GPIO Expander

April 2011 Reference Design RD1065

Introduction

Most microprocessors have a General Purpose Input/Output (GPIO) interface to communicate with external devices and peripherals through various protocols. These GPIO connections are usually very flexible. They can be individually configured as an input to read signals from other parts of a circuit, or as an output to control or send signals to other devices. GPIOs are typically arranged into 8-port groups that are often sufficient for data and control support for external devices.

Despite the convenience and flexibility of this interface, microprocessors sometimes offer a limited number of GPIO ports for the purpose of reducing pin counts and package sizes. For many applications, designers need more GPIO ports than those available on the microprocessor. This design provides a solution that uses a Lattice CPLD as a GPIO Expander. It provides additional control (control signal and data output signal) and monitoring (input data signal) capabilities when the microprocessor has insufficient I/O ports.

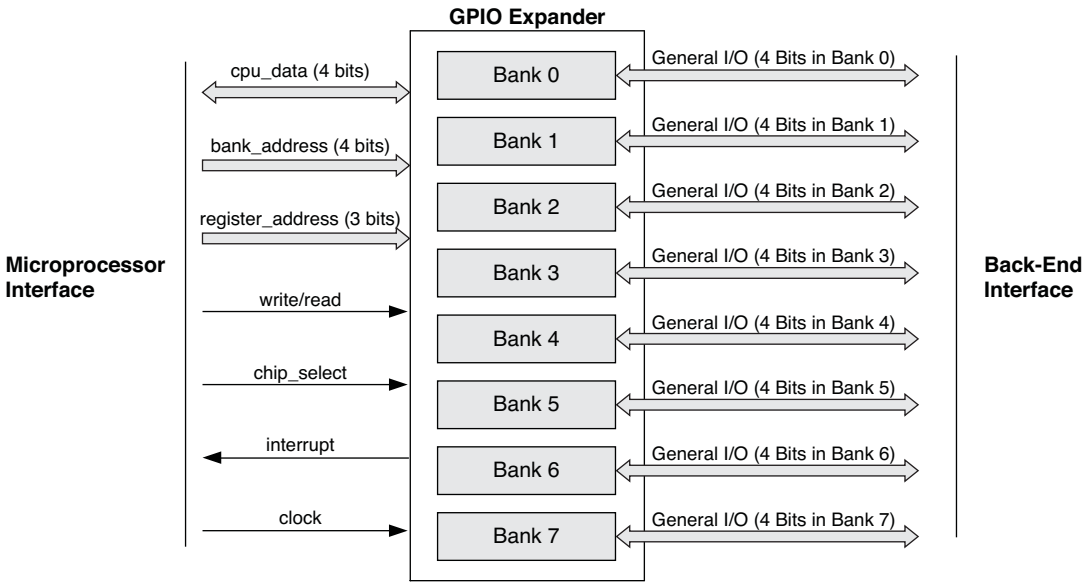
Theory of Operation

Overview

The GPIO Expander design is used to interface a microprocessor with a back-end device through a common timing specification. This design is configured as eight banks that are independent of each other. Each bank consists of a few GPIO ports. The number of GPIO ports in each bank can be configured by the designer, with a default value of four. Each GPIO port in a bank can be individually configured as an input, output or bi-directional port by the microprocessor. The module provides an active low interrupt pin. When related internal registers are configured correctly, the interrupt may be activated when any of the GPIO inputs reach the programmed interrupt level.

Figure 1 is the top-level block diagram of this design. It provides a generic interface for the microprocessor and eight GPIO banks. The microprocessor's data width and the number of GPIO ports are set by the default value. Table 1 lists the I/O pins used in this design.

Figure 1. GPIO Expander I/O Interface Block Diagram



Note: The cpu_data width and general I/O width can be configured by the designer. The default value is 4.

© 2011 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal. All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

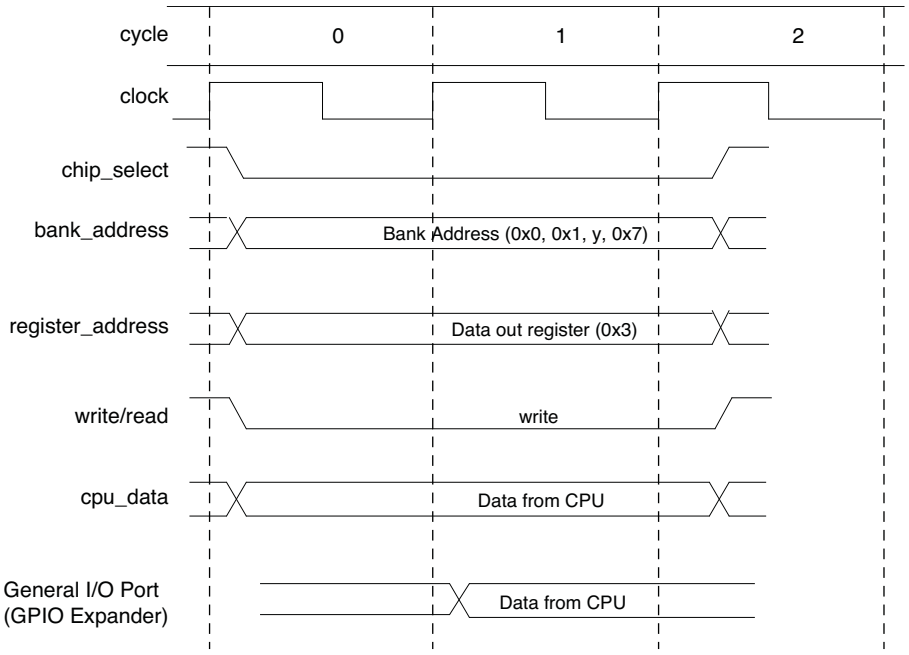
Table 1. Expander I/O Interface Descriptions

Signal Name	Signal Direction	Active State	Definition
Microprocessor Interface			
cpu_data (4 bits)	Bi-directional	N/A	Data bus with CPU.
bank_address (4 bits)	Input	0x0,0x1….0xf	Address used to select one of the 16 banks.
register_address (3 bits)	Input	0x0,0x1,0x2,0x3,0x4	Address used to select one of the four registers in each bank or input data from back-end device.
write/read	Input	Low = write High = read	Write or read control signal from the CPU.
chip_select	Input	Low	Chip select control signal from the CPU.
interrupt	Output	Low	Interrupt signal generated by the back-end device.
clock	Input	N/A	Clock signal from CPU to synchronize operation.
Back-end Interface			
General I/O (4 bits)	Input,Output or Bi-directional	N/A	Eight independent GPIO banks.

Common Microprocessor Interface

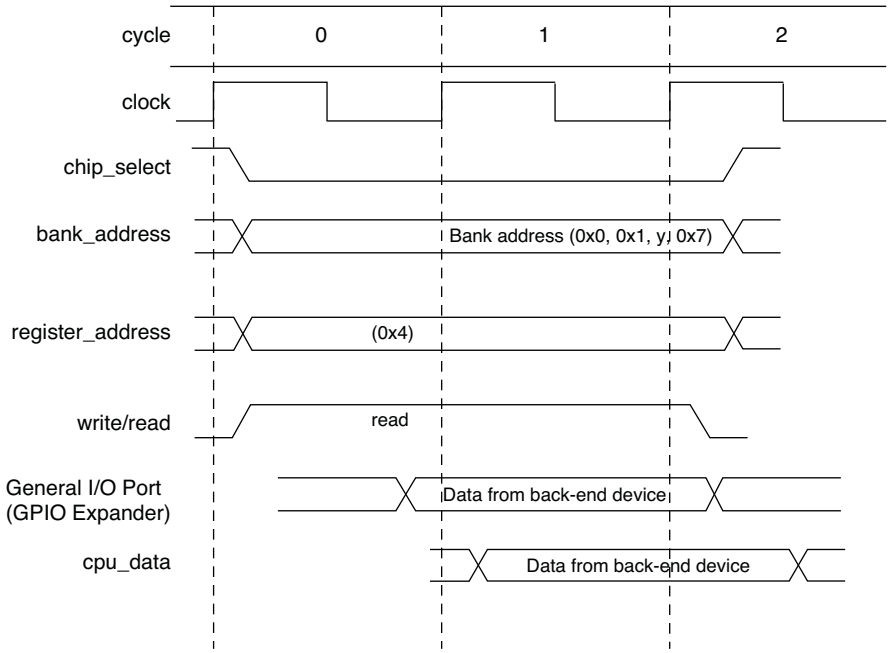
Various microprocessors require different timing specifications to read from or write to the external device. The microprocessor timing defined in this design is based on the ARM7 timing specification (refer to ARM7TDMI-S Revision r4p3). Figures 2 and 3 show the functional representation of the write and read operations respectively.

Figure 2. Microprocessor Write Data to GPIO



Initially, the microprocessor pulls down the signals chip_select and write/read to implement a write cycle. Meanwhile, the signal's bank_address and register_address must be set to the appropriate values in order to write a specific bank and a register. If the microprocessor writes data to the back-end device, the value of register_address must be set to 0x3. The data from the microprocessor must be available at the same time. It takes two clock cycles for the microprocessor to complete the write operation. During this time, the control signals such as chip_select, bank_address, register_address, write/read, and cpu_data must not be updated.

Figure 3. Microprocessor Read Data from GPIO

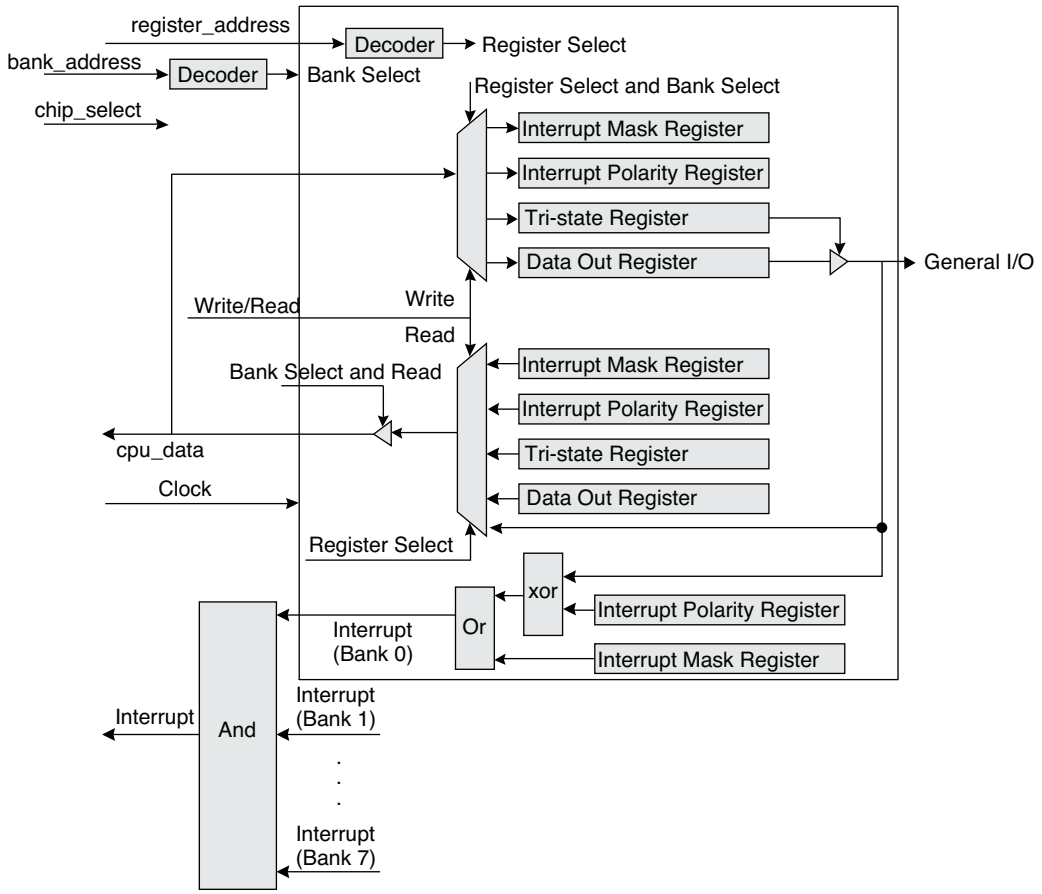


To initiate a read cycle, the microprocessor pulls down the signal chip_select and pulls up the signal write/read. Meanwhile, the signals bank_address and register_address must be set to the appropriate values to determine which bank and which register to read from. If the microprocessor reads data from the back-end device, the value of register_address must be set to 0x4. Like the write operation, the microprocessor also takes two clock cycles to complete the read operations. Therefore the control signals from the microprocessor interface should not be updated during this period. The data from the back-end device is available at the second cycle.

GPIO Expander Operation

The eight GPIO banks have the same structure and are independent of each other. For every microprocessor operation, the signal bank_address will determine the bank with which the microprocessor will communicate. The designer can easily add more banks if additional GPIO ports are needed. Figure 4 shows the function of each bank.

Figure 4. Functional Block Diagram a GPIO Bank



There are four registers in each bank. These include the interrupt mask register, interrupt polarity register, tri-state register and data out register. These registers are all four bits wide by default. The microprocessor can read from or write to the registers by providing the proper bank select signal and register address.

Interrupt Mask Register Definition

The interrupt mask register defines which GPIO input can generate an interrupt to the microprocessor. When the bit in this register is set to '0', the corresponding GPIO input can generate interrupt.

Register Name	Register Address	Width (Default)	Reset Value	Access
Interrupt mask register	0x0	4	0xf	Read/Write

Interrupt Polarity Register Definition

The interrupt polarity register defines the active level of a GPIO input which can generate an interrupt. When the bit is set to '1' in this register, the corresponding GPIO input generates an interrupt when its level is high. When the bit is set to '0', the corresponding GPIO input generates an interrupt when its level is low.

Register Name	Register Address	Width (Default)	Reset Value	Access
Interrupt polarity register	0x1	4	0x0	Read/Write

Tri-state Register Definition

The tri-state register enables or disables the output and bi-directional modes of operation for each GPIO. When the bit is set to '1' in this register, the corresponding GPIO output driver is enabled. When the bit is set to '0', the corresponding GPIO is configured as input and the output/bi-directional driver operates in tri-state mode.

Register Name	Register Address	Width (Default)	Reset Value	Access
Tri-state register	0x2	4	0x0	Read/Write

Data Out Register Definition

The data out register drives GPIO outputs.

Register Name	Register Address	Width (Default)	Reset Value	Access
Tri-state register	0x3	4	0x0	Read/Write

The first four register address values (0x0, 0x1, 0x2, 0x3) select the specific register to be written to or read from. If the value of register address is 0x4, this indicates that the microprocessor reads data from the GPIO port directly. The GPIO expander ignores the other values of the signal register address.

Before every application, the microprocessor selects one or more banks to be used depending on how many GPIO ports the application uses. At the same time, the microprocessor sets the appropriate values for these four registers before the read or write operation begins on the back-end device. Figure 5 shows a timing diagram of how the microprocessor writes data to the register and Figure 6 shows timing diagram of how the microprocessor reads data from the register.

Figure 5. Microprocessor Writes Data to Registers

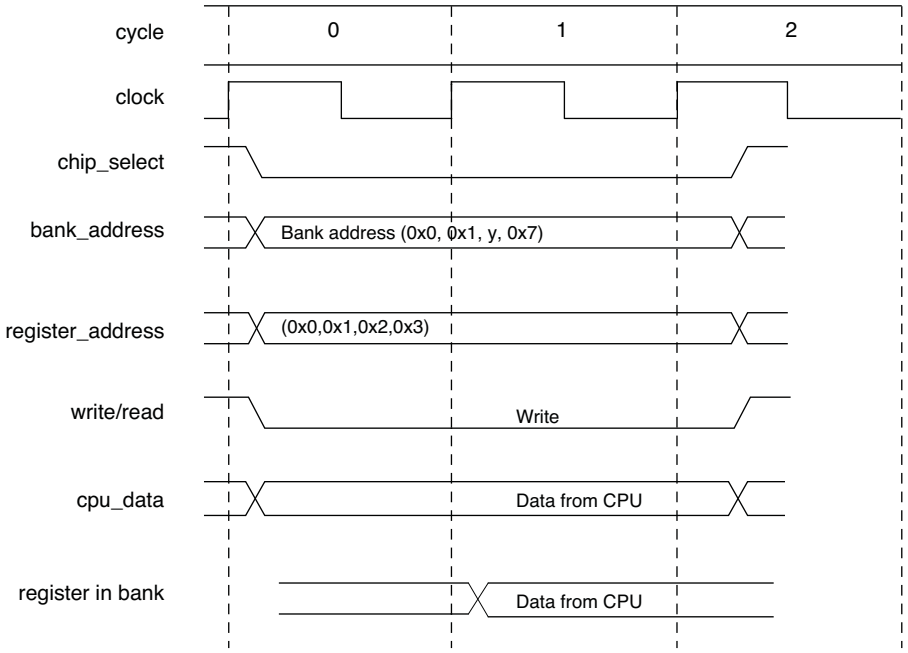
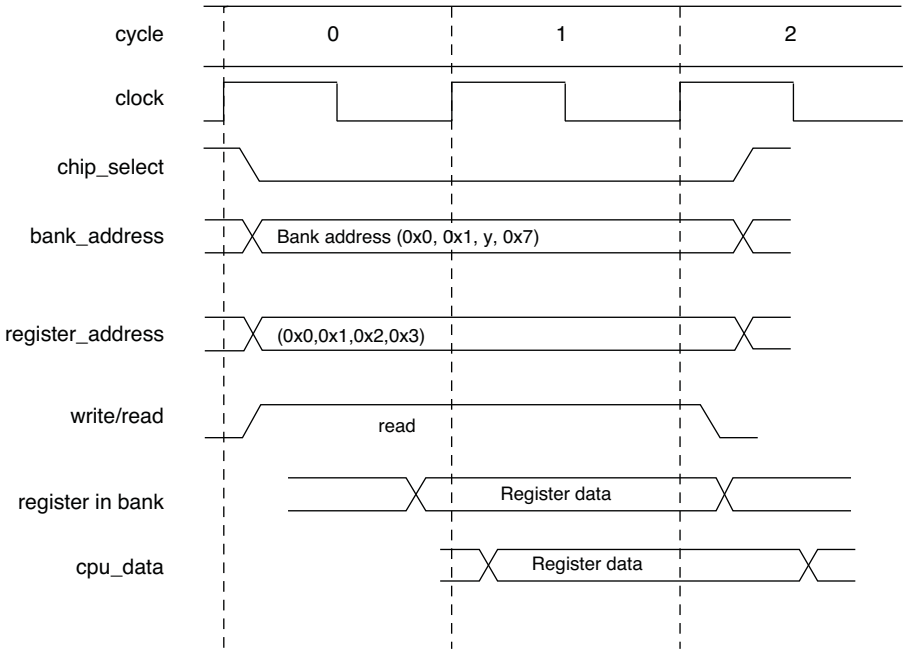


Figure 6. Microprocessor Reads Data from the Register



GPIO Configuration

Each bank is connected with four GPIO ports by default. These GPIO ports can be dynamically and individually configured as input, output or bi-directional ports by setting the tri-state register as '1' or '0'.

GPIO as an Input Signal

When the GPIO is used as an input, the corresponding bit in the tri-state register must be set to '0' and the corresponding bit in the interrupt mask register must be set to '1' by the microprocessor. The microprocessor can set the signal register_address to 0x4 to read the input GPIO value.

GPIO as an Output Signal

When the GPIO is used as an output, the corresponding bit in the tri-state register must be set to '1' and the corresponding bit in the data out register must be written to the value that is required to be driven on the GPIO. The corresponding bit in the interrupt mask register must be set to '1' to disable generation of spurious interrupt. All these operations should be done by the microprocessor.

GPIO as a Bi-directional Signal

To use the GPIO as a bi-directional signal, the corresponding bit in the tri-state register must be toggled up and down to enable or disable tri-state of the bi-directional driver. The corresponding bit in the data out register must be written to the value that is required to be driven on the output driver. The corresponding bit in the interrupt mask register must be set to '1' to disable generation of spurious interrupt. All these operations should be done by the microprocessor.

GPIO as an Interrupt Input

When the GPIO is used as an interrupt input, the corresponding bit in the tri-state register must be set to '0' and the corresponding bit in the interrupt mask register must be set to '0' as well by the microprocessor. The corresponding bit in the interrupt polarity register indicates the active level of the GPIO interrupt.

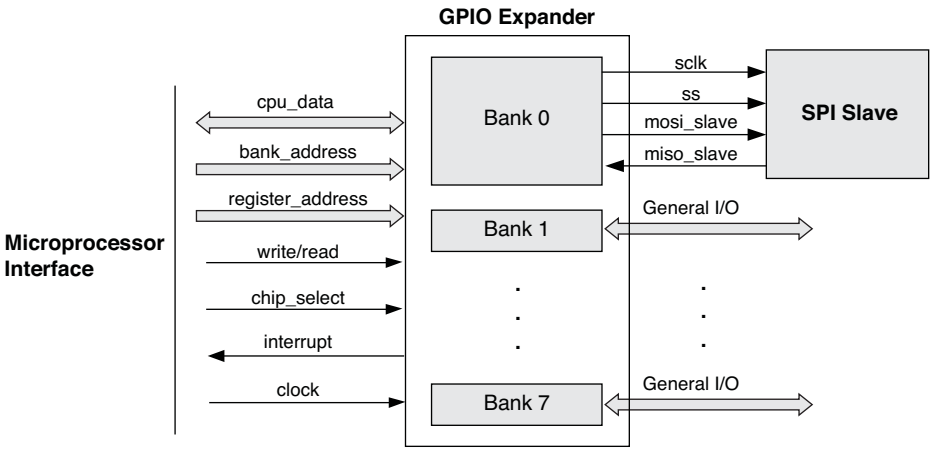
Reset Value

This design has a reset signal not shown in Figure 1. This reset signal is used to set all the registers and the GPIO to the initial values. When this signal is active, all GPIOs are configured as inputs and the output drivers are disabled. The interrupt mask registers are set to '1' to disable spurious interrupt generation.

Test Bench Description

The test bench for this design is an application example that shows how to use the GPIO Expander design. The GPIO Expander is used to interface with a SPI slave device as shown in Figure 7.

Figure 7. Interfacing the Microprocessor to the SPI Slave



This example selects bank 0 of the GPIO Expander to connect with the SPI slave device. Three general I/O ports in bank 0 are configured as outputs by the microprocessor and are used as the serial clock signal, slave select signal and master-out-slave-in signal respectively, as defined by the SPI protocol. The last port in bank 0 is configured as an input by the microprocessor and is used as the master-in-slave-out signal.

After reset, the microprocessor initializes the registers in bank 0. Figure 8 shows the timing of this operation.

Figure 8. Microprocessor Initializes the Registers in Bank 0

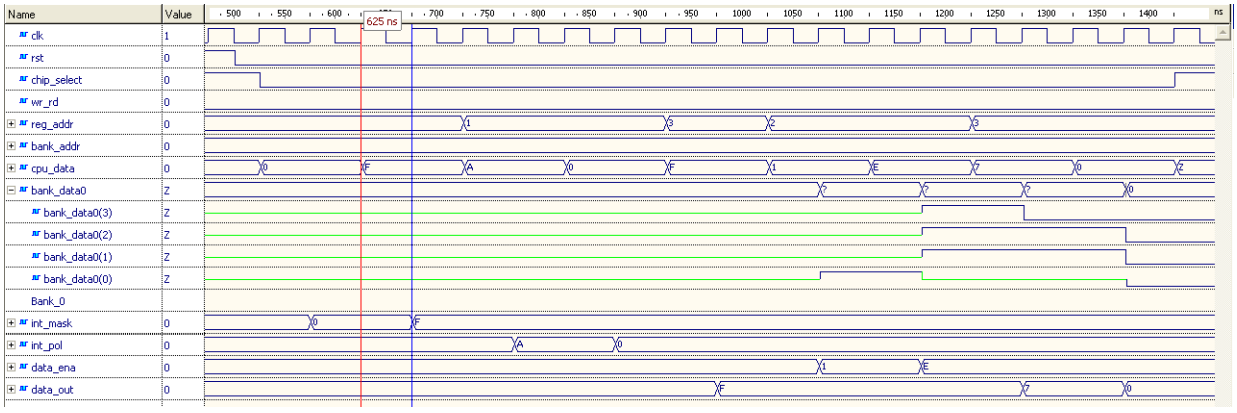
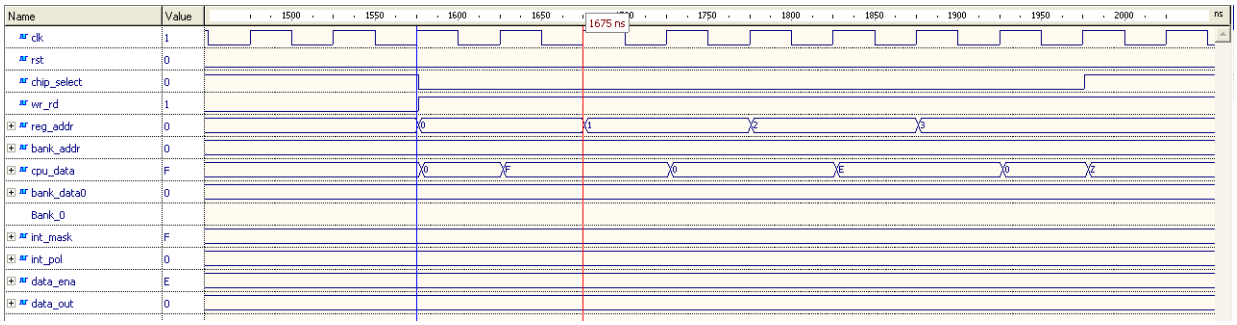


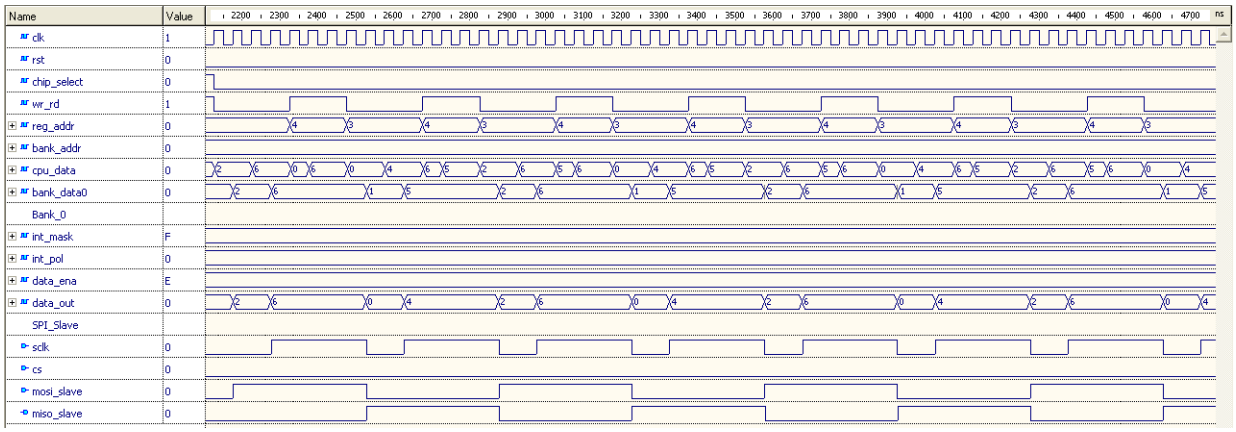
Figure 9 shows the timing of the read operation of the registers in bank 0.

Figure 9. Microprocessor Reads the Registers in Bank 0



The GPIO is able to generate the SPI timing based on the SPI protocol. Figure 10 shows the read/write timing between the GPIO Expander and the read/write device.

Figure 10. SPI Timing



Implementation

This design is implemented in Verilog and VHDL. When using this 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 2. Performance and Resource Utilization

Device Family	Language	Speed Grade	Utilization	f_{MAX} (MHz)	I/Os	Architecture Resources
LatticeECP3™ [1]	Verilog	-6	242 LUTs	>100	47	N/A
	VHDL	-6	211 LUTs	>100	47	N/A
LatticeXP2™ [2]	Verilog	-5	277 LUTs	>100	47	N/A
	VHDL	-5	274 LUTs	>100	47	N/A
MachXO™ [3]	Verilog	-3	238 LUTs	>100	47	N/A
	VHDL	-3	206 LUTs	>100	47	N/A
ispMACH® 4000ZE[4]	Verilog	-5 (ns)	187 Macrocells	>100	47	N/A
	VHDL	-5 (ns)	194 Macrocells	>100	47	N/A
Platform Manager[5]	Verilog	-3	250 LUTs	>100	47	N/A
	VHDL	-3	207 LUTs	>100	47	N/A

1. Performance and utilization characteristics are generated using LFE3-17EA-6FTN256C, with Lattice Diamond™ 1.2 design software.
2. Performance and utilization characteristics are generated using LFXP2-5E-5TN144C, with Lattice Diamond 1.2 design software.
3. Performance and utilization characteristics are generated using LCMXO640C-3T100C, with Lattice Diamond 1.2 design software.
4. Performance and utilization characteristics are generated using LC4256ZE-5TN100C, with Lattice ispLEVER® Classic 1.4 software.
5. Performance and utilization characteristics are generated using LPTM10-12107-3FTG208CES,with ispLEVER 8.1 SP1 software.

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
December 2009	01.0	Initial release.
March 2010	01.1	Added VHDL support.
		Added support for the LatticeXP2 device family.
December 2010	01.2	Added support for Platform Manager device family.
		Added support for Lattice Diamond 1.1 and ispLEVER 8.1 SP1 design software.
April 2011	01.3	Added support for the LatticeECP3 device family.
		Added support for Lattice Diamond 1.2 design software.