IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

<table>
<thead>
<tr>
<th>Products</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amplifiers</td>
<td>Audio</td>
</tr>
<tr>
<td>Data Converters</td>
<td>Automotive</td>
</tr>
<tr>
<td>DSP</td>
<td>Broadband</td>
</tr>
<tr>
<td>Interface</td>
<td>Digital Control</td>
</tr>
<tr>
<td>Logic</td>
<td>Military</td>
</tr>
<tr>
<td>Power Mgmt</td>
<td>Optical Networking</td>
</tr>
<tr>
<td>Microcontrollers</td>
<td>Security</td>
</tr>
<tr>
<td>Low Power Wireless</td>
<td>Telephony</td>
</tr>
<tr>
<td></td>
<td>Video &amp; Imaging</td>
</tr>
<tr>
<td></td>
<td>Wireless</td>
</tr>
</tbody>
</table>

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

Copyright © 2007, Texas Instruments Incorporated
Preface

Read This First

About This Manual

This document describes the inter-integrated circuit (I2C) module in the TMS320C6000™ DSP family. This document assumes the reader is familiar with the I2C-bus specification.

Notational Conventions

This document uses the following conventions.

- Hexadecimal numbers are shown with the suffix h. For example, the following number is 40 hexadecimal (decimal 64): 40h.

- Registers in this document are shown in figures and described in tables.

- Each register figure shows a rectangle divided into fields that represent the fields of the register.

- Each field is labeled with its bit name, its beginning and ending bit numbers above, and its read/write properties below. A legend explains the notation used for the properties.

- Reserved bits in a register figure designate a bit that is used for future device expansion.

Related Documentation From Texas Instruments

The following documents describe the C6000™ devices and related support tools. Copies of these documents are available on the Internet at www.ti.com. Tip: Enter the literature number in the search box provided at www.ti.com.

**TMS320C6000 CPU and Instruction Set Reference Guide** (literature number SPRU189) describes the TMS320C6000™ CPU architecture, instruction set, pipeline, and interrupts for these digital signal processors.

**TMS320C6000 DSP Peripherals Overview Reference Guide** (literature number SPRU190) describes the peripherals available on the TMS320C6000™ DSPs.
**TMS320C6000 Technical Brief** (literature number [SPRU197]) gives an introduction to the TMS320C62x™ and TMS320C67x™ DSPs, development tools, and third-party support.

**TMS320C6000 Programmer’s Guide** (literature number [SPRU198]) describes ways to optimize C and assembly code for the TMS320C6000™ DSPs and includes application program examples.

**TMS320C6000 Code Composer Studio Tutorial** (literature number [SPRU301]) introduces the Code Composer Studio™ integrated development environment and software tools.

**Code Composer Studio Application Programming Interface Reference Guide** (literature number [SPRU321]) describes the Code Composer Studio™ application programming interface (API), which allows you to program custom plug-ins for Code Composer.

**TMS320C6x Peripheral Support Library Programmer’s Reference** (literature number [SPRU273]) describes the contents of the TMS320C6000™ peripheral support library of functions and macros. It lists functions and macros both by header file and alphabetically, provides a complete description of each, and gives code examples to show how they are used.

**TMS320C6000 Chip Support Library API Reference Guide** (literature number [SPRU401]) describes a set of application programming interfaces (APIs) used to configure and control the on-chip peripherals.

**Trademarks**

Code Composer Studio, C6000, C62x, C64x, C67x, TMS320C6000, TMS320C62x, TMS320C64x, TMS320C67x, and VelociTI are trademarks of Texas Instruments.
## Contents

1 Introduction to the I2C Module ......................................................... 1
   1.1 Features ........................................................................... 2
   1.2 Features Not Supported .................................................. 2

2 Functional Overview ...................................................................... 3

3 Operational Details ...................................................................... 5
   3.1 Input and Output Voltage Levels ....................................... 5
   3.2 Data Validity ...................................................................... 5
   3.3 Operating Modes .................................................................. 5
   3.4 START and STOP Conditions ............................................ 7
   3.5 Serial Data Formats ........................................................... 8
      3.5.1 7-Bit Addressing Format ........................................... 8
      3.5.2 10-Bit Addressing Format .......................................... 9
      3.5.3 Free Data Format ....................................................... 10
      3.5.4 Using a Repeated START Condition .......................... 10
   3.6 NACK Bit Generation ......................................................... 11
   3.7 Arbitration ......................................................................... 12
   3.8 Clock Generation ............................................................... 13
   3.9 Clock Synchronization ....................................................... 14

4 Interrupt Requests Generated by the I2C Module ......................... 15

5 EDMA Events Generated by the I2C Module ................................. 17

6 Resetting/Disabling the I2C Module ............................................. 17

7 Programming Guide ...................................................................... 18
   7.1 Main Program ................................................................. 18
      7.1.1 State After Reset ..................................................... 18
      7.1.2 Initialization Procedure .......................................... 18
      7.1.3 Program Clock Control Registers (I2CCLKL and I2CCLKH) ........................................................................ 18
      7.1.4 Configure Address Registers .................................... 18
      7.1.5 Program Transmit Data Register (I2CDXR) .................. 18
      7.1.6 Configure Status and Mode Register (I2CSTR) ............. 18
      7.1.7 Poll Receive Data ..................................................... 18
      7.1.8 Poll Transmit Data ................................................. 19
# Figures

1. Multiple I2C Modules Connected ................................................................. 1
2. I2C Module Conceptual Block Diagram ......................................................... 4
3. Bit Transfer on the I2C-Bus ...................................................................... 5
4. I2C Module START and STOP Conditions .................................................. 7
5. I2C Module Data Transfer ........................................................................... 8
6. I2C Module 7-Bit Addressing Format (FDF = 0, XA = 0 in I2CMDR) .......... 9
7. I2C Module 10-Bit Addressing Format With Master-Transmitter Writing to Slave-Receiver (FDF = 0, XA = 1 in I2CMDR) ......................... 9
8. I2C Module Free Data Format (FDF = 1 in I2CMDR) .................................. 10
9. I2C Module 7-Bit Addressing Format With Repeated START Condition (FDF = 0, XA = 0 in I2CMDR) ................................................... 10
10. Arbitration Procedure Between Two Master-Transmitters ...................... 12
11. I2C Input Clock ....................................................................................... 14
12. Synchronization of Two I2C Clock Generators During Arbitration ........ 15
13. Enable Paths of I2C Interrupt Requests .................................................... 18
14. Setup Procedure ....................................................................................... 21
15. Using Bit Polling for Master-Transmitter Operation, Repeat Mode (RM = 1) ........ 22
16. Using Bit Polling for Master-Transmitter Operation, Nonrepeat Mode (RM = 0) .... 23
17. Using Bit Polling for Master-Receiver Operation, Repeat Mode (RM = 1), Fixed Number of Data Words ...................................................... 24
18. Using Bit Polling for Master-Receiver Operation, Repeat Mode (RM = 1), Variable (Data-Dependent) Number of Data Words ................. 25
19. Using Bit Polling for Master-Receiver Operation, Nonrepeat Mode (RM = 0) ...... 26
20. Using Bit Polling for Slave-Transmitter/Receiver Operation, Repeat Mode (RM = 1) .......................................................... 27
21. Using Interrupts for Master-Transmitter Operation, Nonrepeat Mode (RM = 0) .... 28
22. Using Interrupts for Master-Receiver Operation, Nonrepeat Mode (RM = 0) ...... 29
23. Using Interrupts for Master-Transmitter/Receiver Operation, Repeat Mode (RM = 1) .......................................................... 30
24. Using Interrupts for Slave-Transmitter/Receiver Operation, Repeat Mode (RM = 1) .......................................................... 31
25. Using EDMA Events for Master-Transmitter Operation, Nonrepeat Mode (RM = 0) .... 32
26. Using EDMA Events for Master-Receiver Operation, Nonrepeat Mode (RM = 0) .... 33
27. I2C Own Address Register (I2COAR) ......................................................... 35
28. I2C Interrupt Enable Register (I2CIER) ....................................................... 36
29. I2C Status Register (I2CSTR) .................................................................. 37
30. Roles of the Clock Divide-Down Values (ICCL and ICCH) ..................... 43
31. I2C Clock Low-Time Divider Register (I2CCLKL) ................................... 44
32. I2C Clock High-Time Divider Register (I2CCLKH) ................................... 45
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>33.</td>
<td>I2C Data Count Register (I2CCNT)</td>
<td>46</td>
</tr>
<tr>
<td>34.</td>
<td>I2C Data Receive Register (I2CDRR)</td>
<td>47</td>
</tr>
<tr>
<td>35.</td>
<td>I2C Slave Address Register (I2CSAR)</td>
<td>48</td>
</tr>
<tr>
<td>36.</td>
<td>I2C Data Transmit Register (I2CDXR)</td>
<td>49</td>
</tr>
<tr>
<td>37.</td>
<td>I2C Mode Register (I2CMDR)</td>
<td>50</td>
</tr>
<tr>
<td>38.</td>
<td>Block Diagram Showing the Effects of the Digital Loopback Mode (DLB) Bit</td>
<td>57</td>
</tr>
<tr>
<td>39.</td>
<td>I2C Interrupt Source Register (I2CISR)</td>
<td>58</td>
</tr>
<tr>
<td>40.</td>
<td>I2C Prescaler Register (I2CPSC)</td>
<td>59</td>
</tr>
<tr>
<td>41.</td>
<td>I2C Peripheral Identification Register 1 (I2CPID1)</td>
<td>60</td>
</tr>
<tr>
<td>42.</td>
<td>I2C Peripheral Identification Register 2 (I2CPID2)</td>
<td>61</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1.</td>
<td>Operating Modes of the I2C Module</td>
<td>6</td>
</tr>
<tr>
<td>2.</td>
<td>Ways to Generate a NACK Bit</td>
<td>11</td>
</tr>
<tr>
<td>3.</td>
<td>Descriptions of the I2C Interrupt Requests</td>
<td>17</td>
</tr>
<tr>
<td>4.</td>
<td>I2C Module Registers</td>
<td>34</td>
</tr>
<tr>
<td>5.</td>
<td>I2C Own Address Register (I2COAR) Field Descriptions</td>
<td>35</td>
</tr>
<tr>
<td>6.</td>
<td>I2C Interrupt Enable Register (I2CIER) Field Descriptions</td>
<td>36</td>
</tr>
<tr>
<td>7.</td>
<td>I2C Status Register (I2CSTR) Field Descriptions</td>
<td>38</td>
</tr>
<tr>
<td>8.</td>
<td>I2C Clock Low-Time Divider Register (I2CCLKL) Field Descriptions</td>
<td>44</td>
</tr>
<tr>
<td>9.</td>
<td>I2C Clock High-Time Divider Register (I2CCLKH) Field Descriptions</td>
<td>45</td>
</tr>
<tr>
<td>10.</td>
<td>I2C Data Count Register (I2CCNT) Field Descriptions</td>
<td>46</td>
</tr>
<tr>
<td>11.</td>
<td>I2C Data Receive Register (I2CDRR) Field Descriptions</td>
<td>47</td>
</tr>
<tr>
<td>12.</td>
<td>I2C Slave Address Register (I2CSAR) Field Descriptions</td>
<td>48</td>
</tr>
<tr>
<td>13.</td>
<td>I2C Data Transmit Register (I2CDXR) Field Descriptions</td>
<td>49</td>
</tr>
<tr>
<td>14.</td>
<td>I2C Mode Register (I2CMDR) Field Descriptions</td>
<td>50</td>
</tr>
<tr>
<td>15.</td>
<td>Master-Transmitter/Receiver Bus Activity Defined by RM, STT, and STP Bits</td>
<td>56</td>
</tr>
<tr>
<td>16.</td>
<td>How the MST and FDF Bits Affect the Role of TRX Bit</td>
<td>56</td>
</tr>
<tr>
<td>17.</td>
<td>I2C Interrupt Source Register (I2CISR) Field Descriptions</td>
<td>58</td>
</tr>
<tr>
<td>18.</td>
<td>I2C Prescaler Register (I2CPSC) Field Descriptions</td>
<td>59</td>
</tr>
<tr>
<td>19.</td>
<td>I2C Peripheral Identification Register 1 (I2CPID1) Field Descriptions</td>
<td>60</td>
</tr>
<tr>
<td>20.</td>
<td>I2C Peripheral Identification Register 2 (I2CPID2) Field Descriptions</td>
<td>61</td>
</tr>
<tr>
<td>21.</td>
<td>Document Revision History</td>
<td>62</td>
</tr>
</tbody>
</table>
The inter-integrated circuit (I2C) module provides an interface between a TMS320C6000™ DSP and other devices compliant with Philips Semiconductors Inter-IC bus (I2C-bus) specification version 2.1 and connected by way of an I2C-bus. External components attached to this 2-wire serial bus can transmit/receive up to 8-bit data to/from the C6000™ DSP through the I2C module. To determine whether a particular C6000 DSP has an I2C module, see the data sheet for that DSP. This chapter assumes the reader is familiar with the I2C-bus specification.

1 Introduction to the I2C Module

The I2C module supports any slave or master I2C-compatible device. Figure 1 shows an example of multiple I2C modules connected for a two-way transfer from one device to other devices.

Figure 1. Multiple I2C Modules Connected

C6000 and TMS320C6000 are trademarks of Texas Instruments.
1.1 Features

The I2C module has the following features:

- Compliance with the Philips Semiconductors I^2^C-bus specification (version 2.1):
  - Support for byte format transfer
  - 7-bit and 10-bit addressing modes
  - General call
  - START byte mode
  - Support for multiple master-transmitters and slave-receivers
  - Support for multiple slave-transmitters and master-receivers
  - Combined master transmit/receive and receive/transmit mode
  - Data transfer rate of from 10 kbps up to 400 kbps (Philips Fast-mode rate)

- One read EDMA event and one write EDMA event, which can be used by the EDMA controller

- One interrupt that can be used by the CPU. This interrupt can be generated as a result of one of the following conditions: transmit-data ready, receive-data ready, register-access ready, no-acknowledgement received, arbitration lost.

- Module enable/disable capability

- Free data format mode

1.2 Features Not Supported

The I2C module does not support:

- High-speed mode (Hs-mode)
- CBUS compatibility mode
2 Functional Overview

Each device connected to an I²C-bus, including any C6000 DSP connected to the bus with an I²C module, is recognized by a unique address. Each device can operate as either a transmitter or a receiver, depending on the function of the device. In addition, a device connected to the I²C-bus can also be considered as the master or the slave when performing data transfers. Note that a master device is the device that initiates a data transfer on the bus and generates the clock signals to permit that transfer. During this transfer, any device addressed by this master is considered a slave. The I²C module supports the multi-master mode, in which one or more devices capable of controlling an I²C-bus can be connected to the same I²C-bus.

For data communication, the I²C module has a serial data pin (SDA) and a serial clock pin (SCL), as shown in Figure 2. These two pins carry information between the C6000 device and other devices connected to the I²C-bus. The SDA and SCL pins both are bidirectional. They each must be connected to a positive supply voltage using a pull-up resistor. When the bus is free, both pins are high. The driver of these two pins has an open-drain configuration to perform the required wired-AND function.

The I²C module consists of the following primary blocks:

- A serial interface: one data pin (SDA) and one clock pin (SCL)
- Data registers to temporarily hold receive data and transmit data traveling between the SDA pin and the CPU or the EDMA controller
- Control and status registers
- A peripheral data bus interface to enable the CPU and the EDMA controller to access the I²C module registers
- A clock synchronizer to synchronize the I²C input clock (from the DSP clock generator) and the clock on the SCL pin, and to synchronize data transfers with masters of different clock speeds
- A prescaler to divide down the input clock that is driven to the I²C module
- A noise filter on each of the two pins, SDA and SCL
- An arbitrator to handle arbitration between the I²C module (when it is a master) and another master
- Interrupt generation logic, so that an interrupt can be sent to the CPU
- EDMA event generation logic, so that activity in the EDMA controller can be synchronized to data reception and data transmission in the I²C module
Figure 2 shows the four registers used for transmission and reception. The CPU or the EDMA controller writes data for transmission to I2CDXR and reads received data from I2CDRR. When the I2C module is configured as a transmitter, data written to I2CDXR is copied to I2CXSR and shifted out on the SDA pin one bit at a time. When the I2C module is configured as a receiver, received data is shifted into I2CRSR and then copied to I2CDRR.

**Figure 2. I2C Module Conceptual Block Diagram**
3 Operational Details

3.1 Input and Output Voltage Levels

One clock pulse is generated by the master device for each data bit transferred. Due to a variety of different technology devices that can be connected to the I^2C-bus, the levels of logic 0 (low) and logic 1 (high) are not fixed and depend on the associated level of V_{DD}. For details, see the device-specific datasheet.

3.2 Data Validity

The data on SDA must be stable during the high period of the clock (see Figure 3). The high or low state of the data line, SDA, can change only when the clock signal on SCL is low.

Figure 3. Bit Transfer on the I^2C-Bus

3.3 Operating Modes

The I^2C module has four basic operating modes to support data transfers as a master and as a slave. See Table 1 for the names and descriptions of the modes.

If the I^2C module is a master, it begins as a master-transmitter and, typically, transmits an address for a particular slave. When giving data to the slave, the I^2C module must remain a master-transmitter. In order to receive data from a slave, the I^2C module must be changed to the master-receiver mode.

If the I^2C module is a slave, it begins as a slave-receiver and, typically, sends acknowledgement when it recognizes its slave address from a master. If the master will be sending data to the I^2C module, the module must remain a slave-receiver. If the master has requested data from the I^2C module, the module must be changed to the slave-transmitter mode.
Operational Details

Table 1. Operating Modes of the I2C Module

<table>
<thead>
<tr>
<th>Operating Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slave-receiver mode</td>
<td>The I2C module is a slave and receives data from a master. All slave modules begin in this mode.</td>
</tr>
<tr>
<td></td>
<td>In this mode, serial data bits received on SDA are shifted in with the clock pulses that are generated by the master. As a slave, the I2C module does not generate the clock signal, but it can hold SCL low while the intervention of the DSP is required (RSFULL = 1 in I2CSTR) after data has been received.</td>
</tr>
<tr>
<td>Slave-transmitter mode</td>
<td>The I2C module is a slave and transmits data to a master.</td>
</tr>
<tr>
<td></td>
<td>This mode can only be entered from the slave-receiver mode; the I2C module must first receive a command from the master. When you are using any of the 7-bit/10-bit addressing formats, the I2C module enters its slave-transmitter mode if the slave address is the same as its own address (in I2COAR) and the master has transmitted (R/W = 1). As a slave-transmitter, the I2C module then shifts the serial data out on SDA with the clock pulses that are generated by the master. While a slave, the I2C module does not generate the clock signal, but it can hold SCL low while the intervention of the DSP is required (XSMT = 0 in I2CSTR) after data has been transmitted.</td>
</tr>
<tr>
<td>Master-receiver mode</td>
<td>The I2C module is a master and receives data from a slave.</td>
</tr>
<tr>
<td></td>
<td>This mode can only be entered from the master-transmitter mode; the I2C module must first transmit a command to the slave. When you are using any of the 7-bit/10-bit addressing formats, the I2C module enters its master-receiver mode after transmitting the slave address and (R/W = 1). Serial data bits on SDA are shifted into the I2C module with the clock pulses generated by the I2C module on SCL. The clock pulses are inhibited and SCL is held low when the intervention of the DSP is required (RSFULL = 1 in I2CSTR) after data has been received.</td>
</tr>
<tr>
<td>Master-transmitter mode</td>
<td>The I2C module is a master and transmits control information and data to a slave. All master modules begin in this mode.</td>
</tr>
<tr>
<td></td>
<td>In this mode, data assembled in any of the 7-bit/10-bit addressing formats is shifted out on SDA. The bit shifting is synchronized with the clock pulses generated by the I2C module on SCL. The clock pulses are inhibited and SCL is held low when the intervention of the DSP is required (XSMT = 0 in I2CSTR) after data has been transmitted.</td>
</tr>
</tbody>
</table>
3.4 START and STOP Conditions

START and STOP conditions can be generated by the I2C module when the module is configured to be a master on the I2C-bus. As shown in Figure 4:

- The START condition is defined as a high-to-low transition on the SDA line while SCL is high. A master drives this condition to indicate the start of a data transfer.
- The STOP condition is defined as a low-to-high transition on the SDA line while SCL is high. A master drives this condition to indicate the end of a data transfer.

Figure 4. I2C Module START and STOP Conditions

After a START condition and before a subsequent STOP condition, the I2C-bus is considered busy, and the bus busy (BB) bit of I2CSTR is 1. Between a STOP condition and the next START condition, the bus is considered free, and BB is 0.

For the I2C module to start a data transfer with a START condition, the master mode (MST) bit and the START condition (STT) bit in I2CMODR must both be 1. For the I2C module to end a data transfer with a STOP condition, the STOP condition (STP) bit must be set to 1. When BB is set to 1 and STT is set to 1, a repeated START condition is generated. For a description of I2CMODR (including the MST, STT, and STP bits), see section 8.9.
3.5 Serial Data Formats

Figure 5 shows an example of a data transfer on the I2C-bus. The I2C module supports 1-bit to 8-bit data values. Figure 5 is shown in an 8-bit data format (BC = 000 in I2CMDR). Each bit put on the SDA line equates to 1 pulse on the SCL line and the data is always transferred with the most-significant bit (MSB) first. The number of data values that can be transmitted or received is unrestricted; however, the transmitters and receivers must agree on the number of data values being transferred. The I2C module supports the following data formats.

- 7-bit addressing mode
- 10-bit addressing mode
- Free data format mode

Figure 5. I2C Module Data Transfer

3.5.1 7-Bit Addressing Format

In the 7-bit addressing format (Figure 6), the first byte after a START condition (S) consists of a 7-bit slave address followed by a R/W bit. The R/W bit determines the direction of the data:

- R/W = 0: The master writes (transmits) data to the addressed slave.
- R/W = 1: The master reads (receives) data from the slave.

An extra clock cycle dedicated for acknowledgement (ACK) is inserted after the R/W bit. If the ACK bit is inserted by the slave, it is followed by n bits of data from the transmitter (master or slave, depending on the R/W bit). n is a number from 1 to 8 determined by the bit count (BC) bits of I2CMDR. After the data bits have been transferred, the receiver inserts an ACK bit.

To select the 7-bit addressing format, write 0 to the expanded address enable (XA) bit of I2CMDR.
Figure 6. **I2C Module 7-Bit Addressing Format (FDF = 0, XA = 0 in I2CMDR)**

<table>
<thead>
<tr>
<th>1</th>
<th>7</th>
<th>1</th>
<th>1</th>
<th>n</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Slave address</td>
<td>R/W</td>
<td>ACK</td>
<td>Data</td>
<td>ACK</td>
<td>Data</td>
</tr>
<tr>
<td>n</td>
<td>Data</td>
<td>ACK</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** \( n \) = The number of data bits (from 1 to 8) specified by the bit count (BC) field of I2CMDR.

### 3.5.2 10-Bit Addressing Format

The 10-bit addressing format (Figure 7) is like the 7-bit addressing format, but the master sends the slave address in two separate byte transfers. The first byte consists of 11110\(_b\), the two MSBs of the 10-bit slave address, and \( R/W = 0 \) (write). The second byte is the remaining 8 bits of the 10-bit slave address. The slave must send acknowledgement (ACK) after each of the two byte transfers. Once the master has written the second byte to the slave, the master can either write data or use a repeated START condition to change the data direction. (For more details about using 10-bit addressing, see the Philips Semiconductors I\(^2\)C-bus specification.)

To select the 10-bit addressing format, write 1 to the XA bit of I2CMDR.

Figure 7. **I2C Module 10-Bit Addressing Format With Master-Transmitter Writing to Slave-Receiver (FDF = 0, XA = 1 in I2CMDR)**

<table>
<thead>
<tr>
<th>1</th>
<th>7</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A A</td>
<td>ACK</td>
<td>A A A A A A A A</td>
<td>ACK</td>
</tr>
<tr>
<td>R/W</td>
<td>8 MSBs of slave address</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** \( n \) = The number of data bits (from 1 to 8) specified by the bit count (BC) field of I2CMDR.
3.5.3 Free Data Format

In the free data format (Figure 8), the first bits after a START condition (S) are a data word. An ACK bit is inserted after each data word, which can be from 1 to 8 bits, depending on BC of I2CMDR. No address or data-direction bit is sent. Therefore, the transmitter and the receiver must both support the free data format, and the direction of the data must be constant throughout the transfer.

To select the free data format, write 1 to the free data format (FDF) bit of I2CMDR.

Figure 8. I2C Module Free Data Format (FDF = 1 in I2CMDR)

<table>
<thead>
<tr>
<th>1</th>
<th>n</th>
<th>1</th>
<th>n</th>
<th>1</th>
<th>n</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Data</td>
<td>ACK</td>
<td>Data</td>
<td>ACK</td>
<td>Data</td>
<td>ACK</td>
<td>P</td>
</tr>
</tbody>
</table>

Note: $n =$ The number of data bits (from 1 to 8) specified by the bit count (BC) field of I2CMDR.

3.5.4 Using a Repeated START Condition

The repeated START condition can be used with the 7-bit addressing, 10-bit addressing, and free data formats. The 7-bit addressing format using a repeated START condition (S) is shown in Figure 9. At the end of each data word, the master can drive another START condition. Using this capability, a master can transmit/receive any number of data words before driving a STOP condition. The length of a data word can be from 1 to 8 bits and is selected with BC of I2CMDR.

Figure 9. I2C Module 7-Bit Addressing Format With Repeated START Condition (FDF = 0, XA = 0 in I2CMDR)

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>n</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>n</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Slave address</td>
<td>R/W</td>
<td>ACK</td>
<td>Data</td>
<td>ACK</td>
<td>Slave address</td>
<td>R/W</td>
<td>ACK</td>
</tr>
</tbody>
</table>

Note: $n =$ The number of data bits (from 1 to 8) specified by the bit count (BC) field of I2CMDR.
3.6 NACK Bit Generation

When the I2C module is a receiver (master or slave), it can acknowledge or ignore bits sent by the transmitter. To ignore any new bits, the I2C module must send a no-acknowledge (NACK) bit during the acknowledge cycle on the bus. Table 2 summarizes the various ways the I2C module sends a NACK bit.

Table 2. Ways to Generate a NACK Bit

<table>
<thead>
<tr>
<th>I2C Module Condition</th>
<th>Basic</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slave-receiver mode</td>
<td>Disable data transfers (STT = 0 in I2CMDR).</td>
<td>Set the NACKMOD bit of I2CMDR before the rising edge of the last data bit you intend to receive.</td>
</tr>
<tr>
<td></td>
<td>Allow an overrun condition (RSFULL = 1 in I2CSTR).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reset the module (IRS = 0 in I2CMDR).</td>
<td></td>
</tr>
<tr>
<td>Master-receiver mode AND Repeat mode</td>
<td>Generate a STOP condition (STOP = 1 in I2CMDR).</td>
<td>Set the NACKMOD bit of I2CMDR before the rising edge of the last data bit you intend to receive.</td>
</tr>
<tr>
<td>(RM = 1 in I2C MDR)</td>
<td>Reset the module (IRS = 0 in I2CMDR).</td>
<td></td>
</tr>
<tr>
<td>Master-receiver mode AND Nonrepeat</td>
<td>If STP = 1 in I2CMDR, allow the internal data counter to count down to 0 and force a STOP condition.</td>
<td>Set the NACKMOD bit of I2CMDR before the rising edge of the last data bit you intend to receive.</td>
</tr>
<tr>
<td>mode (RM = 0 in I2C MDR)</td>
<td>If STP = 0, make STP = 1 to generate a STOP condition.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reset the module (IRS = 0 in I2CMDR).</td>
<td></td>
</tr>
</tbody>
</table>
3.7 Arbitration

If two or more master-transmitters simultaneously start a transmission on the same bus, an arbitration procedure is invoked. The arbitration procedure uses the data presented on the serial data bus (SDA) by the competing transmitters. Figure 10 illustrates the arbitration procedure between two devices. The first master-transmitter, which drives SDA high, is overruled by another master-transmitter that drives SDA low. The arbitration procedure gives priority to the device that transmits the serial data stream with the lowest binary value. Should two or more devices send identical first bytes, arbitration continues on the subsequent bytes.

If the I2C module is the losing master, it switches to the slave-receiver mode, sets the arbitration lost (AL) flag, and generates the arbitration-lost interrupt.

If during a serial transfer the arbitration procedure is still in progress when a repeated START condition or a STOP condition is transmitted to SDA, the master-transmitters involved must send the repeated START condition or the STOP condition at the same position in the format frame. Arbitration is not allowed between:

- A repeated START condition and a data bit
- A STOP condition and a data bit
- A repeated START condition and a STOP condition

Figure 10. Arbitration Procedure Between Two Master-Transmitters
3.8 Clock Generation

The DSP clock generator receives a signal from an external clock source and produces an I2C input clock as shown in Figure 11 below. A programmable prescaler in the I2C module divides the I2C input clock down to produce a prescaled module clock. You must operate the prescaled module clock within the 6.7–13.3 MHz range. The I2C clock dividers divide the high and low portions of the prescaled module clock signal down to produce the I2C serial clock, which appears on the SCL pin when you configure the I2C module to be a master on the I2C bus.
Operational Details

**Figure 11. I2C Input Clock**

![Diagram of I2C Input Clock]

**Register bits**

- IPSC (I2CPSC[7:0])
- ICCL (I2CCLKL[15:0])
- ICCH (I2CCLKH[15:0])

**Prescaled Module Clock Frequency Range**

The I2C module must be operated with a prescaled module clock frequency of 6.7 to 13.3 MHz. The I2C prescaler register (I2CPSC) must be configured to this frequency range.

<table>
<thead>
<tr>
<th>IPSC value</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2h–FFh</td>
<td>5</td>
</tr>
</tbody>
</table>
You must only initialize the prescaler (I2CPSC) while the I2C module is in the reset state (the IRS bit = 0 in the I2CMDR register). The prescaled frequency only takes effect when you change the IRS bit to 1. Changing the IPSC value while IRS = 1 has no effect. Likewise, you must configure the I2C clock dividers (I2CCLKL and I2CCLKH) while the I2C module is still in reset (the IRS bit = 0 in the I2CMDR register).

### 3.9 Clock Synchronization

Under normal conditions, only one master device generates the clock signal, SCL. During the arbitration procedure, however, there are two or more masters and the clock must be synchronized so that the data output can be compared. Figure 12 illustrates the clock synchronization. The wired-AND property of SCL means that a device that first generates a low period on SCL (device #1) overrules the other devices. At this high-to-low transition, the clock generators of the other devices are forced to start their own low period. The SCL is held low by the device with the longest low period. The other devices that finish their low periods must wait for SCL to be released, before starting their high periods. A synchronized signal on SCL is obtained, where the slowest device determines the length of the low period and the fastest device determines the length of the high period.

If a device pulls down the clock line for a longer time, the result is that all clock generators must enter the wait state. In this way, a slave slows down a fast master and the slow device creates enough time to store a received data word or to prepare a data word to be transmitted.

**Figure 12. Synchronization of Two I2C Clock Generators During Arbitration**
4  **Interrupt Requests Generated by the I2C Module**

The I2C module can generate the interrupt requests described in Table 3. Two of these requests indicate to the CPU when to write transmit data and when to read receive data. As shown in Figure 13, all requests are multiplexed through an arbiter to a single I2C interrupt request to the CPU. Each interrupt request has a flag bit in the status register (I2CSTR) and an enable bit in the interrupt enable register (I2CIER). When one of the specified events occurs, its flag bit is set. If the corresponding enable bit is 0, the interrupt request is blocked. If the enable bit is 1, the request is forwarded to the CPU as an I2C interrupt.

The I2C interrupt is one of the maskable interrupts of the CPU. As with any maskable interrupt request, if it is properly enabled in the CPU, the CPU executes the corresponding interrupt service routine (ISR). The ISR for the I2C interrupt can determine the interrupt source by reading the interrupt source register, I2CISR (described in section 8.10). Then the ISR can branch to the appropriate subroutine.

After the CPU reads I2CISR, the following events occur:

1) The flag for the source interrupt is cleared in I2CSTR. *Exception*: The ARDY, ICRRDY, and ICXRDY bits in I2CSTR are not cleared when I2CISR is read. To clear one of these bits, write a 1 to the corresponding bit.

2) The arbiter determines which of the remaining interrupt requests has the highest priority, writes the code for that interrupt to I2CISR, and forwards the interrupt request to the CPU.
## Table 3. Descriptions of the I2C Interrupt Requests

<table>
<thead>
<tr>
<th>I2C Interrupt Request</th>
<th>Interrupt Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>XRDYINT</td>
<td>Transmit ready condition: The data transmit register (I2CDXR) is ready to accept new data because the previous data has been copied from I2CDXR to the transmit shift register (I2CXSR). As an alternative to using XRDYINT, the CPU can poll the ICXRDY bit of I2CSTR (described in section 8.3).</td>
</tr>
<tr>
<td>RRDYINT</td>
<td>Receive ready condition: The data receive register (I2CDRR) is ready to be read because data has been copied from the receive shift register (I2CRSR) to I2CDRR. As an alternative to using RRDYINT, the CPU can poll the ICRRDY bit of I2CSTR (described in section 8.3).</td>
</tr>
<tr>
<td>ARDYINT</td>
<td>Register-access ready condition: The I2C module registers are ready to be accessed because the previously programmed address, data, and command values have been used. The specific events that generate ARDYINT are the same events that set the ARDY bit of I2CSTR (described in section 8.3). As an alternative to using ARDYINT, the CPU can poll the ARDY bit.</td>
</tr>
<tr>
<td>NACKINT</td>
<td>No-acknowledgement condition: The I2C module is configured as a master-transmitter and did not receive acknowledgement from the slave-receiver. As an alternative to using NACKINT, the CPU can poll the NACK bit of I2CSTR (described in section 8.3).</td>
</tr>
<tr>
<td>ALINT</td>
<td>Arbitration-lost condition: The I2C module has lost an arbitration contest with another master-transmitter. As an alternative to using ALINT, the CPU can poll the AL bit of I2CSTR (described in section 8.3).</td>
</tr>
</tbody>
</table>
5 EDMA Events Generated by the I2C Module

For the EDMA controller to handle transmit and receive data, the I2C module generates the following two EDMA events. Activity in EDMA channels can be synchronized to these events.

- Receive event (REVT): When receive data has been copied from the receive shift register (I2CRSR) to the data receive register (I2CDRR), the I2C module sends an REVT signal to the EDMA controller. In response, the EDMA controller can read the data from I2CDRR.

- Transmit event (XEVT): When transmit data has been copied from the data transmit register (I2CDXR) to the transmit shift register (I2CXSR), the I2C module sends an XEVT signal to the EDMA controller. In response, the EDMA controller can write the next transmit data value to I2CDXR.

6 Resetting/Disabling the I2C Module

The I2C module can be reset/disabled in two ways:

- Write 0 to the I2C reset bit (IRS) in the I2C mode register (I2CMDR). All status bits (in I2CSTR) are forced to their default values, and the I2C module remains disabled until IRS is changed to 1. The SDA and SCL pins are in the high-impedance state.
Initiate a DSP reset by driving the \texttt{RESET} pin low. The entire DSP is reset and is held in the reset state until you drive the pin high. When \texttt{RESET} is released, all I2C module registers are reset to their default values. The IRS bit is forced to 0, which resets the I2C module. The I2C module stays in the reset state until you write 1 to IRS.

IRS must be 0 while you configure/reconfigure the I2C module. Forcing IRS to 0 can be used to save power and to clear error conditions.
7 Programming Guide

7.1 Main Program

7.1.1 State After Reset

1) Program the I2C prescaler register (I2CPSC) to obtain the desired prescaled module clock for the operation of the I2C module (see section 8.11). This value is to be calculated and is dependent on the CPU frequency.

2) Take the I2C module out of reset (IRS = 1).
   a) If using interrupt for transmit/receive data, enable the appropriate interrupt in I2CIER.
   b) If using the EDMA controller for transmit/receive data, enable the EDMA and program the EDMA controller.

7.1.2 Initialization Procedure

Configure the I2C mode register (I2CMDR) (see section 8.9).

7.1.3 Program Clock Control Registers (I2CCLKL and I2CCLKH)

Program the I2C prescaled module clock (I2CCLKL and I2CCLKH) to obtain the desired SCL clock timing (see section 8.4). These values are to be calculated and are dependent on the CPU frequency.

7.1.4 Configure Address Registers

1) Configure its own address register (I2COAR) (see section 8.1)

2) Configure the slave address register (I2CSAR) (see section 8.7)

7.1.5 Program Transmit Data Register (I2CDXR)

If in master-transmitter mode, program the data transmit register (I2CDXR) (see section 8.8).

7.1.6 Configure Status and Mode Register (I2CSTR)

Poll the bus busy (BB) bit in the I2C status register (I2CSTR), if it is cleared to 0 (bus not busy), configure START/STOP condition to initiate a transfer (see section 8.3).

7.1.7 Poll Receive Data

Poll the receive-data-ready interrupt flag bit (ICRRDY) in the I2C status register (I2CSTR), use the RRDY interrupt, or use the EDMA controller to read the receive data in the data receive register (I2CDRR).
7.1.8 Poll Transmit Data

Poll the transmit data ready interrupt flag bit (ICXRDY) in the I2C status register (I2CSTR), use the XRDY interrupt, or use the EDMA controller to write data into the data transmit register (I2CDXR).

7.2 Interrupt Subroutines

1) Test for arbitration lost and resolve accordingly.
2) Test for no-acknowledge and resolve accordingly.
3) Test for register access ready and resolve accordingly.
4) Test for receive data and resolve accordingly.
5) Test for transmit data and resolve accordingly.

7.3 Flow Diagrams

Figure 14. Setup Procedure

```
Start
Write I2COAR
Write I2CIER
Write I2CCLKL
Write I2CCLKH
Use Repeat Mode (RM=1)?
No
Write I2CCNT
Yes
Write I2CSAR
End
```
7.3.1 Bit Polling Methods to Control Data Flow

Figure 15. Using Bit Polling for Master-Transmitter Operation, Repeat Mode (RM = 1)

1. Set appropriate values to every bit of I2CMDR. IRS bit should be set to 1 to take the I2C module out of reset condition. Setting IRS=1 and setting other mode bits can be done simultaneously.

2. Because RM=1, hardware counter does not run. Thus, software counter counts the number of the required transfer.

3. The I2C module goes into slave-receiver mode.

[EXPECTED COMMAND]
At the beginning, (STT,STP)=(1,0)
In the middle, (STT,STP)=(0,0)
At the end, (STT,STP)=(0,1)

[EXPECTED I2CIER]
i2CIER=00000b

Bus activities:
- START is generated
- Slave address is sent
- New START is generated
- STOP is generated
- DATA is sent
- DATA is sent
Figure 16. Using Bit Polling for Master-Transmitter Operation, Nonrepeat Mode (RM = 0)

Start

Read I2CSTR

Is bus free (BB=0)?

No

Read I2CSTR

Yes

Is ACK returned (NACK=0)?

No

STT and STP are cleared to 0 by hardware

Write I2CMDR with 2E20h

Set appropriate values to every bit of I2CMDR.
IRS bit should be set to 1 to take the I2C module out of reset condition. Setting IRS=1 and setting other mode bits can be done simultaneously.

[EXPECTED COMMAND]
At the beginning,
(STT,STP)=(1,0), (1,1)
In the middle,
(STT,STP)=(0,0), (0,1)
At the end,
(STT,STP)=(0,1)

[EXPECTED I2CIER]
I2CIER=00000b

Yes

Can update the MMRs (ARDY=1)?

No

Is send data being requested (ICXRDY=1)?

No

Yes

Write I2CDXR

End

The I2C module goes into slave-receiver mode

Yes

STT=1 (new start)?

No

STP=1?

Yes

The I2C module goes into slave-receiver mode

No

Reprogram the MMRs

Yes

Can update the MMRs (ARDY=1)?

No

Yes

Write I2CDXR
Figure 17. Using Bit Polling for Master-Receiver Operation, Repeat Mode (RM = 1), Fixed Number of Data Words

1. Set appropriate values to every bit of I2CMDR. IRS bit should be set to 1 to take the I2C module out of reset condition. Setting IRS=1 and setting other mode bits can be done simultaneously.

2. Because RM=1, hardware counter does not run. Thus, software counter counts the number of the required transfer.

3. The I2C module goes into slave-receiver mode.

4. Set STP=1

[EXPECTED COMMAND]
At the beginning, (STT,STP)=(1,0)
In the middle, (STT,STP)=(0,0)
At the end, (STT,STP)=(0,1)

[EXPECTED I2CIER]
I2CIER=00000b
Figure 18. Using Bit Polling for Master-Receiver Operation, Repeat Mode (RM = 1), Variable (Data-Dependent) Number of Data Words

---

1. Set appropriate values to every bit of I2CMDR. IRS bit should be set to 1 to take the I2C module out of reset condition. Setting IRS=1 and setting other mode bits can be done simultaneously.

2. Dummy read. The contents of this read data has no meaning.

3. The I2C module goes into slave-receiver mode.

[EXPECTED COMMAND]
At the beginning, (STT,STP)=(1,0)
In the middle, (STT,STP)=(0,0)
At the end, (STT,STP)=(0,1)

[EXPECTED I2CIER]
I2CIER=00000b
Figure 19. Using Bit Polling for Master-Receiver Operation, Nonrepeat Mode (RM = 0)

Start

Read I2CSTR

Is bus free (BB=0)?

No

Write I2CMDR with 2C20h

Read I2CSTR

Is ACK returned (NACK=0)?

No

STT and STP are cleared to 0 by hardware

Yes

Can update the MMRs (ARDY=1)?

No

Is received data in I2CDRR (ICRRDY=1)?

No

Reprogram the MMRs

No

STT=1 (new start)?

Yes

Read I2CDRR

No

STP=1?

Yes

End

The last received data is still in I2CDRR and it has not been read.

The I2C module goes into slave-receiver mode

Set appropriate values to every bit of I2CMDR. IRS bit should be set to 1 to take the I2C module out of reset condition. Setting IRS=1 and setting other mode bits can be done simultaneously.

[EXPECTED COMMAND]
At the beginning, (STT,STP)=(1,0), (1,1)
In the middle, (STT,STP)=(0,0), (0,1)
At the end, (STT,STP)=(0,1)

[EXPECTED I2CIER]
I2CIER=00000b
Figure 20. Using Bit Polling for Slave-Transmitter/Receiver Operation, Repeat Mode (RM = 1)

[EXPECTED I2CIER]
I2CIER=00000b

Start

Read I2CSTR

Write data (ICXRDY=1)

No

Write I2CDXR

Yes

Read data (ICRRDY=1)

No

Read I2CDRR

Yes
### 7.3.2 Interrupts Methods to Control Data Flow

**Figure 21.** Using Interrupts for Master-Transmitter Operation, Nonrepeat Mode (RM = 0)

Start

Read I2CSTR

Is bus free (BB=0)?

- No
  - Write I2CMDR with 2E20h
  - [EXPECTED COMMAND]
    - At the beginning, (STT,STP)=(1,0), (1,1)
    - In the middle, (STT,STP)=(0,0), (0,1)
    - At the end, (STT,STP)=(0,1)
  - [EXPECTED I2CIER]
    - I2CIER=1111b

- Yes
  - Set appropriate values to every bit of I2CMDR. IRS bit should be set to 1 to take the I2C module out of reset condition. Setting IRS=1 and setting other mode bits can be done simultaneously.

Is interrupt received?

- No
  - Read I2CISR
  - Is ACK returned (NACK=0)?
    - Yes
      - STT and STP are cleared to 0 by hardware
    - No
      - Reprogram the MMRs

- Yes
  - Are n words transferred (ARDY=1)?
    - Yes
      - Write I2CDXR
    - No
      - Is send data being requested (ICXRDY=1)

  - Is send data being requested (ICXRDY=1)?
    - Yes
      - The I2C module goes into slave-receiver mode
    - No
      - STT=1 (new start)

End

[EXPECTED COMMAND]

- At the beginning, (STT,STP)=(1,0), (1,1)
- In the middle, (STT,STP)=(0,0), (0,1)
- At the end, (STT,STP)=(0,1)

[EXPECTED I2CIER]

- I2CIER=1111b
Figure 22. Using Interrupts for Master-Receiver Operation, Nonrepeat Mode (RM = 0)

Start
- Read I2CSTR

Is bus free (BB=0)?

Yes
- Write I2CMDR with 2C20h

No
- Set appropriate values to every bit of I2CMDR. IRS bit should be set to 1 to take the I2C module out of reset condition. Setting IRS=1 and setting other mode bits can be done simultaneously.

[EXPECTED COMMAND]
- At the beginning, (STT,STP)=(1,0), (1,1)
- In the middle, (STT,STP)=(0,0), (0,1)
- At the end, (STT,STP)=(0,1)

[EXPECTED I2CIER]
- I2CIER=11111b

Is interrupt received?

Yes
- Read I2CISR

No
- Is ACK returned (NACK=0)?

Yes
- Are n words transferred (ARDY=1)?

Yes
- Reprogram the MMRs

No
- STT and STP are cleared to 0 by hardware

Is received data in I2CDRR (ICRRDY=1)?

Yes
- Read I2CDRR

No
- STT=1 (new start)?

Yes
- STP=1?

Yes
- The I2C module goes into slave-receiver mode

No
- No

End
Figure 23. Using Interrupts for Master-Transmitter/Receiver Operation, Repeat Mode (RM = 1)

Start

- Read I2CSTR

- Is bus free (BB=0)?
  - No
  - Write I2CMDR with 25A0h
  - Set appropriate values to every bit of I2CMDR. IRS bit should be set to 1 to take the I2C module out of reset condition. Setting IRS=1 and setting other mode bits can be done simultaneously.

- Yes
  - [EXPECTED COMMAND]
    - At the beginning, (STT,STP)=(1,0)
    - In the middle, (STT,STP)=(0,0)
    - At the end, (STT,STP)=(0,1)

  - [EXPECTED I2CIER]
    - I2CIER=00111b
    - I2CIER=11111b after arbitration lost

- Is interrupt received?
  - No
  - [EXPECTED COMMAND]
    - At the beginning, (STT,STP)=(1,0)
    - In the middle, (STT,STP)=(0,0)
    - At the end, (STT,STP)=(0,1)

    - [EXPECTED I2CIER]
      - I2CIER=00111b
      - I2CIER=11111b after arbitration lost

  - Yes
    - Read I2CISR
    - Is ACK returned (NACK=0)?
      - Yes, STT and STP are cleared to 0 by hardware
        - Reprogram the MMRs
          - STT=1 (new start) ?
            - Yes
            - The I2C module goes into slave-receiver mode
            - End
          - No
            - STP ?
              - Yes
              - The I2C module goes into slave-receiver mode
              - End
            - No
              - No

    - No
      - Are n words transferred (ARDY=1)?
        - Yes
          - Read I2CDRR
          - Write I2CDXR
        - No
          - Write I2CDXR
Figure 24. Using Interrupts for Slave-Transmitter/Receiver Operation, Repeat Mode (RM = 1)

Start

Is interrupt received?

Yes

Read I2CISR

Write I2CDXR (I2CISR=101b)

No

Write?

Yes

No

Read I2CDRR (I2CISR=100b)

[EXPECTED I2CIER]
I2CIER=11111b
7.3.3 EDMA Event Methods to Control Data Flow

Figure 25. Using EDMA Events for Master-Transmitter Operation, Nonrepeat Mode (RM = 0)

Start

Read I2CSTR

Is bus free (BB=0)?

No

Write I2CMDR with 2E20h

Yes

[EXPECTED COMMAND]
At the beginning, (STT,STP)=(1,0), (1,1).
In the middle, (STT,STP)=(0,0), (0,1).
At the end, (STT,STP)=(0,1).

[EXPECTED I2CIER]
I2CIER=00111b

Set appropriate values to every bit of I2CMDR. IRS bit should be set to 1 to take the I2C module out of reset condition. Setting IRS=1 and setting other mode bits can be done simultaneously.

Is I2C interrupt received?

No

Yes

Read I2CISR

Is ACK returned (NACK=0)?

No

Yes

STT and STP are cleared to 0 by hardware

Are n words transferred (ARDY=1)?

No

Yes

Reprogram the MMRs

STT=1 (new start)?

No

Yes

End

No

STP=1?

Yes

The I2C module goes into slave-receiver mode

Yes

Take necessary actions

No

Is EDMA interrupt received?
Figure 26.  Using EDMA Events for Master-Receiver Operation, Nonrepeat Mode (RM = 0)

Start

Read I2CSTR

No

Is bus free (BB=0)?

Yes

Write I2CMDR with 2C20h

[EXPECTED COMMAND]
At the beginning, (STT,STP)=(1,0), (1,1)
In the middle, (STT,STP)=(0,0), (0,1)
At the end, (STT,STP)=(0,1)

[EXPECTED I2CIER]
I2CIER=00111b

Set appropriate values to every bit of I2CMDR. IRS bit should be set to 1 to take the I2C module out of reset condition. Setting IRS=1 and setting other mode bits can be done simultaneously.

In I2C interrupt received?

Yes

Read I2CISR

Are n words transferred (ARDY=1)?

No

STT and STP are cleared to 0 by hardware

Yes

Reprogram the MMRs

STT=1 (new start)?

No

End

The I2C module goes into slave-receiver mode

Yes

Take necessary actions

Is EDMA interrupt received?

Yes

Is ACK returned (NACK=0)?

No

No

Yes

No

Yes

No

Yes
8 Registers

Table 4 lists the I2C module registers. All but the data registers (I2CRSR and I2CXSR) are accessible to the CPU. See the device-specific datasheet for the memory address of these registers.

Table 4. I2C Module Registers

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Register Name</th>
<th>Address Offset (hex)</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2COAR</td>
<td>I2C own address register</td>
<td>00</td>
<td>8.1</td>
</tr>
<tr>
<td>I2CIER</td>
<td>I2C interrupt enable register</td>
<td>04</td>
<td>8.2</td>
</tr>
<tr>
<td>I2CSTR</td>
<td>I2C status register</td>
<td>08</td>
<td>8.3</td>
</tr>
<tr>
<td>I2CCLKL</td>
<td>I2C clock low-time divider register</td>
<td>0C</td>
<td>8.4</td>
</tr>
<tr>
<td>I2CCLKH</td>
<td>I2C clock high-time divider register</td>
<td>10</td>
<td>8.4</td>
</tr>
<tr>
<td>I2CCNT</td>
<td>I2C data count register</td>
<td>14</td>
<td>8.5</td>
</tr>
<tr>
<td>I2CDRR</td>
<td>I2C data receive register</td>
<td>18</td>
<td>8.6</td>
</tr>
<tr>
<td>I2CSAR</td>
<td>I2C slave address register</td>
<td>1C</td>
<td>8.7</td>
</tr>
<tr>
<td>I2CDXR</td>
<td>I2C data transmit register</td>
<td>20</td>
<td>8.8</td>
</tr>
<tr>
<td>I2CMDR</td>
<td>I2C mode register</td>
<td>24</td>
<td>8.9</td>
</tr>
<tr>
<td>I2CISR</td>
<td>I2C interrupt source register</td>
<td>28</td>
<td>8.10</td>
</tr>
<tr>
<td>I2CPSC</td>
<td>I2C prescaler register</td>
<td>30</td>
<td>8.11</td>
</tr>
<tr>
<td>I2CPID1</td>
<td>I2C peripheral identification register 1</td>
<td>—</td>
<td>8.12</td>
</tr>
<tr>
<td>I2CPID2</td>
<td>I2C peripheral identification register 2</td>
<td>—</td>
<td>8.12</td>
</tr>
<tr>
<td>I2CRSR</td>
<td>I2C receive shift register (not accessible to the CPU or EDMA)</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>I2CXSR</td>
<td>I2C transmit shift register (not accessible to the CPU or EDMA)</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
8.1 I2C Own Address Register (I2COAR)

The I2C own address register (I2COAR) is a 32-bit register mapped used to specify its own slave address, which distinguishes it from other slaves connected to the I2C-bus. If the 7-bit addressing mode is selected (XA = 0 in I2CMDR), only bits 6–0 are used; bits 9–7 are ignored. The I2COAR is shown in Figure 27 and described in Table 5.

**Figure 27. I2C Own Address Register (I2COAR)**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31–10</td>
<td>Reserved</td>
<td>–</td>
<td>0</td>
<td>These reserved bit locations are always read as zeros. A value written to this field has no effect.</td>
</tr>
<tr>
<td>9–0</td>
<td>A</td>
<td>OF(value)</td>
<td></td>
<td>In 7-bit addressing mode (XA = 0 in I2CMDR): Bits 6–0 provide the 7-bit slave address of the I2C module. Bits 9–7 are ignored.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00h–7Fh</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>000h–3FFh</td>
<td></td>
<td>In 10-bit addressing mode (XA = 1 in I2CMDR): Bits 9–0 provide the 10-bit slave address of the I2C module.</td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2COAR_A_symval

Table 5. I2C Own Address Register (I2COAR) Field Descriptions
8.2 I2C Interrupt Enable Register (I2CIER)

The I2C interrupt enable register (I2CIER) is used by the CPU to individually enable or disable I2C interrupt requests. The I2CIER is shown in Figure 28 and described Table 6.

Figure 28. I2C Interrupt Enable Register (I2CIER)

<table>
<thead>
<tr>
<th>31–5</th>
<th>16</th>
<th>15</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R-0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
</tr>
</tbody>
</table>

Legend: R = Read only; R/W = Read/write; -n = value after reset

Table 6. I2C Interrupt Enable Register (I2CIER) Field Descriptions

<table>
<thead>
<tr>
<th>Bit</th>
<th>field†</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31–5</td>
<td>Reserved</td>
<td>–</td>
<td>0</td>
<td>These reserved bit locations are always read as zeros. A value written to this field has no effect.</td>
</tr>
<tr>
<td>4</td>
<td>ICXRDY</td>
<td></td>
<td></td>
<td>Transmit-data-ready interrupt enable bit.</td>
</tr>
<tr>
<td></td>
<td>MSK</td>
<td></td>
<td>0</td>
<td>Interrupt request is disabled.</td>
</tr>
<tr>
<td></td>
<td>UNMSK</td>
<td></td>
<td>1</td>
<td>Interrupt request is enabled.</td>
</tr>
<tr>
<td>3</td>
<td>ICRRDY</td>
<td></td>
<td></td>
<td>Receive-data-ready interrupt enable bit.</td>
</tr>
<tr>
<td></td>
<td>MSK</td>
<td></td>
<td>0</td>
<td>Interrupt request is disabled.</td>
</tr>
<tr>
<td></td>
<td>UNMSK</td>
<td></td>
<td>1</td>
<td>Interrupt request is enabled.</td>
</tr>
<tr>
<td>2</td>
<td>ARDY</td>
<td></td>
<td></td>
<td>Register-access-ready interrupt enable bit.</td>
</tr>
<tr>
<td></td>
<td>MSK</td>
<td></td>
<td>0</td>
<td>Interrupt request is disabled.</td>
</tr>
<tr>
<td></td>
<td>UNMSK</td>
<td></td>
<td>1</td>
<td>Interrupt request is enabled.</td>
</tr>
<tr>
<td>1</td>
<td>NACK</td>
<td></td>
<td></td>
<td>No-acknowledgement interrupt enable bit.</td>
</tr>
<tr>
<td></td>
<td>MSK</td>
<td></td>
<td>0</td>
<td>Interrupt request is disabled.</td>
</tr>
<tr>
<td></td>
<td>UNMSK</td>
<td></td>
<td>1</td>
<td>Interrupt request is enabled.</td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2CIER_field_symval
### Table 6. I2C Interrupt Enable Register (I2CIER) Field Descriptions (Continued)

<table>
<thead>
<tr>
<th>Bit</th>
<th>field†</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AL</td>
<td></td>
<td>0x0</td>
<td>Arbitration-lost interrupt enable bit</td>
</tr>
</tbody>
</table>

|     |         |         | 0     | Interrupt request is disabled. |
|     |         |         | 1     | Interrupt request is enabled. |

† For CSL C macro implementation, use the notation `I2C_I2CIER_field_symval`

---

### 8.3 I2C Status Register (I2CSTR)

The I2C status register (I2CSTR) is used by the CPU to determine which interrupt has occurred and to read status information. The I2CSTR is shown in Figure 29 and described in Table 7.

### Figure 29. I2C Status Register (I2CSTR)

31 16

<table>
<thead>
<tr>
<th>31</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R-0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend: R = Read; W1C = Write 1 to clear (writing 0 has no effect); *n* = value after reset
## Table 7. I2C Status Register (I2CSTR) Field Descriptions

<table>
<thead>
<tr>
<th>Bit</th>
<th>field†</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31–14</td>
<td>Reserved</td>
<td>–</td>
<td>0</td>
<td>These reserved bit locations are always read as zeros. A value written to this field has no effect.</td>
</tr>
<tr>
<td>13</td>
<td>NACKSNT</td>
<td></td>
<td></td>
<td>NACK sent bit is used when the I2C module is in the receiver mode. One instance in which NACKSNT is affected is when the NACK mode is used (see the description for NACKMOD in section 8.9).</td>
</tr>
<tr>
<td>NONE</td>
<td>0</td>
<td>NACK is not sent. NACKSNT bit is cleared by any one of the following events:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>□ It is manually cleared. To clear this bit, write a 1 to it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>□ The I2C module is reset (either when 0 is written to the IRS bit of I2CMDR or when the whole DSP is reset).</td>
</tr>
<tr>
<td>INT</td>
<td>1</td>
<td>NACK is sent: A no-acknowledge bit was sent during the acknowledge cycle on the I2C-bus.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>BB</td>
<td></td>
<td></td>
<td>Bus busy bit. BB indicates whether the I2C-bus is busy or is free for another data transfer.</td>
</tr>
<tr>
<td>NONE</td>
<td>0</td>
<td>Bus is free. BB is cleared by any one of the following events:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>□ The I2C module receives or transmits a STOP bit (bus free).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>□ BB is manually cleared. To clear this bit, write a 1 to it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>□ The I2C module is reset.</td>
</tr>
<tr>
<td>INT</td>
<td>1</td>
<td>Bus is busy: The I2C module has received or transmitted a START bit on the bus.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>RSFULL</td>
<td></td>
<td></td>
<td>Receive shift register full bit. RSFULL indicates an overrun condition during reception. Overrun occurs when the receive shift register (I2CRSR) is full with new data but the previous data has not been read from the data receive register (I2CDRR). The new data will not be copied to I2CDRR until the previous data is read. As new bits arrive from the SDA pin, they overwrite the bits in I2CRSR.</td>
</tr>
<tr>
<td>NONE</td>
<td>0</td>
<td>No overrun is detected. RSFULL is cleared by any one of the following events:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>□ I2CDRR is read.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>□ The I2C module is reset.</td>
</tr>
<tr>
<td>INT</td>
<td>1</td>
<td>Overrun is detected.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2CSTR_field_symval
### Table 7. I2C Status Register (I2CSTR) Field Descriptions (Continued)

<table>
<thead>
<tr>
<th>Bit</th>
<th>field†</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>XSMT</td>
<td></td>
<td>0</td>
<td>Transmit shift register empty bit. XSMT indicates that the transmitter has experienced underflow. Underflow occurs when the transmit shift register (I2CXSR) is empty but the data transmit register (I2CDXR) has not been loaded since the last I2CDXR-to-I2CXSR transfer. The next I2CDXR-to-I2CXSR transfer will not occur until new data is in I2CDXR. If new data is not transferred in time, the previous data may be re-transmitted on the SDA pin.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Underflow is detected.</td>
</tr>
<tr>
<td>9</td>
<td>AAS</td>
<td></td>
<td>0</td>
<td>Addressed-as-slave bit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>The AAS bit has been cleared by a repeated START condition or by a STOP condition.</td>
</tr>
<tr>
<td>8</td>
<td>AD0</td>
<td></td>
<td>0</td>
<td>Address 0 bit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>An address of all zeros (general call) is detected.</td>
</tr>
<tr>
<td>7−5</td>
<td>Reserved</td>
<td>–</td>
<td>0</td>
<td>These reserved bit locations are always read as zeros. A value written to this field has no effect.</td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2CSTR_field_symval
### Table 7. I2C Status Register (I2CSTR) Field Descriptions (Continued)

<table>
<thead>
<tr>
<th>Bit</th>
<th>field†</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>ICXRDY</td>
<td></td>
<td>0</td>
<td>I2CDXR is not ready. ICXRDY is cleared by one of the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Data is written to I2CDXR.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ICXRDY is manually cleared. To clear this bit, write a 1 to it.</td>
</tr>
<tr>
<td>3</td>
<td>ICRRDY</td>
<td></td>
<td>0</td>
<td>I2CDRR is not ready. ICRRDY is cleared by any one of the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I2CDRR is read.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ICRRDY is manually cleared. To clear this bit, write a 1 to it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The I2C module is reset.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>I2CDRR is ready: Data has been copied from I2CRSR to I2CDRR.</td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2CSTR_field_symval
### Table 7. I2C Status Register (I2CSTR) Field Descriptions (Continued)

<table>
<thead>
<tr>
<th>Bit</th>
<th>field†</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>ARDY</td>
<td></td>
<td>0</td>
<td>NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The registers are not ready to be accessed. ARDY is cleared by any one of the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- The I2C module starts using the current register contents.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- ARDY is manually cleared. To clear this bit, write a 1 to it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- The I2C module is reset.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>INT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The registers are ready to be accessed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>In the nonrepeat mode (RM = 0 in I2CMDR): If STP = 0 in I2CMDR, the ARDY bit is set when the internal data counter counts down to 0. If STP = 1, ARDY is not affected (instead, the I2C module generates a STOP condition when the counter reaches 0).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>In the repeat mode (RM = 1): ARDY is set at the end of each data word transmitted from I2CDXR.</td>
</tr>
<tr>
<td>1</td>
<td>NACK</td>
<td></td>
<td>0</td>
<td>NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ACK received/NACK is not received. This bit is cleared by any one of the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- An acknowledge bit (ACK) has been sent by the receiver.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NACK is manually cleared. To clear this bit, write a 1 to it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- The CPU reads the interrupt source register (I2CISR) when the register contains the code for a NACK interrupt.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- The I2C module is reset.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>INT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NACK bit is received. The hardware detects that a no-acknowledge (NACK) bit has been received.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Note: While the I2C module performs a general call transfer, NACK is 1, even if one or more slaves send acknowledgement.</td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2CSTR_field_symval
Arbitration-lost interrupt flag bit (only applicable when the I2C module is a master-transmitter). AL primarily indicates when the I2C module has lost an arbitration contest with another master-transmitter. The CPU can poll AL or use the AL interrupt request (see section 4).

NONE: 0 Arbitration is not lost. AL is cleared by any one of the following events:
- AL is manually cleared. To clear this bit, write a 1 to it.
- The CPU reads the interrupt source register (I2CISR) when the register contains the code for an AL interrupt.
- The I2C module is reset.

INT: 1 Arbitration is lost. AL is set by any one of the following events:
- The I2C module senses that it has lost an arbitration with two or more competing transmitters that started a transmission almost simultaneously.
- The I2C module attempts to start a transfer while the BB (bus busy) bit is set to 1.

When AL becomes 1, the MST and STP bits of I2CMDR are cleared, and the I2C module becomes a slave-receiver.

† For CSL C macro implementation, use the notation I2C_I2CSTR_field_symval
8.4 I2C Clock Divider Registers (I2CCLKL and I2CCLKH)

When the I2C is a master, the prescaled module clock is divided down for use as the master clock on the SCL pin. As shown in Figure 30, the shape of the master clock depends on two divide-down values: ICCL and ICCH.

The frequency of the master clock is calculated as:

\[
\text{master clock frequency} = \frac{\text{prescaled module clock frequency}}{(\text{ICCL} + d) + (\text{ICCH} + d)}
\]

Figure 30. Roles of the Clock Divide-Down Values (ICCL and ICCH)

\[(A)\ T_{\text{mod}} = \text{prescaled module clock period} = \frac{1}{\text{prescaled module clock frequency}}; \ d = 5, 6, \text{ or } 7.\]
Figure 31.  I2C Clock Low-Time Divider Register (I2CCLKL)

31 16

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31−16</td>
<td>Reserved</td>
<td>–</td>
<td>0</td>
<td>These reserved bit locations are always read as zeros. A value written to this field has no effect.</td>
</tr>
<tr>
<td>15−0</td>
<td>ICCL</td>
<td>OF(value)</td>
<td>0</td>
<td>Not supported. Note that 0 is the default value; therefore, this field must be set to a value of 1 or greater.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1−FFFFh</td>
<td>Clock low-time divide-down value of 2–65536. The period of the prescaled module clock is multiplied by (ICCL + d) to produce the low-time duration of the master clock on the SCL pin.</td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2CCLKL_ICCL_symval
**Figure 32. I2C Clock High-Time Divider Register (I2CCLKH)**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31−16</td>
<td>Reserved</td>
<td>–</td>
<td>0</td>
<td>These reserved bit locations are always read as zeros. A value written to this field has no effect.</td>
</tr>
<tr>
<td>15−0</td>
<td>ICCH</td>
<td>OF(value)</td>
<td>0</td>
<td>Not supported. Note that 0 is the default value; therefore, this field must be set to a value of 1 or greater.</td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2CCLKH_ICCH_symval

Legend:  R = Read only; R/W = Read/write; -n = value after reset
8.5 I2C Data Count Register (I2CCNT)

The I2C data count register (I2CCNT) is used to indicate how many data words to transfer when the I2C module is configured as a master-transmitter (MST = 1 and TRX = 1 in I2CMDR) and the repeat mode is off (RM = 0 in I2CMDR). In the repeat mode (RM = 1), I2CCNT is not used. The I2CCNT is shown in Figure 33 and described in Table 10.

The value written to I2CCNT is copied to an internal data counter. The internal data counter is decremented by 1 for each data word transferred (I2CCNT remains unchanged). If a STOP condition is requested (STP = 1 in I2CMDR), the I2C module terminates the transfer with a STOP condition when the countdown is complete (that is, when the last data word has been transferred).

Figure 33. I2C Data Count Register (I2CCNT)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31−16</td>
<td>Reserved</td>
<td>–</td>
<td>0</td>
<td>These reserved bit locations are always read as zeros. A value written to this field has no effect.</td>
</tr>
<tr>
<td>15−0</td>
<td>ICDC</td>
<td>OF(value)</td>
<td></td>
<td>Data count value. ICDC indicates the number of data words to transfer in the nonrepeat mode (RM = 0 in I2CMDR). The value in I2CCNT is a don’t care when the RM bit in I2CMDR is set to 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0000h</td>
<td>The start value loaded to the internal data counter is 65536.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0001h–FFFFh</td>
<td>The start value loaded to internal data counter is 1–65535.</td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2CCNT_ICDC_symval
8.6 I2C Data Receive Register (I2CDRR)

The I2C data receive register (I2CDRR) is used by the DSP to read the receive data. The I2CDRR can receive a data value of up to 8 bits; data values with fewer than 8 bits are right-aligned in the D bits and the remaining D bits are undefined. The number of data bits is selected by the bit count bits (BC) of I2CMDR. The I2C receive shift register (I2CRSR) shifts in the received data from the SDA pin. Once data is complete, the I2C module copies the contents of I2CRSR into I2CDRR. The CPU and the EDMA controller cannot access I2CRSR. The I2CDRR is shown in Figure 34 and described in Table 11.

Figure 34. I2C Data Receive Register (I2CDRR)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31–8</td>
<td>Reserved</td>
<td>_</td>
<td>0</td>
<td>These reserved bit locations are always read as zeros. A value written to this field has no effect.</td>
</tr>
<tr>
<td>7–0</td>
<td>D</td>
<td>OF(value)</td>
<td>00h–FFh</td>
<td>Receive data.</td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2CDRR_D_symval
8.7 I2C Slave Address Register (I2CSAR)

The I2C slave address register (I2CSAR) contains a 7-bit or 10-bit slave address. When the I2C module is not using the free data format (FDF = 0 in I2CMDR), it uses this address to initiate data transfers with a slave or slaves. When the address is nonzero, the address is for a particular slave. When the address is 0, the address is a general call to all slaves. If the 7-bit addressing mode is selected (XA = 0 in I2CMDR), only bits 6–0 of I2CSAR are used; bits 9–7 are ignored. The I2CSAR is shown in Figure 35 and described in Table 12.

Figure 35. I2C Slave Address Register (I2CSAR)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31–10</td>
<td>Reserved</td>
<td>–</td>
<td>0</td>
<td>These reserved bit locations are always read as zeros. A value written to this field has no effect.</td>
</tr>
<tr>
<td>9–0</td>
<td>A</td>
<td>OF(value)</td>
<td>00h–7Fh</td>
<td>In 7-bit addressing mode (XA = 0 in I2CMDR): Bits 6–0 provide the 7-bit slave address that the I2C module transmits when it is in the master-transmitter mode. Bits 9–7 are ignored. In 10-bit addressing mode (XA = 1 in I2CMDR):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>000h–3FFh</td>
<td>Bits 9–0 provide the 10-bit slave address that the I2C module transmits when it is in the master-transmitter mode.</td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2CSAR_A_symval

Table 12. I2C Slave Address Register (I2CSAR) Field Descriptions
8.8 I2C Data Transmit Register (I2CDXR)

The DSP writes transmit data to the I2C data transmit register (I2CDXR). The I2CDXR can accept a data value of up to 8 bits. When writing a data value with fewer than 8 bits, the DSP must make sure that the value is right-aligned in the D bits. The number of data bits is selected by the bit count bits (BC) of I2CMDR. Once data is written to I2CDXR, the I2C module copies the contents of I2CDXR into the I2C transmit shift register (I2CXSR). The I2CXSR shifts out the transmit data from the SDA pin. The CPU and the EDMA controller cannot access I2CXSR. The I2CDXR is shown in Figure 36 and described in Table 13.

Figure 36. I2C Data Transmit Register (I2CDXR)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31–8</td>
<td>Reserved</td>
<td>_</td>
<td>0</td>
<td>These reserved bit locations are always read as zeros. A value written to this field has no effect.</td>
</tr>
<tr>
<td>7–0</td>
<td>D</td>
<td>OF(value)</td>
<td>00h–FFh</td>
<td>Transmit data</td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2CDXR_D_symval
8.9 I2C Mode Register (I2CMDR)

The I2C mode register (I2CMDR) contains the control bits of the I2C module. The I2CMDR is shown in Figure 37 and described in Table 14.

Figure 37. I2C Mode Register (I2CMDR)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31−16</td>
<td>Reserved</td>
</tr>
<tr>
<td>15</td>
<td>NACKMOD</td>
</tr>
<tr>
<td>14</td>
<td>FREE</td>
</tr>
<tr>
<td>13</td>
<td>STT</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
</tr>
<tr>
<td>11</td>
<td>STP</td>
</tr>
<tr>
<td>10</td>
<td>MST</td>
</tr>
<tr>
<td>9</td>
<td>TRX</td>
</tr>
<tr>
<td>8</td>
<td>XA</td>
</tr>
<tr>
<td>7</td>
<td>RM</td>
</tr>
<tr>
<td>6</td>
<td>DLB</td>
</tr>
<tr>
<td>5</td>
<td>IRS</td>
</tr>
<tr>
<td>4</td>
<td>STB</td>
</tr>
<tr>
<td>3</td>
<td>FDF</td>
</tr>
<tr>
<td>2</td>
<td>BC</td>
</tr>
<tr>
<td>1</td>
<td>NACK</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table 14. I2C Mode Register (I2CMDR) Field Descriptions

<table>
<thead>
<tr>
<th>Bit</th>
<th>field†</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31−16</td>
<td>Reserved</td>
<td>–</td>
<td>0</td>
<td>These reserved bit locations are always read as zeros. A value written to this field has no effect.</td>
</tr>
<tr>
<td>15</td>
<td>NACKMOD</td>
<td></td>
<td>0</td>
<td>NACK mode bit is only applicable when the I2C module is acting as a receiver.</td>
</tr>
<tr>
<td>14</td>
<td>ACK</td>
<td>0</td>
<td></td>
<td>In slave-receiver mode: The I2C module sends an acknowledge (ACK) bit to the transmitter during each acknowledge cycle on the bus. The I2C module only sends a no-acknowledge (NACK) bit if you set the NACKMOD bit.</td>
</tr>
<tr>
<td>13</td>
<td>NACK</td>
<td>1</td>
<td></td>
<td>In either slave-receiver or master-receiver mode: The I2C module sends a NACK bit to the transmitter during the next acknowledge cycle on the bus. Once the NACK bit has been sent, NACKMOD is cleared.</td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2CMDR_field_symval
Table 14. I2C Mode Register (I2CMDR) Field Descriptions (Continued)

<table>
<thead>
<tr>
<th>Bit</th>
<th>field†</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>FREE</td>
<td></td>
<td>0x0</td>
<td>FREE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This emulation mode bit is used to determine what the state of the I2C module will be when a breakpoint is encountered in the high-level language debugger.</td>
</tr>
<tr>
<td></td>
<td>BSTOP</td>
<td>0</td>
<td></td>
<td>When I2C module is master:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If SCL is low when the breakpoint occurs, the I2C module stops immediately and keeps driving SCL low, whether the I2C module is the transmitter or the receiver. If SCL is high, the I2C module waits until SCL becomes low and then stops.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>When I2C module is slave:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A breakpoint forces the I2C module to stop when the current transmission/reception is complete.</td>
</tr>
<tr>
<td></td>
<td>RFREE</td>
<td>1</td>
<td></td>
<td>The I2C module runs free; that is, it continues to operate when a breakpoint occurs.</td>
</tr>
<tr>
<td>13</td>
<td>STT</td>
<td></td>
<td></td>
<td>START condition bit (only applicable when the I2C module is a master). The RM, STT, and STP bits determine when the I2C module starts and stops data transmissions (see Table 15). Note that the STT and STP bits can be used to terminate the repeat mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NONE</td>
<td>In the master mode, STT is automatically cleared after the START condition has been generated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>In the slave mode, if STT is 0, the I2C module does not monitor the bus for commands from a master. As a result, the I2C module performs no data transfers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>START</td>
<td>In the master mode, setting STT to 1 causes the I2C module to generate a START condition on the I2C-bus.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>In the slave mode, if STT is 1, the I2C module monitors the bus and transmits/receives data in response to commands from a master.</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
<td>–</td>
<td>0x0</td>
<td>This reserved bit location is always read as zero. A value written to this field has no effect.</td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2CMDR_field_symval
Table 14. I2C Mode Register (I2CMDR) Field Descriptions (Continued)

<table>
<thead>
<tr>
<th>Bit</th>
<th>field†</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>STP</td>
<td></td>
<td>STOP</td>
<td>STOP condition bit (only applicable when the I2C module is a master). In the master mode, the RM, STT, and STP bits determine when the I2C module starts and stops data transmissions (see Table 15). Note that the STT and STP bits can be used to terminate the repeat mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NONE</td>
<td>STP is automatically cleared after the STOP condition has been generated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>STOP</td>
<td>STP has been set by the DSP to generate a STOP condition when the internal data counter of the I2C module counts down to 0.</td>
</tr>
<tr>
<td>10</td>
<td>MST</td>
<td></td>
<td></td>
<td>Master mode bit. MST determines whether the I2C module is in the slave mode or the master mode. MST is automatically changed from 1 to 0 when the I2C master generates a STOP condition.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SLAVE</td>
<td>Slave mode. The I2C module is a slave and receives the serial clock from the master.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MASTER</td>
<td>Master mode. The I2C module is a master and generates the serial clock on the SCL pin.</td>
</tr>
<tr>
<td>9</td>
<td>TRX</td>
<td></td>
<td></td>
<td>Transmitter mode bit. When relevant, TRX selects whether the I2C module is in the transmitter mode or the receiver mode. Table 16 summarizes when TRX is used and when it is a don’t care.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RCV</td>
<td>Receiver mode. The I2C module is a receiver and receives data on the SDA pin.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>XMT</td>
<td>Transmitter mode. The I2C module is a transmitter and transmits data on the SDA pin.</td>
</tr>
<tr>
<td>8</td>
<td>XA</td>
<td></td>
<td></td>
<td>Expanded address enable bit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7BIT</td>
<td>7-bit addressing mode (normal address mode). The I2C module transmits 7-bit slave addresses (from bits 6–0 of I2CSAR), and its own slave address has 7 bits (bits 6–0 of I2COAR).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10BIT</td>
<td>10-bit addressing mode (expanded address mode). The I2C module transmits 10-bit slave addresses (from bits 9–0 of I2CSAR), and its own slave address has 10 bits (bits 9–0 of I2COAR).</td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2CMDR_field_symval
**Table 14. I2C Mode Register (I2CMDR) Field Descriptions (Continued)**

<table>
<thead>
<tr>
<th>Bit</th>
<th>field†</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>RM</td>
<td></td>
<td></td>
<td>Repeat mode bit (only applicable when the I2C module is a master-transmitter). The RM, STT, and STP bits determine when the I2C module starts and stops data transmissions (see Table 15).</td>
</tr>
<tr>
<td></td>
<td>NONE</td>
<td>0</td>
<td></td>
<td>Nonrepeat mode. The value in the data count register (I2CCNT) determines how many data words are received/transmitted by the I2C module.</td>
</tr>
<tr>
<td></td>
<td>REPEAD</td>
<td>1</td>
<td></td>
<td>Repeat mode. Data words are continuously received/transmitted by the I2C module until the STP bit is manually set to 1, regardless of the value in I2CCNT.</td>
</tr>
<tr>
<td>6</td>
<td>DLB</td>
<td></td>
<td></td>
<td>Digital loopback mode bit. This bit disables or enables the digital loopback mode of the I2C module. The effects of this bit are shown in Figure 38. Note that DLB in the free data format mode (DLB = 1 and FDF = 1) is not supported.</td>
</tr>
<tr>
<td></td>
<td>NONE</td>
<td>0</td>
<td></td>
<td>Digital loopback mode is disabled.</td>
</tr>
</tbody>
</table>
|     | LOOPBACK | 1      |       | Digital loopback mode is enabled. In this mode, the MST bit must be set to 1 and data transmitted out of I2CDXR is received in I2CDRR after n DSP cycles by an internal path, where:  
\[
n = ((I2C \text{ input clock frequency/prescaled module clock frequency}) \times 8)
\]  
The transmit clock is also the receive clock. The address transmitted on the SDA pin is the address in I2COAR. |
| 5   | IRS    |         |       | I2C module reset bit. |
|     | RST    | 0       |       | The I2C module is in reset/disabled. When this bit is cleared to 0, all status bits (in I2CSTR) are set to their default values. |
|     | NRST   | 1       |       | The I2C module is enabled. |

† For CSL C macro implementation, use the notation I2C_I2CMDR_field_symval
### Table 14. I2C Mode Register (I2CMDR) Field Descriptions (Continued)

<table>
<thead>
<tr>
<th>Bit</th>
<th>field†</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>STB</td>
<td></td>
<td></td>
<td>START byte mode bit. This bit is only applicable when the I2C module is a master. As described in version 2.1 of the Philips I2C-bus specification, the START byte can be used to help a slave that needs extra time to detect a START condition. When the I2C module is a slave, the I2C module ignores a START byte from a master, regardless of the value of the STB bit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NONE 0</td>
<td>The I2C module is not in the START byte mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SET 1</td>
<td>The I2C module is in the START byte mode. When you set the START condition bit (STT), the I2C module begins the transfer with more than just a START condition. Specifically, it generates:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1) A START condition</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2) A START byte (0000 0001b)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3) A dummy acknowledge clock pulse</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4) A repeated START condition</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The I2C module sends the slave address that is in I2CSAR.</td>
</tr>
<tr>
<td>3</td>
<td>FDF</td>
<td></td>
<td></td>
<td>Free data format mode bit. Note that DLB in the free data format mode (DLB = 1 and FDF = 1) is not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NONE 0</td>
<td>Free data format mode is disabled. Transfers use the 7-/10-bit addressing format selected by the XA bit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SET 1</td>
<td>Free data format mode is enabled. Transfers have the free data (no address) format described in section 3.5.</td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2CMDR_field_symval
Table 14. I2C Mode Register (I2CMDR) Field Descriptions (Continued)

<table>
<thead>
<tr>
<th>Bit field†</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2−0 BC</td>
<td>OF(value)</td>
<td></td>
<td>Bit count bits. BC defines the number of bits (1 to 8) in the next data word that is to be received or transmitted by the I2C module. The number of bits selected with BC must match the data size of the other device. Notice that when BC = 000b, a data word has 8 bits. If the bit count is less than 8, receive data is right aligned in the D bits of I2CDRR and the remaining D bits are undefined. Also, transmit data written to I2CDXR must be right aligned.</td>
</tr>
<tr>
<td>BIT8FDF</td>
<td>0</td>
<td>8</td>
<td>8 bits per data word</td>
</tr>
<tr>
<td>BIT1FDF</td>
<td>1h</td>
<td>1</td>
<td>1 bit per data word</td>
</tr>
<tr>
<td>BIT2FDF</td>
<td>2h</td>
<td>2</td>
<td>2 bits per data word</td>
</tr>
<tr>
<td>BIT3FDF</td>
<td>3h</td>
<td>3</td>
<td>3 bits per data word</td>
</tr>
<tr>
<td>BIT4FDF</td>
<td>4h</td>
<td>4</td>
<td>4 bits per data word</td>
</tr>
<tr>
<td>BIT5FDF</td>
<td>5h</td>
<td>5</td>
<td>5 bits per data word</td>
</tr>
<tr>
<td>BIT6FDF</td>
<td>6h</td>
<td>6</td>
<td>6 bits per data word</td>
</tr>
<tr>
<td>BIT7FDF</td>
<td>7h</td>
<td>7</td>
<td>7 bits per data word</td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2CMDR_field_symval
### Table 15. Master-Transmitter/Receiver Bus Activity Defined by RM, STT, and STP Bits

<table>
<thead>
<tr>
<th>I2CMDR Bit</th>
<th>RM</th>
<th>STT</th>
<th>STP</th>
<th>Bus Activity†</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>None</td>
<td>No activity</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>P</td>
<td>STOP condition</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>P</td>
<td>START condition, slave address, (n) data words ((n = \text{value in I2CCNT}))</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>D-P</td>
<td>START condition, slave address, (n) data words, STOP condition ((n = \text{value in I2CCNT}))</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>None</td>
<td>No activity</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>P</td>
<td>STOP condition</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>S-A-D-D-D…</td>
<td>Repeat mode transfer: START condition, slave address, continuous data transfers until STOP condition or next START condition</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Reserved bit combination (No activity)</td>
<td></td>
</tr>
</tbody>
</table>

† A = Address; D = Data word; P = STOP condition; S = START condition

### Table 16. How the MST and FDF Bits Affect the Role of TRX Bit

<table>
<thead>
<tr>
<th>I2CMDR Bit</th>
<th>MST</th>
<th>FDF</th>
<th>I2C Module State</th>
<th>Function of TRX Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>In slave mode but not free data format mode</td>
<td>TRX is a don’t care. Depending on the command from the master, the I2C module responds as a receiver or a transmitter.</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>In slave mode and free data format mode</td>
<td>The free data format mode requires that the transmitter and receiver be fixed. TRX identifies the role of the I2C module: TRX = 0: The I2C module is a receiver. TRX = 1: The I2C module is a transmitter.</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td></td>
<td>In master mode; free data format mode on or off</td>
<td>TRX = 0: The I2C module is a receiver. TRX = 1: The I2C module is a transmitter.</td>
</tr>
</tbody>
</table>
Figure 38.  Block Diagram Showing the Effects of the Digital Loopback Mode (DLB) Bit
8.10 I2C Interrupt Source Register (I2CISR)

The I2C interrupt source register (I2CISR) is used by the CPU to determine which event generated the I2C interrupt. For more information about these events, see the descriptions of the I2C interrupt requests in Table 3. The I2CISR is shown in Figure 39 and described in Table 17.

Figure 39. I2C Interrupt Source Register (I2CISR)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31−3</td>
<td>Reserved</td>
<td>–</td>
<td>0</td>
<td>These reserved bit locations are always read as zeros. Always write 0 to this field.</td>
</tr>
<tr>
<td>2−0</td>
<td>INTOCODE</td>
<td></td>
<td></td>
<td>Interrupt code bits. The binary code in INTOCODE indicates which event generated an I2C interrupt.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>symval†</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>000</td>
<td>None</td>
</tr>
<tr>
<td>AL</td>
<td>001</td>
<td>Arbitration is lost.</td>
</tr>
<tr>
<td>NACK</td>
<td>010</td>
<td>No-acknowledgement condition is detected.</td>
</tr>
<tr>
<td>RAR</td>
<td>011</td>
<td>Registers are ready to be accessed.</td>
</tr>
<tr>
<td>RDR</td>
<td>100</td>
<td>Receive data is ready.</td>
</tr>
<tr>
<td>XDR</td>
<td>101</td>
<td>Transmit data is ready.</td>
</tr>
<tr>
<td>–</td>
<td>110−111</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

† For CSL C macro implementation, use the notation I2C_I2CISR_INTCODE_symval
8.11 I2C Prescaler Register (I2CPSC)

The I2C prescaler register (I2CPSC) is used for dividing down SYSCLK2 to obtain the desired prescaled module clock for the operation of I2C.

The IPSC bits must be initialized while I2C is in reset (IRS = 0 in I2CMDR). The prescaled frequency takes effect only when the IRS bit is changed to 1. Changing the IPSC value while IRS = 1 has no effect.

**CAUTION**

Prescaled Module Clock Frequency Range

The I2C module must be operated with a prescaled module clock frequency of 6.7 to 13.3 MHz. The I2C prescaler register (I2CPSC) must be configured to this frequency range.

The I2C prescaler register (I2CPSC) is shown in Figure 40 and described in Table 18.

**Figure 40. I2C Prescaler Register (I2CPSC)**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31−8</td>
<td>Reserved</td>
<td>0</td>
<td>These reserved bit locations are always read as zeros. A value written to this field has no effect.</td>
</tr>
<tr>
<td>7−0</td>
<td>IPSC</td>
<td>0−FFh</td>
<td>I2C prescaler divide-down value. The I2C module must be operated with a prescaled module clock frequency of 6.7 to 13.3 MHz. IPSC must be set to a value for this frequency range. IPSC determines how much SYSCLK2 is divided to create the I2C clock: prescaled module clock frequency = SYSCLK2 frequency/(IPSC + 1)</td>
</tr>
</tbody>
</table>

**Note:** IPSC must be initialized while I2C is in reset (IRS = 0 in I2CMDR).
8.12 I2C Peripheral Identification Registers (I2CPID1 and I2CPID2)

The peripheral identification registers (PID) contain identification data for the I2C module. I2CPID1 is shown in Figure 41 and described in Table 19. I2CPID2 is shown in Figure 42 and described in Table 20.

Table 19. I2C Peripheral Identification Register 1 (I2CPID1) Field Descriptions

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31−16</td>
<td>Reserved</td>
<td>0</td>
<td>These reserved bit locations are always read as zeros. A value written to this field has no effect.</td>
</tr>
<tr>
<td>15−8</td>
<td>CLASS</td>
<td></td>
<td>Identifies class of peripheral.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Serial port</td>
</tr>
<tr>
<td>7−0</td>
<td>REV</td>
<td></td>
<td>Identifies revision of peripheral.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x</td>
<td>See the device-specific datasheet for the value.</td>
</tr>
</tbody>
</table>
**Figure 42. I2C Peripheral Identification Register 2 (I2CPID2)**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31–8</td>
<td>Reserved</td>
<td>0</td>
<td>These reserved bit locations are always read as zeros. A value written to this field has no effect.</td>
</tr>
<tr>
<td>7–0</td>
<td>TYPE</td>
<td>05h</td>
<td>I2C</td>
</tr>
</tbody>
</table>

**Legend:**  R = Read only; -x = value after reset

**Table 20. I2C Peripheral Identification Register 2 (I2CPID2) Field Descriptions**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31–8</td>
<td>Reserved</td>
<td>0</td>
<td>These reserved bit locations are always read as zeros. A value written to this field has no effect.</td>
</tr>
<tr>
<td>7–0</td>
<td>TYPE</td>
<td>05h</td>
<td>I2C</td>
</tr>
</tbody>
</table>
Table 21 lists the changes made since the previous version of this document.

**Table 21. Document Revision History**

<table>
<thead>
<tr>
<th>Page</th>
<th>Additions/Modifications/Deletions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2</td>
<td>Updated Basic column of Slave-Receiver Mode cell.</td>
</tr>
</tbody>
</table>
This page is intentionally left blank.
A

A (own address) bits 35
A (slave address) bits 48
AAS bit 37
AD0 bit 37
addressing formats (serial data formats) 8
AL bit
   in I2CIER 36
   in I2CSTR 37
arbitration among master devices 12
ARDY bit
   in I2CIER 36
   in I2CSTR 37

B

BB bit 37
BC bits 50
bit polling methods to control data flow 22
block diagrams
   effect of digital loopback mode (DLB) bit 57
   I2C module 4
   I2C module clocking 14
   I2C module interrupt requests 18
   multiple I2C modules connected 1

C

CLASS bits 60
clock generation 13
clock synchronization 15

D

data (data receive) bits 47
D (data transmit) bits 49
data flow control
   using bit polling 22
   using EDMA events 32
   using interrupts 28
data formats 8
data validity 5
disabling the I2C module 18
DLB bit 50

E

EDMA event methods to control data flow 32
EDMA events generated by the I2C module 18

F

FDF bit 50
features 2
features not supported 2
flow diagrams
   setup procedure 21
   using bit polling 21
FREE bit 50
free data format 10
functional overview 3

I

I2C clock high-time divider register (I2CCLKH) 43
I2C clock low-time divider register (I2CCLKL) 43
I2C data count register (I2CCNT) 46
I2C data receive register (I2CDRR) 47
I2C data transmit register (I2CDXR) 49
I2C interrupt enable register (I2CIER) 36
I2C interrupt source register (I2CISR) 58
Index

I2C mode register (I2CMDR) 50
I2C own address register (I2COAR) 35
I2C peripheral identification register (I2CPID) 60
I2C prescaler register (I2CPSC) 59
I2C slave address register (I2CSAR) 48
I2C status register (I2CSTR) 37
I2CCLKH 43
I2CCLKL 43
I2CCNT 46
I2CDRR 47
I2CDXR 49
I2CIER 36
I2CISR 58
I2CMODR 50
I2COAR 35
I2CPID 60
I2CPSC 59
I2CSAR 48
I2CSTR 37
ICCH bits 45
ICCL bits 44
ICDC bits 46
ICRDY bits
  in I2CIER 36
  in I2CSTR 37
ICXRDY bits
  in I2CIER 36
  in I2CSTR 37
input and output voltage levels 5
INTCODE bits 58
interrupt flags in I2CSTR 37
interrupt methods to control data flow 28
interrupt requests generated by the I2C module 16

introduction 1
IPSC bits 59
IRS bit 50

M
master-receiver mode 6
master-transmitter mode 6
MST bit 50

N
NACK bit
generation of
  in I2CIER 36
  in I2CSTR 37
NACKMOD bit 50
NACKSNT bit 37

O
operating modes
  master-receiver mode 6
  master-transmitter mode 6
  slave-receiver mode 6
  slave-transmitter mode 6
operational details 5
output and input voltage levels 5

P
programming guide 20

R
registers
  I2C
clock high-time divider register
    (I2CCLKH) 43
clock low-time divider register
    (I2CCLKL) 43
data count register (I2CCNT) 46
data receive register (I2CDRR) 47
data transmit register (I2CDXR) 49
interrupt enable register (I2CIER) 36
interrupt source register (I2CISR) 58
mode register (I2CMODR) 50
own address register (I2COAR) 35
peripheral identification register (I2CPID) 60
prescaler register (I2CPSC) 59
slave address register (I2CSAR) 48
status register (I2CSTR) 37
resetting the I2C module 18
REV bits 60
revision history 62
RM bit 50
RSFULL bit 37
Index

S

serial data formats 8
  10-bit addressing format 9
  7-bit addressing format 9
  free data format 10
slave-receiver mode 6
slave-transmitter mode 6
START and STOP conditions 7
STB bit 50
STP bit 50
STT bit 50

T

TRX bit 50
TYPE bits 61

V

voltage levels 5

X

XA bit 50
XSMT bit 37