Volume 4, Issue 9, September 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Soft-Core - Independent Customizable-Platform of TX-RX of UART

Abhijit
Dept of Electronics& Communication
Ashoka College of Eng. & Tech
JNTU Hyderabad, India

Abstract— This paper we propose a technique for software- implementation of an UART (Universal-Asynchronous-Receive-Transmit) with the goal of getting a customizable UART-core which can be used as a module in implementing a bigger system irrespective of one's choice of implementation platform. Here we have written the core in VHDL (VHSIC hardware description language), implemented using XILINX ISE 10.1 Design suite and tested in SPARTAN-3AN FPGA evaluation kit by interfacing a test circuit with the PC using the RS232 cable. The simulation results as well as the test results are seen to be satisfactory. The area taken and the power consumed are also evaluated.

Keywords— UART; VHDL; Softcore

I. INTRODUCTION

It is a well established fact that the serial communication option wins over parallel in any form whenever we think of power consumption in the chip or in the board. Though we now have a lot of options for communicating serially between two processing modules through wire as well as wirelessly, the UART (Universal-Asynchronous-Receive-Transmit) protocol has been used frequently in the recent past for short distance off-board data transfer, software debugging etc. In case of on-board networking or on-chip networking, various synchronous serial communication protocols(e.g. I2C, SPI etc.) have dominated for the last few decades or so. But the ever growing dynamic power due
to increase of speed of operation and so also other power losses due to ever-decreasing size to achieve the high speed have already changed the research trend to asynchronous domain. Again the ever-increasing complexity of the processing systems to design, the modular approach has become a must. With those goals in our mind, we have designed an UART-core which can be used as a module in building bigger systems incorporating the UART protocol as their serial communication option. Various parameters of this module can be customized by the user as suitable for him/her. The configurable parameters of our module are: i) System clock frequency (default-50Meg) ii) Baud-rate (any one is generally used, default-9600) iii) No. of data bits (7/8, default-8) iv) No. of stop bits (1.0/1.5/2.0, default-1) v) Over-sampling rate (8/16/24/32 etc. default-16) vi) No. of buffers needed to cope up with the speed difference between the system using the UART and the rate at which data are coming (default-8).And the core is made available in full VHDL-which makes it platform-independent.

II. UART PROTOCAL

The UART protocol is a serial communication protocol that 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. The UART usually does not directly generate or receive the external signals used between different items of equipment. Separate interface devices are used to convert the logic level signals of the UART to and from the external signalling levels. External signals may be of many different forms. Examples of standards for voltage signalling are RS-232, RS-422 and RS-485 from the EIA.
Typically it's a 3-line (transmit, receive, ground) communication. Communication which enables it to be "full duplex" (both send and receive at the same time) or "half duplex" (devices take turns transmitting and receiving).

A. Character encoding

Each character is sent(shown in fig.1) as a logic low start bit, a configurable number of data bits (usually 7 or 8, sometimes 5), an optional parity bit, and one or more logic high stop bits. The start bit signals the receiver that a new character is coming. The next five to eight bits, depending on the code set employed, represent the character. Following the data bits may be a parity bit. The next one or two bits are always in the mark (logic high, i.e., „1) condition and called the stop bit(s). They signal the receiver that the character is completed. Since the start bit is logic low (0) and the stop bit is logic high (1) then there is always a clear demarcation between the previous character and the next one.

S	D1	D2	D3	D4	D5	D6	D7	PB	P

Figure 1: 8-bit character in one frame

B. Receiver

All operations of the UART hardware are controlled by a clock signal which runs at a multiple (say, 16) of the data rate - each data bit is as long as 16 clock pulses. The receiver tests the state of the incoming signal on each clock pulse, looking for the beginning of the start bit. If the apparent start bit lasts at least one-half of the bit time, it is valid and signals the start of a new character. If not, the spurious pulse is ignored. After waiting a further bit time, the state of the line is again sampled and the resulting level clocked into a shift register. After the required number of bit periods for the character length (5 to 8 bits, typically) have elapsed, the contents of the shift register is made available (in parallel fashion) to the receiving system. The UART will set a flag indicating new data is available, and may also generate a processor interrupt to request that the host processor to transfer the received data. In some common types of UART, a small first-in, first-out FIFO buffer memory is inserted between the receiver shift register and the host system interface. This allows the host processor more time to handle an interrupt from the UART and prevents loss of received data at high rates.

C. Transmitter

Transmission operation is simpler since it is under the control of the transmitting system. As soon as data is deposited in the shift register after completion of the previous character, the UART hardware generates a start bit, shifts the required number of data bits out to the line, generates and appends the parity bit (if used), and appends the stop bits. Since transmission of a single character may take a long time relative to CPU speeds, the UART will maintain a flag showing busy status so that the host system does not deposit a new character for transmission until the previous one has been completed; this may also be done with an interrupt. Since full-duplex operation requires characters to be sent and received at the same time, practical UARTs use two different shift registers for transmitted characters and received characters.

III. THE ART MODULE

The UART module [4] (fig. 3) that we have designed consists of five parts namely (i) “uart-rx”, which takes in the serial data (as a frame) coming through the „rx‟ line, retrieves the actual data and converts to parallel form (usually as a byte). (ii) “uart-tx”, which does the opposite function of the “uart-rx” module and transmits the frame through the „tx‟ line. (iii) “baudgen”, which generates a clock which occurs 16-times(the default over-sampling rate)in one bit-time period. (iv) “tx-fifo”, which stores temporarily the bytes (that usually comes from a faster processor) to send, as the sending process takes some time. (v) “rx-fifo”, which is the replica of the “tx-fifo” module used to store the received bytes temporarily such that the processor may read them at its own pace. The UART-top-module (fig.2) has (i) two data bus (wr-data and rd-data) for data in or out in parallel form from or to the processor it is used with. (ii) Two lines („tx‟ and „rx‟) through which data comes-in or goes-out serially bit-by-bit from or to the device it is communicating. (iii) Four lines (rd-uart, wr-uart, tr-full, rx-empty) are used for handshaking purpose with the processor it is used with. (iv) One `system-clk' signal that controls all the activities.

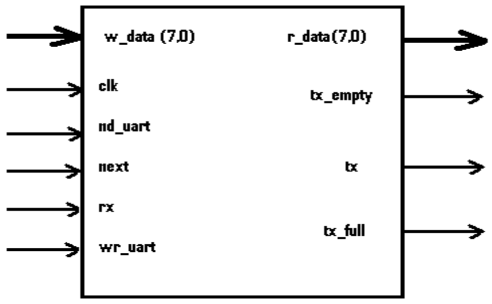


Fig. 2: UART top-module

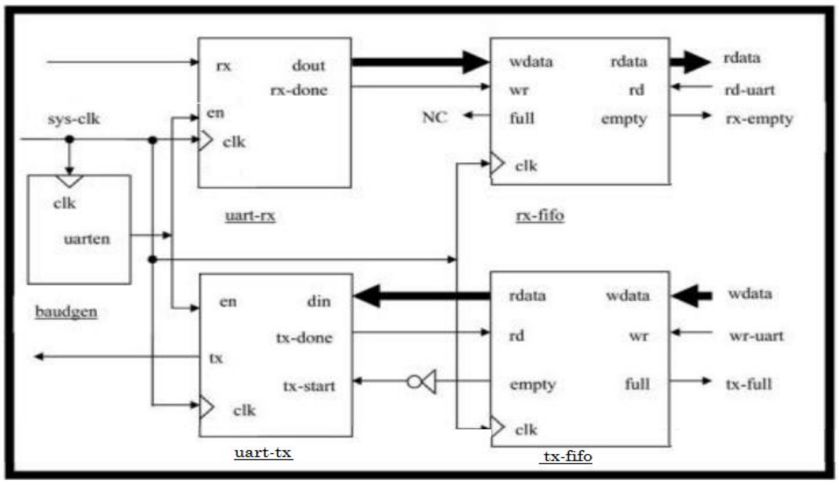


Fig. 3: UART core internal block diagram

IV. RESULTS

We have simulated each and every part of our module separately in „Xilinx-ISE simulator". The simulation results for the „baudgen", „fifo", „uart-rx", and „uart-tx" submodules are shown in the figs 4,5,6,7 respectively.

A. Testing

The complete core was also tested by connecting it with the PC by a RS232 cable after implementing a test circuit incorporating our core in Xilinx spartan-3an starter kit. The test circuit does have an incremented that takes one parellel data stored in the rx-fifo after getting received by the UART module, increments it by one and then gives it back to the tx-fifo which is then transmitted serially by the module the reading from the fifo, incrementation and writing to the fifo is done one character at a time when a switch(connected with the test circuit through adebouncer otherwise multiple eading and writing of NULL-character could produce erroneous result)is pressed .Here we first typed-in 4 characters(displayed locally in the heper terminal window) which are stored in the fifo, then pressed the assigned switch 4 times which eturns the stored characters incremented by one and are displayed again in the hyperterminal (as shown in fig:8)

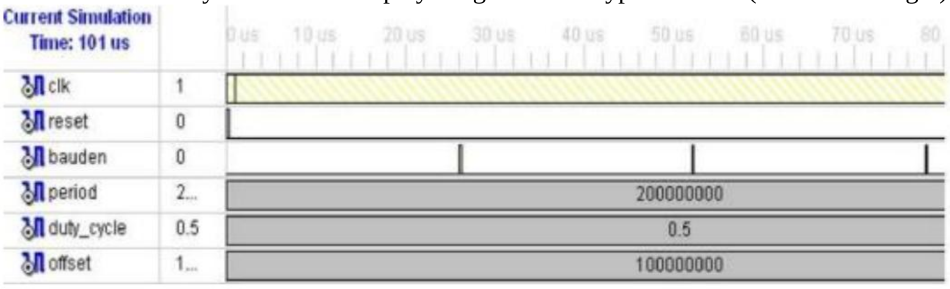


Figure 4: Simulation result for the submodule "baudgen"

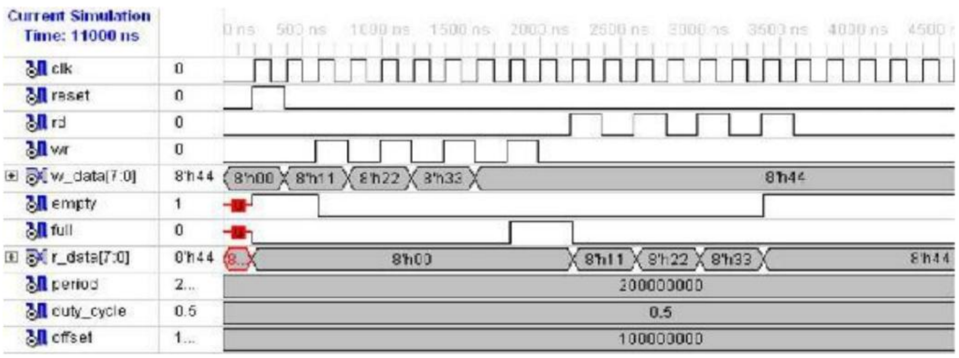


Figure 5: Simulation result for the submodule "fifo"

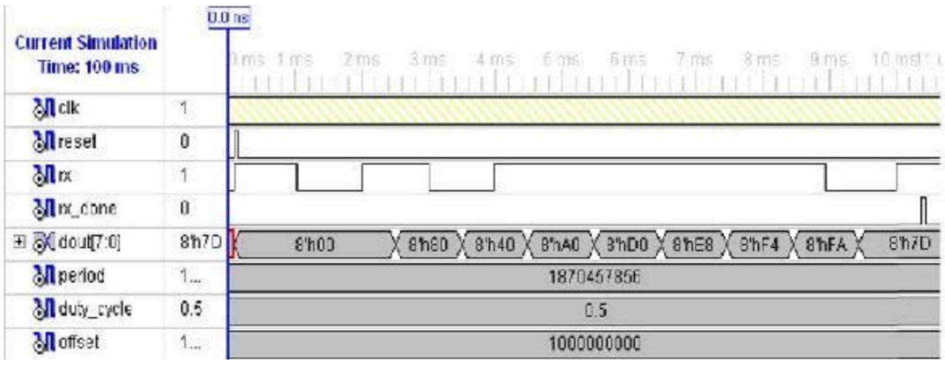


Figure 6: Simulation result for the submodule "uartrx"

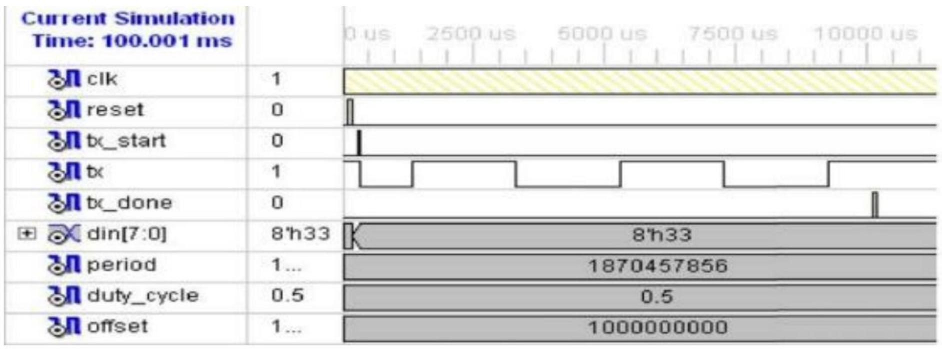
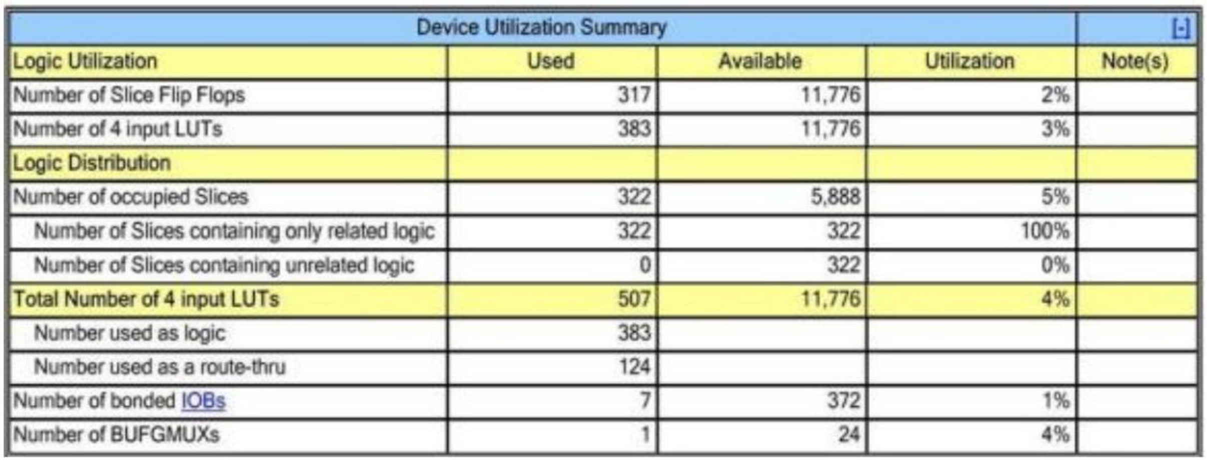


Figure 7: Simulation result for the submodule "uartrx"

B. Area and Power

Total percentage of the device (spartan 3-an) resources utilized to implement our module has been calculated by Xilinx-XST synthesizer which is found to be nominal and is shown in the figure (fig.9).



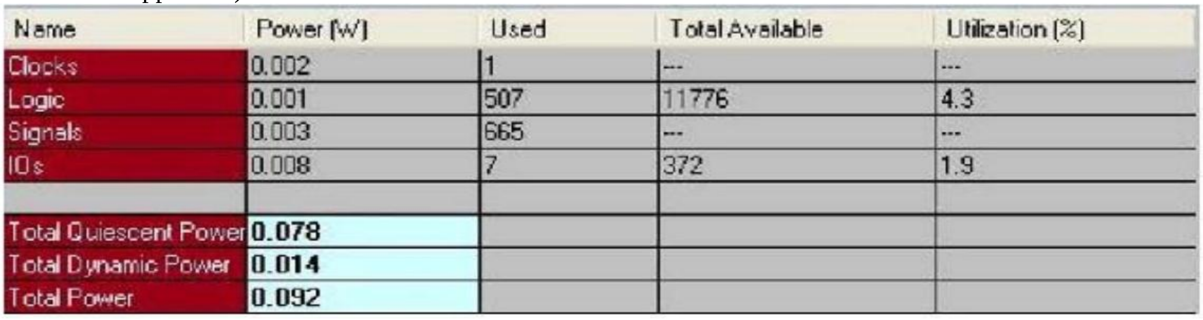
Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	317	11,776	2%	
Number of 4 input LUTs	383	11,776	3%	
Logic Distribution				
Number of occupied Slices	322	5,888	5%	
Number of Slices containing only related logic	322	322	100%	
Number of Slices containing unrelated logic	0	322	0%	
Total Number of 4 input LUTs	507	11,776	4%	
Number used as logic	383			
Number used as a route-thru	124			
Number of bonded IOBs	7	372	1%	
Number of BUFGMUXs	1	24	4%	

Figure 8. Device resource utilized

The power consumed by our module when operated from a 50 MHz system clock and with a 12.5-percent overall activity rate has been calculated with the help of `Xilinx XPower-Analyzer' which is small and is shown in the figure(fig.10).

V. CONCLUSION

We have designed our UART module in generic form which is operating fine with no under run error and can be customized to make it free from overrun error with the capability provided and so can be made available as IP core(Intellectual-Property-Core) by simply coating it with a proper wrapper(e.g. IBM-core connect SPLB-wrapper, AMBA APB-wrapper etc.).



Name	Power (W)	Used	Total Available	Utilization (%)
Clocks	0.002	1	---	---
Logic	0.001	507	11776	4.3
Signals	0.003	665	---	---
IOs	0.008	7	372	1.9
Total Quiescent Power	**0.078**			
Total Dynamic Power	**0.014**			
Total Power	**0.092**			

Figure 9. Power Consumed

REFERENCES

[1] James O. Hamblen, Tyson S. Hall, Michael D. Furman, Rapid prototyping of digital systems, 2nd ed., Springer Publication, 2001.

[2] Dauglas L. Perry, VHDL Programming by Example, 4th ed., The Tata-McGrawHill Pub, 2002.

[3] Volnei A. Pedroni, Circuit Design with VHDL, 3rd ed., The MIT Press, 2004..

[4] Xilinx Inc, xps uart lite v1.00, 3 ed. DS571, January 14, 2008.

Xilinx Inc, xpower analyzer user's guide, 3rd ed., January 14, 2008.

ARM Ltd, AMBA 3 APB Protocol Specification, 2nd ed.