DESIGN AND IMPLEMENTATION OF CUSTOMIZABLE UART SOFT-CORE IN FPGA TECHNOLOGY

[1]**K.PAVAN KUMAR**, PG Scholar in VLSI System design,
[2]**P.GIRIBABU**, M.Tech, Assoc. Professor, ECE Department,
[1]pavank576@gmail.com,
[2]giribabu.pantra@gmail.com.

Abstract:

UART (Universal Asynchronous Receiver Transmitter) is used for short-distance, low speed, low-cost data exchange between computer and peripheral. They provide a means to send data with a minimum of wires. The data is sent bit-serially, and no clock signal is sent along with it. The fact that a clock is not transmitted with the data complicates the design of a UART. The two systems (sender and receiver) have separate, unsynchronized, clock signals. The programmable logic devices can be used for such application by developing core for UART. By using hardware descriptive language UART simulation can be tested before it can be loaded on programmable device. In this project we present UART which includes three modules which are the baud rate generator, receiver and transmitter.

Keywords: Universal Asynchronous Receives and Transmits, Soft Core Implementation, Independent Platform, VHDL Respectively

I. Introduction

There is a huge establishment of the system takes place in the environment of the communication related to the aspect of the serial phenomena plays a crucial role in its representative analysis of the transmission of the data in a simultaneous fashion respectively. Here apart fromthe strategy of the consumption of the power playsa crucial role in its representative analysis point of view where there is an accurate consumption of the power that is programming based on the on board strategy plays an efficient role and the responsibility in its representative analysis point of view respectively.

Here there is a communication of the data takes place in the serial manner respectively. Asynchronous serial communication has advantages of high reliability, less transmission line and long transmission distance, therefore is widely used to exchange data between a computer and external devices. Asynchronous serial communication is implemented by UART. It provides full-duplex communication in serial link; this has been widely used in the data communications. UART includes a transmitter and a receiver. Transmitter controls transmission by taking a data word in parallel format and directing the UART to transmit it in a serially. Likewise, the Receiver must detect transmission, receive the data in serially, and store the data word in a parallel format. The conversion of serial to parallel data is handled by UART. Serial communication reduces the distortion of a signal; therefore data transfer is possible between two systems separated by great distance. The UART serial module is divided into three sub-modules: The baud rate generator, receiver module and transmitter module. The baud rate generator is used to produce a local clock signal. In data transmission through the UART, once the baud-rate has been established, both the transmitter and the receiver's internal clock are set to the same frequency. TXD is the transmit side, i.e. the output of the UART RXD is the receiver, i.e. the input of the UART. The UART receiver module is used to receive the serial signals at RXD and convert them into parallel data. The UART transmit module converts the data bytes into serial bits according to the frame format
and transmits those bits through TXD.UART's basic features are: There are two states in the signal line, using logic high and logic low to distinguish respectively. UART frame format consist of a start bit, data bit, parity bit and stop bit. After the Start Bit the data bits are sent, with the Least Significant Bit (LSB) sent first. The start bit is always low and the stop bit is always high. When the complete data word

has been sent, it adds a parity bit this parity bit may be used by the receiver to perform error checking. Then at least one Stop Bit is sent by the transmitter. Because asynchronous data are “self-synchronizing”, if there is no data to transmit, the transmission line will be idle.

Soft-core processors are becoming increasingly common in modern technology. A soft-core processor is a programmable processor that can be synthesized to a circuit, typically integrated into a larger system existing on an application-specific integrated circuit (ASIC) or field-programmable gate array (FPGA). Popular commercially available soft-cores include ARM [1], Tensillica [13], Microblaze [14], and Nios [2]. Several trends of modern technology catalyze soft-core usage, including stabler synthesis tools, higher-capacity ASICs and FPGAs, new commercial toolsets for application-specific instruction-set processors, and increasing demands for high-performance and low-power embedded processing.

Whereas traditional pre-fabricated processors must be optimized for good performance across an entire domain of applications, soft-cores can instead be customized to the particular applications they execute. For example, a particular application may perform best on a processor having a large cache and a floating-point unit, while another application may instead require a hardware multiplier and have no need for a cache. Nearly all soft-core providers therefore include parameters that may be customized by a soft-core user. Common parameters include instantiatable coprocessor units such as hardware floating-point, multiplier, divider, or barrel shifter units; cache architecture settings such as the use of one or two levels of cache, separate versus unified cache, cache size, line size, associativity, and write back policy; and processor microarchitecture features such as pipeline depth, bypass logic, or register-file size. The majority of the soft-core processors mentioned above contain more thanten customizable parameters, and the trend in newer versions is towards increasing the number of parameters. In this work, we only consider customizations that consist of tuning parameter values; such customizations are in a different category than are application-specific instruction-set extensions [13], which involvethe design of new hardware and/or the introduction of new instructions. Extensions could be integrated into a tuning approach by pre-determining good possible extensions, and treating each possibility as a particular value of an “extension” parameter.

The process of customizing a soft-core by tuning parameter values can yield improved performance, lower-energy, and/or smaller size. A newly evolving size-constrained scenario involves dozens of FPGA soft-cores competing for limited hardware-resources [9], for which tuning will be important to make best use of that limited hardware.

Soft-core providers offer little or no automated support to assist users in customizing a soft-core’s parameters to a particular application, other thanproviding simulation tools. As a consequence, users must manually guess and simulate (or implement) a set of candidate configurations in order to find the best parameters for a particular application. Each such simulation may require tens of minutes, limiting the number of candidate configurations that can be examined. Some recent research addresses automated soft-core configuration using custom heuristics developed for a particular parameters.

II. Literature Survey

Kumar [6][7] showed the benefit of multi-core generalpurpose processor chips aving heterogeneous rather than homogenous cores. They considered superscalar processor parameters related to cache, instantiations of floating-point, multiply, and arithmetic-logic units, and sizes of the register file, translation lookaside buffer, and load/store queue, yielding 480 possible single-core configurations. Via exhaustive search, they showed that an optimally configured four-core system has up to 40% better performance for a given workload, versus the best homogeneous four-core system for that workload.

Givargis [5] developed a tuning approach for parameterized system-on-a-chip platforms, considering parameters related to cache, bus, processor voltage, and a few parameters in peripherals. They used a user’s denotation of independent subsets of parameters to extensively prune the configuration space before searching dependent parameters exhaustively or using heuristics. They showed roughly 5x tradeoffs between power and performance for different applications.

Sekar [11] discussed trends toward highly parameterized platforms, including parameterizedprocessor cores, peripherals, caches, etc., and then described a technique for dynamically tuning the voltage and frequency of the processor.

Yiannacouras [15][16] developed a framework for generating and customizing a soft-core for FPGAs, with parameters including hardware versus software multiplication, different shifter implementations, and pipeline depth. They showed 30% improvements obtained by optimallytuning soft-core parameters for a specific application, using exhaustive search to carry out the tuning. Their work motivates the need to develop efficient automated customization heuristics.

We previously [12] developed heuristics for soft-core parameter tuning. The approachassumed that synthesis and execution (or simulation) of soft-core configurations, rather than pure estimation approaches, is essential for accurate evaluation of FPGA soft-cores, due to the tremendous variation of soft-core performance for different applications and across the hundreds of different FPGA devices by an FPGA vendor. Because synthesis/execution runs are costly, requiring tens of minutes or more, we developed several tuning heuristics that utilized only about a dozen synthesis/execution runs, thus executing in 1-2 hours. We considered a Xilinx Microblaze soft-core processor whose parameters each involved the option of instantiating a hardware component, including a hardware multiplier, barrel shifter, divider, floating point unit, or a fixed-sized cache. That work showed 2x application speedups of a customized core versus a base core having no optional components instantiated.

Our previous best heuristic (as well as our other heuristics) used what we will call a “single factor” analysis, a common analysis approach. The heuristic was guided by the speedup versus the base core when instantiating exactly one (single factor) of the core’s optional hardware components. The heuristic then sorted each component by the ratio of its speedup over size, yielding an “impact-ordered tree” of parameters, which the heuristic then descended (encountering two choices per tree level) to find a solution. While a single-factor analysis is effective for on/off-type parameters, such an approach lacks an obvious extension for parameters that have two non-zero values (which value would be the base value?) or that have three or more values. Furthermore, a single-factor analysis may be inaccurate if parameters are interdependent. For example, neither of two components may individually yield speedup, but the two together may; conversely, two components may individually each yield speedup, but instantiating both may yield little benefit beyond instantiating just one, due to functionality overlap. In contrast, the approach we introduce here performs a multi-factor analysis, supporting multi-valued parameters and considering interdependent parameters, as will be described.

III. UART Protocol

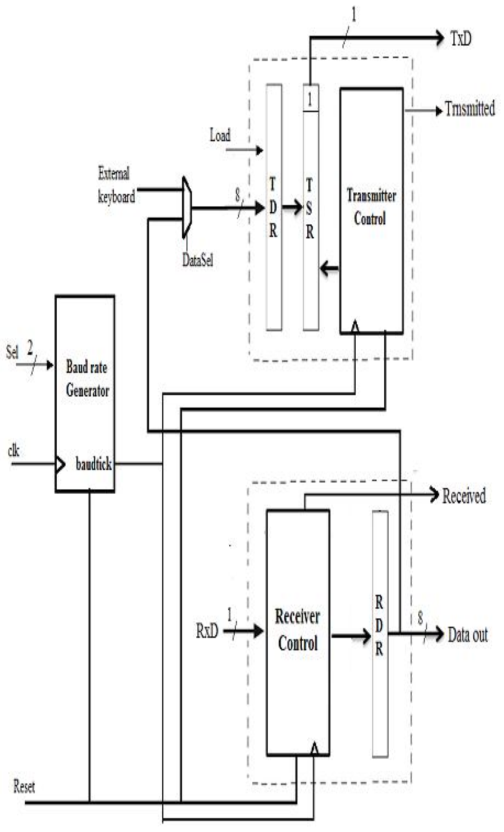


Fig.1 Block Diagram of UART

UART Components

Volume 5, Issue 1 OCT 2015

A UART is composed of three main component receiver, transmitter and baud rate generator. Inputs:
Sel: Selects baud rate (4 options are available).
RxD: Data coming from PC serial port.
DataSel: Selects whether data to be transmitted is coming from external keyboard attached to FPGA or data coming from pc serial port.
Reset: Resets all components.
Outputs:
TxD: Data is sent to pc serial port and displayed.
Data Out: Data received is shown on LED's on FPGA. In case parity check fails, then output is always shown as "E".

Baud Rate Generator

The baud rate generator generates a sampling signal whose frequency is exactly 16 times UART's baud rate. If the baud rate is X, the sampling rate has to be 16*X ticks per second. The system clock rate is 50 MHz the baud generator needs a mod-m (50*106/16*X) counter, in which 1 clock-cycle-tick asserted once every m clock cycle [6].The baud generator has 2-select bit to decide baud rate, since we are using two bits, we have the choice of four baud rates.

Table 1. Baud Selection

Bit Select	BAUD Rate
00	9600
01	4800
10	38400
11	19200

Transmitter

Transmitter takes parallel data and sends it serially on the TxD pin. The transmitter consists of TDR (Transmit Data Register), TSR (Transmit Shift Register) and controller. As load signal goes high transmitter transfers data from TDR to TSR and outputs start bit "0" to the TxD pin then shifts TSR right eight times to transmit 8 bits. When eight data bits transmitted ,transmitter sends parity bit and finally outputs stop bit"1" to the TxD pin and signal "transmitted" goes high.

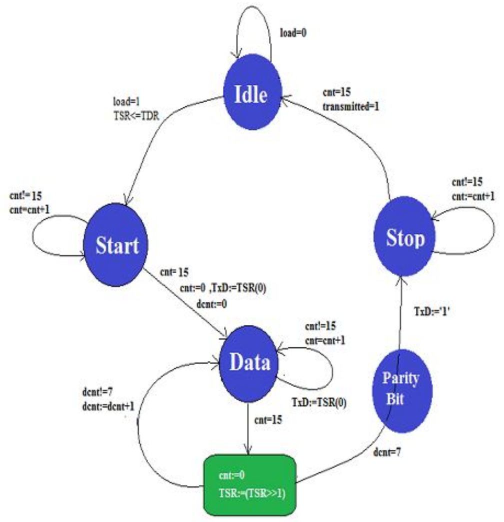


Fig.2 State Diagram of Transmitter

Receiver

Receiver takes data serially in RxD pin, and provides the parallel to the Data out pin. UART receiver consists of RDR (Received Data Register) and controller. When the UART detects start bit receiver reads and shifts 8 data bits serially into a temporary register. When 8 data bits has been received and parity check passes then after stop bit has been received controller transfers data from temporary register to RDR and received signal goes high. If parity check fails then, output of receiver is always shown as "E".

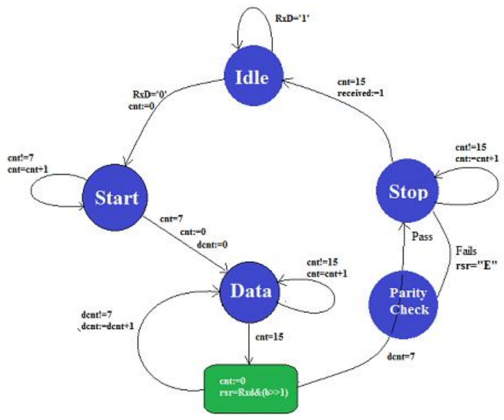


Fig.3 State Diagram of Receiver

IV. Simulation and Synthesis Results

We have simulated each and every part of our moduleseparately in modelsim6.4b. The simulation resultsfor the 'baudgen', 'fifo', 'uart-rx', and 'uart-tx' sub modules are shown in the figures below respectively. We have synthesis top module in Xilinx10.1

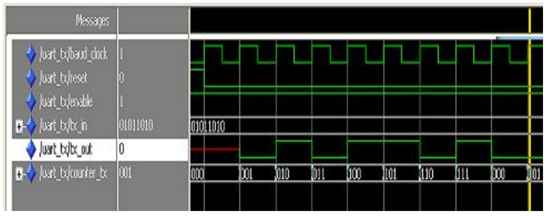


Fig.4 Uart Transmitter

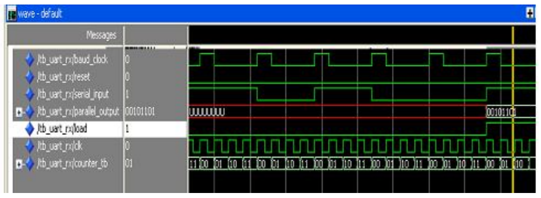


Fig.5 UART Receiver

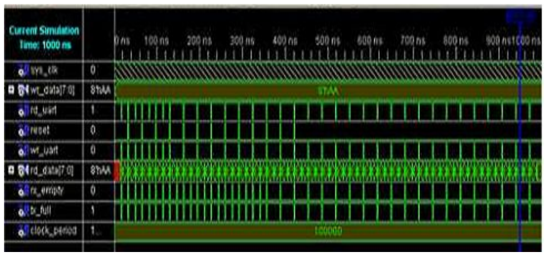


Fig.6 Topmodule

Synthesis Results

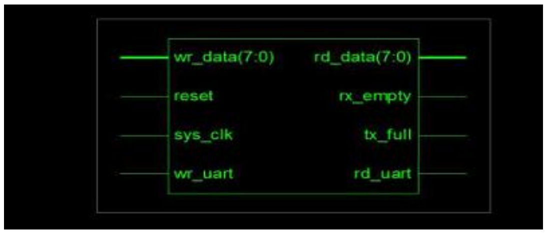


Fig 7.RTL Schematic

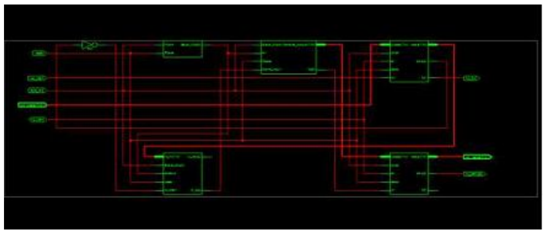
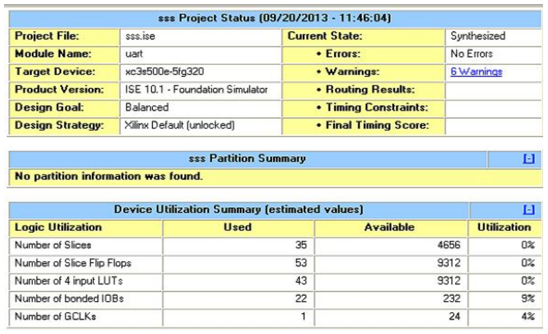


Fig 8.Technology Schematic

Synthesis Report



sss Project Status (09/20/2013 - 11:46:04)			
Project File:	sss.ise	Current State:	Synthesized
Module Name:	uart	• Errors:	No Errors
Target Device:	xc3s500e-5fg320	• Warnings:	6 Warnings
Product Version:	ISE 10.1 - Foundation Simulator	• Routing Results:	
Design Goal:	Balanced	• Timing Constraints:	
Design Strategy:	Xilinx Default (unlocked)	• Final Timing Score:	

sss Partition Summary [-]

No partition information was found.

Device Utilization Summary (estimated values) [-]

Logic Utilization	Used	Available	Utilization
Number of Slices	35	4656	0%
Number of Slice Flip Flops	53	9312	0%
Number of 4 input LUTs	43	9312	0%
Number of bonded IOBs	22	232	9%
Number of GCLKs	1	24	4%

Fig 9.Design Summary

CONCLUSION

We have concluded that the authors had used FIFO and Shift register separately for storing the data and in some paper they have used baud rate generator with single frequency. In this we had used FIFO as well as Shift register and also an automatic baud rate generator which will change its baud rate according to the change in frequency.This design uses VHDL language to achieve the modules of the UART. 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 IPcore (Intellectual-Property-Core) by simply coating it witha proper wrapper.

References

[1] Arm http://www.arm.com.
[2] Altera Corp. Nios II Processors. http://www.altera.com/products/ rocessors/nios2/ni2-index.html, 2005.
[3] DOE Pro XL http://sigmazone.com/doepro_faqs.htm.
[4] EEMBC. http://www.eembc.org/, 2005.
[5] Givargis, T., F. Vahid. Platune: A Tuning Framework for Systemon-a-Chip Platforms. IEEE Transactions on Computer Aided Design, Vol. 21, No. 11, Nov. 2002, pp. 1317-1327.
[6] Kumar, R., D. Tullsen, N. Jouppi. Core Architecture Optimization for Heterogeneous Chip Multiprocessors. International Conference on Parallel Architectures and Compilation Techniques, PACT, Seattle, April 2006.
[7] Kumar. R., D. Tullsen, P. Ranganathan, N. Jouppi, K. Farkas. Single-ISA Heterogeneous Multi-core Architectures for Multithreaded Workload Performance. In31st International Symposium on Computer Architecture, ISCA-31, June 2004.
[8] McLean, R., V. Anderson, Applied Factorial and Fractional Designs. Marcel Dekker, Inc. New York, New York, 1984.
[9] Moyer, B., Tune Multicore Hardware for Software. Xcell Journal, Issue 58, 2006, pp 55-57.
[10] Petersen, R., Design and Analysis of Experiments. Mercel Dekker Inc. New York, New York, 1985
[11] Sekar, K., Kanishka Lahiri, Sujit Dey. Dynamic Platform
Management for Configurable Platform-Based System-on-Chips.
Intl. Conf. on Computer-Aided Design (ICCAD), 2003.
[12] Sheldon, D., R. Kumar, R. Lysecky, F. Vahid, D. Tullsen.
Application-Specific Customization of Paramaterized FPGA SoftCore Processors. Intl. Conf. on Computer-Aided Design (ICCAD),
2006.
[13] Tensilica, Inc. The XPRES Compiler: Triple-Threat Solution to
Code Performance Challenges. http://www.tensilica.com/
pdf/XPRES-Triple-Threat_Solution.pdf, 2005.
[14] Xilinx, Inc. MicroBlaze Soft Processor Core.
http://www.xilinx.com/, 2005.
[15] Yiannacouras, P., J. G. Steffan,J. Rose. Application-Specific
Customization of Soft Processor Microarchitecture. FPGA 2006.
[16] Yiannacouras, P., J. Rose, J. G. Steffan. The Microarchitecture of
FPGA-based soft processors International Conference on Compilers,
Architecture, and Synthesis For Embedded Systems (CASES), 2005.