SPI-to-I2C Interface Design Example

Contents

Introduction

This application note provides a design example for an interface between the standard SPI of a host and a serial I2C bus. The I2C is a two-wire bus used to enable communication between two or more devices that are normally on the same board. The speed is 100 Kbps or 400 Kbps for normal devices and 1 Mbps for fast devices.

SPI is a serial bus and is very common in the embedded world. SPI supports full duplex communication with higher throughput than I2C. Many embedded systems have only SPI interfaces, making them difficult to connect with I2C peripheral devices. You can modify the connections, but the resulting system is not efficient. One of the best ways to deal with this problem is to create an SPI-to-I2C interface and implement it in an Actel IGLOO® device, such as an IGLOO/e, IGLOO nano, or IGLOO PLUS device. This provides design flexibility as well as power flexibility. In addition, IGLOO FPGAs are reprogrammable and designed to meet the demanding power and area requirements of today's portable and power-conscious electronics. This application note provides a design example implemented in the Actel IGLOO FPGA.

I2C and SPI

I2C, a serial bus invented by Philips, is used to communicate with low-speed peripherals. It uses two bidirectional open-drain lines: Serial Data (SDA) and Serial Clock (SCL). The master initially sends a start bit, followed by the 7-bit address of the slave it wishes to communicate with, which is finally followed by a single bit representing whether it wishes to write (0) to or read (1) from the slave. If the slave exists on the bus, it will respond with an ACK bit (active low for acknowledged) for that address. The master then continues in either transmit or receive mode, and the slave continues in its complementary mode. Every data byte put on the SDA line must be 8-bits long. Figure 1 shows the I2C communication scheme.

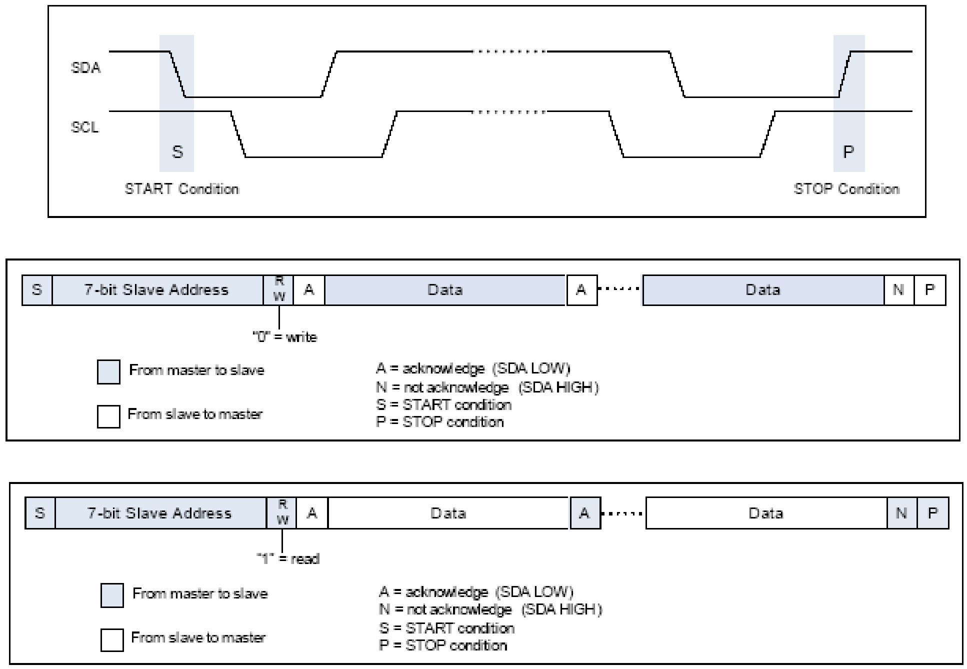


Figure 1 • **I2C Communication**

SPI, on the other hand, is a synchronous serial data link standard named by Motorola that operates in full duplex mode. In addition, SPI is not limited to 8-bit words, so you can send any message size with arbitrary content and purpose. Figure 2 shows the SPI communication scheme. Multiple slave devices are connected to a single master with individual slave select (chip select) lines. The master pulls the slave select low for the desired chip. The master then issues clock cycles. During each SPI clock cycle, a full duplex data transmission occurs:

- Master sends a bit on the MOSI line; the slave reads it from that same line.
- Slave sends a bit on the MISO line; the master reads it from that same line.

The two clock parameters, CPOL and CPHA, set the clock polarity and clock phase.

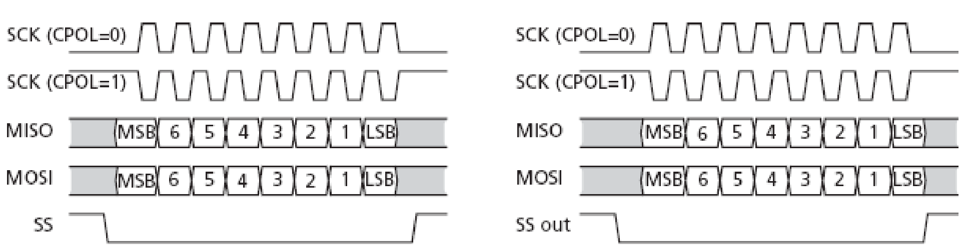


Figure 2 • **SPI Communication**

Design Description

The SPI-to-I2C interface design has three main blocks: the SPI Slave, SPI_I2C Controller, and I2C Master. Figure 3 shows the block diagram of the design.

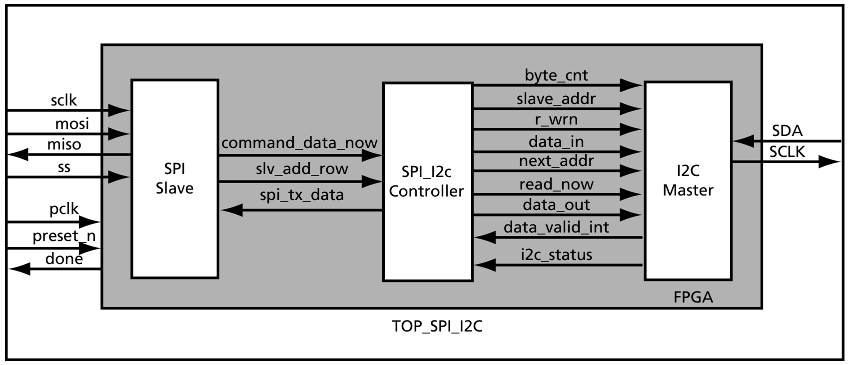


Figure 3 • **SPI-to-I2C Interface Top-Level Block Diagram**

1. SPI Slave block: This is the SPI slave. This is a 32-bit SPI slave, which can operate with different CPOL and CPHA settings. The top-level generic sets the CPOL and CPHA settings. The default settings are CPOL = 0 and CPHA = 0. The SPI slave receives a command from the external SPI master and passes it to SPI_I2C controller. It also sends the data read from the external I2C slave and status of I2C master back to the external SPI master through the MISO port.
2. SPI_I2C Controller block: This is the main block that performs the SPI-to-I2C function. The SPI slave receives the command from an external SPI master and passes it to this block. The SPI_I2C controller decodes the command and takes appropriate action. The four MSB of the signal from the SPI master (MOSI) defines the message, as shown in Table 1 on page 3. For a write or read operation to the I2C slave, the SPI_I2C controller configures the I2C master to send the I2C slave address, and then sends or receives data to or from the external I2C slave. During the "Send the read data back to external SPI master" and "Read the status of I2C master" commands, the SPI_I2C controller sends the data or status back to the SPI master through the SPI slave block.

Table 1 • **SPI Commands**

Command	Message
0001	Write one or two bytes to I2C slave
0010	Read one or two bytes from I2C slave
0100	Send the read data back to external SPI master
1000	Read the status of I2C Master

3. I2C Master Block: The I2C master uses a custom I2C core. Refer to "Appendix A: Custom I2C Core" on page 10 for more information on this core. This IP core has four generics that are used to configure different modes of operation.

- SYNC_BE: This is used to configure the output signals from the core. If '0', the output signals are synchronous to the SCLK clock and valid for an entire SCLK clock. If '1', the output signals are synchronous to the input clock.
- MODE: This is used to specify the speed of the I2C transactions. If '0', the STD speed (100 Kbps) is selected and if '1', the FAST speed (400 Kbps) is selected. This generic and REFCLK_SPEED are used to configure the clock rate for the mode desired.
- REFCLK_SPEED: This is used to specify the frequency input to the reference clock input.
- BC_WIDTH: This is used to specify the width of the byte counter in the master core. Valid values are any integer from 2 through 10.

Figure 4 shows the status register bit definitions for a generic BC_WIDTH setting of 4. The status bit (3:0) keeps track of the number of data written to I2C slave or read from the I2C slave. By reading the I2C status and comparing against the SPI command, you can decide whether to repeat the last command or not. This is useful when theI2C slave does not send an acknowledgement and the transmission stops without any write or read phase.

7				0
Done	Busy	"0"	RW	Bytecount (3:0)
1=Done	1=Busy		0=write 1=read	Number of bytes received / transmitted

Figure 4 • **I2C Master Status Register**

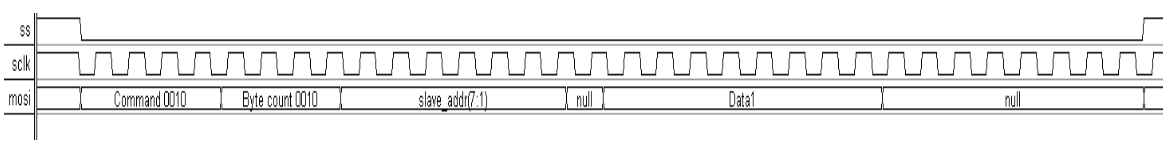
Files for this design example can be downloaded from the Actel website:
www.actel.com/download/rsc/?f=SPI_I2C_Interface_DF.

SPI Message Format

The following section explains SPI messages and operation of the SPI-to-I2C interface in detail.

Write 2 Bytes to I2C Slave Device

0001 Command	0001 Number of Bytes	Slave Address + 0	Data1	null



Start	Slave Address	W	A	Data 1	A/Ā	Stop

Figure 5 • **Write 2 Bytes to I2C Slave Device**

The SPI host issues the write command by sending a "0001" command followed by the total number of data bytes to be sent ("0010" for 2 bytes) and the address of the I2C slave followed by the data bytes, beginning with the first byte and ending with the second byte. Note that there is a redundant zero bit after the address, which has no effect on the read or write command. After receiving the message, the SPI-to-I2C interface accesses the I2C bus and begins sending the I2C bus signal. The interface sends the start bit followed by the I2C slave address and I2C write command. It waits for an acknowledgement from the slave and sends the data bytes using the standard I2C protocol. If the I2C slave does not send an acknowledgement, the interface sends a stop and ends the transmission. When the I2C bus write transaction has successfully finished, it asserts the done signal for one clock cycle.

Write 1 Byte to I2C Slave Device

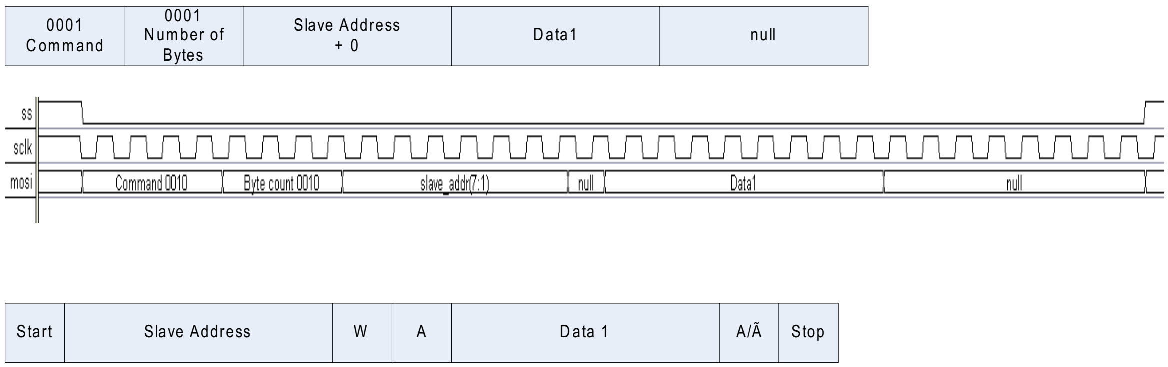


Figure 6 • **Write 1 Byte to I2C Slave Device**

The SPI host issues the write command by sending a "0001" command followed by the total number of data bytes to be sent ("0001" for 1 bytes) and the address of the I2C slave followed by the data bytes. After receiving the message, the SPI-to-I2C interface accesses the I2C and begins sending the I2C bus signal. It sends the start bit followed by the I2C slave address and I2C write command. It waits for an acknowledgement from the slave and sends the data byte using the standard I2C protocol. When the I2C bus write transaction has successfully finished, it asserts the done signal for one clock cycle.

Read 2 Bytes from I2C Slave Device

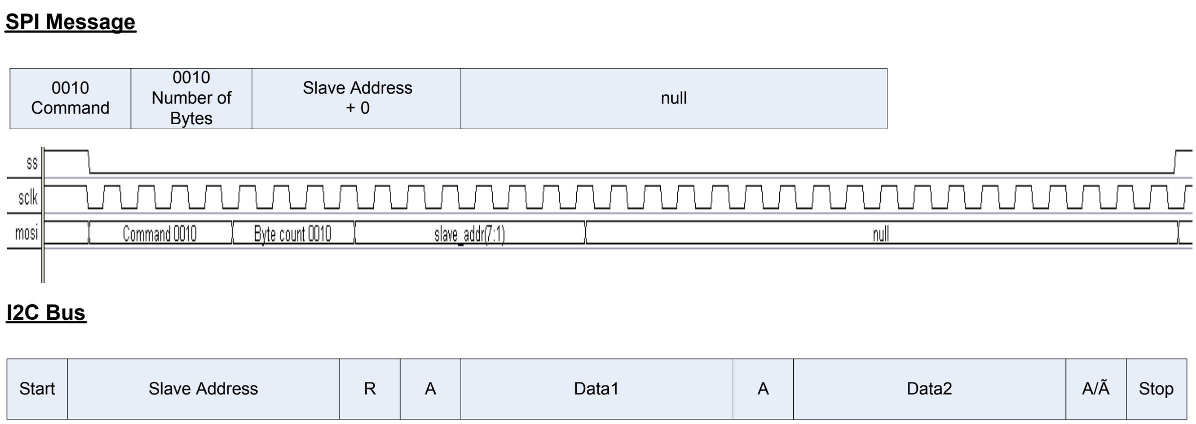


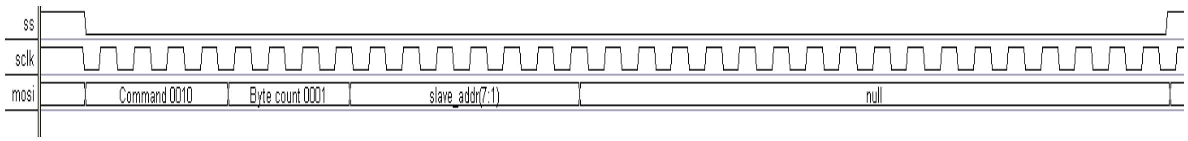
Figure 7 • **Read 2 Bytes from I2C Slave Device**

The SPI host issues the read command by sending a "0010" command followed by the total number of data bytes to be read ("0010" for 2 bytes) and the address of the I2C slave. Note that there is a redundant zero bit after the address, which has no effect on the read or write command. After receiving the message, the SPI-to-I2C interface accesses the I2C and begins sending the I2C bus signal. It sends the start bit followed by the I2C slave address and I2C read command. It waits for an acknowledgement from the slave and reads the data bytes using the standard I2C protocol. If the I2C slave does not send an acknowledgement, it sends a stop and ends the transmission. When the I2C bus read transaction has successfully finished, it asserts the done signal for one clock cycle.

Read 1 Bytes from I2C Slave Device

SPI Message

0010 Command	0001 Number of Bytes	Slave Address + 0	null



I2C Bus

Start	Slave Address	R	A	Data1	A/Ã	Stop

Figure 8 • **Read 1 Bytes from I2C Slave Device**

The SPI host issues the read command by sending a "0010" command followed by the total number of data bytes to be sent ("0001" for 1 bytes) and the address of the I2C slave followed by the data bytes. After receiving the message, the SPI-to-I2C interface accesses the I2C and begins sending I2C bus signal and I2C read command. It sends the start bit followed by the I2C slave address. It waits for an acknowledgement from the slave and reads the data byte using the standard I2C protocol. If the I2C slave does not send an acknowledgement, it sends a stop and ends the transmission. When the I2C bus read transaction has successfully finished, an interrupt is generated on the done pin.

When the I2C bus read transaction has successfully finished, it asserts the done signal for one clock cycle.

Send the Read Data Back to External SPI Master

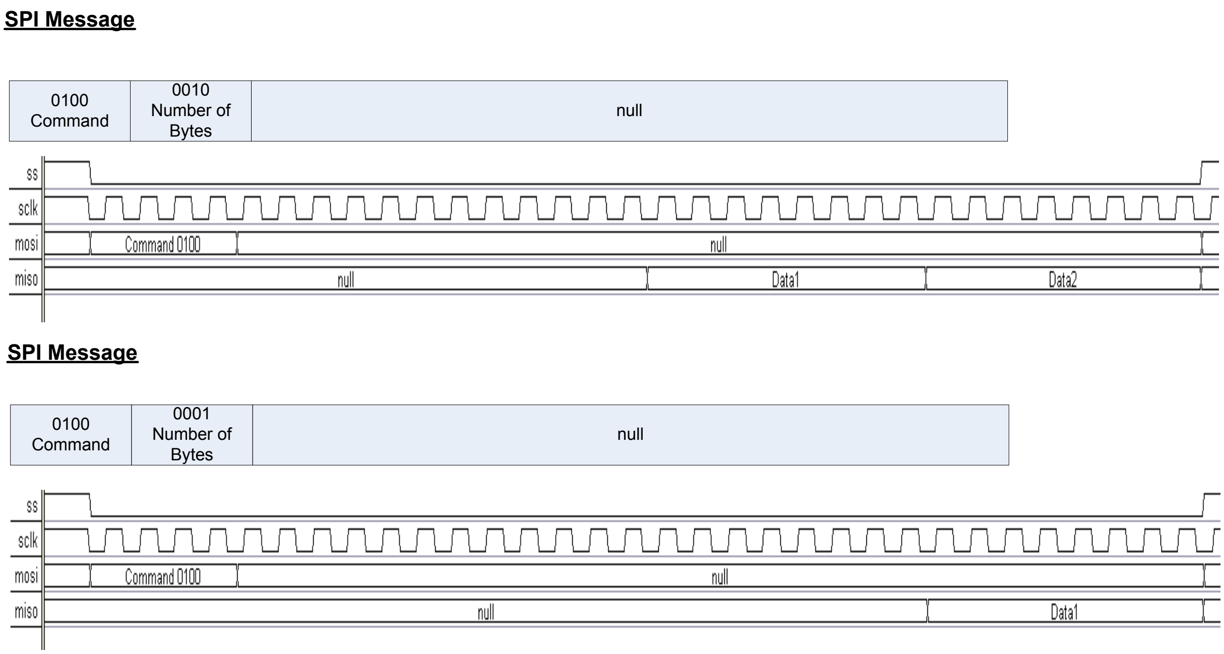


Figure 9 • **Send the Read Back Data (2 bytes or 1 byte) to External SPI Master**

The SPI host issues the "Send the read data back to external SPI master" command by sending a "0100" command followed by the total number of data bytes to be read ("0010" for 2 bytes, "0001" for 1 bytes). The SPI-to-I2C interface block sends the data back using the MISO pin. This occurs in the same transaction cycle.

Read the Status of the I2C Master

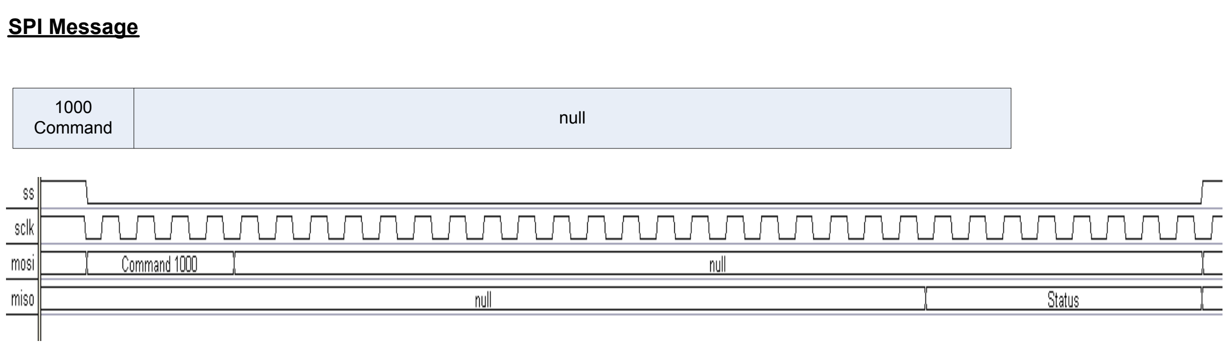


Figure 10 • **Send the Status of I2C Master**

The SPI host issues "Read the status of I2C master" command by sending a "1000" command. The SPI-to-I2C interface block sends the status back using the MISO pin. This occurs in the same transaction cycle.

SPI-to-I2C Top-Level I/O Description

Table 2 shows descriptions for the top-level ports for the SPI-to-I2C interface design.

Table 2 • **Ports Description of SPI-to-I2C Interface Design**

Port	Direction	Description
sclk	Input	Input Clock from the SPI master. Frequency depends upon the SPI master.
MOSI	Input	SPI data input from SPI master
MISO	Output	SPI data output to SPI master from SPI_I2C interface
ss	Input	Active low slave select output signal
PCLK	Input	Input clock (20 Mhz)
PRESET_N	Input	Active low reset signal
done	Output	Active high signal indicates the I2C transaction is complete
SDA	Input	I2C Serial Data
SCL	Output	I2C Serial Clock

Utilization Details

This design is targeted to Actel's AGLP125V2-CS289 device. The utilization details are given in Table 3.

Table 3 • **SPI-to-I2C Interface Design Utilization**

Resource	Utilized	Total	Percentage
Core	474	3,120	15.19%
I/Os	9	212	4.25%
Global (Chip + Quadrant)	3	18	16.67%
PLL	0	1	0.00%
RAM/FIFO	0	8	0.00%
Low Static ICC	0	1	0.00%
FlashROM	0	1	0.00%
User JTAG	0	1	0.00%

Testing Scheme

Verification of the core is done by simulation in ModelSim®. Hardware validation is done on Actel's IGLOO PLUS development board. In simulation verification, the testbench creates a system with an SPI master, SPI-to-I2C interface design, and an I2C slave. The SPI master signals are generated using CPOL = 0 and CPHA = 0 settings. However, you can easily modify the testbench for other CPOL and CPHA settings. The backend

I2C slave uses a custom I2C slave block. The Simulation results for the write and read cycle are illustrated in Figure 11 and Figure 12 on page 9.

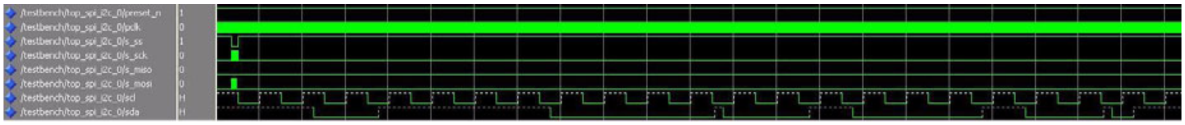


Figure 11 • **Write Cycle**

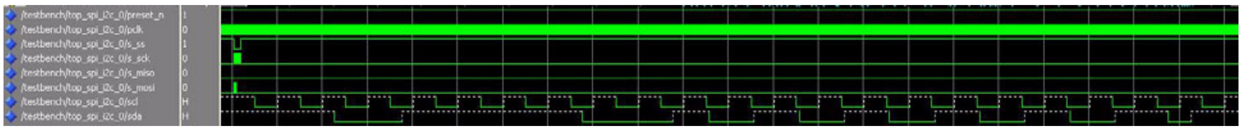


Figure 12 • **Read Cycle**

Conclusion

In the portable electronics market, the final product must be as small as possible. In addition, it must be power friendly. The SPI-to-I2C interface in IGLOO family FPGAs is a very good fit for this application.

Appendix A: Custom I2C Core

This custom core is compliant to the I2C specification v2.1 and meets all AC timing specifications. It supports single-master multiple-slave systems. Figure 13 shows the architecture of the custom master core. The filter block on the SDA and SCL inputs filter out glitches and noise that may be present on these inputs so that clean edge transitions are presented to the state machine block for processing. The SDA logic block controls the assertion and release of the SDA line for the transmission of data and ACKs to the slave I2C block.

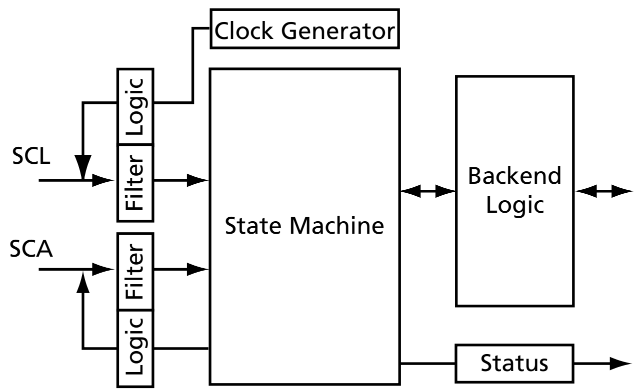


Figure 13 • **Custom I2C Master Architecture**

Table 4 • **I/O Descriptions for Custom I2C Core**

Signal	Direction	Description
Go	Input	HI active pulse that initiates an I2C operation controlled by Master
Restart	Input	HI active level that is used to continue a transaction with an addressed slave or address a new slave without having to STOP and then START again.
Data_in(7:0)	Input	Data input to be transmitted to the slave during a master write operation
Slave_addr(6:0)	Input	7-bit value assigned to the I2C slave resource—usually hardwired
R_WRN	Input	0=write to slave; 1=read from slave
Byte_cnt(X:0)[1]	Input	Any non-zero value is used by the master to determine the number of bytes to send or receive from the master before issuing a NACK (master read from slave) or STOP (master write to slave) to complete the transfer
Refclk	Input	Reference clock can be from 2 Mhz to 64 Mhz based on whether STD or FAST mode I2C is desired. See below for valid frequencies of REFCLK.
Reset_n	Input	Lo active asynchronous clear signal for all internal registers
SCL	Bidi	I2C serial clock
SDA	Bidi	I2C serial data
Data_out(7:0)	Output	Data output received from the master during a write to slave operation
Data_valid	Output	HI active pulse indicating data_out is valid
Next_addr(X:0)[1]	Output	Next address for data to be stored/transmitted
Read_now	Output	HI active pulse to request next data word; Only valid during a read from slave
I2C_status(Y:0)[2]	Output	8-bit status register indicating the state of the I2C master core.

Notes:

1. *The X value is defined by as BC_WIDTH-1*
2. *The Y value is defined by as 4+BC_WIDTH-1*

Figure 14 shows the state transitions of the state machine block. Once the GO signal is asserted to the master core, the SLAVE_ADDRESS and R_WRN are sampled and shifted out onto the I2C bus to address the slave device of interest. For a data transfer from the slave (READ), the master will issue ACKs until the value of BYTE_CNT (BC) is reached. The value of BC is used by the master to determine when to generate a NACK to the slave thereby completing the data transfer. For a write of data from the master to the slave (WRITE), the BC value is used to determine when the master is to expect a NACK from the slave and if none is received, then the master issues a STOP bus condition.

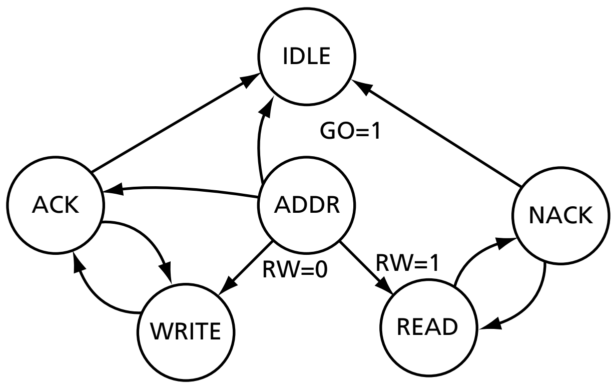


Figure 14 • **State Machine Diagram**

The backend logic block is used to simplify interfacing to the I2C master core. The NEXT_ADDR signal can be used to address registers or RAM outside the core for retrieval and storage of I2C bus transactions.

When starting an I2C, the GO, R_WRN, and SLAVE_ADDR signals must be present at the I2C core 20.0 ns before the rising edge of the SCL clock.

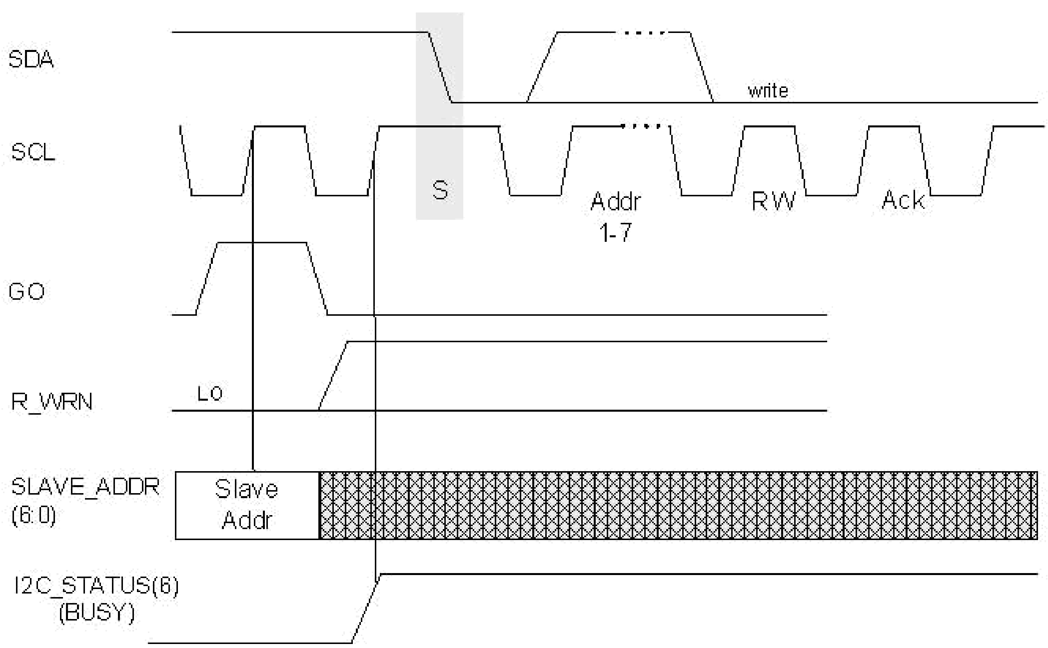


Figure 15 • **Starting an I2C Transaction**

During the I2C write, READ_NOW pulses to request the next data word. The SCL signal is used to capture the data presented by the backend on the DATA_IN bus.

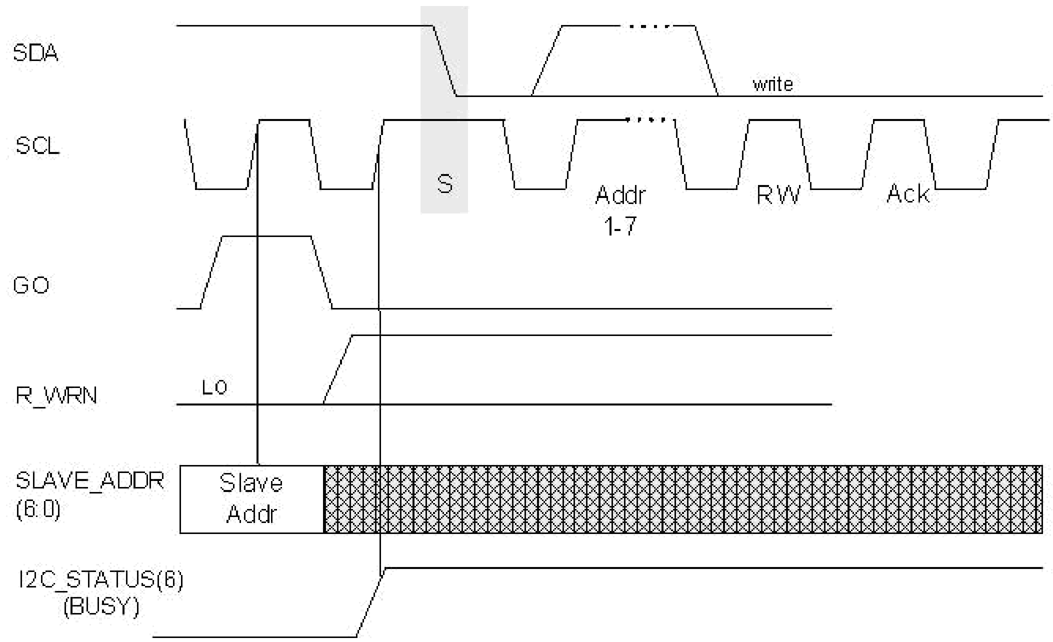


Figure 16 • **Timing Diagram for I2C Master Write Operation**

For a read operation from the slave, the DATA_VALID signal indicates to the backend when a valid data byte has been received by the I2C master core. DATA_VALID is valid for 10 µs in STD speed mode and 2.5 µs in FAST speed mode. The DATA_OUT bus is valid for 20 µs in STD speed mode and 5.0 µs in FAST speed mode.

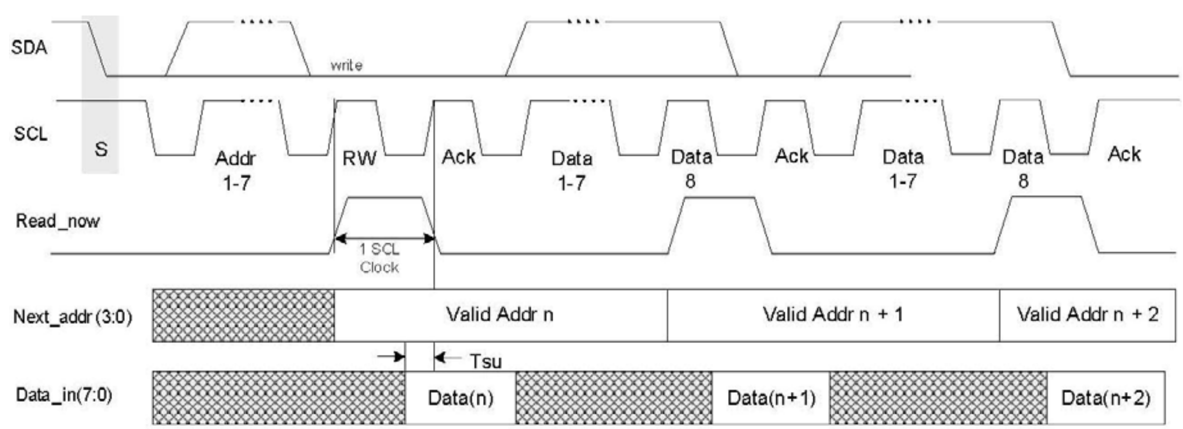


Figure 17 • **Timing Diagram for I2C Master Read Operation**

Actel is the leader in low-power and mixed-signal FPGAs and offers the most comprehensive portfolio of system and power management solutions. Power Matters. Learn more at www.actel.com.

Actel Corporation
2061 Stierlin Court
Mountain View, CA
94043-4655
USA
Phone 650.318.4200
Fax 650.318.4600

Actel Europe Ltd.
River Court, Meadows Business Park
Station Approach, Blackwater
Camberley Surrey GU17 9AB
United Kingdom
Phone +44 (0) 1276 609 300
Fax +44 (0) 1276 607 540

Actel Japan
EXOS Ebisu Building 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150 Japan
Phone +81.03.3445.7671
Fax +81.03.3445.7668
http://jp.actel.com

Actel Hong Kong
Room 2107, China Resources Building
26 Harbour Road
Wanchai, Hong Kong
Phone +852 2185 6460
Fax +852 2185 6488
www.actel.com.cn

51900192-1/4.09