International Journal of VLSI System Design and Communication Systems

ISSN 2322-0929
Vol.03, Issue.04,
July-2015,
Pages:0367-0371

www.ijvdcs.org

Simulation and Implementation of Multi-Channel UART Controller Based on FIFO and FPGA

BOORTHULA KIRAN[1], MD. SHAZIA FATIMA[2], P. V. N. SWAMY[3]

[1]PG Scholar, Jyothishmathi Institute of Technological Sciences, Karimnagar, TS, India.
[2]Assistant Professor, Jyothishmathi Institute of Technological Sciences, Karimnagar, TS, India.
[3]Associate Professor, Jyothishmathi Institute of Technological Sciences, Karimnagar, TS, India.

Abstract: Multi-channel UART controller implementing on FPGA (UART a kind of serial communication circuit is used widely. A universal asynchronous receive/transmit (UART) is an integrated circuit which plays the most important role in serial communication. The architecture of the system is introduced FIFO at the Trans meter side and receiver side. This FIFO plays main role in trans meting data and reviving data The flow charts of data processing as well as the implementation state machine are also presented in detail. The controller can be used to implement communications in complex system with different Baud Rates of sub-controllers. The controller is reconfigurable and scalable and it also can be used to reduce time delays between sub-controllers of a complex control system to improve the synchronization of each sub-controller.

Keywords: Universal Asynchronous Receive/Transmit (UART), FIFO.

I. INTRODUCTION

Serial communication is the process of sending data and receiving one bit of data at one time sequentially through a communications channel or computer bus .On the other hand, parallel communications is a process where all the bits of each symbol are sent together. In general, serial communication is used for all long-haul communications and most computer networks where it is impractical to use parallel communications due to the cost of cable and synchronization. Nowadays computer buses or network communication using serial communications are becoming more common as improved technology enables them to transfer data at higher speeds. There are 2 types of serial communication, full-duplex and half duplex. A full duplex device can send and receive data at the same time. Thus, a full duplex communication needs 2 different ports, one for serial in data while another for serial out data. On the other hand, half duplex serial devices support only one-way communications and therefore only able either receiving or transmitting data at a time. Normally half duplex devices share the same port for both serial in and out. Although IOP designed in this project has 2 dedicated port serial in and serial out for transmitting and receiving data, however IOP is considered as half-duplex device as IOP only have one control unit to manage the receive and transmit traffic at a time.

Serial communication is an essential to computers and allows them to communicate with low speed peripheral devices, such as the keyboard, the mouse, modems etc. Thus, the UART or Universal Asynchronous Receiver/ Transmitter is the most important component required in serial communication. Asynchronous communication is performed between two (or more) devices if the devices operate on independent clocks. This is because there is no guarantee that the clocks of the communicating devices will have the exact frequency and phase overextended periods. To combat this problem, asynchronous communication requires additional synchronization bits to be added around actual data in order to maintain signal integrity. Fig.1 below illustrates the waveform of an asynchronous serial data stream.

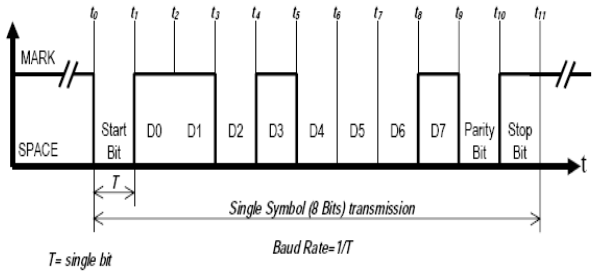

Fig.1. serial data stream.

In asynchronous communication, data is preceded with a start bit (Space) that indicates to the receiver that a word (a chunk of data broken up into individual bits) is about to begin. The end of a word is followed by a stop bit (Mark), which tells the receiver that the word has come to an end. Now it should begin looking for the next start bit. Any bits it receives before getting the start bit should be ignored. To insure data integrity, a parity bit is often added between the last bit of data and the stop bit. The parity bit ensures that the data received is composed of the same number of bits in the same order in which they were sent.

II. UART DESIGN

Universal asynchronous receiver/transmitter (UART) is an asynchronous serial receiver/transmitter. It is a piece of computer hardware that commonly used in PC serial port to translate data between parallel and serial interfaces. The UART takes bytes of data and transmits the individual bits in a sequential fashion. At the receiving point, UART re-assembles the bits into complete bytes. Asynchronous transmission allows data to be transmitted without having to send a clock signal to the receiver. Thus, the sender and receiver must agree on timing parameters in advance and special bits are added to each word which is used to synchronize the sending and receiving units. In general, UART contains of two main block, the transmitter and receiver block. The transmitter sends a byte of data bit by bit serially out from UART while UART receiver receives the serial in data bit by bit and converts them into a byte of data. UART starts the data transmission by asserting a bit called the "Start Bit" to the beginning of each data that is to be transmitted. The Start Bit is also used to inform the receiver that a byte of data is about to be sent. After the Start Bit, the individual bits of the byte of data are sent, with the Least Significant Bit (LSB) being sent first. Each bit in the transmission is transmitted for exactly the same amount of time as all of the other bits.

On the other, UART the receiver will need to sample the logic value that being received at approximately halfway through the period assigned to each bit to determine if it is logic 1 or logic 0. When a byte of data has been sent, the transmitter may add a Parity Bit. The Parity Bit may be used by the receiver to perform simple error checking. In this project, parity bit is not being implemented. After this, a Stop Bit is sent by the transmitter to indicate the transmitter has completed the data transmission. If another byte of data is to be transmitted, the Start Bit for the new data can be sent as soon as the Stop Bit for the previous word has been sent. Figure 2.1 below shows the typical UART data frames format that used by the IOP UART module in this project. The speed of the serial connection is measured in bits-per-second or normally expressed as "baud rate". The duration of a bit is dependent on the baud rate. The baud rate is the number of times the signal can switch states in one second. Thus, if the line is operating at 9600 baud, the line can switch states 9,600 times per second. This means each bit has the duration of 1/9600 of a second or about 100 micro seconds. In this project the baud rate of UART module in IOP is set as 9600. As shown in Figure 2.1, each character or byte requires 10 bits to be transmitted. Thus, IOP is able to transfer 960 bytes of data in a second.

A. UART Architecture

The UART circuit enables a computing processing unit (CPU) serial access to the external peripheral. The interface between the CPU and the UART is usually byte parallel and can be synchronous (i.e. Register Map interface). The transmission properties of the UART, such as parity check, number of symbol bits, number of stop bits etc., can be programmed via a control register which is part of the UART circuitry. The CPU can configure the UART by writing the specific control bits via the parallel interface. Fig.2 below illustrates the block diagram of an UART circuit, including its interface.

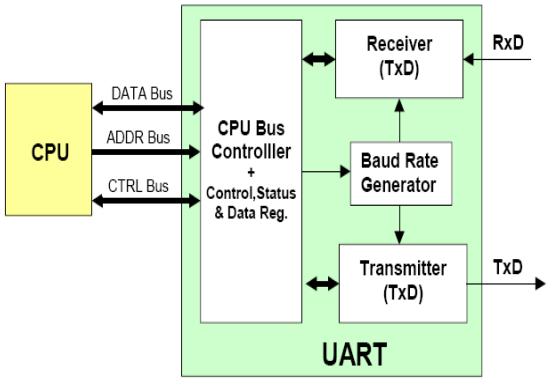

Fig 2.Simplified Block Diagram Of An UART.

The UART consists of the following four main function blocks:

- " CPU Bus Controller
- " Baud rate generator
- " Receiver
- " Transmitter

B. CPU Bus Controller

The CPU Bus Controller provides the parallel data I/O interface to the local processor bus. It generates the necessary control signal to enable the uP (or CPU) to access onto the data, status and control register of the UART circuit. Furthermore, it generates the local control signal within the UART circuit according to the control register content and updates the status register content according to the local status signal values generated by the other function blocks. It also accommodates the transmitter and receiver buffer (Hold Register or FIFO) that is essential for asynchronous transmission.

C. Baud Rate Generator

The Baud Rate Generator is a programmable transmit and receive bit timing device. Given the programmed value, it generates a periodic pulse, which determines the baud rate of the UART transmission. This pulse is used by the receiver and transmitter circuit to generate a sampling pulse for sampling the received serial data and to determine the bit width of the transmit data.

D. Receiver

The Receiver block detects the start bit of an incoming serial data and samples the data bits, bit by bit, according to the baud clock of the baud rate generator. It completes the receive process of a single symbol of 6,7 or 8 bits with the detection of the stop bit (the stop bit can be 1, 1.5 or 2 bits width). Parity check of the received symbol ensures that the data has been received correctly. In the case of invalid stop bit or parity check error, the status signals parity error or

frame error will be set. Finally the receiver writes the received symbol data onto the local data bus, which connected to the CPU Bus controller, and sets a signal to indicate _Receiver data Write_. This signal initiates that the CPU Bus controller informs the CPU via an interrupt about the data arrival.

E. Transmitter

The transmitter block is responsible for the serial transmitting of the data, which is written by the uP (CPU) onto the TxD Hold register (or FIFO) at the CPU Bus controller block. First the transmitter detects whether the UART transmitter buffer (FIFO or TxD Hold Register) contains data for transmission. If it does, it loads the data onto the transmit register at the transmitter circuit via the local data bus (which connects the CPU Bus controller and the transmitter) and sets a signal to indicate _Transmit data Read_. This signal initiates a sequence where the CPU Bus controller informs the CPU via an interrupt about the transmitted data, so that the CPU can load a new value. Synchronous to the baud clock which is generated by the baud rate generator, the transmitter sets the start bit on the TxD signal line to initiate the start of a frame and then bit by bit the symbol data. It finally completes the transmission by sending a parity bit that represents the parity of transmitted data and completes the frame with the final stop bit. The procedure will be repeated for another symbol, if the transmitter buffer contains another symbol, else the transmitter goes to an idle mode, transmitting _Marks.

III. FIFO ARCHITECTURE FIFO TYPES

Every memory in which the data word that is written in first also comes out first when the memory is read is a first-in first-out memory. Fig.3 illustrates the data flow in a FIFO. There are three kinds of FIFO:

- Shift register – FIFO with an invariable number of stored data words and, thus, the necessary synchronism between the read and the write operations because a data word must be read every time one is written
- Exclusive read/write FIFO – FIFO with a variable number of stored data words and, because of the internal structure, the necessary synchronism between the read and the write operations.
- Concurrent read/write FIFO – FIFO with a variable number of stored data words and possible asynchronism between the read and the write operation.

The shift register is not usually referred to as a FIFO, although it is first-in first-out in nature. On sequently, this application report focuses exclusively on FIFOs that handle variable-length data. Two electronic systems always are connected to the input and output of a FIFO: one that writes and one that reads. If certain timing conditions must be maintained between the writing and the reading systems, we speak of exclusive read/write FIFOs because the two systems must be synchronized. But, if there are no timing restrictions in how the systems are driven, meaning that the writing system and the reading system can work out of synchronism, the FIFO is called concurrent read/write.

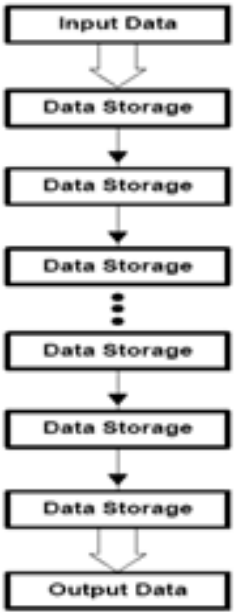

Fig.3.fifo data flow.

A. Synchronous FIFO$_S$

Synchronous FIFOs are controlled based on methods of control proven in processor systems. Every digital processor system works synchronized with a system-wide clock signal. This system timing continues to run even if no actions are being executed. Enable signals, also often called chip-select signals, start the synchronous execution of write and read operations in the various devices, such as memories and ports. The block diagram in Fig.4 shows all the signal lines of a synchronous FIFO. It requires a free-running clock from the writing system and another from the reading system. Writing is controlled by the WRITE ENABLE input synchronous with WRITE CLOCK. The FULL status line can be synchronized entirely with WRITE CLOCK by the free-running clock. In an analogous manner, data words are read out by a low level on the READ ENABLE input synchronous with READ CLOCK. Here, too, the free-running clock permits 100 percent synchronization of the EMPTY signal with READ CLOCK.

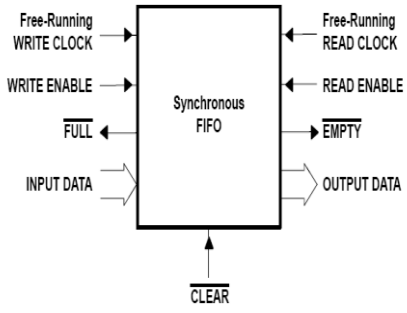

Fig.4.synchronous FIFO.

Thus, synchronous FIFOs are integrated easily into common processor architectures, offering complete synchronism of the FULL and EMPTY status signals with the particular free-running clock.

B. Design of Asynchronous FIFOs

An asynchronous FIFO refers to a FIFO design where data values are written to a FIFO buffer from one clock domain and the data value are read from the same FIFO buffer from another clock domain, where the two clock domains are asynchronous to each other. FIFOs are always used for data cache, storing differences of frequency or phase of asynchronous signals. And asynchronous FIFOs are often used to quickly and safely pass data from one clock domain to another asynchronous clock domain. In asynchronous clock circuit, periods and phases of each clock domain are completely independent so the probability of data loss is always not zero. This paper introduces a way of designing FIFO based on FPGAs with high write/read speed and high reliability. Generally, a FIFO consists of a RAM Array block, a Status block, a writer pointer (WR ptr) and a read point (RD ptr) and its structure is showing in fig.5. A RAM array with separate read and write ports is used to stored data. The writer pointer points to the location that will be written next, and the read pointer points to the location that will be read currently. A write operation increments the writer pointer and a read operation increments the read pointer. On reset, both pointers are reset to zero, the FIFO is empty.

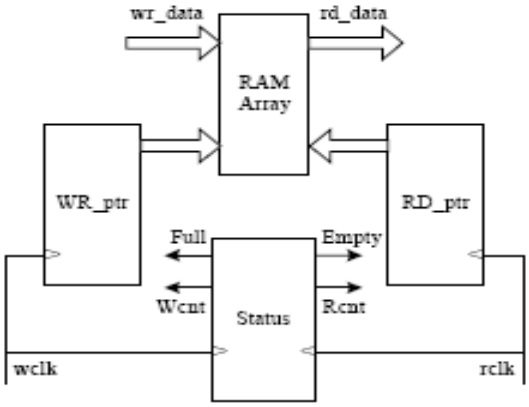

Fig.5.asynchronous FIFO structure diagram.

The writer pointer happens to be the next FIFO location to be written and the reader pointer is pointing to invalid data. The responsibility of the status block is to generate the _Empty_ and _Full_ signals to the FIFO. If the _Full_ is active then the FIFO cannot accommodate more data and if the _Empty_ is active then the FIFO cannot provide more data to readout. When writing data into the FIFO _wclk_ will be used as the clock domain and when reading data out of the FIFO _rclk_ will be used as the clock domain. These both clock domains are asynchronous. In designing of asynchronous FIFOs, two difficult problems cannot be ignored. One is how to judge FIFOs status according to the writer pointer and read pointer. The other is how to design circuit to synchronize asynchronous clock domains to avoid Metastability.

IV. DESIGNING OF URT WITH FIFO

In the multi-channel controller, there are different blocks including UART block, Status Detectors, asynchronous FIFOs block and Baud Rate Generator block. Each block has different function in the controller. The first part is UART circuit block and its structure is shown in fig.6. It consists of three parts Receive Circuit, Transmit Circuit and Control/Status Registers. The Transmit Circuit consists of a Transmit Buffer and a Shift Register. Transmit Buffer loads data being transmitted from local CPU. And Shift Register accepts data from the Transmit Buffer and send it to the TXD pin one by one bit. The Receive Circuit consists of a Receive Shift Register and a Receive Buffer. The Receive Shift Register receives data from RXD one by one bit. The Receive Buffer loads data from long-distance MCU and gets it ready for the local PC to read. The Control Register a special function register is used to control the UART and indicate status of it. According to each bit's value the UART will choose different kind of communication method and the UART knows what to do to receive or transmit data. FIFOs are used to store data received from the PC and get ready for sub MCUs. When writing data into FIFOs and reading data out of FIFOs we could set different clock domains according to the PC's and MCUs' Baud Rate. So it can be used to implement communications between MCUs at different Baud Rate. The controller also has a block of Baud Rate Generator to engender different Baud Rates to content requirements for different kind of systems. This block is constituted by timers (32/16 bits timers), frequency dividers and a Baud Rate setting register.

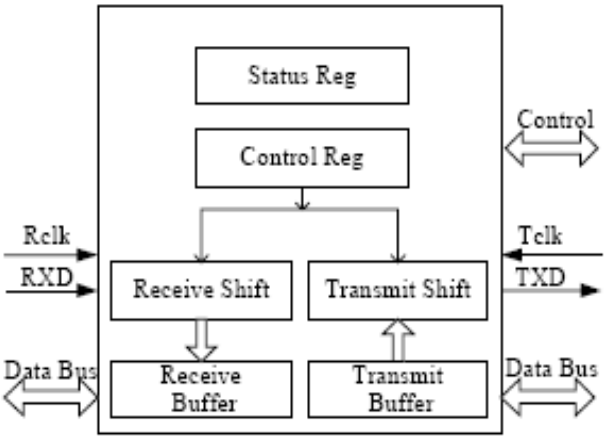

Fig.6.structure of UART block.

Using FIFO technique and the COM block as mentioned before, we design a multi channel controller. It can be used to implement communications between MCUs in a complex system. And it can also be used to complete communication between high speed device and low speed device. Structure of the controller is showing in fig.7. The controller is built within a FPGA – EP1C6Q240 which is based on SRAM technique produced by ALTERA. It is possible to design small scale memorizer like FIFOs. When designing FIFOs within FPGAs, you should consider the capacity of FIFO in practice and also consider the FPGAs' capacities.

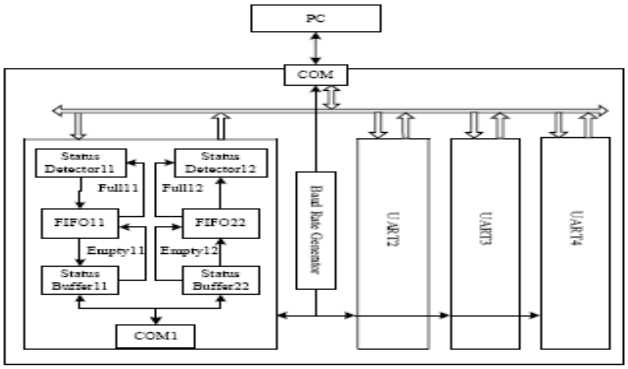

Fig.7.structure of the controller.

V. SIMULATION AND SYNTHESIS RESULT

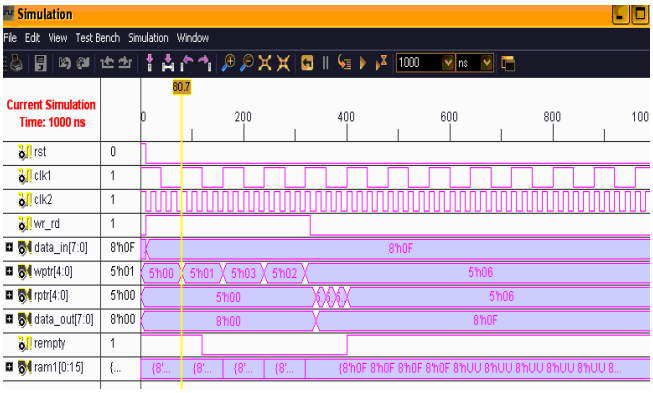

Fig.8. UART controller simulation wave form.

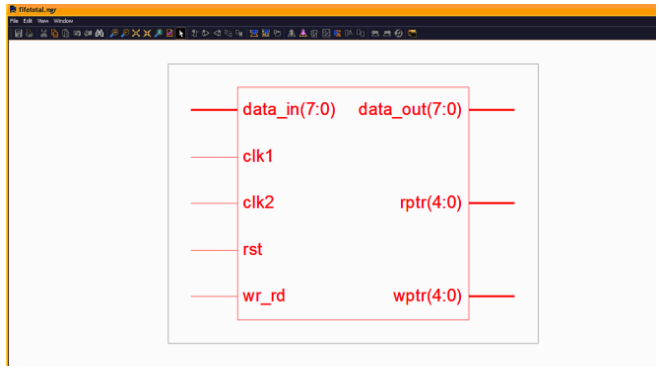

Fig9. RTL Schematic.

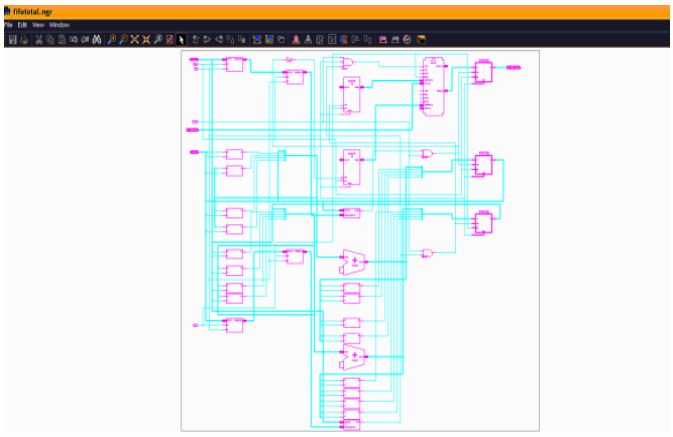

Fig .9. RTL Schematic UART.

VI. CONCLUSION

This project introduces a method to design a synchronous FIFO based on FPGA. And using synchronous FIFO technique implements a multi-channel UART controller within FPGA .The controller can be used to implement communications in complex system with different Baud Rates of sub-controllers. It can be used to reduce time delays between sub-controllers of a complex control system to improve the synchronization of each sub-controller. The controller is reconfigurable and scalable. In the serial communication between a computer and different devices, UART play a major role in the data transmission. Here the computer is working with a particular frequency and the devices connected to it work with different frequency. To convert these asynchronous domains to synchronous domains , we use a UART consisting of a FIFO. FIFO converts asynchronous to synchronous where as the UART converts the serial data to parallel and then parallel to serial. Here we conclude that using FIFO in the UART is very advantageous when compared with the normal UART.

VII. REFERENCES

[1]FIFO Architecture, Functions, and Applications by Peter Forstner TEXAS INSTRUMENTS.
[2]Simulation and Synthesis Techniques for synchronous FIFO Design by Clifford E. Cummings.
[3]Synchronous FIFO 5.0-XILINX.
[4]http://en.wikipedia.org/wiki/First-in,_first-out.
[5]http://www.xilinx.com/support/documentation/ipmeminterfacestorelement_fifo_synchfifo.htm.
[6]http://semiconductors.globalspec.com/Industrial-Directory/synchronous_fifo.