SPI to I2C Bridge

February 2015 Reference Design RD1173

Introduction

I2C and SPI are the two widely used bus protocols in today's embedded systems. The I2C bus has a minimum pin count requirement and therefore a smaller footprint on the board. The SPI bus provides a synchronized serial link with performance in MHz range. As embedded systems are required to support an increasing number of protocols and interfaces, bridge designs targeting popular protocols provide solutions to reduce development time and cost. The SPI to Dual I2C Master Controllers Bridge extends the available I2C ports for the Applications processor.

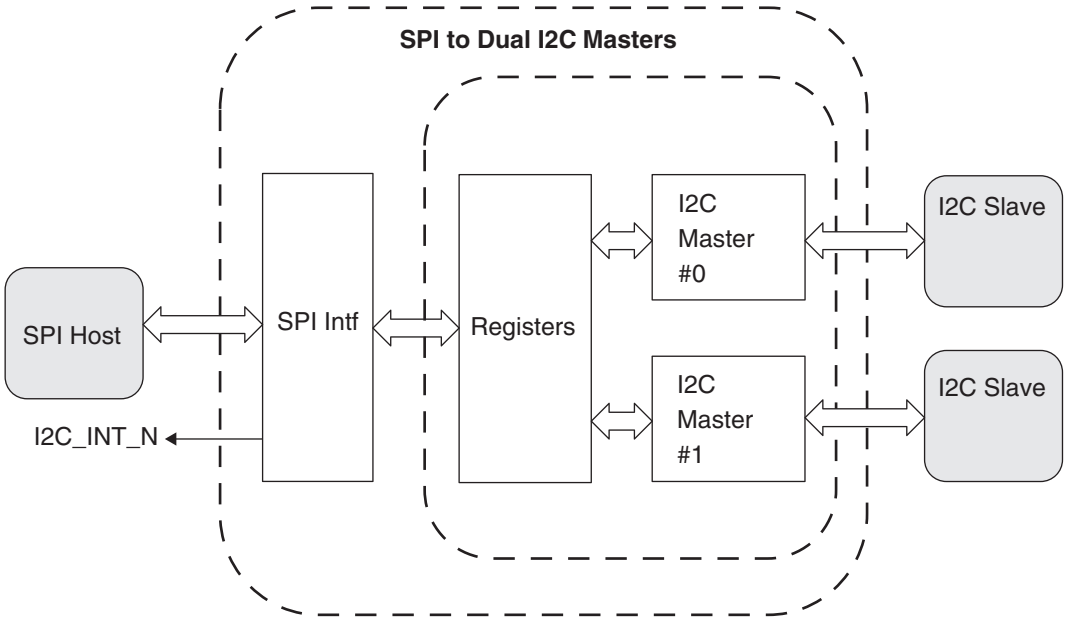
The design is implemented in VHDL. The Lattice iCEcube2™ Place and Route tool integrated with the Synopsys Synplify Pro® synthesis tool is used for the implementation of the design. The design can be targeted to other iCE40™ FPGA product family devices.

Features

- Supports SPI interface up to 25 MHz
- Two compliant I2C masters supports 100 kHz and 400 kHz (Fast Mode)
- Each I2C master operates independently
- Each I2C master supports clock stretching and repeated start operations
- Each I2C master has 8-deep transmit and receive FIFOs for efficient data handling
- Supports interrupt-driven or polling software interfaces
- Single chip, low power, aggressive packaging options

System Block Diagram

Figure 1. System Block Diagram



Signal Description

Table 1. Signal Description

Signal Name	Pin Type	Pull Up/Down Required	Signal Description
SPI_SS_N	Input	Optional External Pull-low for CPU Configura-tion	SPI Interface: Chip Select This signal is generated by the SPI Host to indicate that the bridge is selected for access
SPI_MISO	Output	No	SPI Interface: Master In Slave Out (MISO) This signal allows data transfer from SPI Slave to SPI Host
SPI_MOSI	Input	No	SPI Interface: Master Out Slave In (MOSI) This allows data transfer from SPI host to SPI Slave
SPI_SCK	Input	No	SPI Interface: Clock This clock is generated by the SPI Host to SPI slave. Need to be on a clock pin
SCL0	Bidir, Open Drain	External See I2C spec for value	I2C Master #0 Serial Clock This bidirectional signal is generated by the I2C Master #0. Clock rates up to 400 kHz are supported (“Fast Mode”). The slave may stretch the clock if necessary
SDA0	Bidir, Open Drain	External See I2C spec for value	I2C Master #0: I2C Serial Data This bidirectional signal allows data transfer over the I2C bus.
SCL1	Bidir, Open Drain	External See I2C spec for value	I2C Master #1 Serial Clock This bidirectional signal is generated by the I2C Master #1. Clock rates up to 400 kHz are supported (“Fast Mode”). The slave may stretch the clock if necessary.
SDA1	Bidir, Open Drain	External See I2C spec for value	I2C Master #1: I2C Serial Data This bidirectional signal allows data transfer over the I2C bus.
I2C_INT_N	Output	No	I2C Master: Interrupt Request This active low output is the mechanism for the I2C Master#0 and #1 to inform the application processor that service is required. If necessary, interrupt pins can be separate into I2C0_INT_N and I2C1_INT_N
RST_N	Input	No	Host Interface: Asynchronous Reset This active low input is an asynchronous reset for the application processor to reset the bridge. The bridge must be reset as part of the system initialization sequence, and may be optionally reset at other times during system operation.
CLK	Input	No	Host Interface: Clock Input This clock input is used to operate the I2C side of the bridge. Prefer 19.2 MHz or frequency that is easier to be divided down to 400 kHz for I2C clock speed requirement. Need to be on a Clock Pin

Functional Overview

SPI Interface

SPI stands for Serial Peripheral Interface. This is a full duplex synchronous bus that allows a SPI host to transmit or receive from several slaves serially via the common MISO and MOSI signals of the bus. In this application example, the SPI interface (intf) only works as a slave device. This SPI Slave Interface requires a SPI host located on the Applications Processor (AP) to arbitrate the bus. The Application processor performs interrupt (INT) servicing if the SPI Slave requires attention.

Registers

Each register is 8 bit in length. These register functions are described in the “Register Description” section in more details and can be useful for firmware engineers when implementing device driver.

I2C Master #0 and #1

Each I2C Master Controller has an 8-byte deep transmit and receive FIFOs for efficient data transfers. The dual I2C masters operate independently. For example, one master may be running Standard mode (100 kHz) while the other operates in Fast mode (400 kHz).

Register Description

Internal Registers	I2C Master #0	I2C Master #1
FIFO Status Register	0x1[1]	0x1[1]
Revision ID Register	0x3[1]	0x3[1]
Configuration Register	0x4	0xA
Mode Register	0x5	0xB
Command Status Register	0x6	0xC
1. Registers are shared		

Note that the unused addresses are reserved for future use

SPI Command/Internal Register (SPI Host Write)

Table 2. SPI Command/Internal Register (SPI Host Write)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Write	CMD[3]=0 CMD[3]=0	CMD[2]=0 CMD[2]=0	CMD[1]=0 CMD[1]=0	CMD[0]=0 CMD[0]=1	RADR[3]	RADR[2]	RADR[1]	RADR[0]
	CMD[3]=0 CMD[3]=0 CMD[3]=0 CMD[3]=0	CMD[2]=0 CMD[2]=0 CMD[2]=1 CMD[2]=1	CMD[1]=1 CMD[1]=1 CMD[1]=0 CMD[1]=0	CMD[0]=0 CMD[0]=1 CMD[0]=0 CMD[0]=1	I2CM[2]	I2CM[1]	I2CM[0]	Reserved
	CMD[3]=1 CMD[3]=1	CMD[2]=0 CMD[2]=0	CMD[1]=0 CMD[1]=0	CMD[0]=0 CMD[0]=1	1 1	1 1	0 1	1 1
Reset	0	0	0	0	0	0	0	0

CMD[3:0] – SPI Command

0000 – Write to a Specific Internal Register [WR_REG]

0001 – Read from a Specific Internal Register [RD_REG]

0010 – Interrupt Check [INT_CHK]

0011 – Write N number of bytes to a I2C Slave where N = 8 is at maximum [WR_I2C]

0100 – Read N number of bytes from a I2C Slave where N = 8 is at maximum [RD_I2C]

0101 – Read Receive FIFO [RD_FIFO]

Others – Reserved

RADR[3:0] – 4-bit Internal Register Address

I2CM[2:0] – I2C Master Number (Identify which I2C master to write to)

0000 – I2C Master #0

0010 – I2C Master #1

Others – Reserved

FIFO Status Register (Read Only)

Offset: 0x1 – I2C Master #0 and I2C Master #1 share this register

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Read	RX0FULL	RXOEMPTY	TXOFULL	TXOEMPTY	RX1FULL	RX1EMPTY	TX1FULL	TX1EMPTY
Reset	0	1	0	1	0	1	0	1

RX0FULL – I2C Master #0 Receive FIFO Full

0 – Receive FIFO is Not Full yet

1 – Receive FIFO is Full

RX0EMPTY – I2C Master #0 Receive FIFO Empty

0 – Receive FIFO is Not Empty yet

1 – Receive FIFO is Empty

TX0FULL – I2C Master #0 Transmit FIFO Full

0 – Transmit FIFO is Not Full yet

1 – Transmit FIFO is Full

TX0EMPTY – I2C Master #0 Transmit FIFO Empty

0 – Transmit FIFO is Not Empty yet

1 – Transmit FIFO is Empty

RX1FULL – I2C Master #1 Receive FIFO Full

0 – Receive FIFO is Not Full yet

1 – Receive FIFO is Full

RX1EMPTY – I2C Master #1 Receive FIFO Empty

0 – Receive FIFO is Not Empty yet

1 – Receive FIFO is Empty

TX1FULL – I2C Master #1 Transmit FIFO Full

0 – Transmit FIFO is Not Full yet

1 – Transmit FIFO is Full

TX1EMPTY – I2C Master #1 Transmit FIFO Empty

0 – Transmit FIFO is Not Empty yet

1 – Transmit FIFO is Empty

Revision ID Register (Read only at initialization)

Offset: 0x3

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Read	ID[7]	ID[6]	ID[5]	ID[4]	ID[3]	ID[2]	ID[1]	ID[0]
Reset	ID[7]	ID[6]	ID[5]	ID[4]	ID[3]	ID[2]	ID[1]	ID[0]

Upon reset, this register is loaded with the revision ID of this design. Software can read this register after power-up to determine the hardware revision.

Configuration Register (Read/Write)

Offset: 0x4 – I2C Master #0 and 0xA – I2C Master #1

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Read; Write:	0 RESET	RXFIFO_CLR	TXFIFO_CLR	ABORT	Reserved	Reserved	INT_CLR	START
Reset:	0	0	0	0	0	0	0	0

RESET – Writing a “one” will reset this I2C Master Controller. This bit will always read as a “zero”. Note that asserting RESET bit will not affect the other I2C Master Controllers.

RXFIFO_CLR – Writing a “one” will clear the receive FIFO. The user needs to write a “zero” afterward.

TXFIFO_CLR – Writing a “one” will clear the transmit FIFO. The user needs to write a “zero” afterward.

ABORT – Writing a “one” will stop the current I2C transaction in progress. This bit is cleared by the

ABORT_ACK status bit in the Command Status Register.

INT_CLR – Writing a “one” will clear all bits in the Command Status Register, except the I2C_BUSY bit. The user needs to write a “zero” to clear this bit.

START – Write a “one” to start an I2C transaction. This bit is auto-cleared after the master has successfully acquired the I2C bus.

Mode Register (Read/Write)

Offset: 0x5 – I2C Master #0 and 0xB – I2C Master #1

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Read: Write:	[1] BPS[1]	[1] BPS[0]	[1] TX_IE	[1] ACK_POL	[1] RX_IE	[1] Reserved	[1] Reserved	[1] Reserved
Reset:	0	0	0	0	0	0	0	0

1. Unimplemented

BPS[1:0] – Selects the I2C speed mode (2'b00 = Standard, 2'b01 = Fast, other are reserved)

TX_IE – Set this bit high to enable interrupt generation on transmit completion and STOP issued, or on any error.

ACK_POL – Set the behavior of ACK during the last byte of a master read transaction. This bit should be "0" for ACK and "1" for NACK. If repeat START (read followed by read/write) is not used, this bit should be set to "1" (NACK) for I2C compliance.

RX_IE – Set this bit high to enable interrupt generation on receive completion and STOP issued, or any error.

Command Status Register (Read – only)

Offset: 0x6 – I2C Master #0 and 0xC – I2C Master #1

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Read:	I2C_BUSY	NO_ANS	NO_ACK	TX_ERR	RX_ERR	ABORT_ACK	TS	Reserved
Reset:	0	0	0	0	0	0	0	0

I2C_BUSY – This read only status bit indicates that the bridge is busy performing a data transaction and a STOP has not been issued. This bit reflects the state of the I2C bus and cannot be cleared by the user.

N_ANS – This read only status bit goes high when I2C Slave does not response to address+R/W

N_ACK – This read only status bit goes high when I2C Slave does not acknowledge

TX_ERR – This read only status bit indicates that an error has occurred during the I2C write operation.

RX_ERR – This read only status bit indicates that an error has occurred during the I2C read operation.

ABORT_ACK – This read only status bit indicates that the ABORT command has been completed. The user should clear the proper FIFO and status bits afterwards.

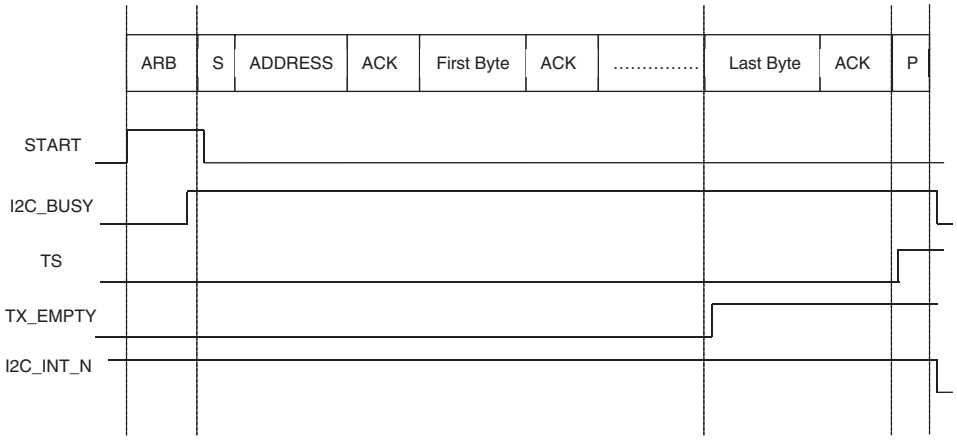
TS – This read only status bit changes to one to indicate that the TX FIFO has completed transmitting its data or the RX FIFO has completed receiving its data.

Note that all status bits, except I2C_BUSY, are cleared by writing a "one" to the INTR_CLR bit in the Configuration Register.

Bridge Transmit Operations

The following diagram illustrates the basic transmit operation of the SPI to I2C bridge. Note that “S” is notation for START and “P” is notation for “STOP”.

Figure 2. Transmit Operation of an active I2C transaction



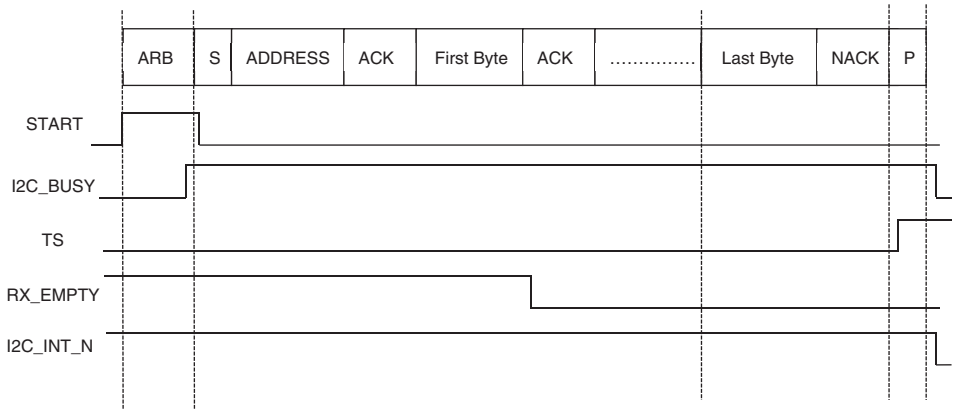
1. Note that the START bit will clear itself after the I2C master has acquired the I2C and issued a START command. The I2C_BUSY bit will go high before the START bit is cleared to indicate an active I2C transaction. The I2C_BUSY bit will clear itself after the entire I2C transaction is completed and the bus is idled.
2. The TX_EMPTY bit only monitors the TX_FIFO status. It is asserted after the last byte from the FIFO has been passed to the I2C master controller, but the data has not yet been send by the controller.
3. The TS signal is asserted after the last byte of transmit data have been sent, but STOP condition may not have completed yet.
4. I2C_BUSY is cleared after STOP condition has completed and bus is idled.
5. The interrupt is asserted if the transmit interrupt enable bit (TX_IE) is set.If the slave device did not assert ACK during the address phase, then NO_ANS, TX_ERR and INT_N (if enabled) will all be asserted. If the slave device did not assert ACK during the data phase, then NO_ACK,

TX_ERR, and INT_N (if enabled) will all be asserted. The TX_FIFO will need to be flushed before the next transaction.

Bridge Receive Operations

The following diagram illustrates the basic receive operation of the SPI to I2C bridge:

Figure 3. Receive operation of an active I2C transmission



By writing a “one” to the START bit of the Configuration Register, the operation is started:

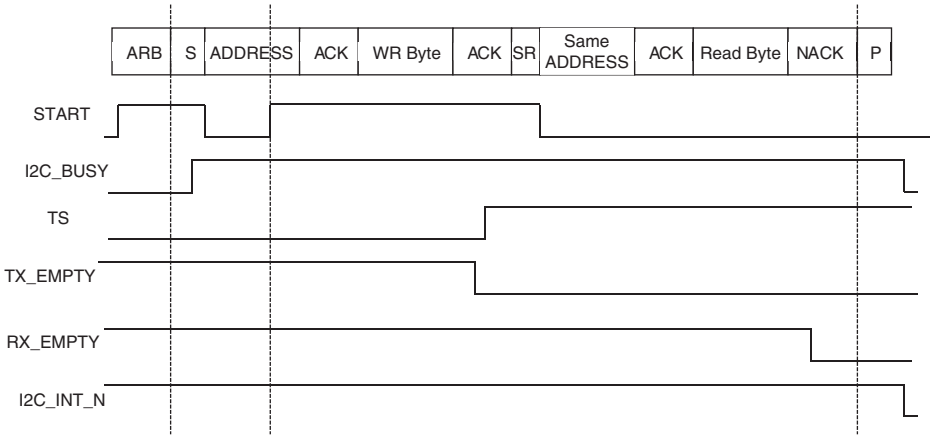
1. Note that the START bit will clear itself after the I2C master has acquired the bus and issued START command. The I2C_BUSY bit will go high before the START bit is cleared to indicate an active I2C transaction. The I2C_BUSY bit will clear itself after the entire I2C transaction is completed and the bus is idled.
2. The RX_EMPTY bit will go low after successfully receiving the first byte.
3. The TS signal is asserted after the last byte of receive data have been latched, but STOP condition may not have completed yet.
4. I2C_BUSY is cleared after STOP condition has completed and bus is idled.
5. The interrupt is asserted if the receiver interrupt enable bit (RX_IE) is set. If the slave device did not assert ACK during the address phase, then NO_ANS, RX_ERR and I2C_ INT_N (if enabled) will all be asserted. The RX_FIFO should be flushed before the next transaction.

Repeated Start Transactions

To overcome the limited addresses provide by the I2C bus, some I2C slave uses “repeated Start” transactions to allow indirect access of internal registers. The I2C master controller supports the generation of “Repeat Start” transactions. Figure 4 below illustrates a typical write-write I2C transaction:

Note that “SR” is the I2C notation for a repeated Start.

Figure 4. Write followed by a repeated start and read I2C Transaction



By writing a “one” to the START bit of the Configuration Register, the operation is started:

1. The first parts of sequence are identical to the set up for a normal read or write transaction shown earlier.
2. To perform a repeated Start transaction, a second START command needs to be issued after the START bit has cleared. This is done by polling the START bit until it has cleared. Next, the user needs to issue the second I2C command (read or write). Finally, the Configuration Register is written with “1” for the START bit. Note that the second START command needs to be issued before the TS is asserted from the first transaction.
3. Do not change the slave address for the repeated Start transaction; otherwise, a STOP is generated and another full transaction is executed.
4. I2C_BUSY is cleared after STOP condition has completed and bus is idled.
5. Note that chained repeated Start transactions are possible by continuing monitoring the START bit, modifying the Mode register, and then issuing another START before STOP is issued.

   Note that the interrupt is asserted at the end of the transaction after STOP.

Clocking Requirements

To generate accurate I2C timing, an external reference clock is required. The proper SCL timing is based on the CLK input. CLK frequency should be easily divided down to 400 kHz. Example: 19.2 MHz.

Reset and Initialization

This example assumes that CLK is 19.2 MHz and the desired SCL is approximately 400 kHz for Fast Mode. The desired master last byte acknowledge is NACK and interrupts are to be use. Appropriate configuration settings are:

- BPS[1:0] = “01” (to set Fast Mode – 400 kHz)
- ACK_POL =”1” (to set last byte acknowledge as NACK)
- TX_IE=”1” (to set transmit completion interrupt)
- RX_IE = “1” (to set receive completion interrupt)

Example Transactions

The SPI sequence for the initializing the I2C controller is illustrated in Figure 5 and Figure 6.

Figure 5. SPI Host to set I2C master#0 Configuration Register for Initialization

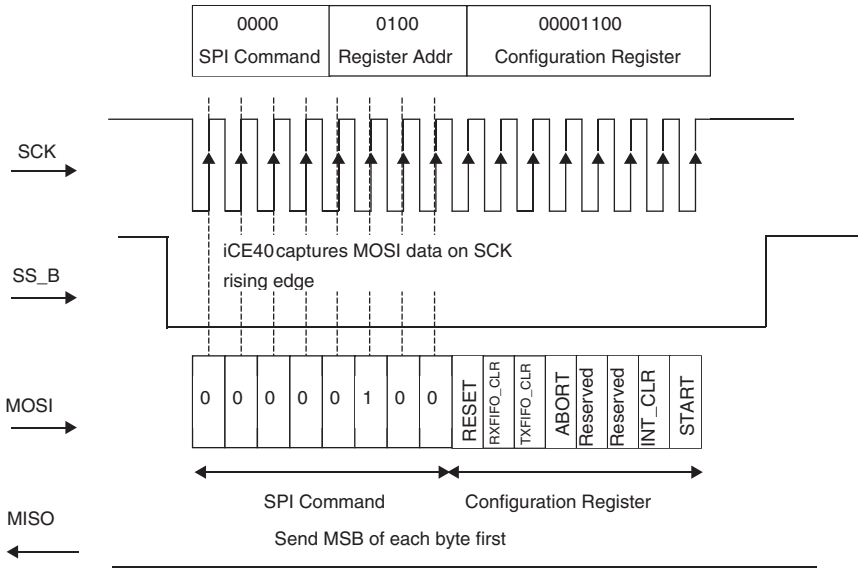
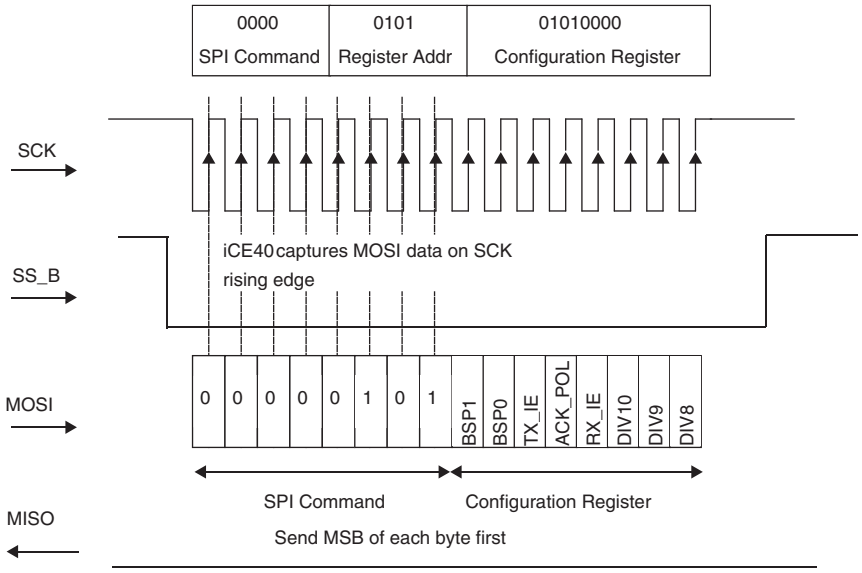


Figure 6. SPI Host to set I2C master#0 Mode Register for Initialization



The SPI sequence for the issuing a write command to the I2C controller is illustrated in Figure 7 and Figure 8.

Figure 7. SPI Host to write 2 bytes of data into I2C Master#0 TX FIFO

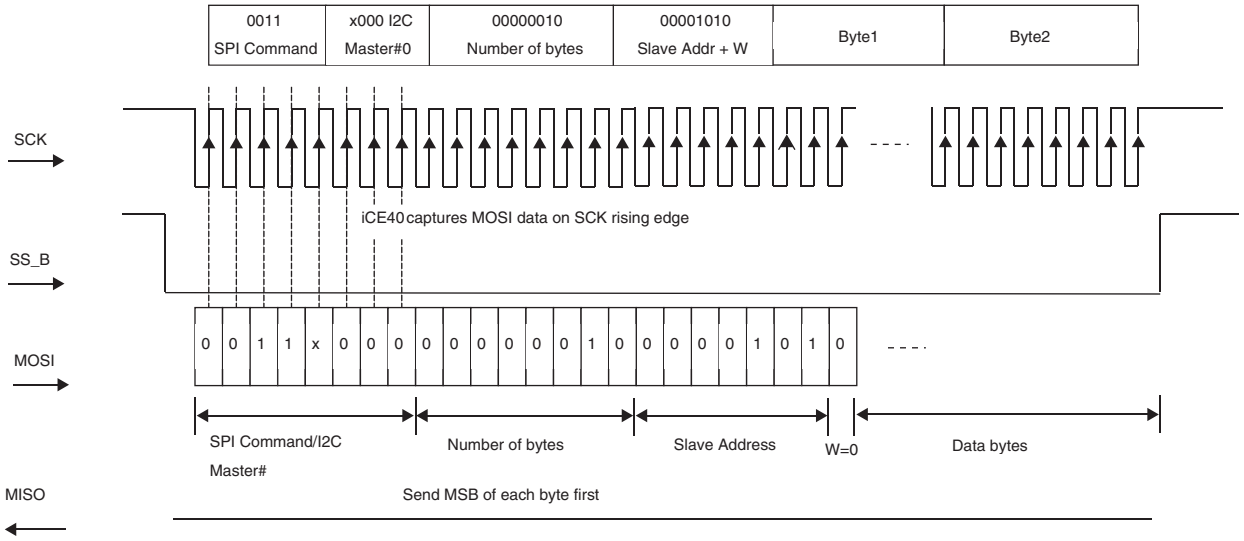
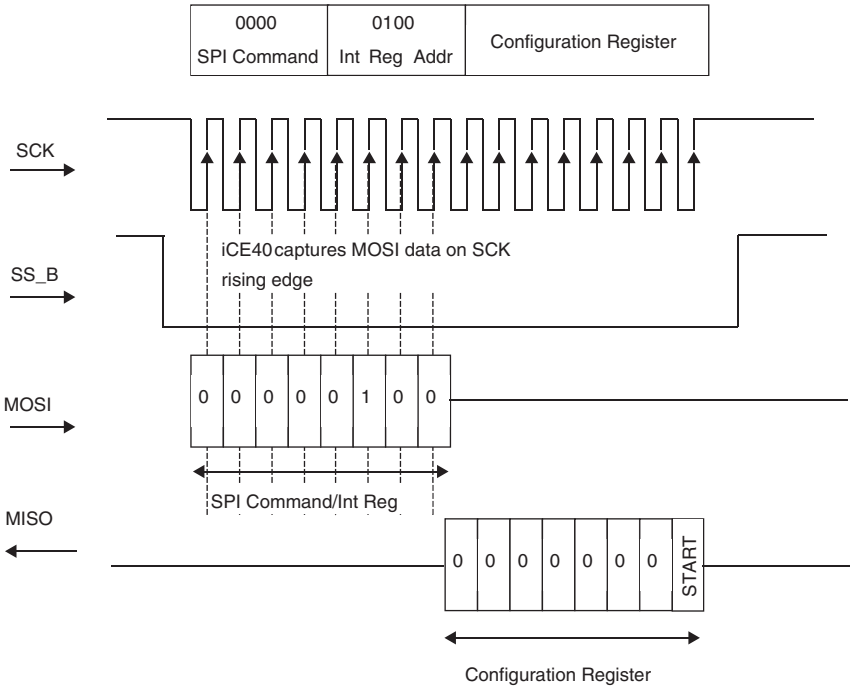


Figure 8. SPI Host to check or set START Bit in the Configuration Register



The SPI sequence for the checking the “i2c_busy” bit and clearing interrupt flags are illustrated in Figure 9 and Figure 10.

Figure 9. SPI Host to check I2C Master #0 I2C_BUSY bit from Command Status Register

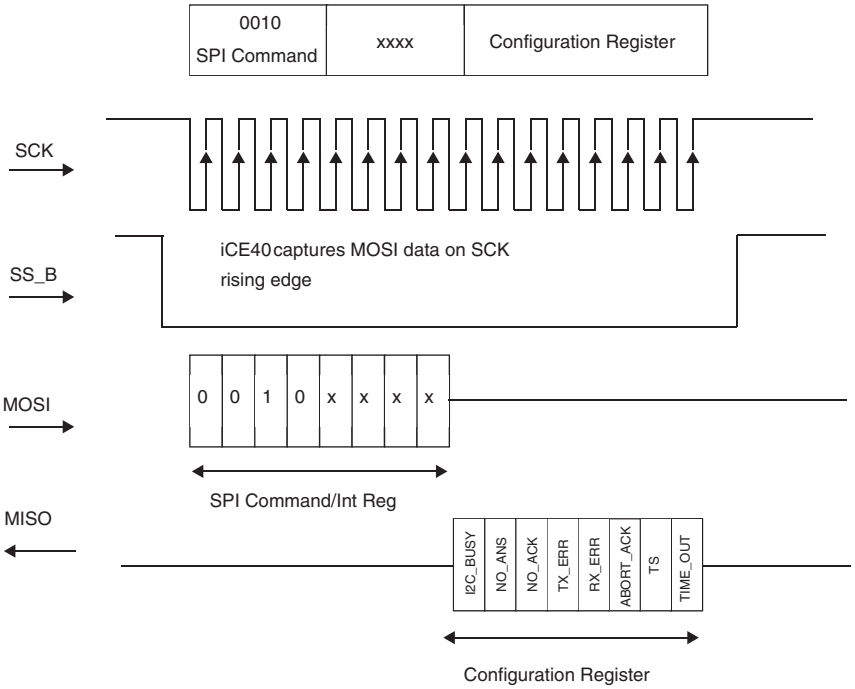
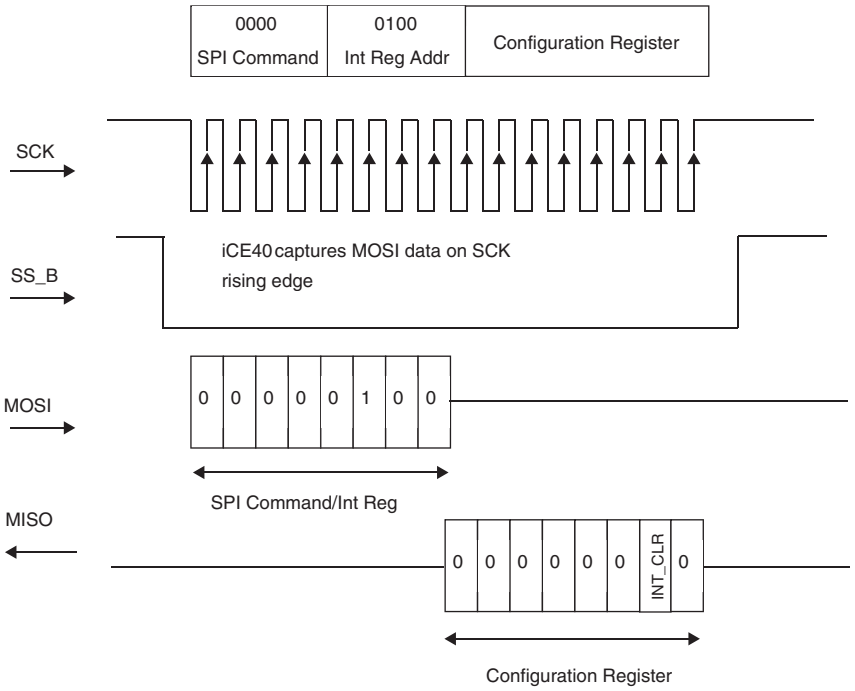
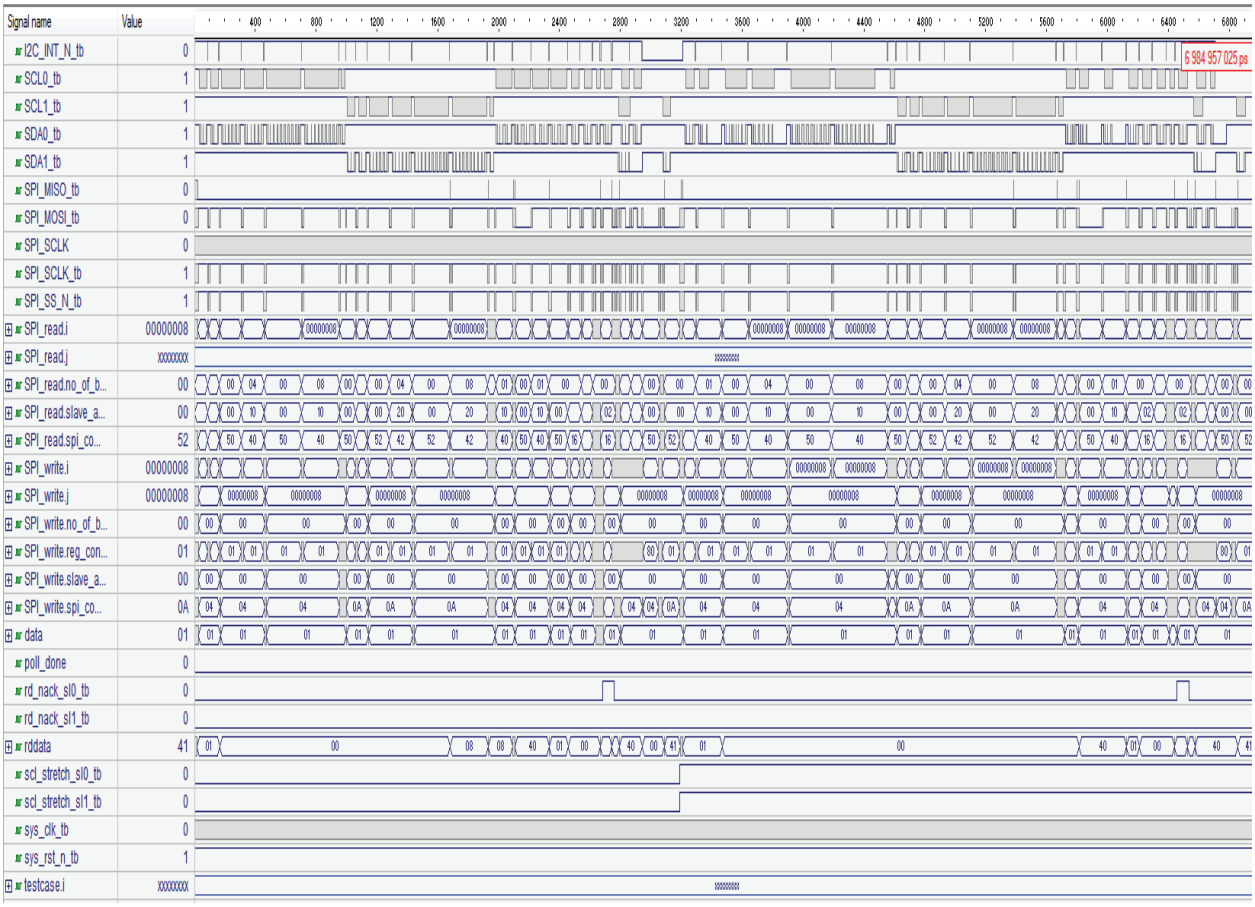


Figure 10. SPI Host to set INT_CLR to clear interrupt flags in Command Status Register



Simulation Waveforms

Figure 11. Simulation Waveforms



Implementation

This design is implemented in mixed VHDL and Verilog language. When using this design in a different device, density, speed or grade, performance and utilization may vary.

Note: In the RTL, data_ms and data_bus signals have been taken as two separate input and output buses instead of inout bus.

Table 3. Performance and Resource Utilization

Device Family	Language	Synthesis Tool	Utilization (LUTs)	fMAX (MHz)	I/Os	Architecture Resources
iCE40[1]	Verilog	LSE	966	>50	11	(151/160) PLBs
		Syn Pro	960	>50	11	(153/160) PLBs

1. Performance and utilization characteristics are generated using iCE40LP1K-CM121 with iCEcube2 2014.12 design software.

References

- DS1040, iCE40 LP/HX Family Data Sheet

Technical Support Assistance

e-mail: techsupport@latticesemi.com

Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
February 2015	1.1	Updated Implementation section. Updated Table 3, Performance and Resource Utilization. — Added LSE support. — Implementation is carried out for iCEcube2014.12.
		Updated References section.
		Updated Technical Support Assistance information.
April 2013	01.0	Initial release.