I²C Slave/Peripheral

December 2014 Reference Design RD1054

Introduction

I²C (Inter-IC) bus is a simple, low-bandwidth, short-distance protocol. It is often seen in systems with peripheral devices that are accessed intermittently. It is also a common communication solution in a close system where minimum trace on the board is desired. Many semiconductor vendors support I²C-compatible devices in embedded systems including EEPROM, temperature sensors, current sensors and clocks. The 2-wire interface allows serial transmission of 8-bit bytes of data and 7-bit addresses with control bits. The device that initiates the transmission on the I²C bus is commonly known as the master, while the device being addressed is called the slave.

This design implements an I²C slave module in a FPGA or CPLD. It follows the I²C specification to provide device addressing, read/write operation and an acknowledgement mechanism. It adds an instant I²C compatible interface to any component in the system. The programmable nature of FPGA and CPLD devices provides users with the flexibility of configuring the I²C slave device to any legal slave address. This avoids the potential slave address collision on an I²C bus with multiple slave devices.

Features

The I²C slave module is based on the I²C specification. Users are expected to have a basic understanding of the I²C specification and protocols.

- Compatible with I²C Bus Specification version 2.1
- Software programmable slave address
- Supports random read/write
- Support sequential read/write
- Uses ACK /NACK to communicate with I²C master
- Supports clock stretching as handshake
- Start/Stop/Repeated Start detection
- Supports 7-bit addressing mode

Functional Description

The I²C slave module supports the major features listed in the I²C specification. It responds to the commands issued by an I²C master. In normal situations, the I²C slave acknowledges the successful receive of address or data by pulling down the sda line during the ACK stage. When a slave cannot process the information fast enough, it uses one of two methods to inform the master. The slave simply does not respond during the ACK stage, which only happens in a write operation where a slave can choose to acknowledge or not to acknowledge. This causes the master to issue a stop command and retry. The slave can also use a clock stretching technique by pulling down the scl line as long as it needs. The master will wait until the scl is released and continue to complete the previous transmission.

Figure 1. I²C Slave Block Diagram

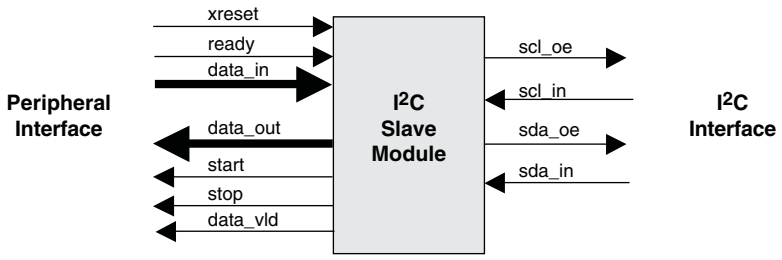


Table 1 lists the signals for I²C slave module. The scl and sda signals are separated into data signals and output enable signals. User can manually tie the signals to create a bidi sda signal, as shown in Figure 2 and demonstrated in the testbench.

Figure 2. scl and sda Signals for I²C Channel

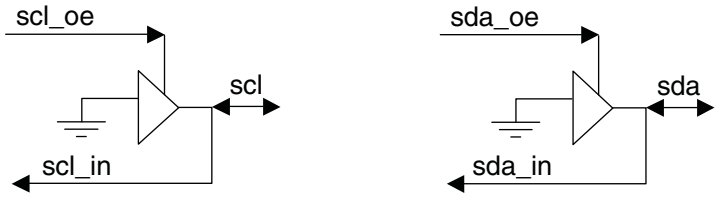


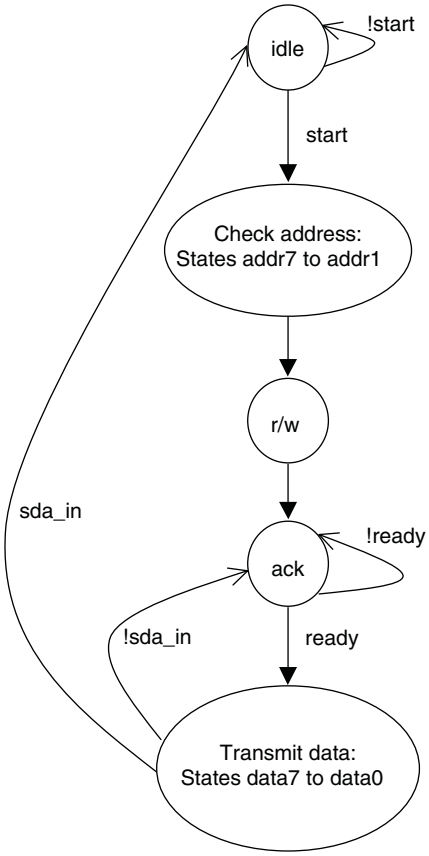
Table 1. Pin Descriptions

Signal	Direction	Description
I²C Interface		
scl_in	Input	I²C clock signal
scl_oe	Output	I²C clock output enable control
sda_in	Input	I²C data input signal
sda_oe	Output	I²C data output enable control
Peripheral Interface		
xreset	Input	Global reset signal
ready	Input	Ready signal from slave side
data_in[7:0]	Input	Parallel input data to be transmitted through I²C bus
data_out[7:0]	Output	Parallel output data received from I²C bus
start	Output	Status signal – indicates the start of I²C transmission
stop	Output	Status signal – indicates the stop of I²C transmission
data_vld	Output	Status signal – indicates data is ready
r_w	Output	Status signal – indicates master is reading or writing to the slave

Design Module Description

The I²C slave design (i2c_slave.v) consists of a state machine which steps through the I²C operation mechanism. The top-level representation of the state machine is shown in Figure 3. The start and stop conditions are detected asynchronously to initiate or terminate the I²C operations. The ACK is generated when the slave address matching the address of the slave device and when a byte of data is successfully received. The state machine supported sequential/continuous read and write by returning to the ACK state. The inactivated “ready” signal indicates that the slave needs time to process the data from the master. This signal pulls down the scl line during the ACK state and holds the scl there until slave is ready to process the data again. Since scl is not available, the state machine stays at the ACK state and will continue once the scl is released. This “clock stretching” capability allows the handshaking between slave and master in addition to using the ACK or NACK bit. The “data_vld” signal indicates a byte of data is successfully received or transmitted.

Figure 3. Top-level I²C Slave State Diagram



The actual interpretation of the received data is handled in the testbench. This gives an example of how the data received from the I²C channel can be interpreted while giving users the flexibility to add their own peripheral interface.

Common I²C Operation Sequence

This design supports common I²C operations. The sequence of each operation is described in this section.

Burst Read or Sequential Read after sending slave address (no change in direction):

1. Master issues a START condition
2. Master sends I²C slave address + one H bit (READ)
3. Slave issues ACK to master if address matches
4. Master receives READ data, and issues ACK for each received byte
5. Master stops the operation by issuing NACK after last read byte, followed by a STOP condition

Burst Write or Sequential Write (no change in direction):

1. Master issues a START condition
2. Master sends I²C slave address + one L bit (WRITE)
3. Slave issue ACK to master if address matches
4. Master sends address to be written to
5. Slave issue ACK to master
6. Master sends data to the specified address, and slave issues ACK for each byte of Write data
7. Master stops the operation by issuing a STOP condition after the slave's ACK.

Random Read (change in direction):

1. Master issues a START condition
2. Master sends I²C slave address + one L bit (WRITE)
3. Slave issues ACK to master if address matches
4. Master sends address to be read from
5. Slave issues ACK to master
6. Master issues a repeated START condition
7. Master sends I2C slave address + one H bit (READ)
8. Slave sends ACK to master if address matches
9. Master receives data, and issues a ACK for each received byte
10. Master stops the operation by issuing a NACK after the last byte, follow by a STOP condition

Random Write (no change in direction):

1. Master issues a START condition
2. Master sends I²C slave address + one L bit (WRITE)
3. Slave issues ACK to master if address matches
4. Master sends the address the data to be written to
5. Slave issues ACK after receiving the address
6. Master sends a byte of data to slave, slave issues a ACK for each byte written
7. Master stops the operation by issuing a STOP condition after slave's ACK

HDL Simulation and Verification

The I²C slave module is simulated with an I²C master module (i2c_mstr.v) and a top-level testbench (i2c_peri_tf.v). Figure 4 shows the structure of the testbench. The master module generates slave address, and read/write commands. The slave module corresponds to the commands and either receive data or supply data through the I²C channel. The top-level testbench provides a module to determine if the received information is the back-end memory address or data. The information is then used to read/write from an emulated memory bank (i2c.mem). The following diagrams show the simulation results of the module.

Figure 4. I²C Simulation Structure

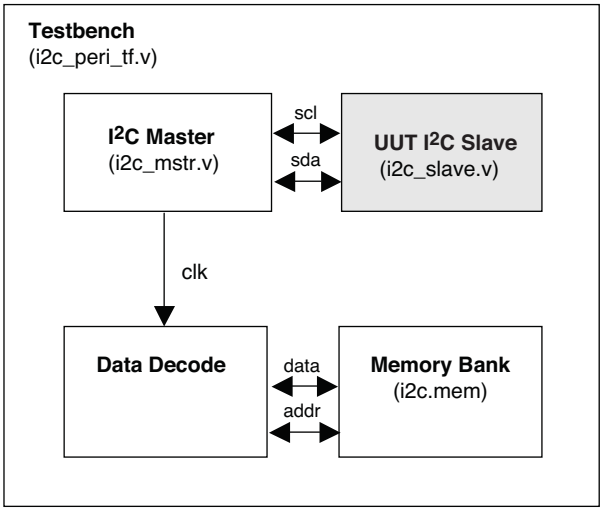


Figure 5. Sequential Read, Slave Address=h52, Data Read are hC0, h35 and h11

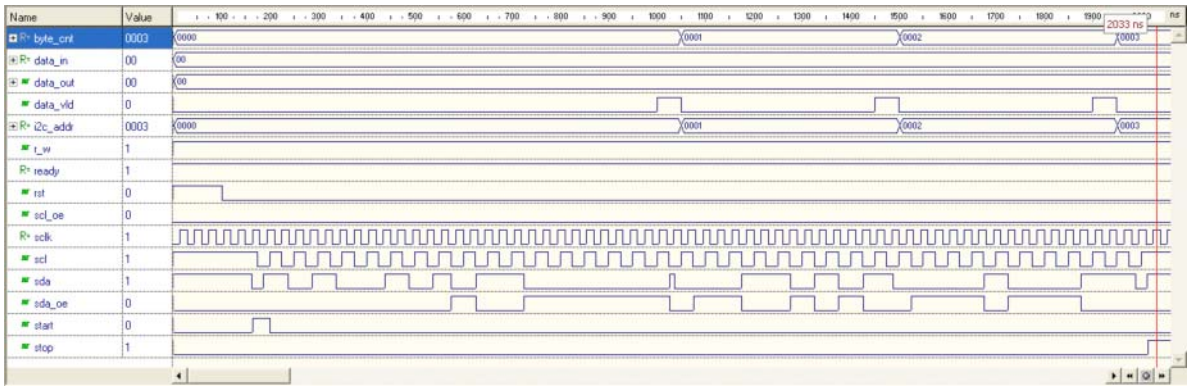


Figure 6. Random Write at Memory Address h0A, with Data h55

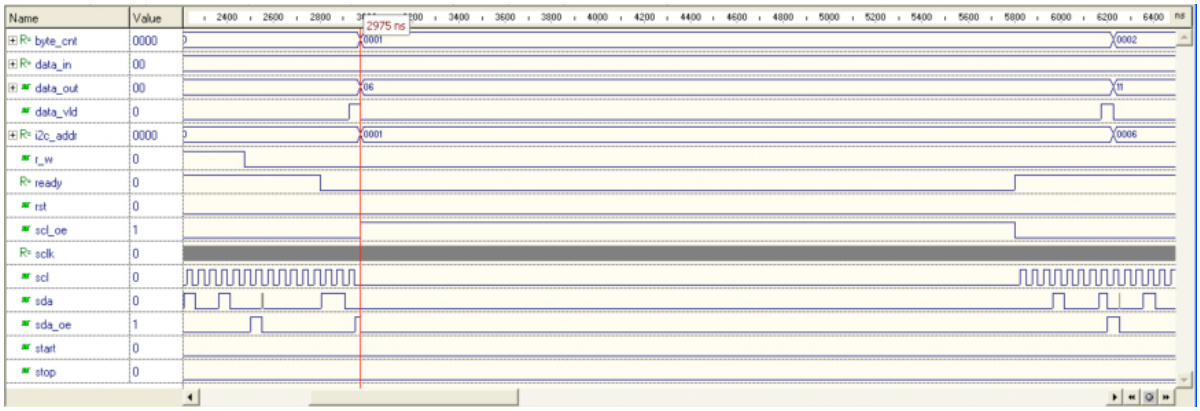
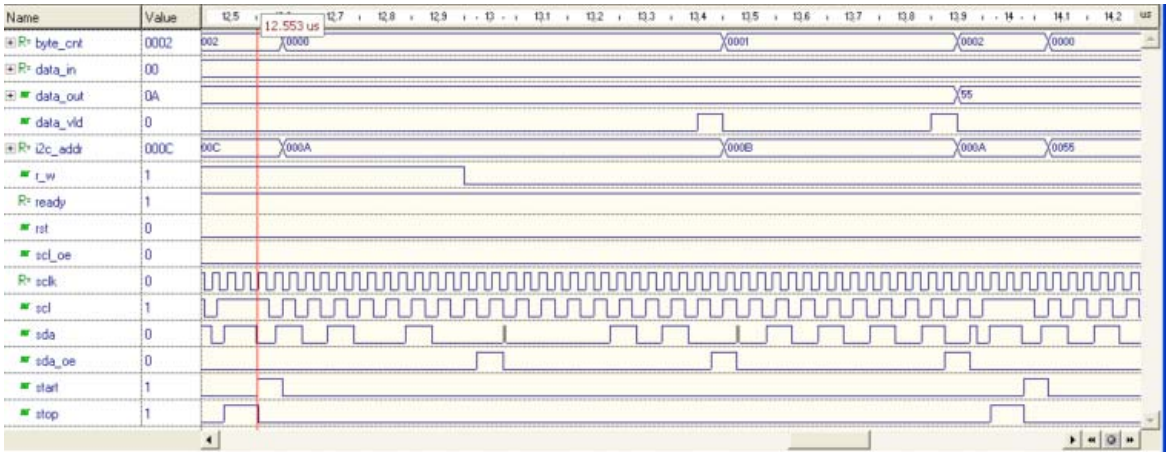


Figure 7. Random Write at Memory Address h0A, with Data h55



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 2. Performance and Resource Utilization

Device Family	Language	Speed Grade	Utilization	f_{MAX} (MHz)	I/Os	Architecture Resources
ECP5™ [6]	Verilog-LSE	–6	62 LUTs	>15	26	N/A
	Verilog-Syn	–6	71 LUTs	>15	26	N/A
	VHDL-LSE	–6	61 LUTs	>15	26	N/A
	VHDL-Syn	–6	87 LUTs	>15	26	N/A
LatticeECP3™ [1]	Verilog	–6	81 LUTs	>15	26	N/A
	VHDL	–6	72 LUTs	>15	26	N/A
LatticeXP2™ [2]	Verilog	–5	72 LUTs	>15	26	N/A
	VHDL	–5	86 LUTs	>15	26	N/A
MachXO™ [3]	Verilog-LSE	–3	60 LUTs	>15	26	N/A
	Verilog-Syn	–3	57 LUTs	>15	26	N/A
	VHDL-LSE	–3	61 LUTs	>15	26	N/A
	VHDL-Syn	–3	57 LUTs	>15	26	N/A
ispMACH® 4000ZE[4]	Verilog	–5 (ns)	48 Macrocells	>15	26	N/A
	VHDL	–5 (ns)	48 Macrocells	>15	26	N/A
Platform Manager[5]	Verilog-LSE	–3	57 LUTs	>15	26	N/A
	Verilog-Syn	–3	57 LUTs	>15	26	N/A
	VHDL-LSE	–3	61 LUTs	>15	26	N/A
	VHDL-Syn	–3	57 LUTs	>15	26	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 LFXP2-5E-5M132C with Lattice Diamond 3.3 design software.
3. Performance and utilization characteristics are generated using LCMXO256C-3T100C with Lattice Diamond 3.3 design software with LSE (Lattice Synthesis Engine) and Synplify Pro®.
4. Performance and utilization characteristics are generated using LC4128ZE-5TN100C with Lattice ispLEVER® Classic 1.4 software.
5. Performance and utilization characteristics are generated using LPTM10-12107-3FTG208CES with Lattice Diamond 3.3 design software with LSE Synplify Pro.
6. Performance and utilization characteristics are generated using LFE5U-45F-6MG285C with Lattice Diamond 3.3 design software with LSE and Synplify Pro.

Technical Support Assistance

e-mail: techsupport@latticesemi.com

Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
December 2014	1.6	Updated Table 2, Performance and Resource Utilization.
		— Added support for ECP5 device family.
		— Updated Utilization values.
		— Updated to support Lattice Diamond 3.3 design software.
		— Added Synplify Pro support.
March 2014	01.5	Updated Table 2, Performance and Resource Utilization.
		— Added support for ECP5 device family.
		— Added support for Lattice Diamond 3.1 design software.
		Updated corporate logo.
		Updated Technical Support Assistance information.
April 2011	01.4	Added support for the LatticeECP3 device family.
		Updated for Lattice Diamond 1.2 software support.
December 2010	01.3	Added support for Platform Manager device family.
August 2010	01.2	Updated for Lattice Diamond software support.
		Added Common I²C Operation Sequence section.
		Updated waveform screen shots.
January 2010	01.1	Added VHDL support.
		Added support for the LatticeXP2 device family.
July 2009	01.0	Initial release.