

Key Design Features

- Synthesizable, technology independent VHDL Core
- Phillips® I2C-bus compliant
- Intuitive command interface featuring a simple valid-ready handshake protocol
- Slave instruction FIFO permits buffering of incoming I2C Master requests
- Slave read-data FIFO permits buffering of slave read data
- Compatible with Standard (100kHz), Fast (400kHz) and user-defined data rates up to any desired frequency¹
- Data rate auto-detection allows direct connection to a standard I2C bus without any special configuration.
- Timeout recovery circuit in the case of incomplete, corrupted or non-standard bus behaviour.

Applications

- I2C slave communication
- 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]	out	Master instruction	data
mast_data[7:0]	out	De-serialized I2C Master data	data
mast_val	out	Master instruction valid	high
mast_rdy	in	Master instruction ready handshake	high
scl	in	I2C input SCL clock pin	As per Phillips® I2C specification
sda	i/o	I2C bi-directional SDA data pin	As per Phillips® I2C specification
slv_data[7:0]	in	I2C slave data to be sent to the Master device	data
slv_val	in	Slave data valid	high
slv_rdy	out	Slave data ready handshake	high

Block Diagram

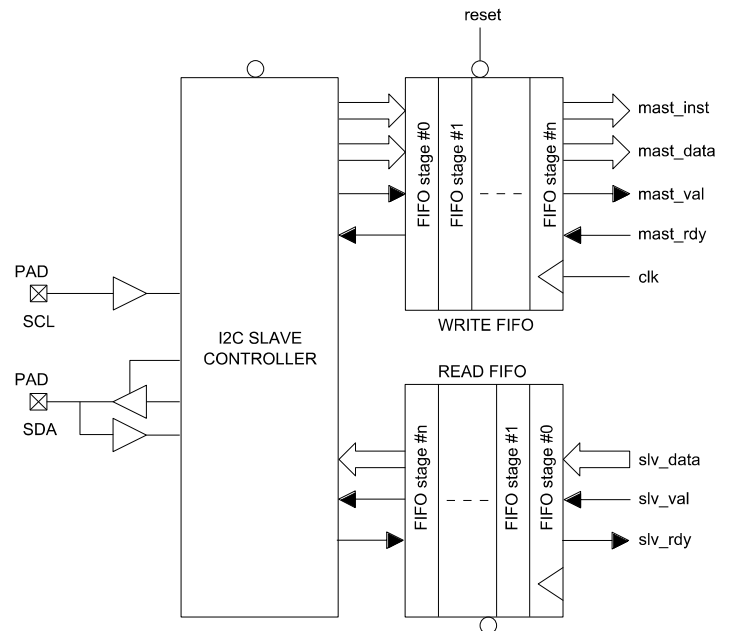


Figure 1: I2C Slave Serial Interface Controller Architecture

Generic Parameters

Generic name	Description	Type	Valid range
slave_address	8-bit slave address of the device on the I2C bus (LSB is don't care)	std_logic vector	"0000000X" to "1111111X"
use_timeout	Enable timeout functionality	Boolean	TRUE/FALSE
timeout	Timeout counter value	integer	≤ 65535
wfifo_depth	Slave instruction FIFO depth	integer	≥ 2
wfifo_depth_log2	Slave instruction 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_SLAVE is a Phillips® I2C compliant Slave interface controller. The controller decodes the SCL and SDA bus signals and converts them into a simple series of instructions and de-serialized 8-bit data. These instructions may then be interpreted by the user application circuit and processed as required.

The SCL port is an input driven by the I2C Master. The SDA port is connected to bi-directional tristate buffer.

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

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 I2C slave controller is comprised of three main blocks as described by Figure 1. These blocks are the I2C Slave Controller core the Master instruction FIFO and the Slave read-data FIFO.

I2C Slave Controller Core

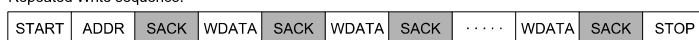
The slave controller core is a state-machine that continually monitors the state of the SCL and SDA lines and generates the appropriate signals on the I2C bus.

To begin a data transfer, the state machine looks for a start command which is defined by a stable SCL high signal with a falling SDA line. On receipt of a start command, the slave will latch the slave address and the r/w flag over the next eight consecutive bits.

If the slave address on the bus corresponds to the generic parameter *slave_address*, then the controller will generate an acknowledge signal and data transfer may commence. If the slave address mismatches, then the controller reverts back to its idle state waiting for the next start condition.

Once the controller has been addressed correctly, the master may continue to send a sequence of writes or reads as required. Writes are terminated with the Master issuing a stop command. Reads are terminated with a non-ack then stop. A change in data direction e.g. a write followed by a read must be accompanied by a repeated start followed by the slave address. The series of commands understood by the slave controller are summarized by Figure 2 below:

Repeated Write sequence:



Repeated Read sequence:



Write then Read:



Read then Write:



Legend:

START Start command
 STOP Stop command
 ADDR 7-bit Slave Address + r/w flag
 WDATA 8-bit Write data
 RDATA 8-bit Read data
 MACK Master Ack
 SACK Slave Ack
 NACK Master Non-Ack

Figure 2: List of command sequences understood by Slave Controller

As a general rule, a start command or stop command will always reset the state machine to its initial state. However, in some conditions (for example if a master transfer is corrupted in the middle of a slave read) the slave controller may be left in control of the bus. In this instance the master may not be able to drive the SDA line. To avoid a possible deadlock, the generic parameter *use_timeout* may be specified. When enabled, if the Slave Controller is stuck in the middle of a transfer, it will revert back to its initial idle state after the number of system clock cycles specified in *timeout*.

Master Instruction FIFO

The commands and data decoded by the slave controller core are buffered in the Master Instruction FIFO. The depth of this FIFO is determined by the generic parameter *wfifo_depth*. The FIFO interface operates in accordance with the valid/ready pipeline protocol meaning that instructions and data are read from the FIFO on the rising edge of *clk* when *mast_val* is high and *mast_rdy* is high²

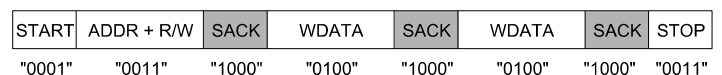
If required, the Master Instruction FIFO may be used to buffer incoming requests while the Slave is 'busy'. The user application circuit may stall/advance requests by asserting *mast_rdy* low or high accordingly. If no buffering is required, the signal *mast_rdy* may be tied high.

As a general rule, for each request, the maximum number of system clock cycles that the user may safely stall the FIFO for is determined by Equation 1:

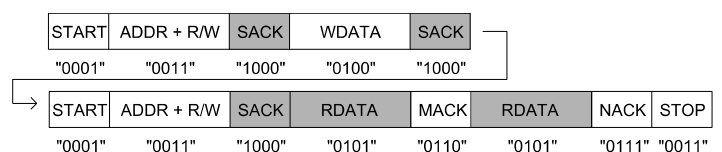
$$Stall\ Cycles \leq f_{clk} / f_{SCL} * 2 \quad (Equ. 1)$$

Where f_{clk} is the system clock frequency and f_{SCL} is the SCL clock frequency. For example, if the system clock is running at 100MHz and the SCL clock is running at 400kHz then the maximum number of stall cycles permitted would be 125. In practice, the system clock frequency will generally be much higher than the SCL clock frequency, and as such, stalling the FIFO for a few cycles will not pose any problem.

The decoded instructions are very intuitive and follow the exact sequence of commands that were sent by the Master on the I2C bus. As an example, consider the following two byte write sequence. The 4-bit values under each command show how the instruction *mast_inst* would be decoded in this instance.



As another example, the following is a two byte read sequence sent by the I2C Master:



A full description of instructions is given in the next table. Note that the user may use the master output port to drive their application circuit. Usually, this involves the implementation of a simple index register and a number of writeable data registers that are addressed by the index register.

The user circuit is not obliged to perform any particular action *except* in the case of slave read (RDATA) instruction. For every RDATA instruction, the application circuit must write the appropriate read data into the Slave Read FIFO.

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

MASTER INSTRUCTION OUTPUT FORMAT

mast_inst[3:0]	mast_data[7:0]	Description
"0001"	[7:0] : 00000000	START I2C start command (SCL high, SDA falling edge)
"0010"	[7:0] : 00000000	STOP I2C stop command (SCL high, SDA rising edge)
"0011"	[7:1] : Slave Address [0] : R/W flag	ADDR 8-bit Slave Address
"0100"	[7:0] : Write data	WDATA Write 8-bit slave data
"0101"	[7:0] : 00000000	RDATA Read 8-bit slave data
"0110"	[7:0] : 00000000	MACK Master ack signal (SDA low, SCL clock pulse)
"0111"	[7:0] : 00000000	NACK Master no-ack signal (SDA high, SCL clock pulse)
"1000"	[7:0] : 00000000	SACK Slave ack (SDA tristate, SCL clock pulse)
"1111"	[7:0] : 00000000	TIMEOUT Slave Controller has timed out (Only applicable if the generic <i>use_timeout</i> is enabled)
Other values	[7:0] : Reserved	NULL Reserved

Slave Read FIFO

The Slave Read FIFO accepts read data in response to a RDATA instruction. The Slave read-data port follows the standard valid-ready handshake protocol. For every RDATA instruction, the user application circuit must provide one byte of read data. The slave controller core then serializes the 8-bit data for presentation on the I2C bus.

It is important to note that after a RDATA instruction is received, the user circuit must provide the read data as soon as possible – or at least in sufficient time before the next rising SCL clock edge (plus a margin for setup). To avoid possible data corruption on the bus, it is advised that the read data be made valid according to Equation 2.

$$Cycles \leq f_{clk} / f_{SCL} * 4 \quad (\text{Equ. 2})$$

Where f_{clk} is the system clock frequency and f_{SCL} is the SCL clock frequency. For example, if the system clock is running at 100MHz and the SCL clock is running at 400kHz then the maximum number of cycles permitted before read data is valid would be 62.

Functional Timing

Figure 3 demonstrates a simple sequence of I2C bus signals comprising of a START, ADDR, SACK, RDATA, NACK then STOP. In this instance the slave address is 0x54 and the slave read data is 0xA4.

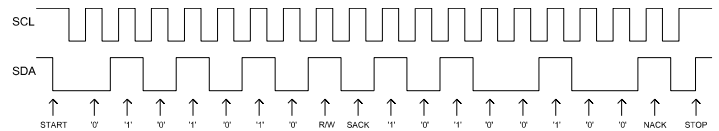


Figure 3: I2C single read operation

Figure 4 shows the resulting set of commands issued on the master port in response to the I2C signalling in Figure 3. Note that the FIFO is stalled after the second instruction due to *mast_rdy* being de-asserted for one clock cycle. In the following cycle, *mast_rdy* goes high and the transfer continues. In this example, on receipt of the RDATA instruction (0x5) the application circuit provides the slave read data (0xA4) after two system clock cycles.

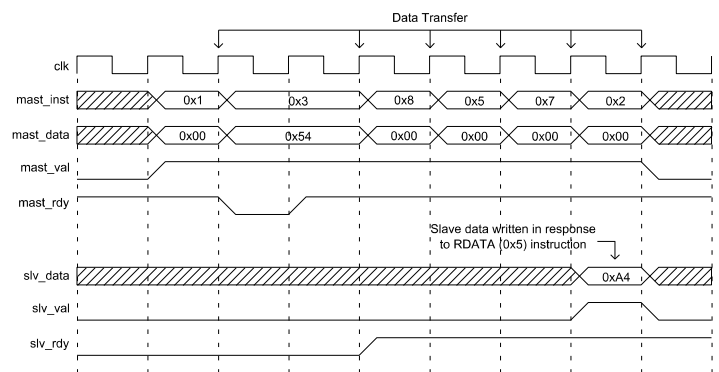


Figure 4: Master port output instructions

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_slave_stim.txt	Input stimulus text file
i2c_inbuf.vhd	Input buffer
i2c_iobuf.vhd	Bi-directional tristate buffer
i2c_fifo.vhd	Input/output FIFOs
i2c_slave_cont.vhd	Main I2C slave controller
i2c_slave.vhd	Top-level block
i2c_slave_file_reader.vhd	Reads the I2C bus signals from a text file
i2c_slave_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_jobuf.vhd
3. i2c_fifo.vhd
4. i2c_slave_cont.vhd
5. i2c_slave.vhd
6. i2c_slave_file_reader.vhd
7. i2c_slave_bench.vhd

The VHDL test bench instantiates the `i2c_slave` component together with a file-reader module that reads the I2C bus signals from a text file. The SCL clock frequency may be modified by changing the generic setting `t_period` on the file reader component.

The input text file is called `i2_slave_stim.txt` and should be put in the current top-level VHDL simulation directory.

The I2C bus signalling is split into 4 phases on 4 consecutive lines. Each line is comprised of two bits in the format 'A B' where 'A' specifies the state of the SCL line and 'B' is the state of the SDA line. The values 'A' and 'B' can either be specified as '0' (logic 0), '1' (logic '1') or 3 (tristate).

As an example, in order to send a start command, the text file should read:

```
0 1 # SCL = 0, SDA = 1
1 1 # SCL = 1, SDA = 1
1 0 # SCL = 1, SDA = 0
1 0 # SCL = 1, SDA = 0
```

To send a single bit (in this case a '1') the text file may be written as:

```
0 1 # SCL = 0, SDA = 1
0 1 # SCL = 0, SDA = 1
1 1 # SCL = 1, SDA = 1
1 1 # SCL = 1, SDA = 1
```

In addition to setting up the input stimulus file with the desired I2C commands, the user may also modify the generic parameters on the I2C slave component as required.

In the default set up, the simulation must be run for around 10 ms during which time the file-reader module will drive the I2C bus with the input commands. The test bench will generate random slave data in response to any slave read requests.

The simulation generates the text file `i2c_slave_out.txt` which contains the output data captured at the master instruction ports during the course of the test. The contents of this file may be examined to verify the operation of the I2C slave controller.

Development Board Testing

The I2C Slave Serial Interface Controller was implemented on a Xilinx® 2V3000 FPGA running at a system clock frequency of 65MHz. In addition to the FPGA-based Slave, there were a number of other Slave devices connected to the I2C bus 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 I2C Master device was also implemented on the FPGA and was initially set up for 400kHz (Fast mode) operation.

After testing was performed at 400kHz, further testing was performed to verify correct operation at 100kHz (Standard mode) and at the custom data rates of 2MHz and 4MHz. The I2C Slave Serial Interface Controller functioned correctly at both these higher data rates but the other slave devices failed at 4MHz operation (They were only specified for 1.7MHz).

Figure 5 below demonstrates a read of the FPGA-based Slave with I2C bus signals operating at 400kHz. The top trace is the SCL line and the bottom trace is the SDA line. In this particular example the slave address was set to 0x54.

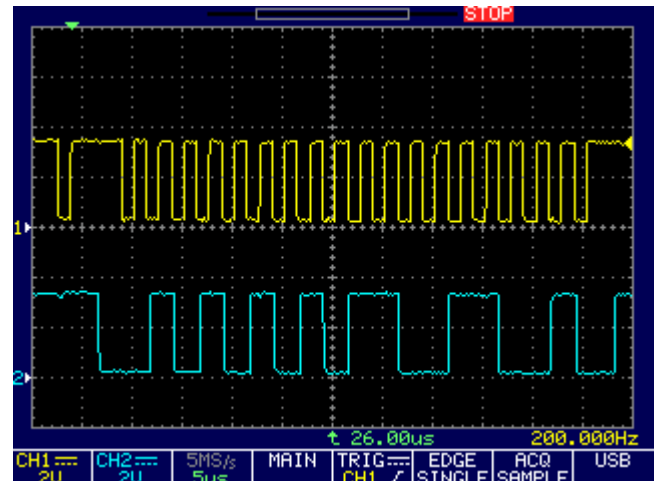


Figure 5: I2C Slave read at 400kHz (Fast mode)

The next figure shows the same device being addressed but this time, the operation is a write with a custom data rate of 2MHz.

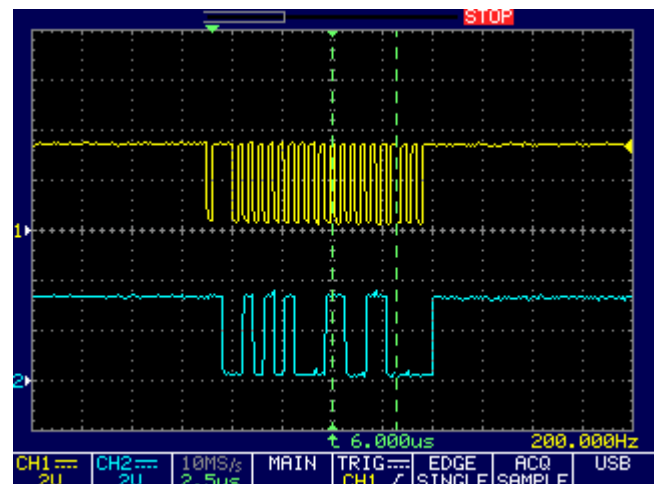


Figure 6: I2C Slave write at 2MHz custom data rate

Figure 7 shows a read of the FPGA-based slave at a 4MHz data-rate.

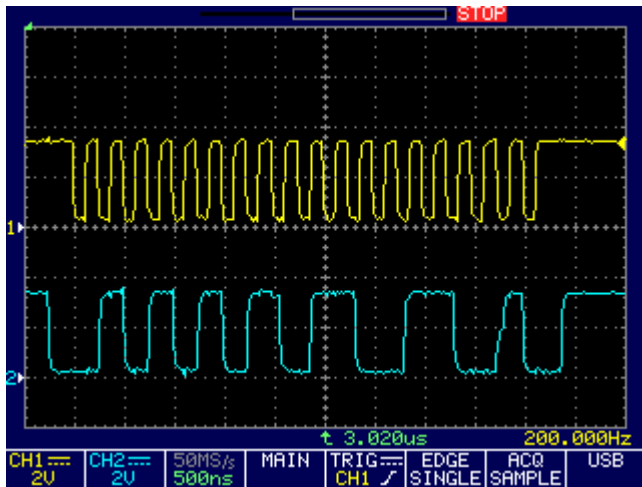


Figure 7: I2C Slave read at 4MHz custom data rate

Finally, Figure 8 shows detail of the clock measurement showing correct operation at 4MHz. Note that this was achieved at LVTTTL voltage levels with x12 drive strength output buffers and 4K7 pull-ups on the bus.

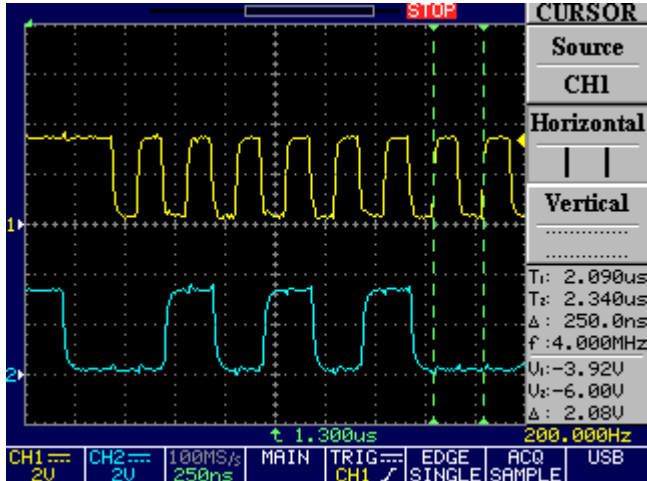


Figure 8: Clock detail at 4MHz custom data rate

Synthesis

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

- i2c_slave.vhd
 - i2c_slave_cont.vhd
 - i2c_fifo.vhd
 - i2c_delay.vhd
 - i2c_iobuf.vhd
 - i2c_inbuf.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 design, the parameter *use_timeout* must be set to *false*. In addition, the size of the FIFOs will have a substantial effect on the overall size of the circuit.

Trial synthesis results are shown with the generic parameters set to: *slave_address* = "01010101", *use_timeout* = true, *timeout* = 10000, *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	110
Slice LUT	152
Block RAM	0
DSP48	0
Clock frequency (worst case)	233 MHz
Clock frequency (best case)	334 MHz

STRATIX III

Resource type	Quantity used
Register	159
ALUT	166
Block Memory bit	160
DSP block 18	0
Clock frequency (worse case)	285 MHz
Clock frequency (best case)	350 MHz

Revision History

Revision	Change description	Date
1.0	Initial revision	09/10/2008