# OMAP5910 Dual-Core Processor Inter-Integrated Circuit ($I^2C$) Controller Reference Guide

Literature Number: SPRU681B
October 2003 − Revised October 2005

| **Products** | | **Applications** | |
|---|---|---|---|
| Amplifiers | amplifier.ti.com | Audio | www.ti.com/audio |
| Data Converters | dataconverter.ti.com | Automotive | www.ti.com/automotive |
| DSP | dsp.ti.com | Broadband | www.ti.com/broadband |
| Interface | interface.ti.com | Digital Control | www.ti.com/digitalcontrol |
| Logic | logic.ti.com | Military | www.ti.com/military |
| Power Mgmt | power.ti.com | Optical Networking | www.ti.com/opticalnetwork |
| Microcontrollers | microcontroller.ti.com | Security | www.ti.com/security |
| | | Telephony | www.ti.com/telephony |
| | | Video & Imaging | www.ti.com/video |
| | | Wireless | www.ti.com/wireless |

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

**Preface**

# Read This First

## *About This Manual*

This document describes the I$^2$C protocol. Figure 1 shows the overall OMAP5910 device architecture (with the I$^2$C peripheral highlighted), while Figure 2 shows the I$^2$C system overview. References to a local host in this section refer to the MPU processor.

## *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.

## *Related Documentation From Texas Instruments*

The following documents describe the OMAP5910 device and related peripherals. 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.

***OMAP5910 Dual-Core Processor MPU Subsystem Reference Guide*** (literature number SPRU671)

***OMAP5910 Dual-Core Processor DSP Subsystem Reference Guide*** (literature number SPRU672)

***OMAP5910 Dual-Core Processor Memory Interface Traffic Controller Reference Guide*** (literature number SPRU673)

***OMAP5910 Dual-Core Processor System DMA Controller Reference Guide*** (literature number SPRU674)

***OMAP5910 Dual-Core Processor LCD Controller Reference Guide*** (literature number SPRU675)

***OMAP5910 Dual-Core Processor Universal Asynchronous Receiver/Transmitter (UART) Devices Reference Guide*** (literature number SPRU676)

***OMAP5910 Dual-Core Processor Universal Serial Bus (USB) and Frame Adjustment Counter (FAC) Reference Guide*** (literature number SPRU677)

***OMAP5910 Dual-Core Processor Clock Generation and System Reset Management Reference Guide*** (literature number SPRU678)

***OMAP5910 Dual-Core Processor General-Purpose Input/Output (GPIO) Reference Guide*** (literature number SPRU679)

***OMAP5910 Dual-Core Processor MMC/SD Reference Guide*** (literature number SPRU680)

***OMAP5910 Dual-Core Processor Inter-Integrated Circuit (I2C) Controller Reference Guide*** (literature number SPRU681)

***OMAP5910 Dual-Core Processor Timer Reference Guide*** (literature number SPRU682)

***OMAP5910 Dual-Core Processor Inter-Processor Communication Reference Guide*** (literature number SPRU683)

***OMAP5910 Dual-Core Processor Camera Interface Reference Guide*** (literature number SPRU684)

***OMAP5905 Dual-Core Processor Multichannel Serial Interface (MCSI) Reference Guide*** (literature number SPRU685)

***OMAP5910 Dual-Core Processor Micro-Wire Interface Reference Guide*** (literature number SPRU686)

***OMAP5910 Dual-Core Processor Real-Time Clock (RTC) Reference Guide*** (literature number SPRU687)

***OMAP5910 Dual-Core Processor HDQ/1-Wire Interface Reference Guide*** (literature number SPRU688)

***OMAP5910 Dual-Core Processor PWL, PWT, and LED Peripheral Reference Guide*** (literature number SPRU689)

***OMAP5910 Dual-Core Processor Multichannel Buffered Serial Port (McBSP) Reference Guide*** (literature number SPRU708)

## Trademarks

OMAP and the OMAP symbol are trademarks of Texas Instruments.

# Contents

# Figures

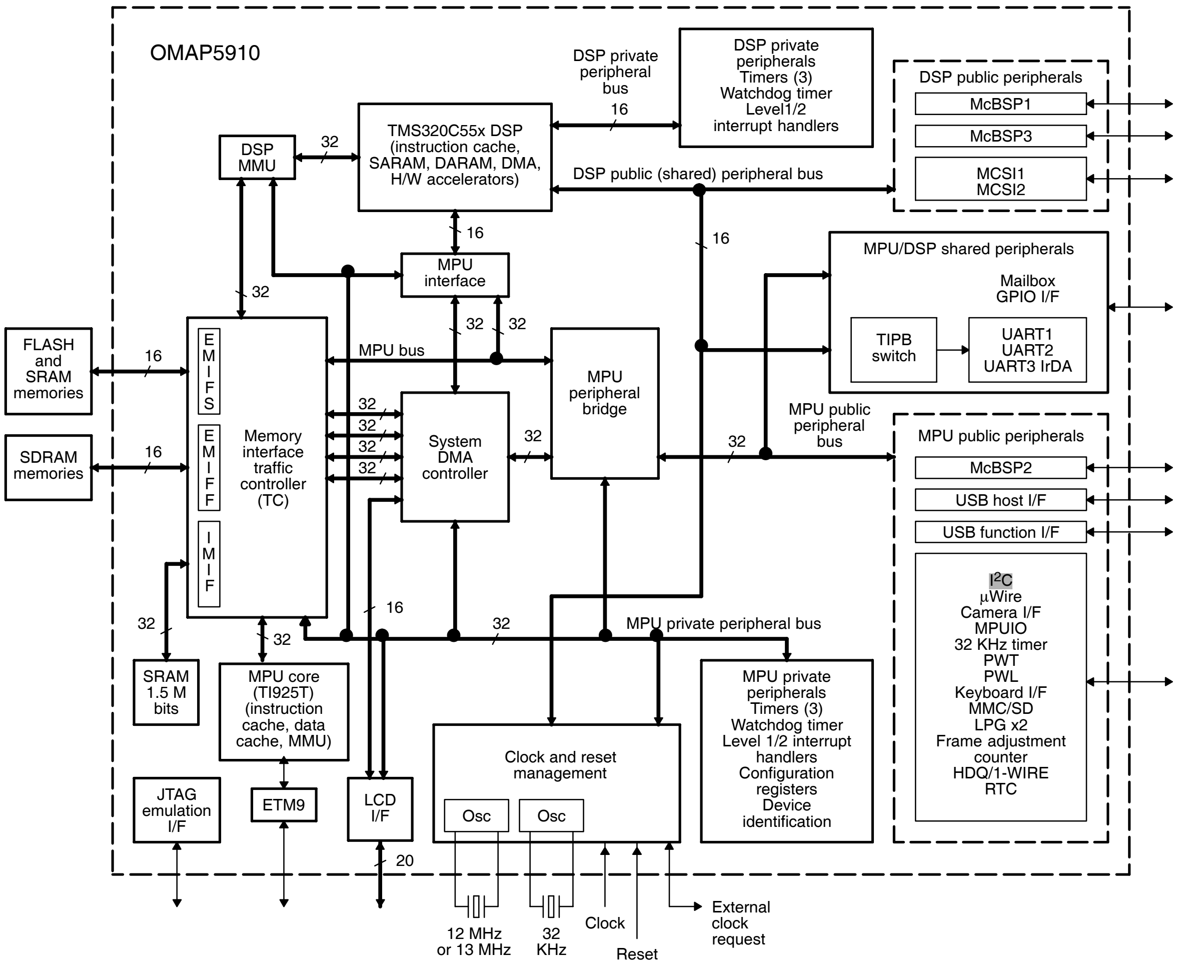
# Tables

# I$^2$C Controller

## 1 I$^2$C Protocol Description

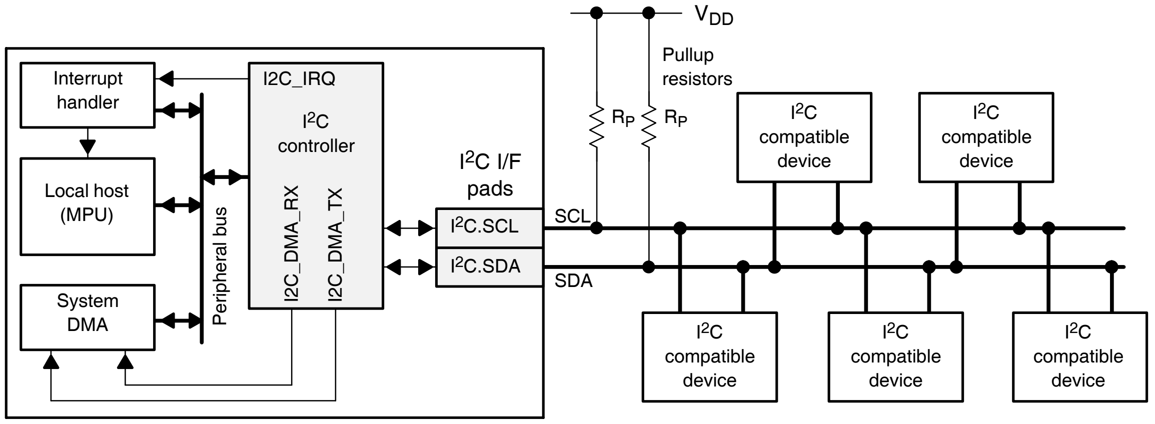
This document describes the I$^2$C protocol. Figure 1 shows the overall OMAP5910 device architecture (with the I$^2$C peripheral highlighted), while Figure 2 shows the I$^2$C system overview. References to a local host in this section refer to the MPU processor.

*Figure 1. OMAP5910 Functional Overview*


OMAP5910
DSP private peripheral bus
DSP private peripherals Timers (3) Watchdog timer Level1/2 interrupt handlers
DSP public peripherals
McBSP1
McBSP3
MCSI1 MCSI2
TMS320C55x DSP (instruction cache, SARAM, DARAM, DMA, H/W accelerators)
DSP MMU
DSP public (shared) peripheral bus
MPU interface
MPU/DSP shared peripherals
Mailbox GPIO I/F
TIPB switch
UART1 UART2 UART3 IrDA
FLASH and SRAM memories
SDRAM memories
EMIFS
EMIFF
IMIF
Memory interface traffic controller (TC)
MPU bus
System DMA controller
MPU peripheral bridge
MPU public peripheral bus
MPU public peripherals
McBSP2
USB host I/F
USB function I/F
I2C
µWire
Camera I/F
MPUIO
32 KHz timer
PWT
PWL
Keyboard I/F
MMC/SD
LPG x2
Frame adjustment counter
HDQ/1-WIRE
RTC
MPU private peripheral bus
SRAM 1.5 M bits
MPU core (TI925T) (instruction cache, data cache, MMU)
MPU private peripherals Timers (3) Watchdog timer Level 1/2 interrupt handlers Configuration registers Device identification
Clock and reset management
Osc
Osc
JTAG emulation I/F
ETM9
LCD I/F
12 MHz or 13 MHz
32 KHz
Clock
Reset
External clock request
16
32
20

The I2C controller is shown in the MPU Public Peripherals block (lower right portion of drawing).

*Figure 2. $I^2C$ System Overview*



## 1.1 Functional Overview

The $I^2C$ controller function supports the multimaster mode using the multimaster bus, to which devices capable of controlling the bus can be connected. Each $I^2C$ device (including the OMAP5910) has a unique address and can operate as either transmitter or receiver. In addition to being a transmitter or receiver, a device connected to the $I^2C$ bus can also be considered a master or slave when performing data transfers. A master device 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.

## 1.2 $I^2C$ Controller Signals

Data is communicated to $I^2C$ devices via the serial data pin (SDA) and the serial clock pin (SCL). These two wires carry information between the OMAP5910 and other devices connected to the $I^2C$ bus. Both SDA and SCL are bidirectional pins that must be connected to a positive supply voltage via pullup resistors. When the bus is free, both pins are high. The pin drivers have open-drains to perform the required wired-AND function. See Table 1 for a list of the signal pads and Table 2 for the reset state of the $I^2C$ signals.

*Table 1. Signal Pins*

| Name | Type | Description | Reset Value |
|---|---|---|---|
| I2C.SCL | In/Out(OD) | I²C serial CLK line.<br>Open-drain output buffer—requires an external pullup resistor (Rp) | Input |
| I2C.SDA | In/Out(OD) | I²C serial data line.<br>Open-drain output buffer—requires an external pullup resistor (Rp) | Input |

*Table 2. Reset State of I²C Signals*

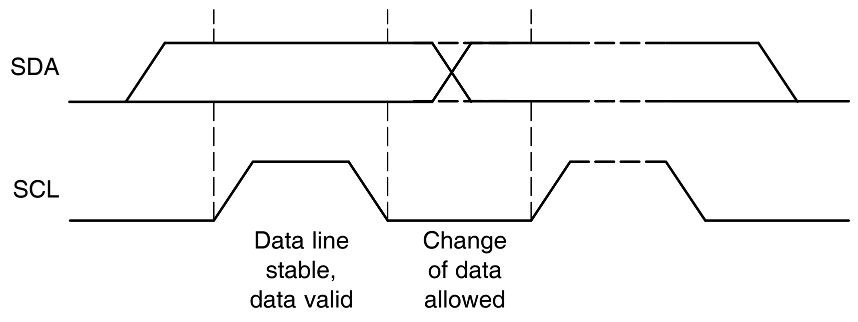
| Pin | Pads | System Reset | I²C Reset (I2C_EN =0) |
|---|---|---|---|
| SDA | I/O | High impedance | High impedance |
| SCL | I/O | High impedance | High impedance |

The master device generates one clock pulse for each data bit transferred. Due to the variety of devices (CMOS, NMOS, bipolar) that can be connected to the I²C bus, the levels of logical 0 (low) and 1 (high) are not fixed and depend on the associated VDD level.

## 1.3 I²C Bus Principal

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

*Figure 3. Data Validity on the I²C Bus*

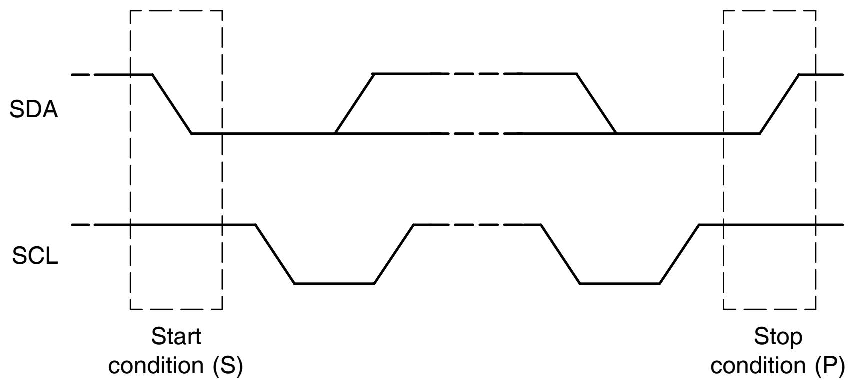


The I²C module generates start and stop conditions when it is configured as a master (see Figure 4):

- ❑ The start condition is a high-to-low transition on the SDA line while SCL is high.
- ❑ The stop condition is a low-to-high transition on the SDA line while SCL is high.

The bus is considered busy after the start condition (BB = 1) and free after the stop condition (BB = 0).

*Figure 4. Start and Stop Conditions*

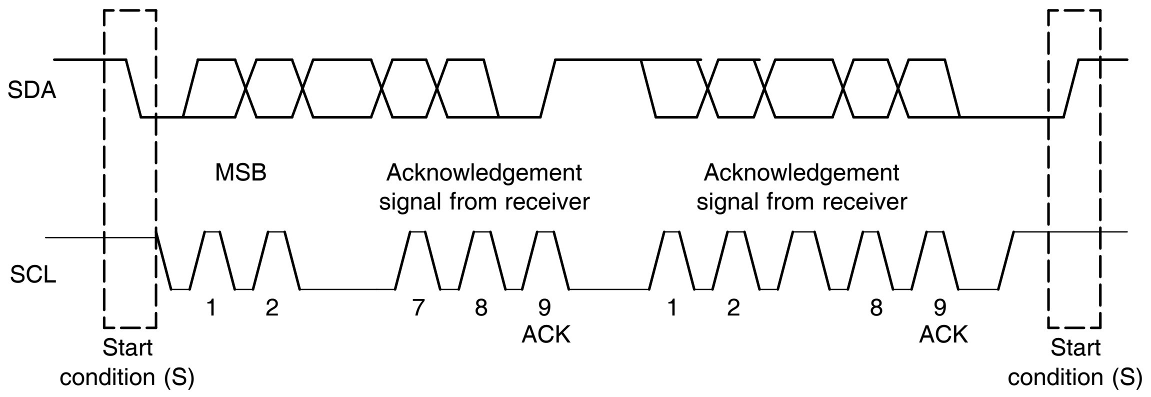


## 1.4 I$^2$C Operation

### 1.4.1 Serial Data Formats

Each byte sent to the SDA line is 8 bits long. The number of bytes that can be transmitted or received is unrestricted. The data is transferred with the most significant bit (MSB) first. Each byte is followed by an acknowledge bit from the I$^2$C module if it is in the receive mode (see Figure 5).

*Figure 5. I$^2$C Data Transfer*


SDA
MSB
Acknowledgement signal from receiver
Acknowledgement signal from receiver
SCL
1
2
7
8
9
ACK
1
2
8
9
ACK
Start condition (S)
Start condition (S)

The $I^2C$ protocol supports the two data formats shown in Figure 6.

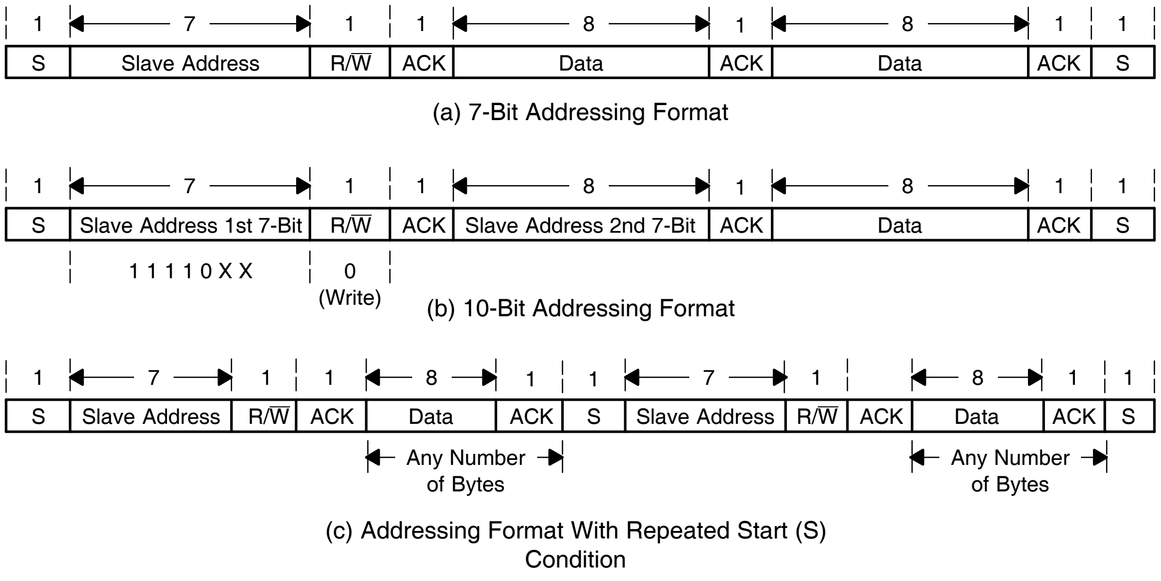
- ❑ 7-bit/10-bit addressing format
- ❑ 7-bit/10-bit addressing format with repeated start condition

The first byte after a start condition (S) always consists of 8 bits. In the acknowledge mode, an extra acknowledgement bit is inserted after each byte.

In the addressing formats with 7-bit addresses, the first byte is composed of seven MSB slave address bits and one LSB R/$\overline{W}$ bit. While in the addressing formats with 10-bit addresses, the first byte is composed of a seven MSB slave address, such as 1111*0XX*, where 0XX are the two MSBs of the 10-bit addresses, and one LSB R/$\overline{W}$ bit, which is 0 in this case.

The least significant R/$\overline{W}$ of the address byte indicates the direction of transmission of the following data bytes. If R/$\overline{W}$ is 0, the master writes (transmits) data to the selected slave; if it is 1, the master reads (receive) data from the slave.

*Figure 6. $I^2C$ Data Transfer Formats*


1 7 1 1 8 1 8 1 1
S | Slave Address | R/W̄ | ACK | Data | ACK | Data | ACK | S
(a) 7-Bit Addressing Format
1 7 1 1 8 1 8 1 1
S | Slave Address 1st 7-Bit | R/W̄ | ACK | Slave Address 2nd 7-Bit | ACK | Data | ACK | S
1 1 1 1 0 X X
0 (Write)
(b) 10-Bit Addressing Format
1 7 1 1 8 1 1 7 1 8 1 1
S | Slave Address | R/W̄ | ACK | Data | ACK | S | Slave Address | R/W̄ | ACK | Data | ACK | S
Any Number of Bytes
Any Number of Bytes
(c) Addressing Format With Repeated Start (S) Condition

## 1.4.2 Master Transmitter

In this mode, data assembled in one of the previously described data formats is shifted out on the serial data line SDA in synchronism with the self-generated clock pulses on the serial clock line SCL. The clock pulses are inhibited and the SCL bus is held low when the intervention of the processor is required after a byte has been transmitted.

### 1.4.3 Master Receiver

This mode can only be entered from the master transmitter mode. With any of the address formats (Figure 6 (a), (b), and (c)), the master receiver is entered after the slave address byte and bit $R/\overline{W}$ has been transmitted if $R/\overline{W}$ is high. Serial data bits received on bus line SDA are shifted in synchronism with the self-generated clock pulses on the SCL. The clock pulses are inhibited and the SCL held low when the intervention of the processor is required after a byte has been transmitted. At the end of a transfer, a stop condition is generated.

### 1.4.4 Slave Transmitter

This mode can only be entered from the slave receiver mode. With any of the address formats (Figure 6 (a), (b), and (c)), the slave transmitter is enabled if the slave address byte is the same as its own address and bit $R/\overline{W}$ has been transmitted if $R/\overline{W}$ is high. The slave transmitter shifts the serial data out on the data line SDA in synchronism with the clock pulses that are generated by the master device. It does not generate the clock, but it can hold the clock line (SCL) low while intervention of the local host is required.

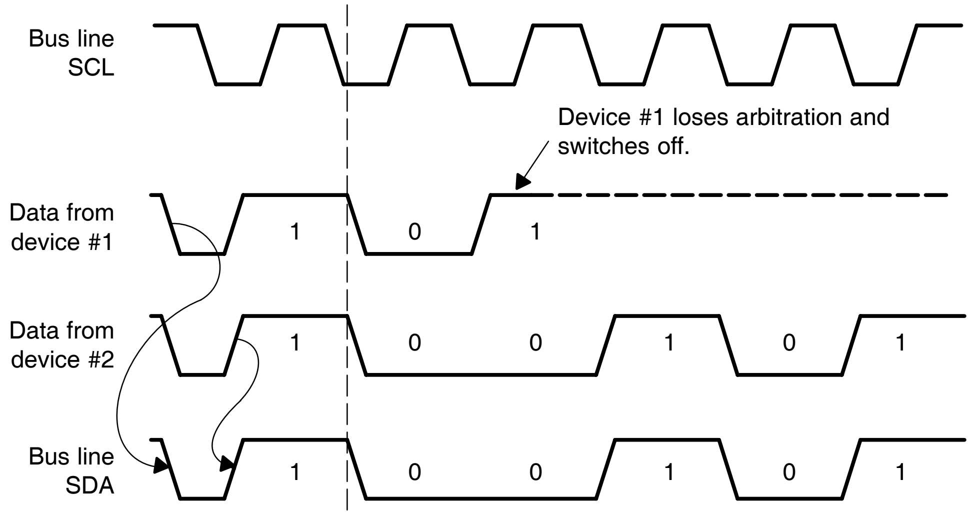
### 1.4.5 Slave Receiver

In this mode serial data bits received on the SDA bus are shifted synchronously with the clock pulses on the SCL, which are generated by the master device. It does not generate the clock, but it can hold the clock line SCL low while intervention of the local host is required following the reception of a byte.

### 1.4.6 Arbitration

If two or more master transmitters start a transmission on the same bus almost simultaneously, an arbitration procedure is invoked. The arbitration procedure uses the data presented on the serial bus by the competing transmitters. When a transmitter senses that a high signal it has presented on the bus has been overruled by a low signal, it switches to the slave receiver mode. Figure 7 shows the arbitration procedure between two devices. The arbitration procedure gives priority to the device that transmits the serial data stream with the lowest binary value. If two or more devices send identical first bytes, arbitration continues on the subsequent bytes.

*Figure 7. Arbitration Procedure Between Two Master Transmitters*

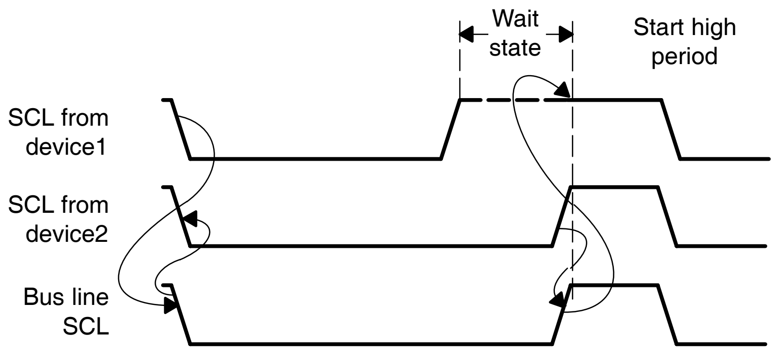


## 1.4.7 I$^2$C Clock Generation and Synchronization

Under normal conditions, only one master device generates the clock signal (SCL). During the arbitration procedure, however, there are two or more master devices and the clock must be synchronized so that the data output can be compared. The wired-AND property of the clock line means that a device that first generates a low period of the clock line overrules the other devices. At this high/low transition, the clock generators of the other devices are forced to start generation of their own low period. The clock line then is held low by the device with the longest low period, while the other devices that finish their low periods must wait for the clock line to be released before starting their high periods. A synchronized signal on the clock line 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 can slow down a fast master and the slow device can create enough time to store a received byte or to prepare a byte to be transmitted. Figure 8 illustrates the clock synchronization.

*Figure 8. Synchronization of Two I2C Clock Generators*



## 2 OMAP5910 I2C (Master/Slave I2C Controller)

The multimaster I2C peripheral provides an interface between the peripheral bus and any I2C-bus compatible devices that connect via the I2C serial bus. External components attached to the I2C bus can serially transmit/receive up to 8-bit data to/from the local host device through the two-wire I2C interface.

This I2C peripheral supports any slave or master I2C-compatible device. Figure 2 shows an example of a system with multiple I2C-compatible devices in which the I2C serial ports are all connected together for a two-way transfer from one device to other devices.

### 2.1 I2C Controller Features

The main features of the I2C controller are as follows:

- Compliant with Philips I2C specification version 2.1 [1]
- Support standard mode (up to 100 kbit/s) and Fast mode (up to 400 kbit/s)
- 7-bit and 10-bit device addressing modes
- General call
- Start/Restart/Stop
- Multimaster transmitter/slave receiver mode
- Multimaster receiver/slave transmitter mode
- Combined master transmit/receive and receive/transmit mode
- Built-in FIFO for buffered read or write
- Module enable/disable capability
- Programmable clock generation
- 16-bit wide access to maximize bus throughput
- Designed for low power
- Two DMA channels
- Wide interrupt capability

The present $I^2C$ does *not* support:

- ❏ High-speed (HS) mode for transfer up to 3.4M bits
- ❏ C-bus compatibility mode.

## 2.2 Data Format

The $I^2C$ controller operates in a 16-bit word data format (byte-write access supported for the last access), and supports endianism.

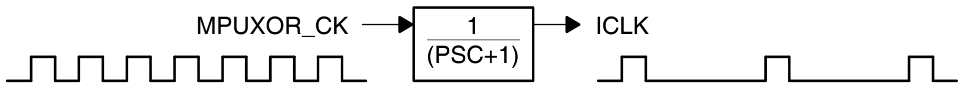
## 2.3 $I^2C$ Reset

The I2C_EN bit in the $I^2C$ configuration register (I2C_CON) can also reset the $I^2C$ module. When the system bus reset is removed (RESET_ = 1), I2C_EN = 0 keeps the $I^2C$ module in reset state.

## 2.4 Prescaler (ICLK)

The $I^2C$ module is operated with an internal ~12 MHz clock (ICLK). This clock is generated via the $I^2C$ prescaler block. ICLK must be in the range from 7 MHz to 12 MHz. This is necessary for proper operation of the I2C module. The prescaler consists of an 8-bit register; I2C_PSC is used for dividing down the system peripheral clock (MPUXOR_CK) to obtain a ~12 MHz clock for the $I^2C$ module (see Figure 9).

*Figure 9. Prescale Sampling Clock Divider Value*



| | |
|---|---|
| 0x0: | Divide by 1 |
| 0x1: | Divide by 2 |
| ↓ | ↓ |
| 0xFF: | Divide by 256 |

Values after reset are low (All 8 bits).

### 2.4.1 Noise Filter

The noise filter suppresses any noise with a duration of 50 ns or less. It is designed to suppress noise with one ICLK assuming the lower and upper limits of ICLK are 7 MHz to 12 MHz.

## 2.5 $I^2C$ Interrupts

The $I^2C$ module generates five types of interrupt: arbitration-lost, no-acknowledge, registers-ready-for-access, receive, and transmit. These five interrupts are accompanied with five interrupt masks and flags defined in the I2C_IE and I2C_STAT registers respectively.

An **arbitration-lost interrupt** (AL) is generated when the I$^2$C arbitration procedure is lost.

A **no-acknowledge interrupt** (NACK) is generated when the master I$^2$C does not receive an acknowledge from the receiver.

A **registers-ready-for-access interrupt** (ARDY) is generated by the I$^2$C when the previously programmed address, data and command have been performed and the status bits have been updated. This interrupt is used to let the local host know that the I$^2$C registers are ready to be accessed.

A **receive interrupt/status** (RRDY) is generated when there is received data ready to be read by the MPU from the I2C_DATA register. This bit can also be polled by the MPU to read the received data from the I2C_DATA register.

**A transmit interrupt/status** (XRDY) is generated when the MPU needs to put data in the I2C_DATA register after the transmitted data has been shifted out on the SDA pin. This bit can also polled by the MPU to write the next transmitted data into the I2C_DATA register.

The interrupt vector register, I2C_IVR, contains one of the binary-coded-interrupt-vectors to indicate which interrupt has occurred. Reading the I2C_IVR clears the interrupt flag; if other interrupts are pending, a new interrupt is generated. If there is more than one interrupt flag, reading the I2C_IVR clears the highest priority interrupt flag.

The I$^2$C interrupt signal (I2C_IRQ) is one MPU pulse-clock-wide active-high signal. It must be considered an edge-sensitive input by the interrupt handler.

## 2.6 DMA Events

The I$^2$C module can generate two DMA requests events, read (I2C_DMA_RX) and write (I2C_DMA_TX), that can be used by the DMA controller to synchronously read received data from the I2C_DATA and write transmitted data to the I2C_DATA register. The DMA read and write requests are generated in a similar manner as RRDY and XRDY.

The I$^2$C DMA request signals (I2C_DMA_TX and I2C_DMA_RX) are one MPU-pulse-clock-wide, active high signals for every new 16-bit word to be read or written in the FIFOs. They must be considered as edge sensitive inputs by the DMA.

## 2.7 I$^2$C Registers

Table 3 lists the I$^2$C registers. Tables 4 through 22 describe the register bits.

*Table 3. I$^2$C Registers*

| Register | Description | Access | Offset Address |
|---|---|---|---|
| I2C_REV† | I$^2$C module version | R | 0x00 |
| I2C_IE | I$^2$C interrupt enable | R/W | 0x04 |
| I2C_STAT | I$^2$C status | R | 0x08 |
| I2C_IV | I$^2$C interrupt vector | R | 0x0C |
| Reserved | | | 0x10 |
| I2C_BUF | I$^2$C buffer configuration | R/W | 0x14 |
| I2C_CNT | I$^2$C data counter | R/W | 0x18 |
| I2C_DATA | I$^2$C data access | R/W | 0x1C |
| Reserved† | | | 0x20 |
| I2C_CON | I$^2$C configuration | R/W | 0x24 |
| I2C_OA | I$^2$C own address | R/W | 0x28 |
| I2C_SA | I$^2$C slave address | R/W | 0x2C |
| I2C_PSC | I$^2$C clock prescaler | R/W | Ox30 |
| I2C_SCLL | I$^2$C SCL low time control | R/W | 0x34 |
| I2C_SCLH | I$^2$C SCL high time control | R/W | 0x38 |
| I2C_SYSTEST | I$^2$C system test | R/W | 0x3C |

† Writing to this register prevents subsequent register accesses to the I2C peripheral.

### 2.7.1 I$^2$C Module Version Register (I2C_REV)

The read-only I$^2$C module version register (I2C_REV) contains the hard coded revision number of the module. A write to this register has no effect. This 8-bit field (7:0) indicates the revision number of the current I$^2$C controller module. Its value is fixed by hardware and corresponds to the RTL revision of this module.

The four LSBs indicate a minor revision.

The four MSBs indicate a major revision.

- Ex: 0x10: version 1.0
- 0x11: version 1.1

A reset has no effect on the value returned. Writing to this register prevents subsequent register accesses to the I2C peripheral.

*Table 4. I$^2$C Module Version Register (I2C_REV)*

| Bits | Field | Description |
|---|---|---|
| 15–8 | – | Reserved |
| 7-0 | REV | Module version number |

### 2.7.2 I$^2$C Interrupt Enable Register (I2C_IE)

The read/write I$^2$C interrupt enable register (I2C_IE) controls interrupts mask/unmask function.

*Table 5. I$^2$C Interrupt Enable Register (I2C_IE)*

| Bits | Field | Description |
|---|---|---|
| 15–5 | – | Reserved |
| 4 | XRDY_IE | Transmit data ready interrupt enable |
| 3 | RRDY_IE | Receive data ready interrupt enable |
| 2 | ARDY_IE | Register access ready interrupt enable |
| 1 | NACK_IE | No acknowledgment interrupt enable |
| 0 | AL_IE | Arbitration lost interrupt enable |

Common to all bits:

When a bit location is set to 1 by the MPU, an interrupt is signaled to the MPU if the corresponding bit location in the I$^2$C status register (I2C_STAT) is asserted to 1 by the core.

If set to 0 the interrupt is masked and not signaled to the MPU.

- 0: Interrupt disabled
- 1: Interrupt enabled

Values after reset are low (all bits)

### 2.7.3 I$^2$C Status Register (I$^2$C_STAT)

The read-only I$^2$C status register (I2C_STAT) provides core status information for interrupt handling and other I$^2$C control management. This register is always read before reading the I²C interrupt vector (I2C_IV) register itself to retain an accurate status (some bits are cleared following a read into I2C_IV).

*Table 6. I²C Status Register (I2C_STAT)*

| Bits | Field | Description |
|---|---|---|
| 15 | SBD | Single byte data |
| 14–13 | – | Reserved |
| 12 | BB | Bus busy |
| 11 | ROVR | Receive overrun |
| 10 | XUDF | Transmit underflow |
| 9 | AAS | Address as slave |
| 8 | AD0 | Address zero |
| 7:5 | – | Reserved |
| 4 | XRDY | Transmit data ready |
| 3 | RRDY | Receive data ready |
| 2 | ARDY | Register access ready |
| 1 | NACK | No acknowledgment interrupt enable |
| 0 | AL | Arbitration lost interrupt enable |

### *Single Byte Data (SBD)*

This read-only bit (15) is set to 1 in slave receive or master receive modes when the last byte that was read from I2C_DATA register contains a single valid byte.

This bit is cleared to 0 by the core when the MPU reads the I2C_IV register if INTCODE is register access ready.

- When SBD = 1, in little-endian data format (BE = 0) the MS byte reads as 0x00 and in big-endian format (BE = 1) the LS byte reads as 0x00.
- Whenever the number of bytes to be received is unknown (slave receiver), the MPU must poll this bit prior to attempting to read I2C_IV.
  - 0: No action
  - 1: Single valid byte in last 16-bit data read

The value after reset is low.

### *Bus Busy (BB)*

This read-only bit (12) indicates the state of the serial bus.

- ❑ In the slave mode, on reception of a start condition, the device sets BB to 1. BB is clear to 0 after reception of a stop condition.
- ❑ In the master mode, the software controls BB. To start a transmission with a start condition, MST, TRX, and STT must be set to 1. To end a transmission with a stop condition, STP must be set to 1. When BB = 1 and STT is set to a 1, a restart condition is generated.
  - ■ 0: Bus is free.
  - ■ 1: Bus is occupied.

The value after reset is low.

### *Receive Overrun (ROVR)*

Receive mode only.

This read-only bit (11) indicates whether the receiver has experienced overrun. Overrun occurs when the receive shift register (ICRSR) is full and the receive FIFO is full. An overrun condition does not result in a data loss, the peripheral holds the bus (low on SCL) to prevent other bytes from being received.

- ❑ ROVR is set to 1 when the I$^2$C has recognized an overrun.
- ❑ ROVR is clear when reading the I2C_DATA register or resetting the I$^2$C (I2C_EN=0).
  - ■ 0: Normal operation
  - ■ 1: Receiver overrun

The value after reset is low.

### *Transmit Underflow (XUDF)*

This read-only bit (10) indicates whether the transmitter has experienced underflow.

- ❑ In the master transmit mode, underflow occurs when the transmit shift register (ICXSR) is empty, the transmit FIFO is empty, and there are still bytes to transmit (DCOUNT $\neq$ 0).
- ❑ In the slave transmit mode, underflow occurs when the transmit shift register (ICXSR) is empty, the transmit FIFO is empty, and there are still bytes to transmit (read request from external I$^2$C master).

- XUDF is set to 1 when the I$^2$C has recognized an underflow. The core holds the line until the underflow cause has disappeared.
- XUDF is clear when writing I2C_DATA register or resetting the I$^2$C (I2C_EN=0).
  - 0: Normal operation
  - 1: Transmit underflow

The value after reset is low.

### Address As Slave (AAS)

This read-only bit (9) is set to 1 by the device when it has recognized its own slave address or an address of all (8) zeros. The AAS bit is reset to 0 by restart or stop.

- 0: No action
- 1: Address as slave

The value after reset is low.

### Address Zero Status (AD0)/General Call

This read-only bit (8) is set to 1 by the device if it detects the address of all eight zeros (that is, general call). The AD0 bit is reset to 0 (default value) when a start or stop condition is detected.

This bit must be checked following a shared NACK/general call Interrupt to determine the source of the interrupt.

When this bit is set to 1, AAS also reads as set to 1.

- 0: No action
- 1: General call

The value after reset is low.

### Transmit Data Ready (XRDY)

Transmit mode only.

XRDY (bit 4) is set to 1 when the I$^2$C peripheral is a master or slave transmitter, the MPU is able to write a new data into the I2C_DATA register, and the transmitter still requires a new data. A master transmitter requests new data if DCOUNT $\neq$ 0, and a slave transmitter requests new data if a read request from external master.

**Note:**

The transmitter requests 2 bytes to be written even if only a single byte is needed. In this case, the “extra” byte must be filled with a dummy 0x00 value that is not transmitted over the I$^2$C line.

XRDY is automatically cleared to 0 by the core when I2C_DATA is written and the transmit FIFO buffer is full. The MPU can also poll this bit to write newly transmitted data into I2C_DATA register.

- 0: Transmit buffer full (or receiver mode)
- 1: Transmit data ready (for write) and byte is needed.

The value after reset is low.

### *Receive Data Ready (RRDY)*

RRDY (bit 3) is set to 1 when the MPU is able to read new data from the I2C_DATA register. RRDY is automatically cleared to 0 by the core when the I2C_DATA is read and the receive FIFO buffer is empty. The MPU can also poll this bit to read the received data in the I2C_DATA register.

In the interrupt mode, the MPU must poll this bit after each read to I2C_DATA to ensure that there is no other data on the FIFO waiting to be read. The RRDY must be cleared to 0 to receive a new RRDY interrupt.

- 0: Receive buffer empty
- 1: Receive data ready (for read)

The value after reset is low.

### *Register Access Ready (ARDY)*

Bit 2, when set to 1, indicates that the previously programmed data and command (receive or transmit, master or slave) have been performed and the status bit has been updated. This flag is used by the MPU to indicate that the I$^2$C registers are ready to be accessed again.

*Table 7. Register Access Ready (ARDY) Set Conditions*

| Mode | Others | ARDY Set Conditions |
|---|---|---|
| Master transmit | STP = 1, RM = 0 | DCOUNT=0 |
| Master receive | STP = 1, RM = 0 | DCOUNT = 0 and receiver FIFO empty |
| Master transmit or receive | STP = 0, RM = 0 | DCOUNT passed 0 |

*Table 7. Register Access Ready (ARDY) Set Conditions (Continued)*

| Mode | Others | ARDY Set Conditions |
|---|---|---|
| Master transmit or receive | RM=1 | Never |
| Slave transmit | – | Stop condition received from master |
| Slave receive | – | Stop condition and receiver FIFO empty |

This bit is cleared to 0 by the core with a read of the matching interrupt vector in I2C_IV register.

- ❑ 0: No action
- ❑ 1: Access ready

The value after reset is low.

### *No Acknowledgment (NACK)*

The no acknowledge flag bit (1) is set when the hardware detects that no acknowledge has been received.

This bit is cleared to 0 by the core with a read of the matching interrupt vector in I2C_IV register.

- ❑ 0: Normal/no action required
- ❑ 1: NACK

The value after reset is low.

When a NACK occurs, the system has to perform the following actions to recover:

1) Read the INTCODE in the I2C_IV register to release NACK in I2C_STAT.

2) Write to the STP bit in the I2C_CON register to release I$^2$C data line.

   The NACK and AL bits in the I2C_CON register should not be polled because an update could be missed. These bits require an interrupt process. The INTCODE field in the I2C_IV register should be read before any action is taken in the subroutine.

### *Arbitration Lost (AL)*

The arbitration lost flag bit is set to 1 when the device in the master transmitter mode senses it has lost an arbitration when two or more transmitters start a transmission almost simultaneously or when the I$^2$C attempts to start a transfer while BB (bus busy) is 1.

When this bit is set to 1 due to arbitration lost, the MST/STP bits are automatically cleared by the core and the I$^2$C becomes a slave receiver.

The BB bit is cleared to 0 by the core with a read of the matching interrupt vector in I2C_IV register.

- ❑ 0: Normal/no action required
- ❑ 1: Arbitration lost

The value after reset is low.

## 2.7.4 I$^2$C Interrupt Vector Register (I2C_IV)

*Table 8. I$^2$C Interrupt Vector Register (I2C_IV)*

| Bits | Field | Description |
|---|---|---|
| 15–3 | – | Reserved |
| 2–0 | INTCODE | Interrupt code |

### *Interrupt Code (INTCODE)*

The binary-coded-interrupt vector (bit 2–> 0) indicates which interrupt has occurred. Reading the I2C_IV clears the interrupt flag. If other interrupts are pending, a new interrupt is generated. If there is more than one interrupt flag, reading the I2C_IV clears the highest priority interrupt flag.

The values of all 3 bits are low after reset.

### *I²C Buffer Configuration Register (I2C_BUF)*

*Table 9. Interrupt Code (INTCODE) Conditions*

| Interrupt Code | Interrupt Occurred | Priority |
|---|---|---|
| 000 | None | – |
| 001 | Arbitration lost interrupt | Highest |
| 010 | No acknowledgement interrupt/general call | ↓ |
| 011 | Register access ready interrupt | |
| 100 | Receive data ready interrupt | |
| 101 | Transmit data ready interrupt | Lowest |
| Others | Reserved | – |

## 2.7.5 I²C Buffer Configuration Register (I2C_BUF)

The read/write I²C buffer configuration register (I2C_BUF) enables DMA transfers.

*Table 10. I²C Buffer Configuration Register (I2C_BUF)*

| Bits | Field | Description |
|---|---|---|
| 15 | RDMA_EN | Receive DMA channel enable |
| 14−8 | – | Reserved |
| 7 | XDMA_EN | Transmit DMA channel enable |
| 6−0 | – | Reserved |

### *Receive DMA Channel Enable (RDMA_EN)*

When bit 15 is set to 1, the receive DMA channel is enabled and the receive data ready interrupt is automatically disabled (RRDY_IE bit cleared).

- 0: Receive DMA channel disabled
- 1: Receive DMA channel enabled

The value after reset is low.

### *Transmit DMA Channel Enable (XDMA_EN)*

When this bit is set to 1, the transmit DMA channel is enabled and the transmit data ready interrupt is automatically disabled (XRDY_IE bit cleared).

- 0: Transmit DMA channel disabled
- 1: Transmit DMA channel enabled

The value after reset is low.

The read/write I$^2$C data counter register (I2C_CNT) controls the number of bytes in the I$^2$C data payload.

## 2.7.6 I$^2$C Data Counter Register (I2C_CNT)

*Table 11. I$^2$C Data Counter Register (I2C_CNT)*

| Bits | Field | Description |
|---|---|---|
| 15−0 | DCOUNT | Data count |

### *Data Count (DCOUNT)*

Master mode only (receive or transmit).

This 16-bit countdown counter decrements by 1 for every byte received or sent. A write initializes DCOUNT to a saved initial value. A read returns the number of bytes that are yet to be received or sent. A read into DCOUNT returns the initial value only before a start condition and after a stop condition.

When DCOUNT reaches 0, the core generates a stop condition if a stop condition was specified (STP = 1) and the ARDY status flag is set to 1.

If STP = 0, then the I$^2$C asserts SCL = 0 when DCOUNT reaches 0. The MPU can then reprogram DCOUNT to a new value and resume sending or receiving data with a new start condition (restart). This process repeats until the STP is set to 1 by the LH.

The ARDY flag is set each time DCOUNT reaches 0 and DCOUNT is reloaded to its initial value.

In slave mode (receive or transmit), DCOUNT is not used.

- 0x0: Reserved value. Do not use this setting.
- 0x1: Data counter = 1 bytes.
- ↓▯ ↓
- 0xFFFF: Data counter = 65535 bytes ($2^{16}$ -1)

Note that DCOUNT is a *don't care* when RM is set to 1.

The values after reset are low (all 16 bits).

The I$^2$C data access register (I2C_DATA) is the entry point for the MPU to read data from, or write data into, the FIFO buffer. The FIFO size is 2x16bits (4 bytes). Bytes within a word are stored and read in little-endian format (I2C_CON:BE=0) or big-endian format (I2C_CON:BE=1).

### 2.7.7 I$^2$C Data Access Register (I2C_DATA)

*Table 12. I$^2$C Data Access Register (I2C_DATA)*

| Bits | Field | Description |
|---|---|---|
| 15−0 | DATA | Transmit/Receive FIFO data |

When read, this register contains the received I$^2$C data packet (1 or 2 bytes). This register must be accessed in 16-bit mode by the LH. In case of an odd number of bytes received to read, the upper byte of the last access always reads as 0x00. The MPU must check the SBD status bit in I2C_STAT register to flush this null byte.

When written to, this register contains the byte(s) value(s) to transmit over the I$^2$C data line (1or 2 bytes). This register must be accessed in 16-bit mode except for the last byte in case of an odd number of bytes to transmit. The last byte of the data packet may be written using a byte write access or a 16-bit write access.

When writing to the FIFO, the last data transfer must be a 16-bit transfer when it is written by the DMA. It can either be an 8-bit or 16-bit transfer when it is written by the MPU. When an odd number of bytes is to be transferred, the DMA uses all 16-bit transfers and fills the unused byte (upper or lower byte according to the selected endianism) of the last 16-bit transfers with all 0s.

In SYSTEST loop back mode (I2C_SYSTEST:TMODE=11) this register is also the entry/receive point for the data.

The values after reset are low (all 16 bits).

A read access when the buffer is empty returns the previous read data value. A write access when the buffer is full is ignored. In both events, the FIFO pointers are not updated and a remote access error (hardware error) is generated (access qualifier). No remote error is generated if the local host performs a 16-bit access if the buffer contains a single byte.

### 2.7.8 I$^2$C Configuration Register (I2C_CON)

*Table 13. I$^2$C Configuration Register (I2C_CON)*

| Bits | Field | Description |
|---|---|---|
| 15 | I2C_EN | I$^2$C module enable |
| 14 | BE | Big-endian mode |
| 13−12 | Reserved | |
| 11 | STB | Start byte mode (master mode only) |
| 10 | MST | Master/slave mode |
| 9 | TRX | Transmitter/receiver mode (master mode only) |
| 8 | XA | Expand address |
| 7−3 | Reserved | |
| 2 | RM | Repeat mode (master mode only) |
| 1 | STP | Stop condition (master mode only) |
| 0 | STT | Start condition (master mode only) |

#### *I$^2$C Module Enable (I2C_EN)*

When this bit (15) is set to 0, the I$^2$C controller is not enabled and reset. When 0, the receive and transmit FIFOs are cleared and all status bits are set to their default values.

The local host must set this bit to 1 for normal operation.

- 0: I$^2$C controller in reset
- 1: I$^2$C module enabled

The value after reset is low.

#### *I$^2$C Big-Endian (BE)*

When this bit (14) is 0 (default), the FIFO is accessed in little-endian format. In transmit mode, the LSB (I2C_DATA[7:0]) is transmitted first and the MSB (I2C_DATA[15:8]) is transmitted in 2nd position over the I$^2$C line. Conversely, in receive mode, the 1st or odd byte received (1, 3, 5…) is stored in the LSB position and the 2nd or even byte received in the MSB position.

When the MPU sets this bit to a 1, the FIFO is accessed in big-endian format. In the transmit mode, the MSB (I2C_DATA[15:8]) is transmitted first and the

LSB (I2C_DATA[7:0]) is transmitted in 2nd position over the I$^2$C line. Conversely, in receive mode, the 1st or odd byte received (1,3, 5…) is stored in the MSB position and the 2nd or even byte received in the LSB position.

- ❑ 0: Little-endian mode
- ❑ 1: Big-endian mode

The value after reset is low.

### *Start Byte (STB)*

Master mode only.

The start byte mode bit (11) is set to 1 by the local host to configure the I$^2$C in start byte mode (I2C_SA=00000001). See the Philips I$^2$C specification for more details.

- ❑ 0: Normal mode
- ❑ 1: Start byte mode

The value after reset is low.

### *Master/Slave Mode (MST)*

When bit 10 is cleared, the I$^2$C controller is in the slave mode and the serial clock (SCL) is received from the master device.

When this bit is set, the I$^2$C controller is in the master mode and it generates the serial clock.

Once set, this bit is automatically cleared by a stop condition.

- ❑ 0: Slave mode
- ❑ 1: Master mode

The value after reset is low.

### *Transmitter/Receiver Mode (TRX)*

Master mode only.

When bit 9 is cleared, the I$^2$C controller is in the receiver mode and data on data line SDA is shifted into the receiver FIFO and can be read from I2C_DATA register.

When this bit is set, the I$^2$C controller is in the transmitter mode and the data written in the transmitter FIFO via I2C_DATA is shifted out on data line SDA.

- ❑ 0: Receiver mode
- ❑ 1: Transmitter mode

The value after reset is low.

Table 14 defines the operating modes.

*Table 14. Operating Modes*

| MST | TRX | Operating Modes |
|---|---|---|
| 0 | x | Slave receiver |
| 0 | x | Slave transmitter |
| 1 | 0 | Master receiver |
| 1 | 1 | Master transmitter |

### *Expand Address (XA)*

When set, bit 8 expands the address to 10-bits.

- ❑ 0: 7-bit address mode
- ❑ 1: 10-bit address mode

The value after reset is low.

### *Repeat Mode (RM)*

Mater mode only.

Bit 2 is set to a 1 by the MPU to place the I$^2$C in the repeat mode. In this mode, data is continuously transmitted out of the I2C_DATA transmit register until the STP bit is set to 1 regardless of DCOUNT value. This bit is *don't care* if the I$^2$C is configured in slave mode.

- ❑ 0: Normal mode
- ❑ 1: Repeat mode

The value after reset is low.

*Table 15. Repeat Mode Conditions*

| RM | STT | STP | Conditions | Bus Activities | Mode |
|---|---|---|---|---|---|
| 0 | 0 | 0 | Idle | None | NA |
| 0 | 0 | 1 | Stop | P | NA |
| 0 | 1 | 0 | (Re)Start | S-A-D..(n)..D | Repeat n |
| 0 | 1 | 1 | (Re)Start-Stop | S-A-D..(n)..D-P | Repeat n |
| 1 | 0 | 0 | Idle | none | NA |
| 1 | 0 | 1 | Stop | P | NA |
| 1 | 1 | 0 | (Re)Start | S-A-D-D-D….. | Continuous |
| 1 | 1 | 1 | Reserved | None | NA |

### Stop Condition (STP)

Master mode only.

Bit 1 can be set to 1 by the MPU to generate a stop condition. It is reset to 0 by the hardware after the stop condition has been generated. The stop condition is generated when DCOUNT passes 0.

- 0: No action or stop condition detected
- 1: Stop condition queried

The value after reset is low.

### Start Condition (STT)

Master mode only.

Bit 0 can be set to a 1 by the MPU to generate a start condition. It is reset to 0 by the hardware after the start condition has been generated. The start/stop bits can be configured to generate different transfer formats. The STT and STP can be used to terminate the repeat mode.

- 0: No action or start condition generated
- 1: Start

The value after reset is low.

*Table 16. STT Settings*

| STT | STP | Conditions | Bus Activities |
|---|---|---|---|
| 1 | 0 | Start | S-A-D |
| 0 | 1 | Stop | P |

*Table 16. STT Settings (Continued)*

| STT | STP | Conditions | Bus Activities |
|---|---|---|---|
| 1 | 1 | Start/stop (COUNT= n) | S-A-D..(n)..D-P |
| 1 | 0 | Start (DCOUNT= n) | S-A-D..(n)..D |

DCOUNT is data count value.

### 2.7.9 I²C Own Address Register (I2C_OA)

The I²C address register (I2C_OA) specifies the module I²C 7-bit or 10-bit address (own address).

*Table 17. I²C Own Address Register (I2C_OA)*

| Bits | Field | Description |
|---|---|---|
| 15–10 | Reserved | |
| 9–0 | OA | Own address |

This field (bits 9-0) specifies either:

❑ A 10-bit address coded on OA[9:0] when XA (expand address, I2C_MCR[8]) is set to 1.

❑ A 7-bit address coded on OA[6:0] when XA (expand address, I2C_MCR[8]) is set to 0. In this case, OA[9:7] bits must be set to 000 by application software.

The values after reset are low (all 10 bits).

### 2.7.10 $I^2C$_Slave Address Register (I2C_SA)

The $I^2C$ slave address register (I2C_SA) specifies the addressed $I^2C$ module 7-bit or 10-bit address (slave address).

*Table 18. $I^2C$ Slave Address Register (I2C_SA)*

| Bits | Field | Description |
|---|---|---|
| 15−10 | Reserved | |
| 9−0 | SA | Slave address<br>This field (bits 9:0) specifies either:<br>❑ A 10-bit address coded on SA[9:0] when XA (expand address, I2C_MCR[8]) is set to 1.<br>❑ A 7-bit address coded on SA[6:0] when XA (expand address, I2C_MCR[8]) is set to 0. In this case, SA[9:7] bits must be set to 000 by application software.<br>The values after reset are high (all 10 bits). |

### 2.7.11 $I^2C$ Clock Prescaler Register (I2C_PSC)

The $I^2C$ clock prescaler register (I2C_PSC) register is used to specify the internal clocking of the $I^2C$ peripheral core.

*Table 19. $I^2C$ Clock Prescaler Register (I2C_PSC)*

| Bits | Field | Description |
|---|---|---|
| 15−8 | Reserved | |
| 7−0 | PSC | Prescale sampling clock divider value<br>The core (bits 7-0) uses this 8-bit value to divide the peripheral clock (MPUXOR_CK) to generate its own internal sampling clock (ICLK). The core logic is sampled at the clock rate of the system clock for the module divided by (PSC+1):<br>❑ 0x0: Divide by 1<br>❑ 0x1: Divide by 2<br>❑ All other settings are Reserved.<br>The values after reset are low (all 8 bits). |

### 2.7.12 I$^2$C SCL Low-Time Control Register (I2C_SCLL)

This I$^2$C SCL low-time control register (I2C_SCLH) is used to determine the SCL low-time value when master.

*Table 20. I$^2$C SCL Low-Time Control Register (I2C_SCLH)*

| Bits | Field | Description |
|---|---|---|
| 15–8 | Reserved | |
| 7–0 | SCLL | SCL low0x0: 6 * ICLK time period time<br>Master mode only.<br>This 8-bit value (bits 7:0) is used to generate the SCL low-time value ($t_{LOW}$) when the peripheral is operated in master mode.<br>The SCL low-time equals (SCLL+6) * ICLK time period (internal sampling clock rate).<br>❑ 0x0: 6 * ICLK time period<br>❑ 0x1: 7 * ICLK time period<br>❑ ↓□↓<br>❑ 0xFF: 261 * ICLK time period<br>The values after reset are low (all 10 bits). |

### 2.7.13 $I^2C$ SCL High-Time Control Register (I2C_SCLL)

The $I^2C$ SCL high-time control register (I2C_SCLL) determines the SCL high-time value when master.

*Table 21. $I^2C$ SCL High Time Control Register (I2C_SCLH)*

| Bits | Field | Description |
|---|---|---|
| 15−8 | Reserved | |
| 7−0 | SCLH | SCL high time<br>Master mode only.<br>This 8-bit value (bits 7-0) is used to generate the SCL high time value ($t_{HIGH}$) when the peripheral is operated in master mode.<br>The SCL high time equals (SCLH+6) * ICLK time period (internal sampling clock rate).<br>❑ 0x0: 6 * ICLK time period<br>❑ 0x1: 7 * ICLK time period<br>❑ ↓▯↓<br>❑ 0xFF: 261 * ICLK time period<br>The values after reset are low (all 10 bits). |

### 2.7.14 $I^2C$ System Test Register (I2C_SYSTEST)

The $I^2C$ system test register (I2C_SYSTEST) is used to facilitate system level test by overriding some of the standard functional features of the peripheral. It can permit the test of SCL counters, control the signals that connect to I/O pins, or create digital loop-back for self-test when the module is configured in system test (SYSTEST) mode. It also provides stop/no-stop function in debug mode. It is never set for normal $I^2C$ operation.

*Table 22. $I^2C$ System Test Register (I2C_SYSTEST)*

| Bits | Field | Description |
|---|---|---|
| 15 | ST_EN | System test enable |
| 14 | FREE | Free running mode (on breakpoint) |
| 13−12 | TMODE | Test mode select |
| 11−4 | Reserved | |
| 3 | SCL_I | SCL line sense input value |
| 2 | SCL_O | SCL line drive output value |

*Table 22. I$^2$C System Test Register (I2C_SYSTEST) (Continued)*

| Bits | Field | Description |
|---|---|---|
| 1 | SDA_I | SDA line sense input value |
| 0 | SDA_O | SDA line drive output value |

## System Test Enable (ST_EN)

Bit 15 must be set to 1 to permit other system test register bits to be set.

- 0: Normal mode
- 1: System test enabled

The value after reset is low.

## Free Running Mode After Breakpoint (FREE)

Bit 14 is used to determine the state of the I$^2$C controller when a breakpoint is encountered in the HLL debugger. This bit can be set independently of the ST_EN value.

FREE = 0: Stops immediately if SCL is low and keeps driving SCL low whether I$^2$C is master transmitter/receiver. If SCL is high, I$^2$C waits until SCL becomes low and then stops. If the I$^2$C is a slave, it stops when the transmission/ receiving completes.

FREE = 1: The I$^2$C runs free.

- 0: Stop mode (on breakpoint condition)
- 1: Free-running mode

The value after reset is low.

## Test Mode Select (TMODE)

In the normal functional mode (ST_EN = 0), these bits (13-12) are *don't care*. They always read as 00 and a write is ignored.

In the system test mode (ST_EN = 1), these bits can be set according to the following table to permit various system tests.

*Table 23. TMODE Settings*

| TMODE | Mode |
|---|---|
| 00 | Functional mode (default) |
| 01 | Reserved |
| 10 | Test of SCL counters (SCLL, SCLH, PSC) |
| 11 | Loop back mode select + SDA/SCL IO mode select |

The values after reset are low (2 bits).

In the SCL counter test mode, the SCL pin is driven with a permanent clock as
a master with the parameters set in I2C_PSC, I2C_SCLL, and I2C_SCLH registers.

Loopback mode: In the master transmit mode only, data transmitted from the I2C_DATA register (write action) is received in the same I2C_DATA register via an internal path through the one-deep FIFO buffers. The DMA and interrupt requests is normally generated if enabled.

In the SDA/SCL I/O mode, the SCL IO and SDA IO are controlled via the I2C_SYSTEST[3:0] register bits.

### SCL Line Sense Input Value (SCL_I)

In the normal functional mode (ST_EN = 0), this read-only bit (3) always reads as 0.

In the system test mode (ST_EN = 1 and TMODE = 11), this read only-bit returns the logical state taken by the SCL line (either 1 or 0).

The value after reset is low.

### SCL Line Drive Output Value (SCL_O)

In the normal functional mode (ST_EN = 0), this bit (2) is *don't care*, and always reads as 0. Writes are ignored.

In the system test mode (ST_EN = 1 and TMODE = 11), a 0 forces a low level on the SCL line and a 1 puts the I$^2$C output driver in a high-impedance state.

- 0: Force 0 on the SCL data line
- 1: SCL output driver in HI-Z state

The value after reset is low.

### SDA Line Sense Input Value (SDA_I)

In the normal functional mode (ST_EN = 0), this read-only-bit (1) always reads as 0.

In the system test mode (ST_EN = 1 and TMODE = 11), this read-only bit returns the logical state taken by the SDA line (either 1 or 0).

The value after reset is low.

### SDA Line Drive Output Value (SDA_O)

In normal functional mode (ST_EN = 0), this bit (0) is *don't care*, and always reads as 0. Writes are ignored.

In the system test mode (ST_EN = 1 and TMODE = 11), a 0 forces a low level on the SDA line and a 1 puts the $I^2C$ output driver in a high-impedance state.

- 0: Forces 0 on the SDA data line
- 1: SDA output driver in HIZ state

The value after reset is low.

## 3 Programming

## 3.1 Main Program

State after reset:

1) Program the prescaler to obtain an approximately 12-MHz $I^2C$ module clock (I2C_PSC = x; this value is to be calculated and is dependent on the CPU frequency).
   - If using an interrupt for transmit/receive data, enable the interrupt masks.
   - If using DMA for transmit/receive data, enable the DMA and program the DMA controller.
2) Take the $I^2C$ module out of reset (I2C_EN = 1).

Initialization procedure: Configure the $I^2C$ mode register (I2C_CON) bits.

The program clock control registers (I2C_SCLL and I2C_SCLH): Program the $I^2C$ clock to 100K bps or 400K bps (I2C_SCLL = x and I2C_SCLH = x; these values must be calculated and are dependent on the CPU frequency).

- ❑ Configure the address registers:
  - ■ Configure its own address (I2C_OA = x).
  - ■ Configure the slave address (I2C_SA = x).
- ❑ *Program* the transmit data register (I2C_DATA): If in the master transmitter mode, program the data transmit register (I2C_DATA = x).
- ❑ *Configure* the status and mode register (I2C_STAT): Poll the bus-busy (BB) bit in the I$^2$C status register (I2C_STAT); if it is cleared to 0 (bus-not-busy), configure START/STOP condition to initiate a transfer.
- ❑ *Poll receive data*: Poll the receive data ready interrupt flag bit (RRDY) in the I$^2$C status register (I2C_STAT), use the RRDY interrupt, or use the DMA to read the receive data in the data receive register (I2C_DATA).
- ❑ *Poll transmit data*: Poll the transmit data ready interrupt flag bit (XRDY) in the I$^2$C status register (I2C_STAT), use the XRDY interrupt, or use the DMA to write data into the data transmit register (I2C_DATA).

Interrupt subroutines:

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

## 4 Flowcharts

Figure 10 shows the setup procedure and Figures 11 through 21 are the master/slave I$^2$C flowcharts.

*Figure 10. Setup Procedure*

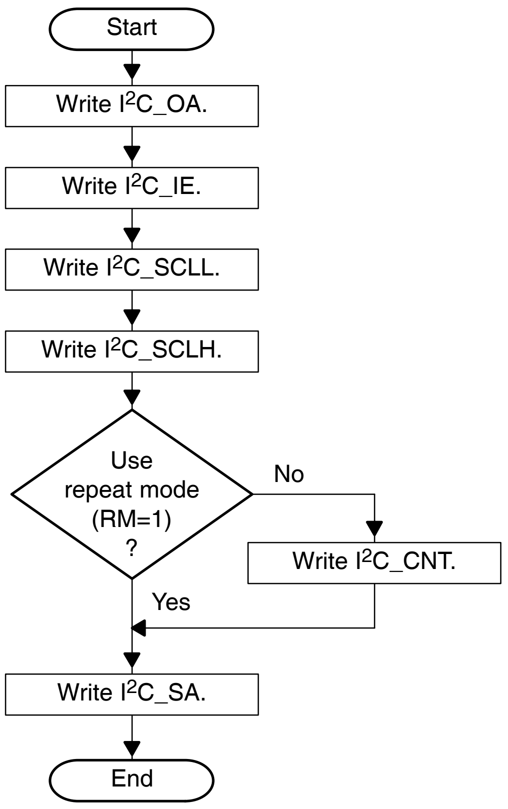
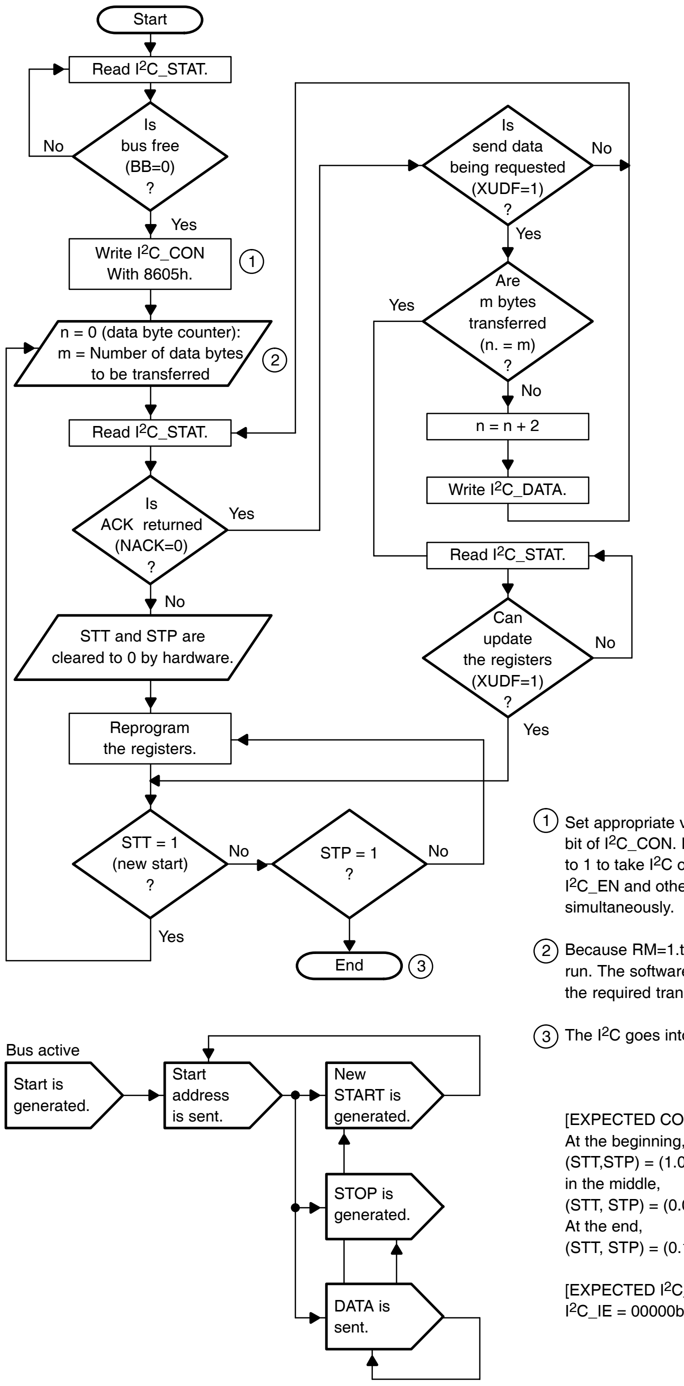


*Figure 11. Master Transmitter Mode, RM = 1*



1 Set appropriate values to every bit of I2C_CON. I2C_EN bit must be set to 1 to take I2C out of reset condition. Setting I2C_EN and other mode bits can be done simultaneously.

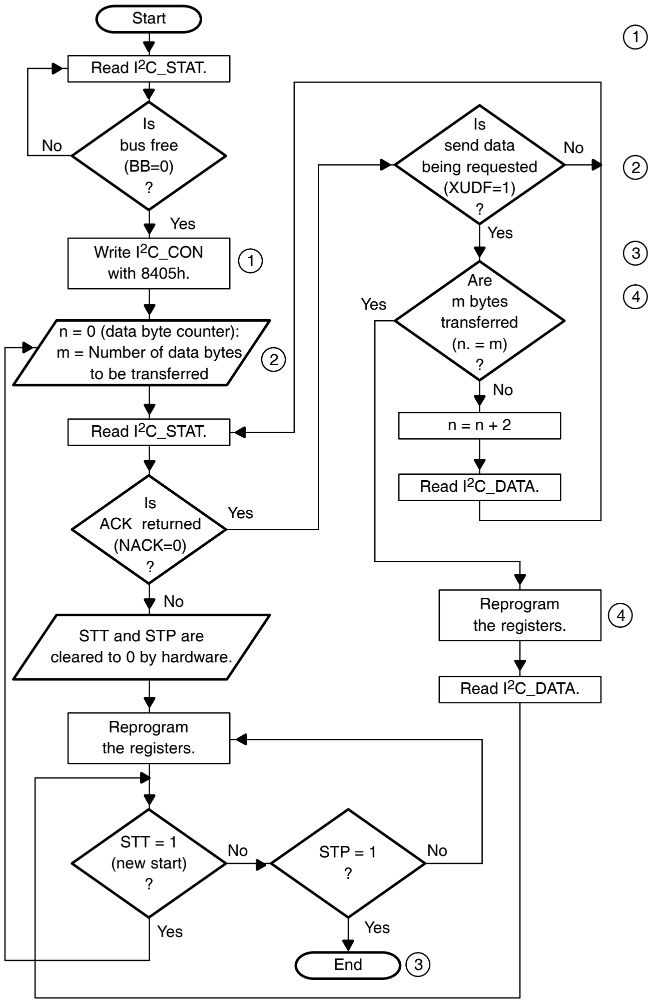
2 Because RM=1.the hardware counter does not run. The software counter counts the number of the required transfer.

3 The I2C goes into slave receiver mode.

[EXPECTED COMMAND]
At the beginning,
(STT,STP) = (1.0)
in the middle,
(STT, STP) = (0.0)
At the end,
(STT, STP) = (0.1)

[EXPECTED I2C_IE]
I2C_IE = 00000b

*Figure 12. Master Receiver Mode, RM = 1, Polling 1 (Software Counter, Number of the Receive Data Fixed)*


Start
Read I2C_STAT.
Is bus free (BB=0) ?
No
Yes
Write I2C_CON with 8405h.
1
n = 0 (data byte counter): m = Number of data bytes to be transferred
2
Read I2C_STAT.
Is ACK returned (NACK=0) ?
Yes
No
STT and STP are cleared to 0 by hardware.
Reprogram the registers.
STT = 1 (new start) ?
No
Yes
STP = 1 ?
No
Yes
End
3
Is send data being requested (XUDF=1) ?
No
Yes
Are m bytes transferred (n. = m) ?
Yes
No
n = n + 2
Read I2C_DATA.
Reprogram the registers.
4
Read I2C_DATA.

1. Set appropriate values to every bit of $I^2C$_CON. $I^2C$_EN bit must be set to 1 to take $I^2C$ out of reset condition. Setting $I^2C$_EN and other mode bits can be done simultaneously.

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

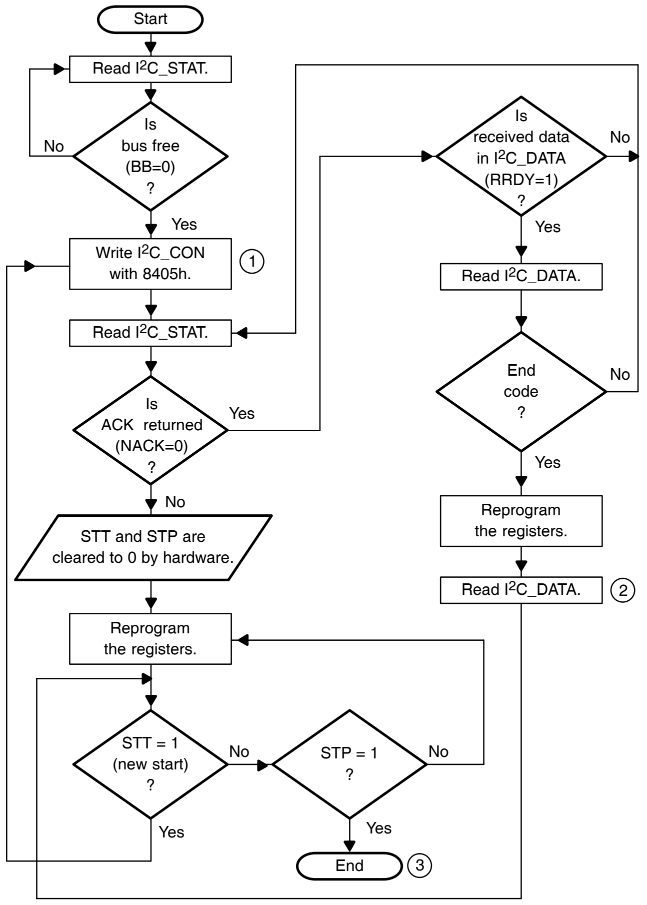
3. The $I^2C$ goes into slave receiver mode.

4. Set STP = 1

[EXPECTED COMMAND]
At the beginning,
(STT,STP) = (1.0)
in the middle,
(STT, STP) = (0.0)
At the end,
(STT, STP) = (0.1)

[EXPECTED $I^2C$_IE]
$I^2C$_IE = 00000b

*Figure 13. Master Receiver Mode, RM =1 , Polling 2 (Number of the Receive Data is Variable, Data Contents Dependent)*



(1) Set appropriate values to every bit of I$^2$C_CON. I$^2$C_EN bit must be set to 1 to take I$^2$C out of reset condition. Setting I$^2$C_EN and other mode bits can be done simultaneously.

(2) Dummy read. The contents of this read data have no meaning.

(3) The I$^2$C goes into slave receiver mode.

[EXPECTED COMMAND]
At the beginning,
(STT,STP) = (1.0)
in the middle,
(STT, STP) = (0.0)
At the end,
(STT, STP) = (0.1)

[EXPECTED I$^2$C_IE]
I$^2$C_IE = 00000b

*Figure 14. Master Transmitter Mode, RM = 0, Polling*

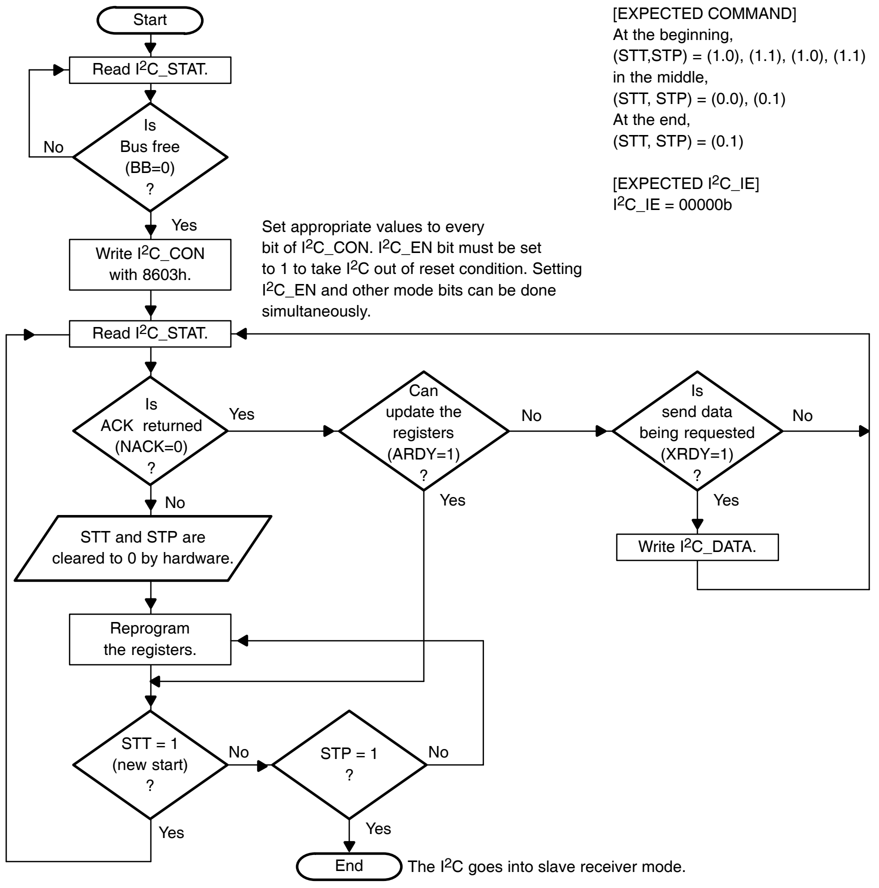
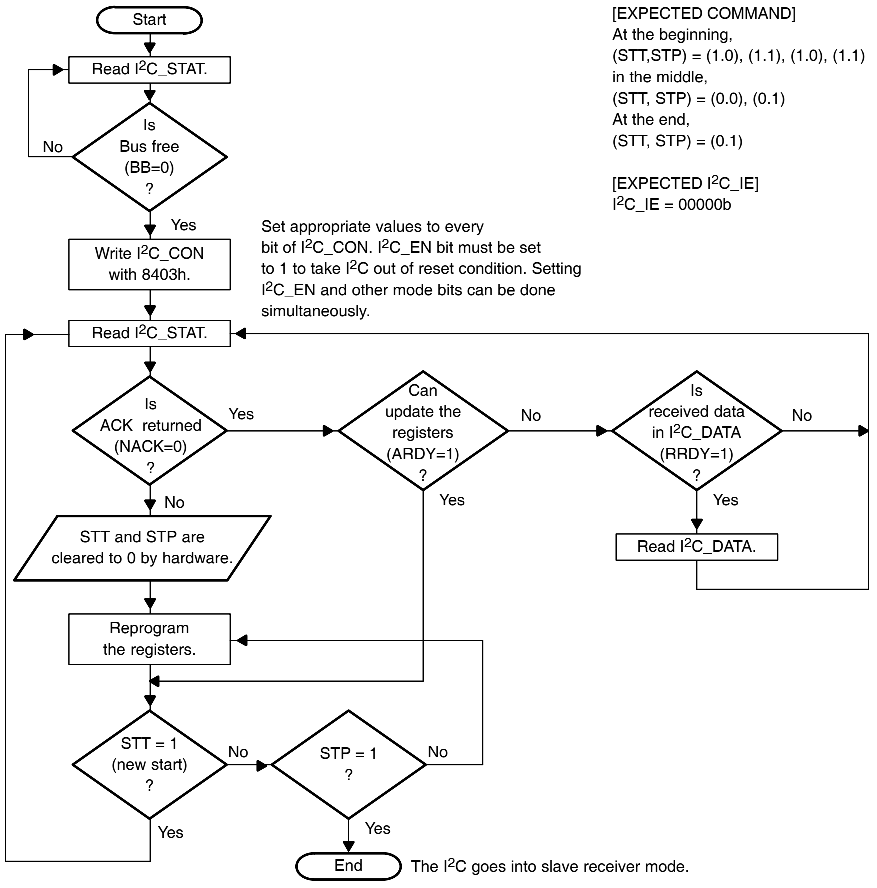

Start
Read I2C_STAT.
Is Bus free (BB=0) ?
No
Yes
Write I2C_CON with 8603h.
Set appropriate values to every bit of I2C_CON. I2C_EN bit must be set to 1 to take I2C out of reset condition. Setting I2C_EN and other mode bits can be done simultaneously.
[EXPECTED COMMAND]
At the beginning,
(STT,STP) = (1.0), (1.1), (1.0), (1.1)
in the middle,
(STT, STP) = (0.0), (0.1)
At the end,
(STT, STP) = (0.1)
[EXPECTED I2C_IE]
I2C_IE = 00000b
Read I2C_STAT.
Is ACK returned (NACK=0) ?
Yes
No
Can update the registers (ARDY=1) ?
No
Yes
Is send data being requested (XRDY=1) ?
No
Yes
Write I2C_DATA.
STT and STP are cleared to 0 by hardware.
Reprogram the registers.
STT = 1 (new start) ?
No
Yes
STP = 1 ?
No
Yes
End
The I2C goes into slave receiver mode.

*Figure 15. Master Receiver Mode, RM = 0, Polling*


Start
Read I2C_STAT.
Is Bus free (BB=0) ?
No
Yes
Write I2C_CON with 8403h.
Read I2C_STAT.
Is ACK returned (NACK=0) ?
Yes
No
Can update the registers (ARDY=1) ?
No
Yes
Is received data in I2C_DATA (RRDY=1) ?
No
Yes
Read I2C_DATA.
STT and STP are cleared to 0 by hardware.
Reprogram the registers.
STT = 1 (new start) ?
No
Yes
STP = 1 ?
No
Yes
End

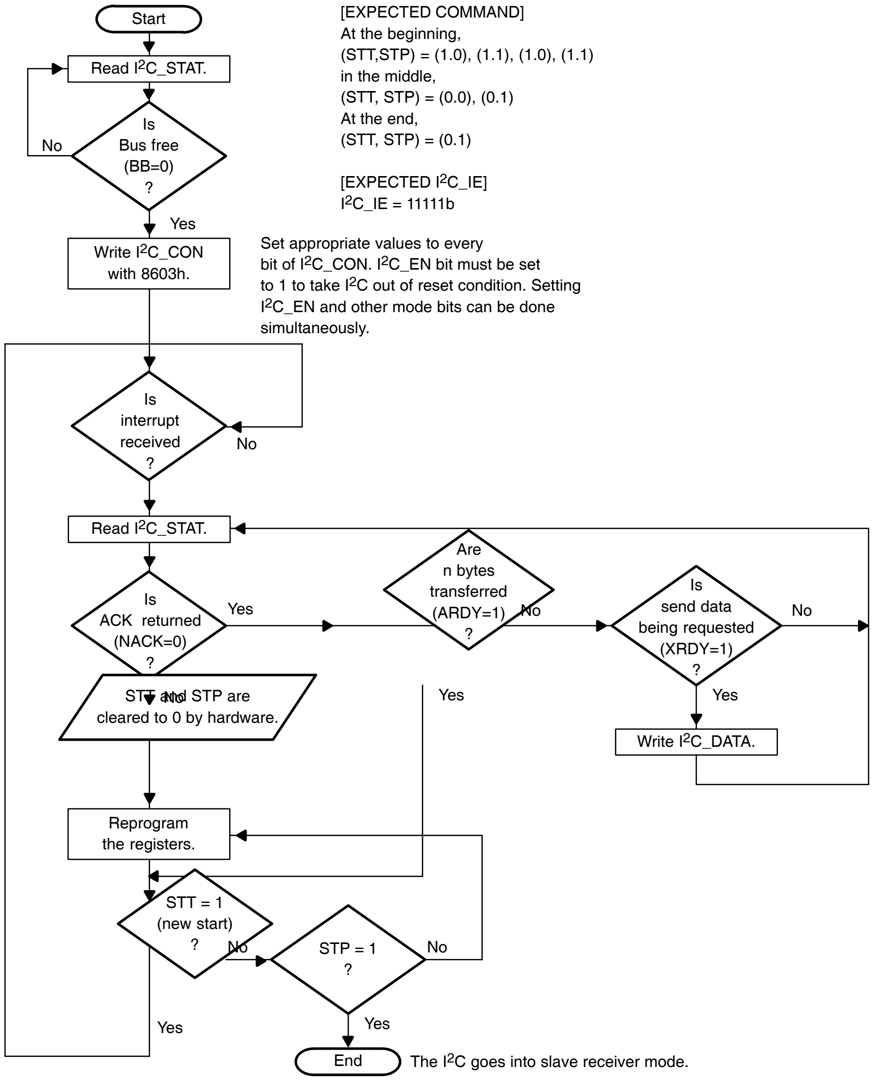
[EXPECTED COMMAND]
At the beginning,
(STT,STP) = (1.0), (1.1), (1.0), (1.1)
in the middle,
(STT, STP) = (0.0), (0.1)
At the end,
(STT, STP) = (0.1)

[EXPECTED I$^2$C_IE]
I$^2$C_IE = 00000b

Set appropriate values to every bit of I$^2$C_CON. I$^2$C_EN bit must be set to 1 to take I$^2$C out of reset condition. Setting I$^2$C_EN and other mode bits can be done simultaneously.

The I$^2$C goes into slave receiver mode.

## *Figure 16. Master Transmitter Mode, RM = 0, Interrupt*



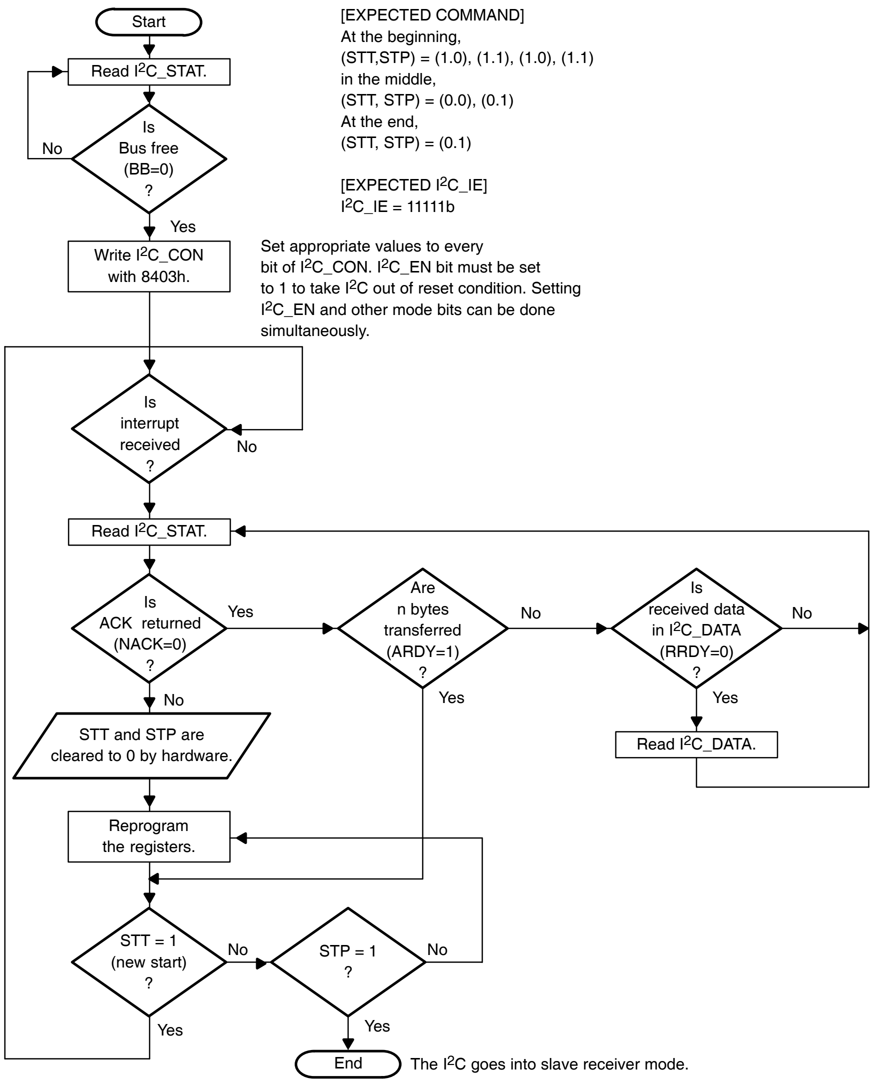
[EXPECTED COMMAND]
At the beginning,
(STT,STP) = (1.0), (1.1), (1.0), (1.1)
in the middle,
(STT, STP) = (0.0), (0.1)
At the end,
(STT, STP) = (0.1)

[EXPECTED I$^2$C_IE]
I$^2$C_IE = 11111b

Set appropriate values to every bit of I$^2$C_CON. I$^2$C_EN bit must be set to 1 to take I$^2$C out of reset condition. Setting I$^2$C_EN and other mode bits can be done simultaneously.

The I$^2$C goes into slave receiver mode.

*Figure 17. Master Receiver Mode, RM = 0, Interrupt*



[EXPECTED COMMAND]
At the beginning,
(STT,STP) = (1.0), (1.1), (1.0), (1.1)
in the middle,
(STT, STP) = (0.0), (0.1)
At the end,
(STT, STP) = (0.1)

[EXPECTED I$^2$C_IE]
I$^2$C_IE = 11111b

Set appropriate values to every bit of I$^2$C_CON. I$^2$C_EN bit must be set to 1 to take I$^2$C out of reset condition. Setting I$^2$C_EN and other mode bits can be done simultaneously.

The I$^2$C goes into slave receiver mode.

## *Figure 18. Master Transmitter Mode, RM = 0, DMA*

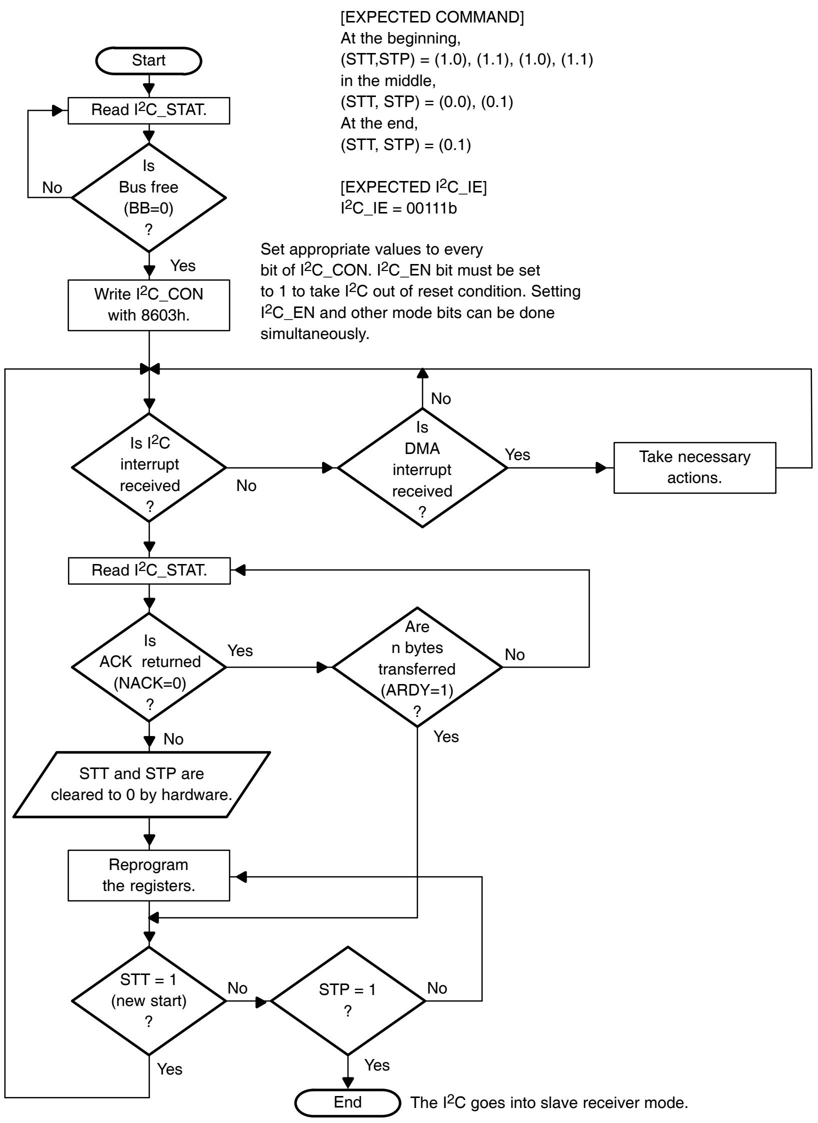

[EXPECTED COMMAND]
At the beginning,
(STT,STP) = (1.0), (1.1), (1.0), (1.1)
in the middle,
(STT, STP) = (0.0), (0.1)
At the end,
(STT, STP) = (0.1)
[EXPECTED I2C_IE]
I2C_IE = 00111b
Start
Read I2C_STAT.
Is Bus free (BB=0) ?
No
Yes
Write I2C_CON with 8603h.
Set appropriate values to every bit of I2C_CON. I2C_EN bit must be set to 1 to take I2C out of reset condition. Setting I2C_EN and other mode bits can be done simultaneously.
Is I2C interrupt received ?
No
Is DMA interrupt received ?
No
Yes
Take necessary actions.
Read I2C_STAT.
Is ACK returned (NACK=0) ?
Yes
Are n bytes transferred (ARDY=1) ?
No
Yes
No
STT and STP are cleared to 0 by hardware.
Reprogram the registers.
STT = 1 (new start) ?
No
Yes
STP = 1 ?
No
Yes
End
The I2C goes into slave receiver mode.

*Figure 19. Master Receiver Mode, RM = 0, DMA*

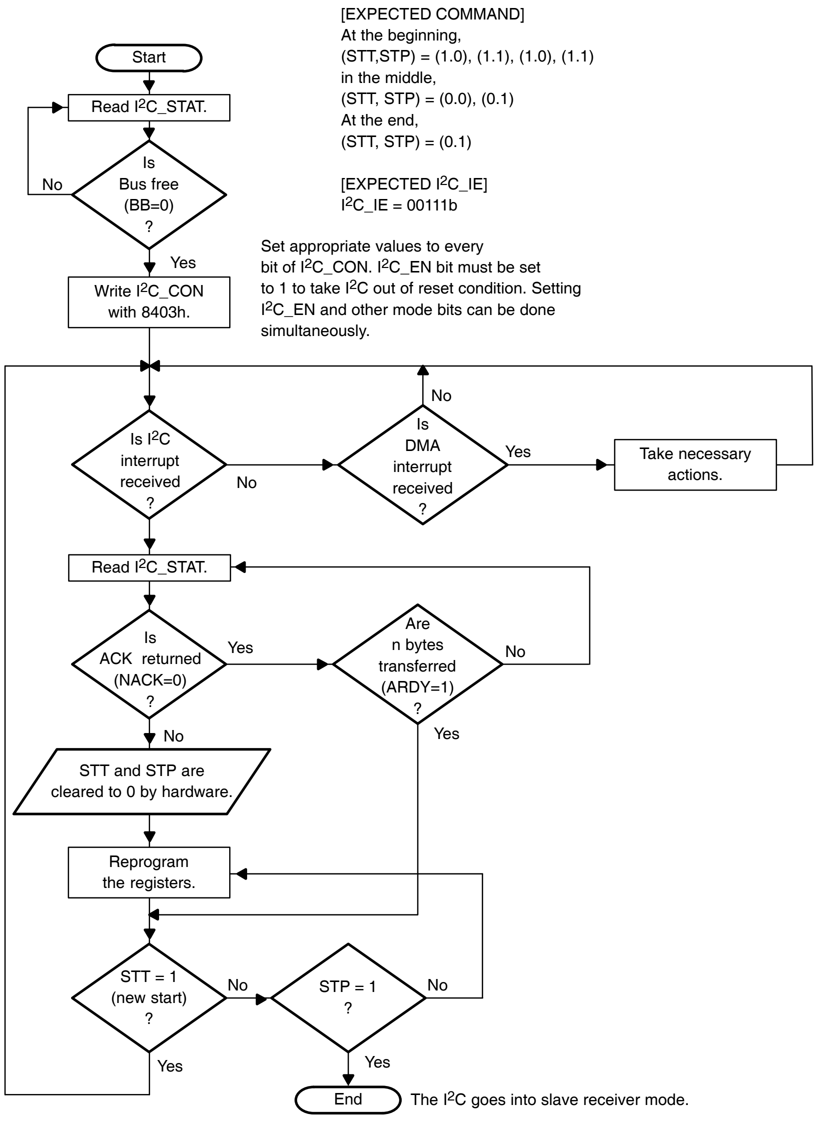

[EXPECTED COMMAND]
At the beginning,
(STT,STP) = (1.0), (1.1), (1.0), (1.1)
in the middle,
(STT, STP) = (0.0), (0.1)
At the end,
(STT, STP) = (0.1)
[EXPECTED I2C_IE]
I2C_IE = 00111b
Start
Read I2C_STAT.
Is Bus free (BB=0) ?
No
Yes
Write I2C_CON with 8403h.
Set appropriate values to every bit of I2C_CON. I2C_EN bit must be set to 1 to take I2C out of reset condition. Setting I2C_EN and other mode bits can be done simultaneously.
Is I2C interrupt received ?
No
Is DMA interrupt received ?
No
Yes
Take necessary actions.
Read I2C_STAT.
Is ACK returned (NACK=0) ?
Yes
No
Are n bytes transferred (ARDY=1) ?
No
Yes
STT and STP are cleared to 0 by hardware.
Reprogram the registers.
STT = 1 (new start) ?
No
Yes
STP = 1 ?
No
Yes
End
The I2C goes into slave receiver mode.

*Figure 20. Slave Transmitter/Receiver Mode-Polling*

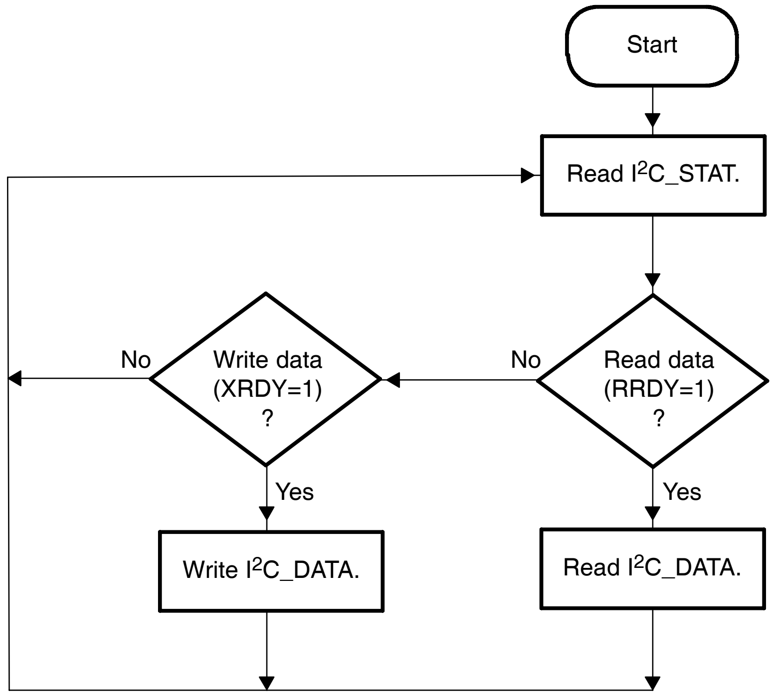
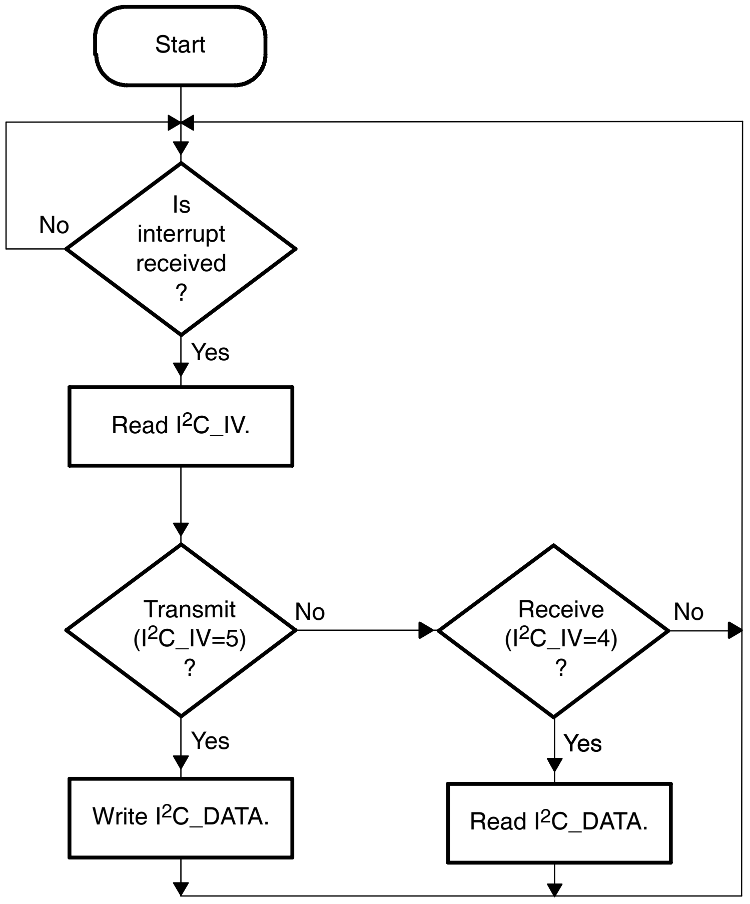


*Figure 21. Slave Transmitter/Receiver Mode-Interrupt*



Appendix A

# Revision History

This document was revised to SPRU681A from SPRU681, which was released in October 2003. The scope of the revisions was limited to technical changes as described in A.1. This appendix lists only revisions made in the most recent version.

## A.1 Changes Made in This Revision

The following changes were made in this revision:

| Page | Additions/Modifications/Deletions |
|---|---|
| 19 | Changed ICLK range from 8MHz to 16 MHz to **7 MHz to 12 MHz** |

# Index