Design and Test of General-Purpose SPI Master/Slave IPs on OPB Bus

A.K. Oudjida, M.L. Berrandjia, A. Liacha, R. Tiar, K. Tahraoui & Y.N. Alhoumays
Microelectronics and Nanotechnology Division
Centre de Développement des Technologies Avancées, CDTA
Algiers, Algeria
Email: a_oudjida@cdta.dz

***Abstract*— SPI is one of the most commonly used serial protocols for both inter-chip and intra-chip low/medium speed data-stream transfers.**

In conformity with design-reuse methodology, this paper introduces high-quality SPI Master/Slave IPs that incorporate all necessary features required by modern ASIC/SoC applications. Based upon Motorola's SPI-bus specifications, version V03.06, release February 2003, the designs are general purpose solutions offering viable ways to controlling SPI-bus, and highly flexible to suit any particular needs.

The purpose of this paper is to provide a full description of an up-to-date SPI Master/Slave FPGA implementations. All related issues, starting from the elaboration of initial specifications, till the final system verification, are comprehensively discussed and justified.

The whole design code, either for synthesis or verification, is implemented in Verilog 2001 (IEEE 1365). The RTL code is technology independent, achieving a transfer rate of 71 and 75 MBPS for the Master and the Slave, respectively, when mapped onto Xilinx's Virtex 5 FPGA devices.

***Keywords*—**Serial Peripheral Interface (SPI), Intellectual Property (IP), System-on-Chip (SoC), On-Chip Peripheral Bus (OPB).

I. INTRODUCTION

In the world of communication protocols, SPI is often considered as "little" communication protocol compared to Ethernet, USB, SATA, PCI-Express and others that present throughput in the ×100 Mb/s range, if not Gb/s. It is important to not forget the purpose of each protocol. Ethernet, USB, and SATA are meant for "outside the box communications" and data exchanges between whole systems. While SPI, as well as others serial protocols such as I2C and 1-wire for instance, are well suited for communications between integrated circuits for low/medium data transfer speed with on-board peripherals.

Consequently, when there is a need to implement a communication between an integrated circuit such as a microcontroller and a set of relatively slow peripherals, there is no point in employing any excessively complex protocols. In this case, SPI stands among the best candidates. This is why it becomes a worldwide standard for modern digital electronics systems and it will probably continue to compete in the future [1][2][3][4][5][6].

This work was supported by Centre de Développement des Technologies Avancées (CDTA) Algiers, Algeria.

The SPI protocol specifications are meticulously defined in [7]. Therefore, they will not be discussed here. Instead, a quick overview is provided in table I.

Although the literature on SPI protocol is so extensive and the topic is so old (early 1980), to the best of the authors knowledge there is no comprehensive analysis of SPI problem. By comprehensive analysis, we mean a treatment that start from Motorola's V03.06 SPI bus specifications and goes down to the actual ASIC/FPGA implementation, discussing all relevant architectural aspects and providing all design details.

In our attempt to implement universal SPI IP cores [8] according to the design-reuse methodology [9], we first made a market study of an important number of recent commercial SPI devices (datasheets) from different vendors [10][11][12] to look at the requirements and what features are to be included to satisfy modern ASIC/SoC applications. The key features required for SPI Master/Slave IP cores as a result of the market investigation are summarized in table II, and their translation into architectures are depicted by figures 1 and 2, respectively.

The paper is organized as follows. In this section we showed the requirement specifications of a recent market investigation for modern SPI IPs and their corresponding architectures. Section two discusses the implementation results. Section three comments the SPI IP integration into SoC environment around OPB bus. And finally some concluding remarks.

TABLE I. SPI PROTOCOL: MAIN FEATURES

Originator	Motorola (1979)
Plug & Play	No
Interface Type	Serial (3+N wires)*
Distance	Short (In-Box Communication)
Application	Transfer of Data-Streams
Protocol Complexity	Low
Design Cost	Low
Transfer Rate	Free (n x MHz to 10n x MHz)
Power Consumption	Low
Transfer Type	Full Duplex
Time Constraint	Synchronous
Multi Master	No
Multi Slave	Yes
I/O Constraints	No Constraint
Addressing	Hardware (Chip Select)
Flow Control	No
Clock Stretching	No

*: N is the number of Slave devices connected to a single Master on the bus.

978-1-4244-7534-6/10/$26.00 ©2010 IEEE

TABLE II. KEY FEATURES OF SPI-MASTER AND SPI-SLAVE IPS

Global Features	Key Features
Data Transfer	• Low/medium (n x MHz to 10n x MHz) data transfer rate depending on implementation issues; • Variable transfer rate (SCLK rate) depending on the SPI-Master baud rate; • User defined SPI word length; • Frame data-transfer (unlimited number of words); • Full duplex mode (bidirectional data transfer); • Interrupt & polling mode transfer.
Addressing	• Hardware addressable (Slave Select input pin nSS); • A maximum of 256 Slave cores can be connected to the same Master core.
Synchronization	• Serial Clock with programmable polarity and phase; • User programmable baudrate (clk/2048 ≤ SCLK ≤clk); • Hardware defined Slave to Master outputs (full & empty) for Master wait-states insertion periods (SCLK clock stretching).
Transfer Error	• Transfer error detection (internal status flags and Slave to Master output signals for overflow and underrun).
Data Integrity	• Spike filtering (removing input spikes shorter than a certain user-defined number of clock cycles).
Host Side Interface	• Simple and basic handshaking protocol easily adaptable to any standard SoC bus [13] via a wrapper [14]. An OPB [15] wrapper is to be developed for SPI IP integration [16][17] into Microblaze [18] based SoC environment.
Performance	• User defined multiple-word write and read buffer (FIFO) to improve transfer performance (reduce lose/duplication of data and wait state occurrences).
Low Power Consumption	• Power conservative state (stop). The SPI Master/Slave are disconnected from the system and internal activity turned off, except for I/O interface.
Parametrizable Functionalities	• Transmitter/Receiver; • With/without FIFO; • With/Without Digital Filter;
Implementation & Test	• Strictly synchronous design with positive edge clocking for straightforward scan-path insertion; • No internal three-state elements.

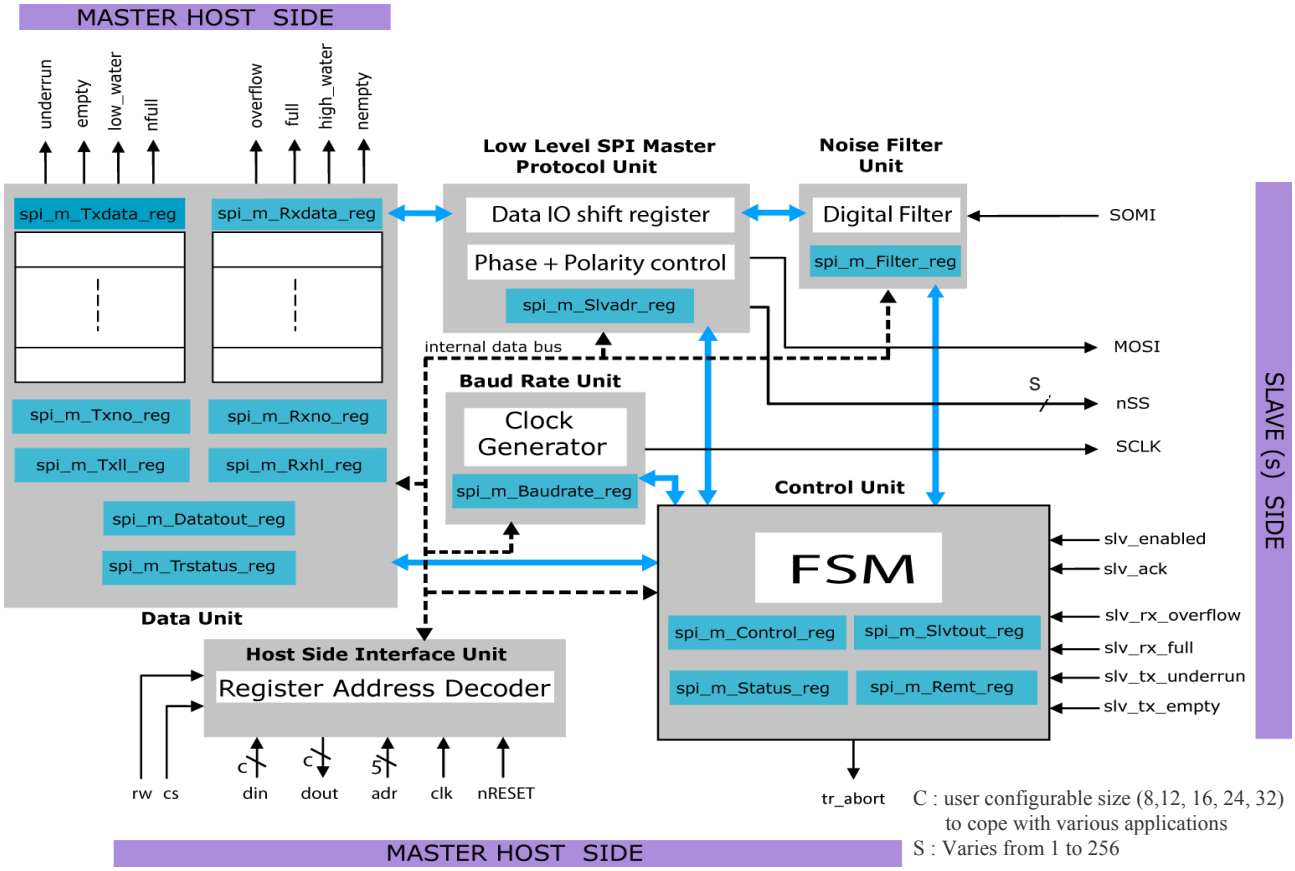

Figure 1. SPI-Master Transceiver Architecture.

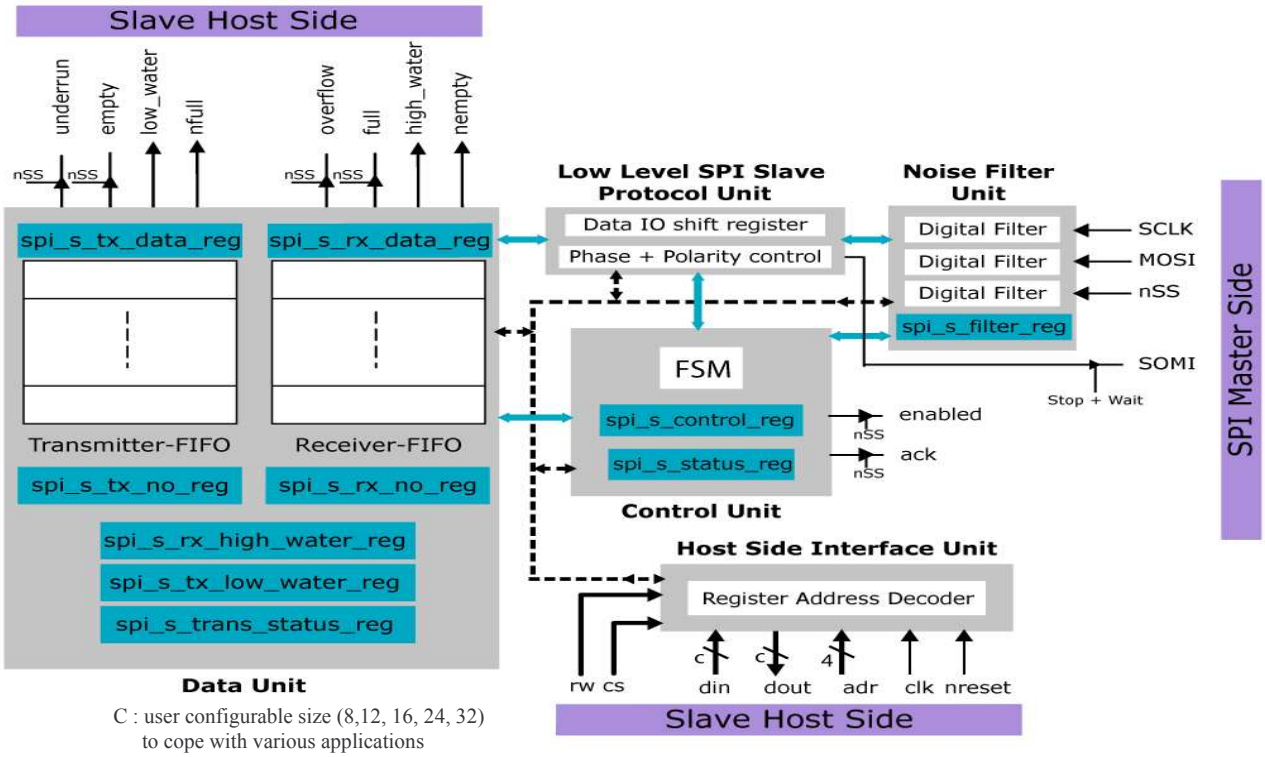

Figure 2. SPI-Slave Transceiver Architecture.

II. IMPLEMENTATION RESULTS

The whole design code, either for synthesis or functional verification, is implemented in Verilog 2001 (IEEE 1365). The synthesis design code is technology independent and was simulated at both RTL and gate level (post place & route netlist) with timing back annotation using ModelSim SE 6.3f and mapped onto Xilinx's FPGAs using Foundation ISE 10.1 version. Both designs have undergone severe functional software verification procedure according to our own IP development methodology summarized in [8]. As for physical test, both designs were integrated around Microblaze SoC environment using V2MB1000 demonstration board [19] with Xilinx's EDK 9.1i version.

The RTL-Code size of SPI-Master is about 1.33 times the code size of SPI-Slave (Table III). The 33% extra code size is mainly due to the additional logic required by the SPI-Master to handle the clock stretching feature (wait-states insertion periods) to cope with different unavailability situations either for the Master or for the Slave (Table I).

The mapping of RTL-code including two 4-byte FIFOs and digital filters onto Xilinx's FPGA devices (Table IV), exhibits a slice utilization average around 510 and 360 for SPI-Master and SPI-Slave respectively, except for Virtex 5 devices where the utilization is around 230 and 140, respectively. This difference is due to the number of look-up-tables (LUTs) per slice, which is: 2 LUTs of 4 inputs each for Spartan 2-3 and Virtex 2-4 devices, and 4 LUTs of 6 inputs each for Virtex 5 devices. Note that some slices are used only for routing.

Nevertheless, whatever the FPGA device; the SPI-Master induces an area-overhead average over SPI-Slave of 40% and 60% when mapped onto Virtex 4 and Virtex 5 devices, respectively.

TABLE III. COMPARISON OF RTL-CODES

Architecture Units	SPI-Master		SPI-Slave	
	Number of Lines	*Size (Ko)*	*Number of Lines*	*Size (Ko)*
Top Module (HSI + FSM)	718	22	434	15
Low-Level Protocol	186	7	217	8
FIFO	232	7	232	7
Filter	67	4	67	4
Baudrate	63	2	/	/
Total	1266*	42	950*	34

* : (1266-950)/950 = 0.33

It is noteworthy to mention that all results, either for slice occupation or delays, are obtained using the default options of the implementation software (Foundation ISE 10.1) with the selection of the fastest speed grade for each FPGA device.

There is almost no significant difference in terms of delays (Table V). Delays are calculated for two types of paths: Clock-To-Setup and all paths together (Pad-To-Setup, Clock-To-Pad and Pad-To-Pad.) The Clock-To-Setup gives more precise information on the delays than other remaining paths, which depend in fact on I/O Block (IOB) configuration (low/high fanout, CMOS, TTL, LVDS…)

TABLE IV. COMPARISON OF OCCUPIED SLICES

		SPI-Master		SPI-Slave	
Xilinx's FPGA Device	***Number of Total Slices***	***Number of Occupied Slices***	***Utilization***	***Number of Occupied Slices***	***Utilization***
xc2s50-6tq144	768 $^{+}$	520	67%	363	47%
xc3s50-5tq144	768^{+}	500	65%	354	46%
xc2v80-6cs144	512^{+}	502	98%	366	71%
xc4vlx15-12sf363	6114^{+}	519	8%	360	5%
xc5vlx30-3ff324	4800^{x}	232	4%	141	2%

+ : each slice includes 2 LUTs of 4 inputs
x : each slice includes 4 LUTs of 6 inputs

TABLE V. COMPARISON OF DELAYS

	SPI-Master		SPI-Slave	
Xilinx's FPGA Device	***Clock-To-Setup Paths***	***All Paths***	***Clock-To-Setup Paths***	***All Paths***
xc2s50-6tq144	12.156 ns	16.750 ns	12.234 ns	17.042 ns
xc3s50-5tq144	8.286 ns	12.802 ns	7.835 ns	12.089 ns
xc2v80-6cs144	6.729 ns	10.022 ns	6.604 ns	10.234 ns
xc4vlx15-12sf363	5.081 ns	9.253 ns	5.006 ns	8.880 ns
xc5vlx30-3ff324	3.520ns	7.449 ns	3.303 ns	6.971 ns

The transfer rate for SPI is unlimited, but a timing relationship between the architecture master clock and the synchronous transfer clock (clk and SCLK on figures 1 and 2) must be known to enable the sampling of smallest events depending on the timing constraints of SPI protocol [7]. The architecture master clock (clk) must be at least 4 times faster than the transfer clock (SCLK). In fact, only a ratio of 2 is required according to SPI bus specifications, but the RTL coding style requires 2 additional clock cycles.

For instance, if we consider the Clock-To-Setup delay of SPI-Slave mapped onto Virtex-5 device (3.303 ns), a transfer rate of 13.212ns can be achieved, corresponding to 75 MBPS.

III. SPI IP INTEGRATION INTO SOC ENVIRONMENT

The SPI Master and Slave IPs are designed as two full dual cores, either in the functional side (Table VI) or in the interface side (Figures 3). According to Motorola's specifications [7], only the four black signals (Figure 4) are required as full-duplex synchronous channel between SPI-Master and SPI-Slave. The six other signals (red and blue) are optional. They are added to raise the transfer security-level and to preserve data-integrity between the Master and the Slave. These added input signals can be masked / unmasked by the Master.

As SPI is mono-Master multi-Slave protocol, up to 256 SPI-Slave cores can be connected to the same SPI-Master core as depicted by figure 5. All shared SPI-Slave outputs driving the SPI-Master inputs (SOMI and red/blue optional signals) are put in high Z except those of the selected SPI-Slave.

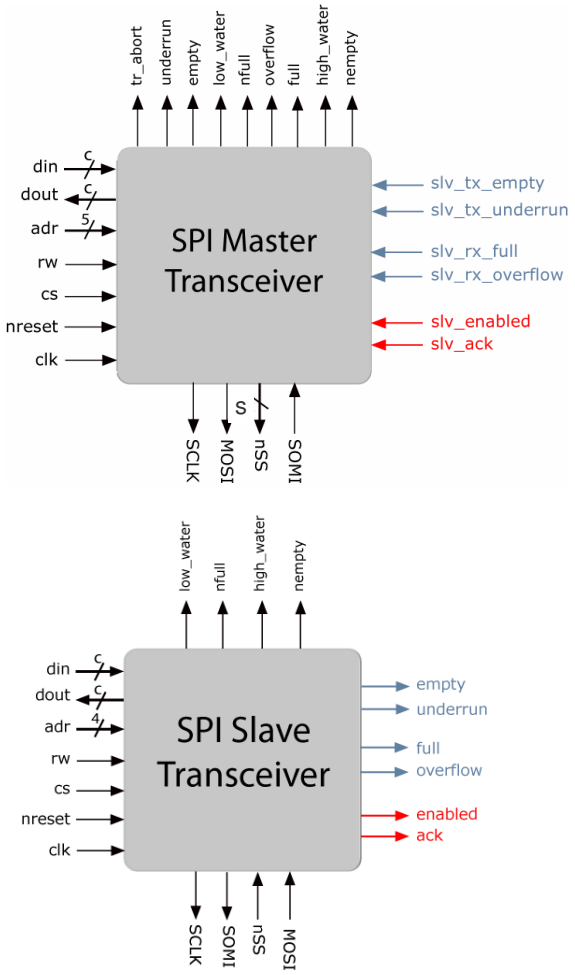

Figure 3. Full Dual SPI Master and Slave IPs

TABLE VI. POSSIBLE STATES OF SPI MASTER AND SLAVE

Possible States	**SPI-Master**	**SPI-Slave**
STOP	SPI interface is disconnected from the system. MOSI and SOMI lines are put into high Z by the master and the Slave, respectively.	
WAIT	Waiting for the permission of the master-host to start transfer.	Waiting to be addressed by the SPI master.
RUN	Frame transfer in progress (full-duplex transfer)	
MASTER BREAK	Clock stretching during user defined period (value of spi_m_Datatout_reg) due to a transceiver interrupt (full or empty) and waiting for the master-host to read/write from/in the Master-FIFO-Transceiver.	Slave in normal *RUN* state
SLAVE BREAK	Clock stretching for a user defined period (value of spi_m_Slvtout_reg) due to a slave interrupt (slv_full or slv_empty) and waiting for the slave-host to read/write from/in the Slave FIFO-Transceiver.	

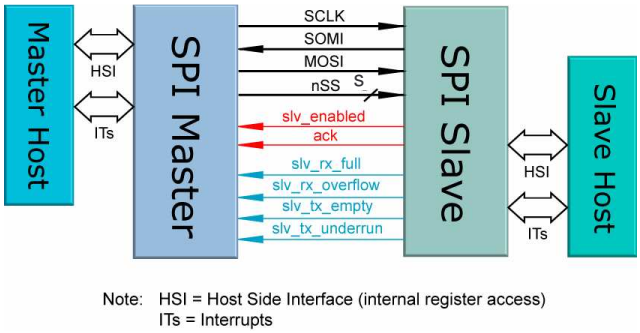

Figure 4. SPI Master and Slave Interaction

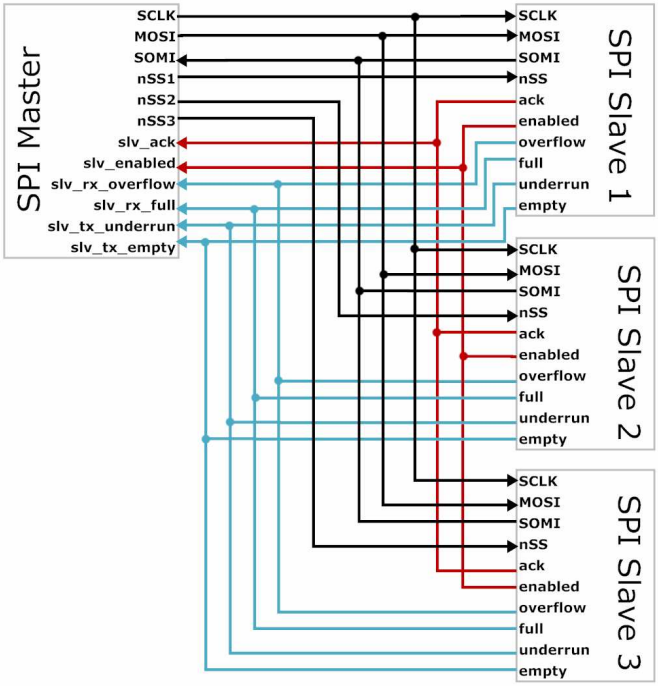

Figure 5. SPI Master/Slave System View

Since we have been using V2MB1000 development board [19], the two SPI IPs have been integrated around Microblaze Microprocessor via OPB bus. MicroBlaze is a standard 32-bit RISC Harvard-style Soft Processor, which is especially developed for the Virtex and Spartan based FPGA architecture [18]. And the OPB is a SoC bus, which is part of the IBM Core Connect on-chip bus standard [15]. The SPI IPs integration was quite straightforward within Xilinx's EDK (Embedded Design Kit) environment. However, as our IPs has been initially provided with a simple and basic Host-Side-Interface (Table I), an OPB wrapper has been developed (Figure 6).

Both Master & Slave IPs have been integrated at the same time (Figure 7). A library of drivers has been developed for each IP to make the C embedded application independent from the hardware.

The test application runs in random mode. First, it initializes the SPI-Master, launches the frame transfer according to a randomly selected polarity and phase (Table I), and then retrieves the transferred frame from the Slave and compares it with the initially transferred frame. If no mismatch occurs, the transfer procedure is repeated again until an error occurs. It was necessary to integrate some other predefined EDK IPs, such as the Interrupt Controller IP and the UART IP to run the test application in interrupt mode and to display some test messages on the screen.

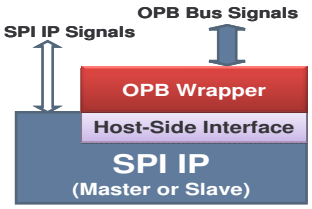

Figure 6. OPB Wrapper

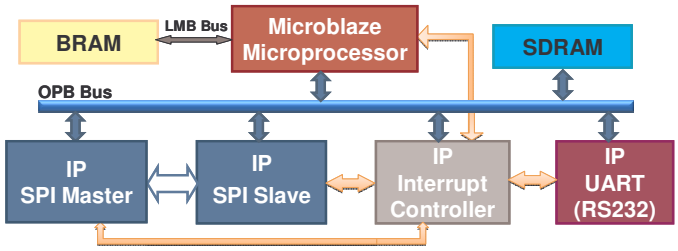

Figure 7. SPI Master/Slave Integration around Microblaze Microprocessor Via OPB Bus

IV. Concluding Remarks

The paper has shown up the results of an up-to-date FPGA implementation of the Master/Slave sides of the standard SPI protocol, which are:

- an utilization ratio of 4% and 2% for SPI-Master and SPI-Slave, respectively, when mapped onto the smallest Viertex-5 FPGA device;
- a maximum transfer rate of 71 & 75 MBPS for SPI-Master and SPI-Slave, respectively.

As the RTL-code is technology independent, much faster transfer rate can be obtained in ASIC implementation using a standard cell library.

In this paper, we have more particularly focused on design features and utilization issues of high-quality SPI Master/Slave IPs based upon design-reuse methodology [9]. We have deliberately occulted the functioning aspect of the design. A meticulous description of the functioning details would require a completely another separate paper.

While the designed Master and Slave cores are full dual IPs, they can also be easily interfaced with any commercial SPI core as both IPs are general-purpose SPI solutions, integrating all necessary features for recent ASIC/SoC application.

Further improvements can be done, such as to integrate a purely digital frequency synthesizer instead of the counter based baud rate to allow a more precise user defined speed rates.

REFERENCES

[1] F. Leens, "An Introduction to I²C and SPI Protocols," IEEE Instrumentation & Measurement Magazine, pp. 8-13, February 2009.

[2] J.M. Irazabel & S. Blozis, Philips Semiconductors, "I²C-Manual," Application Note, ref. AN10216-0, March 24, 2003.

[3] F. Leens, "Solutions for SPI Protocol Testing and Debugging in Embedded System," Byte Paradigm's White Paper, pp. 1-9, Revision 1.00, June 2008.

[4] L. Bacciarelli et al, "Design, Testing and Prototyping of a Software Programmable I²C/SPI IP on AMBA Bus," Conference on Ph.D. Research in MicroElectronics and Electronics (PRIME'*2006*), pp. 373-376, ISBN: 1-4244-0157-7, Ortanto, Italy, June 2006.

[5] R. Hanabusa, "Comparing JTAG, SPI and I²C," Spansion's application note, pp. 1-7, revision 01, April 2007.

[6] P. Myers, "Interfacing Using Serial Protocoles: Using SPI and I²C". http://intranet.daiict.ac.in/~ranjan/esp2005/paper/i2c_spi_341.pdf

[7] Motorola Inc., "SPI Block Guide V03.06," February 2003.

[8] A.K. Oudjida et al, "Front-End IP-Development: Basic Know-How," Revue Internationale des Technologies Avancées, N° 20, pp. 23-30, December 2008, ISSN 1111-0902, Algeria

[9] M. Keating & P. Bricaud, "Reuse Methodology Manual for System on a Chip Designs," ISBN: 1-4020-7141-8, Third Edition, Kluwer Academic Publishers, Copyright 2002.

[10] A.K. Oudjida et al, "FPGA Implementation of I2C and SPI Protocols: A Comparative Study". Proceedings of the 16th edition of the IEEE International Conference on Electronics Circuits and Systems ICECS, pp. 507 -510, ISBN: 978-1-4244-5091-6, December 13-16 2009, Yasmine Hammamet, Tunisia.

[11] A.K. Oudjida et al, "Universal Low/Medium Speed I2C Slave Transceiver: A Detailed FPGA Implementation**,"** Journal of Circuits, Systems and Computers (JCSC), Vol. 17, No. 4, pp. 611-626, August 2008, ISSN: 0218-1266, USA.

[12] "OPB Serial Peripheral Interface (SPI) (V1.00e)," Xilinx Logicore, DS464 July 2006.

[13] R. Usselmann, "OpenCores SoC Bus review," revision 1.0, January 2001.

[14] A.K. Oudjida et al, "Master-Slave Wrapper Communication Protocol: A Case Study," Proceedings of the 1st IEEE International Computer Systems and Information Technology Conference ICSIT'05, pp 461-467, 19-21 July 2006, Algiers, Algeria.

[15] "On-Chip Peripheral Bus, Architecture Specifications," Version 2.1, IBM Corp., Copyright April 2001.

[16] H.P. Rosinger, "Connecting Customized IP to the Microblaze Soft Processor Using the Fast Simplex Link (FSL) Channel," Xilinx XAPP529 (V1.3), May 12 2004.

[17] R. Jesman et al, "MicroBlaze Tutorial, Creating a Simple Embedded System and Adding Custom Peripherals Using Xilinx EDK Software Tools," ECAPS, Illinois Institute of Technology, http://www.ecaps.ece.iit.edu

[18] "Microblaze Processor Reference Guide," Xilinx Inc., EDK (V6.2), June 14 2004.

[19] Xilinx Inc., "Virtex-II™ V2MB1000 Development Board User's Guide". http://ww.cs.lth.se/EDA385/HT06/doc/restricted/V2MB_User_Guide_3_0.pdf