International Journal of Advanced and Innovative Research (2278-844)/

Volume 7 Issue 11

A SURVEY ON: MODULAR APPROACH FOR CUSTOMIZABLE UART

[1]Pratibha Vishwakarma, [2]Mr.Sanjeev Shrivastava

[1]*Research Scholar, RKDF College of Engineering Bhopal (MP), India*

[2]*Guide, RKDF College of Engineering Bhopal (MP), India*

Abstract—This paper 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. This paper is implementing the design through Verilog HDL using Xilinx 14.2 design suite and it is tested on Spartan-6 FPGA after interfacing the circuit under test using PC with the help of RS-232 cable. The simulation results and the test results are supporting our proposal.

Keywords—UART; COTS; CU.

I. INTRODUCTION

A UART (Universal Asynchronous Receiver and Transmitter) is a standard communication component that is provided by most of the available microcontrollers. In order to supply a low-cost solution, two novel field-bus protocols, TTP/A [1] and LIN [2], specify a common UART as communication interface to the network. Both protocols are central master UART protocols for low-cost single-chip smart sensor and actuator nodes, which enable a temporal predictable communication [3]. Case studies [4, 5] have shown that an implementation with COTS (commercial-off the-shelf) hardware is feasible. However, an in-deep analysis of the behavior of standard hardware UARTs has shown that they are hardly suitable for real-time communication. Moreover, both LIN and TTP/A specify a synchronization message that enables a slave node with an imprecise low-cost on-chip oscillator to synchronize with a running network. As a consequence, implementations for LIN and TTP/A exist that prefer a software-implemented UART to a COTS hardware UART component, leading to increased software complexity for the implementation of the protocol.

- **UART PROTOCOL**

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).

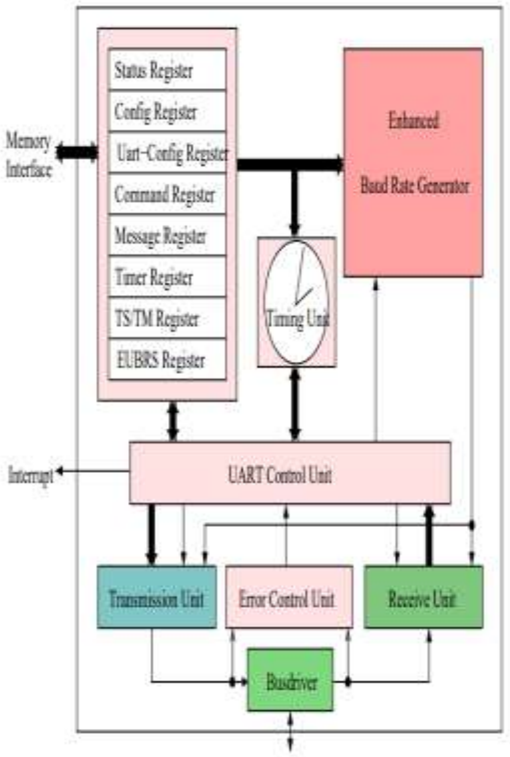
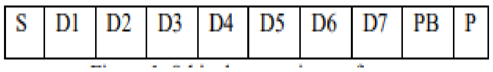


Figure 1: Block Diagram Of The UART Module.

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 2: 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.

II. LITERATURE SURVEY

Kumar [6] [7] showed the benefit of multi-core general purpose 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 [8] 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 [9] discussed trends toward highly parameterized platforms, including parameterized processor cores, peripherals, caches, etc., and then described a technique for dynamically tuning the voltage and frequency of the processor.

Yiannacouras [10] [11] 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 optimally tuning 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 approach assumed 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 multifactor analysis, supporting multi-valued parameters and considering interdependent parameters, as will be described.

III. 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.

REFERENCES

[1] H. Kopetz et al. Specification of the TTP/A protocol. Technical report, Technische Universitat Wien, Institut f ¨ ur Technische Informatik, Vienna, Austria, March 2000.

[2] Audi AG, BMW AG, DaimlerChrysler AG, Motorola Inc. Volcano Communication Technologies AB, Volkswagen AG, and Volvo Car Corporation. LIN specification and LIN press announcement. SAE World Congress Detroit, http://www.lin-subbus.org, 1999.

[3] H. Kopetz, W. Elmenreich, and C. Mack. A comparison of LIN and TTP/A. In Proceedings of the 3rd IEEE International Workshop on Factory Communication Systems, pages 99–107, Porto, Portugal, September 2000.

[4] P. Peti and L. Schneider. Implementation of the TTP/A slave protocol on the Atmel ATmega103 MCU. Technical Report 28/2000, Technische Universitat Wien, Institut f ¨ ur Technische Informatik, ¨ Vienna, Austria, August 2000.

[5] Atmel Corporation. LIN Protocol Implementation on the T89C51CC01/02, March 2003. Application note available at http://www.atmel.com.

[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 Multicore Architectures for Multithreaded Workload Performance. In31st International Symposium on Computer Architecture, ISCA-31, June 2004.

[8] 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.

[9] Sekar, K., Kanishka Lahiri, Sujit Dey. Dynamic Platform Management for Configurable Platform-Based System-on-Chips. Intl. Conf. on Computer-Aided Design (ICCAD), 2003.

[10] Yiannacouras, P., J. G. Steffan,J. Rose. Application-Specific Customization of Soft Processor Microarchitecture. FPGA 2006.

[11] 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.

[12] Sheldon, D., R. Kumar, R. Lysecky, F. Vahid, D. Tullsen. Application-Specific Customization of Paramaterized FPGA SoftCore Processors. Intl. Conf. on ComputerAided Design (ICCAD), 2006.