

OPTIMAL IMPLEMENTATION OF UART-SPI INTERFACE IN SOC

T. Praveen Blessington¹, B. Bhanu Murthy², G.V. Ganesh³, T.S.R Prasad⁴

¹Research Scholar, ECE Dept, KL University , India.

²Professor & Vice- Dean, School of Engg & Comp. Tech, Quest International University, Malaysia.

^{3,4}Asst. Professor, ECE Dept, KL University, India.

praveentblessington@gmail.com, Bhanu.b@gmail.com

Abstract- This paper details the design and implementation of SoC's UART-SPI Interface. The UART-SPI interface provides usage for the universal asynchronous receiver/transmitter (UART) to serial peripheral interface (SPI). This interface can be used to communicate to SPI slave devices from a PC with UART port. The interface consists of three blocks: the UART interface, the UART-to-SPI interfacing block and the SPI Master interface.

Keywords- SOC, UART, SPI, power optimization

I. INTRODUCTION

An UART is a device allowing the transmission and reception of information, in a serial and asynchronous way. Universal Asynchronous Receiver and Transmitter are used asynchronous serial data communication between remote embedded systems. The UART can be used to control the process of breaking parallel data from the PC down into serial data that can be transmitted. It consists of one receiver module and transmitter module. UART has been an important input/output tool for decades and is still widely used. UARTs are used for communication between two devices [1]. SPI stands for Serial Peripheral Interface. It is a synchronous protocol that allows a master device to initiate communication with slave devices.

SPI is a full duplex, serial bus commonly used because of its simple hardware interface requirements and protocol flexibility. SPI consists of two blocks. The SPI master and the SPI slave, the SPI Master which is being used in this design implements the master functionality of the SPI protocol. SPI protocol specifies four signal wires MISO - master out slave in (output from master), MISO - master in slave out (output from slave), SCLK - serial clock (clock output from master) and SS - slave select (active low, output from master) [2]. The SPI Master block generates the control signals to interface to external slave devices using the serial data out port (MOSI), serial data in port (MISO), output clock (SCLK) and slave select (SS) [10]. The SS signal must

be used if more than one slave exists in the system. This signal is most often active low, so a low on this line will indicate the SPI is active, while a high will signal inactivity. UART-to-SPI interfacing block that is the middle block joins the UART and SPI master. It helps the interconnection between these two interfaces.

The main advantage is, the UART- SPI interface [9] can fit in any application where an SPI device has to be used. As the UART-SPI interface can be used to communicate to SPI slave devices from a PC with UART port it can be used for typical applications like interfacing of EEPROM, flash memories and sensors [8].

II. SYSTEM-ON-CHIP

The empirical law of Moore does not only describe the increasing density of transistors permitted by technological advances. It also imposes new requirements and challenges, Systems complexity increases at the same speed. Now-a-days systems could never be designed using the same approaches applied 20 years ago. New architectures are and must be continuously conceived. It is clear now that Moore's law for the last two decades has enabled three main revolutions. The first revolution in the mid-eighties was the way to embed more and more electronic devices in the same silicon die; it was the era of System on Chip [8]. One main challenge was the way to interconnect all these devices efficiently. For this purpose, the Bus interconnect structure was used for the VLSI subsystem.

A system usually has an embedded user interface as a form of software and encompasses many components inside, not only the hardware but also the software that constitutes the system. Such a complicated entity can be handled only with computer-aided design tools, automatic synthesis of the physical layouts, and sound software engineering knowledge. In addition, the system functions to achieve a specific goal, as a whole, are usually described in algorithms that should satisfy user requirements in time.

III. UART DESIGN

An UART (Universal Asynchronous Receiver/Transmitter) is the microchip with programming that controls a computer's interface to its attached serial devices. UART is an integrated circuit designed for implementing the interface for serial communications. It provides the computer with the RS-232C Data Terminal Equipment (DTE) interface so that it can "talk" to and exchange data with modems and other serial devices [1]. As part of this interface, the UART also:

- Converts the bytes it receives from the system along parallel circuits into a single serial bit stream for outbound transmission
- On inbound transmission, converts the serial bit stream into the bytes that the system handles
- Adds a parity bit (if it's been selected) on outbound transmissions and checks the parity of incoming bytes (if selected) and discards the parity bit
- Adds start and stop delineators on outbound and strips them from inbound transmissions
- May handle other kinds of interrupt and device management that require coordinating the on-chip communication of operation with high speed devices

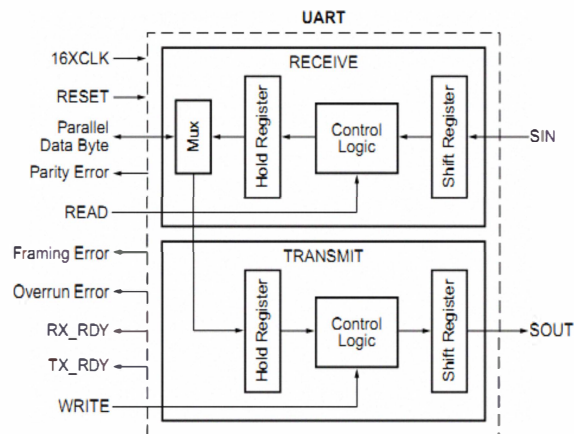


Figure1. UART Block diagram

- The UART includes both transmitter and receiver. The transmitter is a special shift register that loads data in parallel and then shifts it out bit-by-bit. The receiver shifts in data bit-by-bit and reassembles the data byte
- Wait until the incoming signal becomes '0' (the start bit) and then start the sampling tick center. When the center reaches 7, the incoming signal reaches the middle position of the start bit. Clear the center and restart.

- When the center reaches 15, we are at the middle of the first data bit. Retrieve it and shift into a register. Restart the center. Repeat the above step N-1 times to retrieve the remaining data bits. If optional parity bit is used, repeat this step once more. Repeat this step M more times to obtain the stop bits

IV. SPI DESIGN

SPI stands for Serial Peripheral Interface. SPI is a synchronous protocol that allows a master device to initiate communication with a slave device. Data is exchanged between these devices. SPI is implemented by a hardware module called the Synchronous Serial Port or the Master Synchronous Serial Port. This module is built into many different micro devices. It allows serial communication between two or more devices at a high speed and is reasonably easy to implement.

SPI is a Synchronous protocol. The clock signal is provided by the master to provide synchronization. The clock signal controls when data can change and when it is valid for reading [5]. Since SPI is synchronous, it has a clock pulse along with the data. RS-232 and other asynchronous protocols do not use a clock pulse, but the data must be timed very accurately. Since SPI has a clock signal, the clock can vary without disrupting the data. The data rate will simply change along with the changes in the clock rate. This makes SPI ideal when the microcontroller is being clocked imprecisely, such as by a RC oscillator.

SPI is a Master-Slave protocol. Only the master device can control the clock line, SCK. No data will be transferred unless the clock is manipulated. All slaves are controlled by the clock which is manipulated by the master device. The slaves may not manipulate the clock. The SSP configuration registers will control how a device will respond to the clock input. SPI is a Data Exchange protocol [2]. As data is being clocked out, new data is also being clocked in. When one "transmits" data, the incoming data must be read before attempting to transmit again. If the incoming data is not read, then the data will be lost and the SPI module may become disabled as a result. Always read the data after a transfer has taken place, even if the data has no use in your application. Data is always "exchanged" between devices. No device can just be a "transmitter" or just a "receiver" in SPI. However, each device has two data lines, one for input and one for output. These data exchanges are controlled by the clock line, SCK, which is controlled by the master device.

Often a slave select signal will control when a device is accessed. This signal must be used for when more than one slave exists in a system, but can be optional when only one slave exists in the circuit. As a general rule, it should be used. This signal is known as

the SS signal and stands for “Slave Select.” It indicates to a slave that the master wishes to start an SPI data exchange between that slave device and itself. The signal is most often active low, so a low on this line will indicate the SPI is active, while a high will signal inactivity. It is often used to improve noise immunity of the system. Its function is to reset the SPI slave so that it is ready to receive the next byte.

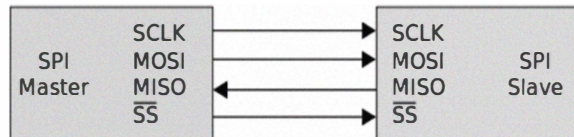


Figure2. SPI Block Diagram

In the master SPI, the bits are sent out of the MOSI pin and received in the MISO pin. The bits to be shifted out are stored in the SPI data register, SP0DR, and are sent out most significant bit (bit 7) first. When bit 7 of the master is shifted out through MOSI pin, a bit from bit 7 of the slave is being shifted into bit 0 of the master via the MISO pin. After 8 clock pulses or shifts, this bit will eventually end up in bit 7 of the master. The least significant bit can be sent out first by setting the LSBF bit to 1 in the SPI Control Register. The clock, which control show fast the bits are shifted out and into SP0DR, is the signal SCLK at PS6. The frequency of this clock can be controlled by the SPI baud rate register SP0BR. The SS pin must be low to select a slave. This signal can come from any pin on the master, including its SS pin when it is configured as an output.

The SPI Control Register1, SP0CR1, is the two bits CPOL and CPHA control the polarity and phase of the clock. If CPOL=0(1), the clock idles low (high) and data are shifted in and out on the rising (falling) edge of the clock if CPHA=0 and on the falling (rising) edge of the clock if CPHA=1(0). If CPHA=1, the SS slave select line can remain low during successive transfers. If CPHA=0, the SS line must be de-asserted and reasserted between each successive byte of data transferred.

SPI is a Serial Interface and uses the following signals to serially exchange data with another device:

SS - This signal is known as Slave Select. When it goes low, the slave device will listen for SPI clock and data signals.

SCK - This is the serial clock signal. It is generated by the master device and controls when data is sent and when it is read.

MOSI – The signal is generated by Master, recipient is the Slave.

MISO -The signals are generated by Slaves, recipient is

the master.

SI - Serial Data Input (used to transfer data into the SPI device).

SO - Serial Data Output (used to transfer data out of the SPI device).

CS - Chip Select Input (for enabling device operation).

W- Write Protect Input (used to guard against program/erase instructions).

HOLD - Hold Input (to pause SPI transaction).

V. INTERFACING

The UART-to-SPI interface can be used to communicate to SPI slave devices from a PC with a UART port. SPI is a full duplex, serial bus commonly used in the embedded world because of its simple hardware interface requirements and protocol flexibility. SPI devices are normally smaller in size (low I/O count) when compared to parallel interface devices. The interfacing diagram is shown below.

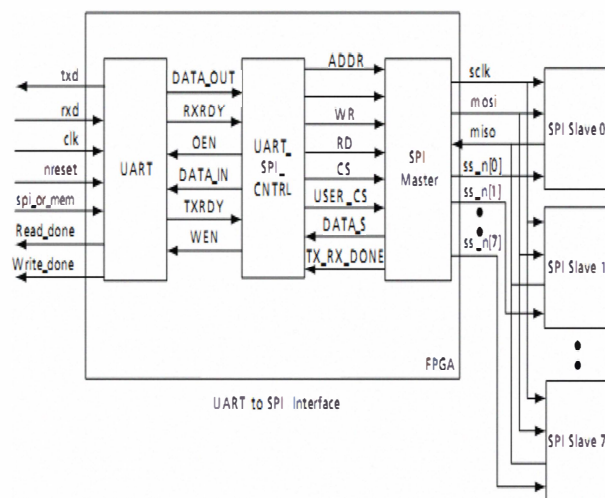


Figure3. UART-SPI Interface Block Diagram

It consists of three blocks, the UART interface, the UART-to-SPI control block, and the SPI master interface. The internal UART-to-SPI control blocks stitches the Core UART and SPI master. The SPI master block generates the control signals to interface to external slave devices. This interface communicates with the slave devices using the serial data out port (MOSI), serial data in port (MISO), output clock (SCLK), and slave select ports (SS_N [7:0]). There are three internal registers in the design: control register, transmit register, and receive register. The control register sets the different control bits, the transmit register sends the TX

TABLE 1 CONTROL BIT POSITIONS

7	6	5	4	3	2	1	0
SS			CPOL	CPHA	CLKDIV		

Other slave devices ignore the clock signal and keep the MISO output pin in a high impedance state, unless the slave select pin is enabled. The SPI_OR_MEM hardcoded value sets the operation mode: when SPI_OR_MEM is set to 1, the slave select signal SS_Nx will be asserted Low for a 1-byte (8 bit) transaction only; when SPI_OR_MEM is set to 0, the SPI slave device will be treated as a SPI memory, and the SS_Nx signal can be asserted Low for multiple bytes of data [3]. This mode is required when performing the page/sector mode of operations with memories.

The slave select will be Low for the command byte, address bytes, and data bytes [10]. When SPI_OR_MEM is set to 1, the command byte 0x01 is used for read operation and the command byte 0x02 is used for write operation shown in table2.

TABLE 2 COMMANDS

Operation	Description
Read	0x01 command byte is sent over UART Tx, Enabling data read from the UART Rx line.
Write	0x02 command byte is sent over UART Tx, followed by the data to be written.

VI. SIMULATION RESULTS

The Interface of UART – SPI in SOC has been synthesized using the Xilinx 10.2. The optimal power calculations are published in table3 and the simulation results are shown in figure4 respectively. The optimal frequency is 239 MHz.

TABLE3. OPTIMAL RESULTS

Estimations	Optimal Results	clock	Input	Output	Signal	Logic
Current (mA)	Ideal	0	26	26	20	20
	Observed	0	18	35	14	14
Power (mW)	Ideal	0	31	31	0.6	1.2
	Observed	0	45	80	1.3	1.7



Figure4. Simulation Results

The utilization of chip area in FPGA is shown in table5

TABLE5. CHIP UTILIZATION

Resource	Availability	Used	Utilization
Slices	768	520	68%
CLBs	228	176	77%
Cells	564	423	75%
LUTs	64	42	66%
IO pins	500	27	6%

VII. CONCLUSION AND FUTURE SCOPE

The Interface of UART – SPI in SOC will come very effective in many applications. The communication in the SOC architecture makes easy as they have been connected with a bus. In future as of more applications will add into the subsystem the routing architecture plays a vital role in the system and it can be implemented in NOC.

REFERENCES

- [1] Design and simulation of UART serial communication module based on VHDL – Fang Yi-Yuan, Chen Xue, IEEE Explore, may 2011.
- [2] Design and test of general purpose SPI master/slave IPs on OPB Bus- systems signals and devices, 7th international multi conference, 2010.
- [3] 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.

- [4] F. Leens, “An Introduction to SPI Protocols,”IEEE Instrumentation & Measurement Magazine, pp. 8-13, February 2009.
- [5] A.K. Oudjida et al, FPGA Implementation of I2C & SPI protocols A Comparative Study”. Proceedings of the 16th edition of the IEEE International Conference on Electronics Circuits and Systems ICECS, pp.507 -510, Dec 13-16 2009.
- [6] REN Yu-fei,ZHANG Xiang,CHENG Nai-ping (Department of Optical and Electrical,Academy of Equipment Command &Tech, Beijing 101416, China); Design and Realization of Two-way Transmission SPI Interface; Telecommunication Engineering; 2009.
- [7] Zhang Rui;A Method to Realize DSP Communicating with Other Device by SPI Interface Protocol [J]; International Electronic Elements; 2003-08.
- [8] A micro- FT- UART for safety critical SOC based Applications, www.doi.ieeecomputersociety.org.
- [9] www.xilinx.com/support/documentation/ipdocumentation/xpspi.pdf
- [10] www.actel.com/documents/UART_to_SPI_AN.pdf.
- [11] www.nxp.com/documents/datasheet/SC16IS752SC16IS762.pdf.
- [12] www.xilinx.com/support/anembeddprocessperipheralother.html.