I2C controller core: FAQ

FAQ

This is a collection of questions and answers from the mailing-list about the I2C core.

Q&A

Q: What is the i2c_master_core ?
A: The i2c_master_core is a Wishbone RevB.3 compliant multi-master I2C controller.

Q: Where can I find the VHDL/Verilog code ?
A: The core is available in VHDL and Verilog and can be downloaded by clicking the "i2c" link in the downloads section.

Q: Are there any licensing issues ?
A: There are two licensing issues to consider:

- **Core: You can use the core as it is for free, provided you do not remove the copyright/license message. There is no patent or licensing issue for the core.**
- **I2C / SMBus (Philips): Philips is the owner of basic (and additional) patent rights on the I2C / SMBus. As far as I know most patents have already expired, but ICs manufactered or exported to the US and/or Canada are/can be subject to licensing fees.**

DISCLAIMER: The information above is not an official statement from Philips, nor is Philips somehow related to it.

I requested an official statement from Philips to be posted on this FAQ. This is Royal Philips Electronics N.V.'s official statement concerning I2C licensing issues for the OpenCores I2C core:
"PATENT NOTICE: Supply and use of I2C cores do not convey nor imply a right under the I2C patent rights of Royal Philips Electronics N.V. to make, use or sell any product employing these patent rights. An I2C patent license from Royal Philips Electronics N.V. is required for any use of such patent rights, including the implementation of this I2C IP core in an Integrated Circuit or any other device. For more information on licensing please refer to Philips' I2C website at http://www.semiconductors.philips.com/buses/i2c/licensing/ ."

Q: How do I read/write a byte from/to the I2C bus ?
A: Here's the sequence:

- **Initialise PRER to proper value (0x00A5).**
- **Initialise CTR (0x00C0).**
- **Write to transmit register TXR (0xE500).**
- **Read status register SR (optional)**
- **-- the Busy/TIP bits should not be set ! --**
- **Write to command register CR (0x00E9).**
- **Again read SR**
- **-- this time Busy/TIP bits should be set ! --**

Q: The core has two reset inputs, how do I connect them ?
A: The core has a synchronous active high wishbone reset (wb_rst_i) and an asynchronous reset (arst_i) input. The asynchronous reset level can be programmed using the ARST_LVL parameter. Use only 1 reset input, hardwire the other (of course to a non-reset level). The compiler/synthesizer should remove all logic related to the unused reset-input.

Q: How do I send/read multiple bytes from/to the I2C bus ?

A: The I2C core is byte based. All multi-byte transfers are broken down into several 1byte transfers. After transfering a byte wait for an interrupt or the negation of TIP (Transfer In Progress), then transfer the next byte.

Q: How does the core behave inbetween transfers ?
A: When the core finished a byte transfer and the STOP bit is not set, it waits for a new command. The SCL and SDA lines are kept in the state they were in when the command was completed. This is a low level for the SCL line.

Q: After a reset the RxACK flag is set to '0' (==ACK), is this correct ?
A: After a reset condition no transfer has been initiated yet. So in fact any value, either ACK or NACK, is wrong. It doesn't affect the operation of the core though. After an ACK bit has been received from the slave, the RxACK flag contains the correct value.

Q: During a read do I have to send an ACK bit ?
A: The I2C specs specify that every access should be acknowledged. During a read, the master must acknowledge the reception of the byte. The core automatically insterts the ACK-bit during the 9th bit-transfer. The value of the ACK-bit is set by the ACK bit in the Command-Register.

Q: How to connect the I2C Core to Altera's Avalon bus?
A: (thanks to Mark McDougall)
1) SOPC Builder can only handle std_logic_vectors. The core's address bus is of type unsigned. Create a wrapper that changes the unsigned bus to std_logic_vector.
2) Use the following Avalon-Wishbone port wiring

```
PORT_WIRING
{
PORT wb_clk_i
type = "clk";
PORT wb_rst_i
type = "reset";
PORT arst_i
type = "export";
PORT wb_adr_i
type = "address";
PORT wb_dat_i
type = "writedata";
PORT wb_dat_o
type = "readdata";
PORT wb_we_i
type = "write";
PORT wb_stb_i
type = "chipselect";
PORT wb_cyc_i
type = "chipselect";
PORT wb_ack_o
type = "waitrequest_n";
PORT wb_inta_o
type = "export";
}
```

***Avalon and SOPC Builder are registered trademarks of Altera.**