OpenCores.Org

i2cSlave Specification

Author: Steve Fielding
sfielding@base2designs.com

Rev. 1.0
November 7, 2008

Revision History

Rev.	Date	Author	Description
1.0	11/07/08	Sfielding	Created

Contents

1 Introduction

i2cSlave is a minimalist I2C slave IP core that provides the basic framework for the implementation of custom I2C slave devices. The core provides a means to read and write up to 256 8-byte registers. These registers can be connected to the users custom logic, thus implementing a simple control and status interface.

2

Architecture

3 Operation

The core has up 256 registers that can be accessed via I2C. I2C write operations are used to set the register address pointer, and write the register data. I2C reads are used to read the register data. Successive data reads or writes result in data being read or written from incremental register addresses. There is no limit on how much data can be read or written in a single access, but the internal register address pointer will wrap round to 0 once it reaches 255. Note that the address pointer is not initialized at reset, and the address pointer must be set via I2C.

Operation is explained through the use of examples. Examples assume 4 R/W registers at address 0x0, and 4 read only registers at address 0x4, with contents = 0x12345678

Set register address pointer = 0x00:

Byte No.	Data	R/W	Description	Start/Stop	Ack/Nak
1	0x78	W	Device address WR	STA	
2	0x00	W	Set register address = 0x00	STO	

Write 4 bytes of data starting at register address 0x00:

Byte No.	Data	R/W	Description	Start/Stop	Ack/Nak
1	0x78	W	Device address WR	STA	
2	0x00	W	Set register address = 0x00		
3	0x89	W	Write reg[0x00]		
4	0xab	W	Write reg[0x01]		
5	0xcd	W	Write reg[0x02]		
6	0xef	W	Write reg[0x03]	STO	

Read 4 bytes of data starting at register address 0x00:

Byte No.	Data	R/W	Description	Start/Stop	Ack/Nak
1	0x78	W	Device address WR	STA	

2	0x00	W	Set register address = 0x00	STO	
3	0x79	W	Device address RD	STA	
4	0x89	R	Read reg[0x00]		ACK
5	0xab	R	Read reg[0x01]		ACK
6	0xcd	R	Read reg[0x02]		ACK
7	0xef	R	Read reg[0x03]		NAK

Read 4 bytes of data starting at register address 0x04:

Byte No.	Data	R/W	Description	Start/Stop	Ack/Nak
1	0x78	W	Device address WR	STA	
2	0x04	W	Set register address = 0x04	STO	
3	0x79	W	Device address RD	STA	
4	0x12	R	Read reg[0x04]		ACK
5	0x34	R	Read reg[0x05]		ACK
6	0x56	R	Read reg[0x06]		ACK
7	0x78	R	Read reg[0x07]		NAK

Modify the existing files

You will need to modify i2cSlave to suit your individual application. Specifically you will need to modify;

i2cSlave_define.v

Change I2C_ADDRESS to your I2C device address.

Change CLK_FREQ to match your system clock frequency.

registerInterface.v

Modify the input/output ports and the read and write processes to implement your own register interface.

i2cSlave.v

Modify the input/output ports and the instantiation of registerInterface to connect the modified registerInterface ports to the i2cSlave ports. The tri-state buffer is included here for convenience, but you may wish to remove it, and implement the tri-state buffer in your top level module. Note that only sdaIn and sdaOut are defined. If you wish for a more conventional tri-state interface, you can implement the following;

assign sda_i = sdaIn;

```
assign sda_o = 1'b0;
assign sda_oe_n = sdaOut;
```

Add your own custom logic

Now you can include i2cSlave in your own top level module that connects the registers to your own custom logic, and connects sda and scl to your top level ports.

4

Clocks

Name	Source	Rates (MHz)			Remarks	Description
		Max	Min	Res		
clk	Input Pad	Limited by hardware	TBD. Only tested at 48MHz	-	Duty cycle 50/50.	System clock.

Table 1: List of clocks

5

IO Ports

Port	Width	Direction	Description
clk	1	input	Clock. If you change this clock from 48MHz you may need to alter some constants in the i2cSlave_define.v file
rst	1	input	1 = reset. Synchronous to clk. Resets all logic.
sda	8	inout	I2C SDA
scl	8	inout	I2C SCL
MyReg[3:0]	8	output	I2C accessible output registers. Modify to implement your own custom outputs
MyReg[7:4]	8	input	I2C accessible input registers. Modify to implement your own custom inputs

Table 2: List of IO ports

6

Resource Utilization

Target Device	Logic Cells / Macrocells	Memory bits
EPM7256 (256 macrocell CPLD)	143 macrocells	0
EP2C20	218 logic cells	0

Table 3 Resource utilization for Altera CycloneEP2C20, and EPM7256