

Key Design Features

- Synthesizable, technology independent VHDL Core
- Phillips® I2C-bus compliant
- Intuitive command interface featuring a simple valid-ready handshake protocol
- Master instruction FIFO permits queuing of sequential I2C requests
- Slave read-data FIFO permits queuing of slave read data
- Fully configurable clocking allows Standard (100kHz), Fast (400kHz) and user-defined data rates up to any desired frequency¹
- Configurable setup and hold times on the SDA line
- Supports standard 8 and 10-bit addressing modes
- Supports slave clock-stretching

Applications

- Driving I2C slave devices
- Inter-chip board-level communications

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 Philips® I2C specification
sda	i/o	I2C bi-directional SDA data pin	As per Philips® 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

¹ Generally, the maximum attainable frequency will be determined by the physical characteristics of the bus and the choice of output buffer

Block Diagram

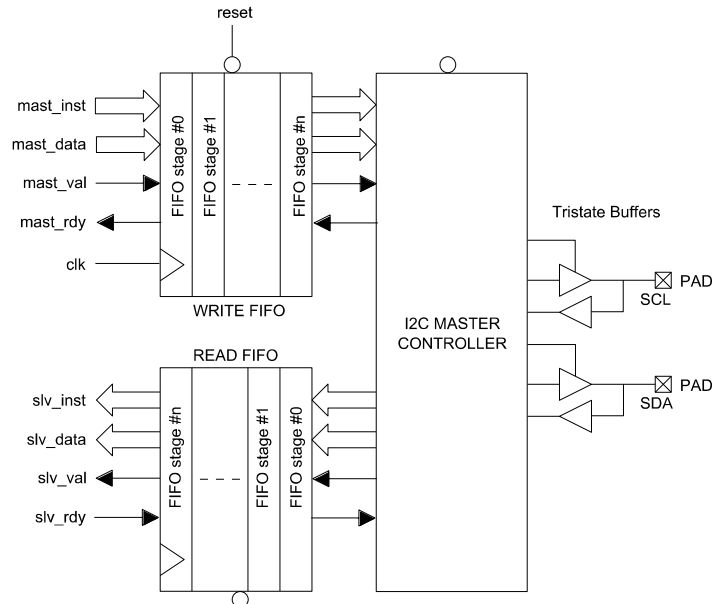


Figure 1: I2C Master Serial Controller Architecture

Generic Parameters

Generic name	Description	Type	Valid range
t_period	SCL clock period (as number of system clock cycles)	integer	≥ 10
t_data_su	SDA setup time (as number of system clock cycles)	integer	≥ 2
cs_enable	Enables slave clock-stretching functionality	boolean	True/False
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

I2C_MASTER 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 core 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 output 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 clk 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.

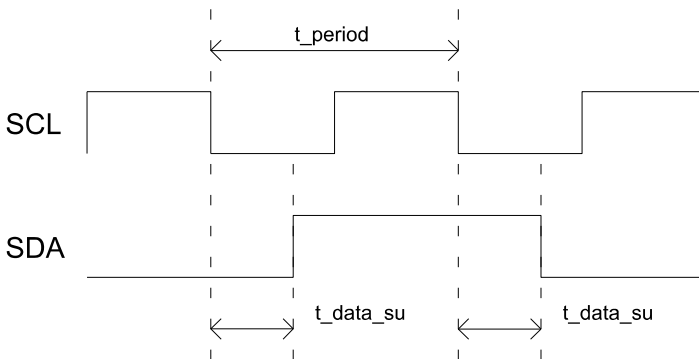


Figure 2: I2C Timing Specification

Slave Clock-stretching Support

Clock-stretching may be enabled by setting the generic parameter cs_enable to *true*. Clock-stretching allows the I2C Master to interface to slower slave devices by allowing the slave to pull the clock line low and delay the clock pulse. If clock-stretching is enabled, then the output enable of the tristate buffer is disabled for the 'high' part of the SCL clock cycle allowing the external pull-up resistor to pull the SCL line high. The master controller state machine performs an extra check to ensure that the SCL line is high before continuing the bus transfer.

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 $mast_val$ is high and $mast_rdy$ is high²

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

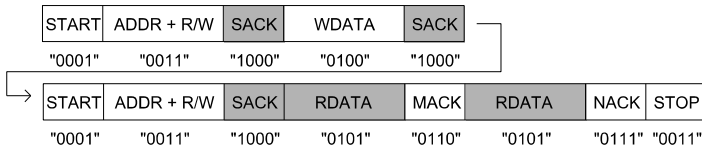
$mast_inst[3:0]$	$mast_data[7:0]$	Description
"0000"	[7:0] : 'X' Don't care	RESET Reset controller to initial conditions and set I2C pins to tristate
"0001"	[7:0] : 'X' Don't care	START Issue a I2C start command (SCL high, SDA falling edge)
"0010"	[7:0] : 'X' Don't care	STOP Issue a I2C stop command (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 sent to the controller:

START	ADDR + R/W	SACK	WDATA	SACK	WDATA	SACK	STOP
"0001"	"0011"	"1000"	"0100"	"1000"	"0100"	"1000"	"0011"

A consecutive two byte read might be performed as:

² See ZIPcores application note: app_note_zc001.pdf for more examples of the valid/ready protocol and it's implementation



Of course, 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 that may be sent to the I2C master controller. This is useful, for example, if 10-bit slave address is required, where some controllers may not allow two consecutive address commands to be sent in series.

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. As each instruction is processed by the controller core, the results are written to the Slave read FIFO.

Slave Read FIFO

For every master instruction received by the controller, the controller also sends a copy of the original instruction plus the slave read data (if applicable) to the Slave read FIFO. In the case that the originating instruction was not a slave read (e.g. a START or ADDR instruction) then the slave read data contains a copy of the original master data. The following table gives a brief summary of the instruction format:

SLAVE INSTRUCTION OUTPUT FORMAT

<i>slv_inst</i> [3:0]	<i>slv_data</i> [7:0]	Description
"0000"	[7:0] : (same as original <i>mast_data</i>)	RESET
"0001"	[7:0] : (same as original <i>mast_data</i>)	START
"0010"	[7:0] : (same as original <i>mast_data</i>)	STOP
"0011"	[7:1] : Slave Address [0] : R/W flag	ADDR
"0100"	[7:0] : (same as original <i>mast_data</i>)	WDATA
"0101"	[7:0] : Slave Data	RDATA
"0110"	[7:0] : (same as original <i>mast_data</i>)	MACK
"0111"	[7:0] : (same as original <i>mast_data</i>)	NACK
"1000"	[7:0] : "00000000"	SACK
"1001"	[7:0] : "00000000"	SNACK
Other values	[7:0] : (same as original <i>mast_data</i>)	NULL

Note that the *slv_inst* outputs are identical to the *mast_inst* inputs with the exception of 'SNACK'. This is a Slave no-ack signal and indicates that the slave responded with a no-ack at the end of a particular 8-bit transfer on the I2C 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.

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 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.

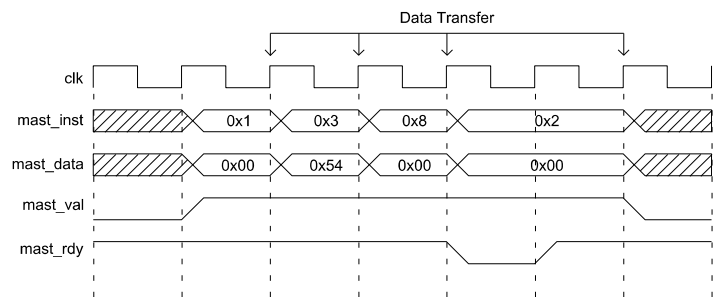


Figure 3: Master Instruction interface timing

Figure 4 demonstrates the corresponding I2C bus signals that are generated in response to the previous instructions in Figure3. The dashed line signifies the point in which the master releases the SDA line. It is then up to the slave device to pull the line low (ack) or high (no-ack) accordingly.

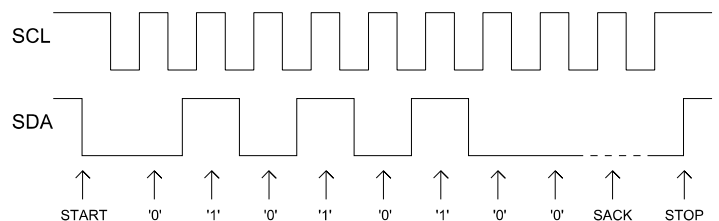


Figure 4: I2C bus signalling

Finally, Figure 5 demonstrates the series of responses on the Slave port for the same set of instructions. The sequence of instructions is the same with the exception that in this particular instance, the Slave generated a SNACK (0x9) indicating that the address transfer failed.

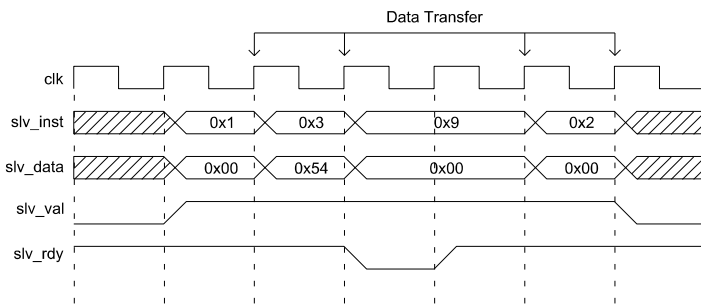


Figure 5: Slave Instruction interface timing

Source File Description

All source files are provided as text files coded in VHDL. The following table gives a brief description of each file.

Source file	Description
i2c_mast_stim.txt	Input stimulus text file
i2c_inbuf.vhd	Input buffer
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_cont_cs.vhd	Main I2C master controller (clock-stretching version)
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_inbuf.vhd
2. i2c_iobuf.vhd
3. i2c_delay.vhd
4. i2c_fifo.vhd
5. i2c_master_cont.vhd
6. i2c_master_cont_cs.vhd
7. i2c_mast.vhd
8. i2c_slave_dummy.vhd
9. i2c_master_file_reader.vhd
10. i2c_master_bench.vhd

The VHDL test bench instantiates the i2c_master component together with a dummy slave I2C device and an 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 current top-level VHDL 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 instruction *mast_inst*, and 'CC' is the 8-bit data. *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 write address is 0x54, the text file would read:

```
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 responses on the I2C bus in response to the master requests.

The simulation generates two text files: *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.

(Note that when comparing the input and output files, there will be a mismatch between the Slave ACK and Slave READ responses. This is because the dummy slave device is a very simple model that drives random data on the bus).

Development Board Testing

The I2C Serial Interface Controller was implemented on a Xilinx® 2V3000 FPGA running at a system clock frequency of 65MHz. The controller was then used to drive a series of I2C slave devices including a serial EEPROM (24LC02B), a temperature sensor (MCP9800), a 12-bit ADC (MCP3221), a 10-bit DAC (TC1321) and an 8-bit I/O Expander (MCP23008). The controller was set up for 400kHz (Fast mode) operation with the generic parameters *t_period* = 163 and *t_data_su* = 41.

After testing was performed at 400kHz, further testing was performed to verify correct operation at 100kHz (Standard mode) and at 1.7MHz which was the maximum permissible clock speed allowed by the Slave devices.

Figure 6 below demonstrates the I2C bus signals operating at 400kHz. The top trace is the SCL line and the bottom trace is the SDA line. As the system clock is 65MHz, with *t_period* = 163, we would expect the SCL line to toggle at 65MHz/163 = 398.8kHz.

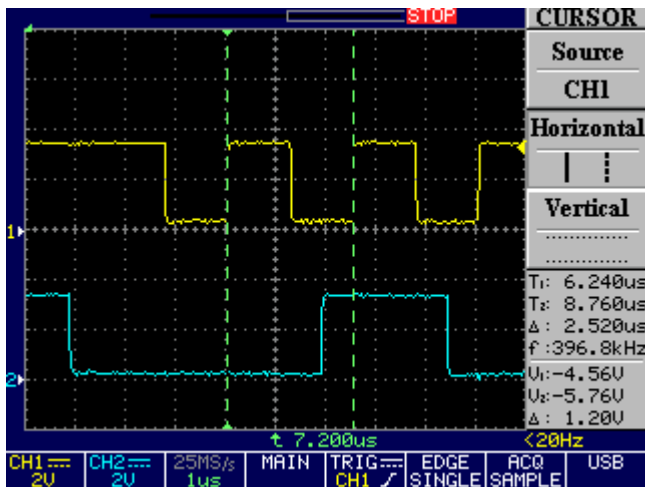


Figure 6: I2C fast-mode SCL-clock detail (400kHz)

Figure 7 shows the measurement of the SDA setup time before the rising-edge of the SCL line. With $t_{data_su} = 41$, the data should change and be stable at about one-quarter cycle before the rising SCL clock edge. An empirical measurement of ~640 ns was measured from the waveform trace.

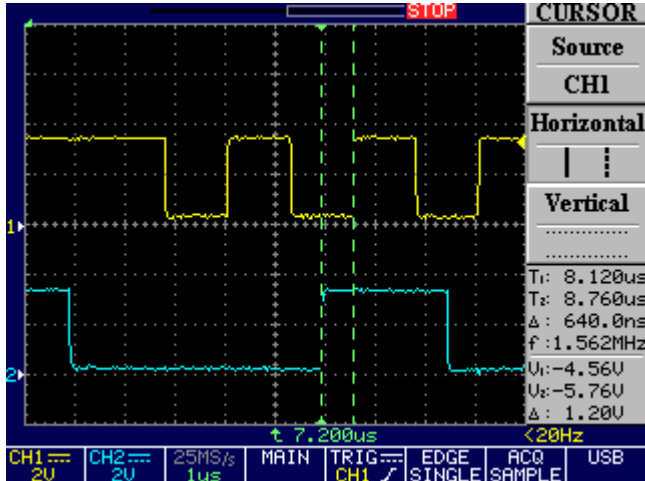


Figure 7: I2C Fast-mode setup-time measurement

Figure 8 demonstrates a series of I2C master commands which comprise of a START, ADDR, SACK and STOP. The Slave write address in this particular example is 0x40. Notice the small 'glitch' during the slave acknowledge cycle. This is perfectly normal and it happens in the low period of SCL as the slave hands back control of the bus to the master.

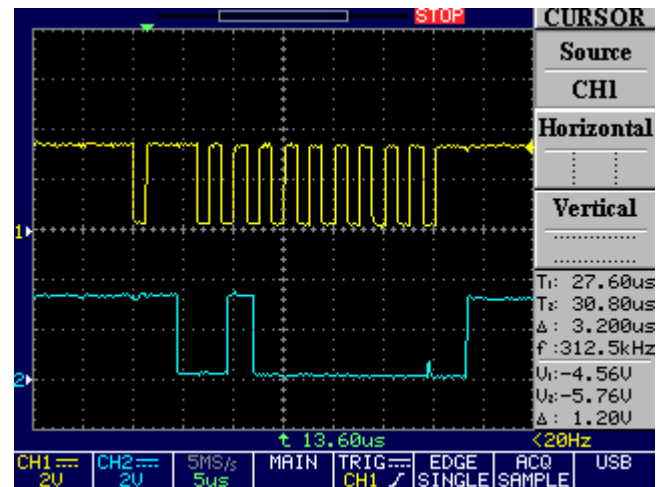


Figure 8: A series of I2C master commands

Finally, figure 9 shows a normal read transfer with clock-stretching enabled. In this case, the Master releases the bus during the 'high' part of the clock cycle. Only when the SCL line has been pulled-up to the logic '1' threshold does the Master take control of the bus once more. Note that in clock-stretching mode, there may be some performance loss due to the slower rise time of the clock.

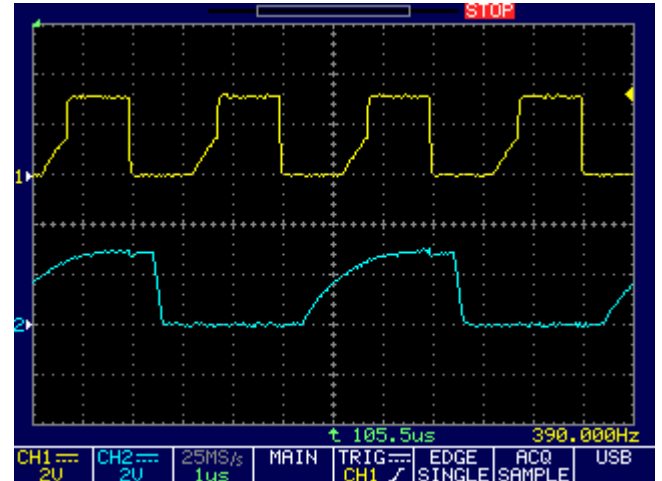


Figure 9: I2C SCL line with Clock-stretching enabled

Synthesis

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

- i2c_master.vhd
 - i2c_master_cont.vhd
 - i2c_master_cont_cs.vhd
 - i2c_fifo.vhd
 - i2c_delay.vhd
 - i2c_jobuf.vhd

The VHDL core is designed to be technology independent. However, as a benchmark, synthesis results have been provided for the Xilinx Virtex 5 and the Altera Stratix III series of FPGA devices. The lowest and highest speed grade devices have been chosen in both cases for comparison.

Note that in order to achieve the fastest and most area efficient designs the size of the FIFOs should be kept to a minimum.

Trial synthesis results are shown with the generic parameters set to: $t_{period} = 100$, $t_{data_su} = 10$, $cs_enable = false$, $wfifo_depth = 8$, $wfifo_depth_log2 = 3$, $rfifo_depth = 8$, $rfifo_depth_log2 = 3$.

Resource usage is specified after Place and Route.

VIRTEX 5

Resource type	Quantity used
Slice register	94
Slice LUT	192
Block RAM	0
DSP48	0
Clock frequency (worst case)	315 MHz
Clock frequency (best case)	348 MHz

STRATIX III

Resource type	Quantity used
Register	160
ALUT	187
Block Memory bit	192
DSP block 18	0
Clock frequency (worse case)	206 MHz
Clock frequency (best case)	260 MHz

Revision History

Revision	Change description	Date
1.0	Initial revision	01/10/2008
1.1	Added clock-stretching feature	16/02/2010