

iCE65 FPGA as an I2C Master Controller

Overview

I2C, or Inter-Integrated Circuit is a popular serial interface protocol that is widely used in many electronic systems. The I2C interface is a 2 wire interface capable of half duplex serial communication at moderate to high speeds of up to a few mega bits per second. This application example illustrates the implementation of an I2C Master Controller using SiliconBlue ultra low power iCE65L04 FPGAs.

Features

- 7 bit slave address support
- Supports operation at 100KHz (Standard Mode) and 400KHz (Fast Mode)
- Supports repeated start operations
- Interrupt generation logic
- VHDL RTL, testbench and ModelSim script for simulation
- IP-XACT version 1.2 compliant

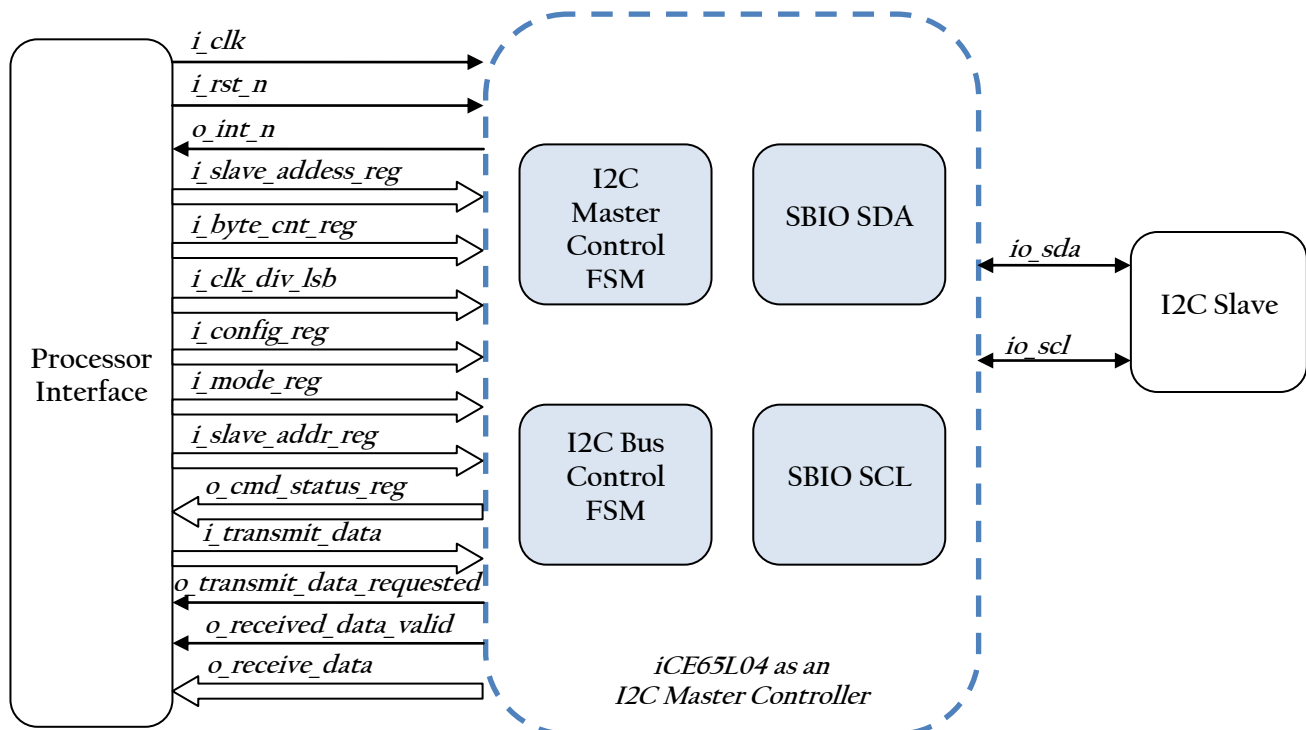
Resource Utilization

Table 1: Resource Utilization

LUTs	Registers	Memory	GBs	I/Os
304	207	0	0	0

Note: Resource Utilization is based on iCECube 2010.12.14671

System Block Diagram



Design Interface

Signal Name	Pin Type	Signal Description
i_clk	Input	System clock operating at 32MHz.
i_rst_n	Input	Asynchronous Active high system reset.
o_int_n	Output	Active low Processor Interrupt.
i_slave_addr_reg[7:0]	Input	7 bit I2C Slave address.
i_byte_cnt_reg[7:0]	Input	Set the number of data bytes to read or written for the I2C transaction.
i_clk_div_lsb[7:0]	Input	Sets the lower byte of the clock divider that is used to generate SCL from CLK. The upper three bits are located in the Mode Register.
i_config_reg[5:0]	Input	Used to configure the I2C Master Controller (See table below).
i_mode_reg[7:0]	Input	Sets the various modes of operation like speed, Read/write (See table below).
o_cmd_status_reg[7:0]	Output	Lets the user know the status of operation, I2C bus (See table below).
o_start_ack	Output	Acknowledge to the start bit provided by the user through i_config_reg
i_transmit_data[7:0]	Input	Data to be transmitted over the the SDA line to the I2C Slave
o_transmit_data_requested	Output	Lets the user know that Transmit data is required.
o_received_data_valid	Output	A '1' on this corresponds to Valid data availability on o_receive_data line
o_receive_data[7:0]	Output	Received data bus.
io_scl	Inout	I2C Clock line.
io_sda	Inout	I2C Data line.

Port/Bit	7	6	5	4	3	2	1	0
i_slave_addr_reg		SADR[6]	SADR[5]	SADR[4]	SADR[3]	SADR[2]	SADR[1]	SADR[0]
i_byte_cnt_reg	BCNT[7]	BCNT[6]	BCNT[5]	BCNT[4]	BCNT[3]	BCNT[2]	BCNT[1]	BCNT[0]
i_clk_div_lsb	DIV[7]	DIV[6]	DIV[5]	DIV[4]	DIV[3]	DIV[2]	DIV[1]	DIV[0]
i_config_reg			RESET	ABORT	TX_IE	RX_IE	INT_CLR	START
i_mode_reg	BPS[1]	BPS[0]		ACK_POL	RW_MODE	DIV[10]	DIV[9]	DIV[8]
o_cmd_status_reg	I2C_BUSY	TX_DONE	RX_DONE	TX_ERR	RX_ERR	ABORT_ACK		

Unimplemented
 Undefined

SADR[7:0] – 7 Bit slave address.

BCNT[7:0] – Set the number of data bytes to be written or read for the I2C transaction. For example, set the register to “8” to transfer eight data bytes.

DIV[7:0] – Sets the lower byte of the clock divider that is used to generate SCL from CLK. The upper three bits are located in the Mode Register. Please note that DIV[0] is currently not used since only even DIV values are supported.

RESET – Writing a “1” will reset this I2C Master Controller.

ABORT – Writing a “1” will stop the current I2C transaction in progress. This bit is cleared by the ABORT_ACK status bit in the Command Status Register.

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

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

INT_CLR – Writing a “1” 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 “1” to start an I2C transaction. This bit is auto-cleared after the master has successfully arbitrated and acquired the I2C bus.

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

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 is not used, this bit should be set to “1” (NACK) for I2C compliance.

RW_MODE – Set the read or write operation on the I2C bus (“0” = write, “1” = read)

DIV[10:8] – The upper three bits of the clock divide factor.

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.

TX_DONE – This read only status bit indicates that the I2C write operation issued has been completed, but the STOP condition may still be in progress.

RX_DONE – This read only status bit indicates that the I2C read operation issued has been completed, but the STOP condition may still be in progress.

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.

Note that all status bits, except I2C_BUSY, are cleared by writing a “1” to the INTR_CLR bit in the Configuration Register.

Configurable Parameters

None

Register Map

This design does not have any user accessible registers or memory.

Design Details

I2C Bus Control FSM : The I2C Bus control FSM comprises of Clock generator/Synchronizer, Start/Stop Generator/Detect Logic and Acknowledge generate/detect logic.

The Clock Generation and Synchronization logic generates I2C clock signal SCL, based on the System clock and Clock Divide factor configured by the processor. Due to the nature of the I2C bus, the actual SCL clock that is seen by all devices on the bus may not be running at the same frequency that the master generates. This module starts counting its SCL low period when the current master drives SCL low. Once a device's clock has gone low, the Master holds the SCL line low until the clock high state is reached. When all devices concerned have counted off their LOW period, the clock line will be released and goes HIGH. There will then be no difference between the device clocks and the state of the SCL line, and all the devices will start counting their HIGH periods. The first device to complete its HIGH period will again pull the SCL line LOW. In this way, a synchronized SCL clock is generated with its LOW period determined by the device with the longest clock LOW period, and its HIGH period determined by the one with the shortest clock HIGH period.

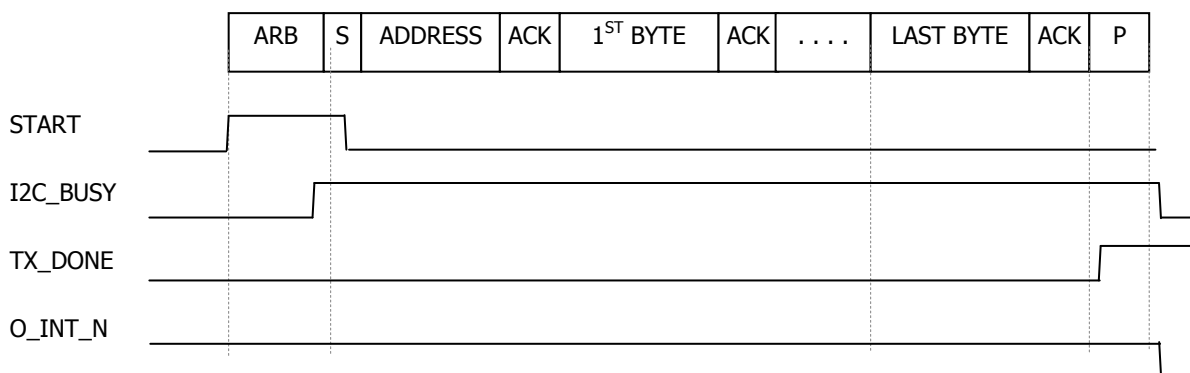
The Start/Stop logic generates and detects start and stop events on the I2C bus. The detection of start and stop events is necessary to determine whether or not the I2C bus is in use by another master on the bus when the primary master gets a START signal from the processor. When the I2C bus is idle, both SCL and SDA are pulled high by passive pullups. A start condition is signaled by transitioning SDA from high to low while SCL is still high. Likewise, a stop condition is signaled by transitioning SDA from low to high while SCL is high.

I2C Master Control FSM : For controlling data transfer, I2C Master makes use a control FSM, along with counters for controlling the bits and bytes. The Byte counter is an 8-bit counter that keeps track of the number of bytes that have been written or read during the I2C transaction. This counter increments after each byte has been written to or read from the external I2C slave device. The count is then compared with the Byte Count Register. If the value is a match, the I2C Master Controller considers the transaction complete, issues a stop signal on the I2C bus, asserts the “RXTX_DONE” flag and waits for the next transaction to be initiated from the processor. This counter is fully controlled by the main control FSM.

I2C Master Transmit operation

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

By writing a “1” to the START bit of the Configuration Register, the operation is started:

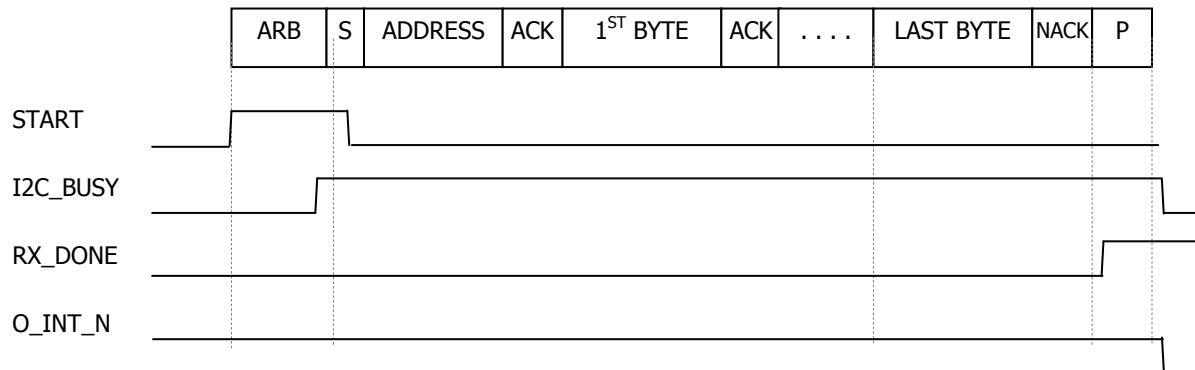


- 1) The I2C_BUSY bit will go high before the START bit is cleared to indicate an active I2C transaction. The I2C_BUSY bit will clear itself after the entire I2C transaction is completed and the bus is idled.
- 2) The TX_DONE signal is asserted after the last byte of transmit data have been sent, but STOP condition may not have completed yet.
- 3) I2C_BUSY is cleared after STOP condition has completed and bus is idled.
- 4) The interrupt is asserted if the transmit interrupt enable bit (TX_IE) is set.

If the slave device did not assert ACK, then TX_ERR and INT_N will both be asserted.

I2C Master Receive operation

The following diagram illustrates the basic receive operation of the I2C Master :



By writing a “1” to the START bit of the Configuration Register, the operation is started:

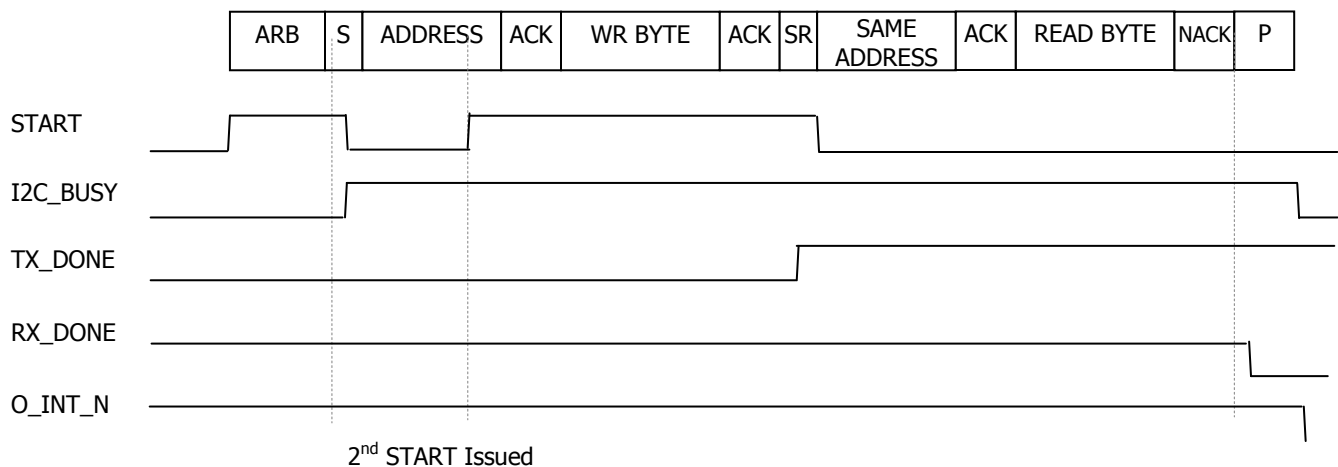
- 1) The I2C_BUSY bit will go high before the START bit is cleared to indicate an active I2C transaction. The I2C_BUSY bit will clear itself after the entire I2C transaction is completed and the bus is idled.
- 2) The RX_DONE signal is asserted after the last byte of receive data have been latched, but STOP condition may not have completed yet.
- 3) I2C_BUSY is cleared after STOP condition has completed and bus is idled.
- 4) 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 RX_ERR and INT_N will both be asserted.

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 below illustrates a typical Repeat Start I2C transaction:

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

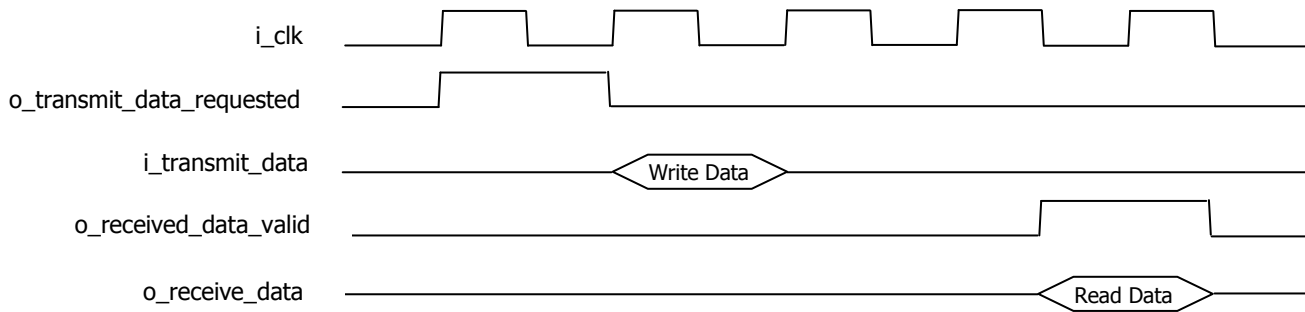


By writing a “1” to the START bit of the Configuration Register, the operation is started :

- 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, the user needs to update the Byte Count Register, Mode Register, finally, the Configure Register is written with “1” for the START bit. Note that the second START command needs to be issued before the TX_DONE /RX_DONE phase of 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.

Read/ Write data transfer to I2C Master Controller module

The figure below shows timing diagram for read and write data transfer to the I2C Master Controller



System Designer Flow

I2C Master Controller is compatible with System Designer/IP-XACT 1.2. The System Designer flow is as follows,

- 1) Launch the System Designer from Synplify Pro using menu 'Import -> Launch System Designer'.
- 2) Create a new project(open an existing old project, as necessary) and import the IP-XACT XML file
- 3) Drag and place the component from the 'Library' pane to the 'Design' pane
- 4) Click on the “Generate Files” button, which generates the necessary files required for synthesis and simulation.
- 5) Go to Synplify Pro, click on the “Run” button to synthesize the System Designer generated files. Synplify Pro generates all the necessary files for P&R in iCECube2.

References

The following references were used in the creation of this design:

- NXP I2C Specification
- SiliconBlue Technologies, Inc. "[iCE65 Ultra Low-Power mobileFPGA Family](#)" datasheet (13-JUL-2009).

Revision History

Version	Date	Description
1.1	07-DEC-2010	IP-XACT format Update and Initial Draft Document

Disclaimer

Copyright © 2007–2009 by SiliconBlue Technologies LTD. All rights reserved. SiliconBlue is a registered trademark of SiliconBlue Technologies LTD in the United States. Specific device designations, and all other words and logos that are identified as trademarks are, unless noted otherwise, the trademarks of SiliconBlue Technologies LTD. All other product or service names are the property of their respective holders. SiliconBlue products are protected under numerous United States and foreign patents and pending applications, maskwork rights, and copyrights. SiliconBlue warrants performance of its semiconductor products to current specifications in accordance with SiliconBlue's standard warranty, but reserves the right to make changes to any products and services at any time without notice. SiliconBlue assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by SiliconBlue Technologies LTD. SiliconBlue customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



SiliconBlue Technologies Corporation

3255 Scott Blvd.,
Building 7, Suite 101
Santa Clara, CA 95054

Tel: 408-727-6101
Fax: 408-727-6085

www.SiliconBlueTech.com