I²C Master with WISHBONE Bus Interface

March 2014 Reference Design RD1046

Introduction

The I²C (Inter-IC Communication) bus has become an industrial de-facto standard for short-distance communication among ICs since its introduction in the early 1980s. The I²C bus uses two bidirectional open-drain wires with pull-up resistors. There is no strict baud rate requirement as with other communication standards. The true multi-master bus allows protection of data corruption if multiple masters initiate data transfer at the same time. These, and many other features of the I²C bus, provide efficient and flexible means for control functions that do not require high speed data transfer, and for applications that require a small amount of data exchanges.

Implementing the I²C bus master in an FPGA adds the popular communication interface to components that do not have I²C interface integrated on chip. At the same time, the FPGA frees up the on-board microcontroller for heavier tasks in the system.

The WISHBONE Bus interface is a free, open-source standard that is gaining popularity in digital systems that require usage of IP cores. This bus interface encourages IP reuse by defining a common interface among IP cores. That in turn provides portability for the system, speeds up time to market, and reduces cost for the end products.

This document and the design are based on the OpenCores I²C master core, which was used as a peripheral component for the LatticeMico32™ IP core (see the I²C-Master Core Specification from OpenCores for further information). The design provides a bridge between the I²C bus and the WISHBONE bus. A typical application of this design includes the interface between a WISHBONE compliant on-board microcontroller and multiple I²C peripheral components. The I²C master core generates the clock and is responsible for the initiation and termination of each data transfer.

Both Verilog and VHDL versions of the reference design are available. Lattice design tools are used for synthesis, place and route and simulation. The design can be targeted to multiple Lattice device families.

Features

- Compatible with I²C specification
  - Multi-master operation
  - Software-programmable SDL clock frequency
  - Clock stretching and wait state generation
  - Interrupt flag generation
  - Arbitration lost interrupt, with automatic transfer cancellation
  - Bus busy detection
  - Supports 7-bit and 10-bit addressing modes
  - Supports 100 KHz and 400 KHz modes
- Compliant with WISHBONE specification
  - RevB.3 compliant WISHBONE Classic interface
  - All output signals are registered
  - Two-cycle access time
  - A non-WISHBONE compatible signal, arst_i, is an asynchronous reset signal provided for FPGA implementations

Functional Description

The I²C master core supports the critical features described in the I²C specification and is suitable for most applications involving I²C slave control. The design responds to the read/write cycles initiated by the microcontroller through the WISHBONE interface. It provides the correct sequences of commands and data to the I²C slave device and then transfers the required data from the I²C slave device through the two open-drain wires. The I²C master with WISHBONE interface offloads the microcontroller from needing to administrate many details of the I²C commands and operation sequences.

Table 1 lists the I/O ports of the design. The signals ending in "_i" indicate an input and those ending in "_o" indicate an output. All signals on WISHBONE side are synchronous to the master clock. The two I²C wires, scl and sda, must be open-drain signals and are externally pulled up to Vcc through resistors.

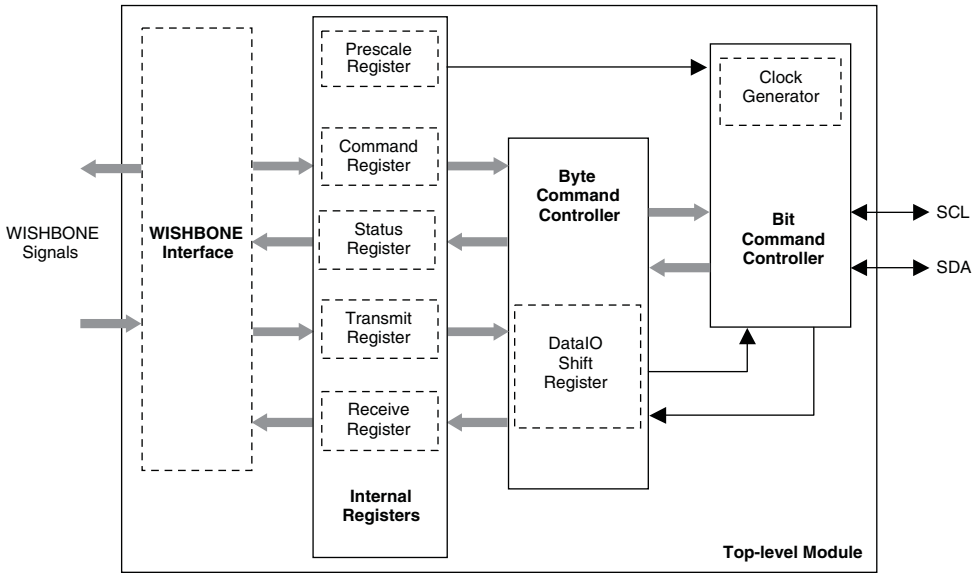
Table 1. Pin Descriptions

Signal	Width	Type	Description
WISHBONE Interface			
wb_clk_i	1	Input	Master clock
wb_rst_i	1	Input	Synchronous reset, active high
arst_i	1	Input	Asynchronous reset
wb_adr_i	3	Input	Lower address bits
wb_dat_i	8	Input	Data towards the core
wb_dat_o	8	Output	Data from the core
wb_we_i	1	Input	Write enable input
wb_stb_i	1	Input	Strobe signal/core select input
wb_cyc_i	1	Input	Valid bus cycle input
wb_ack_o	1	Output	Bus cycle acknowledge output
wb_inta_o	1	Output	Interrupt signal output
I²C Interface			
scl	1	Bidi	Serial clock line
sda	1	Bidi	Serial data line

Design Module Description

The design has four main modules as shown in Figure 1. These include one top-level module and three lower-level modules, which are the register module, the byte command module and a bit command module.

Figure 1. Design Modules



Top-level Module (i2c_master_top.v)

In addition to connecting all the functional blocks together, this module generates byte-wide data, acknowledgement, and interrupt for the WISHBONE interface. Depending on the parameter ARST_LVL, the reset polarity is determined and distributed to all the modules.

Internal Registers Module (i2c_master_registers.v)

A 2-bit by 8-bit register space constitutes the internal register structure of the I²C core. The space houses the six 8-bit registers listed in Table 2. The addresses not used are reserved for future expansion of the core.

Table 2. Internal Register List

Name	Address	Width	Access	Description
PRERlo	0x00	8	RW	Clock Prescale register lo-byte
PRERhi	0x01	8	RW	Clock Prescale register hi-byte
CTR	0x02	8	RW	Control register
TXR	0x03	8	W	Transmit register
RXR	0x03	8	R	Receive register
CR	0x04	8	W	Command register
SR	0x04	8	R	Status register

The **Prescale Register** (address = 0x00 and 0x01) is used to prescale the scl clock line based on the master clock. Since the design is driven by a (5 x scl frequency) internally, the prescale register is programmed according to the equation [master clock frequency / (5 x (sclk frequency)) - 1]. The content of this register can only be modified when the core is not enabled.

Only two bits of the **Control Register** (address = 0x01) are used for this design. The MSB of this register is the most critical one because it enables or disables the entire I²C core. The core will not respond to any command unless this bit is set.

The **Transmit Register** and the **Receive Register** share the same address (address = 0x30) depending on the direction of data transfer. The data to be transmitted via I²C will be stored in the Transmit Register, while the byte received via I²C is available in the Receive register.

The **Status Register** and the **Command Register** share the same address (address = 0x04). The Status Register allows the monitoring of the I²C operations, while the Command Register stores the next command for the next I²C operation. Unlike the rest of the registers, the bits in the Command Register are cleared automatically after each operation. Therefore this register has to be written for each start, write, read, or stop of the I²C operation. Table 3 provides a detailed description of each bit in the internal registers.

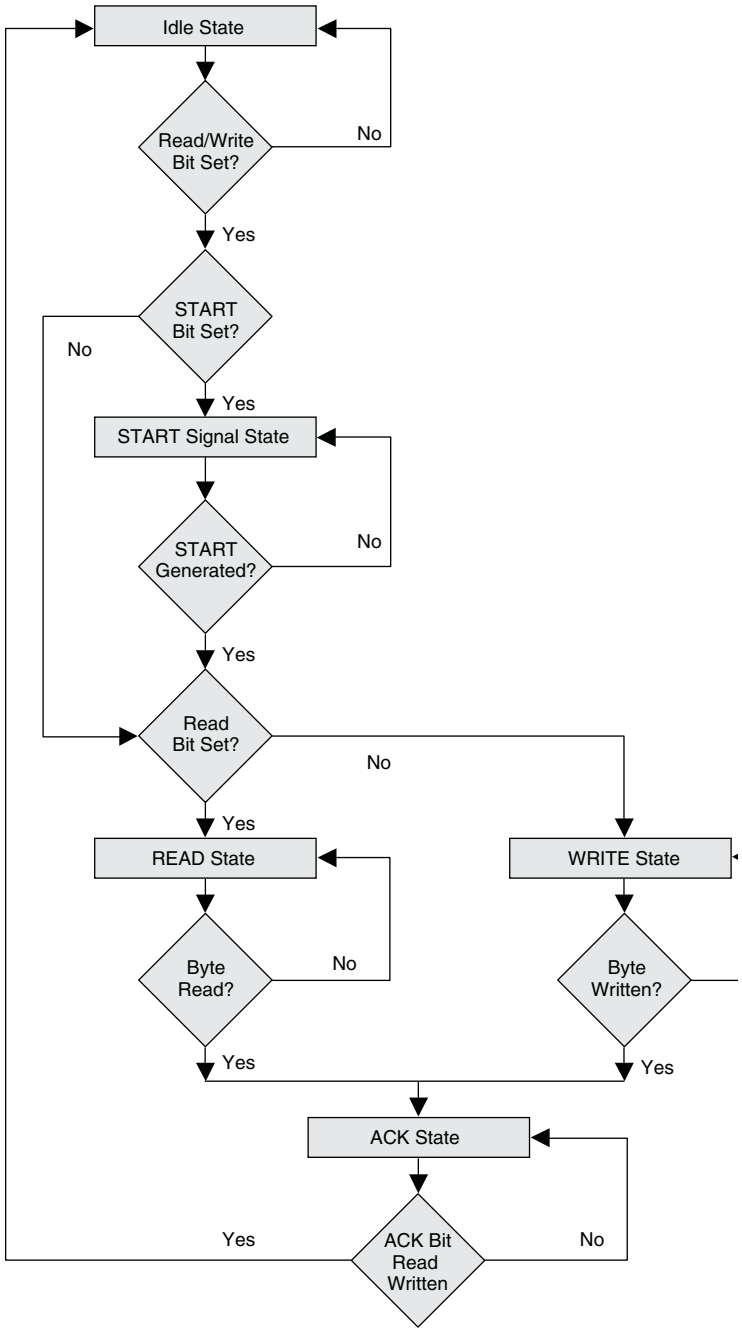
Table 3. Description of Internal Register Bits

Internal Register	Bit #	Access	Description
Control Register (0x02)	7	RW	EN, I²C core enable bit. '1' = the core is enabled; '0' = the core is disabled.
	6	RW	IEN, I²C core interrupt enable bit. '1' = interrupt is enabled; '0'=interrupt is disabled.
	5:0	RW	Reserved
Transmit Register (0x30)	7:1	W	Next byte to be transmitted via I²C
	0	W	a) This bit represents the data's LSB. b) This bit represents the RW bit during slave address transfer '1' = reading from slave; '0' = writing to slave
Receive Register (0x30)	7:0	R	Last byte received via I²C
Status Register (0x04)	7	R	RxACK, Received acknowledge from slave. This flag represents acknowledge from the addressed slave. '1' = No acknowledge received; '0' = Acknowledge received
	6	R	Busy, indicates the I²C bus busy '1' = START signal is detected; '0' = STOP signal is detected
	5	R	AL, Arbitration lost This bit is set when the core lost arbitration. Arbitration is lost when: - A STOP signal is detected, but not requested. - The master drives SDA high, but SDA is low.
	4:2	R	Reserved
	1	R	TIP, Transfer in progress. '1' = transferring data; '0' = transfer is completed
	0	R	IF, Interrupt Flag. This bit is set when an interrupt is pending, which will cause a processor interrupt request if the IEN bit is set. The Interrupt Flag is set when: - One byte transfer has been completed. - Arbitration is lost.
Command Register (0x04)	7	W	STA, generate (repeated) start condition
	6	W	STO, generate stop condition
	5	W	RD, read from slave
	4	W	WR, write to slave
	3	W	ACK, when a receiver, sent ACK (ACK = '0') or NACK (ACK = '1')
	2:1	W	Reserved
	0	W	IACK, Interrupt acknowledge. When set, clears a pending interrupt.

Byte Command Controller Module (i2c_master_byte_ctrl.v)

The microcontroller issues commands and data through the WISHBONE interface in byte format. The information is fed into the Byte Command Controller module and is translated into I²C sequences required for a byte transfer. This module includes a state machine, as shown in Figure 2, to handle normal I²C transfer sequences. The module then breaks up a single command into multiple clock cycles for the Bit Command Controller to work on bit-level I²C operations. This module also contains a shift register which is used for both READ and WRITE cycles. During a READ cycle, the input to the shift register comes from the sda line. After eight scl cycles, the shifted-in data is copied into the Receive Register. During a WRITE cycle, the input to the shift register comes from the WISHBONE data bus. The data in the shift register is shifted out to the sda line during WRITE.

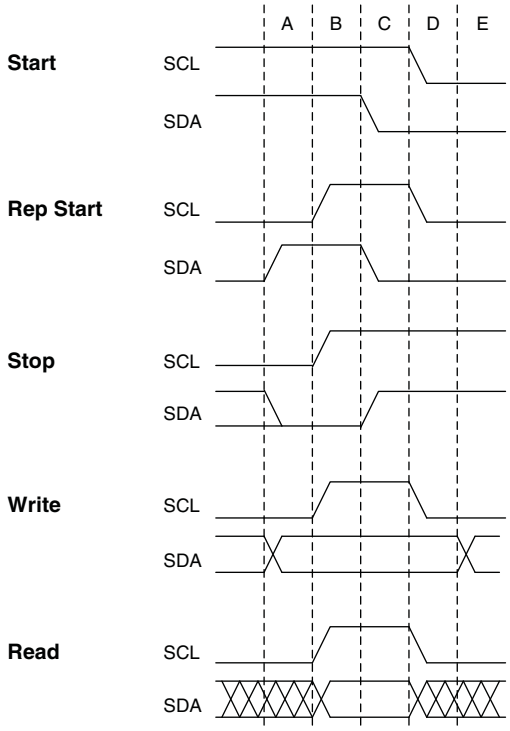
Figure 2. I²C Byte Command State Machine



Bit Command Controller Module (i2c_master_bit_ctrl.v)

This module directly controls the I²C bus, scl and sda lines, by generating the correct sequences for START, STOP, Repeated START, READ, and WRITE commands. Each bit operation is divided into five (5 x scl frequency) clock cycles (idle, A, B, C, and D), except for the START command that has six clock cycles. This ensures that the logical relationship between the scl and sda lines meets the I²C requirement for these critical commands. The internal clock running at 5 x scl frequency is used for the registers in this module.

Figure 3. I²C Bit Command Illustration



Miscellaneous Features

In addition to supporting basic I²C operation, this design also supports the 10-bit addressing schedule specified in the I²C specification. The 10-bit addressing scheme expands the addressable slave devices from less than 128 to more than 1000. The 10-bit addressing differentiates itself from the 7-bit addressing by starting the address with 11110xx. The last two bits of this first address plus the following 8 bits on the I²C sda line define the 10-bit address. The data is still being transferred in byte format, as with the 7-bit addressing.

By the nature of open-drain signal, the I²C provides clock synchronization through a wired-AND connection on the scl line. This clock synchronization capability can be used as a handshake between the slave and master I²C devices. By holding the scl line low, the slave device tells the master to slow down the data transfer until the slave device is ready. This design detects the scl line to determine if the line is being held.

This design supports multiple masters and thus incorporates the arbitration lost detection. The master that loses the arbitration reports the status in Status Register bit 5. The arbitration is lost when the master detects a STOP condition which is not requested, or when the master drives the sda line high but the sda line is pulled low. The arbitration lost resets the bits in the Command Register to clear the current command for the master to start over again.

These features are described in detail in the I²C specification.

Common Operation Sequence

The I²C Master supports common I²C operations. The sequence of I²C WRITE and I²C READ is described in this section.

Initialize the I²C Master Core:

1. Program the clock PRESCALE registers, PRERlo and PRERhi, with the desired value. This value is determined by the clock frequency and the speed of the I²C bus.
2. Enable the core by writing 8'h80 to the Control Register, CTR.

Write to a slave device (no change in direction):

1. Set the Transmit Register TXR with a value of Slave address + Write bit.
2. Set the Command Register CR to 8'h90 to enable the START and WRITE. This starts the transmission on the I²C bus.
3. Check the Transfer In Progress (TIP) bit of the Status Registesr, SR, to make sure the command is done.
4. Set TXR with a slave memory address for the data to be written to.
5. Set CR with 8'h10 to enable a WRITE to send to the slave memory address.
6. Check the TIP bit of SR, to make sure the command is done.
7. Set TXR with 8-bit data for the slave device.
8. Set CR to 8'h10 to enable a WRITE to send data.
9. Check the TIP bit of SR, to make sure the command is done.
10. Repeat steps 7 to 9 to continue to send data to the slave device.
11. Set the TXR with the last byte of data.
12. Set CR to 8'h50 to enable a WRITE to send the last byte of data and then issue a STOP command.

Read from a slave device (change in direction):

1. Set the Transmit Register TXR with a value of Slave address + Write bit.
2. Set the Command Register CR to 8'h90 to enable the START and WRITE. This starts the transmission on the I²C bus.
3. Check the Transfer In Progress (TIP) bit of the Status Register, SR, to make sure the command is done.
4. Set TRX with the slave memory address, where the data is to be read from.
5. Set CR with 8'h10 to enable a WRITE to send to the slave memory address.
6. Check the TIP bit of SR, to make sure the command is done.
7. Set TRX with a value of Slave address + READ bit.
8. Set CR with the 8'h90 to enable the START (repeated START in this case) and WRITE the value in TXR to the slave device.
9. Check the TIP bit of SR, to make sure the command is done.
10. Set CR with 8'h20 to issue a READ command and then an ACK command. This enables the reading of data from the slave device.
11. Check the TIP bit of SR, to make sure the command is done.
12. Repeat steps 10 and 11 to continue to read data from the slave device.
13. When the Master is ready to stop reading from the Slave, set CR to 8'h28. This will read the last byte of data and then issue a NACK.

HDL Simulation and Verification

The I²C master with WISHBONE interface design is simulated using an I²C slave model (i2c_slave_model.v) and a WISHBONE master model (wb_master_model.v). The slave model emulates the responses of an I²C slave device by sending ACK when the address is matching and when the WRITE operation is completed. The master model contains several tasks to emulate WISHBONE READ, WRITE, and Compare commands normally issued by the microcontroller. The top-level testbench (tst_bench_top.v) controls the flow of the I²C operations. The START, WRITE, REPEATED START, READ, consecutive READ, ACK/NACK, STOP, and clock stretching operations are simulated with this testbench.

The following timing diagrams shows the major timing milestones in the simulation.

Figure 4. Writing Prescale Register with 0x64 and 0x00 at Addresses 0x00 and 0x01 Respectively

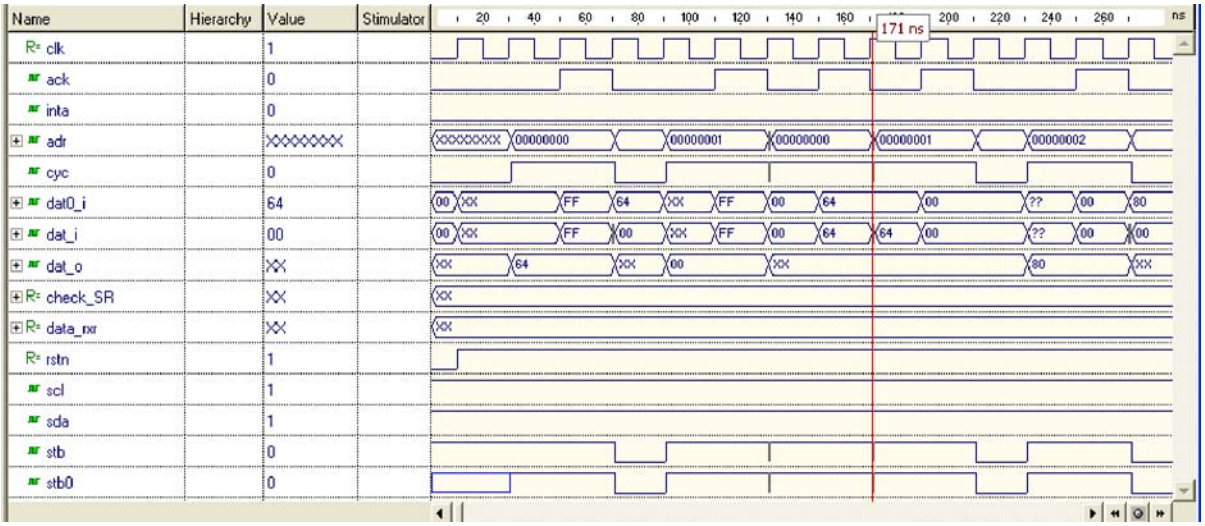


Figure 5. Initiate a START, SR[1] (Transfer in Progress) and SR[6] (Busy) Are Set

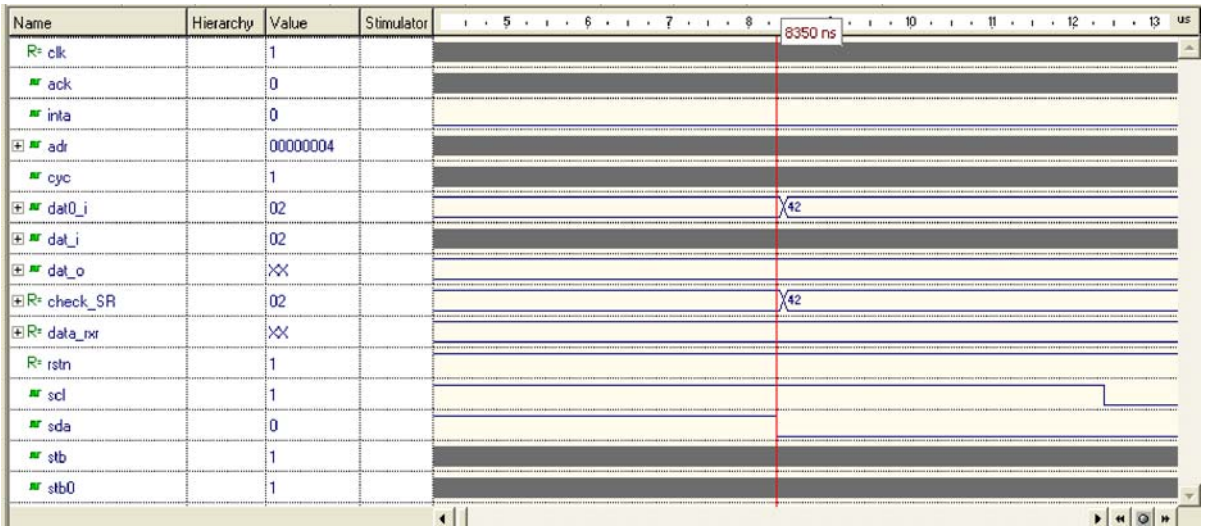


Figure 6. Transfer Slave Address + WR, Receive ACK from Slave, Transfer Slave Memory Address 0x01, Receive ACK from Slave, Release SR[1] (Transfer in Progress)

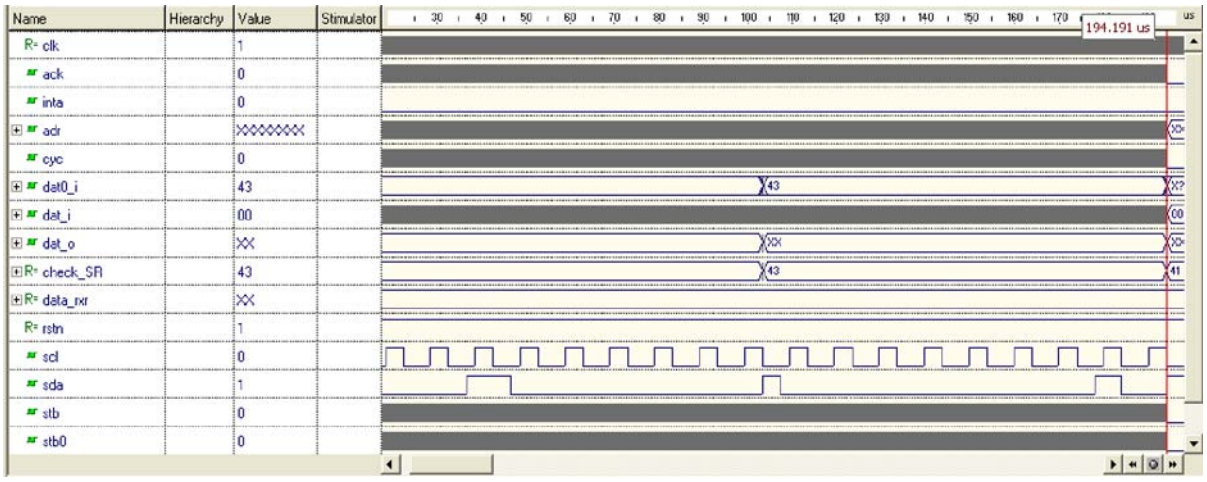


Figure 7. Clock Stretching by Slave, scl Line Held Low

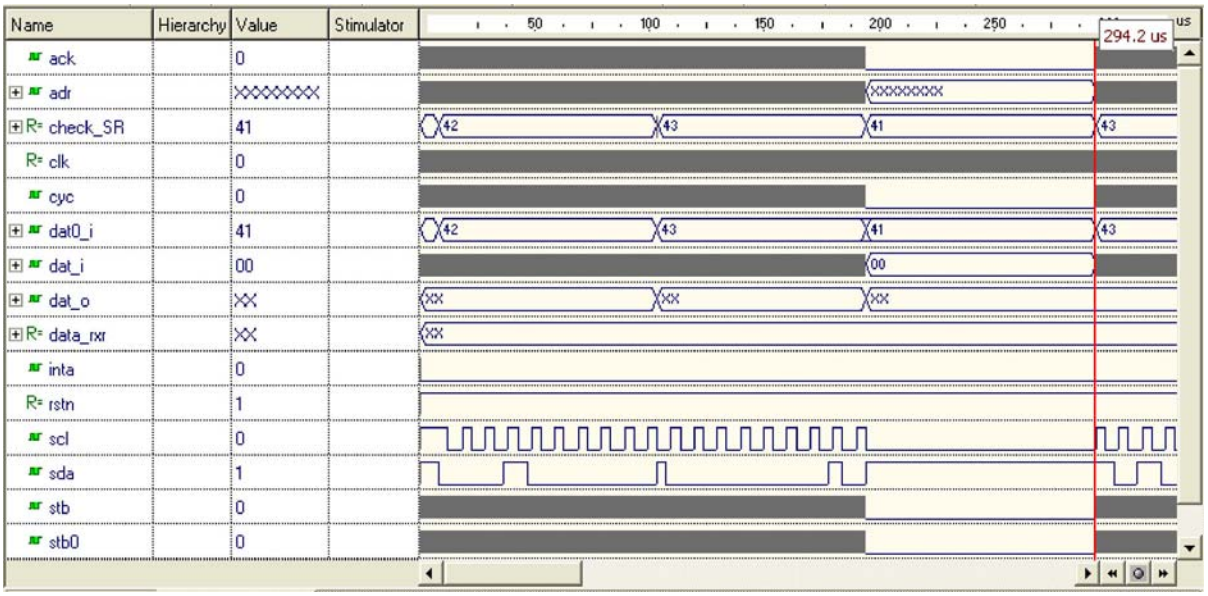


Figure 8. Repeated START with Slave Address + RD Command

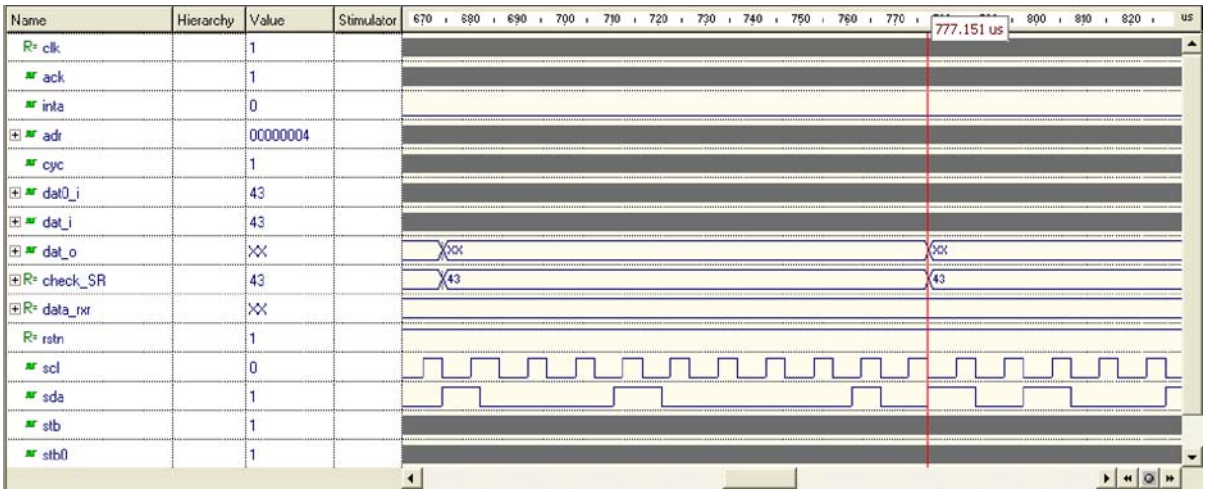


Figure 9. Consecutive READ from the Slave, Data Read are 0xA5, 0x5A, and 0x11

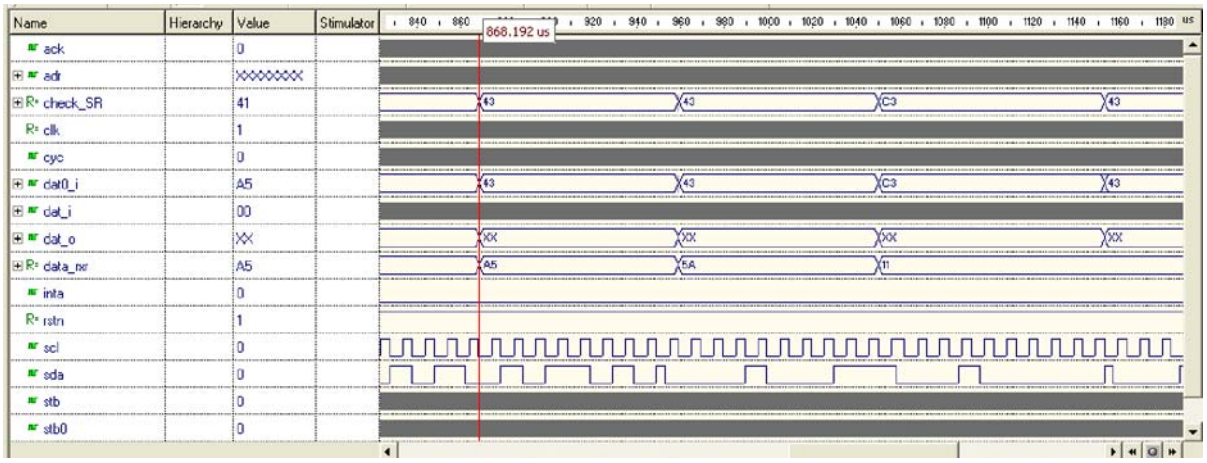
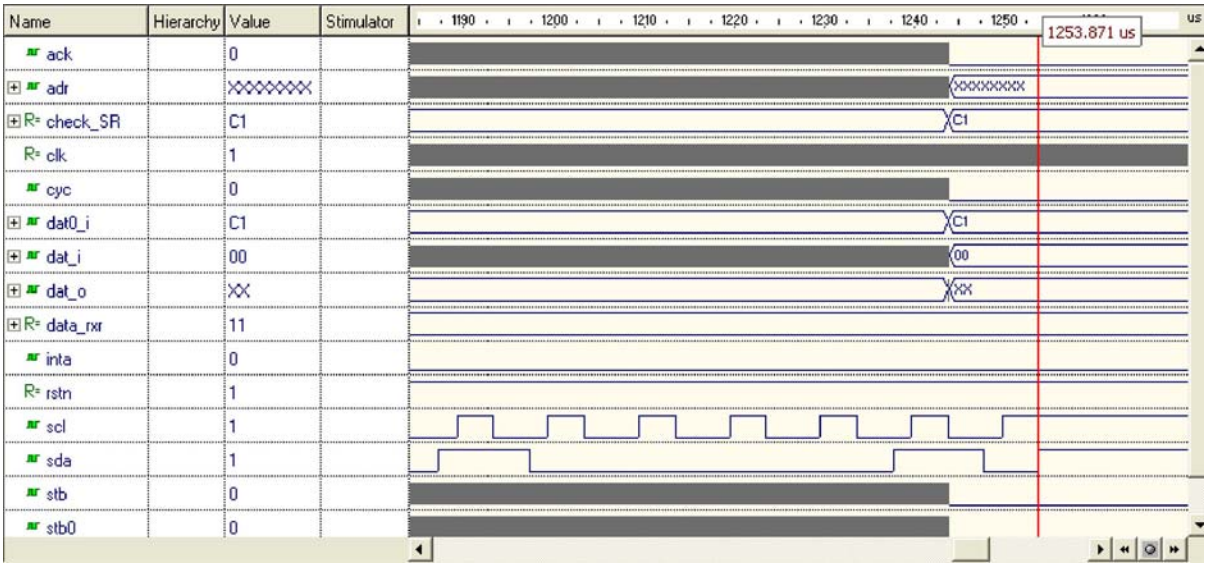


Figure 10. Slave Generates NACK, Master Issues a STOP



Implementation

This design is implemented in Verilog and 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 4. Performance and Resource Utilization[1]

Family	Language	Speed Grade	Utilization (LUTs)	f_{MAX} (MHz)	I/Os	Architecture Resources
MachXO3L[6]	Verilog-LSE	-6	201	>50	29	N/A
	Verilog-Syn	-6	243	>50	29	N/A
	VHDL-LSE	-6	218	>50	29	N/A
	VHDL-Syn	-6	243	>50	29	N/A
MachXO2™ [1]	Verilog	-4	201	>50	29	N/A
	VHDL	-4	218	>50	29	N/A
MachXO™ [2]	Verilog	-3	198	>50	29	N/A
	VHDL	-3	217	>50	29	N/A
ECP5™ [5]	Verilog	-6	203	>50	29	N/A
	VHDL	-6	209	>50	29	N/A
LatticeECP3™ [3]	Verilog	-6	261	>50	29	N/A
	VHDL	-6	252	>50	29	N/A
LatticeXP2™ [4]	Verilog	-5	252	>50	29	N/A
	VHDL	-5	248	>50	29	N/A

1. Performance and utilization characteristics are generated using LCMXO2-1200HC-4TG100C with Lattice Diamond® 3.1 design software with LSE (Lattice Synthesis Engine).
2. Performance and utilization characteristics are generated using LCMXO256C-3T100C with Lattice Diamond 3.1 design software with LSE.
3. Performance and utilization characteristics are generated using LFE3-17EA-6FTN256C with Lattice Diamond 3.1 design software.
4. Performance and utilization characteristics are generated using LFXP2-5E-5M132C with Lattice Diamond 3.1 design software.
5. Performance and utilization characteristics are generated using LFE5U-45F-6MG285C with Lattice Diamond 3.1 design software with LSE.
6. Performance and utilization characteristics are generated using LCMXO3L-4300C-6BG256C with Lattice Diamond 3.1 design software with LSE and Synplify Pro®.

References

- Philips I²C specification
- I²C-Master Core Specification from Open Cores (Author: Richard Herveille)

Technical Support Assistance

e-mail: techsupport@latticesemi.com

Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
February 2009	01.0	Initial release.
August 2009	01.1	Added VHDL source file and testbench.
December 2009	01.2	Added support for LatticeXP2 device family.
November 2010	01.3	Added support for MachXO2 device family and Lattice Diamond design software.
April 2011	01.4	Added support for LatticeECP3 device family and Lattice Diamond 1.2 design software.
March 2014	01.5	Updated Table 4, 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 corporate logo.
		Updated Technical Support Assistance information.