I^2C Master Controller

January 2015 Reference Design RD1005

Introduction

There are thousands of I^2C peripherals on the market today, ranging from data converters to video processors. The I^2C bus is a good choice for designs that need to communicate with low speed peripherals due to its simplicity and low cost.

This reference design is intended to demonstrate how a fast, highly-flexible I^2C Master Controller can be constructed and utilized in a Lattice CPLD/FPGA device. This design is intended to be a general-purpose design offering a viable solution to controlling an I^2C bus. It also provides a foundation from which designers can customize the I^2C Master Controller to meet their design requirements. Customizing the Hardware Description Language (HDL) code allows designers to meet their specific requirements, thus reducing valuable CPLD area while maintaining speed performance.

Design Goals and Limitations

The following goals were considered during development of this reference design:

- I^2C bus speeds of 100 kbits/sec and 400 kbits/sec
- I^2C 7-bit addressing
- Multiple I^2C masters on one I^2C bus
- Up to 256-byte I^2C transactions
- Interrupt mode or polling mode
- Generic microprocessor interface that supports various MPU speeds
- Programmable configuration registers
- Hierarchical HDL design for simple user modification
- Fully automated, self-checking HDL test bench for ease of verification.

The I^2C Bus Master Controller does not support the following features:

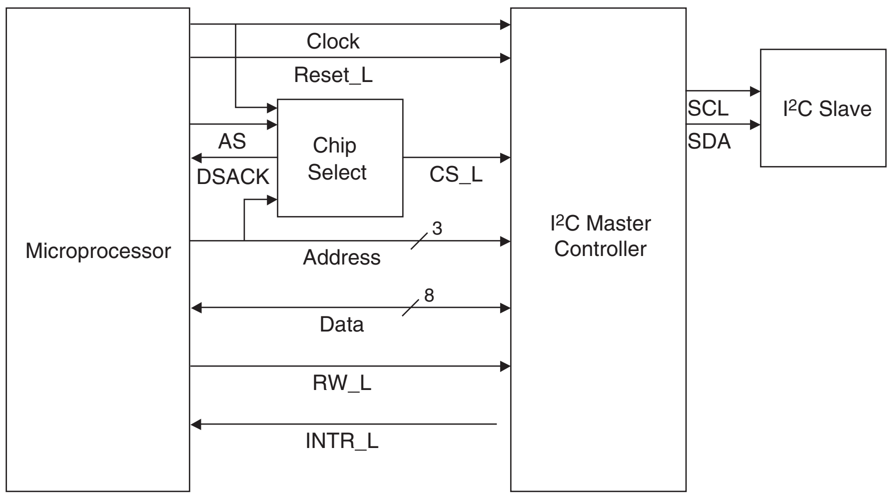
- High speed mode 3.4 Mbps
- Mixed speed modes on one bus
- 10-bit addressing
- Master Controller cannot be used as an I^2C slave device

Theory of Operation

Overview

The I^2C Master Controller is designed to interface with up to 127 different I^2C slave devices. In order to accomplish this task, the I^2C Master Controller requires several components to make a complete I^2C bus interface system. The components required are a microprocessor, chip select unit and I^2C slave devices (Figure 1). The microprocessor initiates and configures all I^2C bus transactions. Next, the chip select unit assures that the microprocessor's bus cycles adhere to the I^2C Master Controller requirements. Finally, the I^2C slave device can be any I^2C slave device operating within the specification created by Philips Semiconductor.

Figure 1. I^2C Master Controller I/O Interface



Functional Description

The I^2C Master Controller accepts commands from a microprocessor. These commands are decoded into I^2C slave device read/write cycle transactions. The I^2C bus transactions can be configured to be 1 to 256 bytes in length. Furthermore, the I^2C Master Controller can operate in interrupt or polling mode. This means that the microprocessor programmer can choose to poll the I^2C Master for a change in status at periodic intervals or wait to be interrupted by the I^2C Master Controller when data needs to be read or written.

Table 1. I^2C Master Controller Signal Descriptions

Signal Name	Signal Direction	Active State	Name – Definition
Microprocessor Interface			
Clock	Input	N/A	Microprocessor clock
Reset_L	Input	Active Low	System reset
CS_L	Input	Active Low	Chip Select – Microprocessor must keep data valid on the bus as long as chip select is asserted. Data must be valid for at least three clocks.
Address	Input	N/A	Address bits reading and writing to configuration and data registers.
Data	Bi-Dir	N/A	Data bus
RW_L	Input	1 = RD 0 = WR	Indicates whether the microprocessor is writing to or reading from I^2C Bus Master Controller registers.
INTR_L	Output	Active Low	Interrupt request signal
I^2C Interface			
SDA	Bi-Dir	N/A	I^2C data bus line
SCL	Bi-Dir	N/A	I^2C clock line

Microprocessor Interface Design Requirements

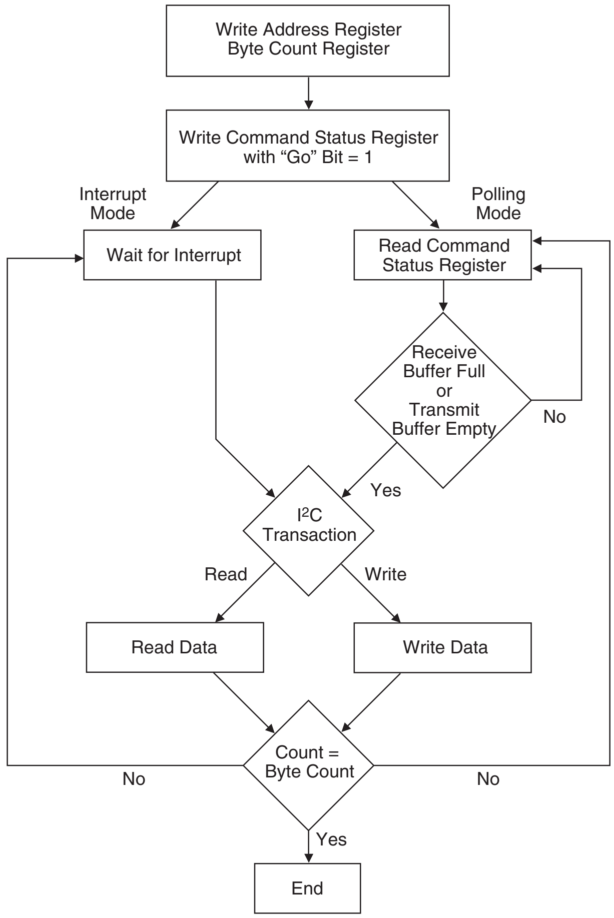
The following list contains requirements that the microprocessor must follow to ensure proper operation of the system:

- Chip Select must be synchronized to the microprocessor clock frequency.
- Address and data must be valid the entire time chip select is asserted during a write cycle.
- Data is latched into the appropriate I^2C Master Controller registers on the rising edge of the third microprocessor clock.
- Data strobe acknowledge is controlled externally to the I^2C Master Controller. It is up to the designer to insert the appropriate wait states to achieve the requirements above.

Interfacing to the I^2C Master Controller from a Microprocessor

There are several operations a user must undertake to interface to the I^2C Master Controller. The first involves loading the appropriate I^2C Master Controller configuration registers and issuing a "go" command. The configuration registers can be loaded in any order with one exception, the "go" bit should be asserted last. The registers that need to be configured are the Address Data, Byte Count and Command Status. The "go" bit may be asserted at the same time that the Command Status Register is written to. Next, the user may be required to write to the *Iack* register. If the user is operating in interrupt mode, writing to the *Iack* register after every interrupt will be necessary. Alternately, if the user is operating in polling mode, it will be necessary to read the Command Status Register. This is necessary in polling mode because this is how the microprocessor knows when to read or write to the I^2C Master Controller. Finally, the user may be required to read from or write to the data bus, depending on the type of I^2C transaction. A summary of these steps is shown in Figure 2.

Figure 2. Programming Flow for the I²C Master Controller



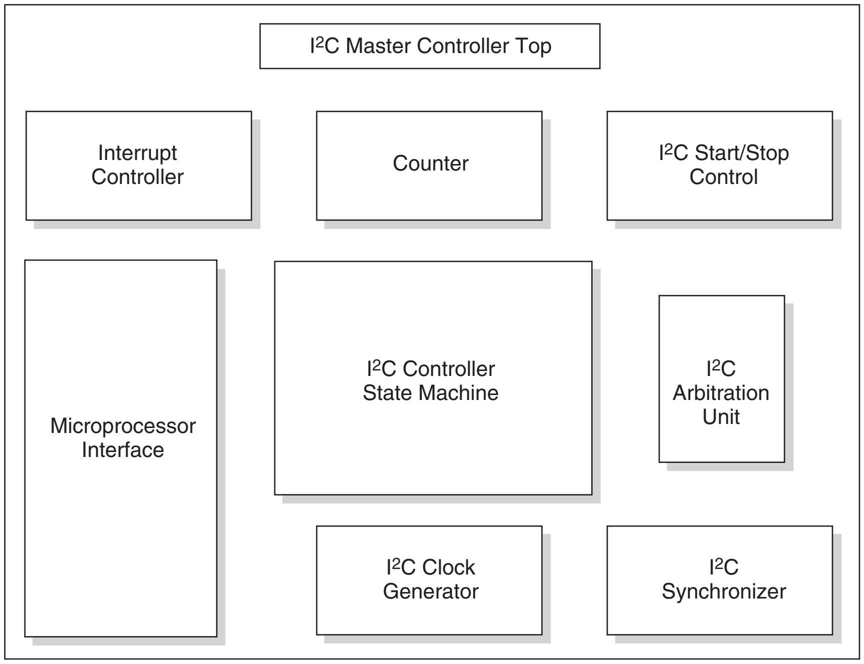
Register Transfer Level (RTL) Implementation

Functional Blocks

The HDL code for the I²C Master Controller reference design contains the following functional blocks:

- I²C master top level
- Microprocessor interface
- Interrupt controller
- Counter
- I²C start/stop control
- I²C clock generator
- I²C synchronizer
- I²C arbiter
- I²C controller state machine

Figure 3. The I^{2}C Master Controller Block Diagram



Register Definitions

The I^{2}C Master Controller uses the following registers:

Data Buffer

Bit	MPU Direction	Name	Description
7-0	Write/read	Data_Buffer	Stores I^{2}C read/write data

Address Data Register

Bit	MPU Direction	Name	Description
7-1	Write	Low_Address_Reg	Holds the first seven address bits of the I^{2}C slave device
0	Write	12C_RW_Bit	I^{2}C read/write bit

Command/Status Register (when written by MPU)

Bit	MPU Direction	Name	Description
7	Write	Go	Starts the I^{2}C transaction. This is asserted after the MPU has finished writing to all configuration registers.
6	Write	Abort	Stops an I^{2}C transaction currently in progress.
5	—	Reserved	Reserved
4	—	Reserved	Reserved
3	Write	I2C_Mode	0 = standard mode, 100 kbits/sec, 1 = fast mode, 400 kbits/sec
2	—	Reserved	—
1	Write	Trans_IE	Interrupt enable for transmission
0	Write	Receive_IE	Interrupt enable for receive

Command/Status Register (when read by MPU)

Bit	MPU Direction	Name	Description
7	Read	I2C_Bus_Busy	I²C bus busy
6	Read	Error	I²C transaction error
5	Read	Abort_Ack	Abort completed by I²C controller
4	Read	Lost_Arb	Lost arbitration bit
3	Read	Done	Transaction done
2	Read	Reserved	Reserved
1	Read	Trans_Buffer_Empty	Transmit buffer empty
0	Read	Receive_Buffer_Full	Receive buffer full

Byte Count

Bit	MPU Direction	Name	Description
7-0	Write	Byte_Count	Number of data bytes for the current transaction

Iack

Bit	MPU Direction	Name	Description
0	Write	Iack	Interrupt Acknowledge

I²C Master Controller Top

This is the top-level HDL block of the I²C Master Controller reference design. This block was created to instantiate all modules of the design and control the tri-state drivers of the SDA and SCL signals.

Microprocessor

This module is the main interface between the microprocessor and the I²C Master Controller. This block has three main functions.

- Accepts write data from the microprocessor and latches the data into the appropriate registers.
- Controls data being sent to the microprocessor from the I²C Master Controller.
- Resets the Interrupt Acknowledge, Transmit Buffer Empty and Read Buffer Full flags at the appropriate time.

Interrupt Controller

This block generates an interrupt signal when either Transmit Buffer Empty flag or Receive Buffer Full flag are set (assuming the appropriate configuration bits are set).

Counter

The Counter module contains two counters. The first is a 3-bit counter (Bit Counter) used for counting each bit of an eight-bit packet transmission or reception during an I²C bus packet transaction. The second counter is an 8-bit counter (Byte Counter) that keeps track of the number of bytes that have been written or read during the I²C transaction. The second counter increments after each byte has been written to or read from the 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 "done" flag and waits for the next transaction to be initiated from the microprocessor.

I²C Start/Stop Control

The Start/Stop Control block 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 go signal from the microprocessor. Furthermore, the start detection is necessary for the primary I²C Master Controller because it cannot proceed with a transaction until the start condition has been accepted by the I²C bus. As for the start and stop generation block, the I²C Master Controller uses this block to generate start and stop events at the appropriate times during the I²C transaction.

I²C Clock Generator

This block generates an I²C clock signal. The I²C Master Controller can be programmed to generate an I²C clock that will run in either standard mode (100 kbits/sec) or fast mode (400 kbits/sec). The mode that the clock runs in depends on the value of the mode bit in the command register. 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 requests or generates. This is because the I²C SCL clock line will run at the speed of the slowest clock being generated by a master. Please refer to the Philips I²C specification for further information regarding the I²C bus.

I²C Synchronizer

This block synchronizes the SCL and SDA signals with the system clock and protects against metastability. Normally, asynchronous signals can cause a synchronous system to become unstable by violating setup and hold times. This reference design avoids this by sending the SCL and SDA signals through two registers before using the signals.

I²C Arbiter

This reference design supports the multi master feature of the I²C bus specification. The Arbiter block contains the code necessary to perform arbitration on the I²C bus.

I²C Main State Machine

The Main State Machine block controls most of the other blocks and serves as the interface between the I²C Master Controller and the I²C bus. This block contains three state machines and many control signals. The three state machines are: main, read and write. The main state machine controls the other two and is where the I²C transactions begin. The read and write state machines shift I²C data into or out of the appropriate registers. All three state machines are one hot encoded and are therefore more efficient at driving the various output control signals.

Address Map

The microprocessor can control the I²C Master Controller by writing to and reading from configuration registers. The I²C Master Controller configuration registers are located at the following addresses:

Address 2-0	Description
000	Data buffer
001	Address data register
010	Reserved
011	Reserved
100	Command status register
101	Byte Count
110	Interrupt Acknowledge

HDL Verification

Overview

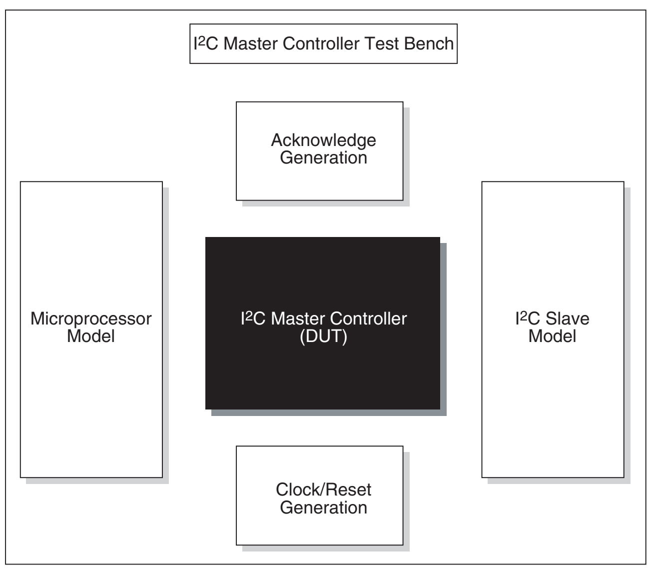
This test suite provides a way of verifying the functionality of the I²C Master Controller reference design at the Register Transfer Level (RTL) as well as the gate level. The test suite was designed to test one- and two-byte I²C read and writes utilizing both interrupt and polling modes. This is accomplished by calling the appropriate functions in the microprocessor model. The microprocessor model initiates all I²C transactions. The I²C Slave model emulates an I²C serial EPROM and responds to the requests of the I²C Master Controller.

Functional Blocks of the HDL Test Suite

The HDL code for the I²C Master Controller reference design test suite contains the following functional blocks:

- I²C Master Controller test bench
- Microprocessor model
- I²C slave model
- Clock/reset generation block
- Acknowledge generation block
- The device-under-test (I²C Master Controller)

Figure 4. The I²C Master Controller Test Suite



I²C Master Controller Top Level Test Bench

This is the module that instantiates all the other modules of the test suite. The HDL Test Bench is a fully automated and self-checking test environment.

The Device-Under-Test (The I²C Master Controller)

The Device-Under-Test (DUT) is the I²C Master Controller reference design.

Microprocessor Model

The microprocessor model is the part of the test suite that simulates the microprocessor in an I²C Master Controller system and is where the designer can issue commands to the I²C Master Controller. This module was used to initiate a two-byte read or a two-byte write I²C transaction using both interrupt and polling modes. The module is made up of several tasks that can be easily reconfigured to stimulate a variety of other I²C bus transactions.

I²C Slave Model

The I²C Slave model simulates an I²C Slave serial EPROM. This part of the test suite provides the I²C Master Controller with an I²C Slave device with which to communicate.

Clock/Reset Generation Block

The Clock/Reset block generates the system clock. It also generates an initial reset when the simulation starts. The module has different clock parameters that can easily be interchanged to generate different system clock rates.

Acknowledge Generation Block

The acknowledge block is used to generate and send an acknowledge signal back to the microprocessor model four clock cycles after every microprocessor transaction. This is necessary because without this block, the microprocessor may not be able to terminate its bus cycle. The functionality of this module may need to be modified or duplicated in the Chip Select unit of the I²C Master Controller system to ensure proper timing and operation of a particular microprocessor.

I²C Master Controller Transaction Waveforms

The following waveforms illustrate the typical operation of the I²C Master Controller reference design. All the waveforms were taken from RTL simulations.

Figure 5. Microprocessor Write Cycle to Low Address Register

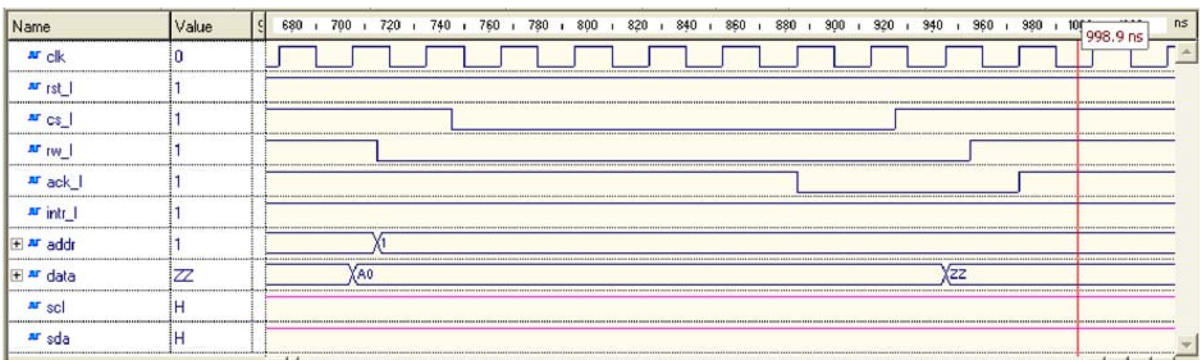


Figure 6. Microprocessor Read Cycle from Status Register

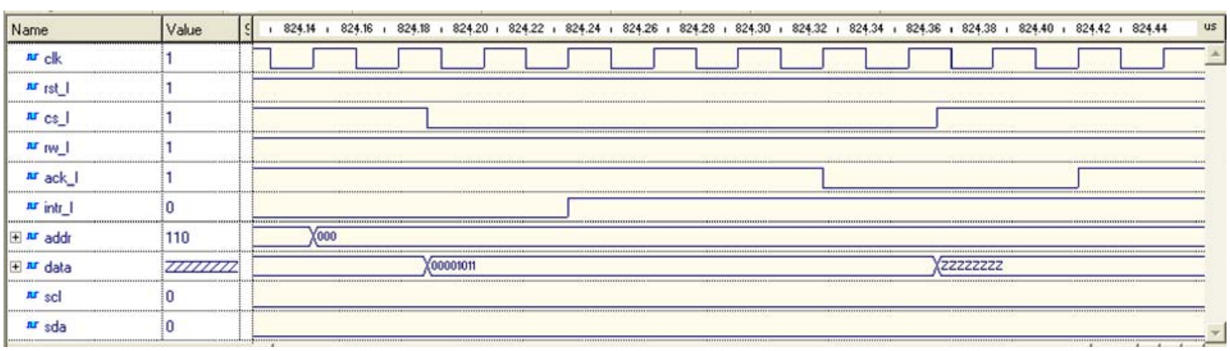
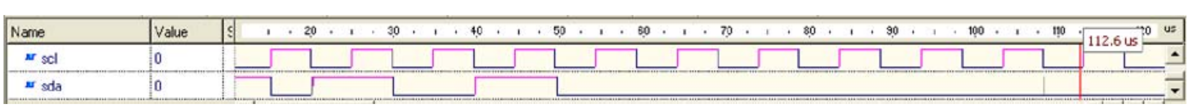


Figure 7. I²C Slave Address Write Cycle



Implementation

This design is implemented in VHDL. When using this design in a different device, density, speed, or grade, performance and utilization may vary. Default settings are used during the fitting of the design.

Table 1. Performance and Resource Utilization

Device Family	Language	Speed Grade	Utilization (LUTs)	f_{MAX}	I/O	Architecture Resources
ECP5™ [7]	Verilog-Syn	–6	195	>33	18	N/A
	VHDL-Syn	–6	198	>33	18	N/A
LatticeECP3™ [1]	Verilog-Syn	–6	200	>33	18	N/A
	VHDL-Syn	–6	200	>33	18	N/A
MachXO3L[8]	Verilog-LSE	–6	206	>33	18	N/A
	Verilog-Syn	–6	190	>33	18	N/A
	VHDL-LSE	–6	204	>33	18	N/A
	VHDL-Syn	–6	190	>33	18	N/A
MachXO2™ [2]	Verilog-LSE	–4	204	>33	18	N/A
	Verilog-Syn	–4	190	>33	18	N/A
	VHDL-LSE	–4	204	>33	18	N/A
	VHDL-Syn	–4	190	>33	18	N/A
MachXO™ [3]	Verilog-LSE	–3	199	>33	18	N/A
	Verilog-Syn	–3	183	>33	18	N/A
	VHDL-LSE	–3	197	>33	18	N/A
	VHDL-Syn	–3	183	>33	18	N/A
LatticeXP2™ [4]	Verilog-Syn	–5	201	>33	18	N/A
	VHDL-Syn	–5	201	>33	18	N/A
Platform Manager™ [6]	Verilog-Syn	–3	199	>33	18	N/A
	VHDL-Syn	–3	197	>33	18	N/A
ispMACH4000ZE™ [5]	VHDL-Syn	–5 (ns)	154	>33	18	N/A

1. Performance and utilization characteristics are generated using LFE3-17EA-6FTN256C with Lattice Diamond® 3.3 design software.
2. Performance and utilization characteristics are generated using LCMXO2-256HC-4TG100C with Lattice Diamond 3.3 design software with LSE (Lattice Synthesis Engine).
3. Performance and utilization characteristics are generated using LCMXO256C-3T100C with Lattice Diamond 3.3 design software with LSE.
4. Performance and utilization characteristics are generated using LFXP2-5E-5M132C with Lattice Diamond 3.3 design software.
5. Performance and utilization characteristics are generated using LC4256ZE-5TN144C with Lattice ispLEVER® Classic 1.4 software.
6. Performance and utilization characteristics are generated using LPTM10-1247-3TG128CES with Lattice Diamond 3.3 design software.
7. Performance and utilization characteristics are generated using LFE5U-45F-6MG285C with Lattice Diamond 3.3 design software with LSE.
8. Performance and utilization characteristics are generated using LCMXO3L-4300C-6BG256C with Lattice Diamond 3.3 design software with LSE and Synplify Pro®.

References

- Philips I2C Bus Specification version 2.1

Technical Support Assistance

e-mail: techsupport@latticesemi.com

Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
January 2015	5.9	Updated Table 1, Performance and Resource Utilization. — Added LSE support for MachXO and MachXO2 device families. — Added support for Lattice Diamond 3.3 design software.
March 2014	05.8	Updated Table 1, Performance and Resource Utilization. — Added support for ECP5 device family. — Added support for MachXO3L device family. — Added support for Lattice Diamond 3.1 design software. Updated Technical Support Assistance information.
February 2012	05.7	Updated document with new corporate logo.
April 2011	05.6	Added support for LatticeECP3 and MachXO2 device families. Added support for Lattice Diamond 1.2 design software.
December 2010	05.5	Added support for Platform Manager device family. Added support for Lattice Diamond 1.1 and ispLEVER 8.1 SP1 design software.
December 2009	05.4	Added support for LatticeXP2 device family.
July 2009	05.3	Added support for ispMACH 4000ZE device family.
February 2009	05.2	Added support for Aldec simulator.
—	—	Previous Lattice releases