Journal of Information Systems and Communication
ISSN: 0976-8742 & E-ISSN: 0976-8750, Volume 3, Issue 1, 2012, pp.-41-44.
Available online at http://www.bioinfo.in/contents.php?id=45

FPGA IMPLEMENTATION & DESIGN OF MICRO UART WITH DIFFERENT BAUD RATES

MAGANDEEP KAUR[1]* AND RUCHI MITTAL[2]

[1]Department of Lingaya's Institute of Management & Technology, Faridabad, India.
[2]Department of Lingaya's University, Faridabad, India.
*Corresponding Author: Email- magandeepkaur@gmail.com

Received: January 12, 2012; Accepted: February 15, 2012

Abstract- In this paper we are designing Micro UART has capable of performing 8- bit Asynchronous data communication at different baud rates with the capability of error detection such as Parity error, Overrun error, Framing Error. The maximum clock frequency achieved is 165.222 MHz and minimum time period is 6.052ns. The code written in Verilog HDL is simulated and synthesized and successfully implemented on FPGA
Keywords- FPGA, FSM, Controller, UART, Baud Rate, RAM, Transmitter, Receiver, Baud Generator, Frame Error, Overrun Error

Citation: Magandeep Kaur and Ruchi Mittal (2012) FPGA Implementation & Design of Micro UART With Different Baud Rates. Journal of Information Systems and Communication. ISSN: 0976-8742 & E-ISSN: 0976-8750, Volume 3, Issue 1, pp.-41- 44.

Copyright: Copyright©2012 Magandeep Kaur and Ruchi Mittal. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Introduction

A UART is a Universal Asynchronous Receiver-Transmitter, which is used to communicate between two devices. Most computers and microcontrollers include one or more serial data ports utilize to communicate with other serial I/O devices, such as keyboards and serial printers. Serial ports are also used to communicate between two computers (figure 1) using a UART in each computer and a crossover cable, which connects the transmitter (TxD) of one UART to the receiver (RxD) of the other, and vice versa. A common ground (GND) wire connects both computers to a common negative voltage source.

In Figure 1, UART provides the means to send information using a minimum number of wires. The data is sent bit serially, without a clock signal. The main function of a UART is the conversion of parallel-to-serial when transmitting and serial to-parallel when receiving. The fact that a clock signal is not sent with the data complicates the design of a UART. The two systems (transmitter and receiver) contain separate and unsynchronized local clocks. A part of the function of UART[3]

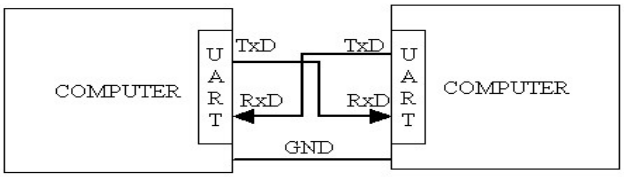

Fig. 1- Block Diagram of communication between two computers using UART is to sample the incoming serial data at the right time to precisely capture the binary stream. This is accomplished by utilising a fast clock to sample the binary stream multiple times for each data bit. Thus, when transmitting, the UART receives the data in parallel from the application, and sends it serially on the TxD pin, and when receiving, the UART receives the data serially on the RxD pin, and provides the parallel data to the application. The UART consists of two independent HDL modules. One module implements the transmitter, while the other module implements the receiver. The transmitter and receiver modules can be combined at the top level of the design, for any combination of transmitter and receiver channels required. Data can be written to the transmitter

and read out from the receiver, all through a single 8 bit bidirectional CPU interface. Address mapping for the transmitter and receiver channels can easily built into the interface at the top level of the design. Both modules share a common master clock called mclkx16. Within each module, mclkx16 is divided down to independent baud rate clocks.

Working Of UART

Figure 2, illustrates a basic UART data packet. While no data is being transmitted, a logic 1 must be placed in the XMIT line. A data packet is composed of 1 start bit, which is always a logic 0, followed by a programmable number of data bits (typically between 6 to 8), an optional parity bit, and a programmable number of stop bits (typically 1). The stop bit must always be logic 1.Most UART uses 8 bits for data, no parity and 1 stop bit. Thus, it takes 10 bits to transmit a byte of data [2].

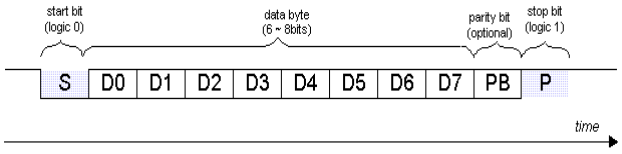

Fig. 2- Basic UART packet format: 1 start bit, 8 data bits, 1 parity bit, 1 stop bit.

In the UART protocol, the transmitter and the receiver do not share a clock signal. That is, a clock signal does not emanate from one UART transmitter to the other UART receiver. Due to this reason, the protocol is said to be asynchronous. Since no common clock is shared, a known data transfer rate (baud rate) must be agreed upon prior to data transmission. That is, the receiving UART needs to know the transmitting UART's baud rate (and conversely the transmitter needs to know the receivers baud rate, if any). In almost all cases the receiving and transmitting baud rates are the same. The transmitter shifts out the data starting with the LSB first.

Once the baud rate has been established (prior to initial communication), both the transmitter and the receivers internal clock is set to the same frequency (though not the same phase). The receiver "synchronizes" its internal clock to that of the transmitter's at the beginning of every data packet received. This allows the receiver to sample the data bit at the bit-cell center.

A key concept in UART design is that UART's internal clock runs at much faster rate than the baud rate. For example, the popular 16450 UART controller runs its internal clock at 16 times the baud rate. This allows the UART receiver to sample the incoming data with granularity of 1/16 the baud-rate period. This "oversampling" is critical since the receiver adds about 2 clock-ticks in the input data synchronizer uncertainty. The incoming data is not sampled directly by the receiver, but goes through a synchronizer which translates the clock domain from the transmitter's to that of the receiver. Additionally, the greater the granularity, the receiver has greater immunity with the baud rate error.

The receiver detects the start bit by detecting the transition from logic 1 to logic 0 (note that while the data line is idle, the logic level is high). In the case of 16450 UART, once the start-bit is detected, the next data bit's "center" can be assured to be 24 ticks minus 2 (worse case synchronizer uncertainty) later. From then on, every next data bit center is 16 clock ticks later.

Figure 3. illustrates this point. Once the start bit is detected, the subsequent data bits are assembled in a de-serializer. Error condition maybe generated if the parity/stop bits are incorrect or missing.

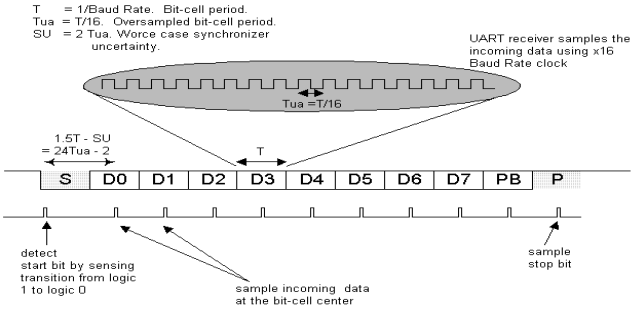

Fig. 3- Data sampling points by the UART receiver

Functional Description of Receiver

The receiver of micro-UART is composed of a control state-machine, de-serializer, and support logic. The main goal of the receiver is to detect the start-bit, then de-serialize the following bit-stream, detect the stop-bit, and make the data available to the host. Figure 3 illustrates the functional block diagram of the receiver

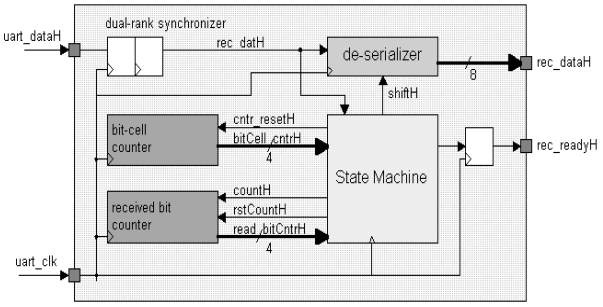

Fig. 4- Functional block-diagram of the UART receiver

Functional Description of Transmitter

In Figure 5, The transmitter of micro-UART is composed of bit cell counter, transmitted bit counter, a serializer and a state machine. Like the receiver counter-part, the design is minimalist and contains no error detecting logic. This also is left for user enhancements.

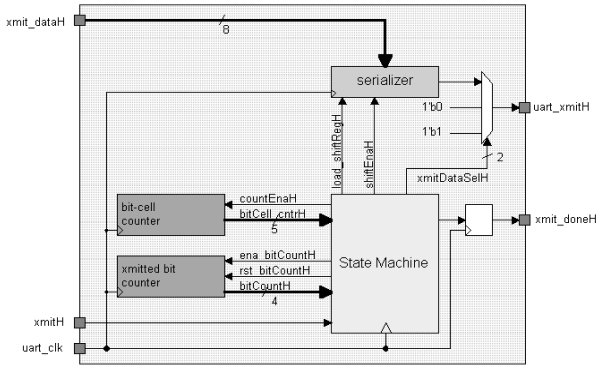

Fig. 5- Functional block diagram of UART transmitter

Functional Description of Baud Generator

Baud Rate may be defined as the number of bauds transmitted/ received per second. The baud generator is a programmable transmit and receive bit timing device. Generates a periodic pulse, which determines the baud rate of the UART transmission. This pulse is used to generate a sampling pulse to sample the received serial data. The transmitter uses to determine the bit width of the transmit data[1].

This baud generator generates two different timing clocks for both transmitter and receiver. Baud generator uses a clock divider circuitry to divide the system clock. It selects the dividing factor by which the clock is to be divided from a mux circuit through which we can select different baud rates for data communication. Receiver clock is further divided by factor 2 of transmitter clock. Due to that receiver clock is slower than the transmitter.

Baud rate factor is the ratio between the clock signal applied to the Tx-Rx inputs and desired baud rate[1].

Baud clock (Tx) = Baud rate * 32

Baud clock (Rx) = Baud rate * 16

Transmission and reception can be driven by their own baud rate generator. However be aware that to communicate correctly, the receiver must have a reception baud rate strictly equal to the transmission baud rate of the transmitter. As long as this condition is met, a wide range of baud rates.

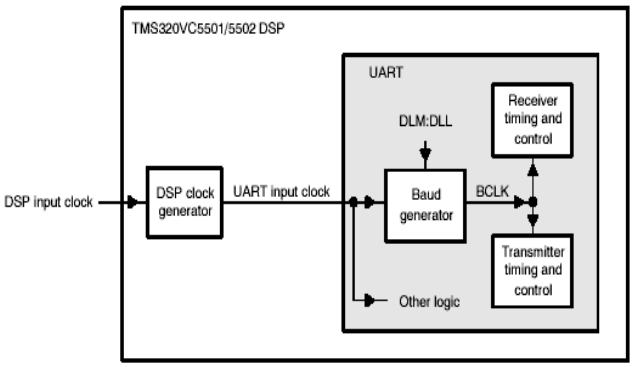

Fig. 6- Block diagram of baud rate generator

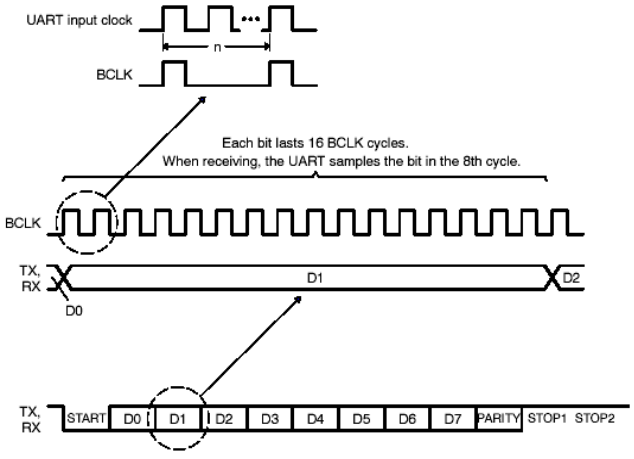

Fig. 7- Relationships Among Data Bit, BCLK, and UART Input Clock

Pin Description

SYMBOL	TYPE	DESCRIPTION
mclkx16	input	Master input clock used for internal baud rate generation.
reset	input	Master reset input signal.
parityerr	output	Indicates whether or not a parity error was detected during receive of a data frame. Encoding can be based on Even or Odd parity mode.
framingerr	output	Indicates if the serial data format sent to the rx input did not match the proper UART data format shown in Figure 2.
overrun	output	Indicates when new data are sent to the receiver, but data previously received is still held internally by the receiver, and has not yet been read out.
rxrdy	output	Indicates that new data has been received, and are ready to be read out.
txrdy	output	Indicates that new data can be written to the transmitter.
read	input	Active low read strobe signal, used for reading out data from the receiver.
write	input	Active low write strobe signal, used for writing data to the transmitter.
data[7:0]	inout	Bi-directional data bus. Data to be transmitted, or data that has been received, are transferred through this data bus.
tx	output	Transmitter serial output. tx will be held high during reset, or when no transmissions are taking place.
rx	input	Receiver serial input. rx should be held high (pulled-up), when no transmissions are taking place.

Fig. 8- pin description of micro-UART

Functional Simulation

Functional or RTL simulation is used to verify the syntax and functionality of the design. The following recommendations were used for simulating the design.

Typically with larger hierarchical HDL designs, one should perform separate simulations on each module before testing the entire design. This makes it easier to debug your code[5].

Once each module functions as expected, a test bench is created to verify that entire design functions as planned. The same test bench is used again for the final timing simulation to confirm that the design functions as expected under worst-case delay conditions.

Simulation Results

UART simulation

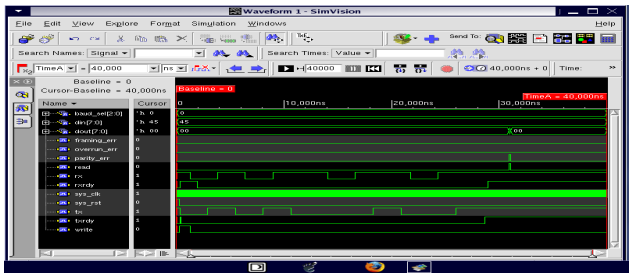

Fig. 9- Simulation result of UART

Baud Generator Simulation

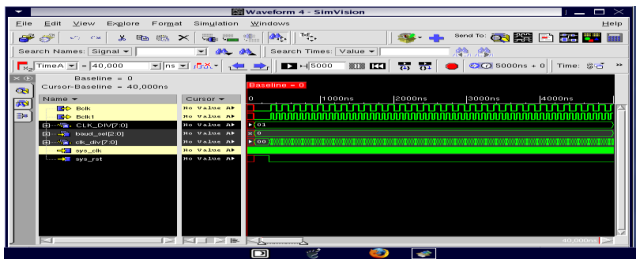

Fig.10- Simulation result of Baud Generator

Errors (Frame error, overrun error, parity error)

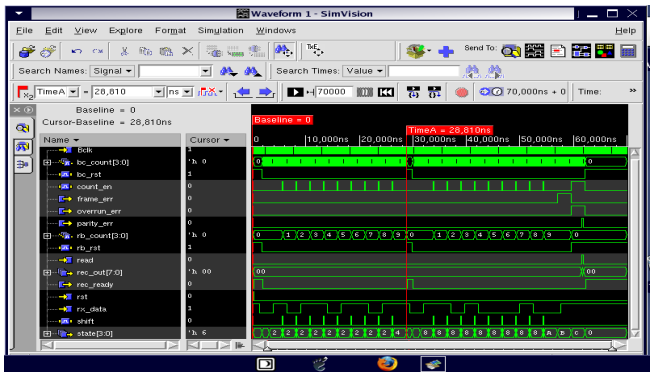

Fig. 11- Simulation result of Errors (Frame error, overrun error, parity error

Conclusion

The future work left is that we can extend this design with FIFO, Interrupt Generation & handling and capability of Retransmission on Error. This design can also be extended for Synchronous data communication mode . It is capable of performing 8 bit. Asynchronous data communication at different baud rates with the capability of error detection such as Parity error, Overrun error, Framing Error. The maximum clock frequency achieved is 165.222 MHz and minimum time period is 6.052 ns.

References

[1] Brunelli Claudio, Cinelli Federico, Rossi Davide, Nurmi Jari (2006) 2*nd Conference on Ph.D. Research in Micro R.K. Electronics and Electronics Proceedings*, 229-232.

[2] Luker Jarrod D., Prasad Vinod B. (2001) *RISC system design in an FPGA*, MWSCAS, v2, 532-536.

[3] Gupta R.K., Zorian Y. (1997) *IEEE Design & Test Of Computers*,14-25.

[4] Wael M ElMedany, Mahmoud R ElSabry (2009) *International conference on computer and communication Engg. Malaysia*.

[5] *www.support.xilinx.com*.