# KeyStone Architecture
# Inter-IC Control Bus ($I^2C$)

# User Guide

Literature Number: SPRUGV3
August 2011

# Release History

| Release | Date | Chapter/Topic | Description/Comments |
|---|---|---|---|
| SPRUGV3 | August 2011 | All | First release |
| | | | |

# Contents

## List of Tables

## List of Figures

# Preface

## About This Manual

This document describes the inter-integrated circuit ($I^2C$) module in the KeyStone Architecture Digital Signal Processor (DSP). The $I^2C$ provides an interface between the KeyStone device and other devices compliant with Philips Semiconductors Inter-IC bus ($I^2C$-bus) specification version 2.1 and connected by way of an $I^2C$-bus. This document assumes the reader is familiar with the $I^2C$-bus specification.

## Notational Conventions

This document uses the following conventions.

- Hexadecimal numbers are shown with the suffix h. For example, the following number is 40 hexadecimal (decimal 64): 40h.
- Registers in this document are shown in figures and described in tables.
  - Each register figure shows a rectangle divided into fields that represent the fields of the register. Each field is labelled 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

| | |
|---|---|
| *C66x CorePac User Guide* | SPRUGW0 |
| *C66x CPU and Instruction Set Reference Guide* | SPRUGH7 |

## Trademarks

TMS320C66x is a trademark of Texas Instruments Incorporated.

All other brand names and trademarks mentioned in this document are the property of Texas Instruments Incorporated or their respective owners, as applicable.

Chapter 1

# Introduction

**IMPORTANT NOTE—**The information in this document should be used in conjunction with information in the device-specific Keystone Architecture data manual that applies to the part number of your device.

## 1.1 Introduction

This document describes the inter-integrated circuit ($I^2C$) module in the KeyStone Architecture Digital Signal Processor (DSP). This document assumes the reader is familiar with the $I^2C$-bus specification.

## 1.2 Purpose of the Peripheral

The $I^2C$ module provides an interface between the KeyStone device and other devices compliant with the $I^2C$-bus specification and connected by way of an $I^2C$-bus. External components attached to this 2-wire serial bus can transmit and receive up to 8-bit wide data to and from the device through the $I^2C$ module.

## 1.3 Features

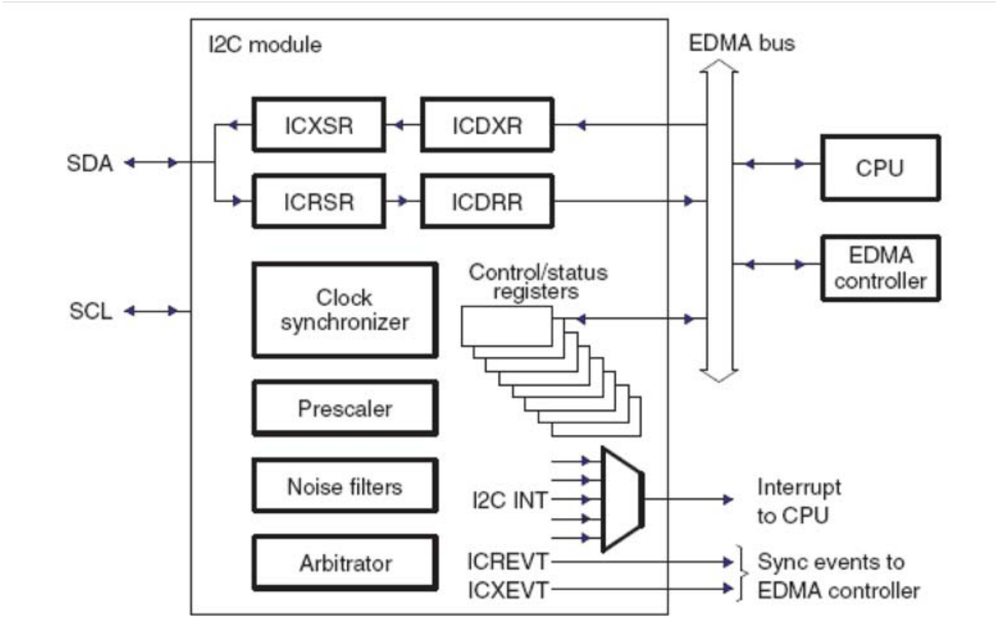
The $I^2C$ module has the following features:

- Compliance with the Philips Semiconductors $I^2C$-bus specification (version 2.1):
  - Support for byte format transfer
  - 7-bit and 10-bit addressing modes
  - General call
  - START byte mode
  - Support for multiple master-transmitters and slave-receivers mode
  - Support for multiple slave-transmitters and master-receivers mode
  - Combined master transmit/receive and receive/transmit mode
  - $I^2C$ data transfer rate of from 10 kbps up to 400 kbps (Philips $I^2C$ rate)
- 2 to 7 bit format transfer
- Free data format mode
- One read DMA event and one write DMA event that can be used by the DMA
- Seven interrupts that can be used by the CPU
- Interface to V-bus (32-bit synchronously slave bus)
- Module enable/disable capability.

## 1.4 Functional Block Diagram

A block diagram of the I$^2$C module is shown in Figure 1-1. Detailed information about the architecture of the I$^2$C module is in Section 2.

**Figure 1-1 Block Diagram**




## 1.5 Industry Standard(s) Compliance Statement

The I$^2$C module is compliant with the Philips Semiconductors Inter-IC bus (I$^2$C-bus) specification version 2.1.

Chapter 2

# Peripheral Architecture

## 2.1 Peripheral Architecture

The I$^2$C module consists of the following primary blocks:

- A serial interface: one data pin (SDA) and one clock pin (SCL)
- Data registers to temporarily hold receive data and transmit data traveling between the SDA pin and the CPU or the EDMA controller
- Control and status registers
- An EDMA bus interface to enable the CPU and the EDMA controller to access the I$^2$C module registers
- A clock synchronizer to synchronize the I$^2$C input clock (from the 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 I$^2$C module
- A noise filter on each of the two pins, SDA and SCL
- An arbitrator to handle arbitration between the I$^2$C module (when it is a master) and another master
- Interrupt generation logic, so that an interrupt can be sent to the CPU
- EDMA event generation logic, so that activity in the EDMA controller can be synchronized to data reception and data transmission in the I$^2$C module

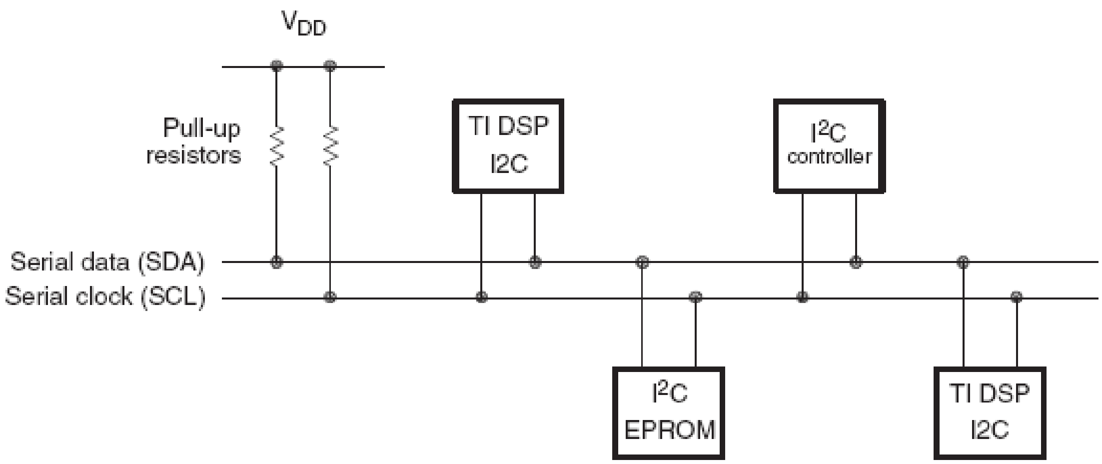
Figure 1-1 on page 1-3 shows the four registers used for transmission and reception. The CPU or the EDMA controller writes data for transmission to ICDXR and reads received data from ICDRR. When the I$^2$C module is configured as a transmitter, data written to ICDXR is copied to ICXSR and shifted out on the SDA pin one bit a time. When the I$^2$C module is configured as a receiver, received data is shifted into ICRSR and then copied to ICDRR.

## 2.2 Bus Structure

Figure 1-1 on page 1-3 shows how the I$^2$C module is connected to the I$^2$C bus. The I$^2$C bus is a multi-master bus that supports a multi-master mode. This allows more than one device capable of controlling the bus that is connected to it. Each I$^2$C device is recognized by a unique address and can operate as either transmitter or receiver depending on the function of the device. In addition to being a transmitter or receiver, devices connected to the I$^2$C bus also can be considered as master or slave when performing data transfers.

Note that a master device is the device that initiates a data transfer on the bus and generates the clock signals to permit that transfer. During this transfer, any device addressed by this master is considered a slave. Figure 2-1 shows an example of multiple I$^2$C modules connected for a two-way transfer from one device to other devices.

**Figure 2-1 Multiple I²C Modules Connected**




## 2.3 Clock Generation

The $I^2C$ module is recommended to operate with a module clock in a range of 7 to 12 MHz. This clock is generated via the $I^2C$ prescaler block. The $I^2C$ prescaler register (ICPSC) is used to divide-down the input clock to obtain a clock within the specified range for the $I^2C$ module.

As shown in Figure 2-2, the PLL1 receives a signal from an external clock source and produces an $I^2C$ input clock with a programmed frequency. The clock is then divided twice more inside the $I^2C$ module to produce to the module clock and the master clock.

The module clock determines the frequency at which the $I^2C$ module operates. Figure 2-2 shows how this clock is generated. A programmable prescaler in the $I^2C$ module divides down the $I^2C$ input clock to produce the module clock. To specify the divide-down value, initialize the IPSC field of the $I^2C$ prescaler register (ICPSC). The resulting frequency is:

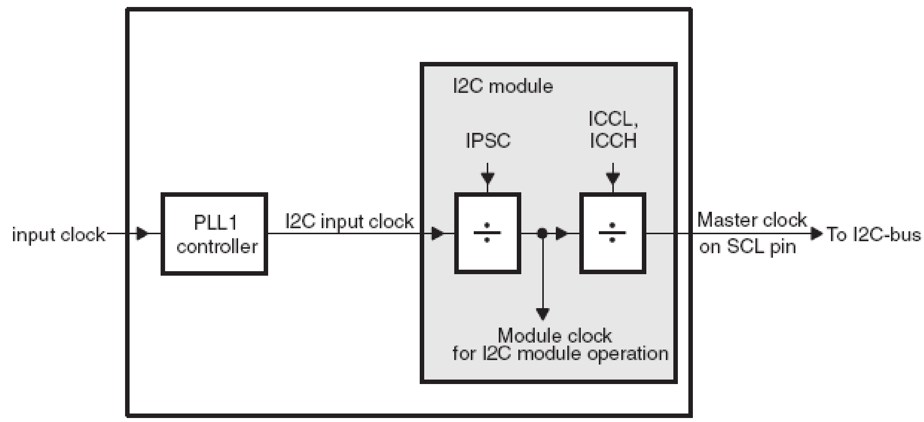
$$\text{module clock frequency} = \frac{I^2C\text{ input clock frequency}}{(IPSC + 1)}$$

The prescaler must be initialized only while the $I^2C$ module is in the reset state (IRS = 0 in ICMDR). 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 $I^2C$ module is configured to be a master on the $I^2C$-bus. This clock controls the timing of communication between the $I^2C$ module and a slave. As shown in Figure 2-2, a second clock divider in the $I^2C$ module divides down the module clock to produce the master clock. The clock divider uses the ICCL value of ICCLKL to divide-down the low portion of the module clock signal and uses the ICCH value of ICCLKH to divide-down the high portion of the module clock signal. The resulting frequency is:

$$\text{master clock frequency} \quad \frac{\text{module clock frequency}}{(ICCL6) + (ICCH6)}$$

**Figure 2-2 Clocking Diagram for the I²C Module**

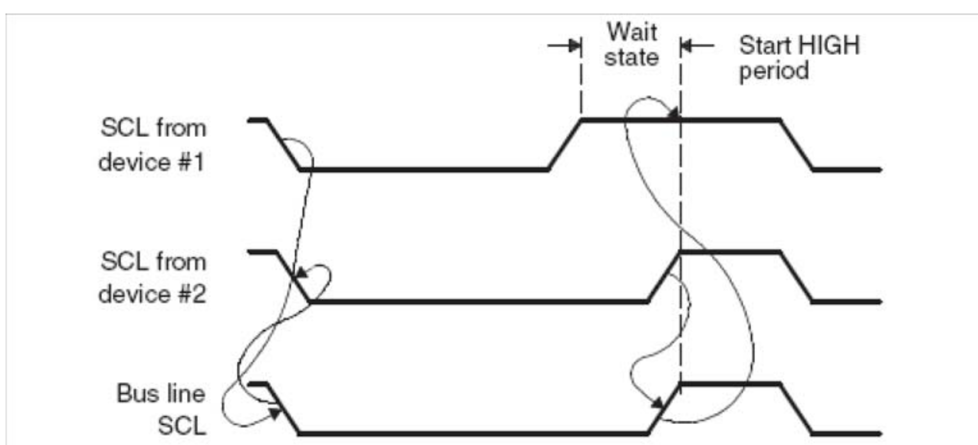



## 2.4 Clock Synchronization

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 2-3 illustrates the clock synchronization. The wired-AND property of SCL means that a device that first generates a low period on SCL (device #1) overrules the other devices. At this high-to-low transition, the clock generators of the other devices are forced to start their own low period. The SCL is held low by the device with the longest low period. The other devices that finish their low periods must wait for SCL to be released, before starting their high periods. A synchronized signal on SCL is obtained, where the slowest device determines the length of the low period and the fastest device determines the length of the high period.

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

**Figure 2-3 Synchronization of Two I²C Clock Generators During Arbitration**



## 2.5 Signal Descriptions

For data communication, the $I^2C$ module has a serial data pin (SDA) and a serial clock pin (SCL), as shown in Figure 1-1 on page 1-3. These two pins carry information between the KeyStone device and other devices connected to the $I^2C$-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.

For additional timing and electrical specifications for these pins, see the device-specific data manual.

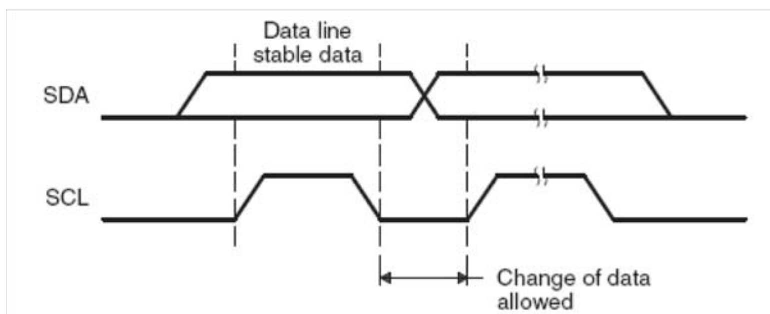
### 2.5.1 Input and Output Voltage Levels

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

### 2.5.2 Data Validity

The data on SDA must be stable during the high period of the clock (see Figure 2-4). The high or low state of the data line, SDA, can change only when the clock signal on SCL is low.

**Figure 2-4 Bit Transfer on the $I^2C$-Bus**

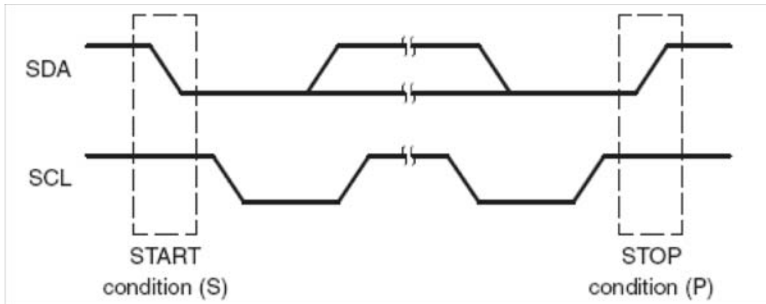



## 2.6 START and STOP Conditions

START and STOP conditions can be generated by the $I^2C$ module when the module is configured to be a master on the $I^2C$-bus. As shown in Figure 2-5:

- 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 2-5 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 ICSTR is 1. Between a STOP condition and the next START condition, the bus is considered free, and BB is 0.

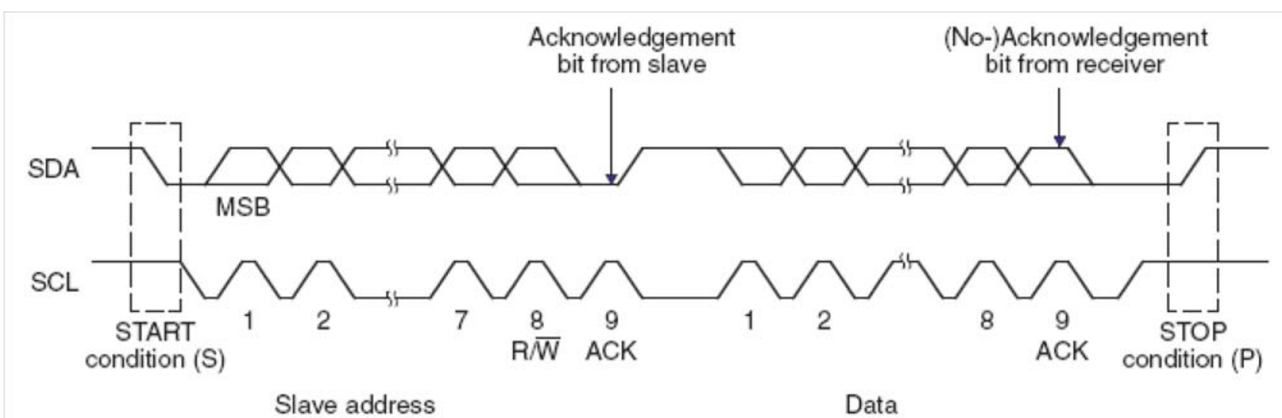
For the I²C module to start a data transfer with a START condition, the master mode (MST) bit and the START condition (STT) bit in ICMDR must both be 1. For the I²C module to end a data transfer with a STOP condition, the STOP condition (STP) bit must be set to 1. When BB is set to 1 and STT is set to 1, a repeated START condition is generated. For a description of ICMDR (including the MST, STT, and STP bits), see Section 3.10.

## 2.7 Serial Data Formats

Figure 2-6 shows an example of a data transfer on the I²C-bus. The I²C module supports 1-bit to 8-bit data values. Figure 2-6 is shown in an 8-bit data format (BC = 000 in ICMDR). Each bit put on the SDA line equates to 1 pulse on the SCL line and the data is always transferred with the most-significant bit (MSB) first. The number of data values that can be transmitted or received is unrestricted; however, the transmitters and receivers must agree on the number of data values being transferred. The I²C module supports the following data formats.

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

**Figure 2-6 I²C Module Data Transfer**



### 2.7.1 7-Bit Addressing Format

In the 7-bit addressing format (Figure 2-7), the first byte after a START condition (S) consists of a 7-bit slave address followed by a R/$\overline{W}$ bit. The R/$\overline{W}$ bit determines the direction of the data:

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

An extra clock cycle dedicated for acknowledgment (ACK) is inserted after the R/$\overline{W}$ bit. If the ACK bit is inserted by the slave, it is followed by *n* bits of data from the transmitter (master or slave, depending on the R/$\overline{W}$ bit). *n* is a number from 1 to 8 determined by the bit count (BC) bits of ICMDR. 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 ICMDR.

**Figure 2-7 $I^2C$ Module 7-Bit Addressing Format (FDF = 0, XA = 0 in ICMDR)**

| 1 | 7 | 1 | 1 | n | 1 | n | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| S | Slave Address | R/$\overline{W}$ | ACK | Data | ACK | Data | ACK | P |

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

### 2.7.2 10-Bit Addressing Format

The 10-bit addressing format (Figure 2-8) is like 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/$\overline{W}$ = 0 (write). The second byte is the remaining 8 bits of the 10-bit slave address. The slave must send acknowledgment (ACK) after each of the two byte transfers. Once the master has written the second byte to the slave, the master can either write data or use a repeated START condition to change the data direction. (For more details about using 10-bit addressing, see the Philips Semiconductors $I^2C$-bus specification.)

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

**Figure 2-8 $I^2C$ Module 10-Bit Addressing Format With Master-Transmitter Writing to Slave-Receiver (FDF = 0, XA = 1 in ICMDR)**

| 1 | 7 | 1 | 1 | n | 1 | n | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| S | 1 1 1 1 0 A A | 0 | ACK | A A A A A A A A | ACK | Data | ACK | P |
| | AA = 2 MSBs | R/$\overline{W}$ | | 8 LSBs of slave address | | | | |

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

### 2.7.3 Free Data Format

In the free data format (Figure 2-9), the first bits after a START condition (S) are a data word. An ACK bit is inserted after each data word, which can be from 1 to 8 bits, depending on the bit count (BC) bits of ICMDR. 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 ICMDR.

**Figure 2-9 I²C Module Free Data Format (FDF = 1 in ICMDR)**

| 1 | n | 1 | n | 1 | n | 1 | 1 |
|---|---|---|---|---|---|---|---|
| S | Data | ACK | Data | ACK | Data | ACK | P |

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

### 2.7.4 Using a Repeated START Condition

The repeated START condition can be used with the 7-bit addressing, 10-bit addressing, and free data formats. The 7-bit addressing format using a repeated START condition (S) is shown in Figure 2-10. At the end of each data word, the master can drive another START condition. Using this capability, a master can transmit/receive any number of data words before driving a STOP condition. The length of a data word can be from 1 to 8 bits and is selected with the bit count (BC) bits of ICMDR.

**Figure 2-10 I²C Module 7-Bit Addressing Format With Repeated START Condition**

| 1 | 7 | 1 | 1 | n | 1 | 1 | 7 | 1 | 1 | n | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | Slave Address | R/$\overline{W}$ | ACK | Data | ACK | S | Slave Address | R/$\overline{W}$ | ACK | Data | ACK | P |
| | | 1 | | Any Number | | | | 1 | | Any Number | | |

## 2.8 Operating Modes

The I²C module has four basic operating modes to support data transfers as a master and as a slave. For the names and descriptions of the modes, see Table 2-1.

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. In order 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 2-1 Operating Modes of the I²C Module**

| Operating Mode | Description |
|---|---|
| Slave-receiver mode | The I²C module is a slave and receives data from a master. All slave modules 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 DSP is required (RSFULL = 1 in ICSTR) after data has been received. |
| Slave-transmitter mode | The I²C module is a slave and transmits data to a master. This mode can only be entered from the slave-receiver mode; the I²C module must first receive a command from the master. When you are using any of the 7-bit/10-bit addressing formats, the I²C module enters its slave-transmitter mode if the slave address is the same as its own address (in ICOAR) and the master has transmitted R/W = 1. As a slave-transmitter, the I²C module then shifts the serial data out on SDA with the clock pulses that are generated by the master. While a slave, the I²C module does not generate the clock signal, but it can hold SCL low while the intervention of the DSP is required (XSMT = 0 in ICSTR) after data has been transmitted. |

**Table 2-1 Operating Modes of the I²C Module**

| Operating Mode | Description |
|---|---|
| Master-receiver mode | The I²C module is a master and receives data from a slave. This mode can only be entered from the master-transmitter mode; the I²C module must first transmit a command to the slave. When you are using any of the 7-bit/10-bit addressing formats, the I²C module enters its master-receiver mode after transmitting the slave address and R/W = 1. Serial data bits on SDA are shifted into the I²C module with the clock pulses generated by the I²C module on SCL. The clock pulses are inhibited and SCL is held low when the intervention of the DSP is required (RSFULL = 1 in ICSTR) after data has been received. |
| Master-transmitter mode | The I²C module is a master and transmits control information and data to a slave. All master modules 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 DSP is required (XSMT = 0 in ICSTR) after data has been transmitted. |
| **End of Table 2-1** | |

## 2.9 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 2-2 summarizes the various ways the I²C module sends a NACK bit.

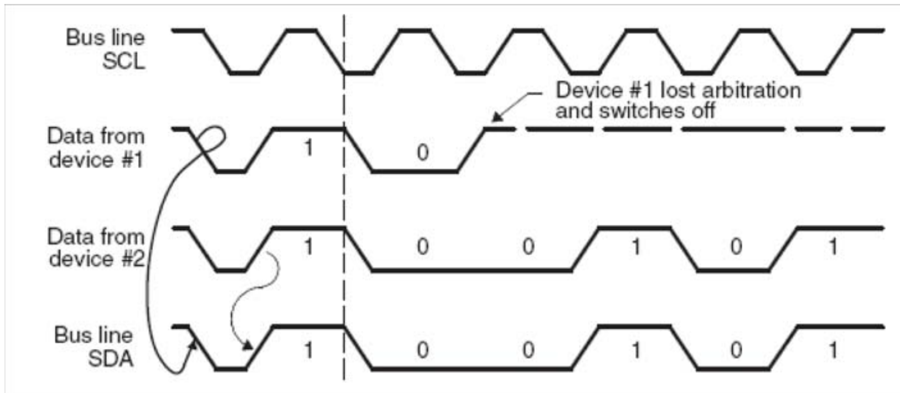
**Table 2-2 Generating a NACK Bit**

| I²C Module Condition | NACK Bit Generation | |
|---|---|---|
| | **Basic** | **Optional** |
| Slave-receiver mode | • Disable data transfers (STT = 0 in ICSTR).<br>• Allow an overrun condition (RSFULL = 1 in ICSTR).<br>• Reset the module (IRS = 0 in ICMDR) | Set the NACKMOD bit of ICMDR before the rising edge of the last data bit you intend to receive. |
| Master-receiver mode<br>AND<br>Repeat mode<br>(RM = 1 in ICMDR) | • Generate a STOP condition (STOP = 1 in ICMDR).<br>• Reset the module (IRS = 0 in ICMDR). | Set the NACKMOD bit of ICMDR before the rising edge of the last data bit you intend to receive. |
| Master-receiver mode<br>AND<br>Nonrepeat mode<br>(RM = 0 in ICMDR) | • If STP = 1 in ICMDR, allow the internal data counter to count down to 0 and force a STOP condition.<br>• If STP = 0, make STP = 1 to generate a STOP condition.<br>• Reset the module (IRS = 0 in ICMDR). | Set the NACKMOD bit of ICMDR before the rising edge of the last data bit you intend to receive. |
| **End of Table 2-2** | | |

## 2.10 Arbitration

If two or more master-transmitters simultaneously start a transmission on the same bus, an arbitration procedure is invoked. The arbitration procedure uses the data presented on the serial data bus (SDA) by the competing transmitters. Figure 2-11 illustrates the arbitration procedure between two devices.

**Figure 2-11 Arbitration Procedure Between Two Master-Transmitters**




\

The first master-transmitter, which drives SDA high, is overruled by another master-transmitter that drives SDA low. The arbitration procedure gives priority to the device that transmits the serial data stream with the lowest binary value. Should two or more devices send identical first bytes, arbitration continues on the subsequent bytes.

If the $I^2C$ module is the losing master, it switches to the slave-receiver mode, sets the arbitration lost (AL) flag, and generates the arbitration-lost interrupt.

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

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

## 2.11 Reset Considerations

The $I^2C$ module has two reset sources: software reset and hardware reset.

### 2.11.1 Software Reset Considerations

To reset the $I^2C$ module write 0 to the $I^2C$ reset (IRS) bit in the $I^2C$ mode register (ICMDR). All status bits in the $I^2C$ interrupt status register (ICSTR) are forced to their default values. The $I^2C$ module remains disabled until IRS is changed to 1. The SDA and SCL pins are in the high-impedance state.

**Note—**If the IRS bit is cleared to 0 during a transfer, this can cause the $I^2C$ bus to hang. For more information, see Section 2.15

### 2.11.2 Hardware Reset Considerations

When a hardware reset occurs, all the registers of the $I^2C$ module are set to the default value and the module remains disabled until the $I^2C$ reset (IRS) bit in the $I^2C$ mode register (ICMDR) is changed to 1.

**Note—**The IRS bit must be cleared to 0 while you configure/reconfigure the $I^2C$ module. Forcing IRS to 0 can be used to save power and to clear error conditions.

## 2.12 Interrupt Support

The $I^2C$ module is capable of interrupting the DSP CPU and sends a single interrupt to the CPU. The CPU can determine which $I^2C$ events caused the interrupt by reading the $I^2C$ interrupt vector register (ICIVR). ICIVR contains a binary-coded interrupt vector type to indicate which interrupt has occurred. Reading ICIVR clears the interrupt flag; if other interrupts are pending, a new interrupt is generated. If there is more than one pending interrupt flag, reading ICIVR clears the highest-priority interrupt flag.

### 2.12.1 Interrupt Events and Requests

The $I^2C$ module can generate the interrupts described in Table 2-3. Each interrupt has a flag bit in the $I^2C$ interrupt status register (ICSTR) and a mask bit in the interrupt mask register (ICIMR). When one of the specified events occurs, its flag bit is set. If the corresponding mask bit is 0, the interrupt request is blocked; if the mask bit is 1, the request is forwarded to the CPU as an $I^2C$ interrupt.

**Table 2-3 Descriptions of the $I^2C$ Interrupt Events**

| $I^2C$ Interrupt | Initiating Event |
|---|---|
| Arbitration-lost interrupt (AL) | Generated when the $I^2C$ arbitration procedure is lost or illegal START/STOP conditions occur. |
| No-acknowledge interrupt (NACK) | Generated when the master $I^2C$ does not receive any acknowledge from the receiver. |
| Registers-ready-for-access (ARDY) | Generated by the $I^2C$ when the previously programmed address, data and command have been interrupt performed and the status bits have been updated. This interrupt is used to let the controlling processor know that the $I^2C$ registers are ready to be accessed. |
| Receive interrupt/status (ICRINT and ICRRDY) | Generated when the received data in the receive-shift register (ICRSR) has been copied into the ICDRR. The ICRRDY bit can also be polled by the DSP to read the received data in the ICDRR. |
| Transmit interrupt/status (ICXINT and ICXRDY) | Generated when the transmitted data has been copied from ICDXR to the transmit-shift register (ICXSR) and shifted out on the SDA pin. This bit can also polled by the DSP to write the next transmitted data into the ICDXR. |
| Stop-Condition-Detection interrupt (SCD) | Generated when a STOP condition has been detected. |
| Address-as-Slave interrupt (AAS) | Generated when the $I^2C$ has recognized its own slave address or an address of all (8) zeros. |
| **End of Table 2-3** | |

### 2.12.2 Interrupt Multiplexing

The $I^2C$ interrupt to the DSP CPU is not multiplexed with any other interrupt source.

## 2.13 DMA Events Generated by the I2C Module

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

- Receive event (ICREVT): When receive data has been copied from the receive shift register (ICRSR) to the data receive register (ICDRR), the I2C module sends an REVT signal to the EDMA controller. In response, the EDMA controller can read the data from ICDRR.
- Transmit event (ICXEVT): When transmit data has been copied from the data transmit register (ICDXR) to the transmit shift register (ICXSR), the I2C module sends an XEVT signal to the EDMA controller. In response, the EDMA controller can write the next transmit data value to ICDXR.

## 2.14 Emulation Considerations

The response of the I2C events to emulation suspend events (such as halts and breakpoints) is controlled by the FREE bit in the I2C mode register (ICMDR). The I2C module either stops exchanging data (FREE = 0) or continues to run (FREE = 1) when an emulation suspend event occurs. How the I2C module terminates data transactions is affected by whether the I2C module is acting as a master or a slave. For a description of the FREE bit in ICMDR, see Section 3.10.

## 2.15 I2C Bus Hang Caused by Reset

It is generally known that the I2C bus can hang if an I2C master is removed from the bus in the middle of a data read. This can occur because the I2C protocol does not mandate a minimum clock rate. Therefore, if a master is reset in the middle of a read while a slave is driving the data line low, the slave will continue driving the data line low while it waits for the next clock edge. This prevents bus masters from initiating transfers. If this condition is detected, the following three steps will clear the bus hang condition:

1. An I2C master must generate up to 9 clock cycles.
2. After each clock cycle, the data pin must be observed to determine whether it has gone high while the clock is high.
3. As soon as the data pin is observed high, the master can initiate a start condition.

Appendix A on page A-1 contains more information on this topic.

Chapter 3

# Registers

This chapter provides a brief description of the peripheral registers and a table listing each of the registers.

## 3.1 Register Overview

Table 3-1 lists the memory-mapped registers for the inter-integrated circuit ($I^2C$) module. For the memory address of these registers, see the device-specific data manual.

**Table 3-1 GPIO Registers**

| Offsets | Acronym | Register Name | See |
|---|---|---|---|
| 0h | ICOAR | $I^2C$ Own Address Register | Section 3.2 |
| 4h | ICIMR | $I^2C$ Interrupt Mask/Status Register | Section 3.3 |
| 8h | ICSTR | $I^2C$ Interrupt Status Register | Section 3.4 |
| Ch | ICCLKL | $I^2C$ Clock Low-Time Divider Register | Section 3.5.1 |
| 10h | ICCLKH | $I^2C$ Clock High-Time divider Register | Section 3.5.2 |
| 14h | ICCNT | $I^2C$ Data Count Register | Section 3.6 |
| 18h | ICDRR | $I^2C$ Data Receive Register | Section 3.7 |
| 1Ch | ICSAR | $I^2C$ Slave Address Register | Section 3.8 |
| 20h | ICDXR | $I^2C$ Data Transmit Register | Section 3.9 |
| 24h | ICMDR | $I^2C$ Mode Register | Section 3.10 |
| 28h | ICIVR | $I^2C$ Interrupt Vector Register | Section 3.11 |
| 2Ch | ICEMDR | $I^2C$ Extended Mode Register | Section 3.12 |
| 30h | ICPSC | $I^2C$ Prescaler Register | Section 3.13 |
| 34h - 38h | ICPIDn | $I^2C$ Peripheral ID Registers | Section 3.14 |
| **End of Table 3-1** | | | |

## 3.2 $I^2C$ Own Address Register (ICOAR)

The $I^2C$ own address register (ICOAR) is used to specify its own slave address, which distinguishes it from other slaves connected to the $I^2C$-bus. If the 7-bit addressing mode is selected (XA = 0 in ICMDR), only bits 6-0 are used; bits 9-7 are ignored. The ICOAR register is shown in Figure 3-1 and described in Table 3-2

**Figure 3-1 $I^2C$ Own Address Register (ICOAR)**

| 31 — 10 | 9 — 0 |
|---|---|
| Reserved | OADDR |
| R-0 | R/W-0 |

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

**Table 3-2 $I^2C$ Own Address Register Field Descriptions**

| Bit | Field | Description |
|---|---|---|
| 31-10 | Reserved | Value = 0<br>These reserved bit location are always read as zeroes. A value written to this field has no effect. |
| 9-0 | OADDR | Value = 0-3FFh<br>Own slave address. Provides the slave address of the $I^2C$.<br>In 7-bit addressing mode (XA = 0 in ICMDR): bits 6-0 provide the 7-bit slave address of the $I^2C$.<br>Bits 9-7 are ignored.<br>In 10-bit addressing mode (XA = 1 in ICMDR): bits 9-0 provide the 10-bit slave address of the $I^2C$. |
| **End of Table 3-2** | | |

## 3.3 I²C Interrupt Mask Register (ICIMR)

The I²C interrupt mask register (ICIMR) is used by the CPU to individually enable or disable I²C interrupt requests. The ICIMR register is shown in Figure 3-2 and described Table 3-3.

**Figure 3-2 I²C Interrupt Mask Register (ICIMR)**

| 31–7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | AAS | SCD | ICXRDY | ICRDRDY | 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 = Read only; R/W = Read/Write; -*n* = value after reset

**Table 3-3 I²C Interrupt Mask Register Field Descriptions**

| Bit | Field | Description |
|---|---|---|
| 31-7 | Reserved | These reserved bit location are always read as zeroes. A value written to this field has no effect. |
| 6 | AAS | Address-as-slave interrupt enable bit.<br>0 = Interrupt request is disabled.<br>1 = Interrupt request is enabled. |
| 5 | SCD | Stop condition detected interrupt enable bit.<br>0 = Interrupt request is disabled.<br>1 = Interrupt request is enabled. |
| 4 | ICXRDY | Transmit-data-ready interrupt enable bit.<br>0 = Interrupt request is disabled.<br>1 = Interrupt request is enabled. |
| 3 | ICRDRDY | Receive-data-ready interrupt enable bit.<br>0 = Interrupt request is disabled.<br>1 = Interrupt request is enabled. |
| 2 | ARDY | Register-access-ready interrupt enable bit.<br>0 = Interrupt request is disabled.<br>1 = Interrupt request is enabled. |
| 1 | NACK | No-acknowledgment interrupt enable bit.<br>0 = Interrupt request is disabled.<br>1 = Interrupt request is enabled. |
| 0 | AL | Arbitration-lost interrupt enable bit<br>0 = Interrupt request is disabled.<br>1 = Interrupt request is enabled. |
| **End of Table 3-3** | | |

## 3.4 I²C Interrupt Status Register (ICSTR)

The I²C interrupt status register (ICSTR) is used by the CPU to determine which interrupt has occurred and to read status information. The ICSTR register is shown in Figure 3-3 and described in Table 3-4.

**Figure 3-3 I²C Interrupt Status Register (ICSTR)**

| 31–16 |
|---|
| Reserved |
| R-0 |

| 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/W1C-0 | R-0 | R-1 | R-0 | R-0 |

| 7–6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| Reserved | SCD | ICXRDY | ICRDRDY | ARDY | NACK | AL |
| R-0 | R/W1C-0 | R/W1C-0 | R/W1C-0 | R/W1C-0 | R/W1C-0 | R/W1C-0 |

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

**Table 3-4 I²C Interrupt Status Register Field Descriptions (Part 1 of 3)**

| Bit | Field | Description |
|---|---|---|
| 31-15 | Reserved | These reserved bit location are always read as zeroes. A value written to this field has no effect. |
| 14 | SDIR | Slave direction bit. In digital-loopback mode (DLB), the SDIR bit is cleared to 0.<br>0 = I²C is acting as a master-transmitter/receiver or a slave-receiver. SDIR is cleared by one of the following events:<br>» A STOP or a START condition.<br>» SDIR is manually cleared. To clear this bit, write a 1 to it.<br>1 = I²C is acting as a slave-transmitter. |
| 13 | NACKSNT | No-acknowledgment sent bit. NACKSNT bit is used when the I²C 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 3.10).<br>0 = NACK is not sent. NACKSNT is cleared by one of the following events:<br>» It is manually cleared. To clear this bit, write a 1 to it.<br>» The I²C is reset (either when 0 is written to the IRS bit of ICMDR or when the DSP is reset).<br>1 = NACK is sent. A no-acknowledge bit was sent during the acknowledge cycle on the I²C-bus. |
| 12 | BB | Bus busy bit. BB bit indicates whether the I²C-bus is busy or is free for another data transfer. In the master mode, BB is controlled by the software.<br>0 = Bus is free. BB is cleared by one of the following events:<br>» The I²C receives or transmits a STOP bit (bus free).<br>» BB is manually cleared. To clear this bit, write a 1 to it.<br>» The I²C is reset (either when 0 is written to the IRS bit of ICMDR or when the DSP is reset).<br>1 = Bus is busy. When the STT bit in ICMDR is set to 1, a restart condition is generated. BB is set by one of the following events:<br>» The I²C has received or transmitted a START bit on the bus.<br>» SCL is in a low state and the IRS bit in ICMDR is 0. |
| 11 | RSFULL | Receive shift register full bit. RSFULL indicates an overrun condition during reception. Overrun occurs when the receive shift register (ICRSR) is full with new data but the previous data has not been read from the data receive register (ICDRR). The new data will not be copied to ICDRR until the previous data is read. As new bits arrive from the SDA pin, they overwrite the bits in ICRSR.<br>0 = No overrun is detected. RSFULL is cleared by one of the following events:<br>» ICDRR is read.<br>» The I²C is reset (either when 0 is written to the IRS bit of ICMDR or when the DSP is reset).<br>1 = Overrun is detected. |

**Table 3-4 I²C Interrupt Status Register Field Descriptions (Part 2 of 3)**

| Bit | Field | Description |
|---|---|---|
| 10 | XSMT | Underflow occurs when the transmit shift register (ICXSR) is empty but the data transmit register (ICDXR) has not been loaded since the last ICDXR-to-ICXSR transfer. The next ICDXR-to-ICXSR transfer will not occur until new data is in ICDXR. If new data is not transferred in time, the previous data may be re-transmitted on the SDA pin.<br>0 = Underflow is detected.<br>1 = No underflow is detected. XSMT is set by one of the following events:<br>» Data is written to ICDXR.<br>» The I²C is reset (either when 0 is written to the IRS bit of ICMDR or when the DSP is reset). |
| 9 | AAS | Addressed-as-slave bit.<br>0 = The AAS bit has been cleared by a repeated START condition or by a STOP condition.<br>1 = AAS is set by one of the following events:<br>» I²C has recognized its own slave address or an address of all zeros (general call).<br>» The first data word has been received in the free data format (FDF = 1 in ICMDR). |
| 8 | AD0 | Address 0 bit.<br>0 = AD0 has been cleared by a START or STOP condition.<br>1 = An address of all zeros (general call) is detected. |
| 7-6 | Reserved | These reserved bit location are always read as zeroes. A value written to this field has no effect. |
| 5 | SCD | Stop condition detected bit. SCD indicates when a STOP condition has been detected on the I²C bus. The STOP condition could be generated by the I²C or by another I²C device connected to the bus.<br>0 = No STOP condition has been detected. SCD is cleared by one of the following events:<br>» By reading the INCODE bits in ICICR as 110b.<br>» SCD is manually cleared. To clear this bit, write a 1 to it.<br>1 = A STOP condition has been detected. |
| 4 | ICXRDY | Transmit-data-ready interrupt flag bit. ICXRDY indicates that the data transmit register (ICDXR) is ready to accept new data because the previous data has been copied from ICDXR to the transmit shift register (ICXSR). The CPU can poll ICXRDY or use the XRDY interrupt request.<br>0 = ICDXR is not ready. ICXRDY is cleared by one of the following events:<br>» Data is written to ICDXR.<br>» ICXRDY is manually cleared. To clear this bit, write a 1 to it.<br>1 I= CDXR is ready. Data has been copied from ICDXR to ICXSR. ICXRDY is forced to 1 when the I²C is reset. |
| 3 | ICRDRDY | Receive-data-ready interrupt flag bit. ICRRDY indicates that the data receive register (ICDRR) is ready to be read because data has been copied from the receive shift register (ICRSR) to ICDRR. The CPU can poll ICRRDY or use the RRDY interrupt request.<br>0 = ICDRR is not ready. ICRRDY is cleared by one of the following events:<br>» ICDRR is read.<br>» ICRRDY is manually cleared. To clear this bit, write a 1 to it.<br>» The I²C is reset (either when 0 is written to the IRS bit of ICMDR or when the DSP is reset).<br>1 = ICDRR is ready. Data has been copied from ICRSR to ICDRR. |
| 2 | ARDY | Register-access-ready interrupt flag bit (only applicable when the I²C is in the master mode). ARDY indicates that the I²C 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.<br>0 = The registers are not ready to be accessed. ARDY is cleared by one of the following events:<br>» The I²C starts using the current register contents.<br>» ARDY is manually cleared. To clear this bit, write a 1 to it.<br>» The I²C is reset (either when 0 is written to the IRS bit of ICMDR or when the DSP is reset).<br>1 = The registers are ready to be accessed.<br>» In the nonrepeat mode (RM = 0 in ICMDR): If STP = 0 in ICMDR, ARDY is set when the internal data counter counts down to 0. If STP = 1, ARDY is not affected (instead, the I2C generates a STOP condition when the counter reaches 0).<br>» In the repeat mode (RM = 1): ARDY is set at the end of each data word transmitted from ICDXR. |

**Table 3-4 I²C Interrupt Status Register Field Descriptions (Part 3 of 3)**

| Bit | Field | Description |
|---|---|---|
| 1 | NACK | No-acknowledgment interrupt flag bit. NACK applies when the I²C is a transmitter (master or slave). NACK indicates whether the I²C 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.<br>0 = ACK received/NACK is not received. NACK is cleared by one of the following events:<br>» An acknowledge bit (ACK) has been sent by the receiver.<br>» NACK is manually cleared. To clear this bit, write a 1 to it.<br>» The CPU reads the interrupt source register (ICISR) when the register contains the code for a NACK interrupt.<br>» The I²C is reset (either when 0 is written to the IRS bit of ICMDR or when the DSP is reset).<br>1= NACK bit is received. The hardware detects that a no-acknowledge (NACK) bit has been received.<br>Note: While the I²C performs a general call transfer, NACK is 1, even if one or more slaves send acknowledgment. |
| 0 | AL | Arbitration-lost interrupt flag bit (only applicable when the I²C is a master-transmitter). AL primarily indicates when the I²C has lost an arbitration contest with another master-transmitter. The CPU can poll AL or use the AL interrupt request.<br>0 = Arbitration is not lost. AL is cleared by one of the following events:<br>» AL is manually cleared. To clear this bit, write a 1 to it.<br>» The CPU reads the interrupt source register (ICISR) when the register contains the code for an AL interrupt.<br>» The I²C is reset (either when 0 is written to the IRS bit of ICMDR or when the DSP is reset).<br>1 = Arbitration is lost. AL is set by one of the following events:<br>» The I²C senses that it has lost an arbitration with two or more competing transmitters that started a transmission almost simultaneously.<br>» The I²C attempts to start a transfer while the BB (bus busy) bit is set to 1.<br>When AL is set to 1, the MST and STP bits of ICMDR are cleared, and the I²C becomes a slave-receiver. |
| **End of Table 3-4** | | |

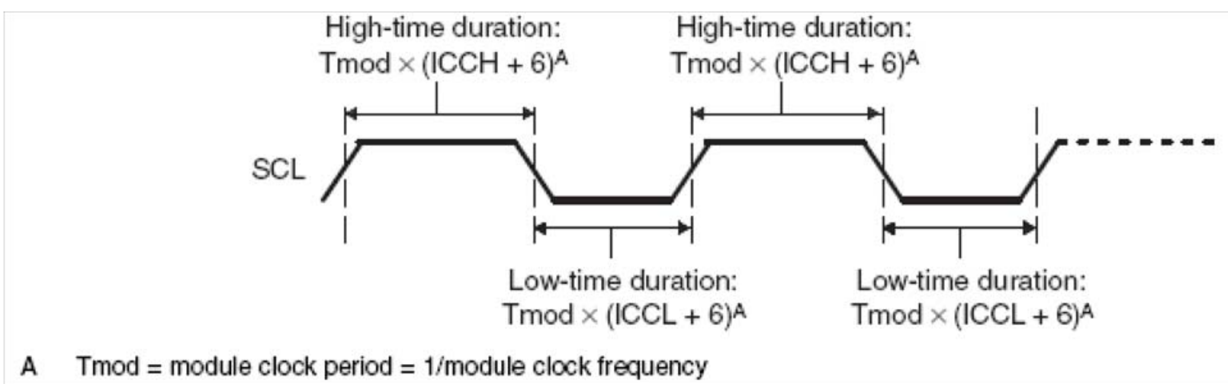
## 3.5 I²C Clock Divider Registers (ICCLKL and ICCLKH)

When the I²C is a master, the module clock is divided down for use as the master clock on the SCL pin. As shown in Figure 3-4, the shape of the master clock depends on two divide-down values, ICCL and ICCH.

The frequency of the master clock can be calculated as:

$$\text{master clock frequency} \quad \frac{\text{module clock frequency}}{(\text{ICCL6}) + (\text{ICCH6})}$$

**Figure 3-4 Roles of the Clock Divide-Down Values (ICCL and ICCH)**



### 3.5.1 I²C Clock Low-Time Divider Register (ICCLKL)

The I2C clock low-time divider register (ICCLKL) is shown in Figure 3-5 and described in Table 3-5. For each master clock cycle, ICCL determines the amount of time the signal is low. ICCLKL must be configured while the I²C is still in reset (IRS = 0 in ICMDR).

**Figure 3-5 I²C Clock Low-Time Divider Register (ICCLKL)**

| 31–16 | 15–0 |
|---|---|
| Reserved | iCCL |
| R-0 | R/W-0 |

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

**Table 3-5 I²C Clock Low-Time Divider Register Field Descriptions**

| Bit | Field | Description |
|---|---|---|
| 31-16 | Reserved | Value = 0<br>These reserved bit location are always read as zeroes. A value written to this field has no effect. |
| 15-0 | ICCL | Value = 0-FFFFh<br>Clock low-time divide-down value of 1-65536. The period of the module clock is multiplied by (ICCL + 6) to produce the low-time duration of the master clock on the SCL pin. |
| **End of Table 3-5** | | |

### 3.5.2 I²C Clock High-Time Divider Register (ICCLKH)

The I²C clock high-time divider register (ICCLKH) is shown in Figure 3-6 and described in Table 3-6. For each master clock cycle, ICCH determines the amount of time the signal is high. ICCLKH must be configured while the I²C is still in reset (IRS = 0 in ICMDR).

**Figure 3-6 I²C Clock High-Time Divider Register (ICCLKH)**

| 31–16 | 15–0 |
|---|---|
| Reserved | iCCH |
| R-0 | R/W-0 |

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

**Table 3-6 I²C Clock High-Time Divider Register Field Descriptions**

| Bit | Field | Description |
|---|---|---|
| 31-16 | Reserved | Value = 0<br>These reserved bit location are always read as zeroes. A value written to this field has no effect. |
| 15-0 | ICCH | Value = 0-FFFFh<br>Clock high-time divide-down value of 1-65536. The period of the module clock is multiplied by (ICCH + 6) to produce the high-time duration of the master clock on the SCL pin. |
| **End of Table 3-6** | | |

## 3.6 I²C Data Count Register (ICCNT)

The I²C data count register (ICCNT) is used to indicate how many data words to transfer when the I²C is configured as a master-transmitter (MST = 1 and TRX = 1 in ICMDR) and the repeat mode is off (RM = 0 in ICMDR). In the repeat mode (RM = 1), ICCNT is not used. The ICCNT register is shown in Figure 3-7 and described in Table 3-7.

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

**Figure 3-7 I²C Data Count Register (ICCNT)**

| 31–16 | 15–0 |
|---|---|
| Reserved | ICDC |
| R-0 | R/W-0 |

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

**Table 3-7 I²C Data Count Register Field Descriptions**

| Bit | Field | Description |
|---|---|---|
| 31-16 | Reserved | Value = 0<br>These reserved bit location are always read as zeroes. A value written to this field has no effect. |
| 15-0 | ICDC | Value = 0-FFFFh<br>Data count value. When RM = 0 in ICMDR, ICDC indicates the number of data words to transfer in the nonrepeat mode. When RM = 1 in ICMDR, the value in ICCNT is a don't care. If STP = 1 in ICMDR, a STOP condition is generated when the internal data counter counts down to 0.<br>0 = The start value loaded to the internal data counter is 65536.<br>1h-FFFFh = The start value loaded to internal data counter is 1-65535. |
| **End of Table 3-7** | | |

## 3.7 I²C Data Receive Register (ICDRR)

The I²C data receive register (ICDRR) is used by the DSP to read the receive data. The ICDRR can receive a data value of up to 8 bits; data values with fewer than 8 bits are right-aligned in the D bits and the remaining D bits are undefined. The number of data bits is selected by the bit count bits (BC) of ICMDR. The I²C receive shift register (ICRSR) shifts in the received data from the SDA pin. Once data is complete, the I²C copies the contents of ICRSR into ICDRR. The CPU and the EDMA controller cannot access ICRSR. The ICDRR register is shown in Figure 3-8 and described in Table 3-8.

**Figure 3-8 I²C Data Count Register (ICCNT)**

| 31–8 | 7–0 |
|---|---|
| Reserved | D |
| R-0 | R-0 |

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

**Table 3-8 I²C Data Count Register Field Descriptions**

| Bit | Field | Description |
|---|---|---|
| 31-8 | Reserved | Value = 0<br>These reserved bit location are always read as zeroes. A value written to this field has no effect. |
| 7-0 | D | Value = 0-FFh<br>Receive data. |
| **End of Table 3-8** | | |

## 3.8 I²C Slave Address Register (ICSAR)

The I²C slave address register (ICSAR) contains a 7-bit or 10-bit slave address. When the I²C is not using the free data format (FDF = 0 in ICMDR), it uses this address to initiate data transfers with a slave or slaves. When the address is non-zero, 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 ICMDR), only bits 6-0 of ICSAR are used; bits 9-7 are ignored. The ICSAR register is shown in Figure 3-9 and described in Table 3-9.

**Figure 3-9 I²C Slave Address Register (ICSAR)**

| 31 - 10 | 9 - 0 |
|---|---|
| Reserved | SADDR |
| R-0 | R/W-3FFH |

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

**Table 3-9 I²C Slave Address Register Field Descriptions**

| Bit | Field | Description |
|---|---|---|
| 31-10 | Reserved | Value = 0<br>These reserved bit location are always read as zeroes. A value written to this field has no effect. |
| 9-0 | SADDR | Value = 0-3FFh<br>Slave address. Provides the slave address of the I²C.<br>In 7-bit addressing mode (XA = 0 in ICMDR): bits 6-0 provide the 7-bit slave address that the I²C transmits when it is in the master-transmitter mode. Bits 9-7 are ignored.<br>In 10-bit addressing mode (XA = 1 in ICMDR): Bits 9-0 provide the 10-bit slave address that the I²C transmits when it is in the master-transmitter mode. |
| **End of Table 3-9** | | |

## 3.9 $I^2C$ Data Transmit Register (ICDXR)

The CPU writes transmit data to the $I^2C$ data transmit register (ICDXR). The ICDXR can accept a data value of up to 8 bits. When writing a data value with fewer than 8 bits, the CPU must make sure that the value is right-aligned in the D bits. The number of data bits is selected by the bit count bits (BC) of ICMDR. Once data is written to ICDXR, the $I^2C$ copies the contents of ICDXR into the $I^2C$ transmit shift register (ICXSR). The ICXSR shifts out the transmit data from the SDA pin. The CPU and the EDMA controller cannot access ICXSR. The ICDXR register is shown in Figure 3-10 and described in Table 3-10.

**Figure 3-10 $I^2C$ Data Transmit Register (ICDXR)**

| 31–8 | 7–0 |
|---|---|
| Reserved | D |
| R-0 | R/W-0 |

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

**Table 3-10 $I^2C$ Data Transmit Register Field Descriptions**

| Bit | Field | Description |
|---|---|---|
| 31-8 | Reserved | Value = 0<br>These reserved bit location are always read as zeroes. A value written to this field has no effect. |
| 7-0 | D | Value = 0-FFh<br>Transmit data. |
| **End of Table 3-10** | | |

## 3.10 $I^2C$ Mode Register (ICMDR)

The $I^2C$ mode register (ICMDR) contains the control bits of the $I^2C$. The ICMDR register is shown in shown in Figure 3-11 and described in Table 3-11.

**Figure 3-11 $I^2C$ Mode Register (ICMDR)**

| 31–16 |
|---|
| Reserved |
| R-0 |

| 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-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 3-11 I²C Mode Register Field Descriptions (Part 1 of 2)**

| Bit | Field | Description |
|---|---|---|
| 31-16 | Reserved | These reserved bit location are always read as zeroes. A value written to this field has no effect. |
| 15 | NACKMOD | No-acknowledge (NACK) mode bit (only applicable when the I²C is a receiver).<br>0 = In slave-receiver mode: The I²C sends an acknowledge (ACK) bit to the transmitter during the each acknowledge cycle on the bus. The I²C only sends a no-acknowledge (NACK) bit if you set the NACKMOD bit.<br>In master-receiver mode: The I²C sends an ACK bit during each acknowledge cycle until the internal data counter counts down to 0. When the counter reaches 0, the I²C sends a NACK bit to the transmitter. To have a NACK bit sent earlier, you must set the NACKMOD bit.<br>1 = In either slave-receiver or master-receiver mode: The I²C 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.<br>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 | This emulation mode bit is used to determine the state of the I²C when a breakpoint is encountered in the high-level language debugger.<br>0 = When I²C is master: If SCL is low when the breakpoint occurs, the I²C stops immediately and keeps driving SCL low, whether the I²C is the transmitter or the receiver. If SCL is high, the I²C waits until SCL becomes low and then stops.<br>When I²C is slave: A breakpoint forces the I²C to stop when the current transmission/reception is complete.<br>1 = The I²C runs free; that is, it continues to operate when a breakpoint occurs. |
| 13 | STT | START condition bit (only applicable when the I²C is a master). The RM, STT, and STP bits determine when the I²C starts and stops data transmissions (see Table 3-12). Note that the STT and STP bits can be used to terminate the repeat mode.<br>0 = In master mode, STT is automatically cleared after the START condition has been generated.<br>In slave mode, if STT is 0, the I²C does not monitor the bus for commands from a master. As a result, the I²C performs no data transfers.<br>1 = In master mode, setting STT to 1 causes the I²C to generate a START condition on the I²C-bus.<br>In slave mode, if STT is 1, the I²C monitors the bus and transmits/receives data in response to commands from a master. |
| 12 | Reserved | These reserved bit location are always read as zeroes. A value written to this field has no effect. |
| 11 | STP | STOP condition bit (only applicable when the I²C is a master). The RM, STT, and STP bits determine when the I²C starts and stops data transmissions (see Table 3-12). Note that the STT and STP bits can be used to terminate the repeat mode.<br>0 = STP is automatically cleared after the STOP condition has been generated.<br>1 = STP has been set by the DSP to generate a STOP condition when the internal data counter of the I²C counts down to 0. |
| 10 | MST | Master mode bit. MST determines whether the I2C 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 (see Table 3-13).<br>0 = Slave mode. The I²C is a slave and receives the serial clock from the master.<br>1 = Master mode. The I²C is a master and generates the serial clock on the SCL pin. |
| 9 | TRX | Transmitter mode bit. When relevant, TRX selects whether the I²C is in the transmitter mode or the receiver mode. Table 3-13 summarizes when TRX is used and when it is a don't care.<br>0 = Receiver mode. The I²C is a receiver and receives data on the SDA pin.<br>1 = Transmitter mode. The I²C is a transmitter and transmits data on the SDA pin. |
| 8 | XA | Expanded address enable bit.<br>0 = 7-bit addressing mode (normal address mode). The I²C transmits 7-bit slave addresses (from bits 6-0 of ICSAR), and its own slave address has 7 bits (bits 6-0 of ICOAR).<br>1 = 10-bit addressing mode (expanded address mode). The I²C transmits 10-bit slave addresses (from bits 9-0 of ICSAR), and its own slave address has 10 bits (bits 9-0 of ICOAR). |
| 7 | RM | Repeat mode bit (only applicable when the I²C is a master-transmitter). The RM, STT, and STP bits determine when the I²C starts and stops data transmissions (see Table 3-12). If the I²C is configured in slave mode, the RM bit is don't care.<br>0 = Nonrepeat mode. The value in the data count register (ICCNT) determines how many data words are received/transmitted by the I²C.<br>1 = Repeat mode. Data words are continuously received/transmitted by the I²C until the STP bit is manually set to 1, regardless of the value in ICCNT. |
| 6 | DLB | Digital loopback mode bit (only applicable when the I²C is a master-transmitter). This bit disables or enables the digital loopback mode of the I²C. The effects of this bit are shown in Figure 3-12. Note that DLB mode in the free data format mode (DLB = 1 and FDF = 1) is not supported.<br>0 = Digital loopback mode is disabled.<br>1 = Digital loopback mode is enabled. In this mode, the MST bit must be set to 1 and data transmitted out of ICDXR is received in ICDRR after n DSP cycles by an internal path, where:<br>n = ((I²C input clock frequency/module clock frequency) × 8)<br>The transmit clock is also the receive clock. The address transmitted on the SDA pin is the address in ICOAR. |

**Table 3-11 I²C Mode Register Field Descriptions (Part 2 of 2)**

| Bit | Field | Description |
|---|---|---|
| 5 | IRS | I²C reset bit. Note that if IRS is reset during a transfer, it can cause the I²C bus to hang. For more information, see Section "2.15 I²C Bus Hang Caused by Reset" on page 2-12.<br>0 = The I²C is in reset/disabled. When this bit is cleared to 0, all status bits (in ICSTR) are set to their default values. SDA and SCL are in a high-impedance state.<br>1 = The I²C is enabled. |
| 4 | STB | START byte mode bit (only applicable when the I²C is a master). As described in version 2.1 of the Philips I²C-bus specification, the START byte can be used to help a slave that needs extra time to detect a START condition. When the I²C is a slave, the I²C ignores a START byte from a master, regardless of the value of the STB bit.<br>0 = The I²C is not in the START byte mode.<br>1 = The I²C is in the START byte mode. When you set the START condition bit (STT), the I²C begins the transfer with more than just a START condition. Specifically, it generates:<br>1. A START condition<br>2. A START byte (0000 0001b)<br>3. A dummy acknowledge clock pulse<br>4. A repeated START condition<br>The I²C sends the slave address that is in ICSAR. |
| 3 | FDF | Free data format mode bit. Note that DLB mode in the free data format mode (DLB = 1 and FDF = 1) is not supported (see Table 3-13).<br>0 = Free data format mode is disabled. Transfers use the 7-/10-bit addressing format selected by the XA bit.<br>1 = Free data format mode is enabled. |
| 2-0 | BC | Bit count bits. BC defines the number of bits (1 to 8) in the next data word that is to be received or transmitted by the I²C. The number of bits selected with BC must match the data size of the other device. Note that when BC = 0, a data word has 8 bits.<br>If the bit count is less than 8, receive data is right aligned in the D bits of ICDRR and the remaining D bits are undefined. Also, transmit data written to ICDXR must be right aligned.<br>0 = 8 bits per data word<br>1h = 1 bit per data word<br>2h = 2 bits per data word<br>3h = 3 bits per data word<br>4h = 4 bits per data word<br>5h = 5 bits per data word<br>6h = 6 bits per data word<br>7h = 7 bits per data word |
| **End of Table 3-11** | | |

**Table 3-12 Master-Transmitter/Receiver Bus Activity Defined by RM, STT, and STP Bits**

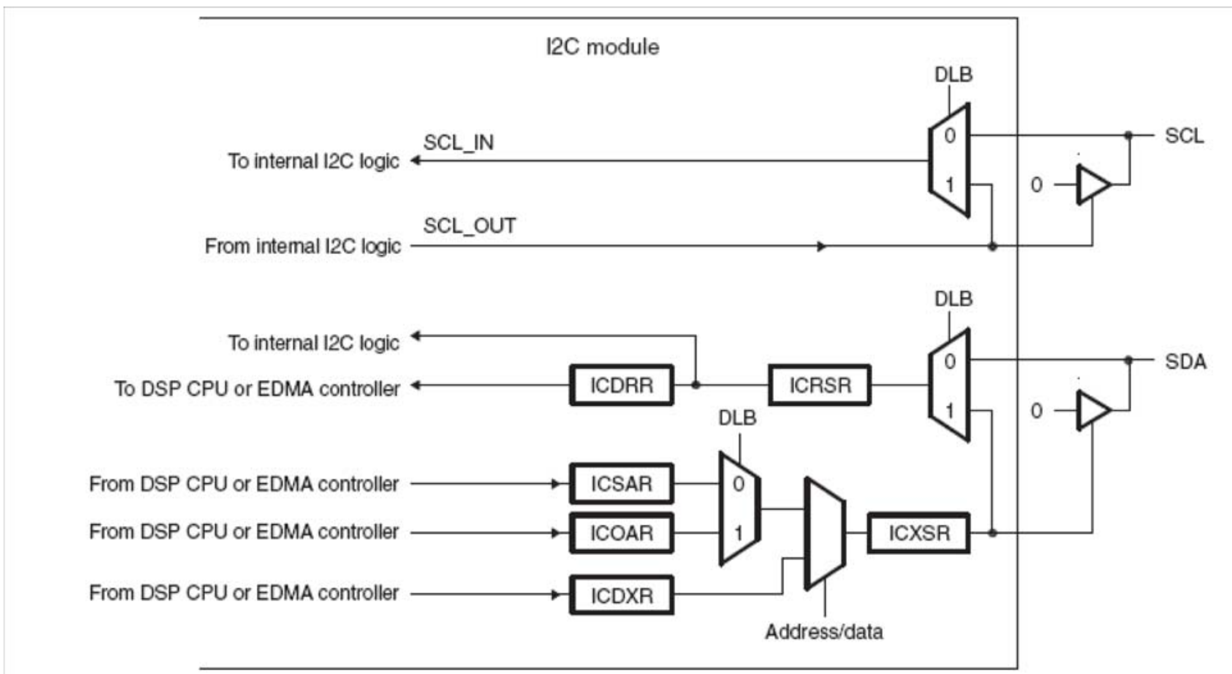
| ICMDR Bit | | | | |
|---|---|---|---|---|
| **RM** | **STT** | **STP** | **Bus Activity[1]** | **Description** |
| 0 | 0 | 0 | None | No activity |
| 0 | 0 | 1 | P | STOP condition |
| 0 | 1 | 0 | S-A-D..(n)..D | START condition, slave address, n data words (n = value in ICCNT) |
| 0 | 1 | 1 | S-A-D..(n)..D-P | START condition, slave address, n data words, STOP condition (n = value in ICCNT) |
| 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) |
| **End of Table 3-12** | | | | |

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

**Table 3-13 How the MST and FDF Bits Affect the Role of TRX Bit**

| ICMDR Bit | | | |
|---|---|---|---|
| **MST** | **FDF** | **I²C State** | **Function of TRX Bit** |
| 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 responds as a receiver or a transmitter. |
| 0 | 1 | In slave mode and free data format mode | The free data format mode requires that the transmitter and receiver be fixed. TRX identifies the role of the I²C:<br>TRX = 0: The I²C is a receiver.<br>TRX = 1: The I²C is a transmitter. |
| 1 | 0 | In master mode but not free data format mode | TRX identifies the role of the I²C:<br>TRX = 0: The I²C is a receiver.<br>TRX = 1: The I²C is a transmitter. |
| 1 | 1 | In master mode and free data format mode | The free data format mode requires that the transmitter and receiver be fixed. TRX identifies the role of the I²C:<br>TRX = 0: The I²C is a receiver.<br>TRX = 1: The I²C is a transmitter. |
| **End of Table 3-13** | | | |

**Figure 3-12 Block Diagram Showing the Effects of the Digital Loopback Mode (DLB) Bit**



## 3.11 I²C Interrupt Vector Register (ICIVR)

The I²C interrupt vector register (ICIVR) is used by the CPU to determine which event generated the I²C interrupt. Reading ICIVR clears the interrupt flag; if other interrupts are pending, a new interrupt is generated. If there are more than one interrupt flag, reading ICIVR clears the highest priority interrupt flag. Note that you must read (clear) ICIVR before doing another start; otherwise, ICIVR could contain an incorrect (old interrupt flags) value. The ICIVR register is shown in Figure 3-13 and described in Table 3-14.

**Figure 3-13 I²C Interrupt Vector Register (ICIVR)**

| 31 – 3 | 2 – 0 |
|---|---|
| Reserved | INTCODE |
| R-0 | R-0 |

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

**Table 3-14 I²C Interrupt Vector Register Field Descriptions**

| Bit | Field | Description |
|---|---|---|
| 31-3 | Reserved | Value = 0<br>These reserved bit location are always read as zeroes. A value written to this field has no effect. |
| 2-0 | INTCODE | Value = 0-7h<br>Interrupt code bits. The binary code in INTCODE indicates which event generated an I²C interrupt.<br>0 = None<br>1h = Arbitration-lost interrupt (AL)<br>2h = No-acknowledgment interrupt (NACK)<br>3h = Register-access-ready interrupt (ARDY)<br>4h = Receive-data-ready interrupt (ICRRDY)<br>5h = Transmit-data-ready interrupt (ICXRDY)<br>6h = Stop condition detected interrupt (SCD)<br>7h = Address-as-slave interrupt (AAS) |
| **End of Table 3-14** | | |

## 3.12 I²C Extended Mode Register (ICEMDR)

The I²C extended mode register (ICEMDR) is used to indicate which condition generates a transmit data-ready interrupt. The ICEMDR register is shown in Figure 3-14 and described in Table 3-15.

**Figure 3-14 I²C Extended Mode Register (ICEMDR)**

| 31 – 2 | 1 | 0 |
|---|---|---|
| Reserved | IGNACK | BCM |
| R-0 | R/W-0 | R/W-1 |

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

**Table 3-15 I²C Extended Mode Register Field Descriptions**

| Bit | Field | Description |
|---|---|---|
| 31-2 | Reserved | These reserved bit location are always read as 0s. A value written to this field has no effect. |
| 1 | IGNACK | Ignore NACK mode.<br>0 = The master transmitter operates normally, discontinue the data transfer, and set the ARDY and NACK status bits when a NACK signal is received from the slave.<br>1 = The master transmitter ignores a NACK received from the slave. |
| 0 | BCM | Backward compatibility mode bit. Determines which condition generates a transmit data ready interrupt.<br>The BCM bit has an effect only when the I2C is operating as a slave-transmitter.<br>0 = The transmit data ready interrupt is generated when the master requests more data by sending an acknowledge signal after the transmission of the last data.<br>1 = The transmit data ready interrupt is generated when the data in ICDXR is copied to ICXSR. |
| **End of Table 3-15** | | |

## 3.13 I²C Prescaler Register (ICPSC)

The I²C prescaler register (ICPSC) is used for dividing down the I²C input clock to obtain the desired module clock for the operation of the I²C. The ICPSC register is shown in Figure 3-15 and described in Table 3-16.

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

**Figure 3-15 I²C Prescaler Register (ICPSC)**

| 31 - 8 | 7 - 0 |
|---|---|
| Reserved | IPSC |
| R-0 | R/W-0 |

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

**Table 3-16 I²C Prescaler Register Field Descriptions**

| Bit | Field | Description |
|---|---|---|
| 31-8 | Reserved | Value = 0<br>These reserved bit location are always read as zeroes. A value written to this field has no effect. |
| 7-0 | IPSC | Value = 0-FFh<br>I²C prescaler divide-down value. IPSC determines how much the I²C input clock is divided to create the I²C module clock:<br>I²C clock frequency = I²C input clock frequency/(IPSC + 1)<br>Note: IPSC must be initialized while the I²C is in reset (IRS = 0 in ICMDR). |
| **End of Table 3-16** | | |

## 3.14 $I^2C$ Peripheral Identification Registers (ICPID1 and ICPID2)

The $I^2C$ peripheral identification registers (ICPIDn) contain identification data (class, revision, and type) for the peripheral. The ICPID1 register is shown in Figure 3-16 and described in Table 3-17. The ICPID2 register is shown in Figure 3-17and described in Table 3-18.

**Figure 3-16 $I^2C$ Peripheral Identification Register 1 (ICPID1)**

| 31 … 16 | 15 … 8 | 7 … 0 |
|---|---|---|
| Reserved | CLASS | REVISION |
| R-0 | R-44h | R-15h |

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

**Table 3-17 $I^2C$ Peripheral Identification Register 1 Field Descriptions**

| Bit | Field | Description |
|---|---|---|
| 31-16 | Reserved | Value = 0<br>These reserved bit location are always read as zeroes. A value written to this field has no effect. |
| 15-8 | CLASS | Identifies the peripheral class. |
| 7-0 | IPSC | Value = 0-FFh<br>Identifies the $I^2C$ revision level. This value should be incremented each time the design is revised. |
| **End of Table 3-17** | | |

**Figure 3-17 $I^2C$ Peripheral Identification Register 2 (ICPID2)**

| 31 … 16 | 15 … 0 |
|---|---|
| Reserved | TYPE |
| R-0 | R-2206h |

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

**Table 3-18 $I^2C$ Peripheral Identification Register 2 Field Descriptions**

| Bit | Field | Description |
|---|---|---|
| 31-16 | Reserved | Value = 0<br>These reserved bit location are always read as zeroes. A value written to this field has no effect. |
| 15-0 | TYPE | Identifies the peripheral type. |
| **End of Table 3-18** | | |

Appendix A

# $I^2C$ Lockup Issue

DSPs can use a memory device connected with an $I^2C$ interface as a source for boot code. A reset of the DSP while it is reading from the memory device can cause the $I^2C$ interface to enter a state which will prevent the DSP from accessing the attached memory device. Once the $I^2C$ interface enters this state it will not recover, even if additional DSP resets are issued. This condition will prevent the DSP from booting until the power is removed from the system or from the $I^2C$ device. This failure mode is possible as part of the normal behavior of the $I^2C$ interface as described in the $I^2C$ specification. To prevent a system from entering this unrecoverable state, a hardware workaround is needed to force the $I^2C$ interface into a usable condition. This appendix describes the problem and a hardware workaround that can be used to fix the issue.

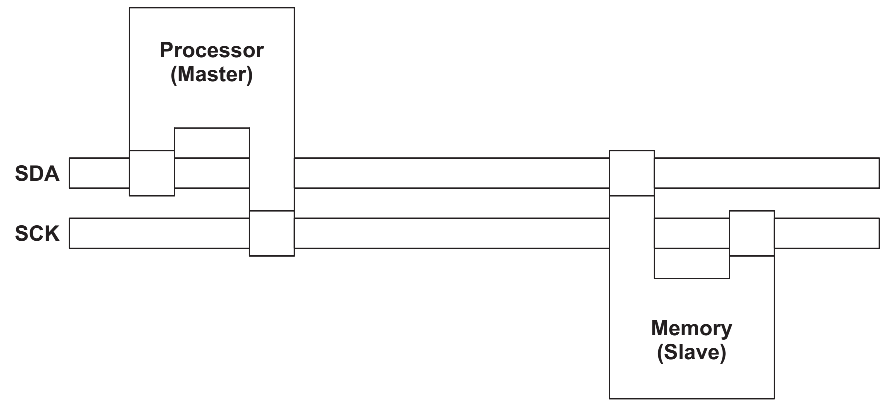
The $I^2C$ interface consists of two bidirectional lines, SDA and SCL, which are pulled to a positive supply voltage with external resistors. The output stages for these signals are open-drain or open-collector, allowing multiple devices to share the same $I^2C$ interfaces. These devices can be either a master-type device or a slave-type device.

Master devices initiate data transfers and generate the clock signals to permit transfers. Once a master device has initiated a transfer and gained control of the bus through the arbitration process it continues to control the bus until the end of an access. Under certain conditions if a reset to the controlling master device occurs during a transfer, the bus can remain in a locked state which does not allow the transfer to be completed and does not allow any other master device to gain control. This condition, inherent to the $I^2C$ specification, must be considered when designing a system dependent on $I^2C$ accesses.

This appendix presents a brief overview of the $I^2C$ bus and describes the condition that can halt accesses. The scenario of a single $I^2C$ master device connected to a slave device is described as well as a strategy to recover from the locked condition. This is followed by a discussion of the additional complications found in a multi-master environment.

In the simplest environment, a single master device will be connected to one or more slave devices. The example, shown in Figure A-1 is a processor attached to a memory device using the I²C interface.

**Figure A-1 Example I²C Interface**

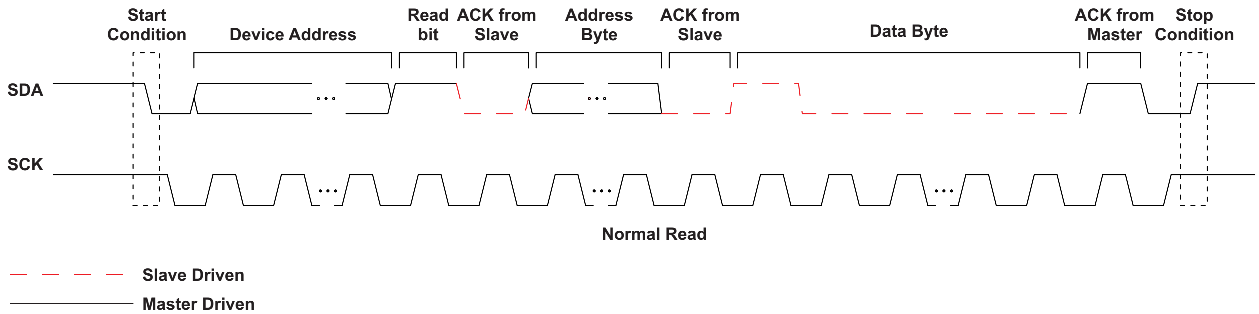



To initiate a transfer the master device will generate a start condition. The start condition is defined as a high-to-low transition on the SDA line while SCL is high. Once the master has initiated a transfer with a start condition, the bus is considered busy until the master generates a stop condition. The stop condition is defined as a low-to-high transition of the SDA line while the SCL is high. Once the transfer is initiated by the start condition, the SDA line must remain stable while SCL is high. SDA will only transition when SCL is low until the master releases the bus with a stop condition.

The start condition is followed by a series of byte transfers consisting of eight bits followed by a single acknowledge bit. The first byte contains the slave device address and a read/write bit. Once the slave device has received the correct address and the read/write bit, it will drive the SDA signal low as an acknowledgement that it has received the information correctly. The bytes that follow the device address vary in function from device to device. In the case of a sequential write to a slave memory device, the device address will be followed by a word address byte and a series of data bytes. Again, each byte transmitted from the master is acknowledged by the slave by driving the ninth bit low. When the master is done transmitting data it will send the stop condition releasing the bus.

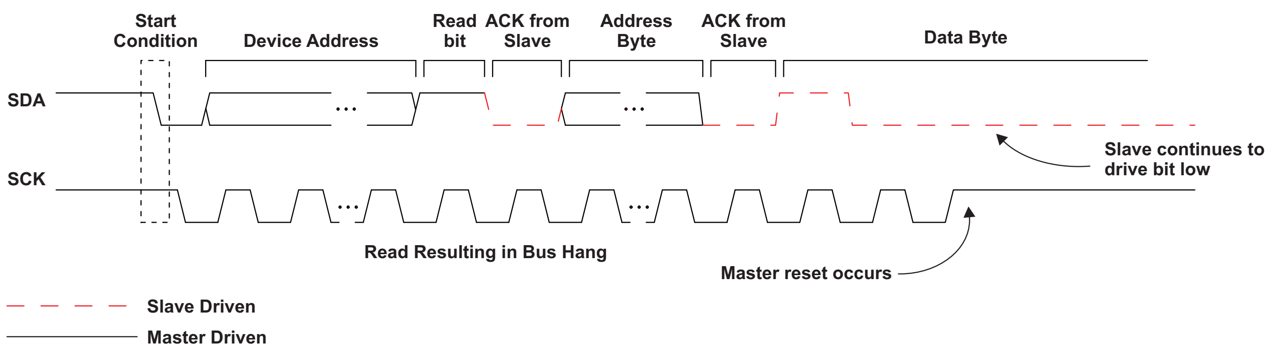
During a write cycle the slave will only drive the SDA low during the acknowledge (ACK). During a read cycle the slave will drive the data byte one bit at a time in response to the read command. There is a period of time during which the slave device will drive eight consecutive bits based on the clock provided by the master. When the master drives the clock low, it will drive the next bit onto the SDA line. When the master stops driving SCK, the pull-up resistor will return it to a high state signaling the slave device to hold the SDA signal level. Once all eight bits are clocked onto the SDA, the master can end the access by generating a stop condition freeing the bus. Figure A-2 shows a normal read cycle. The period when the slave is driving SDA is indicated by the red lines.

**Figure A-2 Normal Read Cycle**




If a reset to the master device occurs during the data transfer of a read cycle, there is the possibility of the bus ending in a state that will prevent any master from regaining control. Consider the situation shown in Figure A-3.

**Figure A-3 Bus-Hang Read Cycle**

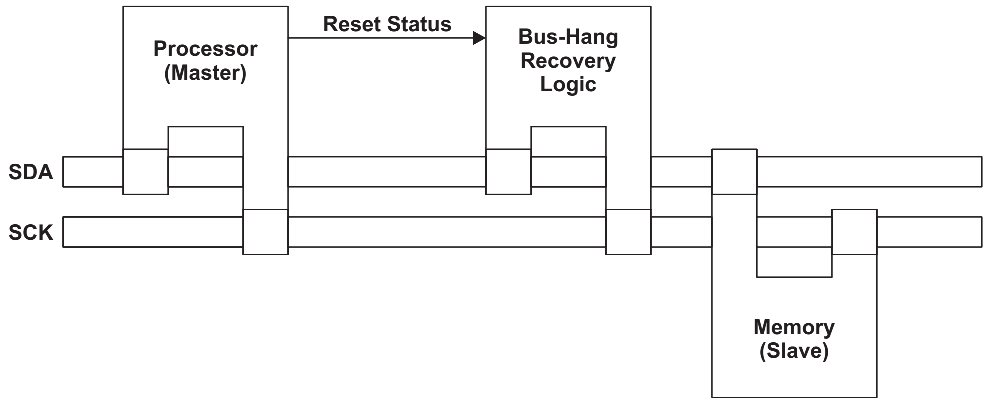



The master initiates a read in the normal fashion but a reset occurs while the SCK is high and the slave is driving the SDA low. The slave device will continue to drive SDA low waiting for the master to provide the next clock however the master has been reset and is no longer attempting to complete the read cycle. In addition, if the master attempts to access the bus after its reset has been released it will fail to arbitrate for control.

Arbitration for the I²C bus is performed on the SDA line. The master will attempt to generate a start condition beginning with both the SDA and the SCK lines high. Remember that the SDA and SCK lines are open-collector so the slave device will continue to hold the SDA line in a low state. When the master detects that the SDA line is not high, it will assume that some other device is in control of the bus and discontinue its attempt to start an access. This state will continue indefinitely blocking all attempts to access devices on the I²C interface.

This state is especially debilitating if the I²C slave is a memory device used as a boot memory. If the bus hang occurs during a reset, the processor acting as an I²C master will not be able to read its boot code and will fail to initialize. The processor cannot recover from this condition, even if additional resets are applied. To resolve this condition, additional hardware attached to the I²C bus is required (see Figure A-4).

**Figure A-4 I²C Master Bus Hang**



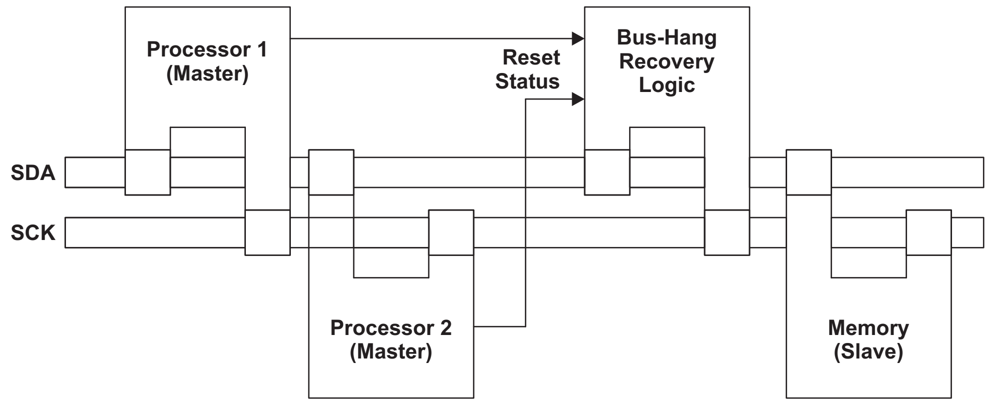

The bus-hang recovery logic needs to have knowledge of the reset status of the processor either through a RESETSTAT or a GPIO signal as well as the ability to monitor and drive the SDA and SCK signals. This logic may be a state machine in a programmable logic device or a software-driven solution in a microcontroller or processor using GPIO pins, but it must meet the open-collector driver requirements for the SDA and SCK signals to ensure the proper operation of the I²C.

Once the logic senses that the processor has entered reset, it should sample the SDA signal to determine if it is stuck in a low state. In a single master environment, the SDA should be pulled high as soon as the reset is applied to the master. If the logic senses that the SDA is low during a reset state, it can take action to free the bus. The logic should toggle SCK nine times ending in a high state.

These clock pulses must meet all the specification timing requirements for the high and low states. This will clock any remaining data bits from the slave memory device. Once these bits have been transmitted on the SDA, the slave device should release SDA and allow it to return to a high state. The recovery logic should sample SDA to ensure it has been released. The I²C bus should now be ready to respond to any requests for data by the processor.

The $I^2C$ interface with multiple master devices requires more complex logic to clear a bus-hang condition. In addition to determining that a master has been reset, the logic must also determine that a second master is not making a valid access to the bus (see Figure A-5).

**Figure A-5 Multiple Master Bus Hang**




In this environment, a reset to one of the processors during a read can cause the same bus-hang condition described above but detection of that condition is more complicated. When a single master is present on the bus, it is clear that there should not be any activity while that master is in reset. In this case, it is sufficient to sample the SDA signal while the RESET to the master is active, as described above. If multiple master devices are connected to the $I^2C$, a second master can access a working $I^2C$ interface while the first master is in reset. This would create the condition where SDA is low while the first processor is in reset. In this environment, the logic must monitor the reset status from both master devices to determine if either is in reset.

Once a reset has been detected, the logic should monitor the SDA signal and no further action is needed. If the SDA signal is low and the SCK is toggling, another master is accessing the slave device and the interface is not hung. Again, no further action is needed. If the SDA signal is low and the SCK signal is high for too long the bus-hang condition is present and the logic must take action to recover.

The period of time that defines a failure can vary, depending on the slave devices on the bus and the speed of the $I^2C$ interface. Slave devices can only add wait time to accesses by holding the SCK low so the average time that the SCK is high and the SDA is low should depend on the frequency that the masters are using to drive the interface. Once the slowest frequency is determined, the longest value for SCK high during an access should be multiplied by four to determine the period of time that defines a bus-hang condition. The logic should sample the SDA and SCK many times during the expected clock period to detect a stall condition.

If a bus-hang condition is present, the logic should send nine clock pulses onto SCK complete the in-process read from the slave memory. The slave should release SDA and allow it to be pulled high. In multi-master environments, a stop sequence should be generated after the nine clock pulses to release the bus for arbitration. This should clear the bus-hang condition and allow the $I^2C$ master devices to access the memory.

# IMPORTANT NOTICE