Arbitration and Switching Between Bus Masters

Reference Design

FPGA-RD-02104-1.2

December 2019

Disclaimers

Contents

Figures

Tables

1. Introduction

Since the development of the system bus that allows multiple devices to communicate with one another through a common channel, bus arbitration has been a critical component of system designs. Devices capable of controlling the bus are called the “masters” of the bus. Bus arbitration is a way to determine which master is allowed access to a bus and when. Its mechanism often grants higher priority to critical devices on the bus, such as a processor, and assigns lower-priority devices a longer waiting time. In addition to arbitration, bus switching is necessary when redundancy is required in the system. Switching between communication channels can protect a system from disruption.

This reference design provides a mode of connection and arbitration between multiple bus masters. While an I^2C bus is used in this design, it is a generic implementation and the algorithm could be applied to any communication protocols. The I^2C bus is chosen for its simple two-wire connection that reduces the board design complexity.

2. Features

- Multiple master arbitration, up to eight masters
- Supports up to eight slave devices
- 1:N switching between masters and slaves
- I^2C compatible master and slave devices

Figure 2.1. is an example application of this reference design.

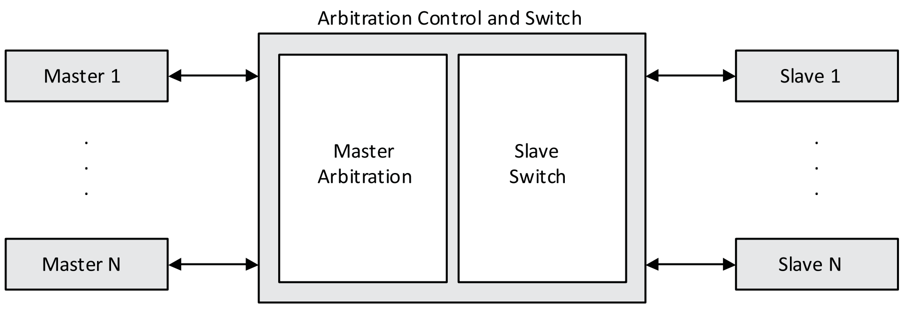

Figure 2.1. Bus Arbitration Application Example

The master devices and slave devices can access the design independently. Once the master arbitration is done, the slave switch is used to select a slave device as requested by the master device.

3. Functional Description

By default, this design connects two master devices (master 1 and master 2) and eight slave devices. The design consists of a master arbitration block and a slave switch block, as shown in Figure 3.1.

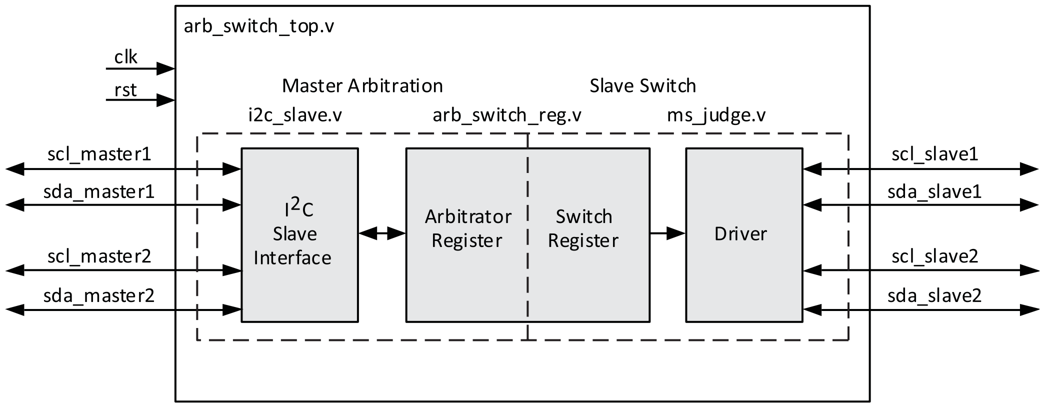

Figure 3.1. Arbitration Control Block Diagram

Table 3.1. Bus Arbitration I/O Interface Descriptions

Signal Name	Signal Direction	Active State	Description
System Interface			
clk	Input	N/A	System clock
rst	Input	High	Reset signal
I^{2}C Master Interface			
scl_master1	Bi-directional	High, Low	Master 1 SCL signal
sda_master1	Bi-directional	High, Low	Master 1 SDA signal
scl_master2	Bi-directional	High, Low	Master 2 SCL signal
sda_master2	Bi-directional	High, Low	Master 2 SDA signal
I^{2}C Slave Interface			
scl_slave(n) (n=1,2….,8)	Bi-directional	High, Low	Slave SCL signal
sda_slave(n) (n=1,2….,8)	Bi-directional	High, Low	Slave SDA signal

3.1. Master Arbitration

Master arbitration determines which master device controls the bus. This work is done by the register arbitrator_control which is defined in the module arb_switch_reg.

The address of this register is 0x00 and the width is 8 bits. Bit descriptions are listed in Table 3.2.

Table 3.2. arbitrator_control Register Bit Descriptions

Bit 7 to 0	Reset Value	Bit Description	Active	Access
0	1	No control from master 1 Master 1 controls the I^{2}C bus Default is master 1	0 = Inactive 1 = Active	Read/Write
1	0	No control from master 2 Master 2 controls the I^{2}C bus Default is master 1	0 = Inactive 1 = Active	Read/Write

Note: Other bits of this register are not defined. The value for these is 0.

When a bit is set to 1 in this register, the corresponding master will control the bus. The least 2 bits in this register control the bus arbitration. Bit 0 corresponds to master 1, and bit 1 corresponds to master 2. When either bit is set to 1, the corresponding master will control the bus.

This register can be read by all masters at any time. If the value of the register is 0x01, master 1 controls the bus. Similarly, a value of 0x02 indicates that master 2 controls the bus. The default value 0x00 indicates that the bus is not controlled by any master.

The register can be written by all masters at any time. If master 2 requests ownership of the bus, it should write the value 0x02 to the register to take control of the bus. In the same manner, master 1 can write the value 0x01 to the register to take control of the bus. Writing any value other than 0x01 for master 1or 0x02 for master 2 will result in a 0x00 value in the register and neither master can control the bus. If master 1 and master 2 write to this register at the same time, master 1 has the higher priority.

Although a master can have ownership of the bus by writing the appropriate value to the arbitrator_control register at any time, this will interrupt the ongoing communication between the current master and its slave device. For example, when master 1 has control of the bus and is communicating with a slave device, writing a value of 0x02 to the arbitrator_control register by master 2 will result in an interruption of master 1's communication with the slave device. This allows quick switching from one master to another if there is the potential for failure in the current link. If such behavior is not desired, the master should read the arbitrator_control register periodically before writing to the register. The writing of the register is carried out only if the value of the register is 0x00. After a master finishes a bus transfer, it should write an invalid value to the arbitrator_control register to make the value of this register become 0x00.

3.2. How to Read From and Write To a Register

The masters access the registers defined in the module arb_switch_reg through the module i2c_slave. The module i2c_slave is used to implement an I^{2}C slave which can connect to the I^{2}C master directly. It receives the serial data from the I^{2}C master and converts these serial data to parallel to write to the registers. Conversely, it also reads the parallel data from registers and converts these parallel data to serial data to send to the I^{2}C master.

The I^{2}C slave module is taken from Lattice reference design RD1054, I^{2}C Slave/Peripheral. It is compatible with the I^{2}C bus specification and supports most of the basic functionalities such as 7-bit addressing, random read/write, and sequential read/write. As an I^{2}C slave, the module i2c_slave has an address. This address value is hard-coded as 0x52 for this design. The module i2c_slave implements the data transfer between the I^{2}C master and the register. The module i2c_slave transmits or receives data to/from the I^{2}C master based on the I^{2}C bus protocol. Figure 3.2. shows the process of the I^{2}C master writing data to the arbitrator_control register. Figure 3.3. shows the I^{2}C master reading data from the arbitrator_control register.

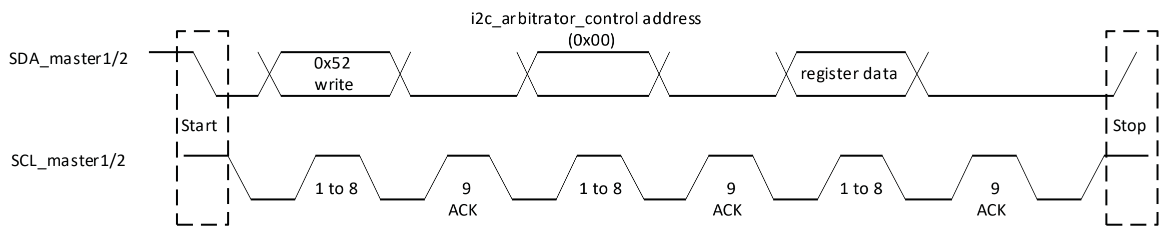

Figure 3.2. I^{2}C Master Writes Data to the arbitrator_control Register

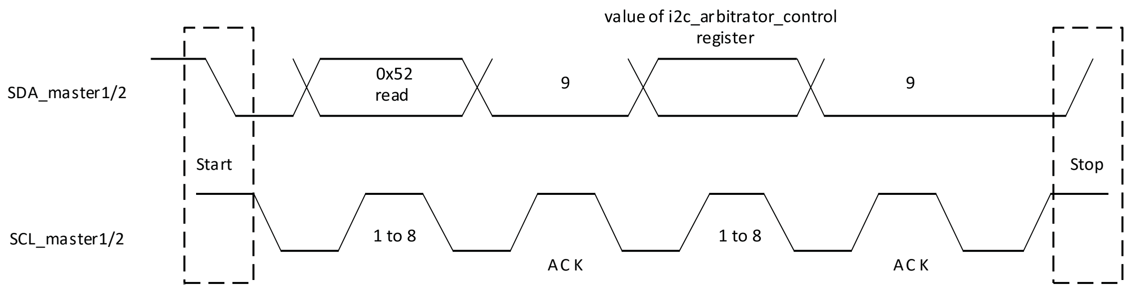

Figure 3.3. I^{2}C Master Reads Data from the arbitrator_control Register

3.3. How the Slave Switch Works

The slave switch determines which slave device can be directly connected to the selected master. This work is done by the register switch_control which is defined in the module arb_switch_reg. The address of this register is 0x01 and the width is 8 bits. The bits are described in Table 3.3.

Table 3.3. switch_control Register Bit Descriptions

Bit	Reset Value	Description	Active	Access
0	0	Enables I^{2}C bus access to I^{2}C slave 1	0 = Inactive 1 = Active	Read/Write
1	0	Enable I^{2}C bus access to I^{2}C slave 2	0 = Inactive 1 = Active	Read/Write
2	0	Enable I^{2}C bus access to I^{2}C slave 3	0 = Inactive 1 = Active	Read/Write
3	0	Enable I^{2}C bus access to I^{2}C slave 4	0 = Inactive 1 = Active	Read/Write
4	0	Enable I^{2}C bus access to I2C slave 5	0 = Inactive 1 = Active	Read/Write
5	0	Enable I^{2}C bus access to I^{2}C slave 6	0 = Inactive 1 = Active	Read/Write
6	0	Enable I^{2}C bus access to I^{2}C slave 7	0 = Inactive 1 = Active	Read/Write
7	0	Enable I2^{2}C bus access to I^{2}C slave 8	0 = Inactive 1 = Active	Read/Write

This register provides point-to-point access for the master to the slave. When the bit is set to 1 in this register, the corresponding slave can be selected to communicate with the master.

This register can be read from and written to by all masters at any time. Even if the master does not control the bus, it is capable of reading from and writing to the register. Only the master that has ownership of the bus can have point-to-point access to a slave to write appropriate values to the register. If a master without control of the bus writes an appropriate value to this register, it is possible to interrupt the communication between the current master and its slave. Therefore, before the master writes data to this register, it should have ownership of the bus. If a master writes multiple 1's into the register, the corresponding salves are selected to communicate with the master. The designer can choose this function by application. Figure 3.4. shows the master writing data to the switch_control register. Figure 3.5. shows the master reading data from the switch_control register.

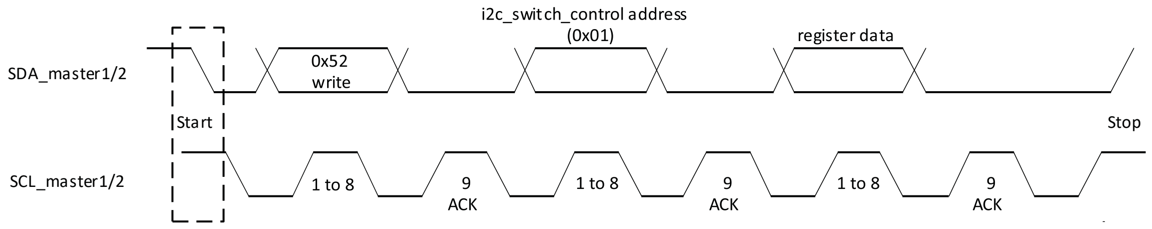

Figure 3.4. I^{2}C Master Writing Data to the switch_control Register

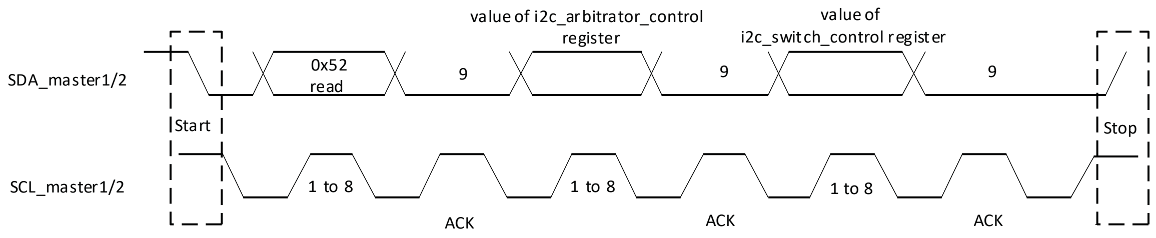

Figure 3.5. I^{2}C Master Reading Data from the switch_control Register

3.4. How the Master Accesses the Slave

1. The master reads the arbitrator_control register until the value of the register is 0x00.
2. The master writes the appropriate data to the arbitrator_control register to control the bus.
3. The master writes the appropriate data to the switch _control register to select the slave.
4. The master accesses the slave.
5. When the master has accessed the slave, it writes data to the arbitrator_control register to relinquish control of the bus.

4. Test Bench Description

The test bench for this design shows how the I^{2}C master accesses the I^{2}C slave.

After resetting master 1 and master 2, read the registers arbitrator_control and switch _control.

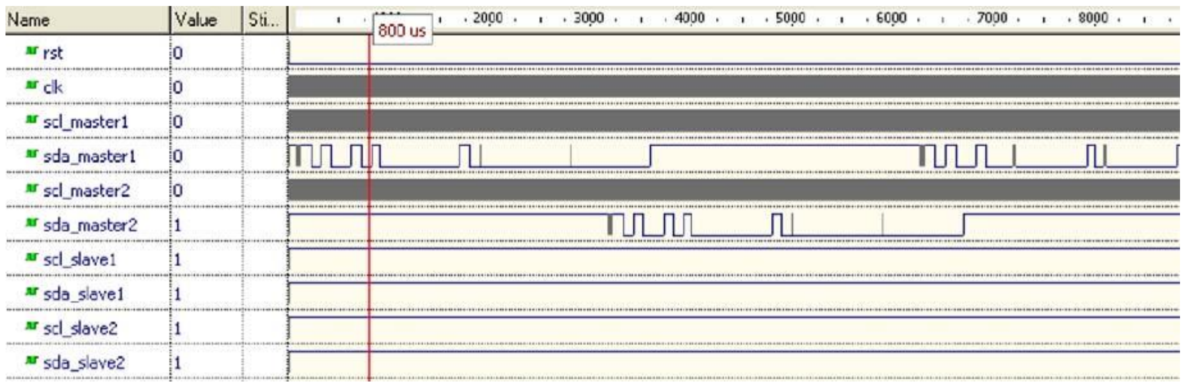

Figure 4.1. Master 1 and Master 2 Read the Register

The value of the arbitrator_control register is 0x01, indicating that master 1 has ownership of the I^{2}C bus. Master 1 writes the data 0x01 to the switch _control register to access slave 1.

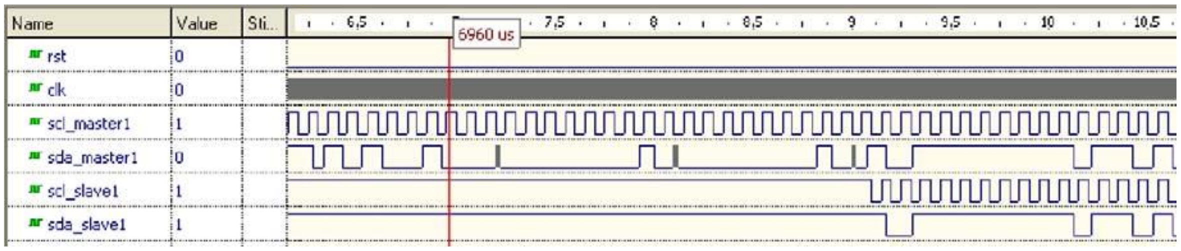

Figure 4.2. Master 1 Accesses I^{2}C Slave 1

When master 1 has accessed the I^{2}C slave 1, master 2 writes data 0x20 to the arbitrator_control register to control the I^{2}C bus. It then writes the data 0x02 to the switch _control register to access I^{2}C slave 2.

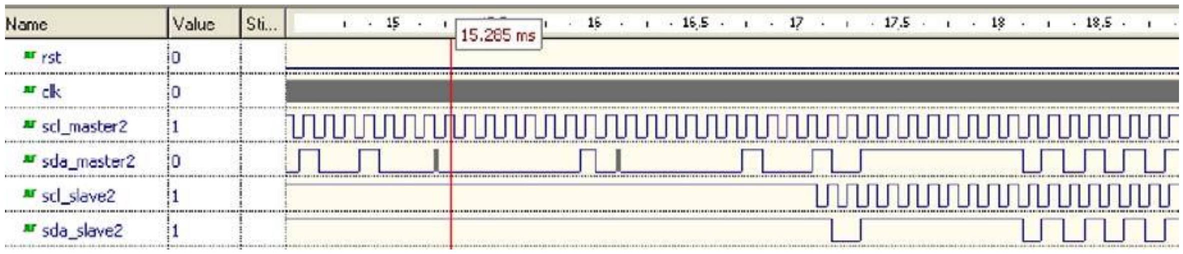

Figure 4.3. Master 2 Accesses I^{2}C Slave 2

5. Implementation

This design is implemented in Verilog/VHDL language and the Lattice ispLEVER® design tool is used for implementation.

Table 5.1. Performance and Resource Utilization

Device Family	Language	Speed Grade	I/O	f_{MAX} (MHz)	Utilization	Architectural Resources
MachXO™ [1]	Verilog	-5	22	>50	245 LUTs	N/A
	VHDL	-5	22	>50	247 LUTs	N/A
ispMACH® 4000ZE[2]	Verilog	-5 (ns)	22	>50	206 Macrocells	N/A
	VHDL	-5 (ns)	22	>50	209 Macrocells	N/A
LatticeXP2™ [3]	Verilog	-5	22	>50	314 LUTs	N/A
	VHDL	-5	22	>50	334 LUTs	N/A

Notes:

1. Performance and utilization characteristics are generated using LCMXO2280C-5T100C with Lattice ispLEVER 8.0 software. When using this design in a different device, density, speed, or grade, performance and utilization may vary.
2. Performance and utilization characteristics are generated using LC4256ZE-5TN100C with Lattice ispLEVER Classic 1.3 software. When using this design in a different device, density, speed, or grade, performance and utilization may vary.
3. Performance and utilization characteristics are generated using LFXP2-5E-5M132C with Lattice ispLEVER 8.0 software. When using this design in a different device, density, speed, or grade, performance and utilization may vary.

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

Revision History

Revision 1.2, December 2019

Section	Change Summary
All	• Changed document number from RD1067 to FPGA-RD-02104. • Updated document template.
Disclaimers	Added this section.

Revision 1.1, February 2010

Section	Change Summary
Implementation	• Added support for LatticeXP2 device family. • Added VHDL support for all device families.

Revision 1.0, January 2010

Section	Change Summary
All	Initial release.