I^2C Slave Core

Theodor-Adrian Stana
CERN, BE-CO-HT

March 22, 2013

Contents

1 Introduction

This document presents the *i2c_slave* VHDL module, implementing a simple I^2C slave core capable of responding to I^2C 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 *i2c_slave* module does not implement clock stretching, all information provided by the module should be handled externally within the time span of an I^2C bit transfer.

2 I^2C Bus Protocol

The I^2C 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 about the widely-used protocol is given here.

Devices on the I^2C bus are connected together via two pins on the bus: the SCL (serial clock) and SDA (serial data) pins. The I^2C bus uses a master-slave topology, with I^2C masters driving the SCL line to send or receive bits on the SDA line. Both the SCL and SDA lines on an I^2C device must be open-collector outputs.

As Fig. 1 shows, one pull-up resistor on the bus connects the line to VCC and I^2C 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.

To initiate a transfer, a master on the bus generates a start condition, driving the SDA line low while the SCL line is high. Then, a series of SCL pulses is used to send or receive bits on the bus, with one SCL pulse sent

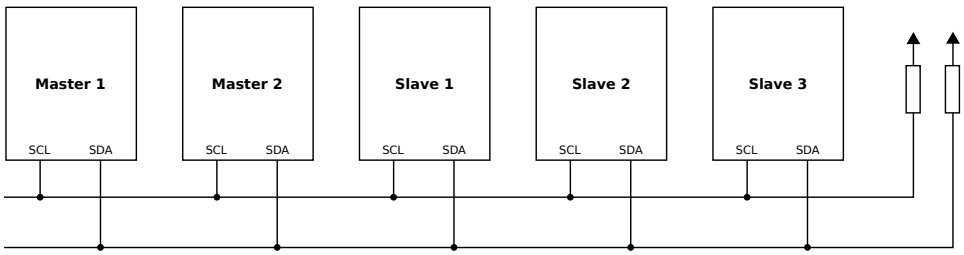

Figure 1: I^2C bus topology

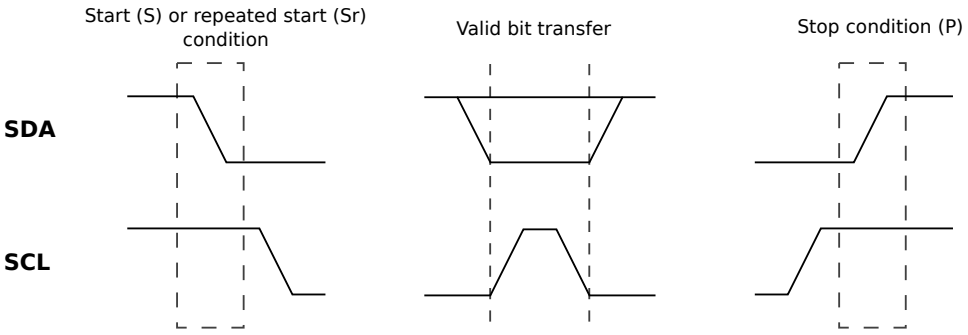

Figure 2: Bit-level transfers on the I^2C bus

for each bit. The signal on the SDA line must be stable for the duration of the SCL high pulse for the bit to be properly received at the receiver side. When the master has finished transferring bits it must release control of the bus by issuing a stop condition. This is done by releasing the SDA line while the SCL line is high. Alternatively, if the master has finished the transfer and still wants to continue sending data, it can issue a repeated start condition, which is identical to the start condition. The various types of bit-level transfers are summarized in Figure 2.

Since there can be more than one slave on the bus, each bus is assigned an address to which it responds when addressed. Two forms of addressing are defined in the I^2C bus specification, a 7-bit addressing mode and a 10-bit addressing mode. Of the two, the 7-bit addressing mode is the most widely used and also the only mode that the *i2c_slave* module supports.

By this mode, after the master issues the start condition on the bus, it sends a series of seven address bits MSB-first on the bus, followed by an eighth bit stating whether the master wishes to read from ('1') or write to ('0') the slave. After this series of eight bits, if a slave with this address exists on the bus, it must acknowledge (ACK) the transfer by driving the SDA line low. If no master with this address exists, the SDA line remains high (not acknowledge–NACK) and the master aborts the transfer. Figure 3

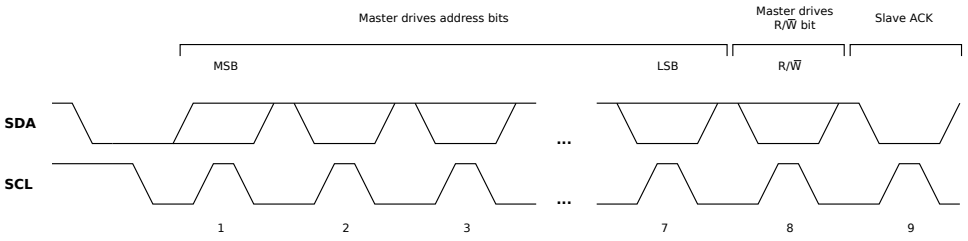

Figure 3: Sending of the address byte by the master

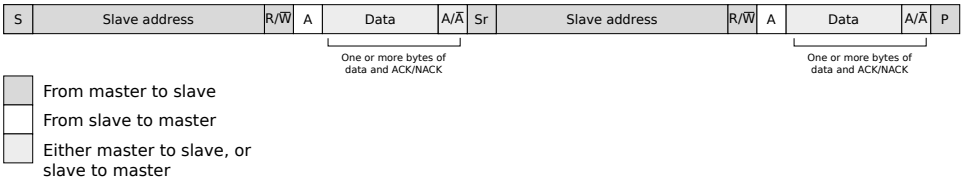

Figure 4: Bytes transferred on the I^{2}C bus

shows how a master sends the address byte to the slave and how a slave acknowledges the transfer.

After a slave acknowledges its address, actual data transfer can begin. Based on the R/$\bar{W}$ bit, the master either reads or writes a byte from/to the slave. After the byte is written, the receiver (master or slave) must acknowledge the byte in the same manner as a slave acknowledges its address. A low level on the 9^{th} SCL cycle means an ACK, while a high level on the 9^{th} SCL cycle means an NACK. Any number of bytes can be sent between transmitter and receiver. The transfer ends with the master sending a stop condition on the bus, after the ACK/NACK bit has been sent. The master can issue a stop condition as a result of both an ACK and an NACK. A stop condition can however not follow a start condition without transferring data.

An example data transfer is shown in Figure 4. The master first issues a start condition and sends the address, which is acknowledged by the slave. Data transfer then starts, with either the master or the slave sending data. At one point, the master decides to reverse the order of data transfer and issues a repeated start with the R/$\bar{W}$ bit flipped. A number of data bytes are transferred, and the master ends the transfer with the stop condition.

3 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 5, via tri-state buffers enabled by the *scl_en_o* lines *sda_en_o*.

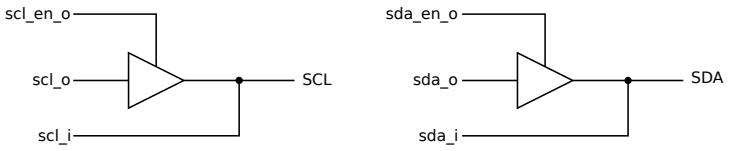

Figure 5: Connecting the I^{2}C ports

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 about the 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 output enable
sda_i	1	SDA line input
sda_o	1	SDA line output
sda_en_o	1	SDA line output 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/$\bar{W}$ 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

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 the byte has been sent/received correctly.

As can be seen from the above description, *done_p_o* is high after every completed I^{2}C transfer. As a general rule, it should be checked periodically and when high, the *stat_o* (possibly together with the *op_o*) output should 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
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/$\bar{W}$ = '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.

Following are the steps that should be performed to read 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.

4.2 Write mode

When a master reads from the slave, the eighth bit of the address byte is high ($R/\bar{W}$ = '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.

Below are the steps which should be followed to write 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.

5 Implementation

This section presents implementation details of the *i2c_slave* module. A simplified block diagram of the module is presented in Figure 6.

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.

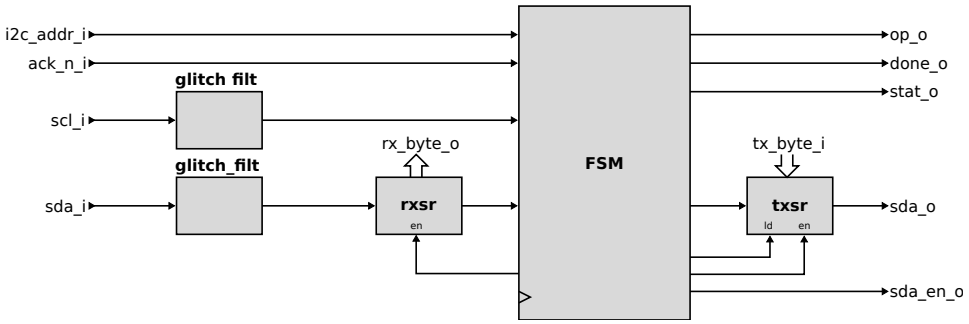

Figure 6: Block diagram of *i2c_slave* module

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.

Table 3: *i2c_slave* module state machine

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.
ADDR	Shift in 7 address bits and R/$\bar{W}$ 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* is set and *done_p_o* as well.
ADDR_ACK	Check received address and send ACK if it corresponds to *i2c_addr_i*. If the R/$\bar{W}$ 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	Clear *done_p_o*. 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.