RESEARCH ARTICLE OPEN ACCESS

Optimal Implementation of UART-SPI Interface in System On Chip

M.Veera Gopi Kishore[1], B.Srinivas[2], Dr M.J.C. Prasad[3]
[1](PG Scholar, Department of ECE, Malla Reddy Engineering College, Hyderabad)
[2](Asst.Professor, Department of ECE, Malla Reddy Engineering College, Hyderabad)
[3](Professor, Department of ECE, Malla Reddy Engineering College, Hyderabad)

ABSTRACT
This paper details the Optimal Implementation of UART-SPI Interface in System On Chip. This paper details the design and implementation of System on chip's UART- SPI Interface. The UART- SPI Interface provides utilization for the universal asynchronous receiver/transmitter (UART) to Serial Peripheral Interface (SPI). This interface can be used to connect 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 enabling the transmission and reception of information, in a sequential and asynchronous way. Universal Asynchronous Receiver and Transmitter are used asynchronous sequential data communication between remote embedded systems. The UART can be used to control the process of breaking parallel data from the PC down into sequential 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 complete duplex, serial bus widely used because of its easy hardware interface specifications 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 MOSI - master out slave in (output from master), MISO - expert in slave out (output from slave), SCLK – serial clock (clock outcome from master) and SS -slave select (active low, output from master) [2]. The SPI Master block produces the control signals to interface to external slave devices using the serial data out port (MOSI), serial data in port (MISO), outcome 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 range will indicate the SPI is effective, while a high will signal inactivity. UART-to-SPI interfacing block that is the middle block joins the UART and SPI master. It allows the interconnection between these two interfaces.

The main benefit is, the UART- SPI interface [9] can fit in any application where an SPI program has to be used. As the UART-SPI interface can be used to connect 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 explain the improving density of transistors allowed by technical developments. It also enforces new specifications and difficulties, Systems complexity improves at the same speed. Now-a-days systems could never be developed using the same techniques used 20 years ago. New architectures are and must be continuously conceived. It is obvious now that Moore's law for the last two decades has allowed three primary 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 primary task was the way to interconnect all these devices effectively. For this purpose, the Bus interconnect structure was used for the VLSI subsystem. A process usually has an embedded user interface as a form of software and encompasses many elements within, not only the hardware but also the software that comprises the system. Such a complicated entity can be managed only with computer-aided design resources, automatic synthesis of the physical layouts, and sound software engineering knowledge. Moreover, it features to accomplish a particular objective, as a whole, are usually described in methods that should fulfill customer requirements in time.

III. UART DESIGN

An UART (Universal Asynchronous Receiver/Transmitter) is the micro-chip with programming that manages a pc's interface to its attached sequential 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 gets from the program along parallel circuits into only one single serial bit stream for outbound transmission.
- On inbound transmission, converts the sequential bit stream into the bytes that the program handles.
- Adds a parity bit (if it's been selected) on outbound transmissions and assessments 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 manage other types of interrupt and device management that need coordinating the on-chip communication of operation with high-speed devices.

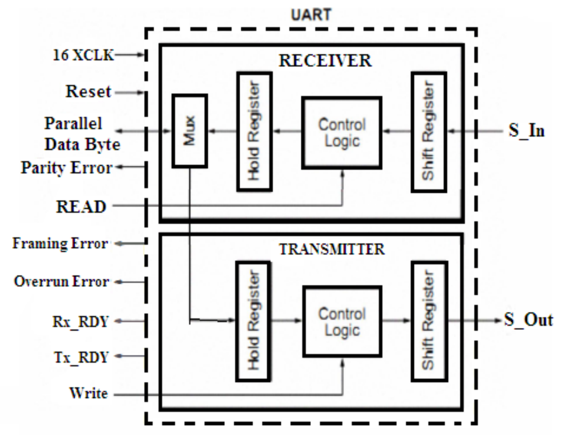


Figure1. UART Block diagram

- The UART contains both transmitter and receiver. The transmitter is a unique shift register that loads data in parallel and then changes it out bit-by-bit. The receiver shifts in information bit-by-bit and reassembles the information byte.
- Wait until the incoming signal becomes ' 0 ' (the begin bit) and then begin the sampling tick center. When the center reaches 7, the incoming signal gets to the middle position of the begin bit. Obvious the center and restart.
- When the center reaches to 15, we are at the center of the first data bit. Retrieve it and move into a register. Restart the center. Do it again the above step N-1 times to retrieve the remaining data bits. If optional parity bit is used, repeat this Process once more. Do it again this step M more times to acquire the stop bits.

IV. SPI DESIGN

SPI means Serial Peripheral Interface. SPI is a synchronous method 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 known as 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 simple 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 modify 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 perfectly. Since SPI has a clock signal, the clock can vary without interfering the data. The data rate will simply modify along with the changes in the clock rate. This creates SPI ideal when the microcontroller is being clocked imprecisely, such as by a RC oscillator.

SPI is a Master-Slave method. Only the master device can manage the clock line, SCLK. No data will be transferred unless clock is manipulated. All slaves are managed by 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 reply to the clock input. SPI is a Data Exchange protocol [2]. As information is being clocked out, new information is also being clocked in. When one "transmits" data, the incoming information must be read before attempting to transmit again. If the incoming information is not read, then the information 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 feedback and one for outcome. These data exchanges are managed by clock line, SCLK, which is controlled by the master device.

Often a slave select signal will control when a device is utilized. 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 concept, it should be used. This signal is known as the SS signal and it means for "Slave Select." It indicates to a slave that the master wishes to begin an SPI data exchange between that slave device and itself. The signal is most often effective low, so a low on this line will indicate the SPI is effective, while a higher 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 get the next byte.

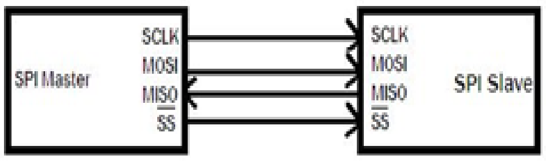


Figure2. SPI Block Diagram

In the master SPI, the bits are sent out of the MOSI pin and obtained 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 servant is being moved into bit 0 of the master via the MISO pin. After 8 clock pulses or shifts, this bit will gradually end up in bit 7 of the master. The least significant bit can be sent out first by establishing the LSBF bit to 1 in the SPI Control Register. The clock, which control show fast the bits are out and into SP0DR, is the signal SCLK at PS6. The frequency of this clock can be managed by the SPI baud amount register SP0BR. The SS pin must be low to decide a slave. This signal can come from any pin on the master, including its SS pin when it is configured as an outcome.

In the master SPI, the bits are sent out of the MOSI pin and obtained 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 servant is being moved into bit 0 of the master via the MISO pin. After 8 clock pulses or shifts, this bit will gradually end up in bit 7 of the master. The least significant bit can be sent out first by establishing the LSBF bit to 1 in the SPI Control Register. The clock, which control show fast the bits are out and into SP0DR is the signal SCLK at PS6. The frequency of this clock can be managed by the SPI baud amount register SP0BR. The SS pin must be low to decide a slave. This signal can come from any pin on the master, including its SS pin when it is configured as an outcome.

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 moved in and out on the increasing (falling) edge of the clock if CPHA=0 and on the dropping (rising) advantage of the clock if CPHA=1(0). If CPHA=1, the SS slave select line can stay low during successive exchanges. If CPHA=0, the SS line must be de-asserted and reasserted between each subsequent 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.

SCLK - 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 110 count) when in comparison to parallel interface devices. The interfacing structure is shown below.

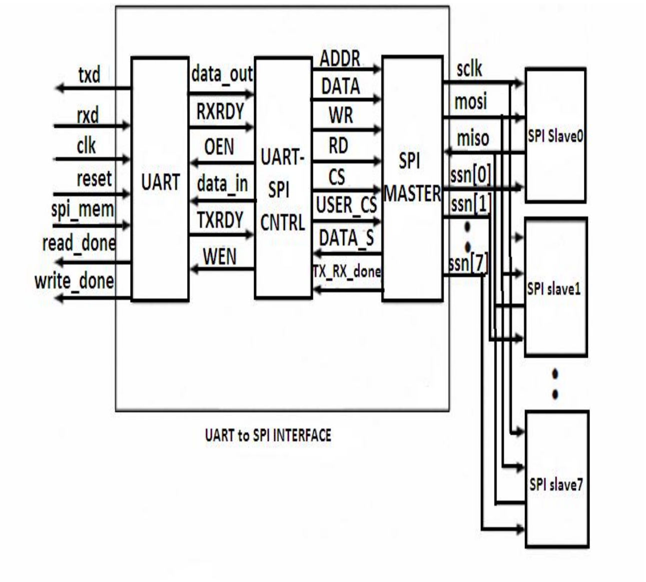


Figure3. UART-SPI Interface Block Diagram

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 data to the SPI bus, and the receive register collects the Rx data from the SPI bus [6]. After every reset, data Collected from the external UART go to the control Register. The control bit positions are given in table 1 which is shown below.

TABLE 1. CONTROL BIT POSITIONS

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

When the UART-to-SPI communicates to any of the slave devices, it enables only the corresponding slave select signal. Only one slave device should be transmitting data during a particular data transfer. Slave devices that are not selected do not interfere with SPI bus activities during that period [7].

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 hard coded 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. SIMUATION RESULTS

The Interface of UART - SPI in SOC has been synthesized using the Xilinx 10.2. The simulation results are shown in figure 4. The optimal frequency is 239 MHz

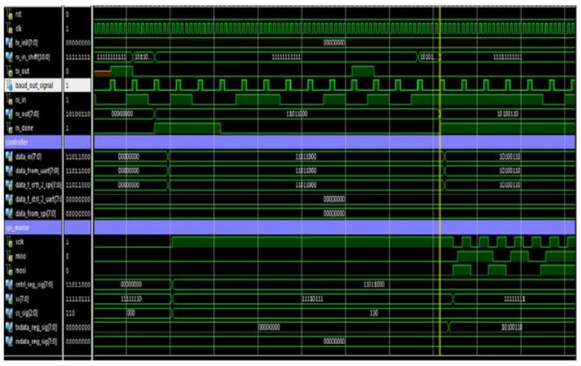
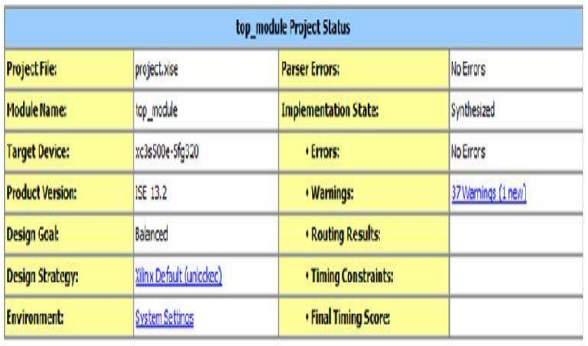


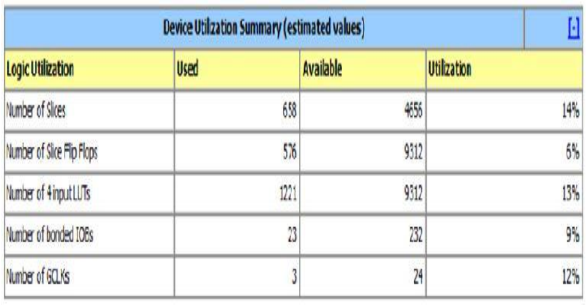
Figure4. Simulation Results

The utilization of chip area in FPGA is shown in table 3.

TABLE3. CHIP UTILIZATION



top_module Project Status			
Project File:	project.xise	Parser Errors:	No Errors
Module Name:	top_module	Implementation State:	Synthesized
Target Device:	xc3s500e-5fg320	• Errors:	No Errors
Product Version:	ISE 13.2	• Warnings:	37 Warnings (1 new)
Design Goal:	Balanced	• Routing Results:	
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	
Environment:	System Settings	• Final Timing Score:	



Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	658	4656	14%
Number of Slice Flip Flops	576	9312	6%
Number of 4 input LUTs	1221	9312	13%
Number of bonded IOBs	23	232	9%
Number of GCLKs	3	24	12%

VII. CONCLUSION AND FUTURE SCOPE

The Interface of UART - SPI in System On Chip will become very efficient method in most of the applications. The communication in the System on chip architecture makes very simple as they are in connection with a bus. In future most of the applications will add into the subsystem the routing architecture plays a vital role in the system and it may be implemented in SOC.

REFERENCES

[I] 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, 7fu international multi conference, 2010.

[3] A.K Oudjida et ai, Master-Slave wrapper communication protocol: A case-study, Proceedings of the 1'1 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 ai, FPGA Implementation of I2C & SPI protocols A Comparative Study". Proceedings of the 16fu 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; Tele communication 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/ipd ocumentationlxpspi.pdf

[10] www.actel.com/documents/UART_to_SPCA N.pdf.

[11] www.nxp.com/documents/datasheet! SCI6IS 752SCI6IS762.pdf.

[12] www.xilinx.com/support/anembeddedproces speripheral other.html