



Features

- Serial peripheral interface (SPI) peripheral, used as either a master or slave device
- Master devices can interface with up to 16 slave devices
- Programmable delay slot from enable active to bus activity
- SPI slave device can interface with an off-chip SPI master device via a tri-state bus
- Programmable word size from 1 to 16 bits
- Configurable clock phase and polarity

General Description

The Nios™ CPU SPI peripheral can be used as either a master or slave device and allows the Nios CPU to communicate with one or more external devices over an SPI bus. The Nios CPU controls and communicates with the SPI peripheral through five memory-mapped, 16-bit registers accessed by a standard Nios peripheral bus connection.

The Nios CPU SPI peripheral is fully configurable at hardware compile time. The SPI peripheral is generated by an SOPC Builder MegaWizard® Plug-In and, when configured as a master device, can interface with up to 16 slave devices. Several important attributes of the SPI bus (e.g., data size, clock speed, etc.) are set in the wizard.

The SPI is a simple, industry-standard communications interface that does not have specifications or a standards organizing group. Thus, there are no licensing requirements. Because of its simplicity, the SPI is commonly used in embedded systems. Many semiconductor manufacturers sell a variety of sensor, conversion, and control devices that use the SPI.

The Nios CPU SPI peripheral extends the conventional point-to-point protocol by allowing more than one external SPI peripheral to share the same `mis0`, `mosi`, and `sclk` pins. Each SPI peripheral instantiated within a Nios system has at least four interface signals (i.e., I/O pins on the Altera® APEX™, ACEX™, or FLEX® device). The interface pins are `mis0`, `mosi`, `sclk`, and `ss_n`. (Master devices interfacing with multiple SPI slave devices will have one `ss_n` signal for each slave device.) The function of these signals depends on whether the SPI peripheral is operating as a master or slave device.

Table 1 lists pin names, numbers, and types for both master and slave configurations.

Pin Name	Polarity	Master Device		Slave Device	
		Number of Pins	Type	Number of Pins	Pin Type
<code>miso</code>	High	1	Input	1	Output
<code>mosi</code>	High	1	Output	1	Input
<code>sclk</code>	–	1	Output	1	Input
<code>ss_n</code>	Low	1-16	Output	1	Input

The `miso` (master in, slave out) signal carries synchronous data from the slave to the master device. The `mosi` (master out, slave in) signal carries synchronous data from the master to the slave device. The `sclk` signal is the shared clock signal, driven by the master, which synchronizes all data transfers. There is one `ss_n` signal for each SPI slave device. The `ss_n` signal is an active-low slave input/master output pin. Slave devices do not respond to transactions unless their `ss_n` input signal is driven low.



Typical SPI peripherals set their `miso` output pins—when not selected—to high impedance. The Nios SPI slave device drives an undefined value on its `miso` output pin when not selected. If a Nios SPI slave device is connected to an external (off-chip) SPI master device, the select input can be used to control a tri-state pin outside the Nios system module. This option is necessary when multiple slave devices share the same SPI bus.

Functional Description

The SPI peripheral consists of two user-visible components:

- A memory-mapped register space
- SPI bus interface pins

The register control bits determine when data can be read from and written to the memory-mapped registers. All data is transmitted via the SPI bus interface pins, `miso`, `mosi`, `sclk`, and `ss_n`.

Master Mode Operation

The SPI peripheral operates as a master device if it is configured as such by the wizard. In general, the SPI protocol does not support multi-master systems. The master initiates all data transactions, and every transaction is both a receive and transmit operation. The master device transmits a new data bit on the `mosi` pin and the slave device drives a new data bit on the `miso` pin on each active clock edge.

The SPI peripheral transmitter logic consists of an n -bit (where n is a value from 1 to 16) `txdata` transmit holding register and an n -bit transmit shift register (the number of data bits is configured at hardware compile time). The Nios CPU starts an SPI transaction by writing a transmit-data value to the `txdata` register. The transmit shift register directly feeds the `mosi` data pin. Data is shifted out to `mosi` either least significant bit (LSB) first or most significant bit (MSB) first, depending on the configuration settings. The transmit shift register is loaded automatically from the `txdata` register whenever a serial transmit shift operation is not currently in process.



The transmit shift register and the `txdata` register provide double buffering during data transmission; i.e., the Nios CPU can write a new value into the `txdata` register while the previously written character is being shifted out of the transmit shift register.

The SPI peripheral receiver logic consists of an n -bit (where n is a value from 1-16) `rxdata` receiver holding register and an n -bit receiver shift register (the number of data bits is configured at hardware compile time). The `rxdata` receiver holding register is read directly by the Nios CPU. The receiver shift register is fed directly by the `miso` data pin. The `rxdata` holding register is loaded from the receiver shift register automatically every time data is fully received.



The receiver shift register and the `rxdata` register provide double buffering during data receiving; i.e., the `rxdata` register can hold a previously received data byte while the subsequent data byte is being shifted into the receiver shift register.

The Nios CPU can monitor the status of a master write operation by reading the `trdy`, `tmt`, and `toe` bits in the status register. Similarly, the Nios CPU can monitor the status of a slave read operation by reading the `roe` and `rrdy` bits in the status register.

Slave Mode Operation

The SPI peripheral slave device behaves similarly to the master device, except it cannot initiate a data transaction.

Before a transaction begins, the slave device polls its `ss_n` pin waiting for the pin to be driven low, which indicates that the device is being addressed by an SPI master device. The SPI slave device will immediately begin sending the transmit shift register contents to the `mis0` pin. The slave device will also read the receiver shift register, simultaneously, polling for data on the `mosi` pin. Thus, a read and write transaction can be carried out simultaneously.

Register Descriptions

The Nios CPU controls the SPI peripheral by reading from and writing to memory-mapped registers inside the Altera device. The SPI peripheral has five 16-bit registers.

Register Map for SPI Master Device

Table 2 shows a register map for an SPI master device with an n -bit transmit/receive shift register operating as master devices.

Internal Address	Register Name	15	14..8	9	8	7	6	5	4	3	2	1	0
0	RxData (1)	RxData ($n-1..0$)											
1	TxData (2)	TxData ($n-1..0$)											
2	Status (3)	Unused		e	rrdy	trdy	tmt	toe	roe	Unused			
3	Control (4)	Unused		e	irrdy	itrdy	itmt	ltoe	iroe	Unused			
4	Reserved	-											
5	Select (5)	Slave Select Mask											

Notes:

- (1) Read-only value.
- (2) Write-event register. A write operation to this address causes an event in the device.
- (3) A write operation to the status register clears the following bits: `roe`, `toe`, and `e`.
- (4) Nios CPU-written control value. Can be read back at any time.
- (5) Write/read register. Bit mask for slave addressing.

Register Map for SPI Slave Device

Table 3 shows a register map for an SPI slave device with an n -bit transmit/receive shift register operating as slave devices.

Table 3 .Register Map for SPI Slave Device

Internal Address	Register Name	15	14..8	9	8	7	6	5	4	3	2	1	0
0	RxData (1)	RxData ($n-1..0$)											
1	TxDatA (2)	TxData ($n-1..0$)											
2	Status (3)	Unused		e	rrdy	trdy	tmt	toe	roe	Unused			
3	Control (4)	Unused		e	irrdy	itrdy	itmt	itoe	iroe	Unused			

Notes:

- (1) Read-only value.
- (2) Write-event register. A write operation to this address causes an event in the device.
- (3) A write operation to the status register clears the following bits: roe, toe, and e.
- (4) Nios CPU-written control value. Can be read back at any time.

RxData Register (Internal Address 0)

The Nios CPU reads received data from the rxdata register. Whenever new data is fully received via the miso input, the rrdy bit in the status register is set to 1, and the data is transferred into the rxdata register. Whenever the Nios CPU reads a value from the rxdata, the rrdy bit in the status register is cleared (i.e., set to 0). If data is transferred into the rxdata register when the rrdy bit is set (i.e., if the Nios CPU has not retrieved the previous data), a receiver-overflow error occurs and the roe bit in the status register is set to 1. New data is always transferred into the rxdata register, whether or not the Nios CPU retrieved the previous data.



Writing data to the rxdat a register has no effect.

TxData Register (Internal Address 1)

The Nios CPU writes data to be transmitted directly into the `txdata` register. The Nios CPU should not write any data to the `txdata` register until the transmitter is ready for new data (as indicated by the status register bit, `trdy`). If the Nios CPU writes data to the `txdata` register when `trdy` is 0, the results are undefined. The `trdy` bit is set to 0 whenever the Nios CPU writes data into the `txdata` register. The `trdy` bit is set to 1 whenever data is transferred from the `txdata` register into the transmitter shift register, indicating an empty `txdata` holding register.

For example, assume that the SPI peripheral is idle and the Nios CPU writes data into the `txdata` holding register. During the idle phase, the `trdy` bit is set to 0, but after the data is transferred into the transmitter shift register, `trdy` is immediately set to 1. The Nios CPU begins writing a second data byte into the `txdata` register; however, the `trdy` bit is set to 0. This time, the original data byte is still in the process of being transmitted over the serial `mosi` pin. Accordingly, the `trdy` bit remains at 0 until the transaction cycle ends. When the cycle ends, the second data byte is transferred into the transmitter shift register and the `trdy` bit is again set to 1.

Status Register (Internal Address 2)

The status register consists of status bits that indicate a particular condition inside the SPI peripheral. The status register can be read at any time by the Nios CPU, and doing so does not change the value of any of the status bits. Each status bit is associated with a corresponding interrupt-enable bit in the control register, see [“Control Register \(Internal Address 3\)” on page 7](#). If a status bit’s corresponding interrupt-enable bit is true (1) at the same time that the status bit is true, an interrupt-request will be sent to the Nios CPU.

Table 4 lists the status register bits.

Data Bit	Mnemonic	Description
2..0	Unused	–
3	roe	Receiver overrun error. The roe bit is true if data is received while the rxdata register is full (i.e., while the rrdy bit is still set to 1.) If the roe error occurs, the new data overwrites the old. The roe bit is cleared (set to 0) when the Nios CPU performs a write operation to the status register.
4	toe	Transmitter overrun error. The toe bit is set when data is written to txdata while the register is still full (i.e., while the trdy bit is set to 0). If the toe error occurs, the new data is ignored. The toe bit is cleared (set to 0) when the Nios CPU performs a write operation to the status register.
5	tmt	Transmitter shift register empty. The tmt bit is set while a transaction is in progress.
6	trdy	Transmitter ready. The trdy bit is set when the txdata register is empty.
7	rrdy	Receiver ready. The rrdy bit is set when the rxdata register is full.
8	e	Error. The e bit is set if either toe or roe is set. This is a convenience for the programmer. The e bit is cleared (set to 0) whenever the Nios CPU performs a write operation to the status register.
15..9	Unused	–

Control Register (Internal Address 3)

The control register consists of data bits that control the internal operation of the SPI peripheral. The value in the control register can be read at any time by the Nios CPU. The bits in the control register are: ie, irrdy, itrdy, itmt, itoe, and iroe (bits 8, 7, 6, 5, 4, and 3, respectively).

The control bits give the Nios CPU complete flexibility to determine which, if any, internal conditions of the SPI peripheral will result in an interrupt request.

Each bit in the status register has a corresponding interrupt enable bit in the control register at the same bit position. (See “[Register Map for SPI Master Device](#)” on page 4.) For example, the rrdy bit is bit 7 of the status register, and the corresponding irrdy (interrupt-enable, receiver ready) bit is bit 7 of the control register. For each bit of the status register, an interrupt request to the Nios CPU is generated if both the status bit and its corresponding interrupt enable bit are true (1).

Select Register (Internal Address 5)

The select register sets the select bit mask for slave devices driven by the SPI bus master device. The select register is only present when the SPI peripheral is configured as a master device.

The select register can be used to address up to 16 slave devices by setting the corresponding bit. For example, to select slave device 0, the user sets bit 0 in the select register to a true value (1).

The Nios SPI master device can also interface to multiple slave devices simultaneously by setting the select bits for each desired device. For example, to select slave devices 1, 5, and 6, users write a 1 to bits 1, 5, and 6 of the select register.

A reset clears every bit of the select register except bit 0. Thus, after a device reset, slave device 0 will automatically be selected.

SPI Software Routines



If there is one or more SPI peripheral present in the Nios system, the SPI peripheral software routines are available in the Nios library (.lib folder in the custom software development kit).

For more information regarding software routine calls and custom software development kits, please refer to the *Nios Software Development Reference Manual*.



101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
<http://www.altera.com>
Applications Hotline:
(800) 800-EPLD
Customer Marketing:
(408) 544-7104
Literature Services:
lit_req@altera.com

Altera, APEX, ACEX, FLEX, MegaWizard, and Nios are trademarks and/or service marks of Altera Corporation in the United States and other countries. Altera acknowledges the trademarks of other organizations for their respective products or services mentioned in this document. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



Copyright © 2001 Altera Corporation. All rights reserved.

I.S. EN ISO 9001