CoreSPI v3.0

Handbook

Actel Corporation, Mountain View, CA 94043

© 2007 Actel Corporation. All rights reserved.

Printed in the United States of America

Part Number: 51700089-0

Release: June 2007

No part of this document may be copied or reproduced in any form or by any means without prior written consent of Actel.

Actel makes no warranties with respect to this documentation and disclaims any implied warranties of merchantability or fitness for a particular purpose. Information in this document is subject to change without notice. Actel assumes no responsibility for any errors that may appear in this document.

This document contains confidential proprietary information that is not to be disclosed to any unauthorized person without prior written consent of Actel Corporation.

Trademarks

Actel and the Actel logo are registered trademarks of Actel Corporation.

Adobe and Acrobat Reader are registered trademarks of Adobe Systems, Inc.

All other products or brand names mentioned are trademarks or registered trademarks of their respective holders.

Table of Contents

Introduction

Core Overview

The Serial Peripheral Interface (SPI) bus allows high-speed synchronous serial data transfer between microprocessors/ microcontrollers and peripheral devices. CoreSPI implements SPI and can operate as either a Master or a Slave. When operating in Master mode, the core generates the serial data clock (m_sck) and selects the Slave device that will be accessed. When operating in Slave mode, another Master device generates s_sck and activates the Slave select input of the core in order to communicate.

The SPI Slave was carefully designed to provide the most reliable communication possible. To achieve a design with a single clock domain, the s_sck line is sampled and synchronized with the system clock. This has the added benefit of increased tolerance on s_sck line noise and glitches.

The design is fully synchronous and has one clock domain, the system clock. This leads to more reliable, trouble-free synthesis and implementation of the core. No special technology features are used, so the source HDL code can be easily transferred to any technology.

Other features incorporated in the core include support for eight Slave select lines used to access up to eight devices when working as a Master, and the ability to select the transfer order of the bits (MSB first or LSB first), which saves valuable time by not implementing this function in software. Figure 1 below shows the CoreSPI block diagram.

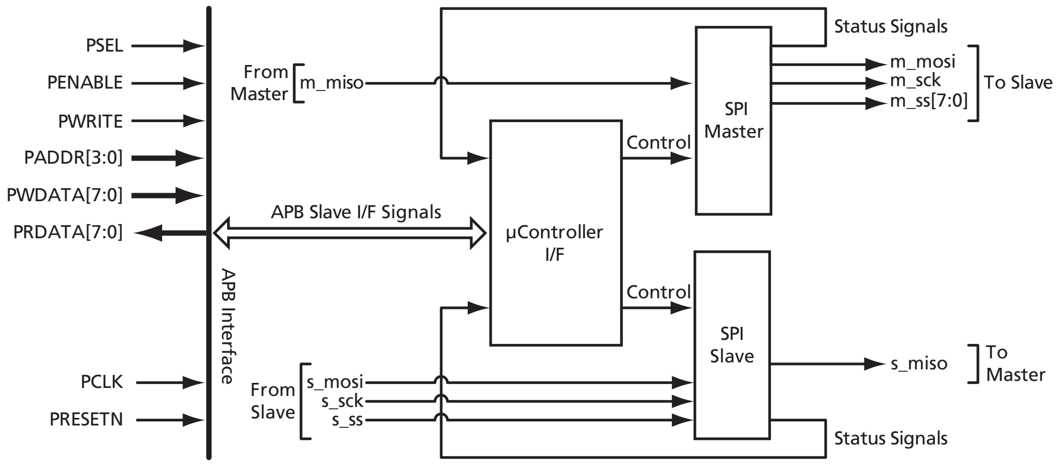


Figure 1 · CoreSPI Block Diagram

Key Features

- Full-duplex, synchronous, 8-bit serial data transfer
- High bit rates
- Master or Slave mode
- Bit rates generated in Master mode: $f_{PCLK} \div 2, \div 4, \div 8, \div 16, \div 32, \div 64, \div 128, \div 256$
- Bit rates supported in Slave mode: $f_{PCLK} \leq f_{PCLK} \div 2$
- 8 Slave select lines
- MSB-first or LSB-first data transfer
- Fully synchronous design with one clock domain

Device Utilization and Performance

CoreSPI can be implemented in any Actel device. A summary of CoreSPI utilization and performance for various devices is listed in Table 1 through Table 3. Speed grades used for layout were as follows: IGLOO™: STD, Fusion: –2, ProASIC®3/E: –2, ProASIC^{PLUS}®: STD, Axcelerator®: –2, RTAX-S: –1.

Table 1 · CoreSPI Device Utilization and Performance (combined mode)

Family	Tiles			Utilization		Performance
	Sequential	Combinatorial	Total	Device	Total	
Fusion	109	221	330	AFS600-2	2.4%	131 MHz
IGLOO	109	218	327	AGL600-STD	2.4%	80 MHz
ProASIC3/E	109	221	330	M7A3P250-2	5.4%	134 MHz
ProASIC^{PLUS}	109	300	409	APA075-STD	13.3%	85 MHz
Axcelerator	107	152	259	AX250-2	6.1%	190 MHz
RTAX-S	107	152	259	RTAX250S-1	6.1%	136 MHz

Note: Data in this table were achieved using typical synthesis and layout settings. Top-level parameters/ generics were set as follows: MASTER_MODE = 1, SLAVE_MODE = 1.

Table 2 · CoreSPI Device Utilization and Performance (Master-only mode)

Family	Tiles			Utilization		Performance
	Sequential	Combinatorial	Total	Device	Total	
Fusion	81	182	263	AFS600-2	1.9%	131 MHz
IGLOO	81	181	262	AGL600-STD	1.9%	86 MHz
ProASIC3/E	81	182	263	M7A3P250-2	4.3%	126 MHz
ProASIC^{PLUS}	78	237	315	APA075-STD	10.3%	87 MHz
Axcelerator	79	127	206	AX250-2	4.9%	188 MHz
RTAX-S	79	127	206	RTAX250S-1	4.9%	138 MHz

Note: Data in this table were achieved using typical synthesis and layout settings. Top-level parameters/ generics were set as follows: MASTER_MODE = 1, SLAVE_MODE = 0.

Table 3 · CoreSPI Device Utilization and Performance (Slave-only mode)

Family	Tiles			Utilization		Performance
	Sequential	Combinatorial	Total	Device	Total	
Fusion	94	100	194	AFS600-2	1.4%	196 MHz
IGLOO	94	102	196	AGL600-STD	1.4%	107 MHz
ProASIC3/E	94	100	194	M7A3P250-2	3.2%	202 MHz
ProASIC^{PLUS}	50	134	184	APA075-STD	6.0%	144 MHz
Axcelerator	50	66	116	AX250-2	2.8%	292 MHz
RTAX-S	60	66	116	RTAX250S-1	2.8%	207 MHz

Note: Data in this table were achieved using typical synthesis and layout settings. Top-level parameters/ generics were set as follows: MASTER_MODE = 0, SLAVE_MODE = 1.

1

Functional Block Descriptions

CoreSPI is primarily a state machine used to interface the Serial Peripheral Interface to the Advanced Peripheral Bus (APB) interface. Figure 1 on page 5 shows the block diagram for CoreSPI. The following sections describe each block's function.

APB Interface

CoreSPI supports the APB interface compatible with the Actel Core8051s and CoreMP7 processor cores and the CoreABC generic state machine control core (for simple FSM applications). This interface provides direct access to the core's internal registers (see "Register Map" on page 15).

µController Interface

The µController interface block is used to translate APB read and write commands to SFR (special function register) reads and writes, as well as to provide control logic for the SPI Master and SPI Slave blocks.

SPI Master

The SPI Master block is the state machine that keeps track of, and updates, the status of the CoreSPI Master functions. It contains an SPI interface to the Slave it is controlling, including a clock line (sck) and a data line (ss).

SPI Slave

The SPI Master block is the state machine that keeps track of, and updates, the status of the CoreSPI Slave functions. It contains an SPI interface to the Master that is controlling it, including a clock line (sck) and a data line (ss).

2

Tool Flows

Licenses

CoreSPI is licensed in three ways. Depending on your license, tool flow functionality may be limited.

Evaluation

Pre-compiled simulation libraries are provided, allowing the core to be instantiated in CoreConsole and simulated within Actel Libero® Integrated Design Environment (IDE) as described below. The design cannot be synthesized, as source code is not provided.

Obfuscated

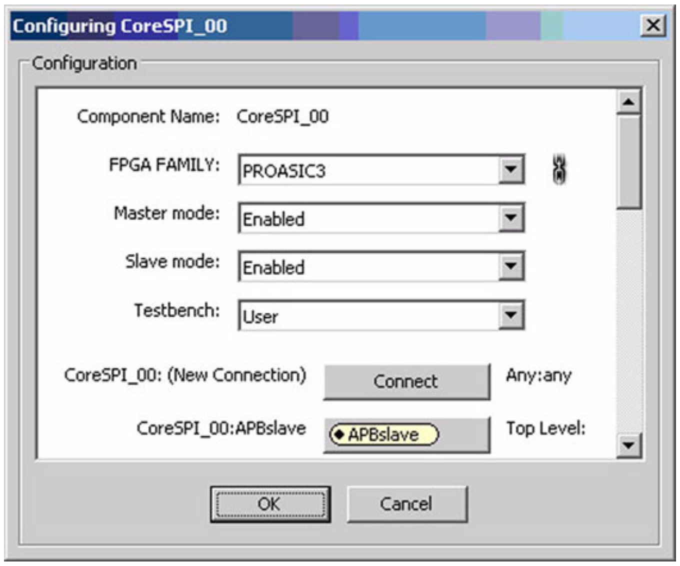
Complete RTL code is provided for the core, allowing the core to be instantiated with CoreConsole, and simulation, synthesis, and layout to be performed in Libero IDE. The RTL code for the core is obfuscated.[1]

RTL

Complete RTL source code is provided for the core and testbenches.

CoreConsole

CoreSPI is preinstalled in the CoreConsole IP Deployment Platform (IDP). To use the core,[2] **simply drag it from the IP core list into the main window**. The core can then be configured using the configuration GUI within CoreConsole, as shown in Figure 2-1. The CoreConsole project can be exported to Libero IDE at this point, providing access only to CoreSPI; or other IP blocks can be interconnected, allowing the complete system to be exported from CoreConsole to Libero IDE.



Component Name: CoreSPI_00
FPGA FAMILY: PROASIC3
Master mode: Enabled
Slave mode: Enabled
Testbench: User
CoreSPI_00: (New Connection) Any:any
CoreSPI_00:APBslave Top Level:

Figure 2-1 · CoreConsole Configuration of CoreSPI

1. *Obfuscated means the RTL source files have had formatting and comments removed, and all instance and net names have been replaced with random character sequences.*
2. *A CoreSPI license is required to generate the design for export to Libero IDE for simulation and synthesis.*

Importing into Actel Libero IDE

After generating and exporting the core from CoreConsole, the core can be imported into Libero IDE. Create a new project in Libero IDE and import the CoreConsole project from the *LiberoExport* directory. Libero IDE will then install the core and the selected testbenches, along with constraints and documentation, into its project.

Note: If two or more DirectCores are required, they can both be included in the same CoreConsole project and imported into Libero IDE at the same time.

Simulation Flows

To run simulations, the required testbench flow must be selected within CoreConsole and **Save & Generate** must be run from the Generate pane. Select the required testbench through the Core Testbench Configuration GUI. Two simulation testbenches are supported with CoreSPI:

- Simple CoreSPI user testbench (VHDL and Verilog)
- Full CoreSPI verification testbench (VHDL and Verilog)

When CoreConsole generates the Libero IDE project, it will install the appropriate testbench files. To run either the simple application or the full verification environment, simply set the design root to the CoreSPI instantiation in the Libero IDE design hierarchy, and click the **Simulation** icon in the Libero IDE Design Flow window. This will invoke Model*Sim*® and automatically run the simulation.

Synthesis in Actel Libero IDE

Having set the design root appropriately, click the **Synthesis** icon in Libero IDE. The synthesis window appears, displaying the Synplicity® project. Set Synplicity to use the Verilog 2001 standard if Verilog is being used. To run synthesis, click the **Run** icon.

Place-and-Route in Actel Libero IDE

Having set the design root appropriately and run Synthesis, click the **Layout** icon in Libero IDE to invoke Designer. CoreSPI requires no special place-and-route settings.

Interface Description

Parameters

CoreSPI has parameters (Verilog) and generics (VHDL) for configuring the RTL code (Table 3-1). All parameters and generics are integer types. These parameters/generics are mapped to configuration options in the CoreConsole configuration window.

Table 3-1 · CoreSPI Configuration Parameters

Parameter	Values	Description
FAMILY	0 to 99	Must be set to match the supported FPGA family: 11 – Axcelerator 12 – RTAX-S 14 – ProASIC$^{\underline{PLUS}}$ 15 – ProASIC3 16 – ProASIC3E 17 – Fusion 20 – IGLOO 21 – IGLOOe
USE_MASTER	0, 1	If 1, SPI Master logic is instantiated; otherwise, SPI Master logic is omitted.
USE_SLAVE	0, 1	If 1, SPI Slave logic is instantiated; otherwise, SPI Slave logic is omitted.

Signals

The port signals for the CoreSPI macro are defined in Table 3-2 on page 12 and illustrated in Figure 3-1. All signals are designated either “Input” (input-only) or “Output” (output-only). The combined Master and Slave implementation of CoreSPI has 42 I/O signals.

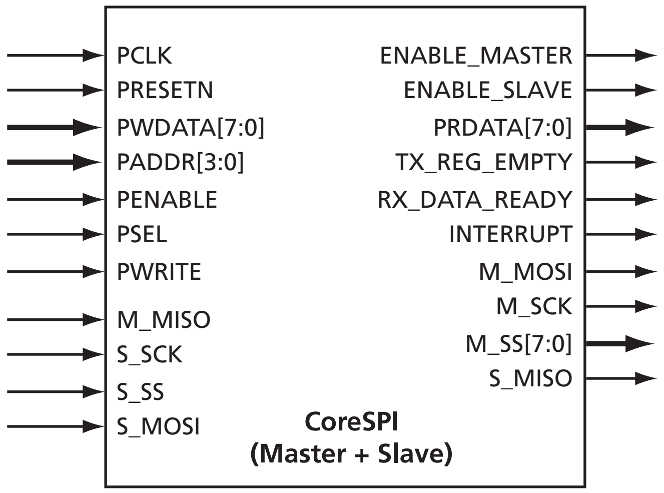


Figure 3-1 · CoreSPI I/O Signal Diagram

Table 3-2 · CoreSPI I/O Signals

Name	Type	Description
PCLK	Input	APB system clock; reference clock for all internal logic
PRESETN	Input	APB active low asynchronous reset
PWDATA[7:0]	Input	APB write data
PRDATA[7:0]	Output	APB read data
PADDR[3:0]	Input	APB address bus. This port is used to address internal CoreSPI registers.
PENABLE	Input	APB strobe. Indicates the second cycle of an APB transfer.
PSEL	Input	APB Slave select. Selects CoreSPI for reads or writes on the APB.
PWRITE	Input	APB write/read select signal. If HIGH (logic 1), a write will occur when an APB transfer to CoreSPI takes place. If LOW (logic 0), a read from CoreSPI will occur.
ENABLE_MASTER	Output	Master mode enable static signal. If active, CoreSPI is operating in Master mode.
ENABLE_SLAVE	Input	Slave mode enable static signal. If active, CoreSPI is operating in Slave mode.
TX_REG_EMPTY	Output	Microcontroller interface interrupt output: transmit register is empty. This pin will become active (logic 1) when the transmit data register is empty and can be written with the next character to be transmitted.
RX_DATA_READY	Output	Microcontroller interface interrupt output: received data is ready. This pin will become active (logic 1) when the received data register contains recently received data and must be read.
INTERRUPT	Output	Microcontroller interface interrupt output. If interrupts are enabled (by software control), this pin will become active (logic 1) when either tx_reg_empty or rx_data_ready is active.
M_MISO	Input	Master input / Slave output; serial data from external Slave to internal Master
M_MOSI	Output	Master output / Slave input; serial data from internal Master to external Slave
M_SCK	Output	Master serial clock reference from internal Master to external Slave; used as reference for all ports with "m_" prefix
M_SS[7:0]	Output	Master Slave select lines (active low); used for selecting up to eight external Slave devices
S_SCK	Input	Slave serial clock reference from external Master to internal Slave; used as reference for all ports with "s_" prefix
S_SS	Input	Slave select line (active low chip select). External Master drives this line LOW to select the internal Slave device.
S_MOSI	Input	Master output / Slave input; serial data from external Master to internal Slave
S_MISO	Output	Master input / Slave output; serial data from internal Slave to external Master

Note: All signals active high (logic 1) unless otherwise noted.

Interface Definitions

CoreSPI includes the following interfaces:

- APB (Slave) register interface
- SPI interface

APB Register Interface

The CoreSPI APB Slave interface conforms to the standard AMBA APB version 2.0 specifications. Figure 3-2 and Figure 3-3 depict typical write cycle and read cycle timing relationships relative to the system clock.

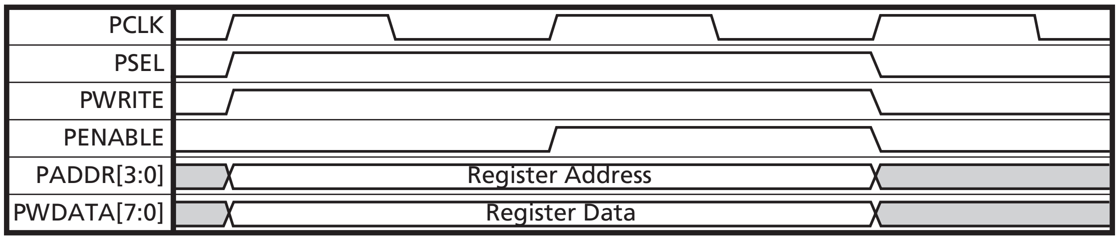


Figure 3-2 · APB Data Write Cycle

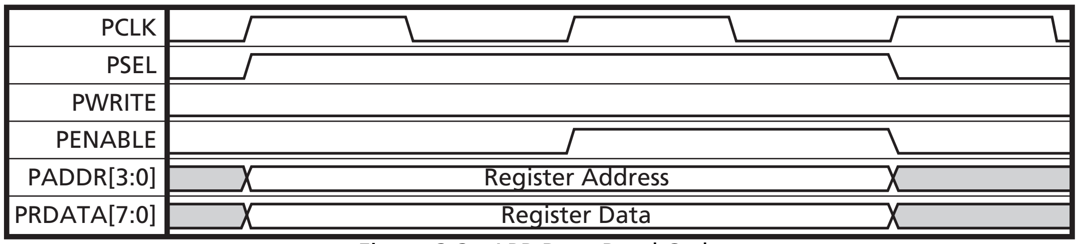


Figure 3-3 · APB Data Read Cycle

SPI Interface

Figure 3-4 shows a typical serial byte transfer with different values of CPHA and CPOL.

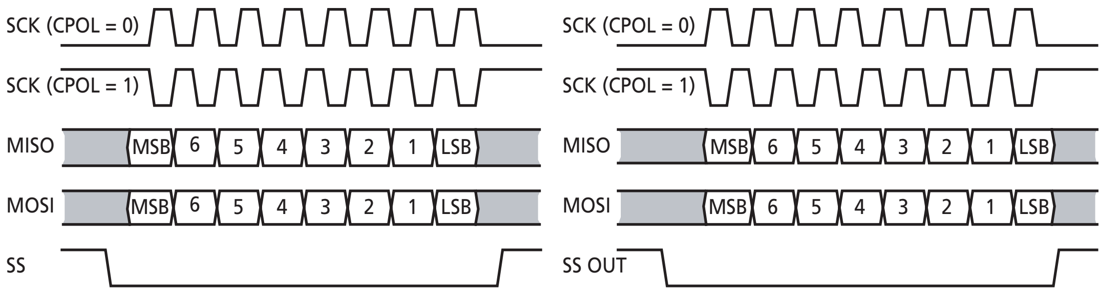


Figure 3-4 · CoreSPI Interface Signals for CPHA = 0 (left) and CPHA = 1 (right)

4

Register Map

APB Register Map

The internal register address map and reset values of each APB-accessible register for CoreSPI are shown in Table 4-1. Type designations: “R” for read-only, “W” for write-only, and “R/W” for read/write.

Table 4-1 · CoreSPI Internal Register Address Map

PADDR (hex)	Type	Reset Value (hex)	Brief Description
0x00	R/W	0x00	SPI Data Register: Read receive data register / write transmit data register to/from serial interface.
0x04	R/W	0x00	Control Register 1: Used to configure the core.
0x08	W	0x00	Control Register 2: Used to enable the core and check for errors.
0x08	R	0x00	Status Register: Read-only values that yield the current status of the core.
0x0c	R/W	0x00	Slave Select Register: Used only in Master mode to select currently addressed Slave devices.

Note: Each of the above registers is 8 bits wide.

Register Descriptions

The following sections and tables detail the APB-accessible registers within CoreSPI.

SPI Data Register

When the SPI Data Register is read, the contents of the RX Data Register are returned. The contents of the RX Data Register should be read as soon as possible when the Rx_Data_Ready status bit is activated so that a newly received character does not overwrite it.

Writing to the SPI Data Register accesses the TX Data Register and initiates data transmission when SPI is enabled. The TX Data Register should be written when the Tx_Register_Empty status bit is activated; otherwise, an unsent character is overwritten.

Control Register 1

A CPU can read from and write to this 8-bit, APB-addressable SFR via the APB Slave interface. Table 4-2 describes the function of each bit of the Control Register.

Table 4-2 · Control Register 1

Bit(s)	Name	Function
7	interrupt_enable	Interrupt enable signal When this bit is set to logic 1, the interrupt pin will become logic 1 when the TX_REG_EMPTY or RX_DATA_READY internal flag is activated. When this bit is cleared to logic 0, the interrupt pin will remain at logic 0. The operation of the TX_REG_EMPTY and RX_DATA_READY interrupt pins is not affected.
6	mode_select	Master/Slave mode select Selects whether CoreSPI is operating in Master or Slave mode. This bit sets CoreSPI to Master mode when set to logic 1 and to Slave mode when reset to logic 0. If Master mode or Slave mode is disabled, this bit does not affect core operation, but is still read/writable.
5	order	Order of data transfer Sets the order of data transfer. When this bit is logic 1, the LSB is transmitted first. When this bit is logic 0, the MSB is transmitted first.
4	cpha	Sets the clock phase. When this bit is logic 1, the serial data is delayed 90° in relation to M_SCK and S_SCK. Refer to Figure 3-1 on page 11 for details. Note that in Figure 3-1 on page 11, SCK refers to either M_SCK or S_SCK. Likewise, MISO, MOSI, and SS refer to M_MISO, M_MOSI, and M_SS[7:0], respectively, for Master mode; and S_MISO, S_MOSI, and S_SS, respectively, for Slave mode operation.
3	cpol	Sets the clock polarity. When this bit is logic 1, the M_SCK line remains HIGH in an idle state between frames; if this bit is 0, the M_SCK line remains LOW when idle.
2:0	scks	When operating in Master mode, bits 2 to 0 select the serial data clock frequency of the M_SCK signal, which defines the data transfer rate. In Slave mode, these bits are ignored. [2:0] SCK Frequency — [2:0] SCK Frequency 000: $f_{PCLK} \div 2$ — 100: $f_{PCLK} \div 32$ 001: $f_{PCLK} \div 4$ — 101: $f_{PCLK} \div 64$ 010: $f_{PCLK} \div 8$ — 110: $f_{PCLK} \div 128$ 011: $f_{PCLK} \div 16$ — 111: $f_{PCLK} \div 256$

Control Register 2

Bit 7 and bit 0 of this register are write-accessible via the CoreSPI APB Slave interface. The rest are either unused or reserved for read-only use. Table 4-3 describes the function of these two bits.

Table 4-3 · Control Register 2

Bit(s)	Name	Function
7	enable	SPI enable bit Used as a synchronous enable for CoreSPI. The SPI interface ports are activated when this bit is set to logic 1. When this bit is cleared to logic 0, all SPI interface ports (S_*, M_*) are either in input or inactive output states.
6:1	Unused	Unused
0	error	Error bit If this bit is at logic 1, it indicates that a character has been received before the previous character has been read from the RX data register.

Status Register

The Status Register is read-only. Table 4-4 describes each bit of the CoreSPI Status Register.

Table 4-4 · Status Register

Bit(s)	Name	Function
7	enable	Indicates that CoreSPI is enabled. If this bit is set to logic 1, CoreSPI is currently enabled.
6:4	Unused	Unused
3	busy	Indicates that the Master is busy. If this bit is at logic 1, the CoreSPI Master is currently transferring data. This status bit is used to check if the SPI Master is busy before disabling it.
2	tx_register_empty	tx_register_empty flag. New data for transmission can be written to the Transmit Data Register when this bit is at logic 1.
1	rx_data_ready	rx_data_ready flag. When this bit is at logic 1, the RX Data Register must be read before the next character is received.
0	error	If this bit is at logic 1, it indicates that a character has been received before the previous character has been read from the RX Data Register.

Slave Select Register

This register acts as a mask for activating a Slave select line. This is used only when CoreSPI is operating in Master mode. A logic 1 in a particular bit position enables communication with the SPI Slave device connected to the respective M_SS[7:0] line.

Table 4-5 outlines the bit use of the Slave Select Register.

Table 4-5 · Slave Select Register

Bit(s)	Name	Function
7	Slave select line 7	1 = Enable, 0 = Disable
6	Slave select line 6	1 = Enable, 0 = Disable
5	Slave select line 5	1 = Enable, 0 = Disable
4	Slave select line 4	1 = Enable, 0 = Disable
3	Slave select line 3	1 = Enable, 0 = Disable
2	Slave select line 2	1 = Enable, 0 = Disable
1	Slave select line 1	1 = Enable, 0 = Disable
0	Slave select line 0	1 = Enable, 0 = Disable

Software Interface Flow

A typical communication flow, using interrupts, is shown in Figure 4-1. Boxes with dashed lines are used only when CoreSPI operates in Master mode.

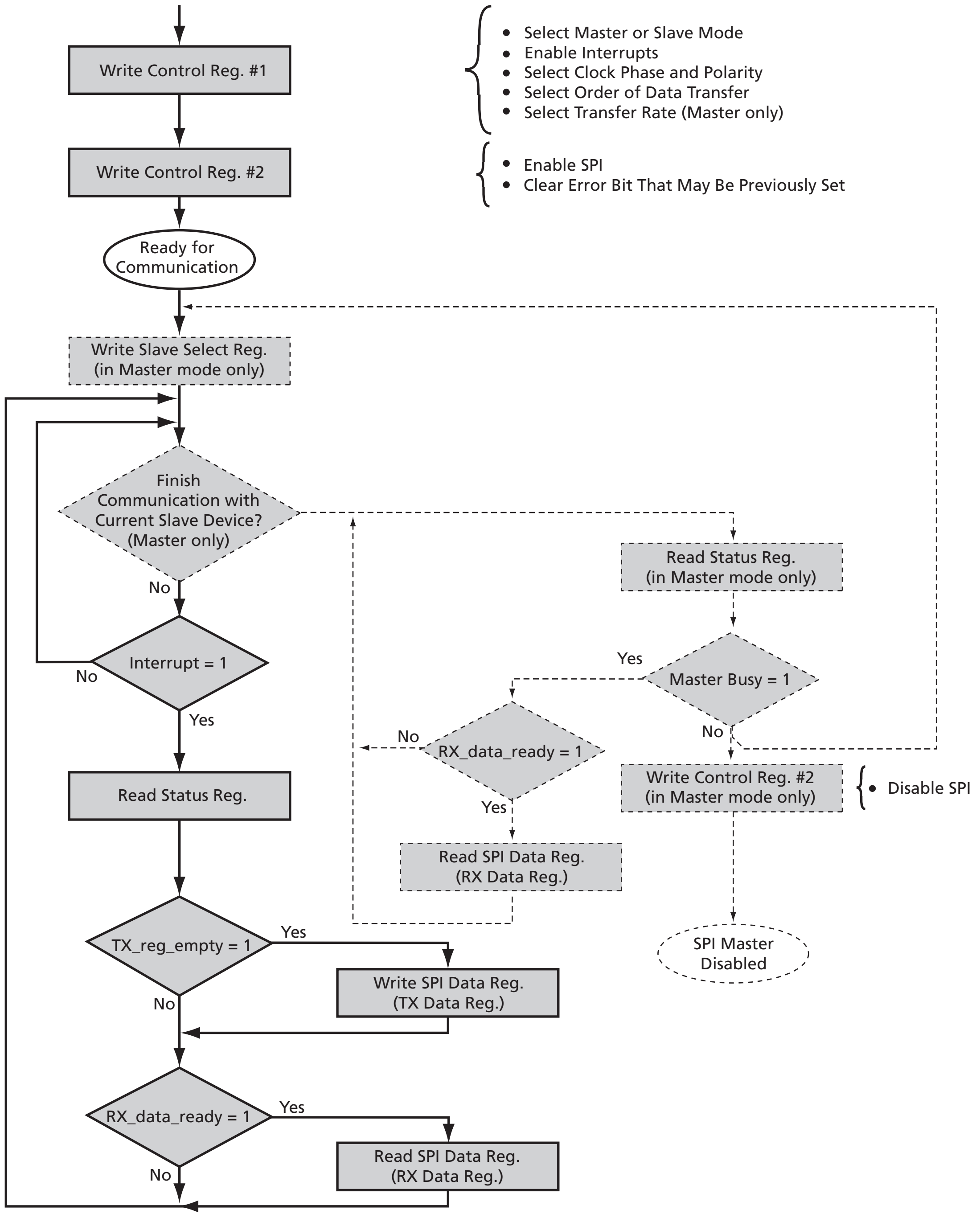

Write Control Reg. #1
Select Master or Slave Mode
Enable Interrupts
Select Clock Phase and Polarity
Select Order of Data Transfer
Select Transfer Rate (Master only)
Write Control Reg. #2
Enable SPI
Clear Error Bit That May Be Previously Set
Ready for Communication
Write Slave Select Reg. (in Master mode only)
Finish Communication with Current Slave Device? (Master only)
No
Interrupt = 1
No
Yes
Read Status Reg.
TX_reg_empty = 1
Yes
No
Write SPI Data Reg. (TX Data Reg.)
RX_data_ready = 1
Yes
No
Read SPI Data Reg. (RX Data Reg.)
Read Status Reg. (in Master mode only)
Master Busy = 1
Yes
No
RX_data_ready = 1
No
Yes
Read SPI Data Reg. (RX Data Reg.)
Write Control Reg. #2 (in Master mode only)
Disable SPI
SPI Master Disabled

Figure 4-1 · Typical Communication Flow with a Host Processor/Microcontroller

5 Testbench Operation and Modification

User Testbench

An example user testbench is included with the Evaluation, Obfuscated, and RTL releases of CoreSPI. The user testbench is provided in precompiled Model*Sim* format for the Evaluation release. The Obfuscated and RTL releases provide the precompiled Model*Sim* format and the source code for the user testbench to ease the process of integrating the CoreSPI macro into a design and verifying it. A block diagram of the example user design and testbench is shown in Figure 5-1.

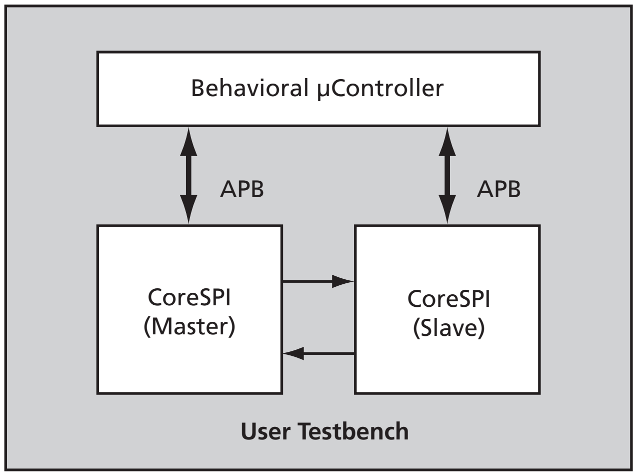


Figure 5-1 · CoreSPI User Testbench

The user testbench includes a simple example design that serves as a reference for users who want to integrate CoreSPI into their own designs. RTL source code for the example design and user testbench shown in Figure 5-1 is included in the *user* directory for all releases of the core. The example design source files and user testbench are listed in Table 5-1.

Table 5-1 · CoreSPI User Testbench RTL Files

Verilog	VHDL
coreconsole/ccproject/CORESPI/rtl/vlog/test/user/ • *tb_user_corespi.v*	*coreconsole/ccproject/CORESPI/rtl/vhdl/test/user/* • *tb_user_corespi.vhd* • *corespi_pkg.vhd*

Conceptually, as shown in Figure 5-1, two instantiations of the CoreSPI macro are connected via SPI. One instantiation acts as a Master-only device (USE_MASTER = 1, USE_SLAVE = 0), and the other acts as a Slave-only device (USE_MASTER = 0, USE_SLAVE = 1). Typical use of the core is emulated by doing various reads and writes via the APB of each CoreSPI device. Data is verified by comparing sent and received data.

To run the user testbench, refer to "Simulation Flows" on page 10.

Verification Testbench

Included with the Obfuscated and RTL releases of CoreSPI is a verification testbench that verifies operation of the CoreSPI macro. A simplified block diagram of the verification testbench is shown in Figure 5-2.

The verification testbench instantiates the DUT (design under test), which is the CoreSPI macro, as well as the test vector modules that provide stimuli sources for the DUT and perform comparisons for expected values throughout the simulation process. A procedural testbench controls each module and applies the sequential stimuli to the DUT.

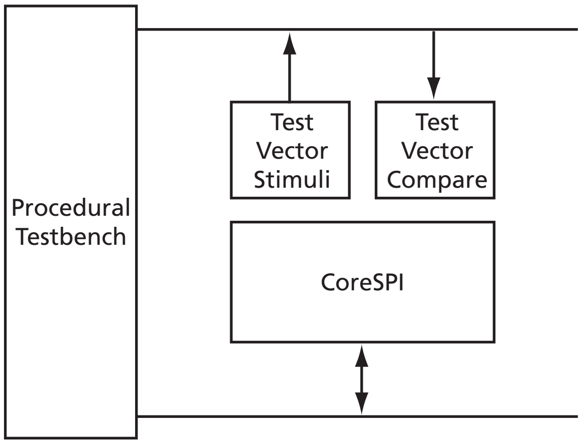


Figure 5-2 · Simplified CoreSPI Verification Testbench Block Diagram

The source code for the verification testbench is only available with the CoreSPI RTL release. A compiled Model*Sim* simulation is available with the Obfuscated and RTL releases.

To run the verification testbench, refer to "Simulation Flows" on page 10.

Verification Tests

CoreSPI is verified through various tests that stimulate program control words, transmission and reception sequences, and loopback tests. CoreSPI is verified for the Master and Slave combined core, as well as for the Slave-only and Master-only implementations. Behavioral microcontroller sequences (APB writes and reads) are used in the verification testbench to emulate the behavior of controlling CoreSPI via internal register reads and writes and by monitoring of the various interrupt flags.

System Operation

This chapter provides various hints to ease the process of implementing CoreSPI into your own design.

Use with CoreMP7

CoreSPI can also be used with CoreMP7, the Actel soft IP version of the popular ARM7TDMI-S™ microprocessor that has been optimized for the M7 Fusion Flash-based FPGA devices. To create a design using CoreMP7, internal flash memory, and CoreSPI, use CoreConsole IDP. Refer to the CoreConsole documentation for information on creating your CoreMP7-based design. Figure 6-1 gives an example design.

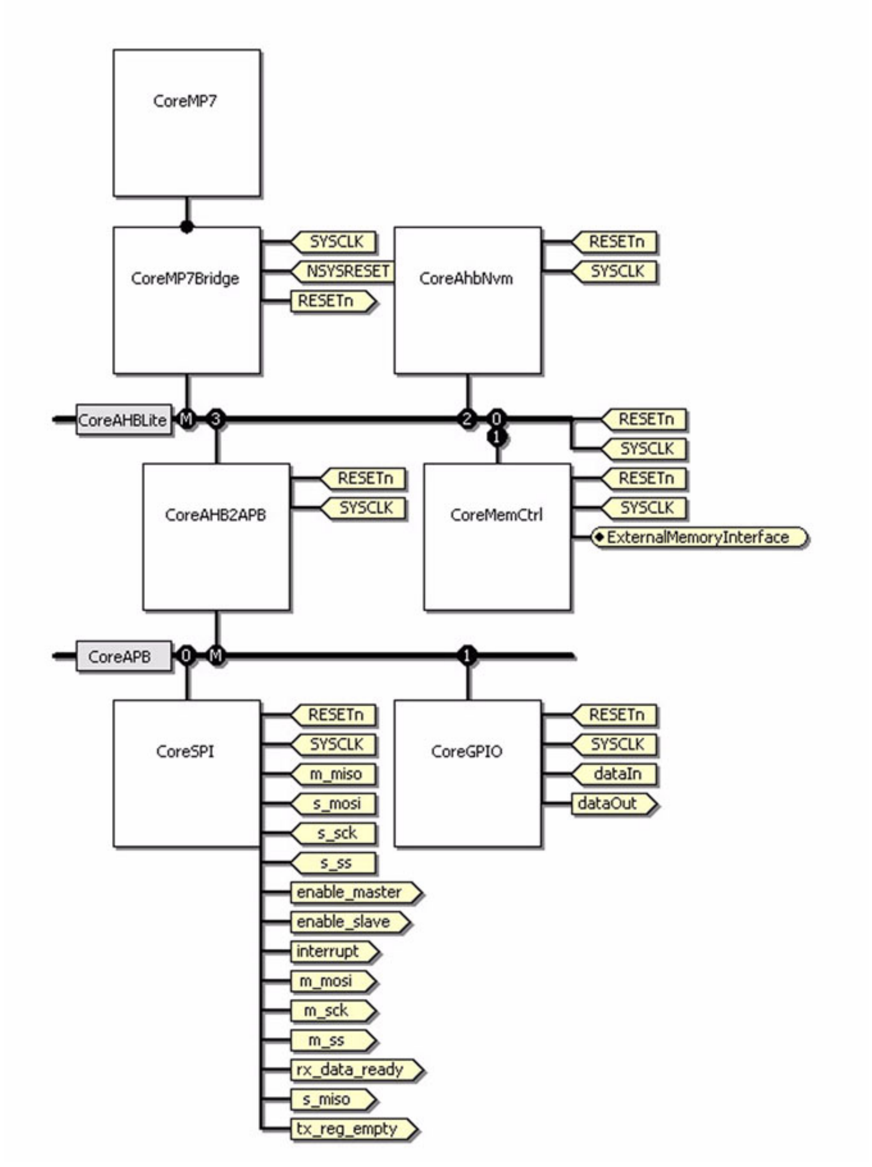


Figure 6-1 · Example System Using CoreMP7 and CoreSPI

Use with Core8051s

CoreSPI can also be used with Core8051s. An example FPGA design using Core8051s and CoreSPI is shown in Figure 6-2.

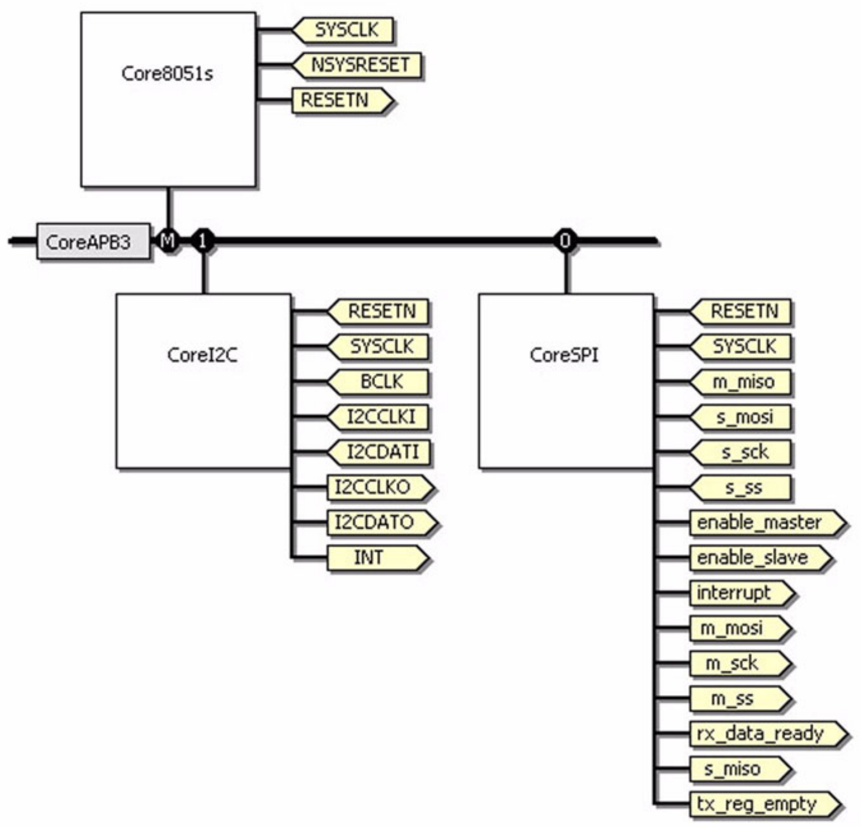


Figure 6-2 · Example System Using Core8051s and CoreSPI

Use with CoreABC

CoreSPI can also be used with CoreABC. An example FPGA design using CoreABC and CoreSPI is shown in Figure 6-3. CoreABC allows a simple set of APB read and write cycles that can be used to configure CoreSPI and then read and compare the analog values to turn the digital outputs on and off.

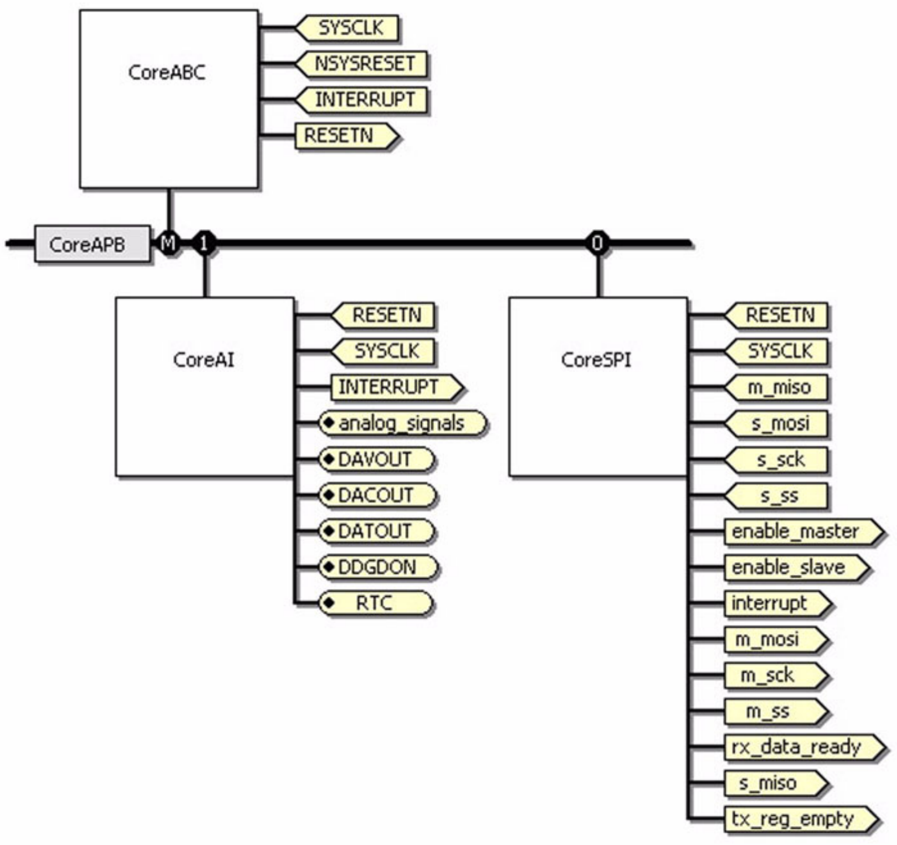


Figure 6-3 · Example System Using CoreABC and CoreSPI

A

Testbench Support Routines

The verification and user testbenches for the CoreSPI macro make use of various support routines, both in VHDL and Verilog. The various support routines are described in this appendix for the VHDL and Verilog testbenches.

VHDL Support

The VHDL support routines (procedures and functions) are provided within a package. The support routines are referenced from within the verification and user testbenches, via *library* and *use* clauses. To include these routines in a custom testbench, add the following two lines:

```
library CoreSPI_lib;
use CoreSPI_lib.CoreSPI_pkg.all;
```

The following function simply converts hexadecimal bit_vector format into std_logic_vector format:

```
function hx (b: bit_vector) return std_logic_vector;
```

For example:

```
A <= hx(x”0123456789abcdef”);
```

The following overloaded procedure checks the given signal or vector against the expected value and prints an error to the screen if a mismatch occurs:

```
procedure checksig (
  d:          std_logic;
  sig_name:   string;
  v:          bit;
  ERRCNT:     inout integer
);
procedure checksig (
  d:          std_logic_vector;
  sig_name:   string;
  v:          bit_vector;
  ERRCNT:     inout integer
);
```

The first parameter of the checksig procedure is the actual signal (std_logic or std_logic_vector) to check. The second parameter is the ASCII string representation of the signal to print to the screen in the event of a signal value mismatch. The third parameter is the expected value (bit or bit_vector) to check the actual signal value against. The fourth parameter is an integer that represents the error count to keep track of any signal value mismatches. For example:

```
checksig(SINGLE_BIT_SIG, "SINGLE_BIT_SIG", '0', simerrors);
checksig(VECTOR_SIG,"VECTOR_SIG",
x"0123456789abcdef0123456789abcdef",simerrors);
```

The first line above checks that the value of the signal SINGLE_BIT_SIG is 0 at the current simulation time, and if it is not, the checksig procedure increments the value of the variable integer simerrors by one. The second line above checks that the value of the signal VECTOR_SIG is 0x0123456789abcdef0123456789abcdef at the current simulation time, and if it is not, the checksig procedure increments the value of the variable integer simerrors by one. A printf procedure is included with the verification and user testbenches that supports printing string, std_logic, boolean, integer, and std_logic_vector types. The printf procedure included is similar to the printf function in the C language. However, the format is slightly different.

For example:

```
printf(“Hello World Decimal Vec %d Hex Vec %x String: %s”,
fmt(slv)&fmt(slv)&fmt(str1));
```

prints to the simulation transcript (slv is a 4-bit wide standard_logic_vector, and str1 is the string “somestring”):

```
Hello World Decimal Vec 15 Hex Vec F String: somestring
```

Verilog Support

The Verilog versions of the testbenches make use of the following task, which is included within the top-level module of the verification and user testbenches. The checksig task is identical in functionality to the checksig procedure included with the VHDL testbenches. It is used to check signals against expected values at the current simulation time. The task argument list is shown below:

```
task checksig;
input [127:0]   d;
input [8*17:1]  sig_name;
input [127:0]   v;
```

The first parameter of the checksig task is the actual signal (up to 128 bits wide) to check. The second parameter is the ASCII string representation of the signal to print to the screen in the event of a signal value mismatch. The third parameter is the expected value (up to 64 bits wide) to check the actual signal value against. The task uses a global integer simerrors, declared in the top-level testbench, to keep track of the number of signal value mismatches, if any. For example:

```
checksig(SINGLE_BIT_SIG, "SINGLE_BIT_SIG", 0);
checksig(VECTOR_SIG, "VECTOR_SIG",
128’h0123456789abcdef0123456789abcdef);
```

The first line above checks that the value of the signal SINGLE_BIT_SIG is 0 at the current simulation time, and if it is not, the checksig task increments the value of the global integer simerrors by one. The second line above checks that the value of the signal VECTOR_SIG is 0x0123456789abcdef0123456789abcdef at the current simulation time, and if it is not, the checksig task increments the value of the global integer simerrors by one.

B Product Support

Actel backs its products with various support services including Customer Service, a Customer Technical Support Center, a web site, an FTP site, electronic mail, and worldwide sales offices. This appendix contains information about contacting Actel and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From Northeast and North Central U.S.A., call **650.318.4480**
From Southeast and Southwest U.S.A., call **650. 318.4480**
From South Central U.S.A., call **650.318.4434**
From Northwest U.S.A., call **650.318.4434**
From Canada, call **650.318.4480**
From Europe, call **650.318.4252** or **+44 (0) 1276 401 500**
From Japan, call **650.318.4743**
From the rest of the world, call **650.318.4743**
Fax, from anywhere in the world **650.318.8044**

Actel Customer Technical Support Center

Actel staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions. The Customer Technical Support Center spends a great deal of time creating application notes and answers to FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Actel Technical Support

Visit the Actel Customer Support website (www.actel.com/custsup/search.html) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the Actel web site.

Website

You can browse a variety of technical and non-technical information on Actel's home page, at www.actel.com.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. Several ways of contacting the Center follow:

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is tech@actel.com.

Phone

Our Technical Support Center answers all calls. The center retrieves information, such as your name, company name, phone number and your question, and then issues a case number. The Center then forwards the information to a queue where the first available application engineer receives the data and returns your call. The phone hours are from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. The Technical Support numbers are:

650.318.4460
800.262.1060

Customers needing assistance outside the US time zones can either contact technical support via email (tech@actel.com) or contact a local sales office. Sales office listings can be found at www.actel.com/contact/offices/index.html.

Index

For more information about Actel's products, visit our website at http://www.actel.com

Actel Corporation *• 2061 Stierlin Court • Mountain View, CA 94043 USA*
Customer Service: 650.318.1010 • Customer Applications Center: 800.262.1060

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 Bldg. 4F • 1-24-14 Ebisu Shibuya-ku • Tokyo 150 • Japan*
Phone +81.03.3445.7671 • Fax +81.03.3445.7668 • www.jp.actel.com

Actel Hong Kong *• Suite 2114, Two Pacific Place • 88 Queensway, Admiralty Hong Kong*
Phone +852 2185 6460 • Fax +852 2185 6488 • www.actel.com.cn

51700089-0 /6.07