ijournals International Journal of Software & Hardware Research in Engineering
ISSN No: 2347-4890 Volume 2 Issue 1/ January 2014

# Design and Implementation of UART-SPI Controller on System-on-Chip

*Mr.Rakesh Manukonda*
M.Tech. in VLSI &ES,
MLEC, Singarayakonda,

*Mr.Suresh Nakkala*
Asst. Prof. in E.C.E
MLEC, Singarayakonda,

*Abstract*-**This paper gives a gist of the design and usage for the universal asynchronous receiver/transmitter to serial peripheral interface .The UART-to-SPI interface can be used for communicate to SPI slave devices from a PC using 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. UART is a parallel bus to communicate the peripherals with PC.**

**This design example is implemented on an ASIC device, but can easily be implemented in any of low-power field programmable gate arrays (FPGAs) to optimize system power, size, or performance requirements. Implementation of UART-SPI on SOC yields good results.**

***Keywords— power optimization, SOC, SPI, UART.***



## INTRODUCTION

A UART is usually an individual (or part of an) integrated circuit used for serial communications over a computer or peripheral device serial port. UARTs are now commonly included in microcontrollers. A dual UART or DUART combines two UARTs into a single chip. Many modern ICs now come with a UART that can also communicate synchronously; these devices are called USARTs (universal synchronous/asynchronous receiver/transmitter. 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 MISO - master in slave out, SCLK - serial clock and SS - slave select [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) [8] .The SS signal must be used if more than one slave exists in the system. UART-to-SPI interfacing block which is middle block joins the UART and SPI master. It helps the interconnection between these two interfaces.

The main advantage is, the UART- SPI interface [7] can fit in any application where an SPI device has to be used without any difficulties. 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 [6].

.

## SYSTEM ON CHIP

The Moore's law not only states that there will be increase in density of transistors as technology advances. It also inflict new demands and challenges, Complexity of systems also varies at the high rate of speed. As technology advances still the old architecture proposed 20 years back cannot be adopted, new architectures has to be followed. As per the Moore's law 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]. The main challenge was the way to interconnect all these devices effectively to work efficiently. Because of this need we use Bus interconnect structure for VLSI sub system.

An embedded system usually has an embedded user interface as a form of software and consists many components inside, not only the hardware but also the software that constitutes the system is important. Such a complicated and complex entity can be handled only with computer-aided

design tools, automatic synthesis of the physical layouts, and 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.

UART DESIGN

The Universal Asynchronous Receiver/Transmitter (UART) controller is the key component of the serial communications subsystem of a computer. The UART takes bytes of data and transmits the individual bits in a sequential fashion. At the destination, a second UART re-assembles the bits into complete bytes. Serial transmission is commonly used with modems and for non-networked communication between computers, terminals and other devices. After the Start Bit, the individual bits of the word of data are sent, with the Least Significant Bit (LSB) being sent first. Each bit in the transmission is transmitted forfive to eight data bits, least-significant-bit first, an optional "parity" bit, and then one, one and a half, or two "stop" bits. The start bit is the opposite polarity of the data-line's idle state. The stop bit is the data-line's idle state, and provides a delay before the next character can start. (This is called asynchronous start-stop transmission). In mechanical teletypes, the "stop" bit was often stretched to two bit times to give the mechanism more time to finish printing a character. A stretched "stop" bit also helps re synchronization. As part of this interface, the UART also [3]:

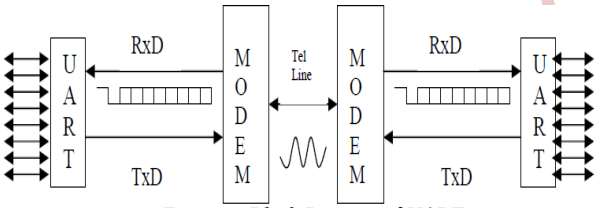


Figure 1. Block Diagram of UART

• 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
• Wait until the incoming signal becomes ' 0 ' (the start bit) and then start the sampling tick centre. When the centre reaches 7, the incoming signal reaches the middle position of the start bit. Clear the centre and restart.

SPI DESIGN

The synchronous four wire serial link i.e. "Serial Peripheral Interface" (SPI) is used to connect microcontrollers to sensors, memory, and peripherals. It's a simple "de facto" standard, not complicated enough to acquire a standardization body. SPI uses a master/slave configuration.

The three signal wires hold a clock (SCK, often on the order of 10 MHz), and parallel data lines with "Master Out, Slave In" (MOSI) or "Master In, Slave Out" (MISO) signals. (Other names are also used.) There are four clocking modes through which data is exchanged; mode 0 and mode-3 are most commonly used. Each clock cycle shifts data out and data in; the clock doesn't cycle except when there is a data bit to shift. Not all data bits are used though; not every protocol uses those full duplex capabilities.

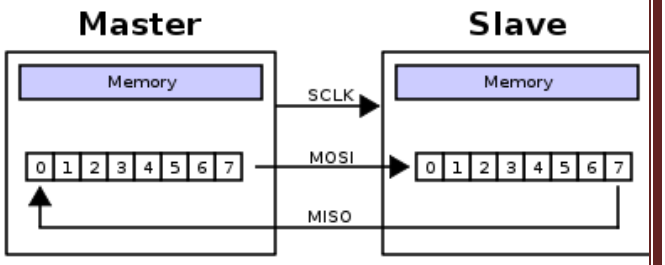


Figure 2. Communication b/w a master and a slave device.
Communication b/w a master and a slave device

SPI consists of one master device and one or more slave devices. If more than one slave device are connected to the master then Slave Select (SS) signal is used which is active low. To begin a communication, the bus master first configures the clock, using a frequency less than or equal to the maximum frequency the slave device supports.

Such frequencies are commonly in the range of 1–100 MHz. The master then transmits the logic 0 for the desired chip over chip select line. A logic 0 is transmitted because the chip select line is active low, meaning its *off* state is a logic 1; *on* is asserted with a logic 0. If a waiting period is required (such as for analog-to-digital conversion), then the master must wait for at least that period of time before starting to issue clock cycles.

During each SPI clock cycle, a full duplex data transmission occurs:
• The master sends a bit on the MOSI line; the slave reads it from that same line
• The slave sends a bit on the MISO line; the master reads it from that same line

Not all transmissions require all four of these operations to be *meaningful* but they do happen. Transmissions normally involve two shift registers of some given word size, such as eight bits, one in the master and one in the slave; they are connected in a ring. Data is usually shifted out with the most significant bit first, while shifting a new least significant bit into the same register. After that register has been shifted out, the master and slave have exchanged register values. Then each device takes that value and does something with it, such

as writing it to memory. If there is more data to exchange, the shift registers are loaded with new data and the process repeats.
Transmissions may involve any number of clock cycles. When there is no more data to be transmitted, the master stops toggling its clock. Normally, it then deselects every slave on the bus that hasn't been activated using its chip select line must disregard the input clock and MOSI signals, and must not drive MISO. The master must select only one slave at a time.

• At CPOL=0 the base value of the clock is zero
• For CPHA=0, data is captured on the clock's rising edge (low→high transition) and data is propagated on a falling edge (high→low clock transition).
• For CPHA=1, data is captured on the clock's falling edge and data is propagated on a rising edge.
• At CPOL=1 the base value of the clock is one (inversion of CPOL=0)

- For CPHA=0, data is captured on clock's falling edge and data is propagated on a rising edge.
- For CPHA=1, data is captured on clock's rising edge and data is propagated on a falling edge.

This is, CPHA=0 means sample on the leading (first) clock edge, while CPHA=1 means sample on the trailing (second) clock edge, regardless of whether that clock edge is rising or falling. Note that with CPHA=0, the data must be stable for a half cycle before the first clock cycle. For all CPOL and CPHA modes, the initial clock value must be stable before the chip select line goes active. The MOSI and MISO signals are usually stable (at their reception points) for the half cycle until the next clock transition. SPI master and slave devices may well sample data at different points in that half cycle. This adds more flexibility to the communication channel between the master and slave

## UART SPI INTERFACING

This design consists of three blocks: the UART block interface, UART-SPI controller block and SPI master interface. The UART block used in this design is core UART. This block handles the data at the UART end. The UART-SPI controller block which is used internally stitches the core UART and SPI master. SPI master block generates control signals to interface external devices. This interface communicates to 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.

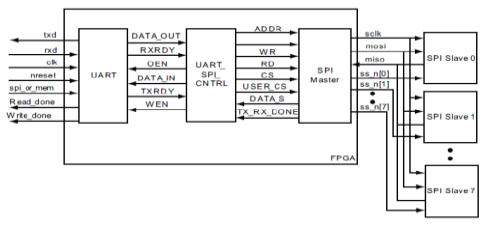


Figure 3. UART-SPI Interface

During write operation rxd receives the input then DATA_OUT to controller and receive ready (RXRDY). Controller generates address (ADDR), data , write (WR) , chip select (CS) and user chip select (USER_CS) and gives to SPI master which then generates serial clock and MOSI. At the same time, data from eeprom has gien to slae interace is forward the data to the spi master, spi master will take the data from spi slae through miso pin after completion of 8-bits it will set the tx_rx done pin. On activating the tx_rx_ done pin interface will send the RD, addr values and it will take the data from the spi master.

Next it will check whether the UART Transmitter is free or not by checking the TXRDY pin. If transmitter is free then interface will send the data to the transmitter. Transmitter will transmit the data and set the TXRDY pin. Otherwise it will wait until transmitter is free.

The control register sets the different control bits, the transmitter register sends the TX data to the SPI bus, and the receive register receives the Rx data from the SPI bus [4]. After every reset, data received from the external UART go to the control register. The control bit positions are given in Tabel 1.

**Table 1**: Control Bit Positions

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

When UART-to-SPI communicates in the design only corresponding slave will be selected. Because only one slave device should be transmitting data during a particular data transfer. Slave devices that are not selected do not interface with SPI bus activities during that period [5] other slave devices will ignore the clock signal and keep MISO in high impedance state, unless the slave select pin is enabled. The SPI_OR_MEM 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 deice will be treated as SPI memory, SS_Nx will be asserted low for multiple bytes of data [3].The Slave select will be low for the command byte, address bytes [8]. When SPI_OR_MEM is set to 1 then the following commands seen in table takes place

**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., following by the data to be written. |

PERFORMANCE EVALUATION

A data transaction from UART to SPI slave devices through UART-SPI controller for an SOC has been synthesized using the Xilinx 14.2 and the simulation results are shown in figure and figure respectively.

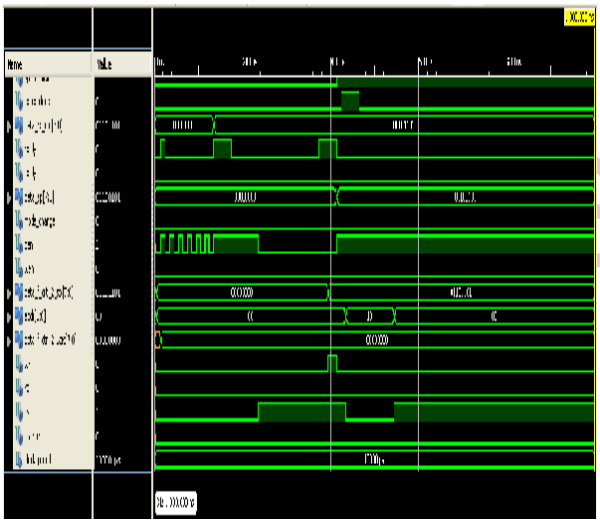

Figure 3. Simulation Results

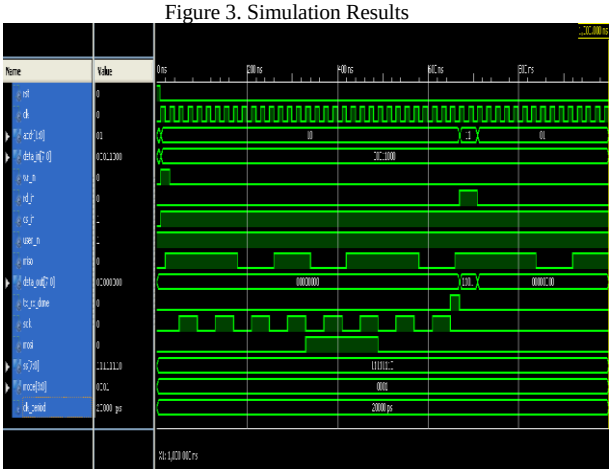

Figure 4. Simulation Results

Table 3: Device Utilization Summary

| 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% |

CONCLUSION

UART-SPI controller for an SOC was very effective in many applications. The SOC architecture makes easy as they have been connected with a bus for communication. The future scope of this design as of more applications will add into the subsystem of the routing architecture plays a vital role in the system and it can be implemented in Network-on-Chip.

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, 7fu international multi conference, 2010.
[3] A.K Oudjida et ai, Master-Slave wrapper communication protocol: A casestudy, Proceedings of the 1'1 IEEE International Computer Systems and Information Technology Conference ICSIT'05, PP 461-467, 19-21 July 2006.
[4] REN Yu-fei,ZHANGXiang,CHENGNai-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.
[5] Zhang Rui;A Method to Realize DSP Communicating with Other Device by SPI Interface Protocol [J]; International Electronic Elements; 2003-08.
[6] A micro- FT- UART for safety critical SOC based Applications, www.doi.ieeecomputersociety.org.
[7] www.xilinx.comlsupport/documentationlipdocumentationlxpspi.pdf
[8] www.actel.comldocuments/uART_to_SPCAN.pdf