INICORE

Datasheet

SPIMMODULE

Revision 2.2

INICORE INC.
5600 Mowry School Road
Suite 180
Newark, CA 94560
t: 510 445 1529 *f:* 510 656 0995 *e:* info@inicore.com
www.inicore.com

Table of Contents

Table of Figures

1 Introduction

Inicore's SPI Master controller core *SPIMmodule* can be used for ASIC and FPGA implementations.

This is a fully synchronous design implemented in technology independent VHDL and Verilog RTL.

1.1 Features

Following special features are available:

- Full-duplex operation
- Programmable frame length (1- 32-bit, cont)
- Programmable transfer delay
- Programmable peripheral select to serial clock delay
- Programmable peripheral slave selects
- Continuous re-transfer mode
- Supports all SPI modes (0,0 / 1,0 / 0,1 / 1,1)
  - Configurable clock phase
  - Configurable clock polarity
- Message buffer for queuing message
  - Supports autonomous message transmission
  - Selectable message buffer size
  - Message buffer is 32-bit wide
- Independent programmable bit rate generator
- Single-master interface
- Synchronous bus interface
  - Zero wait-states
  - Supports system bus' such as AMBA APB version 2.0
- Technology independent

1.2 Implementation Features

- Register based message buffer.
- The message buffer size is selectable enabling the optimal usage of the available gates.
- The number of slave select signals is selectable prior to synthesis.
- Fully synchronous design: one clock, one global asynchronous reset.
- Technology independent implementation. This design can be used for any technology as long as timing constraints are observed.

1.2.1 Deliverables

- VHDL or Verilog RTL source code
- Simulation testbench
- Timing constraints file
- Synthesis script
- User Guide

1.3 Block Diagram

The main building block and interface signals of the SPIMmodule are shown in the block diagram below:

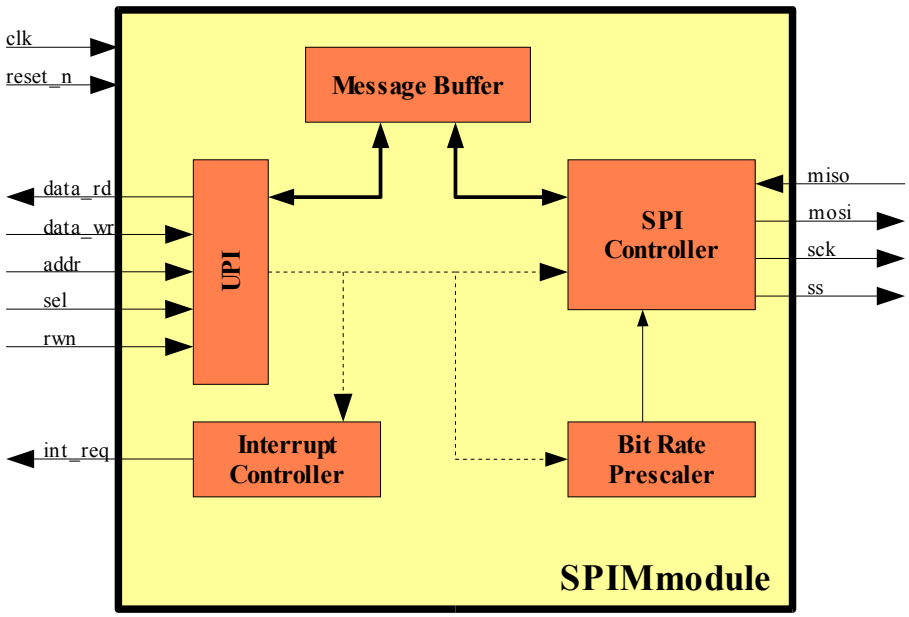

Figure 1: Block Diagram SPIMmodule

1.4 Block Description

1.4.1 System Bus Interface (UPI)

A fully synchronous single clock cycle bus interface provides direct access to all local register resources. See the memory map for detailed description of all registers.

1.4.2 Interrupt Controller

The interrupt controller bundles all local interrupt sources together and provides one interrupt request line to the main system.

1.4.3 Bit Rate Prescaler

This block generates the SPI bit rate.

1.4.4 SPI Controller

This is the low level protocol controller. It oversees the generation of the low-level SPI signal generation and observes the proper interface timing. It implements the serial-parallel and parallel-serial conversion.

1.4.5 Message Buffer

A local message buffer implements a message queue. This enables the user to setup multiple messages and have the SPI module transfer them automatically without CPU interaction. The size of the message buffer can be selected prior to synthesis of the design and enables an optimal usage of the available resources.

2 Signal Descriptions

2.1 I/O Ports

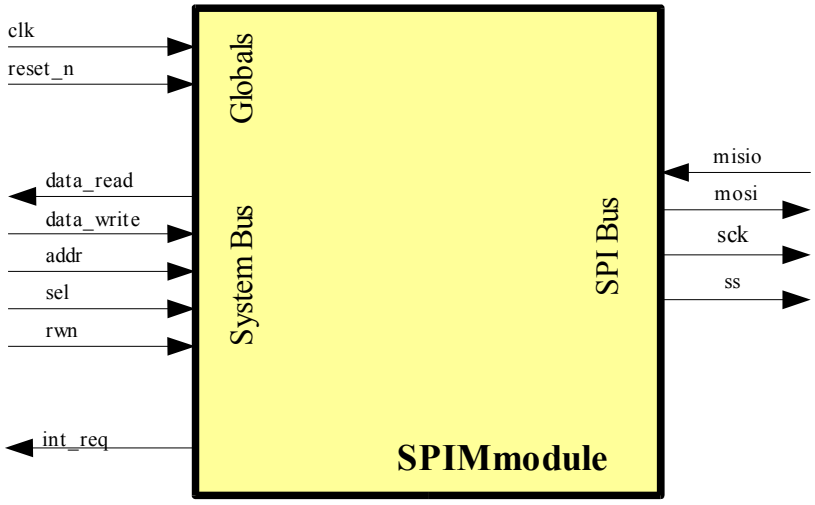

Figure 2: Symbol with I/Os

2.2 I/O Description

The following paragraphs list the inputs and outputs of the SPIMmodule core and provides an overview of their functionality.

2.2.1 Global Signals

All registers in this core are reset with the asynchronous active low reset_n. All registers are clocked with clk_sys. There are no other local clock or asynchronous reset signals.

Pin Name	*Type*	*Description*
reset_n	in	Asynchronous reset, active low
clk	in	System clock

2.2.2 System Bus Interface

The local bus interface supports 0 wait-state access. See paragraph 4.2.3 for timing information.

Pin Name	*Type*	*Description*
data_read[31:0]	out	Data read bus
data_write[31:0]	in	Data write bus
addr[4:0]	in	Address bus
sel	in	Core select signal, active high
rwn	in	Read/write control signal '0': Write '1': Read
int_req	out	Interrupt request

2.2.3 SPI Bus Interface

This is SPI side interface to the SPI slave devices.

Pin Name	*Type*	*Description*
sclk	out	SPI clock
misio	in	Master in, salve out
mosi	out	Master out, slave in
ss[7:0][1]	out	Salve select

1 The available number of chip select signals (ss) depends on the design configuration used for synthesis.

3 Programming Model

3.1 Memory Mapping

The following table shows the memory mapping of the SPIMmodule:

Address	*Name*	*R/W*	*Description*
00h	CmdBuffer	W	Command buffer
04h	Data	R/W	Receive/Transmit data buffer
08h	BufferPtr	r/W	Message buffer pointer
0Ch	ControlReg	R/W	Control register
10h	IntStatus	R/W	Interrupt status register
14h	IntEbl	r/W	Interrupt enable register
18h	ConfigReg	r/W	Configuration register
1Ch	SS Config	r/W	Slave Select Configuration

Following acronyms are used in the table above:

r: Data readback

R: Data read

W: Data write

3.2 Register Description

3.2.1 Command Buffer

This is the command buffer entry that is selected by the buffer pointer.

Register Description:

Address	*Name*	*R/W*	*Description*
+0x00h	CmdBuffer	W	Command Buffer [0]: rx capture: '0': receive data is not captured '1': receive data is captured [5:1]: Frame length Message is n+1 bits long [6]: Continous '0': Message complete '1': Message continues [7]: n/a [15:8]: Delay after transfer enable n+1 SCK cycles [23:16]: PCS to SCK delay n+1 SCK cycles [26:24]: Peripheral chip select '000': ss[0] active '001': ss[1] active ... '111': ss[7] active [29:27]: Reserved for additional chip select signals

The available number of chip select signals (ss) depends on the design configuration used for synthesis.

3.2.2 Data Buffer

This is the Tx buffer entry that is selected by the buffer pointer.

Address	Name	R/W	Description
+0x04h	Data	W	Transmit Data Buffer
		R	Receive Data Buffer

3.2.3 Buffer Pointer

The message buffer can be accessed using the buffer pointer. This pointer is used to access the command, receive and transmit buffer.

Address	Name	R/W	Description
+0x08h	BufferPtr	W	Set buffer pointer [6:0]: Buffer Pointer [7]: Write enable for command buffer pointer '0': write protected '1': write enabled [8]: Increment buffer pointer '0': no action '1': increment buffer pointer by one [31:9]: n/a
		R	Read current buffer pointer [6:0]: Buffer Pointer [22:16]: Mesage Pointer Points to the command that is sent by the SPI controller right now. [others]: '0'

The buffer pointer can be incremented by setting bit [8] to '1'.

Reading back this register always provides the actual buffer pointer position.

Note: Depending on the synthesis configuration, the buffer pointer might have a smaller size, e.g., only 3-bits. Nevertheless, the position of the pointer inside the 32-bit data remains the same. Unused bits are read '0'.

3.2.4 Control Register

The control register is used to control the operation of the core.

Address	Name	R/W	Description
+0x0Ch	ControlReg	W	Control Register [6:0]: Queue Start Pointer (QSP) This is the start address for the message pointer. [7]: Write enable for QSP '0': write protected '1': write enabled [14:8]: Queue End Pointer (QEP) This is the end address for the message pointer. [15]: Write enable for QEP '0': write protected '1': write enabled [16]: TxStart Command '0': no action '1': SPI controller starts sending SPI frames from the queue. [17]: TxStop Request '0': no action '1': SPI controller stops at end of current queue.
		R	Control Register [6:0]: Queue Start Pointer (QSP) [14:8]: Queue End Pointer (QEP) [16]: TxStart '0': SPI controller idle '1': SPI controller running [17]: TxStop Request Pending '0': No stop request pending '1': Stop request is pending [others]: '0'

The user can define a queue inside of the message buffer. Upon issuing the start command (TxStart), the SPI controller starts sending out one message after another. The start address is always the beginning of the queue defined using the Queue Start Pointer QSP. If the wrap mode is enabled (see paragraph 3.2.7 for more information) then the controller restarts at QSP once it arrives at the Queue End Pointer QEP, otherwise it stops.

To avoid disrupting a current message, the controller can only be stopped at the end of the queue. A stop request is issued by setting TxStop='1'. This request is pending as long as TxStop='1'. A stop request is only removed when the controller reaches the end of the queue.

Note: While in operation (TxStart='1') the configuration as well as the QSP/SEP settings may not be changed.

3.2.5 Interrupt Register

Local interrupt status and enable registers are used to bundle several interrupt sources of the core together and control when the external interrupt request is set.

The interrupt request signal (int_req) is only set if its respective interrupt enable flag and the interrupt status register are set.

Acknowledging an interrupt is done by writing a '1' do its respective interrupt status flag.

Address	*Name*	*R/W*	*Description*
+0x10h	IntStatus		Interrupt Status Register [0]: Tx Message Sent This flag is set at the completion of a command. If a message spawns several command entries (indicated by the CONT flag), the interrupt is set at the end of every command! [1]: End Of Queue This interrupt indicates that the end of the programmed queue is reached. This is independent of the WRAP mode setting. [2]: Controller Stopped This flag is set when the controller goes into stop mode after receiving the TxStop command.
+0x14h	IntEbl		Interrupt Enable Register [0]: Tx Message Sent [1]: End Of Queue [2]: Controller Stopped

3.2.6 Configuration Register

Using this register, the configuration of the core can be selected. The content of this register may only be changed while the SPI controller is not busy (TxStart=0).

Address	*Name*	*R/W*	*Description*
+0x18h	ConfigReg	R/W	Configuration Register [0]: Clock polarity (CPOL) '0': The inactive state of SCK is low '1': The inactive state of SCK is high [1]: Clock phase (CPHA) '0': Data is sampled on the leading edge of SCK and set on the next following edge of SCK. '1': Data is sampled on the leading edge of SCK and set on the next following edge of SCK. [2]: Wrap '0': Transmission stops at the end of the programmed queue. '1': Transmission restarts once it reached the end of the queue. [15:8]: Bit rate f_sck = f_clk / (2 * (1 + bit_rate))
	SSconfig	r/W	Slave Select Configuration [7:0][2] Slave Select Polarity '0': Active Low '1': Active High

2 The actual width of this register depends on the selected number of slave select signals.

4 Operation

This paragraph provides some additional details about the internal functionality.

4.1 Message Buffer Architecture

A local message buffer is used to queue multiple messages. The queue is defined with the two message pointers QSP (queue start pointer) and QEP (queue end pointer). To access the message buffer, the CPU uses the BufferPointer.

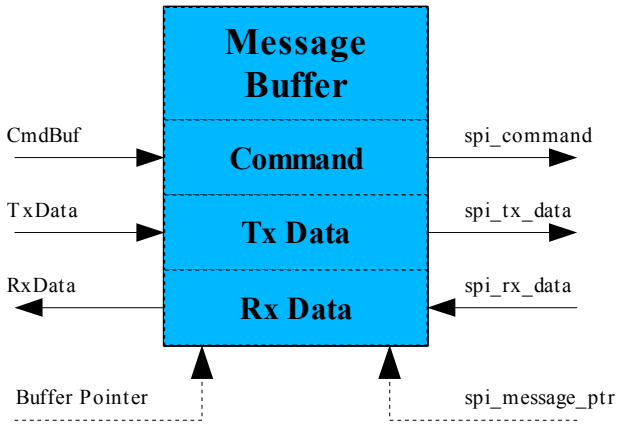

Figure 3: Block Diagram Message Buffer

One of the features is that the size of the buffer is selectable before synthesizing the design. This enables the optimal use of the available registers in a FPGA. At the same time, this versatile core can be used in different projects with an optimal sized message buffer.

5 Timing Diagram

Timing numbers depend on the chosen device, synthesis options, place&route constraints. Typical numbers will be provided.

5.1 SPI Phase = 0

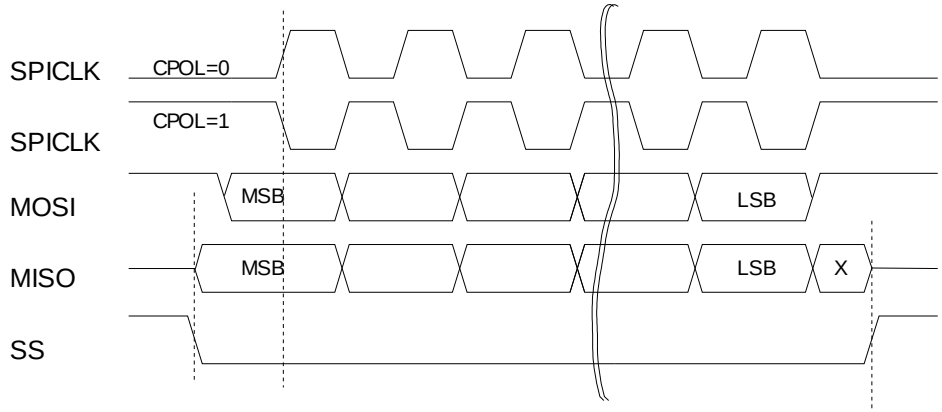

Figure 4: SPI Timing with SPI Phase=0

5.2 SPI Phase = 1

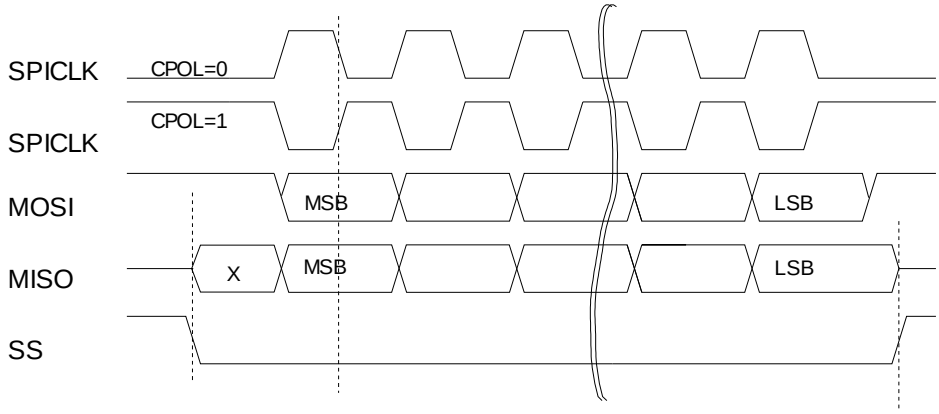

Figure 5: SPI Timing with SPI Phase=1

5.3 Local Bus Interface

The microprocessor interface is designed to operate on a full synchronous bus with zero wait-states. The internal registers are selected using sel, rwn and addr. The selected register is written on the rising edge of the system clock. data_read is asynchronously generated.

Following figures shows the interface timing diagram.

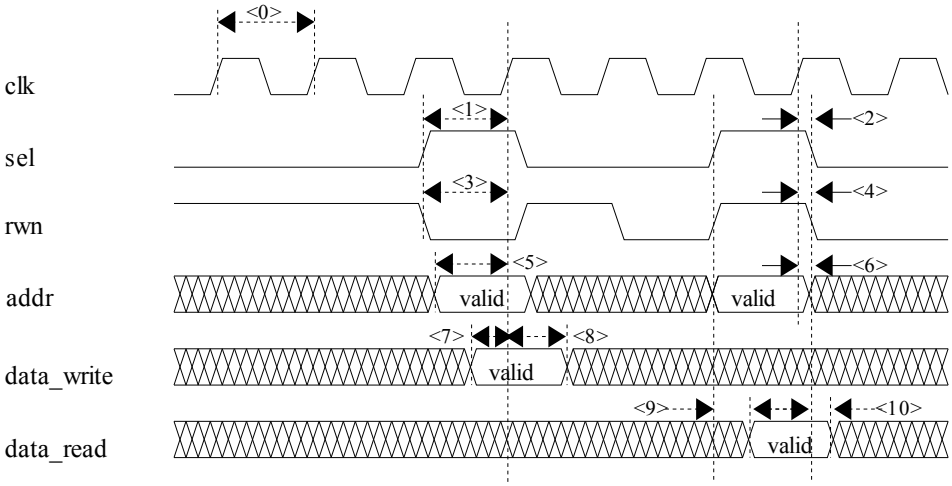

Figure 6: Local Bus Interface Timing

The timing numbers are technology dependent and can only be given once a target technology is selected.

Time nr	***t_{min} [ns]***	***t_{typ} [ns]***	***t_{max} [ns]***	***Comment***
<0>	tbd			Clock period
<1>		tbd		sel setup time
<2>		0		sel hold time
<3>		tbd		rwn setup time
<4>		0		rwn hold time
<5>		tbd		addr setup time
<6>		0		addr hold time
<7>		tbd		data_write setup time
<8>		0		data_write hold time
<9>		tbd		data_read valid after sel, rwn, addr valid
<10>		0		data_read hold time after sel, rwn, addr invalid

About Inicore

- FPGA and ASIC Design
- Easy-to-use IP Cores
- System-on-Chip Solutions
- Consulting Services
- ASIC to FPGA Migration
- Obsolete ASIC Replacements

Inicore is an experienced system design house providing FPGA / ASIC and SoC design services. The company's expertise in architecture, intellectual property, methodology and tool handling provides a complete design environment that helps customers shorten their design cycle and speed time to market. Our offering covers feasibility study, concept analysis, architecture definition, code generation and implementation. When ready, we deliver you a FPGA or take your design to an ASIC provider, whatever is more suitable for your unique solution.

Customer Advantages

We offer one-stop shopping for everything from the specifications to the chip or module solution. Our experience and fast turnaround time reduces your development costs and increases your returns from the market. Your system is not limited by the level of expertise and standard chip solutions you happen to have in-house. Achieve market success by differentiating and optimizing your product. Reusability builds the basis for further developments in the ever-decreasing product life cycle.

Visit us @ www.inicore.com