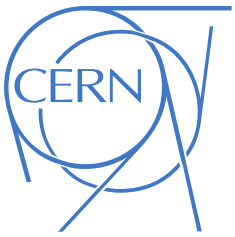
I^{2}C Slave Core

October 29, 2013



Theodor-Adrian Stana (CERN/BE-CO-HT)

Revision history

Date	Version	Change
26-06-2013	0.01	First draft

Contents

List of Figures

List of Tables

List of Abbreviations

ASIC	Application-Specific Integrated Circuit
FPGA	Field-Programmable Gate Array
I^{2}C	Inter-Integrated Circuit
SCL	Serial CLock
SDA	Serial DAta

1 Introduction

The *i2c_slave* VHDL module implements a simple I^{2}C slave core capable of responding to I^{2}C transfers generated by a master. The module is conceived to be controlled by an external module. Basic shifting of bits into the module is handled during read transfers (from the slave's point of view), at the end of which the user is presented with the received byte. Similarly, in the case of a write transfer, the user inputs a byte to be sent, and the module handles shifting out of each of the bits. The status of the module can be obtained via dedicated ports.

The main features of the *i2c_slave* module are:

- simple operation
  - passive until addressed by master
  - read transfers – presents the user with the received byte at specific port
  - write transfer – sends the byte at input port to the master
  - communication status can be checked via dedicated port
- 7-bit addressing
- standard (100 kHz) and fast (400 kHz) modes supported
- no clock stretching, all information provided by the module should be handled externally within the time span of an I^{2}C bit transfer
- internal watchdog timer resets logic in case of bus error
- architecture-independent, can be used with various FPGA types or ASICs

2 Instantiation

This section offers information useful for instantiating the *i2c_slave* core module. Table 1 presents a list of ports of the *i2c_slave* module.

I^{2}C-specific ports should be instantiated as outlined in Figure 1, via tri-state buffers enabled by the *scl_en_o* lines *sda_en_o*.

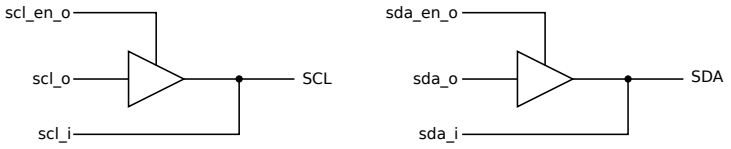


Figure 1: Connecting the I^{2}C ports

To instantiate a tri-state buffer in VHDL:

```
SCL   <= scl_o when (scl_en_o = '1') else
         'Z';
scl_i <= SCL;

SDA   <= sda_o when (sda_en_o = '1') else
         'Z';
sda_i <= SDA;
```

and in Verilog:

```
assign SCL   = (scl_en_o) ? scl_o : 1'bz;
assign scl_i = SCL;
assign SDA   = (sda_en_o) ? sda_o : 1'bz;
assign sda_i = SDA;
```

The rest of the ports should be connected in a normal manner to an external controlling module. A component declaration of the *i2c_slave* module is readily available in the *i2c_slave_pkg.vhd* package file. The package also defines constants for the statuses readable at the *stat_o* pin. Refer to Section 4 for details regarding the various statuses.

Table 1: Ports of *i2c_slave* module

Name	Size	Description
clk_i	1	Clock input
rst_n_i	1	Active-low reset input
scl_i	1	SCL line input
scl_o	1	SCL line output
scl_en_o	1	SCL line tri-state enable
sda_i	1	SDA line input
sda_o	1	SDA line output
sda_en_o	1	SDA line output tri-state enable
i2c_addr_i	7	I^2C slave address of the module, compaired against received address
ack_n_i	1	ACK to be sent to the master in case of master write transfers
op_o	1	State of the R/W bit at the end of the address byte
tx_byte_i	8	Byte of data to be sent over I^2C
rx_byte_o	8	Byte received over I^2C
done_p_o	1	One *clk_i* cycle-wide pulse, signaling the slave module has performed a valid transfer
stat_o	3	Current state of communication

3 I²C Bus Protocol

The I²C bus protocol is a two-wire protocol defined by Philips/NXP. The original specification [1] defines all aspects of the protocol, from hardware connections on the bus, to bit- and byte-level data transfers and electrical characteristics of the bus. A summary of the protocol is given here.

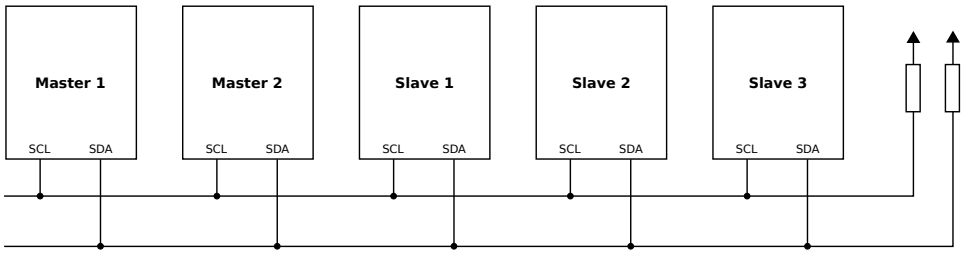


Figure 2: I²C bus topology

Devices on the I²C bus are connected together via two pins on the bus: the SCL (serial clock) and SDA (serial data) pins. I²C masters drive the SCL line to send or receive bits on the SDA line. Both the SCL and SDA lines on an I²C device are open-collector pins; as Figure 2 shows, one pull-up resistor on the bus connects the line to VCC and I²C devices connect the SCL and SDA lines to ground when they drive the lines. In this way, a device can set a logic low level on the bus by driving the pin and a logic high level by releasing the pin.

A typical I²C bit-level transfer (Figure 3) follows the following sequence:

- master sends a start condition, driving the SDA line low while the SCL line is high
- master issues a series of SCL pulses to a slave to read or write bits; the SDA line must be stable for the duration of SCL high pulse for the bit to be properly transferred
- master sends a stop condition by releasing the SDA line while SCL line is high, or a repeated start (similar to start) condition if it wants to continue data transfer

Data are transferred on the bus in bytes, one bit at a time starting with the most significant bit. After each sent byte, the other communicating party ACKs ('0') or NACKs ('1') the transfer on a 9^{th} SCL cycle. Any number of bytes can be sent during a transfer, the master decides when data transfer should stop by sending the stop condition. The folowing steps comprise a complete I²C data transfer (Figure 4):

- master sends start condition

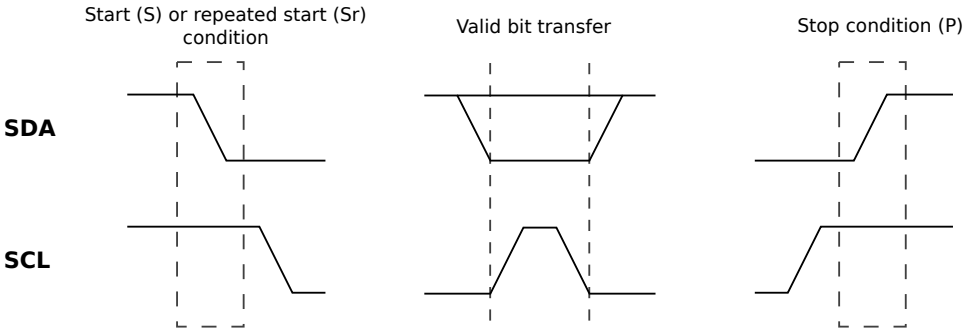


Figure 3: Bit-level transfers on the I²C bus

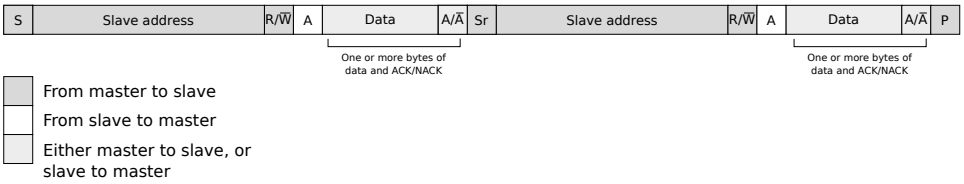


Figure 4: Bytes transferred on the I²C bus

- master sends slave address (7 bits of address + one R/W bit)
- if a slave with this address exists, it ACKs ('0') the master
- based on the R/W bit ('0' for read from slave, '1' for write to slave), the master either reads or writes a byte bit by bit from/to the slave
- the receiver ACKs ('0') or NACKs ('1') the byte on the ninth SCL cycle
- any number of bytes may be sent, each followed by an ACK or NACK from the receiver
- **optional:** the master may (or may not) reverse data transfer by issuing a repeated start and sending the slave address with the R/W bit flipped
- **optional:** any number of bytes may be sent, each followed by an ACK or NACK from the receiver
- the master ends data transfer by sending the stop condition

4 Operation

The *i2c_slave* waits for a start condition to be performed on the I^{2}C bus by a master module. The address is shifted in and if it matches the slave address set via the *i2c_addr_i* input, the *done_p_o* output is set for one *clk_i* cycle and the *stat_o* output signals an address match. Based on the eighth bit of the first I2C transfer byte, the module then starts shifting in or out each byte in the transfer, setting the *done_p_o* output for one clock cycle after each received/sent byte. The *stat_o* output can be checked to see if the byte has been sent/received correctly.

When the cycle-wide *done_p_o* output is high (after every successful transfer, or a stop condition) the *stat_o* (possibly together with the *op_o*) output can be checked to see the appropriate action to be taken. The various statuses possible at the *stat_o* output are listed in Table 2.

Table 2: Statuses at the *stat_o* pin

stat_o	**Description**
00	Slave idle, waiting for start condition. This is the state upon startup and after the I^{2}C stop condition is received
01	Address sent by the master matches that at *i2c_addr_i*; *op_o* valid
10	Read done, waiting for ACK/NACK to send to master
11	Write done, waiting for next byte to send to master

The *ack_n_i* port is used for sending the ACK to the master. The polarity of the bit is that of the I^{2}C ACK signal ('0' – ACK, '1' – NACK). A '0' should be set at the input also when the address is ACKed, otherwise the slave will not acknowledge its own address. This implies that the *ack_n_i* pin can be used to isolate the slave from the bus.

4.1 Read mode

When the eighth bit of the address byte is low (R/W = '0'), the slave goes into read mode. Each bit of the byte sent by the master is shifted in on the falling edge of SCL. After eight bits have been shifted in, *done_p_o* is set for one *clk_i* cycle and the status signals a successful read ("10"). The received byte should be read from the *rx_byte_o* output and an ACK ('0') or NACK ('1') should be sent to the master via the *ack_n_i* pin. The *i2c_slave* module does not implement clock stretching, so the *ack_n_i* pin should be set before the SCL line goes high.

The steps below should be followed when reading one or more bytes sent by the master:

1. Wait for *done_p_o* to go high, signaling the I^2C address of the slave has been read.
2. Check that *stat_o* is "01" (address good) and that *op_o* is '0' (master write, slave read). Set a '0' at the *ack_n_i* input to send the ACK to the address; if *ack_n_i* is '1', the slave does not acknowledge its own address.
3. Wait for *done_p_o* to go high.
4. Check that *stat_o* is "10" (read done), read the received byte from *rx_byte_o* and write a '0' at *ack_n_i* to send an ACK, or a '1' to send an NACK.
5. The transfer is repeated until the master sends a stop condition.
6. After the stop condition is received, the *done_p_o* goes high for one clock cycle and the status is set to "00".

4.2 Write mode

When a master reads from the slave, the eighth bit of the address byte is high (R/W = '1'). In this case, the *i2c_slave* module goes in write mode, where the byte at the *tx_byte_i* port is sent to the master. When the byte has been successfully sent, the *done_p_o* is high for one clock cycle and the *stat_o* port has the value "11", signaling the slave has successfully sent a byte and is awaiting the loading of another byte.

The steps below should be followed when writing one or more bytes to a master:

1. Wait for *done_p_o* to go high, signaling the I^2C address of the slave has been read.
2. Check that *stat_o* is "01" (address good) and *op_o* is '1' (master read, slave write). Set the byte to be sent to the master at the *tx_byte_i* input. Set a '0' at *ack_n_i* to send the ACK to the address; if *ack_n_i* is '1', the slave does not acknowledge its own address.
3. Wait for *done_p_o* to go high.
4. Check that *stat_o* is "11" (write done) and set the next byte to be sent at the *tx_byte_i* port.

5. If the master acknowledges the transfer, the next byte is sent, otherwise, the master will send a stop condition, so the *i2c_slave* module is reset.

Note that if a stop condition is received from the master, the *done_p_o* goes high for one clock cycle and the status is set to "00".

5 Implementation

This section presents implementation details of the *i2c_slave* module. A simplified block diagram of the module is presented in Figure 5.

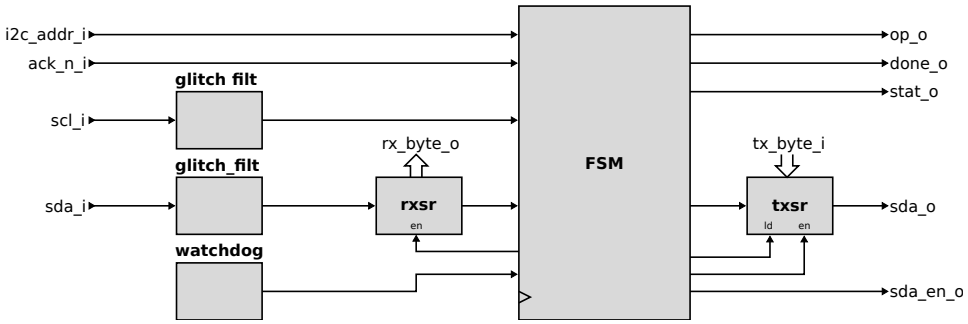


Figure 5: Block diagram of *i2c_slave* module

Deglitched versions of the SCL and SDA lines control operation of the central finite-state machine (FSM), which sets the outputs and controls the rest of the components in the module.

The FSM is sensitive to start and stop conditions and falling edges of the SCL line. It controls how outputs are set, when the reception and transmission shift registers (RXSR/TXSR) are loaded and when they shift, and acknowledging to the address and bytes sent by the master. Table 3 lists the states of the FSM and the operations performed in each state.

An internal watchdog counter is implemented inside the *i2c_slave* module. This counter counts up to 1 second and is reset at the start of each state of the FSM. If the FSM stops in one of the states because of a bus error, the watchdog resets the FSM, thereby stopping the communication.

Table 3: The states of the *i2c_slave* FSM

State	Description
IDLE	Idle state, FSM default state after reset and the state returned to after reception of a stop condition.
STA	State reached after a start condition is received. On the falling edge of SCL, the FSM transitions to *ADDR* state.

State	Description
ADDR	Shift in 7 address bits and R/W bit and go to *ADDR_ACK* state. Each bit is shifted in on the falling edge of SCL. If the received address matches, *op_o* and *done_p_o* are set.
ADDR_ACK	Check received address and send the ACK value at *ack_n_i* if the address corresponds to *i2c_addr_i*. If the R/W bit is high, go to *RD* state, otherwise go to *WR_LOAD_TXSR* state. If received address does not match, NACK and go to *IDLE* state.
RD	Shift in eight bits sent by master and go to *RD_ACK* state. Each bit is shifted in on the falling edge of SCL. When eight bits have been shifted in, set *done_p_o*.
RD_ACK	Read *ack_n_i* and forward it to *sda_o* (ACK/NACK from external controller). If *ack_n_i* is '0', then go back to *RD* state, else to *IDLE* state.
WR_LOAD_TXSR	Load TX shift register with data at *tx_byte_i* input and go to *WR* state.
WR	Shift out the eight bits of the TXSR starting with MSB and go to *WR_ACK* state. TXSR shifts left on falling edge of SCL. When eight bits have been shifted out, *done_p_o* is set.
WR_ACK	Read ACK bit sent by master. If '0', go back to *WR* state, otherwise go to *IDLE* state.

References

[1] "I2C Bus Specification, version 2.1," Jan. 2000. http://www.nxp.com/documents/other/39340011.pdf.