Key Design Features

- Technology independent IP Core for FPGA and ASIC
- Supplied as human readable VHDL (or Verilog) source code
- Phillips® I2C-bus compliant
- Intuitive command interface featuring a simple valid-ready handshake protocol
- Master instruction FIFO permits buffering of sequential I2C requests
- Slave read FIFO permits buffering of slave read data and slave responses
- Fully configurable clocking allows Standard (100 kHz), Fast (400 kHz) custom data rates exceeding 20 MHz
- Configurable setup and hold times on the SDA line relative to the SCL line
- Supports standard 8-bit and 10-bit addressing modes

Applications

- Driving I2C slave devices
- Inter-chip board-level communications
- Standard 2-wire comms between a wide range of I2C peripherals, micro-controllers and COTs ICs

Pin-out Description

Pin name	I/O	Description	Active state
clk	in	Synchronous clock	rising edge
reset	in	Asynchronous reset	low
mast_inst[3:0]	in	Master instruction	data
mast_data[7:0]	in	Master I2C data to be serialized	data
mast_val	in	Master instruction valid	high
mast_rdy	out	Master instruction ready handshake	high
scl	i/o	I2C bi-directional SCL clock pin	As per I2C specification
sda	i/o	I2C bi-directional SDA data pin	As per I2C specification
slv_inst[3:0]	out	Slave instruction	data
slv_data[7:0]	out	Slave I2C data received from slave device	data
slv_val	out	Slave data valid	high
slv_rdy	in	Slave data ready handshake	high

Block Diagram

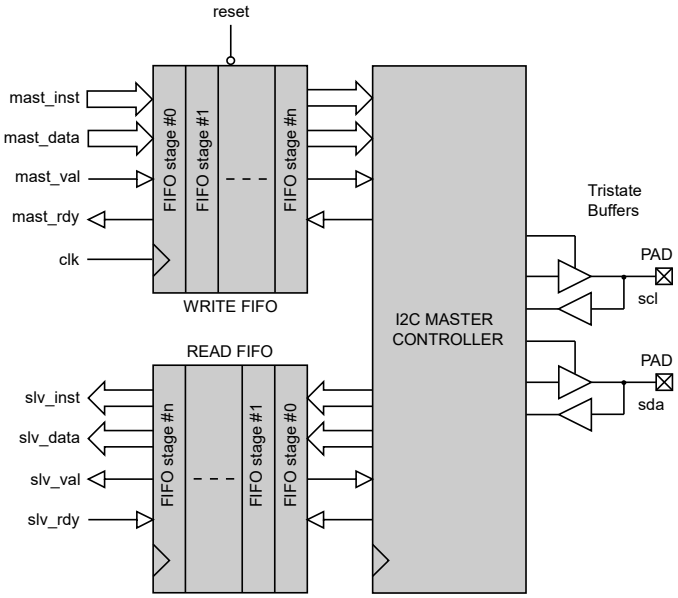


Figure 1: I2C Master serial controller architecture

Generic Parameters

Generic name	Description	Type	Valid range
t_period	SCL clock period (as number of system clock cycles)	integer	≥ 4
t_data_su	SDA setup/hold time (as number of system clock cycles)	integer	≥ 2
wfifo_depth	Master instruction write FIFO depth	integer	≥ 2
wfifo_depth_log2	Master instruction write FIFO depth log2	integer	log2 (wfifo_depth)
rfifo_depth	Slave read data FIFO depth	integer	≥ 2
rfifo_depth_log2	Slave read data FIFO depth log2	integer	log2 (rfifo_depth)

General Description

The I2C_MASTER IP Core is a Philips® I2C compliant serial interface controller capable of driving a standard two-wire bus in single-master mode. The controller receives data and instructions via the master instruction interface. These instructions are then processed by the controller state-machine in order to generate the appropriate responses on the SCL and SDA lines.

Likewise, any slave responses on the I2C-bus are captured by the controller and de-serialized for presentation at the slave read data port.

The I2C master controller is comprised of three main blocks as described by Figure 1. These blocks are the master instruction write FIFO, the I2C controller core and the slave read data FIFO.

The I/O ports SCL and SDA are connected to bi-directional tristate buffers. Note that when the I2C controller is inactive, both the SCL and SDA lines will be tristate and as such, these pins should be externally pulled up as per the I2C specification.

The SCL clock-period is determined by the the generic parameter: *t_period.* This parameter specifies the SCL period in system clock cycles. As an example, if the system clock '*clk*' is running at 100MHz and an SCL clock frequency of 100kHz is required (I2C standard mode), a value of *t_period* = 1000 should be specified.

In addition, the generic parameter *t_data_su* permits the SDA data-line to be delayed by 'n' system clock cycles relative to the SCL line. In this way, the SDA setup and hold specification can be modified accordingly.

Figure 2 demonstrates how the parameters *t_period* and *t_data_su* effect the output signals on the I2C-bus. By modifying *t_data_su*, the user can ensure a stable data window during the active-high SCL pulse.

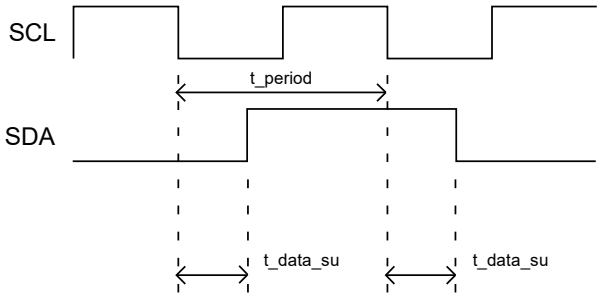


Figure 2: I2C timing specification

Master Write FIFO

Instructions to the I2C master controller are sent via an input FIFO whose depth is determined by the generic parameter *wfifo_depth*. The write FIFO interface operates in accordance with the valid/ready pipeline protocol meaning that instructions and data are written to the FIFO on the rising edge of *clk* when both *mast_val* and *mast_rdy* are high[1]

The write FIFO may be used to 'queue up' a sequence of commands while current commands are being processed on the bus. As soon as the write FIFO becomes full then the FIFO will disable the *mast_rdy* signal signifying that further requests are not possible. Likewise, the *mast_rdy* signal will also be disabled if the slave read-data FIFO becomes full. In both situations, no further commands will be accepted by the I2C controller until the FIFOs have emptied.

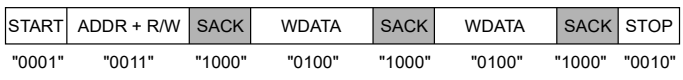
The instructions to the I2C controller are very intuitive and follow the exact sequence of commands that the user wishes to appear on the I2C bus.

The following table outlines the set of commands accepted by the controller via the master write FIFO:

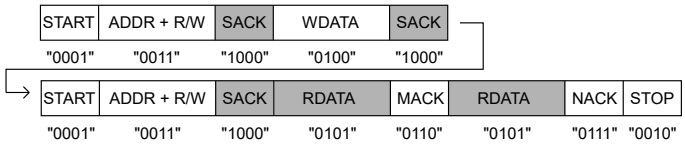
MASTER INSTRUCTION INPUT FORMAT

mast_inst[3:0]	***mast_data[7:0]***	***Description***
“0000”	[7:0] : 'X' Don't care	RESET Reset controller to initial conditions
“0001”	[7:0] : 'X' Don't care	START Issue a I2C start command (SCL high, SDA falling edge)
“0010”	[7:0] : 'X' Don't care	STOP Issue a I2C stop command (SCL high, SDA rising edge)
“0011”	[7:1] : Slave Address [0] : R/W flag	ADDR Write an 8-bit slave address
“0100”	[7:0] : Write data	WDATA Write 8-bit data
“0101”	[7:0] : 'X' Don't care	RDATA Read 8-bit slave data
“0110”	[7:0] : 'X' Don't care	MACK Issue a master ack signal (SDA low, SCL clock pulse)
“0111”	[7:0] : 'X' Don't care	NACK Issue a master no-ack signal (SDA high, SCL clock pulse)
“1000”	[7:0] : 'X' Don't care	SACK Slave ack (SDA tristate, SCL clock pulse)
Other values	[7:0] : 'X' Don't care	NULL Performs no action (other than filling up the FIFO)

As an example, to write two consecutive bytes to a slave device, the following sequence of instructions might be be sent to the controller:



A consecutive two byte read might be performed as:



Note that the exact sequence of instructions required will depend on the functionality of the slave device that is to be addressed. For this reason, there is no restriction in the ordering of instructions.

1 See Zipcores application note: app_note_zc001.pdf for more examples of the valid/ready protocol and it's implementation

I2C Master Controller Core

The master controller is a state-machine that accepts instructions from the write FIFO and generates the appropriate signals on the I2C bus.

Immediately after an asynchronous reset of the core, the state machine starts in the reset state in which both the SCL line and the SDA line are high-impedance (tristate). On receipt of the first valid instruction, the state machine will take control of the bus and drive the SCL/SDA lines in response to the received instructions.

The master controller is also responsible for capturing slave responses on the I2C bus - in particular, the slave ack (or no-ack) and serial slave data bits.

Slave Read FIFO

Assuming that the slave FIFO has capacity, all slave ack responses and slave read data are captured in the slave FIFO. The type of slave response is identified in the 4-bit instruction identified in the table below:

SLAVE INSTRUCTION OUTPUT FORMAT

slv_inst[3:0]	***slv_data[7:0]***	***Description***
“0101”	[7:0] : Slave Data	RDATA
“1000”	[7:0] : “00000000”	SACK (Slave acknowledge)
“1000”	[7:0] : “00000001”	SNACK (Slave no acknowledge)
Other values	[7:0] : “00000000”	NULL (not valid)

Note that the *slv_inst* outputs are identical for a SACK and SNACK. The user must check the LSB of the read data in order to determine how the slave responded to the request on the bus. In the case of a slave no-ack, it is up to the user to decide whether to ignore the response or reissue the desired command.

Note that if the slave FIFO becomes full then slave responses may get lost resulting in subsequent responses becoming out of sync.

Functional Timing

Figure 3 shows a simple series of instructions sent to the the controller. The sequence is: START, ADDR, SACK, STOP. Note that the instruction FIFO is full after the third instruction and *mast_rdy* is de-asserted for one clock cycle. In the following cycle, *mast_rdy* goes high and the final instruction is transferred.

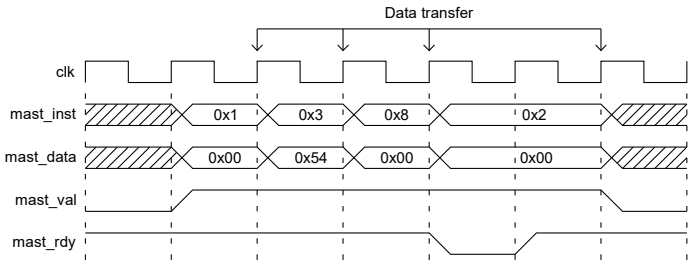


Figure 3: Master instruction interface timing

Figure 4 demonstrates the corresponding I2C bus signals that are generated in response to the previous instructions in Figure 3. The dashed line signifies the point in which the master releases the SDA line.

When the SDA line is released, it is up to the slave device to pull the line low (ack) or high (no-ack) accordingly.

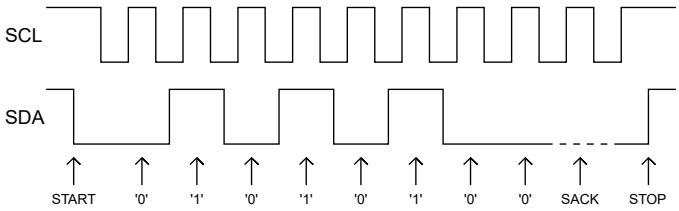


Figure 4: I2C bus signals

Figure 5 gives an example of responses on the slave port. The sequence of responses are: slave no-ack, slave ack, slave read (value 0xBF) and slave read (value 0x7C). Again, the timing diagram shows how the slave FIFO output interface may be stalled by driving *slv_rdy* low.

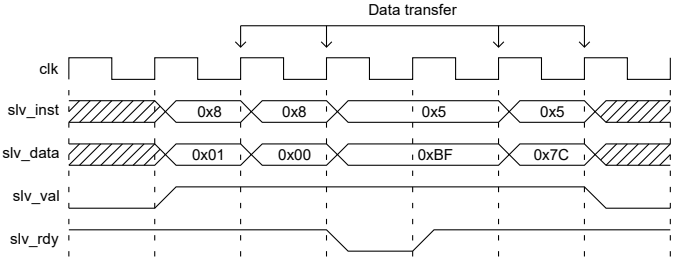


Figure 5: Slave Instruction interface timing

Source File Description

All source files are provided as text files coded in VHDL. Equivalent Verilog versions of the code may be provided on request.

Source file	Description
i2c_mast_stim.txt	Input stimulus text file
i2c_iobuf.vhd	Bi-directional tristate buffer
i2c_delay.vhd	Adds delay to the SDA line
i2c_fifo.vhd	Input/output FIFOs
i2c_master_cont.vhd	Main I2C master controller
i2c_master.vhd	Top-level block
i2c_slave_dummy.vhd	I2C dummy slave device
i2c_master_file_reader.vhd	Reads master instructions from a text file
i2c_master_bench.vhd	Top-level test bench

Functional Testing

An example VHDL test bench is provided for use in a suitable VHDL simulator. The compilation order of the source code is as follows:

1. i2c_iobuf.vhd
2. i2c_delay.vhd
3. i2c_fifo.vhd
4. i2c_master_cont.vhd
5. i2c_mast.vhd
6. i2c_slave_dummy.vhd
7. i2c_master_file_reader.vhd
8. i2c_master_bench.vhd

The testbench instantiates the i2c_master component together with a dummy slave I2C device and a file-reader module that reads the master instructions from a text file.

The input text file is called *i2_master_stim.txt* and should be put in the top-level simulation directory. The format of the input text file is: 'A B CC' where 'A' is either '1' or '0' signifying a valid or invalid instruction, 'B' is the 4-bit *mast_ins*t instruction, and 'CC' is the 8-bit *mast_data*. All values are specified in hexadecimal.

As an example, in order to send the sequence: START, ADDR, SACK, STOP to the controller, where the slave address is 0x54, the text file would be:

```
1 1 00    # start
1 3 54    # write address 0x54
1 8 00    # slave ack
1 2 00    # stop
```

In addition to setting up the input stimulus file with the desired master instructions, the user may also modify the generic parameters on the I2C master component as required. Careful attention must be made to select the correct timing parameters in relation to the system clock frequency in order to conduct a realistic simulation.

In the default set up, the simulation must be run for around 30 ms during which time the file reader module will drive the I2C master with the input instructions. A dummy I2C device will generate random slave data on the I2C bus in response to the master requests.

The simulation generates two text files called: *i2c_master_in.txt* and *i2c_master_out.txt*. These files respectively contain the input and output data captured at the master instruction and slave data ports during the course of the test. The contents of these two files may be compared to verify the operation of the I2C master controller.

Synthesis and Implementation

The files required for synthesis and the design hierarchy is shown below:

- i2c_master.vhd
  - i2c_master_cont.vhd
  - i2c_fifo.vhd
  - i2c_delay.vhd
  - i2c_iobuf.vhd

The VHDL core is designed to be technology independent. However, as a benchmark, synthesis results have been provided for the Xilinx® 7-series FPGAs. Synthesis results for other FPGAs and technologies can be provided on request.

There are no special constraints required for synthesis although reducing the szie of the FIFOs will result in the fastest and most area efficient designs. The IP core is completely technology independent.

Trial synthesis results are shown with the generic parameters set to: t_period = 25, t_data_su = 5, wfifo_depth = 8, wfifo_depth_log2 = 3, rfifo_depth = 8, rfifo_depth_log2 = 3.

Resource usage is specified after place and route of the design.

XILINX® 7-SERIES FPGAS

Resource type	Artix-7	Kintex-7	Virtex-7
Slice register	68	70	70
Slice LUTs	138	181	181
Block RAM	0	0	0
DSP48	0	0	0
Occupied slices	55	56	72
Clock freq. (approx)	300 MHz	350 MHz	400 MHz

Revision History

Revision	*Change description*	*Date*
1.0	Initial revision	01/10/2008
1.1	Added clock-stretching feature	16/02/2010
1.2	Updated synthesis results for Xilinx® 6 series FPGAs	30/05/2012
1.3	Fixed various typos in datasheet. Updated synthesis results	28/03/2014
1.4	Now supports 4:1 system to SCL ratios. Optimized design for speed. Removed clock-stretching option. Updated synthesis results for Xilinx® 7 series FPGAs.	14/06/2015
1.5	Minor source code optimizations	11/04/2016

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

Zipcores:
SKU23