

TMS320x280x, 2801x, 2804x Inter-Integrated Circuit (I2C) Module

Reference Guide



Literature Number: SPRU721C
November 2004–Revised June 2011

| | |
|---|-----------|
| Preface | 7 |
| 1 Introduction to the I2C Module | 11 |
| 1.1 Features | 12 |
| 1.2 Features Not Supported | 12 |
| 1.3 Functional Overview | 12 |
| 1.4 Clock Generation | 13 |
| 2 I2C Module Operational Details | 14 |
| 2.1 Input and Output Voltage Levels | 14 |
| 2.2 Data Validity | 14 |
| 2.3 Operating Modes | 15 |
| 2.4 I2C Module START and STOP Conditions | 16 |
| 2.5 Serial Data Formats | 16 |
| 2.6 NACK Bit Generation | 18 |
| 2.7 Clock Synchronization | 18 |
| 2.8 Arbitration | 19 |
| 3 Interrupt Requests Generated by the I2C Module | 20 |
| 3.1 Basic I2C Interrupt Requests | 20 |
| 3.2 I2C FIFO Interrupts | 21 |
| 4 Resetting/Disabling the I2C Module | 21 |
| 5 I2C Module Registers | 21 |
| 5.1 I2C Mode Register (I2CMR) | 23 |
| 5.2 I2C Extended Mode Register (I2CEMR) | 26 |
| 5.3 I2C Interrupt Enable Register (I2CIER) | 28 |
| 5.4 I2C Status Register (I2CSTR) | 28 |
| 5.5 I2C Interrupt Source Register (I2CISRC) | 31 |
| 5.6 I2C Prescaler Register (I2CPSC) | 32 |
| 5.7 I2C Clock Divider Registers (I2CCLKL and I2CCLKH) | 33 |
| 5.8 I2C Slave Address Register (I2CSAR) | 34 |
| 5.9 I2C Own Address Register (I2COAR) | 34 |
| 5.10 I2C Data Count Register (I2CCNT) | 35 |
| 5.11 I2C Data Receive Register (I2CDRR) | 35 |
| 5.12 I2C Data Transmit Register (I2CDXR) | 36 |
| 5.13 I2C Transmit FIFO Register (I2CFFTX) | 36 |
| 5.14 I2C Receive FIFO Register (I2CFFRX) | 37 |
| Appendix A Revision History | 39 |

List of Figures

| | | |
|----|---|----|
| 1 | Multiple I2C Modules Connected | 11 |
| 2 | I2C Module Conceptual Block Diagram | 13 |
| 3 | Clocking Diagram for the I2C Module | 14 |
| 4 | Bit Transfer on the I2C-Bus | 15 |
| 5 | I2C Module START and STOP Conditions | 16 |
| 6 | I2C Module Data Transfer (7-Bit Addressing with 8-bit Data Configuration Shown) | 16 |
| 7 | I2C Module 7-Bit Addressing Format (FDF = 0, XA = 0 in I2CMMDR)..... | 17 |
| 8 | I2C Module 10-Bit Addressing Format (FDF = 0, XA = 1 in I2CMMDR) | 17 |
| 9 | I2C Module Free Data Format (FDF = 1 in I2CMMDR) | 17 |
| 10 | Repeated START Condition (in This Case, 7-Bit Addressing Format) | 18 |
| 11 | Synchronization of Two I2C Clock Generators During Arbitration | 19 |
| 12 | Arbitration Procedure Between Two Master-Transmitters | 19 |
| 13 | Enable Paths of the I2C Interrupt Requests | 21 |
| 14 | I2C Mode Register (I2CMMDR) | 23 |
| 15 | Pin Diagram Showing the Effects of the Digital Loopback Mode (DLB) Bit | 26 |
| 16 | I2C Extended Mode Register (I2CEMDR) | 26 |
| 17 | BCM Bit, Slave Transmitter Mode..... | 27 |
| 18 | I2C Interrupt Enable Register (I2CIER)..... | 28 |
| 19 | I2C Status Register (I2CSTR) | 29 |
| 20 | I2C Interrupt Source Register (I2CISRC) | 32 |
| 21 | I2C Prescaler Register (I2CPSC) | 32 |
| 22 | The Roles of the Clock Divide-Down Values (ICCL and ICCH)..... | 33 |
| 23 | I2C Clock Low-Time Divider Register (I2CCLKL) | 33 |
| 24 | I2C Clock High-Time Divider Register (I2CCLKH) | 33 |
| 25 | I2C Slave Address Register (I2CSAR) | 34 |
| 26 | I2C Own Address Register (I2COAR) | 34 |
| 27 | I2C Data Count Register (I2CCNT) | 35 |
| 28 | I2C Data Receive Register (I2CDRR) | 35 |
| 29 | I2C Data Transmit Register (I2CDXR)..... | 36 |
| 30 | I2C Transmit FIFO Register (I2CFFTX) | 36 |
| 31 | I2C Receive FIFO Register (I2CFFRX)..... | 37 |

List of Tables

| | | |
|----|---|----|
| 1 | Operating Modes of the I2C Module | 15 |
| 2 | Ways to Generate a NACK Bit..... | 18 |
| 3 | Descriptions of the Basic I2C Interrupt Requests | 20 |
| 4 | I2C Module Registers | 21 |
| 5 | I2C Mode Register (I2CMDR) Field Descriptions | 23 |
| 6 | Master-Transmitter/Receiver Bus Activity Defined by the RM, STT, and STP Bits of I2CMDR | 25 |
| 7 | How the MST and FDF Bits of I2CMDR Affect the Role of the TRX Bit of I2CMDR | 25 |
| 8 | I2C Extended Mode Register (I2CEMDR) Field Descriptions | 26 |
| 9 | I2C Interrupt Enable Register (I2CIER) Field Descriptions | 28 |
| 10 | I2C Status Register (I2CSTR) Field Descriptions | 29 |
| 11 | I2C Interrupt Source Register (I2CISRC) Field Descriptions | 32 |
| 12 | I2C Prescaler Register (I2CPSC) Field Descriptions | 32 |
| 13 | I2C Clock Low-Time Divider Register (I2CCLKL) Field Description | 33 |
| 14 | I2C Clock High-Time Divider Register (I2CCLKH) Field Description | 33 |
| 15 | Dependency of Delay d on the Divide-Down Value IPSC | 34 |
| 16 | I2C Slave Address Register (I2CSAR) Field Descriptions | 34 |
| 17 | I2C Own Address Register (I2COAR) Field Descriptions | 35 |
| 18 | I2C Data Count Register (I2CCNT) Field Descriptions..... | 35 |
| 19 | I2C Data Receive Register (I2CDRR) Field Descriptions | 36 |
| 20 | I2C Data Transmit Register (I2CDXR) Field Descriptions | 36 |
| 21 | I2C Transmit FIFO Register (I2CFFTX) Field Descriptions..... | 36 |
| 22 | I2C Receive FIFO Register (I2CFFRX) Field Descriptions | 37 |
| 23 | Changes in This Revision | 39 |

Read This First

About This Manual

This manual describes the features and operation of the inter-integrated circuit (I2C) module that is available on the TMS320x280x digital signal processors (devices). The I2C module provides an interface between one of these 28x devices and devices compliant with Philips Semiconductors Inter-IC bus (I2C-bus) specification version 2.1 and connected by way of an I2C-bus. This manual assumes the reader has familiarity with the I2C-bus specification.

The I2C module described in this reference guide is a Type 0 I2C. See the *TMS320x28xx, 28xxx DSP Peripheral Reference Guide* ([SPRU566](#)) for a list of all devices with an I2C module of the same type, to determine the differences between types, and for a list of device-specific differences within a type.

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 C28x™ 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.

Data Manuals—

[SPRS230](#) — **TMS320F2809, F2808, F2806, F2802, F2801, C2802, C2801, and F2801x devices Data Manual** contains the pinout, signal descriptions, as well as electrical and timing specifications for the F280x devices.

[SPRZ171](#) — **TMS320F280x, TMS320C280x, and TMS320F2801x DSC Silicon Errata** describes the advisories and usage notes for different versions of silicon.

[SPRS357](#) — **TMS320F28044 Digital Signal Processor Data Manual** contains the pinout, signal descriptions, as well as electrical and timing specifications for the F28044 device.

[SPRZ255](#) — **TMS320F28044 device Silicon Errata** describes the advisories and usage notes for different versions of silicon.

Peripheral Guides—

[SPRU051](#) — **TMS320x28xx, 28xxx Serial Communication Interface (SCI) Reference Guide** describes the SCI, which is a two-wire asynchronous serial port, commonly known as a UART. The SCI modules support digital communications between the CPU and other asynchronous peripherals that use the standard non-return-to-zero (NRZ) format.

- SPRU059**— [TMS320x28xx, 28xxx Serial Peripheral Interface \(SPI\) Reference Guide](#) describes the SPI - a high-speed synchronous serial input/output (I/O) port - that allows a serial bit stream of programmed length (one to sixteen bits) to be shifted into and out of the device at a programmed bit-transfer rate.
- SPRU074**— [TMS320x28xx, 28xxx Enhanced Controller Area Network \(eCAN\) Reference Guide](#) describes the eCAN that uses established protocol to communicate serially with other controllers in electrically noisy environments.
- SPRU430**— [TMS320C28x device CPU and Instruction Set Reference Guide](#) describes the central processing unit (CPU) and the assembly language instructions of the TMS320C28x fixed-point processors. It also describes emulation features available on these devices.
- SPRU513**— [TMS320C28x Assembly Language Tools User's Guide](#) describes the assembly language tools (assembler and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the TMS320C28x device.
- SPRU514**— [TMS320C28x Optimizing C Compiler User's Guide](#) describes the TMS320C28x™ C/C++ compiler. This compiler accepts ANSI standard C/C++ source code and produces TMS320 device assembly language source code for the TMS320C28x device.
- SPRU566**— [TMS320x28xx, 28xxx DSP Peripheral Reference Guide](#) describes the peripheral reference guides of the 28x digital signal processors (DSPs).
- SPRU608**— [The TMS320C28x Instruction Set Simulator Technical Overview](#) describes the simulator, available within the Code Composer Studio for TMS320C2000 IDE, that simulates the instruction set of the C28x™ core.
- SPRU625**— [TMS320C28x device/BIOS Application Programming Interface \(API\) Reference Guide](#) describes development using device/BIOS.
- SPRU712**— [TMS320x28xx, 28xxx System Control and Interrupts Reference Guide](#) describes the various interrupts and system control features of the 28x digital signal processors.
- SPRU716**— [TMS320x280x, 2801x, 2804x Analog-to-Digital Converter \(ADC\) Reference Guide](#) describes how to configure and use the on-chip ADC module, which is a 12-bit pipelined ADC.
- SPRU722**— [TMS320x280x, 2801x, 2804x Boot ROM Reference Guide](#) describes the purpose and features of the bootloader (factory-programmed boot-loading software). It also describes other contents of the device on-chip boot ROM and identifies where all of the information is located within that memory.
- SPRU790**— [TMS320x280x, 2801x, 2804x Enhanced Quadrature Encoder Pulse \(eQEP\) Module Reference Guide](#) describes the eQEP module, which is used for interfacing with a linear or rotary incremental encoder to get position, direction, and speed information from a rotating machine in high performance motion and position control systems. It includes the module description and registers
- SPRU791**— [TMS320x28xx, 28xxx Enhanced Pulse Width Modulator \(ePWM\) Module Reference Guide](#) describes the main areas of the enhanced pulse width modulator that include digital motor control, switch mode power supply control, UPS (uninterruptible power supplies), and other forms of power conversion
- SPRU807**— [TMS320x280x, 2801x, 2804x Enhanced Capture \(eCAP\) Module Reference Guide](#) describes the enhanced capture module. It includes the module description and registers.
- SPRU924**— [TMS320x280x, 2801x, 2804x High-Resolution Pulse Width Modulator Reference Guide](#) describes the operation of the high-resolution extension to the pulse width modulator (HRPWM).

Application Reports—

- SPRAA58**— [TMS320x281x to TMS320x280x Migration Overview](#) describes differences between the Texas Instruments TMS320x281x and TMS320x280x devices to assist in application migration from the 281x to the 280x. While the main focus of this document is migration from 281x to 280x, users considering migrating in the reverse direction (280x to 281x) will also find this document useful.
- SPRA550**— [3.3 V device for Digital Motor Control](#) describes a scenario of a 3.3-V-only motor controller indicating that for most applications, no significant issue of interfacing between 3.3 V and 5 V exists. On-chip 3.3-V analog-to-digital converter (ADC) versus 5-V ADC is also discussed. Guidelines for component layout and printed circuit board (PCB) design that can reduce system noise and EMI effects are summarized.
- SPRA820**— [Online Stack Overflow Detection on the TMS320C28x device](#) presents the methodology for online stack overflow detection on the TMS320C28x™ device. C-source code is provided that contains functions for implementing the overflow detection on both device/BIOS™ and non-device/BIOS applications.
- SPRA861**— [RAMDISK: A Sample User-Defined C I/O Driver](#) provides an easy way to use the sophisticated buffering of the high-level CIO functions on an arbitrary device. This application report presents a sample implementation of a user-defined device driver.
- SPRA873**— [Thermo-Electric Cooler Control Using a TMS320F2812 device & DRV592 Power Amplifier](#) presents a thermoelectric cooler system consisting of a Texas Instruments TMS320F2812 digital signal processor (device) and DRV592 power amplifier. The device implements a digital proportional-integral-derivative feedback controller using an integrated 12-bit analog-to-digital converter to read the thermistor, and direct output of pulse-width-modulated waveforms to the H-bridge DRV592 power amplifier. A complete description of the experimental system, along with software and software operating instructions, is provided.
- SPRA876**— [Programming Examples for the TMS320F281x eCAN](#) contains several programming examples to illustrate how the eCAN module is set up for different modes of operation to help you come up to speed quickly in programming the eCAN. All projects and CANalyzer configuration files are included in the attached SPRA876.zip file.
- SPRA953**— [IC Package Thermal Metrics](#) describes the traditional and new thermal metrics and will put their application in perspective with respect to system level junction temperature estimation.
- SPRA958**— [Running an Application from Internal Flash Memory on the TMS320F281x device \(Rev. B\)](#) covers the requirements needed to properly configure application software for execution from on-chip flash memory. Requirements for both device/BIOS™ and non-device/BIOS projects are presented. Example code projects are included.
- SPRA963**— [Reliability Data for TMS320LF24x and TMS320F281x Devices](#) describes reliability data for TMS320LF24x and TMS320F281x devices.
- SPRA989**— [F2810, F2811, and F2812 ADC Calibration](#) describes a method for improving the absolute accuracy of the 12-bit analog-to-digital converter (ADC) found on the F2810/F2811/F2812 devices. This application note is accompanied by an example program (ADCcalibration.zip) that executes from RAM on the F2812 eZdevice.
- SPRA991**— [Simulation Fulfills its Promise for Enhancing Debug and Analysis - A White Paper](#) describes simulation enhancements that enable developers to speed up the development cycle by allowing them to evaluate system alternatives more effectively.

TMS320x280x Inter-Integrated Circuit Module

This guide describes the features and operation of the inter-integrated circuit (I2C) module that is available on the TMS320x280x digital signal processor (device). The I2C module provides an interface between one of these devices and 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 1 to 8-bit data to/from the device through the I2C module. This guide assumes the reader is familiar with the I2C-bus specification.

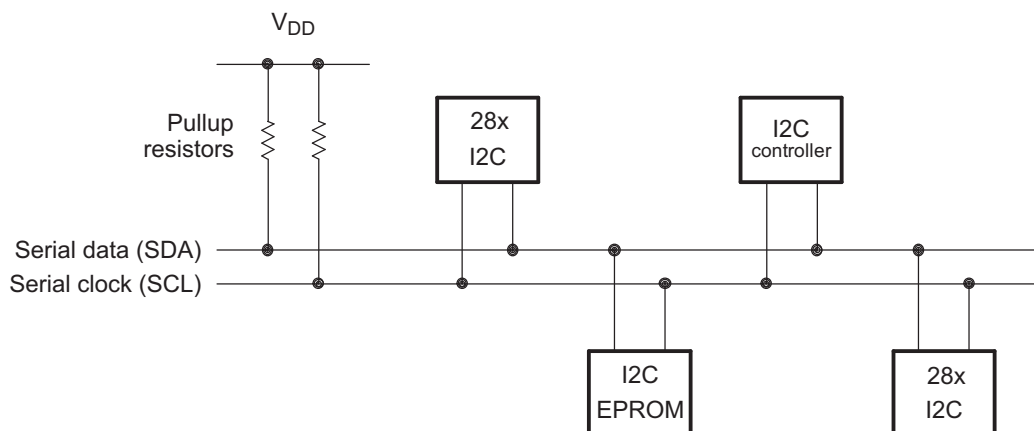
NOTE: A unit of data transmitted or received by the I2C module can have fewer than 8 bits; however, for convenience, a unit of data is called a data byte throughout this document. (The number of bits in a data byte is selectable via the BC bits of the mode register, I2CMDR.)

This reference guide is applicable for the I2C found on the TMS320x280x.

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



1.1 Features

The I2C module has the following features:

- Compliance with the Philips Semiconductors I2C-bus specification (version 2.1):
 - Support for 8-bit format transfers
 - 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 16-byte receive FIFO and one 16-byte transmit FIFO
- One interrupt that can always 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-acknowledgment received, arbitration lost, stop condition detected, addressed as slave.
- An additional interrupt that can be used by the CPU when in FIFO mode
- 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

1.3 Functional Overview

Each device connected to an I2C-bus is recognized by a unique address. Each device can operate as either a transmitter or a receiver, depending on the function of the device. A device connected to the I2C-bus can also be considered as the master or the slave when performing data transfers. 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 28x 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.

There are two major transfer techniques:

- Standard Mode: Send exactly n data values, where n is a value you program in an I2C module register. See [Table 5](#) for register information.
- Repeat Mode: Keep sending data values until you use software to initiate a STOP condition or a new START condition. See [Table 5](#) for RM bit information.

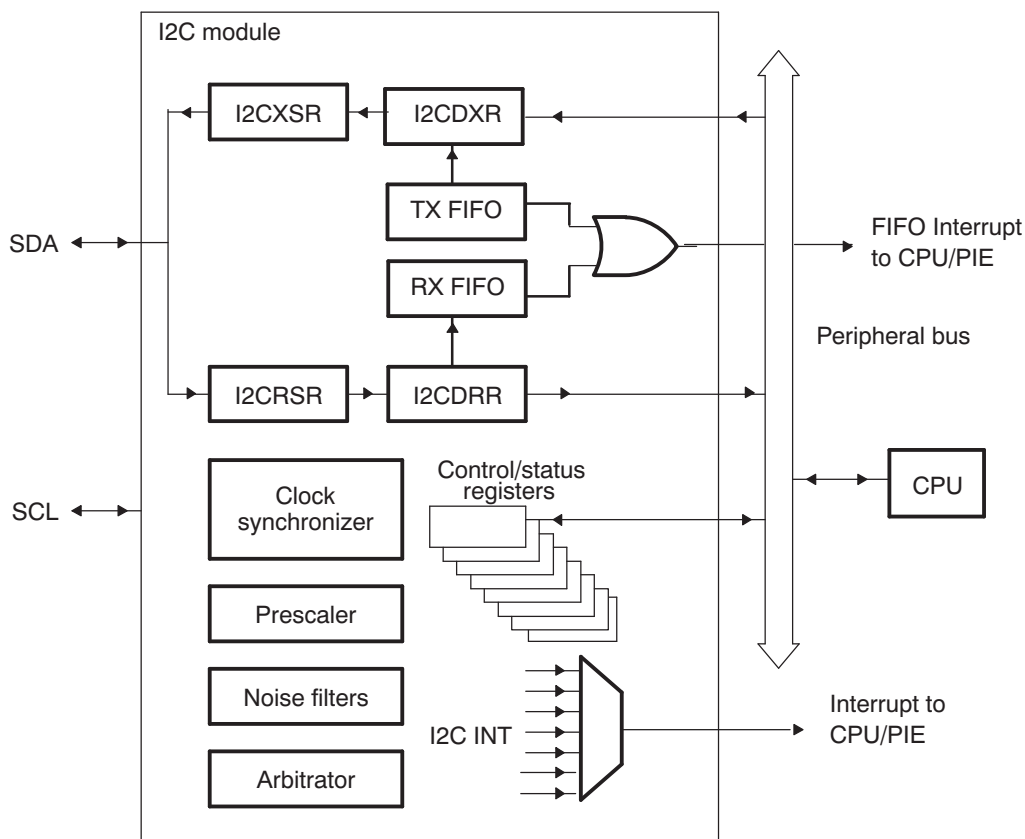
The I2C module consists of the following primary blocks:

- A serial interface: one data pin (SDA) and one clock pin (SCL)
- Data registers and FIFOs to temporarily hold receive data and transmit data traveling between the SDA pin and the CPU
- Control and status registers
- A peripheral bus interface to enable the CPU to access the I2C module registers and FIFOs.
- A clock synchronizer to synchronize the I2C input clock (from the device 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
- FIFO interrupt generation logic, so that FIFO access can be synchronized to data reception and data transmission in the I2C module

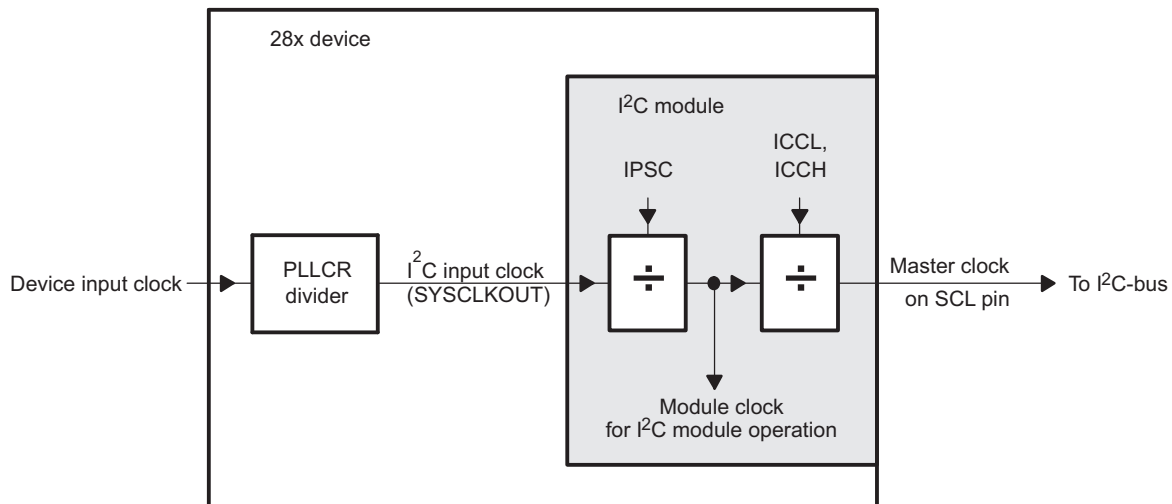
Figure 2 shows the four registers used for transmission and reception in non-FIFO mode. The CPU writes data for transmission to I2CDXR and reads received data from I2CDRR. When the I2C module is configured as a transmitter, data written to I2CDXR is copied to I2CXHR and shifted out on the SDA pin one bit at a time. When the I2C module is configured as a receiver, received data is shifted into I2CHSR and then copied to I2CDRR.

Figure 2. I2C Module Conceptual Block Diagram



1.4 Clock Generation

As shown in Figure 3, the device clock generator receives a signal from an external clock source and produces an I2C input clock with a programmed frequency. The I2C input clock is equivalent to the CPU clock and is then divided twice more inside the I2C module to produce the module clock and the master clock.

Figure 3. Clocking Diagram for the I2C Module


The module clock determines the frequency at which the I2C module operates. A programmable prescaler in the I2C module divides down the I2C input clock to produce the module clock. To specify the divide-down value, initialize the IPSC field of the prescaler register, I2CPSC. The resulting frequency is:

$$\text{module clock frequency} = \frac{\text{I2C input clock frequency}}{(\text{IPSC} + 1)}$$

NOTE: To meet all of the I2C protocol timing specifications, the module clock must be configured between 7 - 12 MHz.

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

The master clock appears on the SCL pin when the I2C module is configured to be a master on the I2C-bus. This clock controls the timing of communication between the I2C module and a slave. As shown in Figure 3, a second clock divider in the I2C module divides down the module clock to produce the master clock. The clock divider uses the ICCL value of I2CCLKL to divide down the low portion of the module clock signal and uses the ICCH value of I2CCLKH to divide down the high portion of the module clock signal. See section Section 5.7.1 for the master clock frequency equation.

2 I2C Module Operational Details

This section provides an overview of the I2C-bus protocol and how it is implemented.

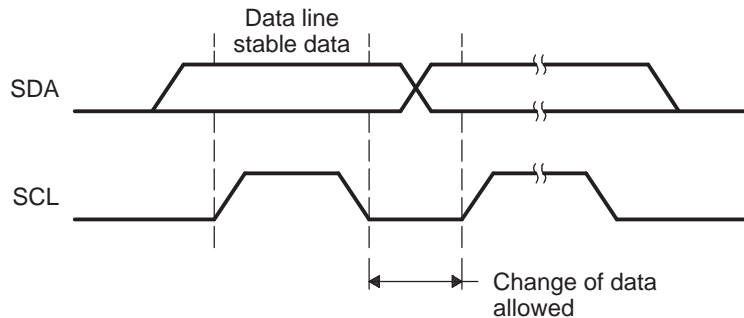
2.1 Input and Output Voltage Levels

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 I2C-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 data manual for your particular device.

2.2 Data Validity

The data on SDA must be stable during the high period of the clock (see Figure 4). The high or low state of the data line, SDA, should change only when the clock signal on SCL is low.

Figure 4. Bit Transfer on the I2C-Bus



2.3 Operating Modes

The I2C 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 I2C 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 I2C module must remain a master-transmitter. To receive data from a slave, the I2C module must be changed to the master-receiver mode.

If the I2C module is a slave, it begins as a slave-receiver and typically sends acknowledgment when it recognizes its slave address from a master. If the master will be sending data to the I2C module, the module must remain a slave-receiver. If the master has requested data from the I2C module, the module must be changed to the slave-transmitter mode.

Table 1. Operating Modes of the I2C Module

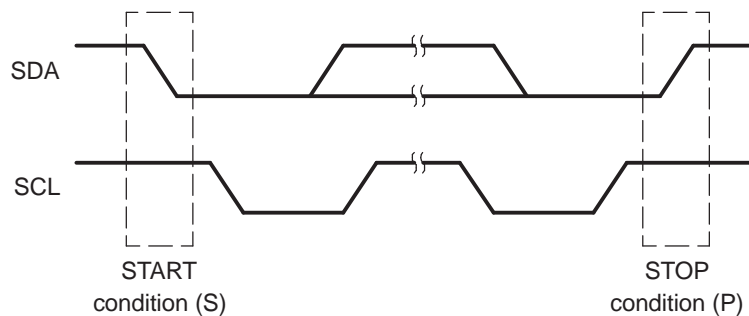
| Operating Mode | Description |
|--------------------------|--|
| Slave-receiver modes | <p>The I2C module is a slave and receives data from a master.</p> <p>All slaves begin in this mode. In this mode, serial data bits received on SDA are shifted in with the clock pulses that are generated by the master. As a slave, the I2C module does not generate the clock signal, but it can hold SCL low while the intervention of the device is required (RSFULL = 1 in I2CSTR) after a byte has been received. See section Section 2.7 for more details.</p> |
| Slave-transmitter mode | <p>The I2C module is a slave and transmits data to a master.</p> <p>This mode can be entered only 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 byte 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 device is required (XSMT = 0 in I2CSTR) after a byte has been transmitted. See section Section 2.7 for more details.</p> |
| Master-receiver mode | <p>The I2C module is a master and receives data from a slave.</p> <p>This mode can be entered only 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 byte 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 device is required (RSFULL = 1 in I2CSTR) after a byte has been received.</p> |
| Master-transmitter modes | <p>The IC module is a master and transmits control information and data to a slave.</p> <p>All masters begin in this mode. 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 device is required (XSMT = 0 in I2CSTR) after a byte has been transmitted.</p> |

2.4 I2C Module 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 5](#):

- 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 5. 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 bit (MST) and the START condition bit (STT) in I2CMDR must both be 1. For the I2C module to end a data transfer with a STOP condition, the STOP condition bit (STP) must be set to 1. When the BB bit is set to 1 and the STT bit is set to 1, a repeated START condition is generated. For a description of I2CMDR and its bits (including MST, STT, and STP), see [Section 5.1](#).

2.5 Serial Data Formats

[Figure 6](#) shows an example of a data transfer on the I2C-bus. The I2C module supports 1 to 8-bit data values. In [Figure 6](#), 8-bit data is transferred. Each bit put on the SDA line equates to 1 pulse on the SCL line, and the values are always transferred with the most significant bit (MSB) first. The number of data values that can be transmitted or received is unrestricted. The serial data format used in [Figure 6](#) is the 7-bit addressing format. The I2C module supports the formats shown in [Figure 7](#) through [Figure 9](#) and described in the paragraphs that follow the figures.

NOTE: In [Figure 6](#) through [Figure 9](#), n = the number of data bits (from 1 to 8) specified by the bit count (BC) field of I2CMDR.

Figure 6. I2C Module Data Transfer (7-Bit Addressing with 8-bit Data Configuration Shown)

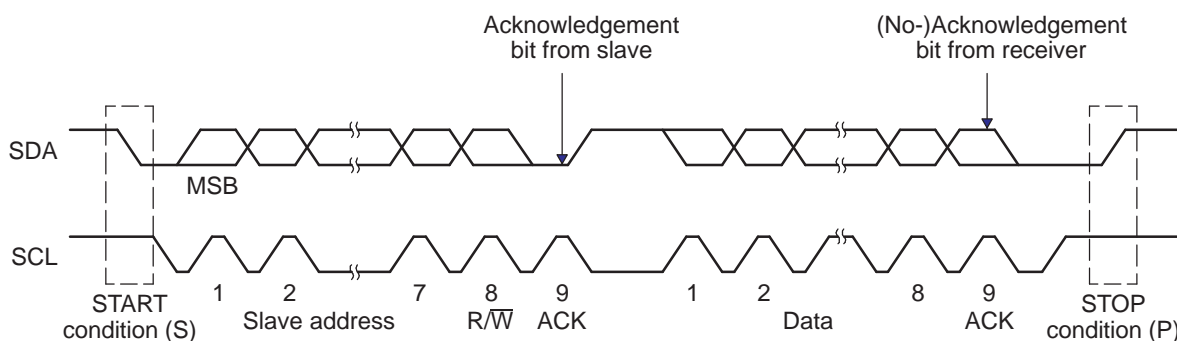


Figure 7. I2C Module 7-Bit Addressing Format (FDF = 0, XA = 0 in I2CMDR)

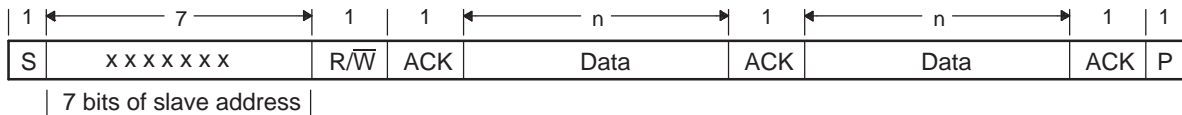


Figure 8. I2C Module 10-Bit Addressing Format (FDF = 0, XA = 1 in I2CMDR)

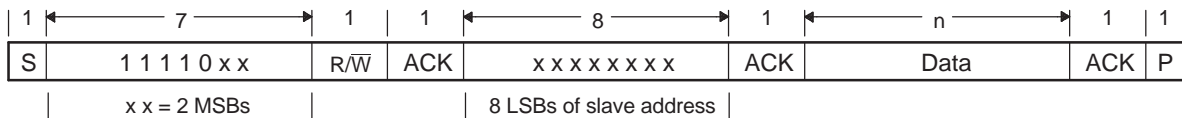
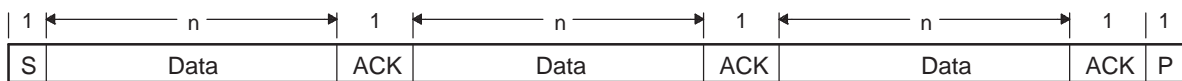


Figure 9. I2C Module Free Data Format (FDF = 1 in I2CMDR)



2.5.1 7-Bit Addressing Format

In the 7-bit addressing format (see [Figure 7](#)), the first byte after a START condition (S) consists of a 7-bit slave address followed by a R/W bit. R/W 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 acknowledgment (ACK) is inserted after each byte. If the ACK bit is inserted by the slave after the first byte from the master, 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) field 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, and make sure the free data format mode is off (FDF = 0 in I2CMDR).

2.5.2 10-Bit Addressing Format

The 10-bit addressing format (see [Figure 8](#)) is similar to the 7-bit addressing format, but the master sends the slave address in two separate byte transfers. The first byte consists of 11110b, 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 acknowledgment 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 I2C-bus specification.

To select the 10-bit addressing format, write 1 to the XA bit of I2CMDR and make sure the free data format mode is off (FDF = 0 in I2CMDR).

2.5.3 Free Data Format

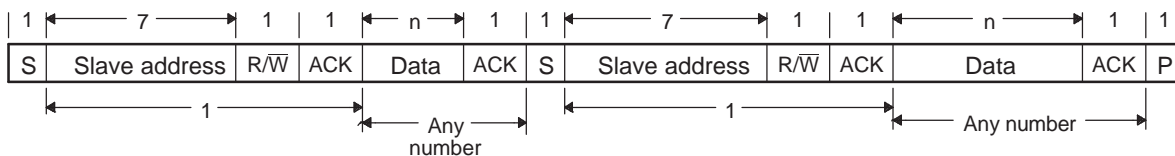
In this format (see [Figure 9](#)), the first byte after a START condition (S) is a data byte. An ACK bit is inserted after each data byte, which can be from 1 to 8 bits, depending on the BC field of I2CMDR. No address or data-direction bit is sent. Therefore, the transmitter and the receiver must both 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. The free data format is not supported in the digital loopback mode (DLB = 1 in I2CMDR).

2.5.4 Using a Repeated START Condition

At the end of each data byte, the master can drive another START condition. Using this capability, a master can communicate with multiple slave addresses without having to give up control of the bus by driving a STOP condition. The length of a data byte can be from 1 to 8 bits and is selected with the BC field of I2CMDR. The repeated START condition can be used with the 7-bit addressing, 10-bit addressing, and free data formats. Figure 10 shows a repeated START condition in the 7-bit addressing format.

Figure 10. Repeated START Condition (in This Case, 7-Bit Addressing Format)



NOTE: In Figure 10, n = the number of data bits (from 1 to 8) specified by the bit count (BC) field of I2CMDR.

2.6 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 you can tell the I2C module to send a NACK bit.

Table 2. Ways to Generate a NACK Bit

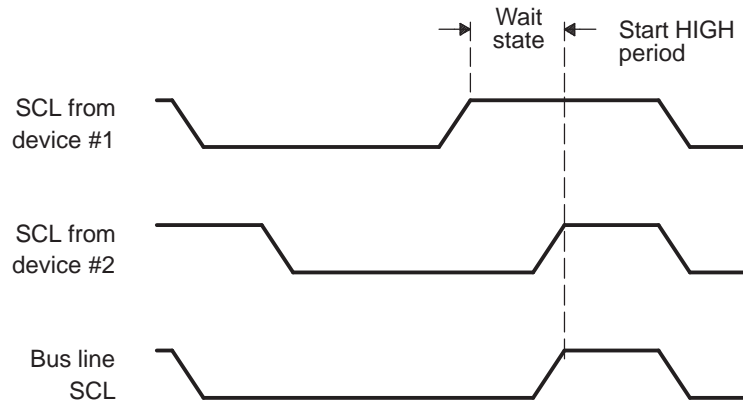
| I2C Module Condition | NACK Bit Generation Options |
|--|--|
| Slave-receiver modes | <ul style="list-style-type: none"> Allow an overrun condition (RSFULL = 1 in I2CSTR) Reset the module (IRS = 0 in I2CMDR) Set the NACKMOD bit of I2CMDR before the rising edge of the last data bit you intend to receive |
| Master-receiver mode AND Repeat mode (RM = 1 in I2CMDR) | <ul style="list-style-type: none"> Generate a STOP condition (STP = 1 in I2CMDR) Reset the module (IRS = 0 in I2CMDR) Set the NACKMOD bit of I2CMDR before the rising edge of the last data bit you intend to receive |
| Master-receiver mode AND Nonrepeat mode (RM = 0 in I2CMDR) | <ul style="list-style-type: none"> If STP = 1 in I2CMDR, allow the internal data counter to count down to 0 and thus force a STOP condition If STP = 0, make STP = 1 to generate a STOP condition Reset the module (IRS = 0 in I2CMDR). = 1 to generate a STOP condition Set the NACKMOD bit of I2CMDR before the rising edge of the last data bit you intend to receive |

2.7 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 11 illustrates the clock synchronization. The wired-AND property of SCL means that a device that first generates a low period on SCL 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 byte or to prepare a byte to be transmitted.

Figure 11. Synchronization of Two I2C Clock Generators During Arbitration



2.8 Arbitration

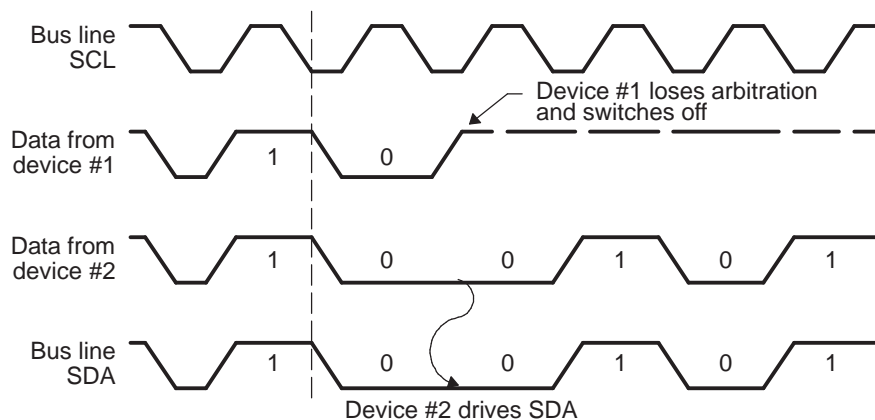
If two or more master-transmitters attempt to start a transmission on the same bus at approximately the same time, an arbitration procedure is invoked. The arbitration procedure uses the data presented on the serial data bus (SDA) by the competing transmitters. Figure 12 illustrates the arbitration procedure between two devices. The first master-transmitter, which release the SDA line 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 request.

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 12. Arbitration Procedure Between Two Master-Transmitters



3 Interrupt Requests Generated by the I2C Module

The I2C module can generate seven types of basic interrupt requests, which are described in [Section 3.1](#). Two of these can tell the CPU when to write transmit data and when to read receive data. If you want the FIFOs to handle transmit and receive data, you can also use the FIFO interrupts described in [Section 3.2](#). The basic I2C interrupts are combined to form PIE Group 8, Interrupt 1 (I2CINT1A_ISR), and the FIFO interrupts are combined to form PIE Group 8, Interrupt 2 (I2CINT2A_ISR).

3.1 Basic I2C Interrupt Requests

The I2C module generates the interrupt requests described in [Table 3](#). 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 (I2CINT1A_ISR). The I2CINT1A_ISR for the I2C interrupt can determine the interrupt source by reading the interrupt source register, I2CISRC. Then the I2CINT1A_ISR can branch to the appropriate subroutine.

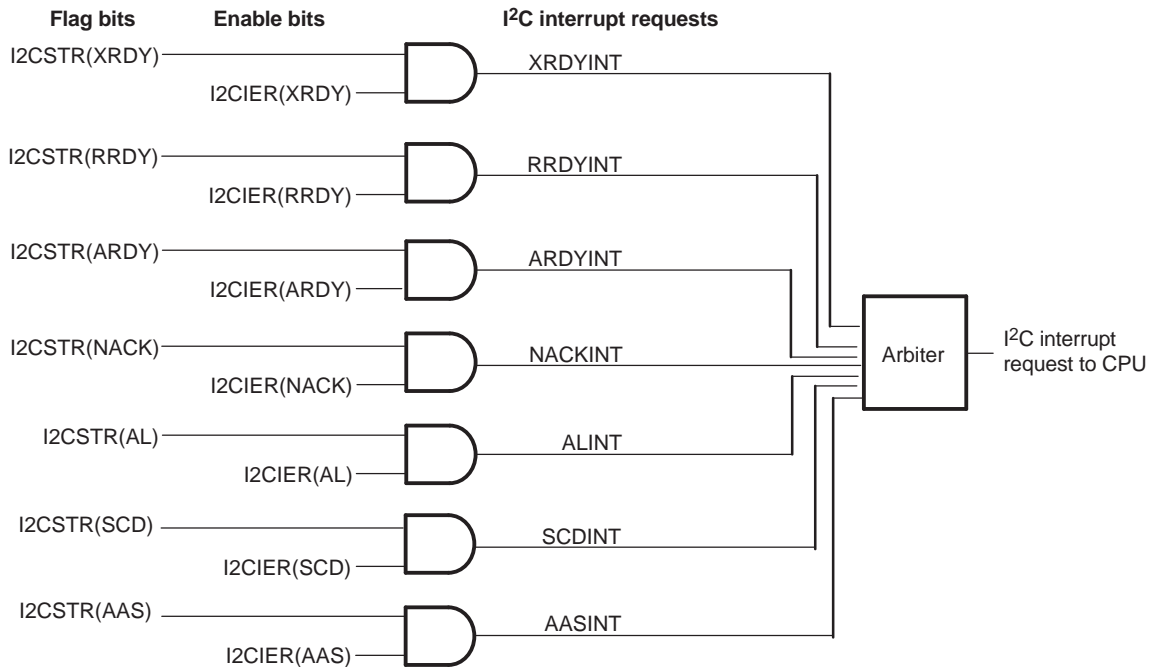
After the CPU reads I2CISRC, the following events occur:

1. The flag for the source interrupt is cleared in I2CSTR. Exception: The ARDY, RRDY, and XRDY bits in I2CSTR are not cleared when I2CISRC is read. To clear one of these bits, write a 1 to it.
2. The arbiter determines which of the remaining interrupt requests has the highest priority, writes the code for that interrupt to I2CISRC, and forwards the interrupt request to the CPU.

Table 3. Descriptions of the Basic I2C Interrupt Requests

| I2C Interrupt Request | Interrupt Source |
|-----------------------|--|
| XRDYINT | 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 XRDY bit of the status register, I2CSTR. XRDYINT should not be used when in FIFO mode. Use the FIFO interrupts instead. |
| RRDYINT | 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 RRDY bit of I2CSTR. RRDYINT should not be used when in FIFO mode. Use the FIFO interrupts instead. |
| ARDYINT | 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. As an alternative to using ARDYINT, the CPU can poll the ARDY bit. |
| NACKINT | No-acknowledgment condition: The I2C module is configured as a master-transmitter and did not receive acknowledgment from the slave-receiver. As an alternative to using NACKINT, the CPU can poll the NACK bit of I2CSTR. |
| ALINT | 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. |
| SCDINT | Stop condition detected: A STOP condition was detected on the I2C bus. As an alternative to using SCDINT, the CPU can poll the SCD bit of the status register, I2CSTR. |
| AASINT | Addressed as slave condition: The I2C has been addressed as a slave device by another master on the I2C bus. As an alternative to using AASINT, the CPU can poll the AAS bit of the status register, I2CSTR. |

Figure 13. Enable Paths of the I2C Interrupt Requests



3.2 I2C FIFO Interrupts

In addition to the seven basic I2C interrupts, the transmit and receive FIFOs each contain the ability to generate an interrupt (I2CINT2A). The transmit FIFO can be configured to generate an interrupt after transmitting a defined number of bytes, up to 16. The receive FIFO can be configured to generate an interrupt after receiving a defined number of bytes, up to 16. These two interrupt sources are ORed together into a single maskable CPU interrupt. The interrupt service routine can then read the FIFO interrupt status flags to determine from which source the interrupt came. See the I2C transmit FIFO register (I2CFFTX) and the I2C receive FIFO register (I2CFFRX) descriptions.

4 Resetting/Disabling the I2C Module

You can reset/disable the I2C module 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 device reset by driving the $\overline{\text{XRS}}$ pin low. The entire device is reset and is held in the reset state until you drive the pin high. When $\overline{\text{XRS}}$ 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.

5 I2C Module Registers

Table 4 lists the I2C module registers. All but the receive and transmit shift registers (I2CRSR and I2CXHR) are accessible to the CPU.

Table 4. I2C Module Registers

| Name | Address | Description |
|--------|---------|-------------------------------|
| I2COAR | 0x7900 | I2C own address register |
| I2CIER | 0x7901 | I2C interrupt enable register |

Table 4. I2C Module Registers (continued)

| Name | Address | Description |
|-------------|----------------|---|
| I2CSTR | 0x7902 | I2C status register |
| I2CCLKL | 0x7903 | I2C clock low-time divider register |
| I2CCLKH | 0x7904 | I2C clock high-time divider register |
| I2CCNT | 0x7905 | I2C data count register |
| I2CDRR | 0x7906 | I2C data receive register |
| I2CSAR | 0x7907 | I2C slave address register |
| I2CDXR | 0x7908 | I2C data transmit register |
| I2CMDR | 0x7909 | I2C mode register |
| I2CISRC | 0x790A | I2C interrupt source register |
| I2CEMDR | 0x790B | I2C extended mode register |
| I2CPSC | 0x790C | I2C prescaler register |
| I2CFFTX | 0x7920 | I2C FIFO transmit register |
| I2CFFRX | 0x7921 | I2C FIFO receive register |
| I2CRSR | - | I2C receive shift register (not accessible to the CPU) |
| I2CXSR | - | I2C transmit shift register (not accessible to the CPU) |

NOTE: To use the I2C module the system clock to the module must be enabled by setting the appropriate bit in the PCLKR0 register. See *TMS320x280x device System Control and Interrupts Reference Guide* (literature number [SPRU712](#)).

5.1 I2C Mode Register (I2CMDR)

The I2C mode register (I2CMDR) is a 16-bit register that contains the control bits of the I2C module. The bit fields of I2CMDR are shown in [Figure 14](#) and described in [Table 5](#).

Figure 14. I2C Mode Register (I2CMDR)

| | | | | | | | |
|---------|-------|-------|----------|-------|-------|-------|-------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| NACKMOD | FREE | STT | Reserved | STP | MST | TRX | XA |
| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 0 | |
| RM | DLB | IRS | STB | FDF | BC | | |
| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | | |

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 5. I2C Mode Register (I2CMDR) Field Descriptions

| Bit | Field | Value | Description |
|-----|----------|-------|---|
| 15 | NACKMOD | 0 | NACK mode bit. This bit is only applicable when the I2C module is acting as a receiver. In the 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. In the master-receiver mode: The I2C module sends an ACK bit during each acknowledge cycle until the internal data counter counts down to 0. At that point, the I2C module sends a NACK bit to the transmitter. To have a NACK bit sent earlier, you must set the NACKMOD bit. |
| | | 1 | 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. Important: To send a NACK bit in the next acknowledge cycle, you must set NACKMOD before the rising edge of the last data bit. |
| 14 | FREE | 0 | This bit controls the action taken by the I2C module when a debugger breakpoint is encountered. When I2C module is master: 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. When I2C module is slave: A breakpoint forces the I2C module to stop when the current transmission/reception is complete. |
| | | 1 | The I2C module runs free; that is, it continues to operate when a breakpoint occurs. |
| 13 | STT | 0 | 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 6). Note that the STT and STP bits can be used to terminate the repeat mode, and that this bit is not writable when IRS = 0. In the master mode, STT is automatically cleared after the START condition has been generated. |
| | | 1 | In the master mode, setting STT to 1 causes the I2C module to generate a START condition on the I2C-bus. |
| 12 | Reserved | | This reserved bit location is always read as a 0. A value written to this bit has no effect. |
| 11 | STP | 0 | 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 6). Note that the STT and STP bits can be used to terminate the repeat mode, and that this bit is not writable when IRS=0. When in non-repeat mode, at least one byte must be transferred before a stop condition can be generated. STP is automatically cleared after the STOP condition has been generated. |
| | | 1 | STP has been set by the device to generate a STOP condition when the internal data counter of the I2C module counts down to 0. |
| 10 | MST | 0 | 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. Slave mode. The I2C module is a slave and receives the serial clock from the master. |
| | | 1 | Master mode. The I2C module is a master and generates the serial clock on the SCL pin. |

Table 5. I2C Mode Register (I2CMDR) Field Descriptions (continued)

| Bit | Field | Value | Description |
|-----|-------|--------|--|
| 9 | TRX | 0 1 | Transmitter mode bit. When relevant, TRX selects whether the I2C module is in the transmitter mode or the receiver mode. Table 7 summarizes when TRX is used and when it is a don't care. Receiver mode. The I2C module is a receiver and receives data on the SDA pin. Transmitter mode. The I2C module is a transmitter and transmits data on the SDA pin. |
| 8 | XA | 0 1 | Expanded address enable bit. 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). 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). |
| 7 | RM | 0 1 | 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 6). Nonrepeat mode. The value in the data count register (I2CCNT) determines how many bytes are received/transmitted by the I2C module. Repeat mode. A data byte is transmitted each time the I2CDXR register is written to (or until the transmit FIFO is empty when in FIFO mode) until the STP bit is manually set. The value of I2CCNT is ignored. The ARDY bit/interrupt can be used to determine when the I2CDXR (or FIFO) is ready for more data, or when the data has all been sent and the CPU is allowed to write to the STP bit. |
| 6 | DLB | 0 1 | Digital loopback mode bit. The effects of this bit are shown in Figure 15 . Digital loopback mode is disabled. Digital loopback mode is enabled. For proper operation in this mode, the MST bit must be 1. In the digital loopback mode, data transmitted out of I2CDXR is received in I2CDRR after n device cycles by an internal path, where: $n = ((I2C \text{ input clock frequency} / \text{module clock frequency}) \times 8)$ The transmit clock is also the receive clock. The address transmitted on the SDA pin is the address in I2COAR. Note: The free data format (FDF = 1) is not supported in the digital loopback mode. |
| 5 | IRS | 0 1 | I2C module reset bit. 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. The I2C module is enabled. This has the effect of releasing the I2C bus if the I2C peripheral is holding it. |
| 4 | STB | 0 1 | 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 Semiconductors 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, it ignores a START byte from a master, regardless of the value of the STB bit. The I2C module is not in the START byte mode. 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: 1. A START condition 2. A START byte (0000 0001b) 3. A dummy acknowledge clock pulse 4. A repeated START condition Then, as normal, the I2C module sends the slave address that is in I2CSAR. |
| 3 | FDF | 0 1 | Free data format mode bit. Free data format mode is disabled. Transfers use the 7-/10-bit addressing format selected by the XA bit. Free data format mode is enabled. Transfers have the free data (no address) format described in Section 2.5 . The free data format is not supported in the digital loopback mode (DLB=1). |

Table 5. I2C Mode Register (I2CMDR) Field Descriptions (continued)

| Bit | Field | Value | Description |
|-----|-------|-------|--|
| 2-0 | BC | | <p>Bit count bits. BC defines the number of bits (1 to 8) in the next data byte 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 byte has 8 bits. BC does not affect address bytes, which always have 8 bits.</p> <p>Note: If the bit count is less than 8, receive data is right-justified in I2CDRR(7-0), and the other bits of I2CDRR(7-0) are undefined. Also, transmit data written to I2CDXR must be right-justified.</p> |
| | | 000 | 8 bits per data byte |
| | | 001 | 1 bit per data byte |
| | | 010 | 2 bits per data byte |
| | | 011 | 3 bits per data byte |
| | | 100 | 4 bits per data byte |
| | | 101 | 5 bits per data byte |
| | | 110 | 6 bits per data byte |
| | | 111 | 7 bits per data byte |

Table 6. Master-Transmitter/Receiver Bus Activity Defined by the RM, STT, and STP Bits of I2CMDR

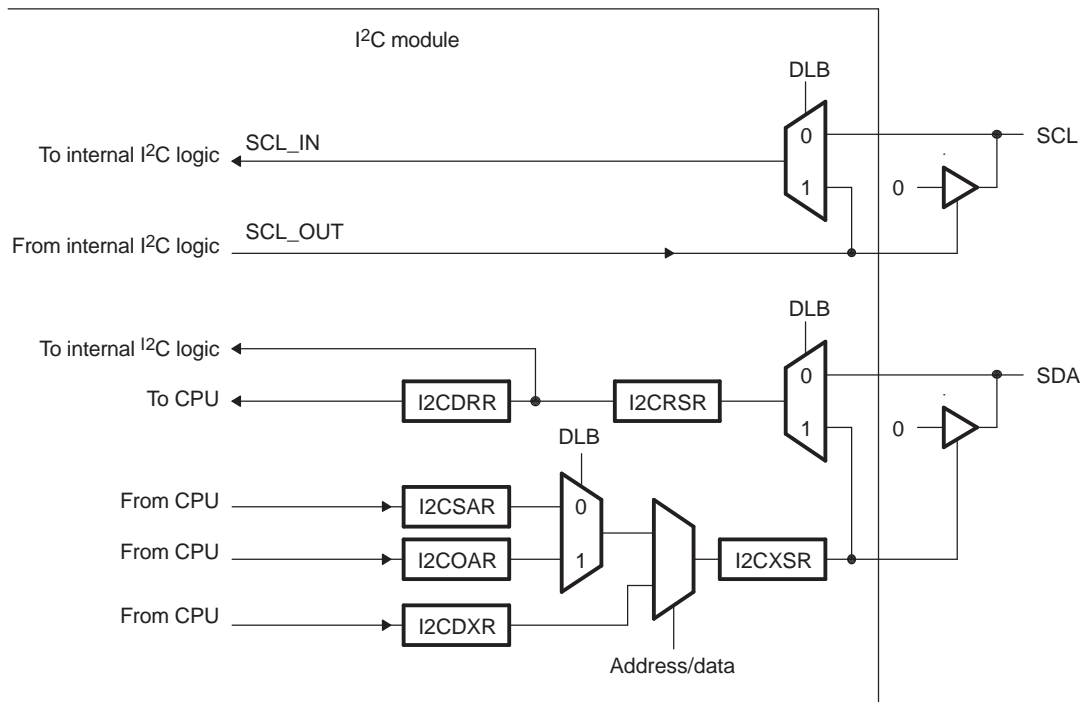
| RM | STT | STP | Bus Activity ⁽¹⁾ | Description |
|----|-----|-----|-----------------------------|--|
| 0 | 0 | 0 | None | No activity |
| 0 | 0 | 1 | P | STOP condition |
| 0 | 1 | 0 | S-A-D..(n)..D. | START condition, slave address, n data bytes (n = value in I2CCNT) |
| 0 | 1 | 1 | S-A-D..(n)..D-P | START condition, slave address, n data bytes, STOP condition (n = value in I2CCNT) |
| 1 | 0 | 0 | None | No activity |
| 1 | 0 | 1 | P | STOP condition |
| 1 | 1 | 0 | S-A-D-D-D. | Repeat mode transfer: START condition, slave address, continuous data transfers until STOP condition or next START condition |
| 1 | 1 | 1 | None | Reserved bit combination (No activity) |

⁽¹⁾ S = START condition; A = Address; D = Data byte; P = STOP condition;

Table 7. How the MST and FDF Bits of I2CMDR Affect the Role of the TRX Bit of I2CMDR

| MST | FDF | I2C Module State | Function of TRX |
|-----|-----|--|--|
| 0 | 0 | In slave mode but not free data format mode | TRX is a don't care. Depending on the command from the master, the I2C module responds as a receiver or a transmitter. |
| 0 | 1 | In slave mode and free data format mode | <p>The free data format mode requires that the I2C module remains the transmitter or the receiver throughout the transfer. TRX identifies the role of the I2C module:</p> <p>TRX = 1: The I2C module is a transmitter. TRX = 0: The I2C module is a receiver.</p> |
| 1 | 0 | In master mode but not free data format mode | <p>TRX = 1: The I2C module is a transmitter. TRX = 0: The I2C module is a receiver.</p> |
| 1 | 1 | In master mode and free data format mode | <p>TRX = 0: The I2C module is a receiver. TRX = 1: The I2C module is a transmitter.</p> |

Figure 15. Pin Diagram Showing the Effects of the Digital Loopback Mode (DLB) Bit



5.2 I²C Extended Mode Register (I2CEMDR)

The I²C extended mode register is shown in Figure 16 and described in Table 8.

Figure 16. I²C Extended Mode Register (I2CEMDR)

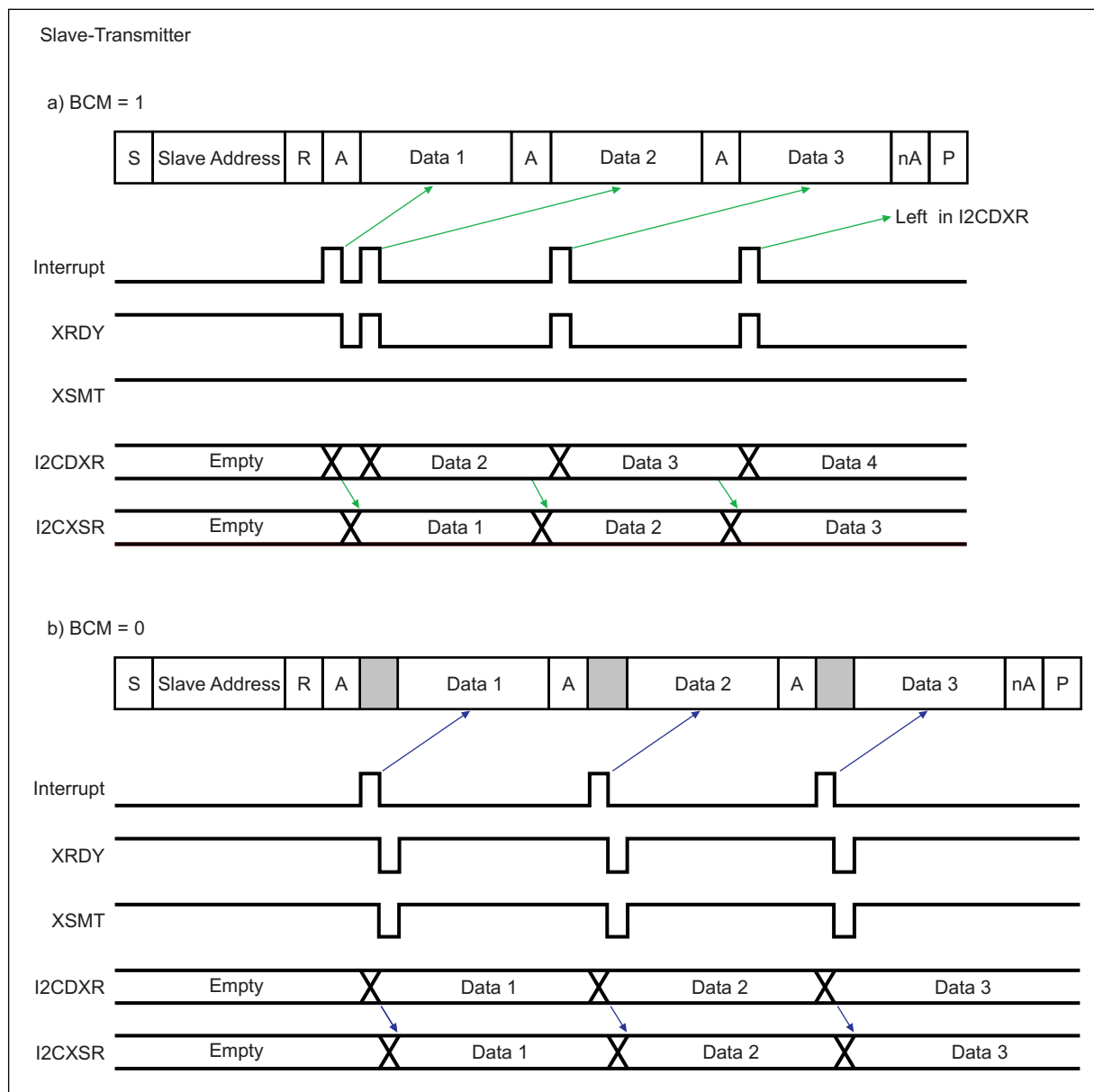
| | | | |
|-----|----------|---|-------|
| 15 | Reserved | 1 | 0 |
| R-0 | | | BCM |
| | | | R/W-1 |

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 8. I²C Extended Mode Register (I2CEMDR) Field Descriptions

| Bit | Field | Value | Description |
|------|----------|-------|--|
| 15-1 | Reserved | | Any writes to these bit(s) must always have a value of 0. |
| 0 | BCM | | Backwards compatibility mode. This bit affects the timing of the transmit status bits (XRDY and XSMT) in the I2CSTR register when in slave transmitter mode. See Figure 17 for details |

Figure 17. BCM Bit, Slave Transmitter Mode



5.3 I2C Interrupt Enable Register (I2CIER)

I2CIER is used by the CPU to individually enable or disable I2C interrupt requests. The bits of I2CIER are shown and described in [Figure 18](#) and [Table 9](#), respectively.

Figure 18. I2C Interrupt Enable Register (I2CIER)

| | | | | | | | | | | | | | | | |
|----------|--|-------|--|-------|--|-------|---|-------|--|-------|--|-------|--|-------|--|
| 15 | | | | | | | 8 | | | | | | | | |
| Reserved | | | | | | | | | | | | | | | |
| R-0 | | | | | | | | | | | | | | | |
| 7 | | 6 | | 5 | | 4 | | 3 | | 2 | | 1 | | 0 | |
| Reserved | | AAS | | SCD | | XRDY | | RRDY | | ARDY | | NACK | | AL | |
| R-0 | | R/W-0 | | R/W-0 | | R/W-0 | | R/W-0 | | R/W-0 | | R/W-0 | | R/W-0 | |

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 9. I2C Interrupt Enable Register (I2CIER) Field Descriptions

| Bit | Field | Value | Description |
|------|----------|--------|---|
| 15-7 | Reserved | | These reserved bit locations are always read as zeros. A value written to this field has no effect. |
| 6 | AAS | 0 1 | Addressed as slave interrupt enable bit Interrupt request disabled Interrupt request enabled |
| 5 | SCD | 0 1 | Stop condition detected interrupt enable bit Interrupt request disabled Interrupt request enabled |
| 4 | XRDY | 0 1 | Transmit-data-ready interrupt enable bit. This bit should not be set when using FIFO mode. Interrupt request disabled Interrupt request enabled |
| 3 | RRDY | 0 1 | Receive-data-ready interrupt enable bit. This bit should not be set when using FIFO mode. Interrupt request disabled Interrupt request enabled |
| 2 | ARDY | 0 1 | Register-access-ready interrupt enable bit Interrupt request disabled Interrupt request enabled |
| 1 | NACK | 0 1 | No-acknowledgment interrupt enable bit Interrupt request disabled Interrupt request enabled |
| 0 | AL | 0 1 | Arbitration-lost interrupt enable bit Interrupt request disabled Interrupt request enabled |

5.4 I2C Status Register (I2CSTR)

The I2C status register (I2CSTR) is a 16-bit register used to determine which interrupt has occurred and to read status information. The bits of I2CSTR are shown and described in [Figure 19](#) and [Table 10](#), respectively.

Figure 19. I2C Status Register (I2CSTR)

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|----------|---------|---------|---------|---------|---------|---------|---------|
| Reserved | SDIR | NACKSNT | BB | RSFULL | XSMT | AAS | AD0 |
| R-0 | R/W1C-0 | R/W1C-0 | R-0 | R-0 | R-1 | R-0 | R-0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Reserved | SCD | XRDY | RRDY | ARDY | NACK | AL | |
| R-0 | R/W1C-0 | R-1 | R/W1C-0 | R/W1C-0 | R/W1C-0 | R/W1C-0 | R/W1C-0 |

LEGEND: R/W = Read/Write; W1C = Write 1 to clear (writing 0 has no effect); R = Read only; -n = value after reset

Table 10. I2C Status Register (I2CSTR) Field Descriptions

| Bit | Field | Value | Description |
|-----|----------|-------|---|
| 15 | Reserved | 0 | This reserved bit location is always read as zeros. A value written to this bit has no effect. |
| 14 | SDIR | 0 | Slave direction bit I2C is not addressed as a slave transmitter. SDIR is cleared by one of the following events: <ul style="list-style-type: none"> It is manually cleared. To clear this bit, write a 1 to it. Digital loopback mode is enabled. A START or STOP condition occurs on the I2C bus. |
| | | 1 | I2C is addressed as a slave transmitter. |
| 13 | NACKSNT | 0 | NACK sent bit. This 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 5.1). NACK not sent. NACKSNT bit is cleared by any one of the following events: <ul style="list-style-type: none"> It is manually cleared. To clear this bit, write a 1 to it. The I2C module is reset (either when 0 is written to the IRS bit of I2CMDR or when the whole device is reset). |
| | | 1 | NACK sent: A no-acknowledge bit was sent during the acknowledge cycle on the I2C-bus. |
| 12 | BB | 0 | Bus busy bit. BB indicates whether the I2C-bus is busy or is free for another data transfer. See the paragraph following the table for more information. Bus free. BB is cleared by any one of the following events: <ul style="list-style-type: none"> The I2C module receives or transmits a STOP bit (bus free). The I2C module is reset. |
| | | 1 | Bus busy: The I2C module has received or transmitted a START bit on the bus. |
| 11 | RSFULL | 0 | Receive shift register full bit. RSFULL indicates an overrun condition during reception. Overrun occurs when new data is received into the shift register (I2CRSR) and the old data has not been read from the receive register (I2CDRR). As new bits arrive from the SDA pin, they overwrite the bits in I2CRSR. The new data will not be copied to ICDRR until the previous data is read. No overrun detected. RSFULL is cleared by any one of the following events: <ul style="list-style-type: none"> I2CDRR is read is read by the CPU. Emulator reads of the I2CDRR do not affect this bit. The I2C module is reset. |
| | | 1 | Overrun detected |
| 10 | XSMT | 0 | Transmit shift register empty bit. XSMT = 0 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. Underflow detected (empty) |
| | | 1 | No underflow detected (not empty). XSMT is set by one of the following events: <ul style="list-style-type: none"> Data is written to I2CDXR. The I2C module is reset |

Table 10. I2C Status Register (I2CSTR) Field Descriptions (continued)

| Bit | Field | Value | Description |
|-----|----------|-------|---|
| 9 | AAS | 0 | Addressed-as-slave bit In the 7-bit addressing mode, the AAS bit is cleared when receiving a NACK, a STOP condition, or a repeated START condition. In the 10-bit addressing mode, the AAS bit is cleared when receiving a NACK, a STOP condition, or by a slave address different from the I2C peripheral's own slave address. |
| | | 1 | The I2C module has recognized its own slave address or an address of all zeros (general call). The AAS bit is also set if the first byte has been received in the free data format (FDF = 1 in I2CMDR). |
| 8 | AD0 | 0 | Address 0 bits AD0 has been cleared by a START or STOP condition. |
| | | 1 | An address of all zeros (general call) is detected. |
| 7-6 | Reserved | 0 | These reserved bit locations are always read as zeros. A value written to this field has no effect. |
| 5 | SCD | 0 | Stop condition detected bit. SCD is set when the I2C sends or receives a STOP condition. STOP condition not detected since SCD was last cleared. SCD is cleared by any one of the following events: <ul style="list-style-type: none"> I2CISRC is read by the CPU when it contains the value 110b (stop condition detected). Emulator reads of the I2CISRC do not affect this bit. SCD is manually cleared. To clear this bit, write a 1 to it. The I2C module is reset. |
| | | 1 | A STOP condition has been detected on the I2C bus. |
| 4 | XRDY | 0 | Transmit-data-ready interrupt flag bit. When not in FIFO mode, XRDY indicates that 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). The CPU can poll XRDY or use the XRDY interrupt request (see Section 3.1). When in FIFO mode, use TXFFINT instead. I2CDXR not ready. XRDY is cleared when data is written to I2CDXR. |
| | | 1 | I2CDXR ready: Data has been copied from I2CDXR to I2CXSR. XRDY is also forced to 1 when the I2C module is reset. |
| 3 | RRDY | 0 | Receive-data-ready interrupt flag bit. When not in FIFO mode, RRDY indicates that the data receive register (I2CDRR) is ready to be read because data has been copied from the receive shift register (I2CRSR) to I2CDRR. The CPU can poll RRDY or use the RRDY interrupt request (see Section 3.1). When in FIFO mode, use RXFFINT instead. I2CDRR not ready. RRDY is cleared by any one of the following events: <ul style="list-style-type: none"> I2CDRR is read by the CPU. Emulator reads of the I2CDRR do not affect this bit. RRDY is manually cleared. To clear this bit, write a 1 to it. The I2C module is reset. |
| | | 1 | I2CDRR ready: Data has been copied from I2CRSR to I2CDRR. |
| 2 | ARDY | 0 | Register-access-ready interrupt flag bit (only applicable when the I2C module is in the master mode). ARDY indicates that the I2C module registers are ready to be accessed because the previously programmed address, data, and command values have been used. The CPU can poll ARDY or use the ARDY interrupt request (see Section 3.1). The registers are not ready to be accessed. ARDY is cleared by any one of the following events: <ul style="list-style-type: none"> The I2C module starts using the current register contents. ARDY is manually cleared. To clear this bit, write a 1 to it. The I2C module is reset. |
| | | 1 | The registers are ready to be accessed. 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). In the repeat mode (RM = 1): ARDY is set at the end of each byte transmitted from I2CDXR. |

Table 10. I2C Status Register (I2CSTR) Field Descriptions (continued)

| Bit | Field | Value | Description |
|-----|-------|-------------------|---|
| 1 | NACK | <p>0</p> <p>1</p> | <p>No-acknowledgment interrupt flag bit. NACK applies when the I2C module is a transmitter (master or slave). NACK indicates whether the I2C module has detected an acknowledge bit (ACK) or a no-acknowledge bit (NACK) from the receiver. The CPU can poll NACK or use the NACK interrupt request (see Section 3.1).</p> <p>ACK received/NACK not received. This bit is cleared by any one of the following events:</p> <ul style="list-style-type: none"> • An acknowledge bit (ACK) has been sent by the receiver. • NACK is manually cleared. To clear this bit, write a 1 to it. • The CPU reads the interrupt source register (I2CISRC) and the register contains the code for a NACK interrupt. Emulator reads of the I2CISRC do not affect this bit. • The I2C module is reset. <p>NACK bit received. The hardware detects that a no-acknowledge (NACK) bit has been received.</p> <p>Note: While the I2C module performs a general call transfer, NACK is 1, even if one or more slaves send acknowledgment.</p> |
| 0 | AL | <p>0</p> <p>1</p> | <p>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 3.1).</p> <p>Arbitration not lost. AL is cleared by any one of the following events:</p> <ul style="list-style-type: none"> • AL is manually cleared. To clear this bit, write a 1 to it. • The CPU reads the interrupt source register (I2CISRC) and the register contains the code for an AL interrupt. Emulator reads of the I2CISRC do not affect this bit. • The I2C module is reset. <p>Arbitration lost. AL is set by any one of the following events:</p> <ul style="list-style-type: none"> • 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. <p>When AL becomes 1, the MST and STP bits of I2CMDR are cleared, and the I2C module becomes a slave-receiver.</p> |

The I2C peripheral cannot detect a START or STOP condition when it is in reset, i.e. the IRS bit is set to 0. Therefore, the BB bit will remain in the state it was at when the peripheral was placed in reset. The BB bit will stay in that state until the I2C peripheral is taken out of reset, i.e. the IRS bit is set to 1, and a START or STOP condition is detected on the I2C bus.

Follow these steps before initiating the first data transfer with I2C :

1. After taking the I2C peripheral out of reset by setting the IRS bit to 1, wait a certain period to detect the actual bus status before starting the first data transfer. Set this period larger than the total time taken for the longest data transfer in the application. By waiting for a period of time after I2C comes out of reset, users can ensure that at least one START or STOP condition will have occurred on the I2C bus, and been captured by the BB bit. After this period, the BB bit will correctly reflect the state of the I2C bus.
2. Check the BB bit and verify that BB=0 (bus not busy) before proceeding.
3. Begin data transfers.

Not resetting the I2C peripheral in between transfers ensures that the BB bit reflects the actual bus status. If users must reset the I2C peripheral in between transfers, repeat steps 1 through 3 every time the I2C peripheral is taken out of reset.

5.5 I2C Interrupt Source Register (I2CISRC)

The I2C interrupt source register (I2CISRC) is a 16-bit register 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](#).

Figure 20. I2C Interrupt Source Register (I2CISRC)

| | | | | | | | |
|----------|----|----|---|----------|---|---------|---|
| 15 | 12 | 11 | 8 | 7 | 3 | 2 | 0 |
| Reserved | | | | Reserved | | INTCODE | |
| R-0 | | | | R/W-0 | | R-0 | |

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 11. I2C Interrupt Source Register (I2CISRC) Field Descriptions

| Bit | Field | Value | Description |
|-------|----------|-------|--|
| 15-12 | Reserved | | These reserved bit locations are always read as zeros. A value written to this field has no effect. |
| 11-8 | Reserved | | These reserved bit locations should always be written as zeros. |
| 7-3 | Reserved | | These reserved bit locations are always read as zeros. A value written to this field has no effect. |
| 2-0 | INTCODE | | Interrupt code bits. The binary code in INTCODE indicates the event that generated an I2C interrupt. |
| | | 000 | None |
| | | 001 | Arbitration lost |
| | | 010 | No-acknowledgment condition detected |
| | | 011 | Registers ready to be accessed |
| | | 100 | Receive data ready |
| | | 101 | Transmit data ready |
| | | 110 | Stop condition detected |
| | | 111 | Addressed as slave |
| | | | A CPU read will clear this field. If another lower priority interrupt is pending and enabled, the value corresponding to that interrupt will then be loaded. Otherwise, the value will stay cleared. |
| | | | In the case of an arbitration lost, a no-acknowledgment condition detected, or a stop condition detected, a CPU read will also clear the associated interrupt flag bit in the I2CSTR register. |
| | | | Emulator reads will not affect the state of this field or of the status bits in the I2CSTR register. |

5.6 I2C Prescaler Register (I2CPSC)

The I2C prescaler register (I2CPSC) is a 16-bit register (see [Figure 21](#)) used for dividing down the I2C input clock to obtain the desired module clock for the operation of the I2C module. See the device-specific data manual for the supported range of values for the module clock frequency. [Table 12](#) lists the bit descriptions. For more details about the module clock, see [Section 1.3](#).

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

Figure 21. I2C Prescaler Register (I2CPSC)

| | | | |
|----------|---|---|---|
| 15 | 8 | 7 | 0 |
| Reserved | | | |
| IPSC | | | |
| R-0 | | | |
| R/W-0 | | | |

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 12. I2C Prescaler Register (I2CPSC) Field Descriptions

| Bit | Field | Value | Description |
|------|----------|-------|---|
| 15-8 | Reserved | | These reserved bit locations are always read as zeros. A value written to this field has no effect. |
| 7-0 | IPSC | | <p>I2C prescaler divide-down value.</p> <p>IPSC determines how much the CPU clock is divided to create the module clock of the I2C module:</p> $\text{module clock frequency} = \text{I2C input clock frequency} / (\text{IPSC} + 1)$ <p>Note: IPSC must be initialized while the I2C module is in reset (IRS = 0 in I2CMDR).</p> |

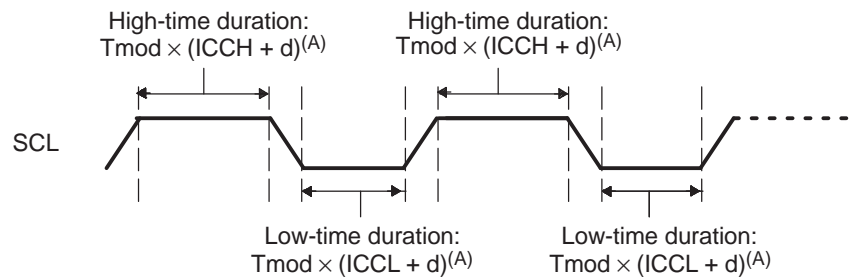
NOTE: To meet all of the I2C protocol timing specifications, the module clock must be configured between 7-12 MHz.

5.7 I2C Clock Divider Registers (I2CCLKL and I2CCLKH)

As explained in [Section 1.3](#), when the I2C module is a master, the module clock is divided down for use as the master clock on the SCL pin. As shown in [Figure 22](#), the shape of the master clock depends on two divide-down values:

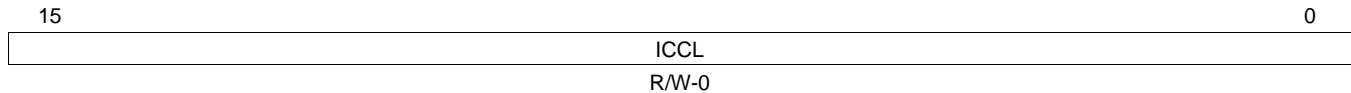
- ICCL in I2CCLKL (summarized by [Figure 23](#) and [Table 13](#)). For each master clock cycle, ICCL determines the amount of time the signal is low.
- ICCH in I2CCLKH (summarized by [Figure 24](#) and [Table 14](#)). For each master clock cycle, ICCH determines the amount of time the signal is high.

Figure 22. The Roles of the Clock Divide-Down Values (ICCL and ICCH)



A As described in [Section 5.7.1](#), T_{mod} is the module clock period, and d is 5, 6, or 7.

Figure 23. I2C Clock Low-Time Divider Register (I2CCLKL)

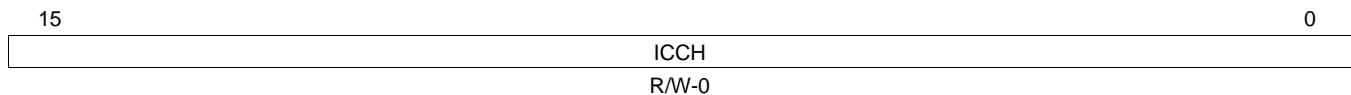


LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 13. I2C Clock Low-Time Divider Register (I2CCLKL) Field Description

| Bit | Field | Value | Description |
|------|-------|-------|---|
| 15-0 | ICCL | | Clock low-time divide-down value. To produce the low-time duration of the master clock, the period of the module clock is multiplied by $(\text{ICCL} + d)$. d is 5, 6, or 7 as described in Section 5.7.1 . Note: These bits must be set to a non-zero value for proper I2C clock operation. |

Figure 24. I2C Clock High-Time Divider Register (I2CCLKH)



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 14. I2C Clock High-Time Divider Register (I2CCLKH) Field Description

| Bit | Field | Value | Description |
|------|-------|-------|---|
| 15-0 | ICCH | | Clock high-time divide-down value. To produce the high-time duration of the master clock, the period of the module clock is multiplied by $(\text{ICCH} + d)$. d is 5, 6, or 7 as described in Section 5.7.1 . Note: These bits must be set to a non-zero value for proper I2C clock operation. |

5.7.1 Formula for the Master Clock Period

The period of the master clock (T_{mst}) is a multiple of the period of the module clock (T_{mod}):

$$T_{mst} = T_{mod} \times [(ICCL + d) + (ICCH + d)]$$

$$T_{mst} = \frac{(IPSC + 1) [(ICCL + d) + (ICCH + d)]}{I2C \text{ input clock frequency}}$$

where d depends on the divide-down value $IPSC$, as shown in Table 15. $IPSC$ is described in Section 5.6.

Table 15. Dependency of Delay d on the Divide-Down Value $IPSC$

| IPSC | d |
|----------------|---|
| 0 | 7 |
| 1 | 6 |
| Greater than 1 | 5 |

5.8 I2C Slave Address Register (I2CSAR)

The I2C slave address register (I2CSAR) is a register for storing the next slave address that will be transmitted by the I2C module when it is a master. It is a 16-bit register with the format shown in Figure 25. As described in Table 16, the SAR field of 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; write 0s to bits 9-7.

Figure 25. I2C Slave Address Register (I2CSAR)

| | | | |
|----------|----|----------|---|
| 15 | 10 | 9 | 0 |
| Reserved | | SAR | |
| R-0 | | R/W-3FFh | |

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 16. I2C Slave Address Register (I2CSAR) Field Descriptions

| Bit | Field | Value | Description |
|-------|----------|-----------|--|
| 15-10 | Reserved | | These reserved bit locations are always read as zeros. A value written to this field has no effect. |
| 9-0 | SAR | 00h-7Fh | 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. Write 0s to bits 9-7. |
| | | 000h-3FFh | In 10-bit addressing mode ($XA = 1$ in I2CMDR): Bits 9-0 provide the 10-bit slave address that the I2C module transmits when it is in the master-transmitter mode. |

5.9 I2C Own Address Register (I2COAR)

The I2C own address register (I2COAR) is a 16-bit register. Figure 26 shows the format of I2COAR, and Table 17 describes its bit fields. The I2C module uses this register 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; write 0s to bits 9-7.

Figure 26. I2C Own Address Register (I2COAR)

| | | | |
|----------|----|-------|---|
| 15 | 10 | 9 | 0 |
| Reserved | | OAR | |
| R-0 | | R/W-0 | |

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 17. I2C Own Address Register (I2COAR) Field Descriptions

| Bit | Field | Value | Description |
|-------|----------|--------------------------|--|
| 15-10 | Reserved | | These reserved bit locations are always read as zeros. A value written to this field has no effect. |
| 9-0 | OAR | 00h-7Fh 000h-3FFh | In 7-bit addressing mode (XA = 0 in I2CMDR): Bits 6-0 provide the 7-bit slave address of the I2C module. Write 0s to bits 9-7. In 10-bit addressing mode (XA = 1 in I2CMDR): Bits 9-0 provide the 10-bit slave address of the I2C module. |

5.10 I2C Data Count Register (I2CCNT)

I2CCNT is a 16-bit register used to indicate how many data bytes to transfer when the I2C module is configured as a transmitter, or to receive when configured as a master receiver. In the repeat mode (RM = 1), I2CCNT is not used. The bits of I2CCNT are shown and described in [Figure 27](#) and [Table 18](#), respectively.

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

Figure 27. I2C Data Count Register (I2CCNT)

| | | | | | | | | | | | | | | | | |
|-------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|---|
| 15 | | | | | | | | | | | | | | | | 0 |
| ICDC | | | | | | | | | | | | | | | | |
| R/W-0 | | | | | | | | | | | | | | | | |

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 18. I2C Data Count Register (I2CCNT) Field Descriptions

| Bit | Field | Value | Description |
|------|-------|--------------------------|--|
| 15-0 | ICDC | 0000h 0001h-FFFFh | Data count value. ICDC indicates the number of data bytes to transfer or receive. The value in I2CCNT is a don't care when the RM bit in I2CMDR is set to 1. The start value loaded to the internal data counter is 65536. The start value loaded to internal data counter is 1-65535. |

5.11 I2C Data Receive Register (I2CDRR)

I2CDRR (see [Figure 28](#) and [Table 19](#)) is a 16-bit register used by the CPU to read received data. The I2C module can receive a data byte with 1 to 8 bits. The number of bits is selected with the bit count (BC) bits in I2CMDR. One bit at a time is shifted in from the SDA pin to the receive shift register (I2CRSR). When a complete data byte has been received, the I2C module copies the data byte from I2CRSR to I2CDRR. The CPU cannot access I2CRSR directly.

If a data byte with fewer than 8 bits is in I2CDRR, the data value is right-justified, and the other bits of I2CDRR(7-0) are undefined. For example, if BC = 011 (3-bit data size), the receive data is in I2CDRR(2-0), and the content of I2CDRR(7-3) is undefined.

When in the receive FIFO mode, the I2CDRR register acts as the receive FIFO buffer.

Figure 28. I2C Data Receive Register (I2CDRR)

| | | | | | | | | | | | | | | | | |
|----------|--|--|--|--|--|--|--|------|--|--|--|--|--|--|--|---|
| 15 | | | | | | | | | | | | | | | | 0 |
| Reserved | | | | | | | | DATA | | | | | | | | |
| R-0 | | | | | | | | R-0 | | | | | | | | |

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 19. I2C Data Receive Register (I2CDRR) Field Descriptions

| Bit | Field | Value | Description |
|------|----------|-------|---|
| 15-8 | Reserved | | These reserved bit locations are always read as zeros. A value written to this field has no effect. |
| 7-0 | DATA | | Receive data |

5.12 I2C Data Transmit Register (I2CDXR)

The CPU writes transmit data to I2CDXR (see [Figure 29](#) and [Table 20](#)). This 16-bit register accepts a data byte with 1 to 8 bits. Before writing to I2CDXR, specify how many bits are in a data byte by loading the appropriate value into the bit count (BC) bits of I2CMDR. When writing a data byte with fewer than 8 bits, make sure the value is right-aligned in I2CDXR.

After a data byte is written to I2CDXR, the I2C module copies the data byte to the transmit shift register (I2CXSR). The CPU cannot access I2CXSR directly. From I2CXSR, the I2C module shifts the data byte out on the SDA pin, one bit at a time.

When in the transmit FIFO mode, the I2CDXR register acts as the transmit FIFO buffer.

Figure 29. I2C Data Transmit Register (I2CDXR)

| | | | |
|----------|---|---|---|
| 15 | 8 | 7 | 0 |
| Reserved | | | |
| DATA | | | |
| R-0 | | | |
| R/W-0 | | | |

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 20. I2C Data Transmit Register (I2CDXR) Field Descriptions

| Bit | Field | Value | Description |
|------|----------|-------|---|
| 15-8 | Reserved | | These reserved bit locations are always read as zeros. A value written to this field has no effect. |
| 7-0 | DATA | | Transmit data |

5.13 I2C Transmit FIFO Register (I2CFFTX)

The I2C transmit FIFO register (I2CFFTX) is a 16-bit register that contains the I2C FIFO mode enable bit as well as the control and status bits for the transmit FIFO mode of operation on the I2C peripheral. The bit fields are shown in [Figure 30](#) and described in [Table 21](#).

Figure 30. I2C Transmit FIFO Register (I2CFFTX)

| | | | | | | | |
|----------|------------|----------|---------|---------|---------|---------|---------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| Reserved | I2CFFEN | TXFFRST | TXFFST4 | TXFFST3 | TXFFST2 | TXFFST1 | TXFFST0 |
| R-0 | R/W-0 | R/W-0 | R-0 | R-0 | R-0 | R-0 | R-0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXFFINT | TXFFINTCLR | TXFFIENA | TXFFIL4 | TXFFIL3 | TXFFIL2 | TXFFIL1 | TXFFIL0 |
| R-0 | R/W1C-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 21. I2C Transmit FIFO Register (I2CFFTX) Field Descriptions

| Bit | Field | Value | Description |
|-----|----------|-------|--|
| 15 | Reserved | | Reserved. Reads will return a 0, writes have no effect. |
| 14 | I2CFFEN | | I2C FIFO mode enable bit. This bit must be enabled for either the transmit or the receive FIFO to operate correctly. |
| | | 0 | Disable the I2C FIFO mode. |
| | | 1 | Enable the I2C FIFO mode. |

Table 21. I2C Transmit FIFO Register (I2CFFTX) Field Descriptions (continued)

| Bit | Field | Value | Description |
|------|------------|-------------------------|---|
| 13 | TXFFRST | 0 1 | I2C transmit FIFO reset bit. Reset the transmit FIFO pointer to 0000 and hold the transmit FIFO in the reset state. Enable the transmit FIFO operation. |
| 12-8 | TXFFST4-0 | 10000 0xxxx 00000 | Contains the status of the transmit FIFO: Transmit FIFO contains 16 bytes. Transmit FIFO contains xxxx bytes. Transmit FIFO is empty. Note: Since these bits are reset to zero, the transmit FIFO interrupt flag will be set when the transmit FIFO operation is enabled and the I2C is taken out of reset. This will generate a transmit FIFO interrupt if enabled. To avoid any detrimental effects from this, write a one to the TXFFINTCLR once the transmit FIFO operation is enabled and the I2C is taken out of reset. |
| 7 | TXFFINT | 0 1 | Transmit FIFO interrupt flag. This bit cleared by a CPU write of a 1 to the TXFFINTCLR bit. If the TXFFIENA bit is set, this bit will generate an interrupt when it is set. Transmit FIFO interrupt condition has not occurred. Transmit FIFO interrupt condition has occurred. |
| 6 | TXFFINTCLR | 0 1 | Transmit FIFO interrupt flag clear bit. Writes of zeros have no effect. Reads return a 0. Writing a 1 to this bit clears the TXFFINT flag. |
| 5 | TXFFIENA | 0 1 | Transmit FIFO interrupt enable bit. Disabled. TXFFINT flag does not generate an interrupt when set. Enabled. TXFFINT flag does generate an interrupt when set. |
| 4-0 | TXFFIL4-0 | | Transmit FIFO interrupt level. These bits set the status level that will set the transmit interrupt flag. When the TXFFST4-0 bits reach a value equal to or less than these bits, the TXFFINT flag will be set. This will generate an interrupt if the TXFFIENA bit is set. |

5.14 I2C Receive FIFO Register (I2CFFRX)

The I2C receive FIFO register (I2CFFRX) is a 16-bit register that contains the control and status bits for the receive FIFO mode of operation on the I2C peripheral. The bit fields are shown in [Figure 31](#) and described in [Table 22](#).

Figure 31. I2C Receive FIFO Register (I2CFFRX)

| | | | | | | | |
|----------|------------|----------|---------|---------|---------|---------|---------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| Reserved | | RXFFRST | RXFFST4 | RXFFST3 | RXFFST2 | RXFFST1 | RXFFST0 |
| R-0 | | R/W-0 | R-0 | R-0 | R-0 | R-0 | R-0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXFFINT | RXFFINTCLR | RXFFIENA | RXFFIL4 | RXFFIL3 | RXFFIL2 | RXFFIL1 | RXFFIL0 |
| R-0 | R/W1C-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 22. I2C Receive FIFO Register (I2CFFRX) Field Descriptions

| Bit | Field | Value | Description |
|-------|-----------|----------------|--|
| 15-14 | Reserved | | Reserved. Reads will return a 0, writes have no effect. |
| 13 | RXFFRST | 0 1 | I2C receive FIFO reset bit Reset the receive FIFO pointer to 0000 and hold the receive FIFO in the reset state. Enable the receive FIFO operation. |
| 12-8 | RXFFST4-0 | 10000 0xxxx | Contains the status of the receive FIFO: Receive FIFO contains 16 bytes Receive FIFO contains xxxx bytes |

Table 22. I2C Receive FIFO Register (I2CFFRX) Field Descriptions (continued)

| Bit | Field | Value | Description |
|-----|------------|--------|--|
| | | 00000 | Receive FIFO is empty. |
| 7 | RXFFINT | 0 1 | Receive FIFO interrupt flag. This bit cleared by a CPU write of a 1 to the RXFFINTCLR bit. If the RXFFIENA bit is set, this bit will generate an interrupt when it is set. Receive FIFO interrupt condition has not occurred. Receive FIFO interrupt condition has occurred. |
| 6 | RXFFINTCLR | 0 1 | Receive FIFO interrupt flag clear bit. Writes of zeros have no effect. Reads return a zero. Writing a 1 to this bit clears the RXFFINT flag. |
| 5 | RXFFIENA | 0 1 | Receive FIFO interrupt enable bit. Disabled. RXFFINT flag does not generate an interrupt when set. Enabled. RXFFINT flag does generate an interrupt when set. |
| 4 | RXFFIL4-0 | | Receive FIFO interrupt level. These bits set the status level that will set the receive interrupt flag. When the RXFFST4-0 bits reach a value equal to or greater than these bits, the RXFFINT flag is set. This will generate an interrupt if the RXFFIENA bit is set. Note: Since these bits are reset to zero, the receive FIFO interrupt flag will be set if the receive FIFO operation is enabled and the I2C is taken out of reset. This will generate a receive FIFO interrupt if enabled. To avoid this, modify these bits on the same instruction as or prior to setting the RXFFRST bit. |

Appendix A Revision History

This document has been revised to include the following technical change(s).

Table 23. Changes in This Revision

| Location | Changes, Additions, Deletions |
|--------------------------|---|
| Table 10 | Changed the description of bit 12 (BB) in the I2C Status register (I2CSTR). Bit is now read-only. |
| Table 5 | For bit 11 (STP), added this sentence to the description: " When in non-repeat mode, at least one byte must be transferred before a stop condition can be generated." |

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 TI 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. Information of third parties may be subject to additional restrictions.

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.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

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

Products

| | |
|-----------------------------|--|
| Audio | www.ti.com/audio |
| Amplifiers | amplifier.ti.com |
| Data Converters | dataconverter.ti.com |
| DLP® Products | www.dlp.com |
| DSP | dsp.ti.com |
| Clocks and Timers | www.ti.com/clocks |
| Interface | interface.ti.com |
| Logic | logic.ti.com |
| Power Mgmt | power.ti.com |
| Microcontrollers | microcontroller.ti.com |
| RFID | www.ti-rfid.com |
| RF/IF and ZigBee® Solutions | www.ti.com/lprf |

Applications

| | |
|-------------------------------|--|
| Communications and Telecom | www.ti.com/communications |
| Computers and Peripherals | www.ti.com/computers |
| Consumer Electronics | www.ti.com/consumer-apps |
| Energy and Lighting | www.ti.com/energy |
| Industrial | www.ti.com/industrial |
| Medical | www.ti.com/medical |
| Security | www.ti.com/security |
| Space, Avionics and Defense | www.ti.com/space-avionics-defense |
| Transportation and Automotive | www.ti.com/automotive |
| Video and Imaging | www.ti.com/video |
| Wireless | www.ti.com/wireless-apps |

TI E2E Community Home Page

e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2011, Texas Instruments Incorporated