ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 5, September 2013

Implementation of Bridging Between SPI & UART

Pooja Patel, M.tech, Embedded system and VLSI design
Lalit Bandil, Assistant Professor
Acropolis Institute of Technology and Research, Indore, M.P.

Abstract: ***There are multiple ways to establish communication link between a device and PC i.e. via parallel port, serial port, ethernet port and as such many other links are also possible. Also there are a lot of devices available, which follow serial communication, but they are unable to communicate directly to PC, since standard PC interface does not follow these serial communication interface (like I2C, SPI and many others).Thus we developed a kind of interface module which enables us to interface SPI device to get directly interfaced with PC through UART. To interface many serial devices to the PC via the single UART port, at present we had worked upon one such interface. In this paper we would present UART-SPI interface which enables us to transfer data to multiple devices through a single UART communication port of the PC. It also opens the gate where we can have multiple SPI device communicating through a single UART interface found in controller. As we found that normally the controller like 8051facilitating on chip UART communication but for the SPI devices, embedded programmer has to develop routine to communicate through such device.***

I. INTRODUCTION

Services using different communication protocols cannot exchange data. Bridging logic is introduced to enable communication between different communication protocols by dynamically converting one protocol to another at runtime. Instead of connecting directly to each other, consumer programs and services connect to a broker, which provides bridging logic that carries out the protocol conversion. For establishing serial communication interface between SPI and PC via UART, we use the bridging technique. Here we interface SPI device which is synchronous device to the UART of PC which is asynchronous via the Bridging Block. The SPI device exchanges the data with the bridging block via SPI controller. The bridging block then converts the data to the format acceptable by the UART of the PC via UART controller. Thus, by using this UART-SPI interface we can interface many other devices to the PC as SPI slaves via the bridging block.

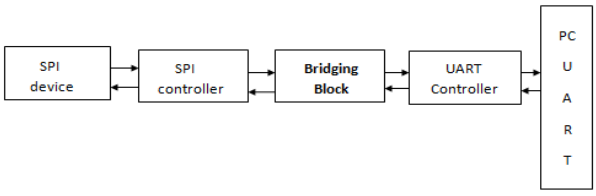

Fig 1: Bridging between SPI and UART

II. LOGIC IMPLEMENTATION

During communication interface, as a bridging block we use a Memory controller block where Asynchronous FIFO is used. FIFOs are used commonly in electronic circuits for buffering and flow control which is from hardware to software. In hardware form a FIFO primarily consists of a set of read and write pointers, storage and control logic. Storage may be SRAM, flip-flops, latches or any other suitable form of storage. For FIFOs of non-trivial size a dual-port SRAM is usually used where one port is used for writing and the other is used for reading [1][2]. An asynchronous FIFO uses different clocks for reading and writing. Asynchronous FIFOs introduce metastability issues. A common implementation of an asynchronous FIFO uses a Gray code (or any unit distance code) for the read and write pointers to ensure reliable flag generation [5]. A hardware FIFO is used for synchronization purposes. It is often implemented as a circular queue, and thus has two pointers:

1. Read Pointer/Read Address Register
2. Write Pointer/Write Address Register

Read and write addresses are initially both at the first memory location and the FIFO queue is **Empty**. FIFO full and FIFO empty flags are of great concern as no data should be written in full condition and no data should be read in empty condition, as it can lead to loss of data or generation of non relevant data. The full and empty

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 5, September 2013

conditions of FIFO are controlled using binary or gray pointers. Below is the block diagram to depict this basic FIFO operation. [3]

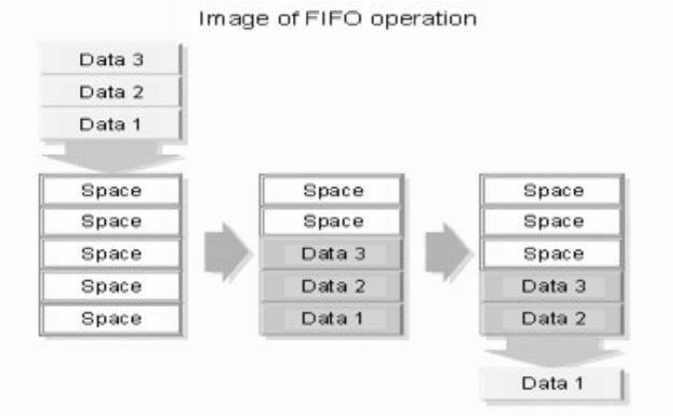

Fig 2:The Basic FIFO Algorithm Technique

Asynchronous FIFO design is done by asynchronous comparisons between the FIFO write and read pointers that are generated in clock domains that are asynchronous to each other. To increase the speed of the FIFO, this design uses combined binary/Gray counters. The Gray code counter style, uses two sets of registers, one a binary counter and a second to capture a binary-to-Gray converted value [4]. The binary counter conditionally increments the binary value, which is passed to both the inputs of the binary counter as the next-binary-count value, and is also passed to the simple binary-to-Gray conversion logic, consisting of one 2-input XOR gate per bit position. The converted binary value is the next Gray-count value and drives the Gray code register inputs. This implementation requires twice the number of flip-flops, but reduces the combinatorial logic and can operate at a higher frequency. In FPGA designs, availability of extra flip-flops is rarely a problem since FPGAs typically contain far more flip-flops than any design will ever use. In FPGA designs, reducing the amount of combinational logic frequently translates into significant improvements in speed. As with any FIFO design, correct implementation of full and empty is the most difficult part of the design. There are two problems with the generation of full and empty: First, both full and empty are indicated by the fact that the read and write pointers are identical. Therefore, something else has to distinguish between full and empty. The Quartus FPGA logic to implement the decoding of the two **wptr** MSBs and the two **rptr** MSBs is easily implemented. The second, and more difficult, problem stems from the asynchronous nature of the write and read clocks. Comparing two counters that are clocked asynchronously can lead to unreliable decoding spikes when either or both counters change multiple bits more or less simultaneously. The solution described in this paper uses a Gray count sequence, where only one bit changes from any count to the next. Any decoder or comparator will then switch only from one valid output to the next one, with no danger of spurious decoding glitches. The FIFO style does asynchronous comparison between Gray code pointers to generate an asynchronous control signal to set and reset the full and empty flip-flops.

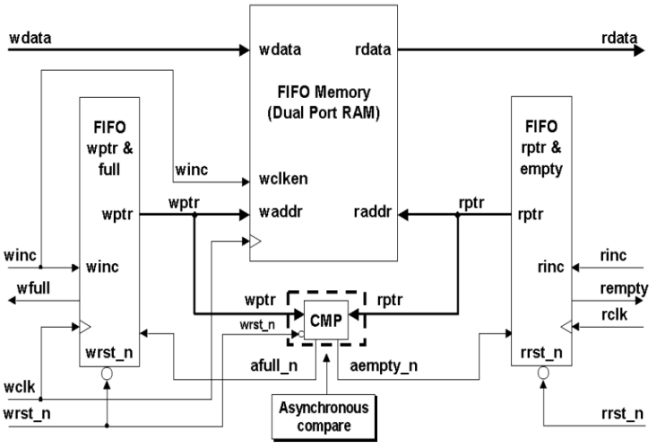

Fig 3:FIFO2 partitioning with asynchronous pointer comparison logic

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 5, September 2013

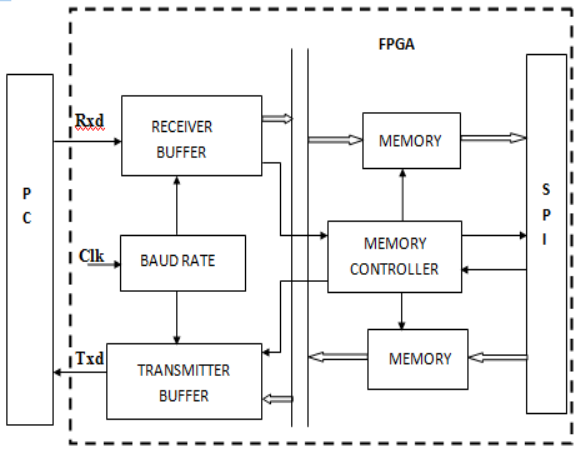

Fig 4: FPGA implementation of the interface between PC and SPI

Here the UART controller's receiver buffer receives data from UART and sends it to the Memory via bus. Memory controller sends the information regarding the received data to the memory, and allows the memory to receive the data placed on the bus by receiver buffer. Memory then sends the received data to the SPI via SPI controller. The information is send to the SPI controller via memory controller of the bridging block. SPI transfers the data to the bridging block via SPI controller. It comes first to the memory controller in turn allows the memory to receive the transmitted data by the SPI. Now the memory controller allows the memory to transmit the data to the transmitter buffer. The transmitter buffer in turn transmits data to the PC through UART. The receiver buffer and transmitter buffer are controlled by the Baud rate having independent clock. Baud rate block is used to set standard baud rate for the receiver buffer and transmitter buffer. The Memory controller's main function is to have an eye on the received and transmitted data. It also controls that which data is to be received and which data is to be transmitted.

III. SIMULATION RESULTS

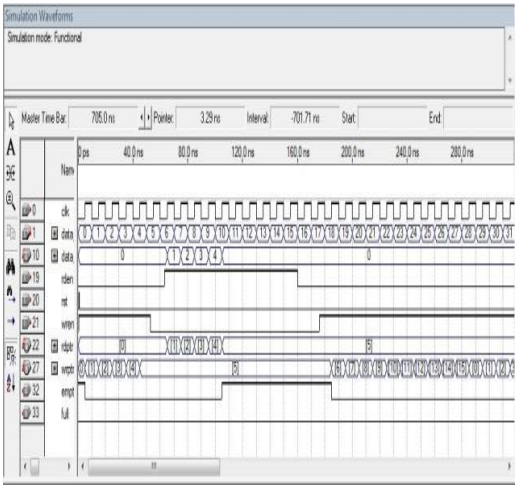

Fig 5: Asynchronous FIFO implemented RAM

The simulation waveform depicts that the read pointer is incremented by one when read enable pin is high and the write pointer is incremented by one when write enable pin is high. Thus the write pointer takes the input value as output upto the point when the write enable pin is high and the read pointer reads the written value upto the point when the read enable pin is high. Thus both the read and write pointers are compared simultaneously the performance is going on, and the full and empty conditions are also satisfied according to the comparison of the two pointers.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 5, September 2013

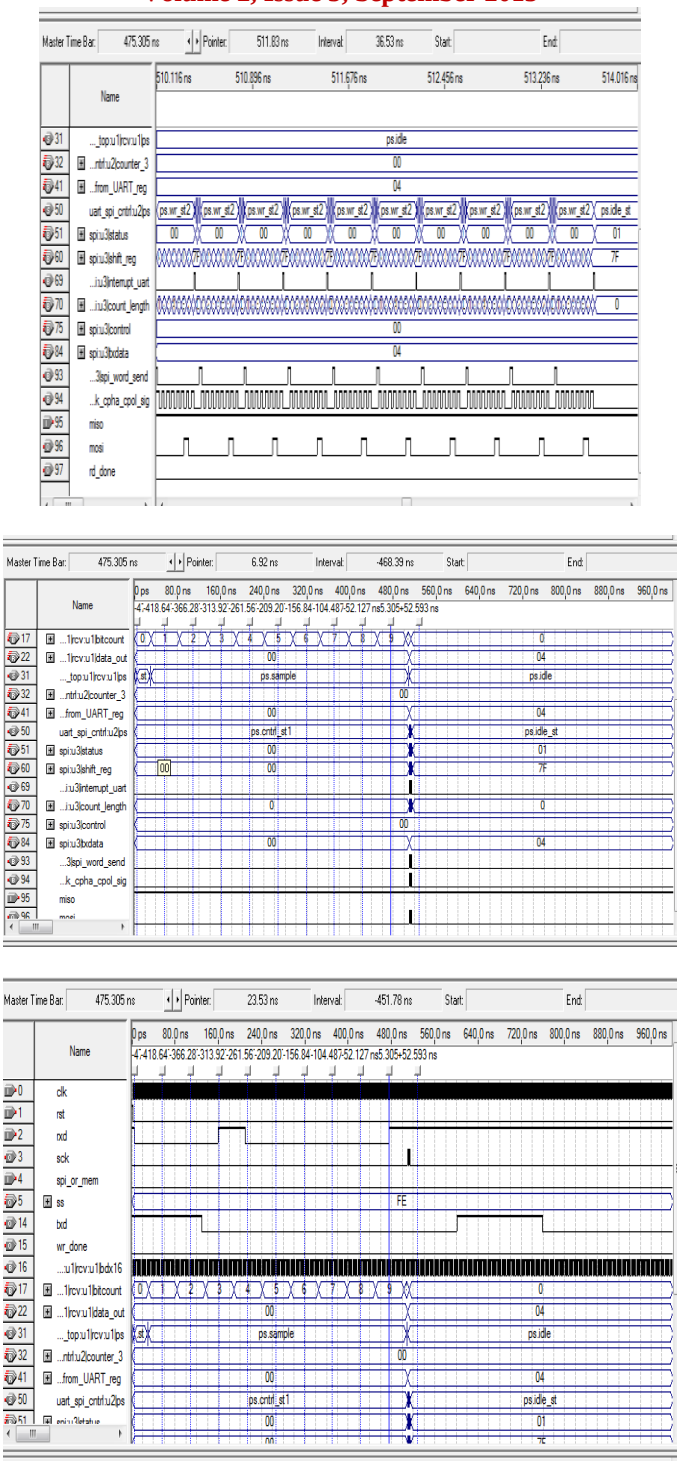

Fig 6: Waveform depicting UART-SPI Interface

The simulation waveform depicts that, we have applied free running clock to the system, and then a reset pulse is applied to reset the system. After that we have serially sent 04 (hex) (00000100) at Rxd. Then we checked whether the rcv_full activates UART_SPI controller. As soon as it receives rcv_full UART_SPI controller gets activated and according to SPI control state machine it comes in execution. Then as per state machine, SCK is generated and the data is being transferred through MOSI. First the state was idle, then delay then output enable

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 5, September 2013

then write state1 then dummy write state1 and then finally write state 2 comes, where the actual data is written from UART to the SPI slave.

IV. CONCLUSION

This paper describes an efficient technique to interface SPI to the UART by implementing a high-speed asynchronous FIFO, using dual-port RAMs addressed by Gray counters. This design uses an asynchronous comparator for detecting full and empty status. This efficient and interesting approach of interfacing is worthy of consideration. The FIFO Memory designed in this paper work is a part of the continuous process of application and there is enough scope to extend further this thesis work beyond the limit of the requirement of the present curriculum. The FIFO memory is essentially built of some logic gates and the most striking feature of any digital circuit is its power consumption. Hence there is always a scope to minimize the power consumption of the constituent parts of the address generation unit by redesigning it using low power techniques. Moreover, this simulation has been primarily done on Quartus platform which may be designed using higher versions of the same software, in future course of work.

REFERENCES

[1] R J Tocci; “Digital Systems- Principles & Applications”; sixth edition; August, 2001, page: 650.

[2] M. Morris Mano; “Computer System Architecture” third edition,; 2008, pp : 450-452 & 690.

[3] “Design, Verification and Synthesis of Synchronous FIFO”; Manipal Institute of Technology, India; Science Direct; 2002.

[4] Clifford E. Cummings, “Simulation and Synthesis Techniques for Asynchronous FIFO Design,” SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers, March 2002, Section TB2, 2nd paper. Also available at www.sunburst-design.com/papers.

[5] Frank Gray, "Pulse Code Communication." United States Patent Number 2,632,058. March 17, 1953.