits ID28:0. For messages with standard identifiers, write the identifier to bits ID28:18. Software can change the identifier during normal operation without requiring a subsequent device reset. Clear the MSGVAL bit in the corresponding message control register 0 to prevent the CAN controller from accessing the message object while the modification takes place, then set the bit to allow access.

<table>
<thead>
<tr>
<th>CAN_MSGxID0–3 †</th>
<th>Address:</th>
<th>1Ex5H, 1Ex4H, 1Ex3H, 1Ex2H (x = 1–F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1–15 (87C196CB)</td>
<td>Reset State:</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>

Write the message object’s identifier to the CAN message object x identifier (CAN_MSGxID0–3) register. Software can change the identifier during normal operation. Clear the MSGVAL bit in the corresponding CAN_MSGxCON0 register to prevent the CPU from accessing the message object, change the identifier in CAN_MSGxID0–3, then set the MSGVAL bit to allow access.

![Figure 7-13. CAN Message Object x Identifier (CAN_MSGxID0–3) Register](image)

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:27</td>
<td>ID4:0</td>
<td>Message Identifier 17:0</td>
</tr>
<tr>
<td>23:16</td>
<td>ID12:5</td>
<td>These bits hold the 18 least-significant bits of an extended identifier. If you write an extended identifier to these bits, but specify a standard identifier (XTD = 0) in the corresponding message object’s configuration register (CAN_MSGxCFG), the CPU clears these bits (ID17:0).</td>
</tr>
<tr>
<td>12:8</td>
<td>ID17:13</td>
<td></td>
</tr>
<tr>
<td>26:24</td>
<td>—</td>
<td>Reserved; for compatibility with future devices, write zeros to these bits.</td>
</tr>
<tr>
<td>15:13</td>
<td>ID20:18</td>
<td>Message Identifier 28:18</td>
</tr>
<tr>
<td>7:0</td>
<td>ID28:21</td>
<td>These bits hold either an entire standard identifier or the 11 most-significant bits of an extended identifier.</td>
</tr>
</tbody>
</table>

† This register is the same as the arbitration register in the standalone 82527 CAN peripheral.
7.5.3 Programming the Message Object Control Registers

Each message object control register consists of four bit pairs — one bit of each pair is in true form and one is in complement form. This format allows software to set or clear any bit with a single write operation, without affecting the remaining bits. Table 7-12 shows how to interpret the bit-pair values.

<table>
<thead>
<tr>
<th>Access Type</th>
<th>MSB</th>
<th>LSB</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>0</td>
<td>0</td>
<td>Not allowed (indeterminate)</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>Clear (0)</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>Set (1)</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>No change</td>
</tr>
<tr>
<td>Read</td>
<td>0</td>
<td>1</td>
<td>Clear (0)</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>Set (1)</td>
</tr>
</tbody>
</table>

7.5.3.1 Message Object Control Register 0

Message object control register 0 (Figure 7-14) indicates whether an interrupt is pending, controls whether a successful transmission or reception generates an interrupt, and indicates whether a message object is ready to transmit.

7.5.3.2 Message Object Control Register 1

Message object control register 1 (Figure 7-15) indicates whether the message object contains new data, whether a message has been overwritten, whether the message is being updated, and whether a transmission or reception is pending. Message objects 1–14 have only a single buffer, so if a second message is received before the CPU reads the first, the first message is overwritten. Message object 15 has two alternating buffers, so it can receive a second message while the first is being processed. However, if a third message is received while the CPU is reading the first, the second message is overwritten.

7.5.4 Programming the Message Object Data

Each message object can have from zero to eight bytes of data. For transmit message objects, write the message data to the data registers (Figure 7-16). For receive message objects, the CAN controller stores the received data in these registers. The CAN controller writes random values to any unused data bytes during operation, so you should not use unused data bytes as scratch-pad memory.
Program the CAN message object x control 0 (CAN_MSGxCON0) register to indicate whether the message object is ready to transmit and to control whether a successful transmission or reception generates an interrupt. The least-significant bit-pair indicates whether an interrupt is pending.

This register consists of four bit-pairs — the most-significant bit of each pair is in true form and the least-significant bit is in complement form. This format allows software to set or clear any bit with a single write operation, without affecting the remaining bits.

**Figure 7-14. CAN Message Object x Control 0 (CAN_MSGxCON0) Register**

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:6</td>
<td>MSGVAL</td>
<td>Message Object Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set this bit-pair to indicate that a message object is valid (configured and ready for transmission or reception).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bit 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

The CAN peripheral will access a message object only if this bit-pair indicates that the message is valid. If multiple message objects have the same identifier, only one can be valid at any given time.

During initialization, software should clear this bit for any unused message objects. Software can clear this bit if a message is no longer needed or if you need to change a message object's contents or identifier.

<table>
<thead>
<tr>
<th>5:4</th>
<th>TXIE</th>
<th>Transmit Interrupt Enable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Receive message objects do not use this bit-pair.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For transmit message objects, set this bit-pair to enable the CAN peripheral to initiate a transmit (TX) interrupt after a successful transmission. You must also set the interrupt enable bit (CAN_CON.1) to enable the interrupt.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bit 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Program the CAN message object x control 0 (CAN_MSGxCON0) register to indicate whether the message object is ready to transmit and to control whether a successful transmission or reception generates an interrupt. The least-significant bit-pair indicates whether an interrupt is pending. This register consists of four bit-pairs — the most-significant bit of each pair is in true form and the least-significant bit is in complement form. This format allows software to set or clear any bit with a single write operation, without affecting the remaining bits.

### Bit Mnemonic Function

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:2</td>
<td>RXIE</td>
<td>Receive Interrupt Enable&lt;br&gt;Transmit message objects do not use this bit-pair.&lt;br&gt;For a receive message object, set this bit-pair to enable this message object to initiate a receive (RX) interrupt after a successful reception. You must also set the interrupt enable bit (CAN_CON.1) to enable the interrupt.</td>
</tr>
<tr>
<td>1:0</td>
<td>INT_PND</td>
<td>Interrupt Pending&lt;br&gt;This bit-pair indicates that this message object has initiated a transmit (TX) or receive (RX) interrupt. Software must clear this bit when it services the interrupt.</td>
</tr>
</tbody>
</table>

Figure 7-14. CAN Message Object x Control 0 (CAN_MSGxCON0) Register (Continued)
The CAN message object x control 1 (CAN_MSGxCON1) register indicates whether a message object has been updated, whether a message has been overwritten, whether the CPU is updating the message, and whether a transmission or reception is pending.

This register consists of four bit-pairs — the most-significant bit of each pair is in true form and the least-significant bit is in complement form. This format allows software to set or clear any bit with a single write operation, without affecting the remaining bits.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
</table>
| 7:6        | RMTPN | Remote Request Pending  
Receive message objects do not use this bit-pair.  
The CAN controller sets this bit-pair to indicate that a remote frame has requested the transmission of a transmit message object. If the CPUUPD bit-pair is clear, the CAN controller transmits the message object, then clears RMTPN. Setting RMTPN does not cause a transmission; it only indicates that a transmission is pending.  
| bit 7 | bit 6 | 0 1 no pending request; 1 0 a remote request is pending |

| 5:4        | TX_REQ | Transmission Request  
Set this bit-pair to cause a receive message object to transmit a remote frame (a request for transmission) or to cause a transmit object to transmit a data frame. Read this bit-pair to determine whether a transmission is in progress.  
| bit 5 | bit 4 | 0 1 no pending request; no transmission in progress; 1 0 transmission request; transmission in progress |

Figure 7-15. CAN Message Object x Control 1 (CAN_MSGxCON1) Register
The CAN message object \( x \) control 1 (CAN_MSGxCON1) register indicates whether a message object has been updated, whether a message has been overwritten, whether the CPU is updating the message, and whether a transmission or reception is pending.

This register consists of four bit-pairs — the most-significant bit of each pair is in true form and the least-significant bit is in complement form. This format allows software to set or clear any bit with a single write operation, without affecting the remaining bits.

### Bit Number 0:7

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
</table>
| 3:2        | MSGLST or CPUUPD | Message Lost (Receive)  
For a receive message object, the CAN controller sets this bit-pair to indicate that it stored a new message while the NEWDAT bit-pair was still set, overwriting the previous message.  
For a transmit message object, software should set this bit-pair to indicate that it is in the process of updating the message contents. This prevents a remote frame from triggering a transmission that would contain invalid data.  
| bit 3  | bit 2 | 0 1 | no overwrite occurred  
1 0 | a message was lost (overwritten)  
CPU Updating (Transmit)  
For a transmit message object, software should set this bit-pair to indicate that it is in the process of updating the message contents. This prevents a remote frame from triggering a transmission that would contain invalid data.  
| bit 3  | bit 2 | 0 1 | the message is valid  
1 0 | software is updating data  

| 1:0        | NEWDAT | New Data  
This bit-pair indicates whether a message object is valid (configured and ready for transmission).  
For receive message objects, the CAN peripheral sets this bit-pair when it stores new data into the message object.  
For transmit message objects, set this bit-pair to indicate that the message contents have been updated. Clearing CPUUPD prevents a remote frame from triggering a transmission that would contain invalid data.  
During initialization, clear this bit for any unused message objects.  

Figure 7-15. CAN Message Object \( x \) Control 1 (CAN_MSGxCON1) Register (Continued)
The CAN message object data (CAN_MSGxDATA0–7) registers contain data to be transmitted or data received. Any unused data bytes have random values that change during operation.

<table>
<thead>
<tr>
<th>CAN_MSGxDATA0–7</th>
<th>Address: 1ExEH, 1ExDH, 1ExCH, 1ExBH, 1ExAH, 1Ex9H, 1Ex8H, 1Ex7H (x = 1–F)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reset State: Unchanged</td>
</tr>
</tbody>
</table>

The CAN message object data (CAN_MSGxDATA0–7) registers contain data to be transmitted or data received. Any unused data bytes have random values that change during operation.

<table>
<thead>
<tr>
<th>87C196CB</th>
<th>CAN_MSGxDATA7</th>
<th>Data 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>87C196CB</th>
<th>CAN_MSGxDATA6</th>
<th>Data 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>87C196CB</th>
<th>CAN_MSGxDATA5</th>
<th>Data 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>87C196CB</th>
<th>CAN_MSGxDATA4</th>
<th>Data 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>87C196CB</th>
<th>CAN_MSGxDATA3</th>
<th>Data 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>87C196CB</th>
<th>CAN_MSGxDATA2</th>
<th>Data 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>87C196CB</th>
<th>CAN_MSGxDATA1</th>
<th>Data 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>87C196CB</th>
<th>CAN_MSGxDATA0</th>
<th>Data 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>Data</td>
</tr>
<tr>
<td></td>
<td>Each message object can use from zero to eight data registers to hold data to be transmitted or data received.</td>
</tr>
<tr>
<td></td>
<td>For receive message objects, these registers accept data during a reception.</td>
</tr>
<tr>
<td></td>
<td>For transmit message objects, write the data that is to be transmitted to these registers. The number of data bytes must match the DLC field in the CAN_MSGxCFG register. (For example, if CAN_MSG1DATA0, CAN_MSG1DATA1, CAN_MSG1DATA2, and CAN_MSG1DATA3 contain data, the DLC field in CAN_MSG1CFG must contain 04H.)</td>
</tr>
</tbody>
</table>

Figure 7-16. CAN Message Object Data (CAN_MSGxDATA0–7) Registers
7.6 ENABLING THE CAN INTERRUPTS

The CAN controller has a single interrupt input (INT13) to the interrupt controller. (Generally, PTS interrupt service is not useful for the CAN controller because the PTS cannot readily determine the source of the CAN controller’s multiplexed interrupts.) To enable the CAN controller’s interrupts, you must enable the interrupt source by setting the CAN bit in INT_MASK1 (see Table 7-2 on page 7-3) and globally enable interrupt servicing (by executing the EI instruction). In addition, you must set bits in the CAN control register (Figure 7-17) and the individual message objects’ control register 0 (Figure 7-18) to enable the individual interrupt sources within the CAN controller.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>—</td>
<td>Reserved; for compatibility with future devices, write zero to this bit.</td>
</tr>
<tr>
<td>6</td>
<td>CCE</td>
<td>Change Configuration Enable</td>
</tr>
<tr>
<td>5:4</td>
<td>—</td>
<td>Reserved; for compatibility with future devices, write zeros to these bits.</td>
</tr>
</tbody>
</table>
| 3          | EIE          | Error Interrupt Enable
This bit enables and disables the bus-off and warn interrupts.
0 = disable bus-off and warn interrupts
1 = enable bus-off and warn interrupts |
| 2          | SIE          | Status-change Interrupt Enable
This bit enables and disables the successful reception (RXOK), successful transmission (TXOK), and error code change (LEC2:0) interrupts.
0 = disable status-change interrupt
1 = enable status-change interrupt
When the SIE bit is set, the CAN controller generates a successful reception (RXOK) interrupt request each time it receives a valid message, even if no message object accepts it. |

Program the CAN control (CAN_CON) register to control write access to the bit timing registers, to enable and disable CAN interrupts, and to control access to the CAN bus.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>—</td>
<td>Reserved; for compatibility with future devices, write zero to this bit.</td>
</tr>
<tr>
<td>6</td>
<td>CCE</td>
<td>Change Configuration Enable</td>
</tr>
<tr>
<td>5:4</td>
<td>—</td>
<td>Reserved; for compatibility with future devices, write zeros to these bits.</td>
</tr>
</tbody>
</table>
| 3          | EIE          | Error Interrupt Enable
This bit enables and disables the bus-off and warn interrupts.
0 = disable bus-off and warn interrupts
1 = enable bus-off and warn interrupts |
| 2          | SIE          | Status-change Interrupt Enable
This bit enables and disables the successful reception (RXOK), successful transmission (TXOK), and error code change (LEC2:0) interrupts.
0 = disable status-change interrupt
1 = enable status-change interrupt
When the SIE bit is set, the CAN controller generates a successful reception (RXOK) interrupt request each time it receives a valid message, even if no message object accepts it. |
IE Interrupt Enable

This bit globally enables and disables interrupts (error, status-change, and message object transmit and receive interrupts).

0 = disable interrupts
1 = enable interrupts

When the IE bit is set, an interrupt is generated only if the corresponding interrupt source’s enable bit (EIE or SIE in CAN_CON; TXIE or RXIE in CAN_MSGx_CON) is also set. If the IE bit is clear, an interrupt request updates the CAN interrupt pending register, but does not generate an interrupt.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IE</td>
<td>Interrupt Enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This bit globally enables and disables interrupts (error, status-change, and message object transmit and receive interrupts).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = disable interrupts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = enable interrupts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When the IE bit is set, an interrupt is generated only if the corresponding interrupt source’s enable bit (EIE or SIE in CAN_CON; TXIE or RXIE in CAN_MSGx_CON) is also set. If the IE bit is clear, an interrupt request updates the CAN interrupt pending register, but does not generate an interrupt.</td>
</tr>
<tr>
<td>0</td>
<td>INIT</td>
<td>Software Initialization Enable</td>
</tr>
</tbody>
</table>

Figure 7-17. CAN Control (CAN_CON) Register (Continued)
When the SIE bit in the CAN control register is set, the CAN controller generates a successful reception (RXOK) interrupt request each time it receives a valid message, even if no message object accepts it. If you set both the SIE bit (Figure 7-17) and an individual message object’s RXIE bit (Figure 7-18), the CAN controller generates two interrupt requests each time a message object receives a message. The status change interrupt is useful during development to detect bus errors caused by noise or other hardware problems. However, you should disable this interrupt during normal operation in most applications. If the status change interrupt is enabled, each status change generates an interrupt request, placing an unnecessary burden on the CPU. To prevent redundant interrupt requests, enable the error interrupt sources (with the EIE bit) and enable the receive and transmit interrupts in the individual message objects.
7.7 DETERMINING THE CAN CONTROLLER’S INTERRUPT STATUS

A successful reception or transmission or a change in the status register can cause the CAN controller to generate an interrupt request. The INT_PEND1 register (see Table 7-2 on page 7-3) indicates whether a CAN interrupt request is pending. The CAN interrupt pending register (Figure 7-19) indicates the source of the request (either the status register or a specific message object). Your interrupt service routine should read the CAN_INT register to ensure that no additional interrupts are pending before executing the return instruction.

The CAN interrupt pending (CAN_INT) register indicates the source of the highest priority pending interrupt. If a status change generated the interrupt request, software can read the status register (CAN_STAT) to determine whether the interrupt request was caused by an abnormal error rate, a successful reception, a successful transmission, or a new error. If an individual message object generated the interrupt request, software can read the associated message object control 0 register (CAN_MSGxCON0). The INT_PND bit-pair will be set, indicating that a receive or transmit interrupt request is pending.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>Pending Interrupt</td>
</tr>
</tbody>
</table>

This field indicates the source of the highest priority pending interrupt.

<table>
<thead>
<tr>
<th>Value</th>
<th>Pending Interrupt</th>
<th>Priority (15 is highest; 0 is lowest)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00H</td>
<td>none</td>
<td>—</td>
</tr>
<tr>
<td>01H</td>
<td>status register</td>
<td>15</td>
</tr>
<tr>
<td>02H</td>
<td>message object 15</td>
<td>14</td>
</tr>
<tr>
<td>03H</td>
<td>message object 1</td>
<td>13</td>
</tr>
<tr>
<td>04H</td>
<td>message object 2</td>
<td>12</td>
</tr>
<tr>
<td>05H</td>
<td>message object 3</td>
<td>11</td>
</tr>
<tr>
<td>06H</td>
<td>message object 4</td>
<td>10</td>
</tr>
<tr>
<td>07H</td>
<td>message object 5</td>
<td>9</td>
</tr>
<tr>
<td>08H</td>
<td>message object 6</td>
<td>8</td>
</tr>
<tr>
<td>09H</td>
<td>message object 7</td>
<td>7</td>
</tr>
<tr>
<td>0AH</td>
<td>message object 8</td>
<td>6</td>
</tr>
<tr>
<td>0BH</td>
<td>message object 9</td>
<td>5</td>
</tr>
<tr>
<td>0CH</td>
<td>message object 10</td>
<td>4</td>
</tr>
<tr>
<td>0DH</td>
<td>message object 11</td>
<td>3</td>
</tr>
<tr>
<td>0EH</td>
<td>message object 12</td>
<td>2</td>
</tr>
<tr>
<td>0FH</td>
<td>message object 13</td>
<td>1</td>
</tr>
<tr>
<td>10H</td>
<td>message object 14</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 7-19. CAN Interrupt Pending (CAN_INT) Register

If a status change generated the interrupt (CAN_INT = 01H), software can read the CAN status register (Figure 7-20) to determine the source of the interrupt request.
The CAN status (CAN_STAT) register reflects the current status of the CAN peripheral.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
</table>
| 7          | BUSOFF       | Bus-off Status  
The CAN peripheral sets this read-only bit to indicate that it has isolated itself from the CAN bus (floated the TX pin) because an error counter has reached 256. A bus-off recovery sequence clears this bit and clears the error counters. (See “Bus-off State” on page 7-41.) |
| 6          | WARN         | Warning Status  
The CAN peripheral sets this read-only bit to indicate that an error counter has reached 96, indicating an abnormal rate of errors on the CAN bus. |
| 5          | —            | Reserved. This bit is undefined. |
| 4          | RXOK         | Reception Successful  
The CAN peripheral sets this bit to indicate that a message has been successfully received (error free, regardless of acknowledgment) since the bit was last cleared. Software must clear this bit when it services the interrupt. |
| 3          | TXOK         | Transmission Successful  
The CAN peripheral sets this bit to indicate that a message has been successfully transmitted (error free and acknowledged by at least one other node) since the bit was last cleared. Software must clear this bit when it services the interrupt. |
| 2:0        | LEC2:0       | Last Error Code  
This field indicates the error type of the first error that occurs in a message frame on the CAN bus. (“Error Detection and Management Logic” on page 7-9 describes the error types.) |

<table>
<thead>
<tr>
<th>LEC2</th>
<th>LEC1</th>
<th>LEC0</th>
<th>Error Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>no error</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>stuff error</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>form error</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>acknowledgment error</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>bit 1 error</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>bit 0 error</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>CRC error</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>unused</td>
</tr>
</tbody>
</table>

Figure 7-20. CAN Status (CAN_STAT) Register

If an individual message object caused the interrupt request (CAN_INT = 02–10H), software can read the associated message object control 0 register (Figure 7-21). The INT_PND bit-pair will be set, indicating that a receive or transmit interrupt request is pending.
Program the CAN message object x control 0 register (CAN_MSGxCON0) to indicate whether the message object is ready to transmit and to control whether a successful transmission or reception generates an interrupt. The most-significant bit-pair indicates whether an interrupt is pending. This register consists of four bit-pairs — the most-significant bit of each pair is in true form and the least-significant bit is in complement form. This format allows software to set or clear any bit with a single write operation, without affecting the remaining bits.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:6</td>
<td>MSGVAL</td>
<td>Message Object Valid</td>
</tr>
<tr>
<td>5:4</td>
<td>TXIE</td>
<td>Transmit Interrupt Enable</td>
</tr>
<tr>
<td>3:2</td>
<td>RXIE</td>
<td>Receive Interrupt Enable</td>
</tr>
<tr>
<td>1:0</td>
<td>INT_PND</td>
<td>Interrupt Pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This bit-pair indicates that the CAN peripheral has initiated a transmit (TX) or receive (RX) interrupt. Software must clear this bit when it services the interrupt.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 = no interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = an interrupt was generated</td>
</tr>
</tbody>
</table>

**Figure 7-21. CAN Message Object x Control 0 (CAN_MSGxCON0) Register**
7.8 FLOW DIAGRAMS

The flow diagrams in this section describe the steps that your software (shown as CPU) and the CAN controller execute to receive and transmit messages. Table 7-13 lists the register bits shown in the diagrams along with their associated registers and a cross-reference to the figure that describes them.

<table>
<thead>
<tr>
<th>Bit Mnemonic</th>
<th>Register Mnemonic</th>
<th>Figure and Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUUPD</td>
<td>CAN_MSGxCON1</td>
<td>Figure 7-15 on page 7-26</td>
</tr>
<tr>
<td>DIR</td>
<td>CAN_MSGxCFG</td>
<td>Figure 7-12 on page 7-21</td>
</tr>
<tr>
<td>DLC</td>
<td>CAN_MSGxCFG</td>
<td>Figure 7-12 on page 7-21</td>
</tr>
<tr>
<td>ID</td>
<td>CAN_MSGxID</td>
<td>Figure 7-13 on page 7-22</td>
</tr>
<tr>
<td>INT_PND</td>
<td>CAN_MSGxCON0</td>
<td>Figure 7-14 on page 7-24</td>
</tr>
<tr>
<td>MSGLTR</td>
<td>CAN_MSGxCON1</td>
<td>Figure 7-15 on page 7-26</td>
</tr>
<tr>
<td>MSGVAL</td>
<td>CAN_MSGxCON0</td>
<td>Figure 7-14 on page 7-24</td>
</tr>
<tr>
<td>NEWDAT</td>
<td>CAN_MSGxCON1</td>
<td>Figure 7-15 on page 7-26</td>
</tr>
<tr>
<td>RMT_PND</td>
<td>CAN_MSGxCON1</td>
<td>Figure 7-15 on page 7-26</td>
</tr>
<tr>
<td>RXIE</td>
<td>CAN_MSGxCON0</td>
<td>Figure 7-14 on page 7-24</td>
</tr>
<tr>
<td>TXIE</td>
<td>CAN_MSGxCON0</td>
<td>Figure 7-14 on page 7-24</td>
</tr>
<tr>
<td>TX_REG</td>
<td>CAN_MSGxCON1</td>
<td>Figure 7-15 on page 7-26</td>
</tr>
<tr>
<td>XTD</td>
<td>CAN_MSGxCFG</td>
<td>Figure 7-12 on page 7-21</td>
</tr>
</tbody>
</table>
Figure 7-22. Receiving a Message for Message Objects 1–14 — CPU Flow

- Power Up: (All bits undefined)
- Initialization:
  - MSGVAL := 1
  - INT_PND := 0
  - TXIE := (Application specific)
  - RXIE := (Application specific)
  - NEWDAT := 0
  - RMTPND := 0
  - TX_REQ := 0
  - MSGLST := 0
  - DLC := (don’t care)
  - DIR := 0 (receive)
  - XTD := (Application specific)
  - ID := (Application specific)
- Process:
  - NEWDAT := 0
  - Process message contents.
  - NEWDAT = 1?
    - Yes: Restart Process
    - No:
      - Request update?
        - Yes: TX_REQ := 1
        - No: Restart Process
Figure 7-23. Receiving a Message for Message Object 15 — CPU Flow
Figure 7-24. Receiving a Message — CAN Controller Flow

- **Bus idle?**
  - Yes
  - No
- **TX_REQ=1? MSGLST=0?**
  - No
  - Yes
  - NewDAT := 0; Load identifier and control into buffer; Send remote frame
- **Transmission successful?**
  - No
  - Yes
  - TX_REQ := 0; RMTPND := 0
- **RXIE = 1?**
  - No
  - Yes
  - INT_PND := 1
- **Bus idle?**
  - Yes
  - No
- **Received frame with same identifier as this message object?**
  - No
  - Yes
  - NewDAT := 1
- **MSGLST := 1**
  - Store message:
    - NewDAT := 1
    - TX_REQ := 0
    - RMTPND := 0
  - RXIE := 1?
  - No
  - Yes
  - INT_PND := 1
Figure 7-25. Transmitting a Message — CPU Flow
Figure 7-26. Transmitting a Message — CAN Controller Flow
7.9 DESIGN CONSIDERATIONS

This section outlines design considerations for the CAN controller.

7.9.1 Hardware Reset

A hardware reset clears the error management counters and the bus-off state and leaves the registers with the values listed in Table 7-14.

<table>
<thead>
<tr>
<th>Table 7-14. Register Values Following Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
</tr>
<tr>
<td>Control</td>
</tr>
<tr>
<td>Status</td>
</tr>
<tr>
<td>Standard Global Mask</td>
</tr>
<tr>
<td>Extended Global Mask</td>
</tr>
<tr>
<td>Message 15 Mask</td>
</tr>
<tr>
<td>Bit Timing 0</td>
</tr>
<tr>
<td>Bit Timing 1</td>
</tr>
<tr>
<td>Interrupt</td>
</tr>
<tr>
<td>Message Object x</td>
</tr>
</tbody>
</table>

7.9.2 Software Initialization

The software initialization state allows software to configure the CAN controller’s RAM without risk of messages being received or transmitted during this time. Setting the INIT bit in the control register causes the CAN controller to enter the software initialization state. Either a hardware reset or a software write can set the INIT bit. While INIT is set, all message transfers to and from the CAN controller are stopped and the error counters and bit timing registers are unchanged. Your software should clear the INIT bit to cause the CAN controller to exit the software initialization state. At this time, the CAN controller synchronizes itself to the CAN bus by waiting for a bus idle state (11 consecutive recessive bits) before participating in bus activities.

7.9.3 Bus-off State

If an error counter reaches 256, the CAN controller isolates itself from the CAN bus, sets the BUSOFF bit in the status register, and sets the INIT bit in the control register. While INIT is set, all message transfers to and from the CAN controller are stopped; the error counters and bit timing registers are unchanged. Software must clear the INIT bit to initiate the bus-off recovery sequence.
The CAN controller synchronizes itself to the CAN bus by waiting for 128 bus idle states (128 occurrences of 11 consecutive recessive bits) before participating in bus activities. During this sequence, the CAN controller writes a bit 0 error code to the LEC2:0 bits of the status register each time it receives a recessive bit. Software can check the status register to determine whether the CAN bus is stuck in a dominant state. Once the CAN controller is resynchronized with the CAN bus, it clears the BUSOFF bit and starts transferring messages again.
8

Special Operating Modes
8.1 CLOCK CIRCUITRY

The 87C196CB’s idle, powerdown, and ONCE modes are the same as those of the 8XC196NT. The only difference is in the way that the power saving modes disable the clock circuitry (Figure 8-1).

Figure 8-1. Clock Circuitry
Interfacing with External Memory
CHAPTER 9
INTERFACING WITH EXTERNAL MEMORY

The 87C196CB’s external memory interface is similar to that of the 8XC196NT. However, the 87C196CB supports only two of the bus timing modes, modes 3 and 0. In addition, the 100-pin 87C196CB has four additional address pins (A23:20).

9.1 ADDRESS PINS

The 100-pin 87C196CB has 24 available address pins, A23:16 and AD15:0. The A23:20 timings are identical to those of A19:16. During the CCB fetch, the 100-pin 87C196CB strongly drives 0FFH on A23:16. The 84-pin 87C196CB strongly drives 0FH on A19:16, as does the 8XC196NT.

9.2 BUS TIMING MODES

The 87C196CB implements only modes 3 and 0. Table 9-1 and Figure 9-1 compare the timings of these two modes. Figure 9-2 illustrates the CCB1 register, which selects the mode.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Timing Specifications †</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T_{CLLH}</td>
</tr>
<tr>
<td>Mode 3</td>
<td>0</td>
</tr>
<tr>
<td>Mode 0</td>
<td>0</td>
</tr>
</tbody>
</table>

† These are ideal timing values for purposes of comparison only. They do not include internal device delays. Consult the datasheet for current device specifications.
Figure 9-1. Modes 0 and 3 Timings
The chip configuration 1 (CCR1) register enables the watchdog timer and selects the bus timing mode. Two of its bits combine with three bits of CCR0 to control wait states and bus width. Another bit controls whether CCR2 is loaded.

### Bit Number | Bit Mnemonic | Function |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7:6</td>
<td>MSEL1:0</td>
<td>External Access Timing Mode Select</td>
</tr>
<tr>
<td></td>
<td></td>
<td>These bits control the bus-timing modes.</td>
</tr>
<tr>
<td></td>
<td>MSEL1</td>
<td>MSEL0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>standard mode plus one wait state</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>reserved</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>reserved</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>standard mode</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>To guarantee proper operation, write zero to this bit.</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>To guarantee proper operation, write one to this bit.</td>
</tr>
<tr>
<td>3</td>
<td>WDE</td>
<td>Watchdog Timer Enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Selects whether the watchdog timer is always enabled or enabled the first time it is cleared.</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>always enabled</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>enabled first time it is cleared</td>
</tr>
<tr>
<td>2</td>
<td>BW1</td>
<td>Buswidth Control</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This bit, along with the BW0 bit (CCR0.1), selects the bus width.</td>
</tr>
<tr>
<td></td>
<td>BW1</td>
<td>BW0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>illegal</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>16-bit only</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>8-bit only</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>BUSWIDTH pin controlled</td>
</tr>
</tbody>
</table>

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset, unless the microcontroller is entering programming modes, in which case the programming chip configuration bytes (PCCBs) are used. The CCBs reside in internal nonvolatile memory at addresses FF2018H (CCB0), FF201AH (CCB1), and FF201CH (CCB2).
This bit, along with IRC0 (CCR0.4) and IRC1 (CCR0.5), limits the number of wait states that can be inserted while the READY pin is held low. Wait states are inserted into the bus cycle either until the READY pin is pulled high or until this internal number is reached.

<table>
<thead>
<tr>
<th>IRC2</th>
<th>IRC1</th>
<th>IRC0</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>zero wait states</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>1</td>
<td>illegal</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>illegal</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>one wait state</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>two wait states</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>three wait states</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>READY pin controlled</td>
</tr>
</tbody>
</table>

If you choose the READY pin controlled option, you must keep P5.6 configured as a special-function input, and add external hardware to count wait states and release READY within a specified time.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>LDCCB2</td>
<td>Load CCB2</td>
</tr>
</tbody>
</table>

Setting this bit causes CCB2 to be read.

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset, unless the microcontroller is entering programming modes, in which case the programming chip configuration bytes (PCCBs) are used. The CCBs reside in internal nonvolatile memory at addresses FF2018H (CCB0), FF201AH (CCB1), and FF201CH (CCB2).
10

Programming the Nonvolatile Memory
The 87C196CB has 56 Kbytes of OTPROM (FF2000–FFFFFFH), while the 8XC196NT has only 32 Kbytes (FF2000–FF9FFFH). The 87C196CB’s programming signals, registers, and procedures are the same as those of the 8XC196NT. This chapter describes the differences in memory mapping and programming circuits for the 87C196CB.

10.1 SIGNATURE WORD AND PROGRAMMING VOLTAGES

The 87C196CB’s programming voltages are the same of those of the 8XC196NT; however, the signature word differs. Table 10-1 lists the signature word and programming voltages.

<table>
<thead>
<tr>
<th>Device</th>
<th>Signature Word</th>
<th>Programming $V_{cc}$</th>
<th>Programming $V_{pp}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>87C196CB</td>
<td>0070H</td>
<td>87CBH</td>
<td>0072H 40H</td>
</tr>
</tbody>
</table>

10.2 MEMORY MAP FOR SLAVE PROGRAMMING MODE

Because the 87C196CB has an additional 24 Kbytes of OTPROM, its memory map (Table 10-2) differs from that of the 8XC196NT. The remaining information on slave programming is correct for the 87C196CB.
10.3 MEMORY MAP AND CIRCUIT FOR AUTO PROGRAMMING

Because the 87C196CB has an additional 24 Kbytes of OTPROM, its auto programming memory map (Table 10-3) and circuit (Figure 10-1) differ from those of the 8XC196NT.

### Table 10-2. Slave Programming Mode Memory Map

<table>
<thead>
<tr>
<th>Description</th>
<th>Address</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTPROM</td>
<td>FF2000–FFFFFH</td>
<td>OTPROM Cells</td>
</tr>
<tr>
<td>OFD</td>
<td>0778H</td>
<td>OTPROM Cell</td>
</tr>
<tr>
<td>DED†</td>
<td>0758H</td>
<td>UPROM Cell</td>
</tr>
<tr>
<td>DEI†</td>
<td>0718H</td>
<td>UPROM Cell</td>
</tr>
<tr>
<td>PCCB</td>
<td>0218H</td>
<td>Test EPROM</td>
</tr>
<tr>
<td>Programming V&lt;sub&gt;CC&lt;/sub&gt;</td>
<td>0072H</td>
<td>Read Only</td>
</tr>
<tr>
<td>Programming V&lt;sub&gt;PP&lt;/sub&gt;</td>
<td>0073H</td>
<td>Read Only</td>
</tr>
<tr>
<td>Signature word</td>
<td>0070H</td>
<td>Read Only</td>
</tr>
</tbody>
</table>

†These bits program the UPROM cells. Once these bits are programmed, they cannot be erased, and dynamic failure analysis of the device is impossible.

### Table 10-3. Auto Programming Memory Map

<table>
<thead>
<tr>
<th>Address Output from 87C196CB (A15:0)</th>
<th>Internal OTPROM Address</th>
<th>Address Using Circuit in Figure 10-1 (P1.3:1, A13:0)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4014H</td>
<td>N/A</td>
<td>00014H</td>
<td>Programming pulse width (PPW) LSB.</td>
</tr>
<tr>
<td>4015H</td>
<td>N/A</td>
<td>00015H</td>
<td>Programming pulse width (PPW) MSB.</td>
</tr>
<tr>
<td>4000–7FFFH</td>
<td>FF2000–FF5FFFH</td>
<td>04000–07FFFH</td>
<td>First 16 Kbytes of code and data.</td>
</tr>
<tr>
<td>4000–7FFFH</td>
<td>FF6000–FF9FFFH</td>
<td>08000–0BFFFH</td>
<td>Second 16 Kbytes of code and data.</td>
</tr>
<tr>
<td>4000–7FFFH</td>
<td>FF8000–FFDFFFH</td>
<td>0C000–0FFFFH</td>
<td>Third 16 Kbytes of code and data.</td>
</tr>
<tr>
<td>4000–5FFFH</td>
<td>FFE000–FFFFFH</td>
<td>10000–11FFFFH</td>
<td>Last 8 Kbytes of code and data.</td>
</tr>
</tbody>
</table>
10.4 MEMORY MAP FOR SERIAL PORT PROGRAMMING

The 87C196CB’s memory map (Table 10-4) for serial port programming differs from that of the 8XC196NT. The remaining information on serial port programming is correct for the 87C196CB.
The lower 24 Kbytes of OTPROM (FF2000–FF7FFFH) are remapped to A000–FFFFH, and the upper 32 Kbytes (FF8000–FFFFFFH) are mapped to 8000–FFFFH. A bank switching mechanism differentiates between the two address ranges. The most-significant bit of an otherwise reserved byte register (location 1FF9H) selects the bank. Bank 0 is the lower 24 Kbytes, and bank 1 is the upper 32 Kbytes. To program the lower 24 Kbytes, you must write 00H to location 1FF9H. To program the upper 32 Kbytes, you must write 80H to location 1FF9H. (See page 10-4 for the required command sequences.)

**WARNING**
Writing any value other than 00H or 80H to location 1FF9H will cause the microcontroller to enter an unsupported test mode.

### 10.4.1 Selecting Bank 0 (FF2000–FF7FFFH)

Send the following RISM command sequence to select bank 0.

**Code Description**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1F</td>
<td>DATA. High byte of address to DATA register.</td>
</tr>
<tr>
<td>F9</td>
<td>DATA. Low byte of address to DATA register.</td>
</tr>
<tr>
<td>0A</td>
<td>DATA_TO_ADDR. Move address from DATA register to ADDR register.</td>
</tr>
<tr>
<td>00</td>
<td>SET_DLE_FLAG. The next data byte is &lt;1FH.</td>
</tr>
<tr>
<td>00</td>
<td>DATA. Data to clear the most-significant bit.</td>
</tr>
<tr>
<td>07</td>
<td>WRITE_BYTE. Move data from the DATA register to memory location 1FF9H.</td>
</tr>
</tbody>
</table>

### 10.4.2 Selecting Bank 1 (FF8000–FFFFFFH)

Send the following RISM command sequence to select bank 1.

**Code Description**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1F</td>
<td>DATA. High byte of address to DATA register.</td>
</tr>
<tr>
<td>F9</td>
<td>DATA. Low byte of address to DATA register.</td>
</tr>
<tr>
<td>0A</td>
<td>DATA_TO_ADDR. Move address from DATA register to ADDR register.</td>
</tr>
<tr>
<td>80</td>
<td>DATA. Data to set the most-significant bit.</td>
</tr>
<tr>
<td>07</td>
<td>WRITE_BYTE. Move data from the DATA register to memory location 1FF9H.</td>
</tr>
</tbody>
</table>

---

Table 10-4. Serial Port Programming Mode Memory Map

<table>
<thead>
<tr>
<th>Description</th>
<th>Normal Operation</th>
<th>Serial Port Programming Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal OTPROM</td>
<td>FF2000–FF7FFFH</td>
<td>A000–FFFFH (bank 0; 1FF9H = 00H)</td>
</tr>
<tr>
<td></td>
<td>FF8000–FFFFFFFH</td>
<td>8000–FFFFH (bank 1; 1FF9H = 80H)</td>
</tr>
<tr>
<td>External memory</td>
<td>—</td>
<td>4000–7FFFH</td>
</tr>
<tr>
<td>Do not address</td>
<td>—</td>
<td>2400–3FFFH</td>
</tr>
<tr>
<td>Test ROM and RISM</td>
<td>—</td>
<td>2000–23FFFH</td>
</tr>
</tbody>
</table>

---

The lower 24 Kbytes of OTPROM (FF2000–FF7FFFH) are remapped to A000–FFFFH, and the upper 32 Kbytes (FF8000–FFFFFFH) are mapped to 8000–FFFFH. A bank switching mechanism differentiates between the two address ranges. The most-significant bit of an otherwise reserved byte register (location 1FF9H) selects the bank. Bank 0 is the lower 24 Kbytes, and bank 1 is the upper 32 Kbytes. To program the lower 24 Kbytes, you must write 00H to location 1FF9H. To program the upper 32 Kbytes, you must write 80H to location 1FF9H. (See page 10-4 for the required command sequences.)

---

10-4
Signal Descriptions
APPENDIX A
SIGNAL DESCRIPTIONS

A.1 FUNCTIONAL GROUPINGS OF SIGNALS

Table A-1 lists the signals for the 87C196CB, grouped by function. A diagram of each package that is currently available shows the pin location of each signal.

NOTE
As new packages are supported, they will be added to the datasheets first. If your package type is not shown in this appendix, refer to the latest datasheet to find the pin locations.

Table A-1. 87C196CB Signals Arranged by Functional Categories

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>Processor Control</th>
<th>Bus Control &amp; Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPORT.7:0 (100-pin CB)</td>
<td>EA#</td>
<td>ALE/ADV#</td>
</tr>
<tr>
<td>EPORT.3:0 (84-pin CB)</td>
<td>EXTINT</td>
<td>BHE#/WRH#</td>
</tr>
<tr>
<td>P0.7:0/ACH7:0</td>
<td>NMI</td>
<td>BREQ#</td>
</tr>
<tr>
<td>P1.0/EPA0/T2CLK</td>
<td>ONCE#</td>
<td>BUSWIDTH</td>
</tr>
<tr>
<td>P1.1/EPA1</td>
<td>RESET#</td>
<td>CLKOUT</td>
</tr>
<tr>
<td>P1.2/EPA2/T2DIR</td>
<td>SLPINT†</td>
<td>HOLD#</td>
</tr>
<tr>
<td>P1.7:3/EPA7:3</td>
<td>XTAL1</td>
<td>HLDA#</td>
</tr>
<tr>
<td>P2.0/TXD</td>
<td>XTAL2</td>
<td>INST</td>
</tr>
<tr>
<td>P2.1/RXD</td>
<td>PLLLEN</td>
<td>INTOUT#</td>
</tr>
<tr>
<td>P2.7:2</td>
<td>READY</td>
<td>READY</td>
</tr>
<tr>
<td>P3.7:0</td>
<td>A23:16 (100-pin CB)</td>
<td>RD#</td>
</tr>
<tr>
<td>P4.7:0</td>
<td>A19:16 (84-pin CB)</td>
<td>SLPALE†</td>
</tr>
<tr>
<td>P5.7:0</td>
<td>AD15:0</td>
<td>SLPCS#†</td>
</tr>
<tr>
<td>P6.0/EPA8/COMP0</td>
<td>SLP7:0†</td>
<td>SLPWR#†</td>
</tr>
<tr>
<td>P6.1/EPA9/COMP1</td>
<td>SLPWR#†</td>
<td>SLPRD#†</td>
</tr>
<tr>
<td>P6.2/T1CLK</td>
<td>AINC#</td>
<td>Programming Control</td>
</tr>
<tr>
<td>P6.3/T1DIR</td>
<td>CPVER</td>
<td></td>
</tr>
<tr>
<td>P6.4/SC0</td>
<td>PACT#</td>
<td></td>
</tr>
<tr>
<td>P6.5/SD0</td>
<td>PALE#</td>
<td></td>
</tr>
<tr>
<td>P6.6/SC1</td>
<td>PBUS15:0</td>
<td></td>
</tr>
<tr>
<td>P6.7/SD1</td>
<td>PMODE.3:0</td>
<td></td>
</tr>
<tr>
<td>RXCAN</td>
<td>PROG#</td>
<td></td>
</tr>
<tr>
<td>TXCAN</td>
<td>PVER</td>
<td>Power &amp; Ground</td>
</tr>
<tr>
<td>† Slave port signal</td>
<td></td>
<td>ANGND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VCC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VPP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VREF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VSS, VSS1</td>
</tr>
</tbody>
</table>
Figure A-1. 87C196CB 84-pin PLCC Package
A.2 SIGNAL DESCRIPTIONS

Table A-2 defines the columns used in Table A-3, which describes the signals.
Table A-2. Description of Columns of Table A-3

<table>
<thead>
<tr>
<th>Column Heading</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Lists the signals, arranged alphabetically. Many pins have two functions, so there are more entries in this column than there are pins. Every signal is listed in this column.</td>
</tr>
<tr>
<td>Type</td>
<td>Identifies the pin function listed in the Name column as an input (I), output (O), bidirectional (I/O), power (PWR), or ground (GND). Note that all inputs except RESET# are sampled inputs. RESET# is a level-sensitive input. During powerdown mode, the powerdown circuitry uses EXTINT as a level-sensitive input.</td>
</tr>
<tr>
<td>Description</td>
<td>Briefly describes the function of the pin for the specific signal listed in the Name column. Also lists the alternate function that are multiplexed with the signal (if applicable).</td>
</tr>
</tbody>
</table>

Table A-3. Signal Descriptions

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| A23:16 (100-pin CB) | I/O  | Address Lines 16–23  
These address lines provide address bits 20–23 during the entire external memory cycle, supporting extended addressing of the 16-Mbyte address space. A23:20 are multiplexed with EPORT.7:0. |
| A19:16 (84-pin CB) | I/O  | Address Lines 16–19  
These address lines provide address bits 16–19 during the entire external memory cycle, supporting extended addressing of the 1 Mbyte address space.  
NOTE: Internally, there are 24 address bits; however, only 20 address lines (A19:16 and AD15:0) are implemented as external pins on the 84-pin 87C196CB. The internal address space is 16 Mbytes (000000–FFFFFFH) and the external address space is 1 Mbyte (00000–FFFFFFH). The device resets to FF2080H in internal OTPROM or F2080H in external memory. A19:16 are multiplexed with EPORT.3:0. |
| ACH7:0          | I    | Analog Channels 0–7  
These pins are analog inputs to the A/D converter. These pins may individually be used as analog inputs (ACHx) or digital inputs (P0.x). While it is possible for the pins to function simultaneously as analog and digital inputs, this is not recommended because reading port 0 while a conversion is in process can produce unreliable conversion results.  
The ANGND and VREF pins must be connected for the A/D converter and port 0 to function.  
ACH7:4 are multiplexed with P0.7:4 and PMODE.3:0. ACH3:0 are multiplexed with P0.3:0. |
| AD15:0          | I/O  | Address/Data Lines  
These pins provide a multiplexed address and data bus. During the address phase of the bus cycle, address bits 0–15 are presented on the bus and can be latched using ALE or ADV#. During the data phase, 8- or 16-bit data is transferred. AD7:0 are multiplexed with SLP7:0, P3.7:0, and PBUS.7:0. AD15:8 are multiplexed with P4.7:0 and PBUS.15:8. |
## SIGNAL DESCRIPTIONS

Table A-3. Signal Descriptions (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ADV#  | O    | Address Valid  
This active-low output signal is asserted only during external memory accesses. ADV# indicates that valid address information is available on the system address/data bus. The signal remains low while a valid bus cycle is in progress and is returned high as soon as the bus cycle completes.  
An external latch can use this signal to demultiplex the address from the address/data bus. A decoder can also use this signal to generate chip selects for external memory.  
ADV# is multiplexed with P5.0, SLPALE, and ALE. |
| AINC# | I    | Auto Increment  
During slave programming, this active-low input enables the auto-increment feature. (Auto increment allows reading or writing of sequential OTPROM locations, without requiring address transactions across the PBUS for each read or write.) AINC# is sampled after each location is programmed or dumped.  
If AINC# is asserted, the address is incremented and the next data word is programmed or dumped.  
AINC# is multiplexed with P2.4 and INTOUT#. |
| ALE   | O    | Address Latch Enable  
This active-high output signal is asserted only during external memory cycles. ALE signals the start of an external bus cycle and indicates that valid address information is available on the system address/data bus. ALE differs from ADV# in that it does not remain active during the entire bus cycle.  
An external latch can use this signal to demultiplex address from the address/data bus.  
ALE is multiplexed with P5.0, SLPALE, and ADV#. |
| ANGND | GND  | Analog Ground  
ANGND must be connected for A/D converter and port 0 operation. ANGND and VSS should be nominally at the same potential. |
| BHE#  | O    | Byte High Enable†  
During 16-bit bus cycles, this active-low output signal is asserted for word reads and writes and high-byte reads and writes to external memory. BHE# indicates that valid data is being transferred over the upper half of the system data bus. Use BHE#, in conjunction with AD0, to determine which memory byte is being transferred over the system bus:  

<table>
<thead>
<tr>
<th>BHE#</th>
<th>AD0</th>
<th>Byte(s) Accessed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>both bytes</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>high byte only</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>low byte only</td>
</tr>
</tbody>
</table>

BHE# is multiplexed with P5.5 and WRH#. |

† The chip configuration register 0 (CCR0) determines whether this pin functions as BHE# or WRH#: CCR0.2 = 1 selects BHE#; CCR0.2 = 0 selects WRH#. |
**Table A-3. Signal Descriptions (Continued)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREQ#</td>
<td>O</td>
<td>Bus Request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This active-low output signal is asserted during a hold cycle when the bus controller has a pending external memory cycle.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The device can assert BREQ# at the same time as or after it asserts HLDA#.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Once it is asserted, BREQ# remains asserted until HOLD# is removed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>You must enable the bus-hold protocol before using this signal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BREQ# is multiplexed with P2.3.</td>
</tr>
<tr>
<td>BUSWIDTH</td>
<td>I</td>
<td>Bus Width</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The chip configuration register bits, CCR0.1 and CCR1.2, along with the BUSWIDTH pin, control the data bus width. When both CCR bits are set, the BUSWIDTH signal selects the external data bus width. When only one CCR bit is set, the bus width is fixed at either 16 or 8 bits, and the BUSWIDTH signal has no effect.</td>
</tr>
<tr>
<td>CCR0.1</td>
<td></td>
<td>fixed 8-bit data bus</td>
</tr>
<tr>
<td>CCR1.2</td>
<td></td>
<td>fixed 16-bit data bus</td>
</tr>
<tr>
<td>BUSWIDTH</td>
<td></td>
<td>8-bit data bus</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BUSWIDTH is multiplexed with P5.7.</td>
</tr>
<tr>
<td>CLKOUT</td>
<td>O</td>
<td>Clock Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Output of the internal clock generator. The CLKOUT frequency is ½ the operating frequency (f). CLKOUT has a 50% duty cycle.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLKOUT is multiplexed with P2.7 and PACT#.</td>
</tr>
<tr>
<td>COMP1:0</td>
<td>O</td>
<td>Event Processor Array (EPA) Compare Pins</td>
</tr>
<tr>
<td></td>
<td></td>
<td>These signals are the output of the EPA compare-only channels. These pins are multiplexed with other signals and may be configured as standard I/O.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COMP1:0 are multiplexed as follows: COMP0/P6.0/EPA8 and COMP1/P6.1/EPA9.</td>
</tr>
<tr>
<td>CPVER</td>
<td>O</td>
<td>Cumulative Program Verification</td>
</tr>
<tr>
<td></td>
<td></td>
<td>During slave programming, a high signal indicates that all locations programmed correctly, while a low signal indicates that an error occurred during one of the programming operations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPVER is multiplexed with P2.6 and HLDA#.</td>
</tr>
</tbody>
</table>
### SIGNAL DESCRIPTIONS

#### EA# I External Access
This input determines whether memory accesses to special-purpose and program memory partitions (FF2000–FF9FFFH) are directed to internal or external memory. These accesses are directed to internal memory if EA# is held high and to external memory if EA# is held low. For an access to any other memory location, the value of EA# is irrelevant.

EA# also controls entry into programming mode. If EA# is at $V_{pp}$ voltage (typically +12.5 V) on the rising edge of RESET#, the device enters programming mode.

**NOTE:** Systems with EA# tied inactive have idle time between external bus cycles. When the address/data bus is idle, you can use ports 3 and 4 for I/O. Systems with EA# tied active cannot use ports 3 and 4 as standard I/O; when EA# is active, these ports will function only as the address/data bus.

EA# is sampled and latched only on the rising edge of RESET#. Changing the level of EA# after reset has no effect.

On devices with no internal nonvolatile memory, always connect EA# to $V_{ss}$.

#### EPA9:0 I/O Event Processor Array (EPA) Input/Output pins
These are the high-speed input/output pins for the EPA capture/compare channels. For high-speed PWM applications, the outputs of two EPA channels (either EPA0 and EPA1 or EPA2 and EPA3) can be remapped to produce a PWM waveform on a shared output pin.

EPA9:0 are multiplexed as follows: EPA0/P1.0/T2CLK, EPA1/P1.1, EPA2/P1.2/T2DIR, EPA3/P1.3, EPA4/P1.4, EPA5/P1.5, EPA6/P1.6, EPA7/P1.7, EPA8/P6.0/COMP0, and EPA9/P6.1/COMP1.

#### EPORT.7:0 (100-pin CB) I/O Extended Addressing Port
This is a 4-bit, bidirectional, memory-mapped I/O port.

EPORT.7:0 are multiplexed with A23:16.

#### EPORT.3:0 (84-pin CB) I/O Extended Addressing Port
This is a 4-bit, bidirectional, memory-mapped I/O port.

EPORT.3:0 are multiplexed with A19:16.

#### EXTINT I External Interrupt
In normal operating mode, a rising edge on EXTINT sets the EXTINT interrupt pending bit. EXTINT is sampled during phase 2 (CLKOUT high). The minimum high time is one state time.

In powerdown mode, asserting the EXTINT signal for at least 50 ns causes the device to resume normal operation. The interrupt need not be enabled, but the pin must be configured as a special-function input. If the EXTINT interrupt is enabled, the CPU executes the interrupt service routine. Otherwise, the CPU executes the instruction that immediately follows the command that invoked the power-saving mode.

In idle mode, asserting any enabled interrupt causes the device to resume normal operation.

EXTINT is multiplexed with P2.2 and PROG#.

---

### Table A-3. Signal Descriptions (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA#</td>
<td>I</td>
<td>External Access</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This input determines whether memory accesses to special-purpose and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>program memory partitions (FF2000–FF9FFFH) are directed to internal or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>external memory. These accesses are directed to internal memory if EA#</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is held high and to external memory if EA# is held low. For an access to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>any other memory location, the value of EA# is irrelevant.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EA# also controls entry into programming mode. If EA# is at $V_{pp}$ voltage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(typically +12.5 V) on the rising edge of RESET#, the device enters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>programming mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>NOTE:</strong> Systems with EA# tied inactive have idle time between external</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bus cycles. When the address/data bus is idle, you can use ports 3 and 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for I/O. Systems with EA# tied active cannot use ports 3 and 4 as</td>
</tr>
<tr>
<td></td>
<td></td>
<td>standard I/O; when EA# is active, these ports will function only as the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>address/data bus.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EA# is sampled and latched only on the rising edge of RESET#. Changing the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>level of EA# after reset has no effect.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>On devices with no internal nonvolatile memory, always connect EA# to $V_{ss}$</td>
</tr>
<tr>
<td>EPA9:0</td>
<td>I/O</td>
<td>Event Processor Array (EPA) Input/Output pins</td>
</tr>
<tr>
<td></td>
<td></td>
<td>These are the high-speed input/output pins for the EPA capture/compare</td>
</tr>
<tr>
<td></td>
<td></td>
<td>channels. For high-speed PWM applications, the outputs of two EPA channels</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(either EPA0 and EPA1 or EPA2 and EPA3) can be remapped to produce a PWM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>waveform on a shared output pin.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EPA9:0 are multiplexed as follows: EPA0/P1.0/T2CLK, EPA1/P1.1, EPA2/P1.2/T2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DIR, EPA3/P1.3, EPA4/P1.4, EPA5/P1.5, EPA6/P1.6, EPA7/P1.7, EPA8/P6.0/COMP0,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and EPA9/P6.1/COMP1.</td>
</tr>
<tr>
<td>EPORT.7:0</td>
<td>I/O</td>
<td>Extended Addressing Port</td>
</tr>
<tr>
<td>(100-pin CB)</td>
<td></td>
<td>This is a 4-bit, bidirectional, memory-mapped I/O port.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EPORT.7:0 are multiplexed with A23:16.</td>
</tr>
<tr>
<td>EPORT.3:0</td>
<td>I/O</td>
<td>Extended Addressing Port</td>
</tr>
<tr>
<td>(84-pin CB)</td>
<td></td>
<td>This is a 4-bit, bidirectional, memory-mapped I/O port.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EPORT.3:0 are multiplexed with A19:16.</td>
</tr>
<tr>
<td>EXTINT</td>
<td>I</td>
<td>External Interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In normal operating mode, a rising edge on EXTINT sets the EXTINT interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pending bit. EXTINT is sampled during phase 2 (CLKOUT high). The minimum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>high time is one state time.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In powerdown mode, asserting the EXTINT signal for at least 50 ns causes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the device to resume normal operation. The interrupt need not be enabled,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>but the pin must be configured as a special-function input. If the EXTINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interrupt is enabled, the CPU executes the interrupt service routine.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Otherwise, the CPU executes the instruction that immediately follows the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>command that invoked the power-saving mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In idle mode, asserting any enabled interrupt causes the device to resume</td>
</tr>
<tr>
<td></td>
<td></td>
<td>normal operation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXTINT is multiplexed with P2.2 and PROG#.</td>
</tr>
</tbody>
</table>
**Table A-3. Signal Descriptions (Continued)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLDA#</td>
<td>O</td>
<td>Bus Hold Acknowledge</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This active-low output indicates that the CPU has released the bus as the result of an external device asserting HOLD#.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HLDA# is multiplexed with P2.6 and CPVER.</td>
</tr>
<tr>
<td>HOLD#</td>
<td>I</td>
<td>Bus Hold Request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An external device uses this active-low input signal to request control of the bus. This pin functions as HOLD# only if the pin is configured for its special function and the bus-hold protocol is enabled. Setting bit 7 of the window selection register (WSR) enables the bus-hold protocol.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HOLD# is multiplexed with P2.5.</td>
</tr>
<tr>
<td>INST</td>
<td>O</td>
<td>Instruction Fetch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This active-high output signal is valid only during external memory bus cycles. When high, INST indicates that an instruction is being fetched from external memory. The signal remains high during the entire bus cycle of an external instruction fetch. INST is low for data accesses, including interrupt vector fetches and chip configuration byte reads. INST is low during internal memory fetches.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INST is multiplexed with P5.1 and SLPCS#.</td>
</tr>
<tr>
<td>INTOUT#</td>
<td>O</td>
<td>Interrupt Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This active-low output indicates that a pending interrupt requires use of the external bus. How quickly the microcontroller asserts INTOUT# depends upon the status of HOLD# and HLDA# and whether the microcontroller is executing from internal or external program memory. If the microcontroller is executing from internal memory and receives an interrupt request while in hold, it asserts INTOUT# immediately. However, if the microcontroller is executing code from external memory and receives an interrupt request while in hold, it asserts BREQ# and waits until the external device deasserts HOLD# to assert INTOUT#. If the microcontroller is executing code from external memory and receives an interrupt request as it is going into hold (between the time that an external device asserts HOLD# and the time that the microcontroller responds with HLDA#), the microcontroller asserts both HLDA# and INTOUT# and keeps them asserted until the external device deasserts HOLD#.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTOUT# is multiplexed with P2.4 and AINC#.</td>
</tr>
<tr>
<td>NMI</td>
<td>I</td>
<td>Nonmaskable Interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In normal operating mode, a rising edge on NMI generates a nonmaskable interrupt. NMI has the highest priority of all prioritized interrupts. Assert NMI for greater than one state time to guarantee that it is recognized.</td>
</tr>
<tr>
<td>ONCE#</td>
<td>I</td>
<td>On-circuit Emulation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Holding ONCE# low during the rising edge of RESET# places the device into on-circuit emulation (ONCE) mode. This mode puts all pins into a high-impedance state, thereby isolating the device from other components in the system. The value of ONCE# is latched when the RESET# pin goes inactive. While the device is in ONCE mode, you can debug the system using a clip-on emulator. To exit ONCE mode, reset the device by pulling the RESET# signal low. To prevent inadvertent entry into ONCE mode, either configure this pin as an output or hold it high during reset and ensure that your system meets the ( V_{\text{IN}} ) specification (see datasheet).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ONCE# is multiplexed with P2.6.</td>
</tr>
</tbody>
</table>
### SIGNAL DESCRIPTIONS

**Table A-3. Signal Descriptions (Continued)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0.7:0</td>
<td>I</td>
<td>Port 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This is a high-impedance, input-only port. Port 0 pins should <strong>not</strong> be left floating. These pins may individually be used as analog inputs (ACHx) or digital inputs (P0.x). While it is possible for the pins to function simultaneously as analog and digital inputs, this is not recommended because reading port 0 while a conversion is in process can produce unreliable conversion results. ANGND and VREF must be connected for port 0 to function. P0.7:4 are multiplexed with ACH7:4 and PMODE.3:0. P0.3:0 are multiplexed with ACH3:0.</td>
</tr>
<tr>
<td>P1.7:0</td>
<td>I/O</td>
<td>Port 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This is a standard, bidirectional port that is multiplexed with individually selectable special-function signals. Port 1 is multiplexed as follows: P1.0/EPA0, P1.1/EPA1, P1.2/EPA2, P1.3/EPA3, P1.4/T1CLK, P1.5/T1DIR, P1.6/T2CLK, and P1.7/T2DIR.</td>
</tr>
<tr>
<td>P2.7:0</td>
<td>I/O</td>
<td>Port 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This is a standard bidirectional port that is multiplexed with individually selectable special-function signals. P2.6 is multiplexed with the ONCE# function. If this pin is held low during reset, the device will enter ONCE mode, so <strong>exercise caution</strong> if you use this pin for input. If you choose to configure this pin as an input, always hold it high during reset and ensure that your system meets the VIH specification (see datasheet) to prevent inadvertent entry into a test mode. Port 2 is multiplexed as follows: P2.0/TXD/PVER, P2.1/RXD/PALE#, P2.2/EXTINT/PROG#, P2.3/BREQ#, P2.4/INTOUT#/AINC#, P2.5/HOLD#, P2.6/HLDA#/ONCE#/CPVER, P2.7/CLKOUT/PACT#.</td>
</tr>
<tr>
<td>P3.7:0</td>
<td>I/O</td>
<td>Port 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This is an 8-bit, bidirectional, memory-mapped I/O port with open-drain outputs. The pins are shared with the multiplexed address/data bus, which has complementary drivers. P3.7:0 are multiplexed with AD7:0, SLP7:0, and PBUS.7:0.</td>
</tr>
<tr>
<td>P4.7:0</td>
<td>I/O</td>
<td>Port 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This is an 8-bit, bidirectional, memory-mapped I/O port with open-drain outputs. The pins are shared with the multiplexed address/data bus, which has complementary drivers. P4.7:0 are multiplexed with AD15:8 and PBUS15:8.</td>
</tr>
<tr>
<td>P5.7:0</td>
<td>I/O</td>
<td>Port 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This is an 8-bit, bidirectional, memory-mapped I/O port. P5.4 is multiplexed with a special test-mode-entry function. If this pin is held low during reset, the device will enter a reserved test mode, so <strong>exercise caution</strong> if you use this pin for input. If you choose to configure this pin as an input, always hold it high during reset and ensure that your system meets the VIL specification (see datasheet) to prevent inadvertent entry into a test mode. Port 5 is multiplexed as follows: P5.0/ALE/ADV#/SLPALE, P5.1/INST/SLPCS#, P5.2/WR#/WRL#/SLPWR#, P5.3/RD#/SLPRD#, /SLPINT, P5.5/BHE#/WRH#, P5.6/READY, and P5.7/BUSWIDTH.</td>
</tr>
</tbody>
</table>
This is a standard 8-bit bidirectional port. Port 6 is multiplexed as follows: P6.0/EPA8/COMP0, P6.1/EPA9/COMP1, P6.2/T1CLK, P6.3/T1DIR, P6.4/SC0, P6.5/SD0, P6.6/SC1, and P6.7/SD1.

During auto programming or ROM-dump, a low signal indicates that programming or dumping is in progress, while a high signal indicates that the operation is complete.

During slave programming, a falling edge causes the device to read a command and address from the PBUS. PALE# is multiplexed with P2.1 and RXD.

During slave programming, ports 3 and 4 serve as a bidirectional port with open-drain outputs to pass commands, addresses, and data to or from the device. Slave programming requires external pull-up resistors. During auto programming and ROM-dump, ports 3 and 4 serve as a regular system bus to access external memory. P4.6 and P4.7 are left unconnected; P1.1 and P1.2 serve as the upper address lines.

Slave programming:
PBUS.7:0 are multiplexed with AD7:0, SLP7:0, and P3.7:0.
PBUS.15:8 are multiplexed with AD15:8 and P4.7:0.

Auto programming:
PBUS.7:0 are multiplexed with AD7:0, SLP7:0, and P3.7:0.
PBUS.13:8 are multiplexed with AD13:8 and P4.5:0; PBUS15:14 are multiplexed with P1.2:1.

The value on the PMODE pins determines the programming mode:
0H = serial port programming
5H = slave programming
6H = ROM-dump
CH = auto programming
PMODE is sampled after a device reset and must be static while the part is operating.
PMODE.3:0 are multiplexed with P0.7:4 and ACH7:4.

This input pin enables and disables the on-chip clock multiplier feature.
0 = standard mode; internal frequency is equal to $F_{XTAL1}$.
1 = quadruple mode; internal frequency is equal to $4F_{XTAL1}$.
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROG#</td>
<td>I</td>
<td>Programming Start. During programming, a falling edge latches data on the PBUS and begins programming, while a rising edge ends programming. The current location is programmed with the same data as long as PROG# remains asserted, so the data on the PBUS must remain stable while PROG# is active. During a word dump, a falling edge causes the contents of an OTPROM location to be output on the PBUS, while a rising edge ends the data transfer. PROG# is multiplexed with P2.2 and EXTINT.</td>
</tr>
<tr>
<td>PVER</td>
<td>O</td>
<td>Program Verification. During slave or auto programming, PVER is updated after each programming pulse. A high output signal indicates successful programming of a location, while a low signal indicates a detected error. PVER is multiplexed with P2.0 and TXD.</td>
</tr>
<tr>
<td>RD#</td>
<td>O</td>
<td>Read. Read-signal output to external memory. RD# is asserted only during external memory reads. RD# is multiplexed with P5.3 and SLPRD#.</td>
</tr>
<tr>
<td>READY</td>
<td>I</td>
<td>Ready Input. This active-high input signal is used to lengthen external memory cycles for slow memory by generating wait states in addition to the wait states that are generated internally. When READY is high, CPU operation continues in a normal manner with wait states inserted as programmed in the chip configuration registers. READY is ignored for all internal memory accesses. READY is multiplexed with P5.6.</td>
</tr>
<tr>
<td>RESET#</td>
<td>I/O</td>
<td>Reset. A level-sensitive reset input to and open-drain system reset output from the microcontroller. Either a falling edge on RESET# or an internal reset turns on a pull-down transistor connected to the RESET# pin for 16 state times. In the powerdown and idle modes, asserting RESET# causes the chip to reset and return to normal operating mode. After a device reset, the first instruction fetch is from FF2080H.</td>
</tr>
<tr>
<td>RXCAN</td>
<td>I</td>
<td>Receive. This signal carries messages from other nodes on the CAN bus to the integrated CAN controller.</td>
</tr>
<tr>
<td>RXD</td>
<td>I/O</td>
<td>Receive Serial Data. In modes 1, 2, and 3, RXD receives serial port input data. In mode 0, it functions as either an input or an open-drain output for data. RXD is multiplexed with P2.1 and PALE#.</td>
</tr>
<tr>
<td>SC1:0</td>
<td>I/O</td>
<td>Clock Pins for SSIO0 and 1. For handshaking mode, configure SC1:0 as open-drain outputs. This pin carries a signal only during receptions and transmissions. When the SSIO port is idle, the pin remains either high (with handshaking) or low (without handshaking). SC0 is multiplexed with P6.4, and SC1 is multiplexed with P6.6.</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SD1:0</td>
<td>I/O</td>
<td>Data Pins for SSIO0 and 1&lt;br&gt;SD0 is multiplexed with P6.5, and SD1 is multiplexed with P6.7.</td>
</tr>
<tr>
<td>SLP7:0</td>
<td>I/O</td>
<td>Slave Port Address/Data bus&lt;br&gt;Slave port address/data bus in multiplexed mode and slave port data bus in demultiplexed mode. In multiplexed mode, SLP1 is the source of the internal control signal, SLP_ADDR.&lt;br&gt;SLP7:0 are multiplexed with AD7:0, P3.7:0, and PBUS.7:0.</td>
</tr>
<tr>
<td>SLPALE</td>
<td>I</td>
<td>Slave Port Address Latch Enable&lt;br&gt;Functions as either a latch enable input to latch the value on SLP1 (with a multiplexed address/data bus) or as the source of the internal control signal, SLP_ADDR (with a demultiplexed address/data bus).&lt;br&gt;SLPALE is multiplexed with P5.0, ADV#, and ALE.</td>
</tr>
<tr>
<td>SLPCS#</td>
<td>I</td>
<td>Slave Port Chip Select&lt;br&gt;SLPCS# must be held low to enable slave port operation.&lt;br&gt;SLPCS# is multiplexed with P5.1 and INST.</td>
</tr>
<tr>
<td>SLPINT</td>
<td>O</td>
<td>Slave Port Interrupt&lt;br&gt;This active-high slave port output signal can be used to interrupt the master processor.&lt;br&gt;SLPINT is multiplexed with P5.4 and a special test-mode-entry pin. See P5.7:0 for special considerations.</td>
</tr>
<tr>
<td>SLPRD#</td>
<td>I</td>
<td>Slave Port Read Control Input&lt;br&gt;This active-low signal is an input to the slave. Data from the P3_REG or SLP_STAT register is valid after the falling edge of SLPRD#.&lt;br&gt;SLPRD# is multiplexed with P5.3 and RD#.</td>
</tr>
<tr>
<td>SLPWR#</td>
<td>I</td>
<td>Slave Port Write Control Input&lt;br&gt;This active-low signal is an input to the slave. The rising edge of SLPWR# latches data on port 3 into the P3_PIN or SLP_CMD register.&lt;br&gt;SLPWR# is multiplexed with P5.2, WR#, and WRL#.</td>
</tr>
<tr>
<td>T1CLK</td>
<td>I</td>
<td>Timer 1 External Clock&lt;br&gt;External clock for timer 1. Timer 1 increments (or decrements) on both rising and falling edges of T1CLK. Also used in conjunction with T1DIR for quadrature counting mode.&lt;br&gt;External clock for the serial I/O baud-rate generator input (program selectable).&lt;br&gt;T1CLK is multiplexed with P6.2.</td>
</tr>
<tr>
<td>T2CLK</td>
<td>I</td>
<td>Timer 2 External Clock&lt;br&gt;External clock for timer 2. Timer 2 increments (or decrements) on both rising and falling edges of T2CLK. Also used in conjunction with T2DIR for quadrature counting mode.&lt;br&gt;T2CLK is multiplexed with P1.0 and EPA0.</td>
</tr>
<tr>
<td>T1DIR</td>
<td>I</td>
<td>Timer 1 External Direction&lt;br&gt;External direction (up/down) for timer 1. Timer 1 increments when T1DIR is high and decrements when it is low. Also used in conjunction with T1CLK for quadrature counting mode.&lt;br&gt;T1DIR is multiplexed with P6.3.</td>
</tr>
</tbody>
</table>
### SIGNAL DESCRIPTIONS

#### Table A-3. Signal Descriptions (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| T2DIR     | I    | Timer 2 External Direction  
External direction (up/down) for timer 2. Timer 2 increments when T2DIR is high and decrements when it is low. Also used in conjunction with T2CLK for quadrature counting mode.  
T2DIR is multiplexed with P1.2 and EPA2. |
| TXCAN     | O    | Transmit  
This signal carries messages from the integrated CAN controller to other nodes on the CAN bus. |
| TXD       | O    | Transmit Serial Data  
In serial I/O modes 1, 2, and 3, TXD transmits serial port output data. In mode 0, it is the serial clock output.  
TXD is multiplexed with P2.0 and PVER. |
| VCC       | PWR  | Digital Supply Voltage  
Connect each VCC pin to the digital supply voltage. |
| VPP       | PWR  | Programming Voltage  
During programming, the VPP pin is typically at +12.5 V (VPP voltage).  
Exceeding the maximum VPP voltage specification can damage the device.  
VPP also causes the device to exit powerdown mode when it is driven low for at least 50 ns. Use this method to exit powerdown only when using an external clock source because it enables the internal phase clocks, but not the internal oscillator. |
| VREF      | PWR  | Reference Voltage for the A/D Converter  
This pin also supplies operating voltage to both the analog portion of the A/D converter and the logic used to read port 0. |
| VSS, VSS¹ | GND  | Digital Circuit Ground (Core Ground, Port Ground)  
Connect each VSS and VSS¹ pin to ground through the lowest possible impedance path. VSS pins are connected to the core ground region of the microcontroller, while VSS¹ pins are connected to the port ground region. (ANGND is connected to the analog ground region.) Separating the ground regions provides noise isolation. |
| WR#       | O    | Write†  
This active-low output indicates that an external write is occurring. This signal is asserted only during external memory writes.  
WR# is multiplexed with P5.2, SLPWR#, and WRL#.  
† The chip configuration register 0 (CCR0) determines whether this pin functions as WR# or WRL#. CCR0.2 = 1 selects WR#; CCR0.2 = 0 selects WRL#. |
| WRH#      | O    | Write High†  
During 16-bit bus cycles, this active-low output signal is asserted for high-byte writes and word writes to external memory. During 8-bit bus cycles, WRH# is asserted for all write operations.  
WRH# is multiplexed with P5.5 and BHE#.  
† The chip configuration register 0 (CCR0) determines whether this pin functions as BHE# or WRH#. CCR0.2 = 1 selects BHE#; CCR0.2 = 0 selects WRH#. |
A.3 DEFAULT CONDITIONS

Table A-5 lists the default functions of the I/O and control pins of the microcontroller with their values during various operating conditions. Table A-4 defines the symbols used to represent the pin status. Refer to the DC Characteristics table in the datasheet for actual specifications for $V_{OL}$, $V_{IL}$, $V_{OH}$, and $V_{IH}$.

### Table A-4. Definition of Status Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Voltage less than or equal to $V_{OL} \cdot V_{IL}$</td>
<td>MD0</td>
<td>Medium pull-down</td>
</tr>
<tr>
<td>1</td>
<td>Voltage greater than or equal to $V_{OH} \cdot V_{IH}$</td>
<td>MD1</td>
<td>Medium pull-up</td>
</tr>
<tr>
<td>HiZ</td>
<td>High impedance</td>
<td>WK0</td>
<td>Weak pull-down</td>
</tr>
<tr>
<td>LoZ0</td>
<td>Low impedance; strongly driven low</td>
<td>WK1</td>
<td>Weak pull-up</td>
</tr>
<tr>
<td>LoZ1</td>
<td>Low impedance; strongly driven high</td>
<td>ODIO</td>
<td>Open-drain I/O</td>
</tr>
</tbody>
</table>

### Table A-5. 87C196CB Pin Status

<table>
<thead>
<tr>
<th>Port Pins</th>
<th>Multiplexed With</th>
<th>Status During Reset</th>
<th>Status During Idle</th>
<th>Status During Powerdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0.7:4</td>
<td>ACH7:4</td>
<td>HiZ</td>
<td>HiZ</td>
<td>HiZ</td>
</tr>
<tr>
<td>P1.7:0</td>
<td>EPA7:0</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>P2.0</td>
<td>TXD</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>P2.1</td>
<td>RXD</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>Port Pins</td>
<td>Multiplexed With</td>
<td>Status During Reset</td>
<td>Status During Idle</td>
<td>Status During Powerdown</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------</td>
<td>---------------------</td>
<td>--------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>P2.2</td>
<td>EXTINT</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>P2.3</td>
<td>BREQ#</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>P2.4</td>
<td>INTOUT#</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>P2.5</td>
<td>HOLD#</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>P2.6</td>
<td>HLDA#</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>P2.7</td>
<td>CLKOUT</td>
<td>CLKOUT active, LoZ0/1</td>
<td>(Note 3)</td>
<td>(Note 4)</td>
</tr>
<tr>
<td>P3.7:0</td>
<td>AD7:0</td>
<td>WK1</td>
<td>(Note 6)</td>
<td>(Note 6)</td>
</tr>
<tr>
<td>P4.7:0</td>
<td>AD15:8</td>
<td>WK1</td>
<td>(Note 6)</td>
<td>(Note 6)</td>
</tr>
<tr>
<td>EPORT.3:0</td>
<td>AD19:17</td>
<td>WK1</td>
<td>(Note 7)</td>
<td>(Note 7)</td>
</tr>
<tr>
<td>P5.0</td>
<td>ALE</td>
<td>WK1</td>
<td>(Note 1)</td>
<td>(Note 1)</td>
</tr>
<tr>
<td>P5.1</td>
<td>INST</td>
<td>WK0</td>
<td>(Note 1)</td>
<td>(Note 1)</td>
</tr>
<tr>
<td>P5.2</td>
<td>WR#/WRL#</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>P5.3</td>
<td>RD#</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>P5.4</td>
<td>SLPINT</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>P5.5</td>
<td>BHE#/WRH#</td>
<td>WK1</td>
<td>(Note 1)</td>
<td>(Note 1)</td>
</tr>
<tr>
<td>P5.6</td>
<td>READY</td>
<td>WK1</td>
<td>(Note 2)</td>
<td>(Note 2)</td>
</tr>
<tr>
<td>P5.7</td>
<td>BUSWIDTH</td>
<td>WK1</td>
<td>(Note 2)</td>
<td>(Note 2)</td>
</tr>
<tr>
<td>P6.1:0</td>
<td>EPA9:8</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>P6.2</td>
<td>T1CLK</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>P6.3</td>
<td>T1DIR</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>P6.4</td>
<td>SC0</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>P6.5</td>
<td>SD0</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>P6.6</td>
<td>SC1</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>P6.7</td>
<td>SD1</td>
<td>WK1</td>
<td>(Note 3)</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>EA#</td>
<td></td>
<td>HiZ</td>
<td>HiZ</td>
<td>HiZ</td>
</tr>
<tr>
<td>NMI</td>
<td></td>
<td>HiZ</td>
<td>HiZ</td>
<td>HiZ</td>
</tr>
<tr>
<td>RXCAN</td>
<td></td>
<td>WK1</td>
<td>WK1</td>
<td>WK1</td>
</tr>
<tr>
<td>TXCAN</td>
<td></td>
<td>LoZ1</td>
<td>LoZ1</td>
<td>LoZ1</td>
</tr>
<tr>
<td>Vpp</td>
<td></td>
<td>HiZ</td>
<td>LoZ1</td>
<td>LoZ1</td>
</tr>
<tr>
<td>XTAL1</td>
<td></td>
<td>Osc input, HiZ</td>
<td>Osc input, HiZ</td>
<td>Osc input, HiZ</td>
</tr>
<tr>
<td>XTAL2</td>
<td></td>
<td>Osc output, LoZ0/1</td>
<td>Osc output, LoZ0/1</td>
<td>(Note 5)</td>
</tr>
</tbody>
</table>

**NOTES:**

1. If P5_MODE.y = 0, port is as programmed. If P5_MODE.y = 1 and HLDA# = 1, P5.0 and P5.1 are LoZ0; P5.5 is LoZ1.
2. If P5_MODE.y = 1 and HLDA# = 0, port is HiZ.
3. If P5_MODE.y = 0, port is as programmed. If P5_MODE.y = 1, port is HiZ.
4. If P5_MODE.y = 1, pin is as specified by Px_DIR and the associated peripheral.
5. If P2_MODE.7 = 0, pin is as programmed. If P2_MODE.7 = 1, pin is LoZ0.
6. If XTAL1 = 0, pin is LoZ1. If XTAL1 = 1, pin is LoZ0.
7. Pins configured as address are high-impedance; pins configured as I/O remain unchanged.
Glossary
This glossary defines acronyms, abbreviations, and terms that have special meaning in this manual. (Chapter 1 discusses notational conventions and general terminology.)

**absolute error**  
The maximum difference between corresponding actual and ideal code transitions. Absolute error accounts for all deviations of an actual A/D converter from an ideal converter.

**accumulator**  
A register or storage location that forms the result of an arithmetic or logical operation.

**actual characteristic**  
A graph of output code versus input voltage of an actual A/D converter. An actual characteristic may vary with temperature, supply voltage, and frequency conditions.

**A/D converter**  
Analog-to-digital converter.

**ALU**  
Arithmetic-logic unit. The part of the RALU that processes arithmetic and logical operations.

**assert**  
The act of making a signal active (enabled). The polarity (high or low) is defined by the signal name. Active-low signals are designated by a pound symbol (#) suffix; active-high signals have no suffix. To assert RD# is to drive it low; to assert ALE is to drive it high.

**attenuation**  
A decrease in amplitude; voltage decay.

**bit**  
A binary digit.

**BIT**  
A single-bit operand that can take on the Boolean values, “true” and “false.”

**break-before-make**  
The property of a multiplexer which guarantees that a previously selected channel is deselected before a new channel is selected. (That is, break-before-make ensures that the A/D converter will not short inputs together.)

**byte**  
Any 8-bit unit of data.

**BYTE**  
An unsigned, 8-bit variable with values from 0 through $2^8 - 1$. 
CAN  Controller area network. The 87C196CB’s integrated networking peripheral, similar to Intel’s standalone 82527 CAN serial communications controller, that supports CAN specification 2.0.

CCBs  Chip configuration bytes. The chip configuration registers (CCRs) are loaded with the contents of the CCBs after a device reset, unless the device is entering programming modes, in which case the PCCBs are used.

CCRs  Chip configuration registers. Registers that specify the environment in which the device will be operating. The chip configuration registers are loaded with the contents of the CCBs after a device reset unless the device is entering programming modes, in which case the PCCBs are used.

channel-to-channel matching error  The difference between corresponding code transitions of actual characteristics taken from different A/D converter channels under the same temperature, voltage, and frequency conditions. This error is caused by differences in DC input leakage and on-channel resistance from one multiplexer channel to another.

characteristic  A graph of output code versus input voltage; the transfer function of an A/D converter.

clear  The “0” value of a bit or the act of giving it a “0” value. See also set.

code  1) A set of instructions that perform a specific function; a program.

2) The digital value output by the A/D converter.

code center  The voltage corresponding to the midpoint between two adjacent code transitions on the A/D converter.

code transition  The point at which the A/D converter’s output code changes from “Q” to “Q+1.” The input voltage corresponding to a code transition is defined as the voltage that is equally likely to produce either of two adjacent codes.
code width
The voltage change corresponding to the difference between two adjacent code transitions. Code width deviations cause differential nonlinearity and nonlinearity errors.

crosstalk
See off-isolation.

DC input leakage
Leakage current from an analog input pin to ground.

deassert
The act of making a signal inactive (disabled). The polarity (high or low) is defined by the signal name. Active-low signals are designated by a pound symbol (#) suffix; active-high signals have no suffix. To deassert RD# is to drive it high; to deassert ALE is to drive it low.

differential nonlinearity
The difference between the actual code width and the ideal one-LSB code width of the terminal-based characteristic of an A/D converter. It provides a measure of how much the input voltage may have changed in order to produce a one-count change in the conversion result. Differential nonlinearity is a measure of local code-width error; nonlinearity is a measure of overall code-transition error.

doping
The process of introducing a periodic table Group III or Group V element into a Group IV element (e.g., silicon). A Group III impurity (e.g., indium or gallium) results in a p-type material. A Group V impurity (e.g., arsenic or antimony) results in an n-type material.

double-word
Any 32-bit unit of data.

DOUBLE-WORD
An unsigned, 32-bit variable with values from 0 through $2^{32} - 1$.

EPA
Event processor array. An integrated peripheral that provides high-speed input/output capability.

EPROM
Erasable, programmable read-only-memory.

ESD
Electrostatic discharge.

feedthrough
The attenuation from an input voltage on the selected channel to the A/D output after the sample window closes. The ability of the A/D converter to reject an input on its selected channel after the sample window closes.
FET

Field-effect transistor.

dominance generator

The 8XC196MD peripheral that generates outputs with a fixed 50% duty cycle and a programmable frequency. The frequency generator can be used for infrared transmission.

full-scale error

The difference between the ideal and actual input voltage corresponding to the final (full-scale) code transition of an A/D converter.

hold latency

The time it takes the microcontroller to assert HLDA# after an external device asserts HOLD#.

ideal characteristic

The characteristic of an ideal A/D converter. An ideal characteristic is unique: its first code transition occurs when the input voltage is 0.5 LSB, its full-scale (final) code transition occurs when the input voltage is 1.5 LSB less than the full-scale reference, and its code widths are all exactly 1.0 LSB. These properties result in a conversion without zero-offset, full-scale, or linearity errors. Quantizing error is the only error seen in an ideal A/D converter.

input leakage

Current leakage from an input pin to power or ground.

input series resistance

The effective series resistance from an analog input pin to the sample capacitor of an A/D converter.

integer

Any member of the set consisting of the positive and negative whole numbers and zero.

INTEGER

A 16-bit, signed variable with values from $-2^{15}$ through $+2^{15}-1$.

interrupt controller

The module responsible for handling interrupts that are to be serviced by interrupt service routines that you provide. Also called the programmable interrupt controller (PIC).

interrupt latency

The total delay between the time that an interrupt is generated (not acknowledged) and the time that the device begins executing the interrupt service routine or PTS routine.

interrupt service routine

A software routine that you provide to service a standard interrupt. See also PTS routine.

interrupt vector

A location in special-purpose memory that holds the starting address of an interrupt service routine.
ISR
See interrupt service routine.

linearity errors
See differential nonlinearity and nonlinearity.

LONG-INTEGER
A 32-bit, signed variable with values from $-2^{31}$ through $+2^{31}-1$.

LSB
1) Least-significant bit of a byte or least-significant byte of a word.
2) In an A/D converter, the reference voltage divided by $2^n$, where $n$ is the number of bits to be converted. For a 10-bit converter with a reference voltage of 5.12 volts, one LSB is equal to 5.0 millivolts ($5.12 \div 2^{10}$).

maskable interrupts
All interrupts except unimplemented opcode, software trap, and NMI. Maskable interrupts can be disabled (masked) by the individual mask bits in the interrupt mask registers, and their servicing can be disabled by the global interrupt enable bit. Each maskable interrupt can be assigned to the PTS for processing.

monotonic
The property of successive approximation converters which guarantees that increasing input voltages produce adjacent codes of increasing value, and that decreasing input voltages produce adjacent codes of decreasing value. (In other words, a converter is monotonic if every code change represents an input voltage change in the same direction.) Large differential nonlinearity errors can cause the converter to exhibit nonmonotonic behavior.

MSB
Most-significant bit of a byte or most-significant byte of a word.

n-channel FET
A field-effect transistor with an n-type conducting path (channel).

n-type material
Semiconductor material with introduced impurities (doping) causing it to have an excess of negatively charged carriers.

no missing codes
An A/D converter has no missing codes if, for every output code, there is a unique input voltage range which produces that code only. Large differential nonlinearity errors can cause the converter to miss codes.
nonlinearity
The maximum deviation of code transitions of the terminal-based characteristic from the corresponding code transitions of the ideal characteristic.

nonmaskable interrupts
Interrupts that cannot be masked (disabled) and cannot be assigned to the PTS for processing. The nonmaskable interrupts are unimplemented opcode, software trap, and NMI.

nonvolatile memory
Read-only memory that retains its contents when power is removed. Many MCS® 96 microcontrollers are available with either masked ROM, EPROM, or OTPROM. Consult the Automotive Products or Embedded Microcontrollers databook to determine which type of memory is available for a specific device.

npn transistor
A transistor consisting of one part p-type material and two parts n-type material.

off-isolation
The ability of an A/D converter to reject (isolate) the signal on a deselected (off) output.

OTPROM
One-time-programmable read-only memory. Similar to EPROM, but it comes in an unwindowed package and cannot be erased.

p-channel FET
A field-effect transistor with a p-type conducting path.

p-type material
Semiconductor material with introduced impurities (doping) causing it to have an excess of positively charged carriers.

PC
Program counter.

PCCBs
Programming chip configuration bytes, which are loaded into the chip configuration registers (CCRs) when the device is entering programming modes; otherwise, the CCBs are used.

PIC
Programmable interrupt controller. The module responsible for handling interrupts that are to be serviced by interrupt service routines that you provide. Also called simply the interrupt controller.
prioritized interrupt
Any *maskable interrupt* or nonmaskable NMI. Two of the *nonmaskable interrupts* (unimplemented opcode and software trap) are not prioritized; they vector directly to the *interrupt service routine* when executed.

program memory
A partition of memory where instructions can be stored for fetching and execution.

protected instruction
An instruction that prevents an interrupt from being acknowledged until after the next instruction executes. The protected instructions are DI, EI, DPTS, EPTS, POPA, POPF, PUSHA, and PUSHF.

PSW
Processor status word. The high byte of the PSW is the status byte, which contains one bit that globally enables or disables servicing of all maskable interrupts, one bit that enables or disables the PTS, and six Boolean flags that reflect the state of the current program. The low byte of the PSW is the INT_MASK register. A push or pop instruction saves or restores both bytes (PSW + INT_MASK).

PTS
Peripheral transaction server. The microcoded hardware interrupt processor.

PTSCB
See *PTS control block*.

PTS control block
A block of data required for each *PTS interrupt*. The microcode executes the proper *PTS routine* based on the contents of the PTS control block.

PTS cycle
The microcoded response to a *single* PTS interrupt request.

PTS interrupt
Any *maskable interrupt* that is assigned to the PTS for interrupt processing.

PTS mode
A microcoded response that enables the PTS to complete a specific task quickly. These tasks include transferring a single byte or word, transferring a block of bytes or words, managing multiple A/D conversions, and generating *PWM* outputs.

PTS routine
The entire microcoded response to multiple PTS interrupt requests. The PTS routine is controlled by the contents of the PTS control block.
PTS transfer
The movement of a single byte or word from the source memory location to the destination memory location.

PTS vector
A location in special-purpose memory that holds the starting address of a PTS control block.

PWM
Pulse-width modulated (outputs). The 8XC196MX devices have several options for producing PWM outputs: the generic pulse-width modulator modules, the waveform generator, and the EPA with or without the PTS. The 8XC196MD also has a frequency generator that produces PWM outputs.

quantizing error
An unavoidable A/D conversion error that results simply from the conversion of a continuous voltage to its integer digital representation. Quantizing error is always ± 0.5 LSB and is the only error present in an ideal A/D converter.

RALU
Register arithmetic-logic unit. A part of the CPU that consists of the ALU, the PSW, the master PC, the microcode engine, a loop counter, and six registers.

repeatability error
The difference between corresponding code transitions from different actual characteristics taken from the same converter on the same channel with the same temperature, voltage, and frequency conditions. The amount of repeatability error depends on the comparator’s ability to resolve very similar voltages and the extent to which random noise contributes to the error.

reserved memory
A memory location that is reserved for factory use or for future expansion. Do not use a reserved memory location except to initialize it with FFH.

resolution
The number of input voltage levels that an A/D converter can unambiguously distinguish between. The number of useful bits of information that the converter can return.

sample capacitor
A small (2–3 pF) capacitor used in the A/D converter circuitry to store the input voltage on the selected input channel.
sample delay
The time period between the time that A/D converter receives the “start conversion” signal and the time that the sample capacitor is connected to the selected channel.

sample delay uncertainty
The variation in the sample delay.

sample time
The period of time that the sample window is open. (That is, the length of time that the input channel is actually connected to the sample capacitor.)

sample time uncertainty
The variation in the sample time.

sample window
The period of time that begins when the sample capacitor is attached to a selected channel of an A/D converter and ends when the sample capacitor is disconnected from the selected channel.

sampled inputs
All input pins, with the exception of RESET#, are sampled inputs. The input pin is sampled one state time before the read buffer is enabled. Sampling occurs during PH1 (while CLKOUT is low) and resolves the value (high or low) of the pin before it is presented to the internal bus. If the pin value changes during the sample time, the new value may or may not be recorded during the read.

RESET# is a level-sensitive input. EXTINT is normally a sampled input; however, the powerdown circuitry uses EXTINT as a level-sensitive input during powerdown mode.

SAR

set
The “1” value of a bit or the act of giving it a “1” value. See also clear.

SFR
Special-function register.

SHORT-INTEGER
An 8-bit, signed variable with values from $-2^7$ through $+2^7 - 1$.

sign extension
A method for converting data to a larger format by filling the upper bit positions with the value of the sign. This conversion preserves the positive or negative value of signed integers.

sink current
Current flowing into a device to ground. Always a positive value.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>source current</td>
<td>Current flowing <strong>out of</strong> a device from $V_{CC}$. Always a negative value.</td>
</tr>
<tr>
<td>SP</td>
<td>Stack pointer.</td>
</tr>
<tr>
<td>special interrupt</td>
<td>Any of the three <strong>nonmaskable interrupts</strong> (unimplemented opcode, software trap, or NMI).</td>
</tr>
<tr>
<td>special-purpose memory</td>
<td>A partition of memory used for storing the <strong>interrupt vectors</strong>, <strong>PTS vectors</strong>, chip configuration bytes, and several reserved locations.</td>
</tr>
<tr>
<td>standard interrupt</td>
<td>Any <strong>maskable interrupt</strong> that is assigned to the <strong>interrupt controller</strong> for processing by an <strong>interrupt service routine</strong>.</td>
</tr>
<tr>
<td>state time (or state)</td>
<td>The basic time unit of the device; the combined period of the two internal timing signals, PH1 and PH2. (The internal clock generator produces PH1 and PH2 by halving the frequency of the signal on XTAL1. The rising edges of the active-high PH1 and PH2 signals generate CLKOUT, the output of the internal clock generator.) Because the device can operate at many frequencies, this manual defines time requirements in terms of state times rather than in specific units of time.</td>
</tr>
<tr>
<td>successive approximation</td>
<td>An A/D conversion method that uses a binary search to arrive at the best digital representation of an analog input.</td>
</tr>
<tr>
<td>temperature coefficient</td>
<td>Change in the stated variable for each degree Centigrade of temperature change.</td>
</tr>
<tr>
<td>temperature drift</td>
<td>The change in a specification due to a change in temperature. Temperature drift can be calculated by using the temperature coefficient for the specification.</td>
</tr>
<tr>
<td>terminal-based characteristic</td>
<td>An actual characteristic that has been translated and scaled to remove zero-offset error and full-scale error. A terminal-based characteristic resembles an actual characteristic with zero-offset error and full-scale error removed.</td>
</tr>
<tr>
<td>transfer function</td>
<td>A graph of output code versus input voltage; the characteristic of the A/D converter.</td>
</tr>
</tbody>
</table>
transfer function errors

Errors inherent in an analog-to-digital conversion process: quantizing error, zero-offset error, full-scale error, differential nonlinearity, and nonlinearity. Errors that are hardware-dependent, rather than being inherent in the process itself, include feedthrough, repeatability, channel-to-channel matching, off-isolation, and $V_{cc}$ rejection errors.

UART

Universal asynchronous receiver and transmitter. A part of the serial I/O port.

$V_{cc}$ rejection

The property of an A/D converter that causes it to ignore (reject) changes in $V_{cc}$ so that the actual characteristic is unaffected by those changes. The effectiveness of $V_{cc}$ rejection is measured by the ratio of the change in $V_{cc}$ to the change in the actual characteristic.

watchdog timer

An internal timer that resets the device if software fails to respond before the timer overflows.

waveform generator

One of the 8XC196Mx peripherals that can be used to produce pulse-width modulated (PWM) outputs. The waveform generator is optimized for controlling 3-phase AC induction motors, brushless DC motors, and other devices requiring multiple PWM outputs.

WDT

See watchdog timer.

word

Any 16-bit unit of data.

WORD

An unsigned, 16-bit variable with values from 0 through $2^{16}-1$.

zero extension

A method for converting data to a larger format by filling the upper bit positions with zeros.

zero-offset error

An ideal A/D converter’s first code transition occurs when the input voltage is 0.5 LSB. Zero-offset error is the difference between 0.5 LSB and the actual input voltage that triggers an A/D converter’s first code transition.
INDEX

A
A/D converter, signals, 6-1
AD_COMMAND register, 6-2
AD_RESULT register, 6-3
Auto programming mode
circuit, 10-3
memory map, 10-2

B
Block diagram
CAN peripheral, 7-2
clock circuitry, 2-2
core and peripherals, 2-2
Bus-timing modes, 9-1–9-2
comparison, 9-1, 9-2

C
CAN serial communications controller, 7-1–7-42
address map, 7-5
bit timing, 7-10–7-12
block diagram, 7-2
bus-off state, 7-41
error detection and management logic, 7-9
message
acceptance filtering, 7-6
frames, 7-7
extended, 7-8
standard, 7-8
identifiers, effect of masking on, 7-7
objects, 7-5–7-6
overview, 7-1–7-2
programming, 7-4–7-31
receive and transmit priorities, 7-6
registers, 7-3–7-4
signals, 7-3
CAN_BTIME0 register, 7-3, 7-15
CAN_BTIME1 register, 7-3, 7-16
CAN_CON register, 7-3, 7-13, 7-29
CAN_EGMSK register, 7-3, 7-19
CAN_INT register, 7-3, 7-32
CAN_MSGxCFG register, 7-3, 7-21
CAN_MSGxCON0 register, 7-3, 7-24, 7-31, 7-34
CAN_MSGxCON1 register, 7-4, 7-26
CAN_MSGxDATA0-7 register, 7-28
CAN_MSGxDATAx register, 7-4
CAN_MSGxID register, 7-4
CAN_MSGxID0-3 register, 7-22
CAN_MSK15 register, 7-4, 7-20
CAN_SGMSK register, 7-4, 7-18
CAN_STAT register, 7-4, 7-33
CCR1 register, 9-3
CLKOUT, and internal timing, 2-2–2-4
Clock circuitry, 2-3
Clock phases, internal, 2-4

D
Documents, related, 1-2

E
EP_DIR register, 5-2
EP_MODE register, 5-2
EPORT, 5-1
EP_PIN register, 5-3
EP_REG register, 5-3

F
Formulas
clock period (t), 2-4
PH1 and PH2 frequency, 2-4
state time, 2-4
Frequency (f), 2-4
F_{XTAL1}, 2-4

I
Idle mode, pin status, A-14
Interrupts, 4-1
INT_MASK1 register, 4-2
INT_PEND1 register, 4-2

M
Manual contents, summary, 1-1
Memory mapping
auto programming mode, 10-2
serial port programming mode, 10-3
**P**

P0_PIN register, 5-1
Period (t), 2-4
Pin diagrams, A-1
Pins, reset status, A-14–A-15
Port 0, 5-1
Powerdown mode, pin status, A-14

**R**

Registers
- AD_COMMAND, 6-2
- AD_RESULT, 6-3
- CAN_BTIME0, 7-3, 7-15
- CAN_BTIME1, 7-3, 7-16
- CAN_CON, 7-3, 7-13, 7-29
- CAN_EGMSK, 7-3, 7-19
- CAN_INT, 7-3, 7-32
- CAN_MSGxCFG, 7-3, 7-21
- CAN_MSGxCON0, 7-3, 7-24, 7-31, 7-34
- CAN_MSGxCON1, 7-4, 7-26
- CAN_MSGxDATA0-7, 7-28
- CAN_MSGxDATAx, 7-4
- CAN_MSGxID, 7-4
- CAN_MSGxID0-3, 7-22
- CAN_MSK15, 7-4, 7-20
- CAN_SGMSK, 7-4, 7-18
- CAN_STAT, 7-4, 7-33
- CCR1, 9-3
- EP_DIR, 5-2
- EP_MODE, 5-2
- EP_PIN, 5-3
- EP_REG, 5-3
- INT_MASK1, 4-2
- INT_PEND1, 4-2
- P0_PIN, 5-1
Reset status, I/O and control pins, A-14

**S**

Serial port programming mode, 10-4
SFRs, windowed direct addresses, 3-11
Signal descriptions, A-4–A-14
State time, defined, 2-4

**T**

Timing
- internal, 2-2, 2-4

selectable bus-timing, 8-1

**W**

Windows
- and memory-mapped SFRs, 3-9
- locations that cannot be windowed, 3-9
- table of, 3-11
- WSR values and direct addresses, 3-9