I²C Master Controller

February 2015 **Reference Design RD1139**

Introduction

I²C or Inter-Integrated Circuit is a popular serial interface protocol that is widely used in many electronic systems. The I²C interface is a two-wire interface capable of half-duplex serial communication at moderate to high speeds of up to a few megabits per second. There are thousands of I²C peripherals on the market today, ranging from data converters to video processors. The I²C bus is a good choice for designs that need to communicate with low-speed peripherals due to its simplicity and low cost.

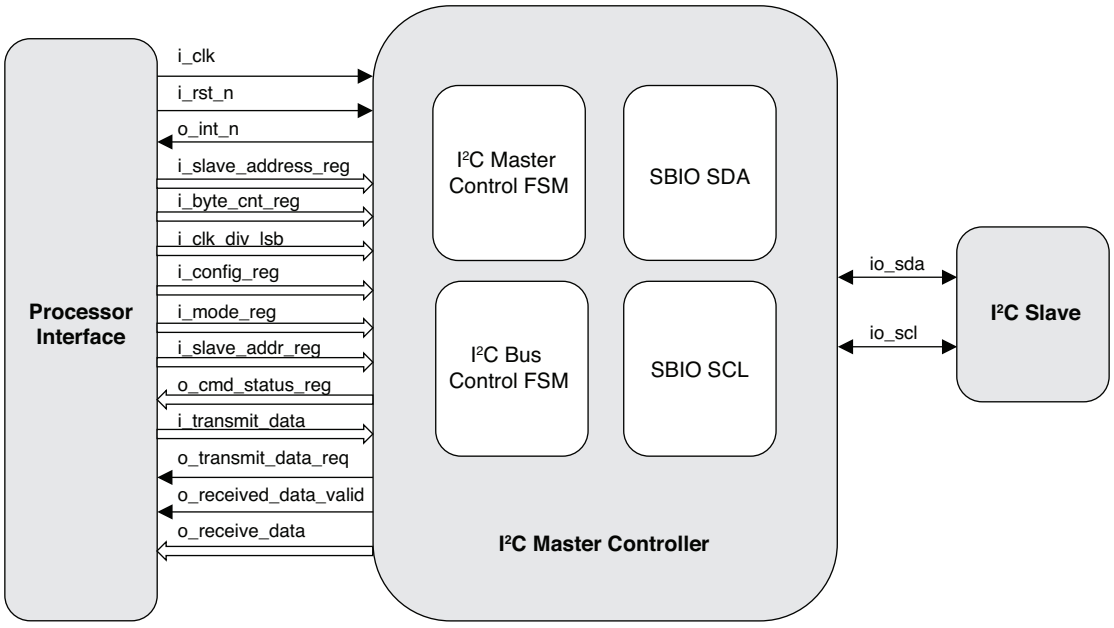
The I²C Master Controller reference design is implemented in Verilog. The Lattice iCECube2™ Place and Route tool integrated with the Lattice Synthesis Engine (LSE) tool is used for the implementation of the design. The design uses an iCE40™ ultra low density FPGA and can be targeted to other iCE40 family members.

Features

- 7-bit slave address support
- Supports operation at 100 kHz (Standard Mode) and 400 kHz (Fast Mode)
- Supports repeated start operations
- Interrupt generation logic
- Verilog RTL, test bench

Functional Description

Figure 1. Block Diagram



Pin Descriptions

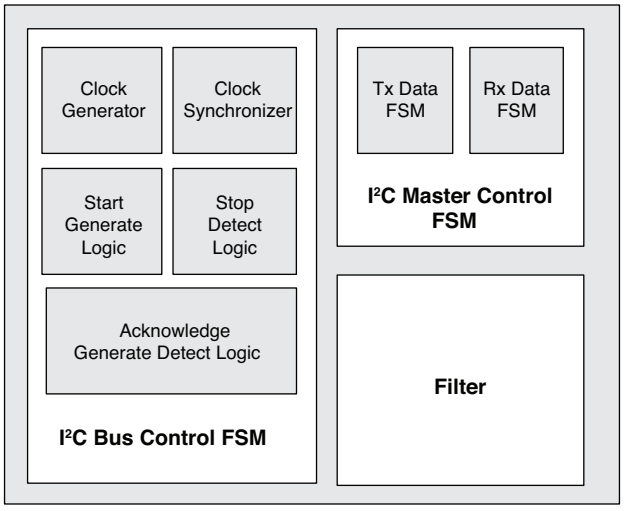
Table 1. Pin Descriptions

Signal	Width	Type	Description
i_clk	1	Input	System clock operating at 32 MHz
i_rst_n	1	Input	Asynchronous active-low system reset
o_int_n	1	Output	Active-low processor interrupt
i_slave_addr_reg	8	Input	7-bit I²C slave address
i_byte_cnt_reg	8	Input	Sets the number of data bytes to be read or written for the I²C transaction
i_clk_div_lsb	8	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	6	Input	Used to configure the I²C Master Controller (see Table 1)
i_mode_reg	8	Input	Sets the various modes of operation like speed, read/write (see Table 1)
o_cmd_status_reg	8	Output	Lets the user know the status of the operation, I²C bus (see Table 1)
o_start_ack	1	Output	Acknowledge to the start bit provided by the user through i_config_reg
i_transmit_data	8	Input	Data to be transmitted over the SDA line to the I²C slave
o_transmit_data_requested	1	Output	Lets the user know that transmit data is required
o_received_data_valid	1	Output	A '1' corresponds to valid data availability on the o_receive_data line
o_receive_data	8	Output	Received data bus
io_scl	1	Inout	I²C clock line
io_sda	1	Inout	I²C data line

Design Modules

The design includes the modules shown in Figure 2.

Figure 2. Functional Block Diagram



I²C Bus Control FSM

The I²C Bus Control FSM is comprised of the Clock Generator/Synchronizer, Start/Stop generate/detect logic and Acknowledge generate/detect logic.

The Clock Generation and Synchronization logic generates the I²C clock signal SCL, based on the system clock and clock divide factors 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 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 pull-ups. A start condition is signalled by transitioning SDA from high to low while SCL is still high. Likewise, a stop condition is signalled by transitioning SDA from low to high while SCL is high.

I²C Master Control FSM

For controlling data transfer, the I²C master makes use of 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.

Internal Register Map

The I²C Master Controller configuration can be performed on run-time. Table 2 lists the registers available to the user.

Table 2. Register List

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

Register Bit Descriptions

Table 3. Register Bit Descriptions

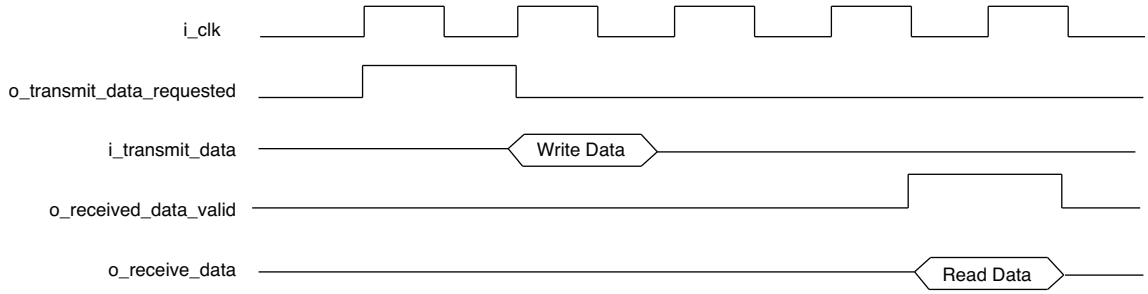
Register Bit	Description
i_slave_addr_reg[7:0]	7-bit slave address
I_byte_cnt_reg[7:0]	Sets the number of data bytes to be written or read for the I²C transaction. For example, set the register to '8' to transfer eight data bytes.
i_clk_div_lsb[7:0]	Sets the lower byte of the clock divider that is used to generate SCL from CLK. The upper three bits are located in the Mode Register. Please note that DIV[0] is not used since only even DIV values are supported.
i_config_reg[5] – RESET –	Writing a '1' will reset this I²C Master Controller.
i_config_reg[4] - 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.
i_config_reg[3] - TX_IE –	Set this bit high to enable interrupt generation on transmit completion and a STOP is issued, or on any error.
i_config_reg[2] - RX_IE	Set this bit high to enable interrupt generation on receive completion and a STOP is issued, or on any error.
i_config_reg[1] - INT_CLR	Writing a '1' will clear all bits in the Command Status Register except the I2C_BUSY bit. Write a '0' to clear this bit.
i_config_reg[0] - 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.
i_mode_reg[7:6] - BPS[1:0]	Selects the I²C speed mode (2'b00 = standard, 2'b01 = fast, others are reserved)
i_mode_reg[4] - ACK_POL	Sets 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 a repeat START is not used, this bit should be set to '1' (NACK) for I²C compliance.
i_mode_reg[3]	RW_MODE – Sets the read or write operation on the I²C bus ('0' = write, '1' = read)
i_mode_reg[2:0]	DIV[10:8] – The upper three bits of the clock divide facto
o_cmd_status_reg[7]	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.
o_cmd_status_reg[6] - 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.
o_cmd_status_reg[5] - 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.
o_cmd_status_reg[4] - TX_ERR	This read-only status bit indicates that an error has occurred during the I²C write operation.
o_cmd_status_reg[3] - RX_ERR	This read-only status bit indicates that an error has occurred during the I²C read operation.
o_cmd_status_reg[2] - 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 3egister.

Timing Diagram

Figure 3 shows the timing diagram for read and write data transfer to the I²C Master Controller.

Figure 3. Timing Diagram for Read and Write Data Transfer



Operation Sequence

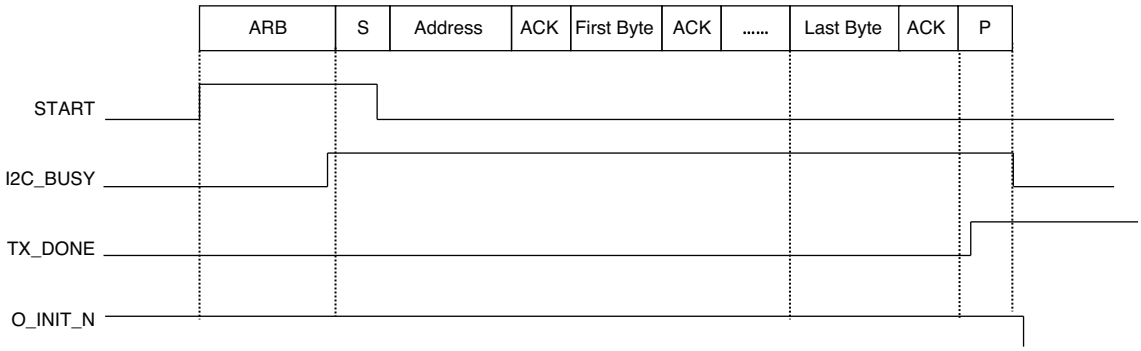
The operation sequence for transmit, receive and repeated start is illustrated below.

I²C Master Transmit Operation

Figure 4 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.

Figure 4. Master Transmit Operation



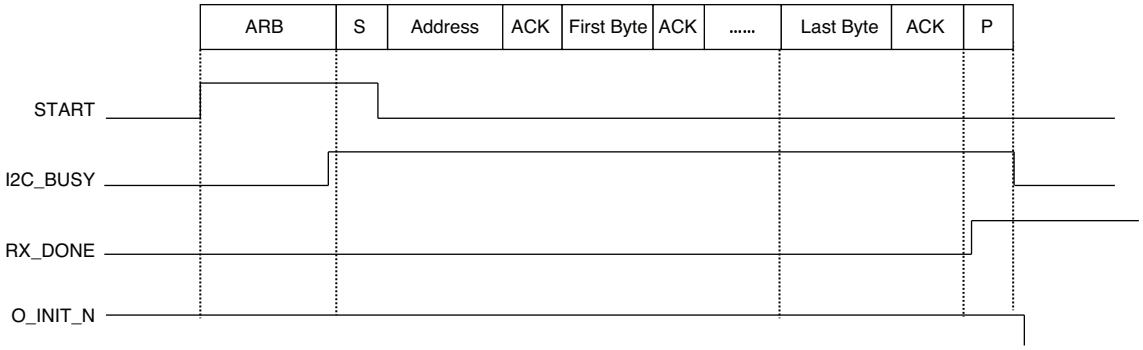
The I2C_BUSY bit will go high before the START bit is cleared to indicate an active I²C transaction.

1. The I2C_BUSY bit will clear itself after the entire I²C transaction is complete and the bus is idled.
2. The TX_DONE signal is asserted after the last byte of transmit data has been sent, but the STOP condition may not have completed yet.
3. I2C_BUSY is cleared after the STOP condition has completed and the bus is idled.
4. The interrupt is asserted if the transmit interrupt enable bit (TX_IE) is set.
5. If the slave device did not assert ACK, then TX_ERR and INT_N will both be asserted.

I²C Master Receive Operation

Figure 5 illustrates the basic receive operation of the I²C Master Controller.

Figure 5. Master Receive Operation



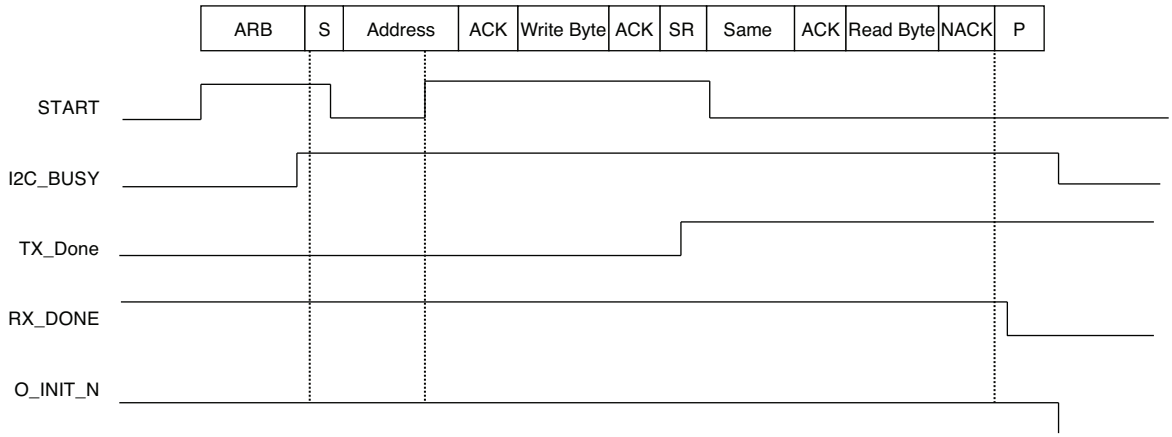
1. By writing a '1' to the START bit of the configuration register, the operation is started.
2. The I2C_BUSY bit will go high before the START bit is cleared to indicate an active I²C transaction. The I2C_BUSY bit will clear itself after the entire I²C transaction is complete and the bus is idled.
3. The RX_DONE signal is asserted after the last byte of receive data have been latched, but the STOP condition may not have completed yet.
4. I2C_BUSY is cleared after the STOP condition has completed and bus is idled.
5. The interrupt is asserted if the receiver interrupt enable bit (RX_IE) is set.
6. 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 slaves uses repeated start transactions to allow indirect access of internal registers. The I²C Master Controller supports the generation of repeat start transactions. Figure 6 illustrates a typical repeat start I²C transaction.

Note that "SR" is the I²C notation for a repeated start.

Figure 6. Repeated Start Operation



By writing a ‘1’ to the START bit of the configuration register, the operation is started.

1. The initial 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, and 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 the STOP condition is complete and the bus is idled.

HDL Simulation

Figure 7. HDL Simulation Waveform

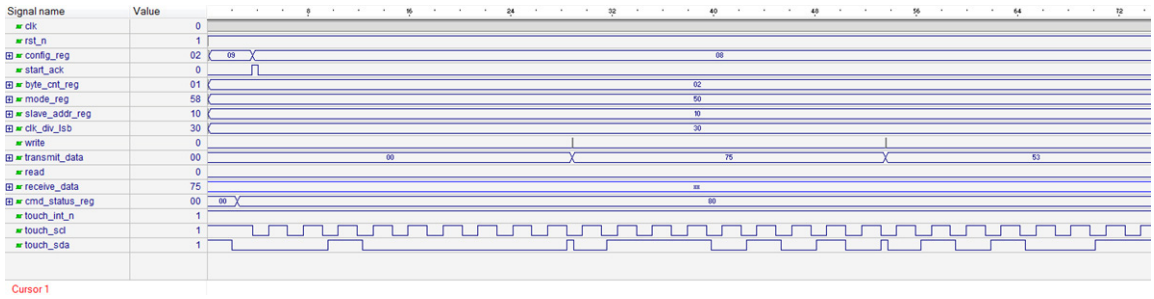


Figure 8. Simulation Waveform Showing the START Bit of the Configuration Register and Acknowledge Bit

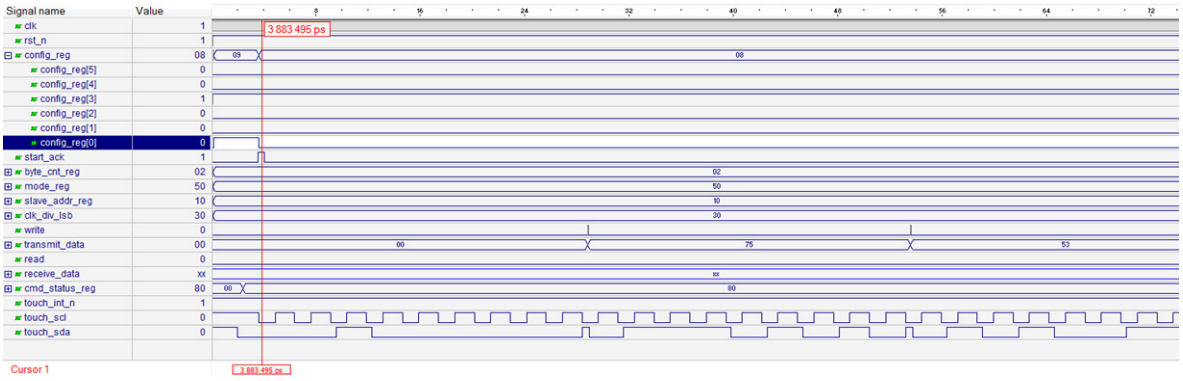


Figure 9. Timing Waveforms Showing Config_reg START and Rx Bits Along with Cmd_status_reg TX_Done and I2C_Busy Bits

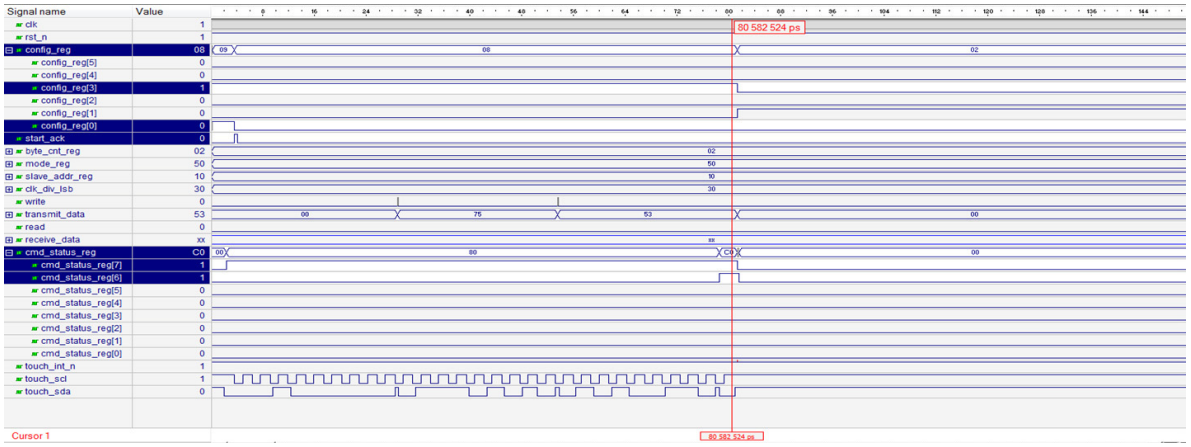
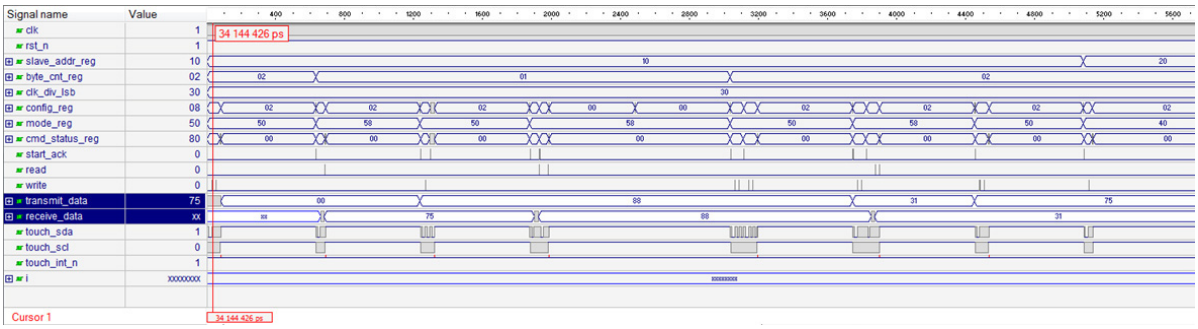


Figure 10. Simulation Timing Diagram Showing Transmit and Receive Operations



Implementation

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

Table 4. Performance and Resource Utilization

Device Family	Language	Synthesis Tool	Utilization (LUTs)	fMAX (MHz)	I/Os	Architecture Resources
iCE40[1]	Verilog	LSE	295	113.55	65	N/A
		Syn Pro	282	77.52	65	N/A

1. Performance utilization characteristics are generated using iCE40LP1K-CM121 with iCEcube2 2014.08 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 Introduction section. Changed Synplify Pro to Lattice Synthesis Engine (LSE).
		Updated Implementation section. Updated Table 4, Performance and Resource Utilization. — Changed Utilization (LUTs) value.
		Updated References section.
		Updated Technical Support Assistance information.
October 2012	01.0	Initial release.