SPI Slave Peripheral Using the Embedded Function Block

January 2015 **Reference Design RD1125**

Introduction

Microprocessors often have a limited number of general purpose I/O (GPIO) ports to reduce pin count and shrink package size. To overcome this limitation, port expanders are often employed to provide I/O expansion capabilities. Most generic GPIO expanders use low pin count serial protocols, such as I^2C or SPI, as the interface to the master. They allow designers to save the GPIO ports on the microprocessor for other critical tasks.

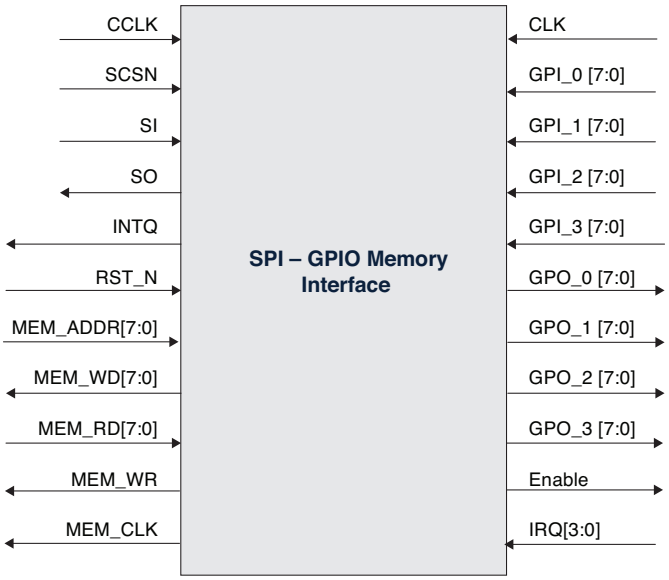
This design provides a programmable solution for serial expansion of GPIOs. It uses a Serial Peripheral Interface (SPI) bus between the microprocessor and the GPIOs. The design provides additional control and monitoring capabilities for the microprocessor when it does not have sufficient GPIOs to do the job.

Apart from the GPIO expander, this design also provides a memory interface to the microprocessor. This memory is accessible via the SPI Interface. The SPI memory command interface is similar to those commonly found in discrete SPI memory devices.

This reference design is intended to provide a familiar and intuitive interface extension to the MachXO2™ and MachXO3L Embedded Function Block (EFB). The EFB SPI module supports the major features of SPI bus. Users can take advantage of the MachXO2 and MachXO3L hardened SPI port to provide a port expansion or a memory extension. The user is spared from learning operational details of the SPI protocol, the WISHBONE bus or the EFB block.

Interface

Figure 1. SPI to GPIO Memory Interface



Functional Description

This design provides a default eight bytes of I/O port and memory interface, which is controlled through the SPI-slave interface. There are four single-byte input ports and four single-byte output ports. The interrupt generation block generates an interrupt signal to the master when at least one of the interrupts pins has transitioned high. This design interfaces with a general purpose memory block, typically Embedded Block RAM (EBR) in the device fabric. The memory is accessed via SPI Master commands.

The internal state machine is designed according to TN1246, Using User Flash Memory and Hardened Control Functions in MachXO2 Devices. and TN1294, Using Hardened Control Functions in MachXO3 Devices Reference Guide. It responds to the commands issued by an external SPI master. The design is robust and will appropriately tolerate erroneous commands.

Figure 2. Functional Block Diagram

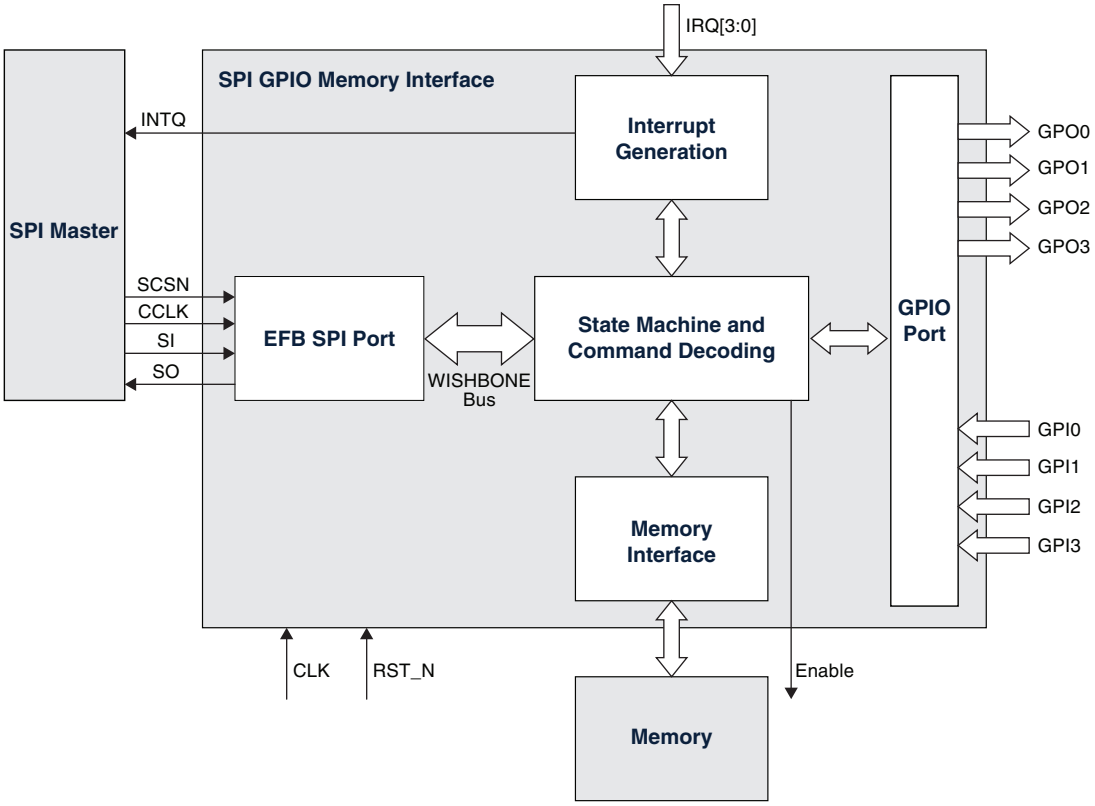


Table 1. SPI GPIO Expander I/O Interface Descriptions

Signal	I/O	Description
SPI Interface		
CCLK	Input	SPI serial clock
SI	Input	SPI serial input data
SO	Output	SPI serial output data
SCSN	Input	SPI chip-select (asserted low)
INTQ	Output	External Interrupt, asserted low. Asserted low when any IRQ status register bit is set (open drain output).
General Signal		
CLK	Input	Master clock

Table 1. SPI GPIO Expander I/O Interface Descriptions (Continued)

Signal	I/O	Description
SPI Interface		
RST_N	Input	Active low reset
Enable	Output	General purpose enable signal
GPIO Interface		
GPO_0[7:0]	Output	General purpose output byte port
GPO_1[7:0]	Output	General purpose output byte port
GPO_2[7:0]	Output	General purpose output byte port
GPO_3[7:0]	Output	General purpose output byte port
GPI_0[7:0]	Input	General purpose input byte port
GPI_1[7:0]	Input	General purpose input byte port
GPI_2[7:0]	Input	General purpose input byte port
GPI_0[7:0]	Input	General purpose input byte port
IRQ[3:0]	Input	Input interrupt
Memory Interface		
MEM_CLK	Output	Clock port for memory
MEM_WR	Output	Memory write signal
MEM_ADDR[7:0]	Output	Memory address
MEM_WD[7:0]	Output	Memory write data
MEM_RD[7:0]	Input	Memory read data

SPI Interface

This design is controlled by a set of instructions that are sent from an external SPI Master. The SPI Master communicates with the design via the SPI bus comprised of four signals: Chip Select (SCSN), Serial Clock (CCLK), Serial Input (SI), and Serial Output (SO). The SCSN must first be asserted low prior to a valid instruction or operation. After the SCSN pin is asserted, the SPI Master clocks out a valid 8-bit Command on the SPI bus (see Table 2). Subsequent operands and data are exchanged per the command as described in Table 3. All commands, address, and data bytes are transferred with the most significant bit (MSB) first. The SPI Master Data is captured on a leading (first) clock edge, and propagated on the opposite clock edge. Operations are terminated by de-asserting the SCSN pin.

HDL Parameter Descriptions

This design uses a number of parameters to control various aspects of the design. This allows the user to specify the interfaces to meet custom requirements without modifying the underlying Verilog RTL code. Table 2 provides descriptions of the parameters used in the SPI GPIO Memory Expander.

Table 2. Parameter Descriptions

Operation	Description	Value
GPI_PORT_NUM	Specifies the number of general purpose input ports	1|7
GPI_DATA_WIDTH	Specifies the width of general purpose input ports	1|8
GPO_PORT_NUM	Specifies the number of general purpose output ports	1|7
GPO_DATA_WIDTH	Specifies the width of general purpose output ports	1|8
MEM_ADDR_WIDTH	Specifies the width of memory address	1|8
IRQ_NUM	Specifies the number of input interrupt pin	1|8
REVISION_ID	Specifies the revision id	0x00|FF
MAX_MEM_BURST_NUM	Specifies the burst width for memory operation	1|255
INTQ_OPENDRAIN	Specify whether INTQ output will be open drain or not	ON/OFF

Commands

Table 3 lists the SPI bus commands support by the SPI GPIO Memory Interface reference design.

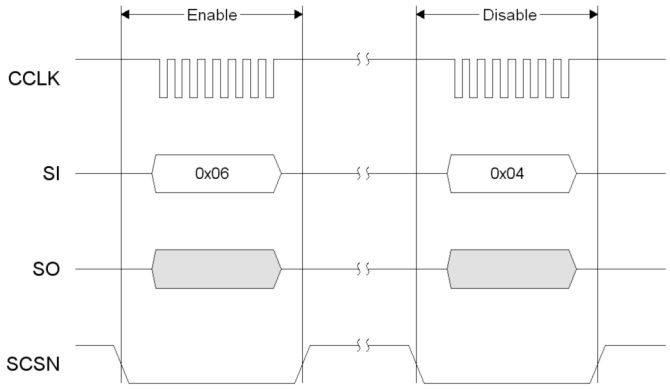
Table 3. SPI Master Command Structure

Operation	Command	Address Byte	Dummy Byte	Data Bytes
Enable	0x06	—	—	—
Disable	0x04	—	—	—
Write GPO	0x01	1	—	1
Latch GPI	0x03	—	—	1
Read GPI	0x05	1	1	1
Write Memory	0x02	1	—	1+
Read Memory	0x0B	1	1	1+
IRQ Enable Write	0x66	—	—	1
IRQ Enable Read	0x6A	—	1	1
IRQ Status	0x65	—	1	1
IRQ Clear	0x61	—	—	1
Revision ID	0x9F	—	1	1

Enable/Disable Command

The enable command will set the enable output port of the design. This enable port may be used for any general-purpose control function. No circuitry internal to the reference design is dependent upon the state of enable. Figure 3 shows the enable command operation. The disable command has the same structure except for the command value.

Figure 3. Enable Command Operation



Interrupt Commands

This design support input interrupts (default 4). Based upon the IRQ status and enable bits they may generate an output interrupt INTQ. These inputs can be enabled/disabled by the SPI master. The INTQ output signal is asserted (active low) whenever an IRQ input transitions high while its corresponding interrupt enable bit is set. The SPI master can read the status of the interrupts and enable settings by issuing a read command.

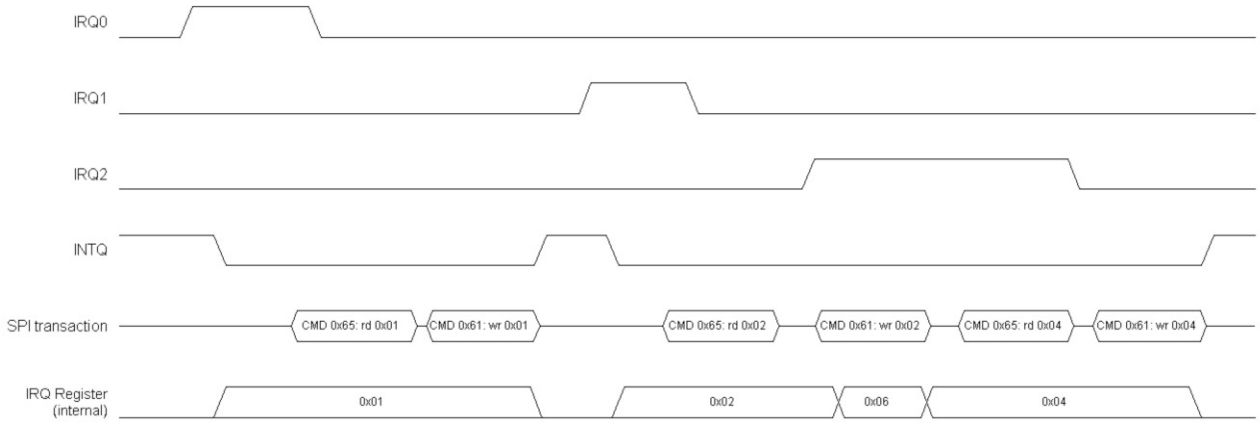
The design maintains two registers to hold the values of interrupt enable and interrupt status. The interrupt enable register holds the current enable state. The SPI master can read and write this register directly. The interrupt status register latches the status of each interrupt. Interrupt status bits indicate that the corresponding IRQ bit has transitioned ‘0’ to ‘1’. The status register bits are cleared by writing the IRQ clear command and setting the corresponding bit to ‘1’. Table 4 shows the structure of the interrupt enable and status register.

Table 4. Interrupt Enable and Status Register (Default Assignments)

Register	Width	Access	Reset Value	Bit Definition
Interrupt Enable	8	R/W	0x00	[7-4] – Reserved for future use 3 – Enable IRQ[3] signal 2 – Enable IRQ[2] signal 1 – Enable IRQ[1] signal 0 – Enable IRQ[0] signal
Interrupt Status	8	R/W	0x00	[7-4] – Reserved for future use 3 – Set when IRQ[3] goes from 0 to 1 2 – Set when IRQ[2] goes from 0 to 1 1 – Set when IRQ[1] goes from 0 to 1 0 – Set when IRQ[0] goes from 0 to 1 [3-0] – Reset when the SPI Master sends IRQ Clear. The corresponding bit will be cleared in the status register.

Figure 4 shows the interrupt enable, set and clear command.

Figure 4. Interrupt Enable, Set and Clear Operation



INTQ is an open drain output by default. Users can change from open drain to drive 0/1 by setting the parameter INTQ_OPENDRAIN at the top level.

The interrupts commands are shown in the figures below.

Figure 5. Interrupt: Enable Read and Write

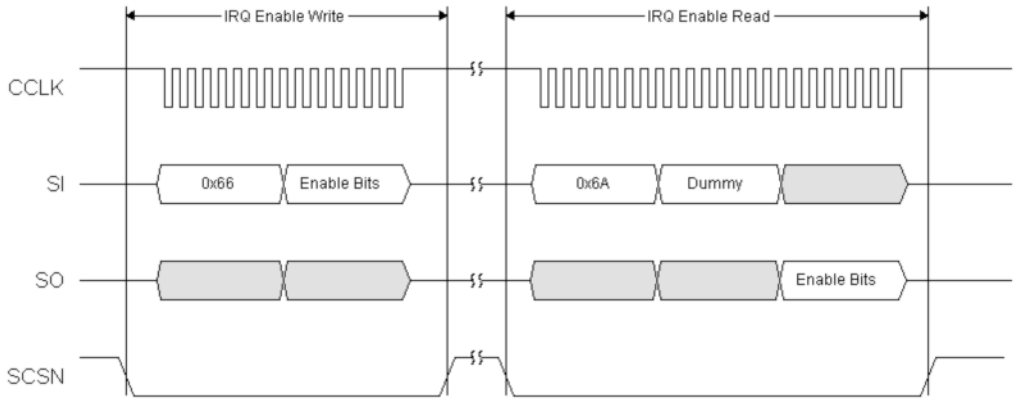
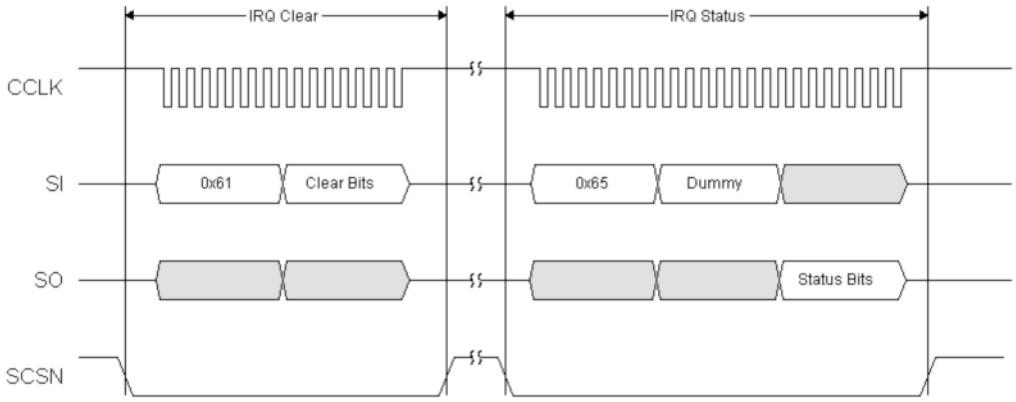


Figure 6. Interrupt: Clear and Status



GPIO Commands

General purpose input and output ports can be controlled by the SPI Master. Read GPI and write GPO commands are provided to control the input and output ports respectively. The width and number of ports are parameterized so that users can change them without making any change in code (the vector size definitions of the GPI and GPO ports may require changes accordingly).

An additional command, latch GPI, provides a common sampling instant among all input ports. All GPI ports are latched at the same time using this command. The LSB controls the latch. By default (0) the latch is in transparent mode. A ‘0’ to ‘1’ transition latches all GPI ports simultaneously, and the read data held until the LSB is reset to ‘0’. GPIO commands are shown in Figures 7 and 8.

Figure 7. Write & Latch GPIO Transaction

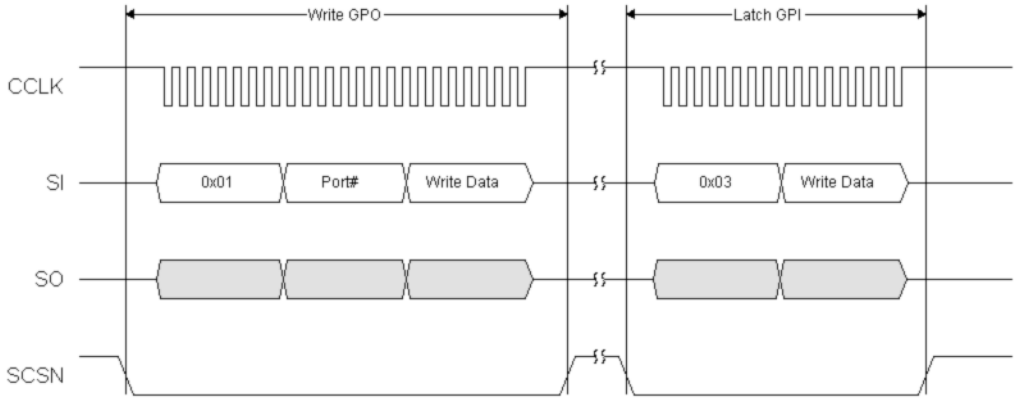
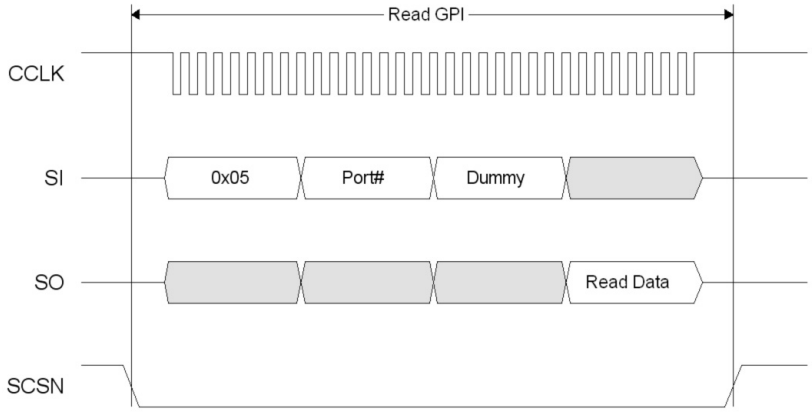


Figure 8. Read GPI Transaction



Memory Commands

This reference design provides a port for interfacing to RAM. All memory operations are controlled by the SPI Master. By default, burst read and write operations are supported by any number of data, up to 8 bytes. Therefore, the size of the SPI memory write command varies from 3 to 10 bytes and the read command varies from 4 to 11 bytes. The burst length may be altered via the parameter MAX_MEM_BURST_NUM. Memory commands are shown in Figures 9 and 10.

Figure 9. Write Memory

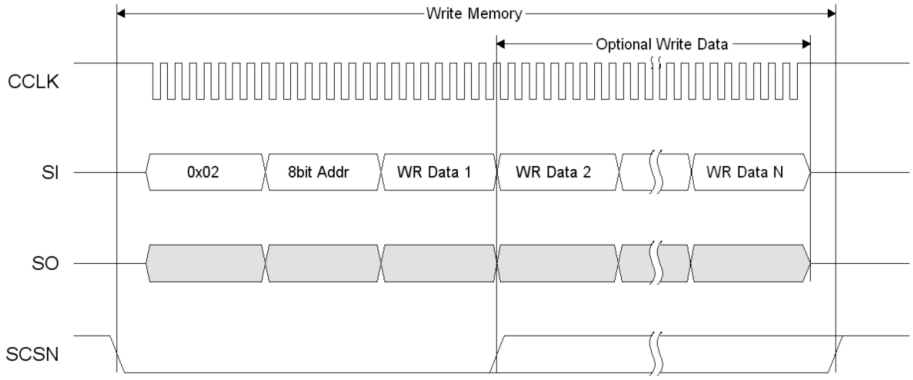
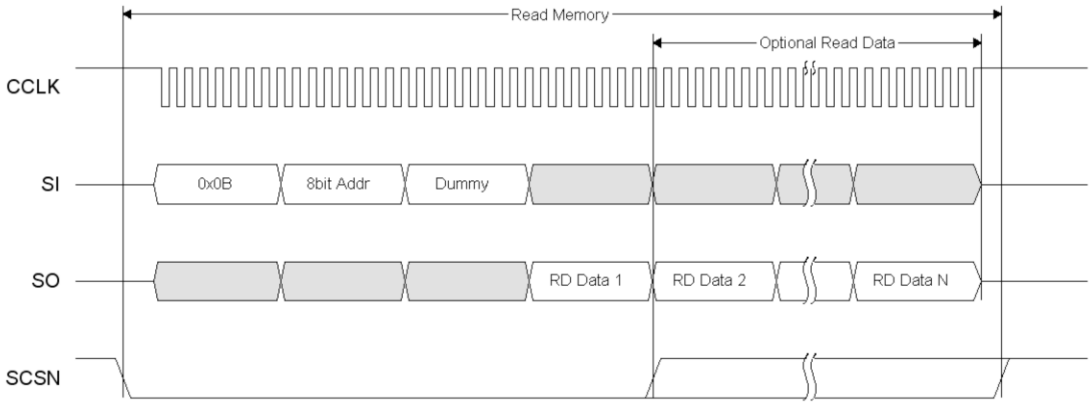


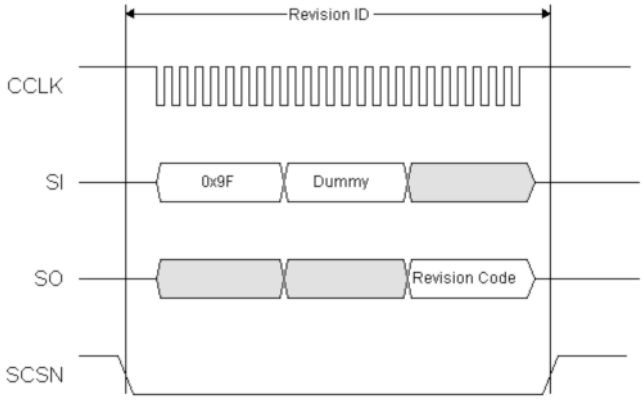
Figure 10. Read Memory



Revision ID Commands

This reference design provides a fixed Revision ID read command. It can be used, for example, to maintain a manufacturer and device ID read methodology, or a design revision or time stamp.

Figure 11. Revision ID



Erroneous Commands

This design is capable of detecting error transfers like unknown commands, too many bytes, too few bytes etc. and the design can respond to different error scenarios. If a command is sent to a design without a chip select enable (SCSN), the design will ignore them. When there are fewer bytes or incorrect commands sent by the master, the slave design will ignore that command and discard the data.

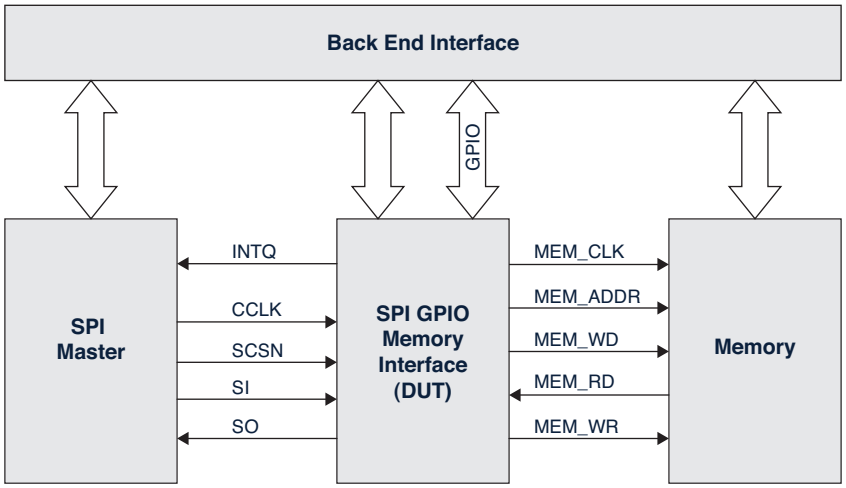
If there are more than the required number of bytes in a write command transfer, the design will consider the relevant bytes and ignore the extra bytes sent by the master. For an extra read request, the design will respond with 0xFF until SCSN is deasserted.

Test Bench Description

The SPI GPIO Memory Interface design is simulated with the aide of SPI Master Bus Functional modules. The test bench for this design consists of the following functional blocks, as shown in Figure 12.

- SPI master module
- Memory module
- Design under test (SPI slave)
- Back End Interface (Clock and Reset Generation)

Figure 12. Test Bench Architecture



The SPI interface variables CLOCK_POLARITY, Transmit Edge, LSB first and CLOCK_PHASE are all set to their default values (0) for simulation. Users can change settings by changing these values in IPexpress™ or setting a corresponding EFB register using the WISHBONE interface.

Figure 13. Enable Disable Command

Figure 14. Set Interrupt Command

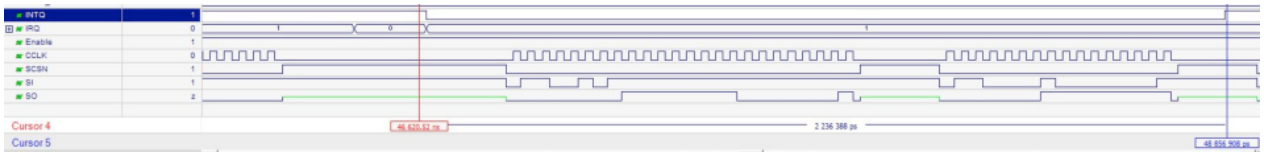


Figure 15. Writing GPO Command

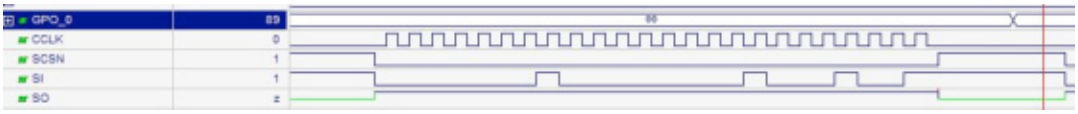


Figure 16. Writing into Memory Command

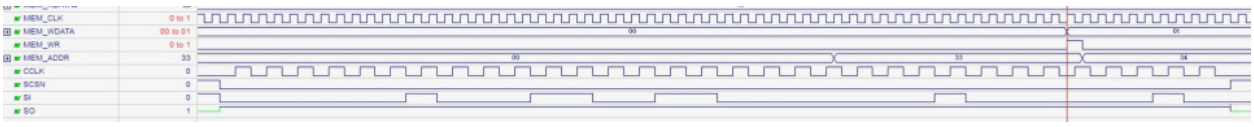
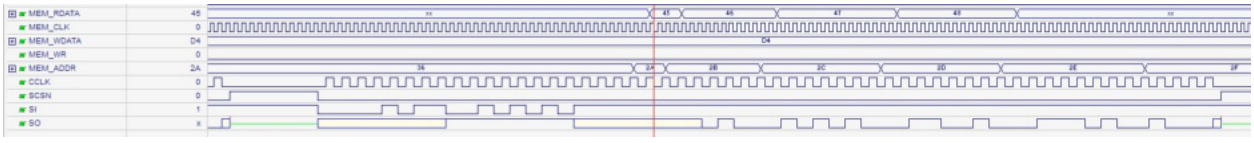


Figure 17. Reading from Memory Command



Implementation

This design is implemented in VHDL and Verilog. When using the design in a different device, density, speed, or grade, performance and utilization may vary. Default settings are used during the fitting of the design.

Table 5. Performance and Resource Utilization

Family	Language	Speed Grade	Utilization	f_{MAX} (MHz)	I/Os	Architecture Resources
MachXO2[1]	Verilog-LSE	–6	307 LUTs	>50	102	EFB
	Verilog-Syn	–6	246 LUTs	>50	102	EFB
	VHDL-LSE	–6	308 LUTs	>50	102	EFB
	VHDL-Syn	–6	242 LUTs	>50	102	EFB
MachXO3L[2]	Verilog-LSE	–6	307 LUTs	>50	102	EFB
	Verilog-Syn	–6	246 LUTs	>50	102	EFB
	VHDL-LSE	–6	308 LUTs	>50	102	EFB
	VHDL-Syn	–6	242 LUTs	>50	102	EFB

1. Performance and utilization characteristics are generated using LCMXO2-2000HC-6BG256C with Lattice Diamond® 3.3 design software with LSE (Lattice Synthesis Engine) and Synplify Pro®.
2. Performance and utilization characteristics are generated using LCMXO3L-4300C-6BG256C with Lattice Diamond 3.3 design software with LSE and Synplify Pro.

References

- TN1246, Using User Flash Memory and Hardened Control Functions in MachXO2 Devices
- TN1294, Using Hardened Control Functions in MachXO3 Devices Reference Guide

Technical Support Assistance

e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
January 2015	1.3	Updated Introduction section. Added MachXO3L support.
		Updated Functional Description section. Revised and added technical note references.
		Updated Implementation section. Updated Table 5, Performance and Resource Utilization. — Modified Utilization data for Verilog and VHDL implementation. — Provided LSE and Synplify Pro data for MachXO2. — Changed device, and updated Lattice Diamond software version in footnote 1. — Changed Lattice Diamond software version in footnote 2.
		Updated References section. Modified and added technical note references.
February 2014	01.2	Updated for VHDL implementation.
		Added support for MachXO3L device family.
		Updated Technical Support information.
May 2012	01.1	Corrected USPI to GPIO Memory Interface diagram.
April 2012	01.0	Initial release.