Uart With FIFO For Serial Communication

R.Rajgowri[1], Akash Senthilkumar[2], Akilan V[3], Arun M[4]
[1]Assistant Professor, Dept of ECE
[2, 3, 4]Dept of ECE
[1, 2, 3, 4] Muthayammal Engineering College, Namakkal, Tamilnadu, India

Abstract- *Universal Asynchronous Receiver Transmitter (UART) has continued to be among the basic serial communication protocols within the embedded systems and digital design. Although used extensively, traditional UART designs have the disadvantage of losing data in a high-continuous rate transmission of both communicating devices because of rate inconsistencies. The current paper is introducing a very solid UART architecture with built-in First-In First-Out (FIFO) buffering feature that has been designed and tested with Verilog HDL using standard VLSI front-end design. The suggested design uses a parameterizable FIFO buffer between transmitter and receiver modules to effectively remove the state of data overrun and underrun and a smooth asynchronous behavior between modules operating at various clock rates. It consists of the entire system, which consists of baud rate generation, serial-to-parallel conversion system, and all-performing frame-handling with programmable data-length, parity setting and stop bits. Effective frame generation, data integrity and FIFO management are confirmed through RTL simulation outcome with ModelSim with several test conditions. The modular architecture makes it simple to fit into System-on-Chip design and FPGA implementation, and therefore has been found to be reliable in communicating even in variably-speed data scenarios. Results of synthesis-based optimization have shown the optimal utilization of areas and timing closure requirements in the face of conventional constraints. In this work, there is an addition of a reusable intellectual property core that is applicable in embedded communication systems that need high throughput and low latency attributes.*

Keywords: UART, FIFO, Serial Communication, Verilog HDL, RTL Design, VLSI Implementation, FPGA.

I. INTRODUCTION

The history of developing digital communication systems has predetermined a growing need in a proper and effective method of data exchange between the processing elements and peripheral devices. Serial communication protocols have become significant because of fewer pin counts requirement, less sensitivity to noise and capability to transmit over a long distance, relative to parallel interfaces. The Universal Asynchronous Receiver Transmitter is one of many varieties of serial protocols that have become de facto in embedded systems, industry automation, Internet of Things devices, and telecommunication devices.

UART works on the principle of asynchronous with transmitter and receiver operating independently without having a common clock signal. Rather, the two devices communicate by a pre-established baud rate which stipulates the time when the bits are to be transmitted. This asynchronous behaviour eases the system design since using special clock distribution networks between communicating systems are unnecessary. But this simplicity also poses basic constraints in dealing with continuous data streams or in system situations with large variations in processing loads [2].

The conventional UART implementation handles data on a byte per byte basis with very little buffering. Figure 2 below shows that when the receiving device is unable to process the incoming data at a higher priority or lower processing rate than the incoming data, data disappear. On the same note, when using buffered data transmission the idle states of the transmitter caused a decrease in throughput. Such restrictions are especially acute in embedded systems associated with multi-tasking where communication interfaces have to share processor resources with other system functions [3].

Such fundamental constraints are overcome by the use of First-In First-Out memory buffers which offer a temporary storage which decouples the rate of data processing and the communication rate. FIFO buffers preserve data sequence and permit burst transfers as well as adjusting timing discrepancies between systems used asynchronously. FIFO buffers used in UART architectures can operate to allow the buffering of incoming data compared with the outgoing data in spite of localized processor unavailability, which in effect prevents overrun and underrun of buffering servers [4]Viewed in the context of VLSI design, UART FIFO implementation An UART with FIFO design can be implemented at many abstraction levels, starting with the behavioralmodeling, an implementation at the register-transfer level design, functional verification, and synthesis. Hardware description languages facilitate means of capturing the behavior of digital circuits at the right level of abstraction and such that can be synthesized

to be implemented in practice. The known benefits of verilog HDL in this case can be explained by its popularity in the industry, ability to embrace various style of models and performance of a simulator [5].

This work is driven by the practical observation of the loss of data in the traditional UART-based systems that are implemented in real-time applications. Serial interfaces are needed in sensor networks constantly getting environmental measurements, industrial control systems probing at several points, and communication gateways combining data of all kinds, serial interfaces are demanded to be reliable maintaining data integrity when load conditions are changing. These requirements are directly met by the proposed design through architectural improvements which keep retro-compatibility with conventional UART protocols.

This paper gives a full design procedure of UART using FIFO buffering features, including infrastructural choices, modules implementer, test plans, and synthesis. The rest of this paper is structured as follows: Section II is a literature review and previous work in UART design and approximate computing. Section III looks at the architectures of the existing systems and limitations. Section IV outlines the suggested approach such as module specification and flow. Section V is a description of the implementation platform of hardware and software. The Verilog implementation is found in section VI. Section VII regulates the results of simulation and verification. The last section, IX, is a conclusion with findings and future directions.

II. LITERATURE REVIEW

Digital communication interface design has seen a high level of research work aimed at increasing the reliability, lessening power usage, and increasing throughput. A number of solutions have risen that deal with various issues of serial communication and design of digital systems.

Examined self-tunable approximate computing architecture which can dynamically tune circuit precision, depending on workload properties [6]. They suggested two control mechanisms in their work: heartbeat-based reset circuits and lightweight internal checkers, which sense workload changes. Although their adaptive approach is mostly concerned with approximate computing as an hardware accelerator, it provides some understanding to the UART systems that work with different data rates. Multi-rate communication environments have a potential of gaining the concept of runtime adaptation in their efforts to synchronize the baud rate.

Investigated support-reducing decomposition methods that were optimized in FPGA mapping [7]. They dealt in their work with the structural biasing issue of technology-independent decomposition remapping look-up table networks and support as cost function. The suggested methodology realized large improvements in delay and area of benchmark circuits. These decomposition methods are applicable to the optimization of UART implementations on the FPGA platforms and where more than one UART channel to be integrated is shared.

Chen et al. [8] explored the power and area efficient FPGA building blocks, which are pegged on ferroelectric field-effect transistors. Their work revealed that look-up tables and routing switch based on FeFETs could potentially make a substantial improvement in the power-delay product relative to their CMOS counterparts. FeFETs do not undergo voltage swamping meaning that they provide valuable opportunities to use UART setups that need settings to be stored across power cycles. Potential Reduction of the area by 8 percent over conventional CMOS implementations implies functionality to add more UART features without adding silicon footprint.

Kang et al. [9] gave a detailed overview of approximate computational methods, circuits to applications. Their discussion identified the inherent issues of power dissipation that continued CMOS scaling confronts and the necessity of new paradigms of computation. Whereas in approximate computing computations are meant to trade accuracy in favor of efficiency, the UART communication model dictates that its behavior must be deterministic and cannot support the use of approximation in transmitting data. Nevertheless, the power minimization methods presented in their article can be also applicable to the low-power UART implementations in battery-operated systems.

Verma et al. [10] examined the low-power consumption methods with special reference to the FPGA implementation of wireless gadgets. They performed work on architectural-level minimization of power on arithmetic logic units, showing that great performance gains were made with XPower Analyzer. The mentioned techniques, such as clock gating and operand isolation, can be applied directly to UART designs, where power efficiency is the key consideration. When implemented on Spartan-3E FPGA, there is also a point of reference on which resource usage and power properties are compared.

The issue of low energy standby-sparing in hard real time system was handled by Ejlali et al. [11], wherein the authors discussed ways of balancing power management and the fault tolerance requirements. Their contribution on

efficient methods of redundancy that consumes less energy can be used in UART systems that need high reliability in safety-critical systems. Their study of trade-offs between fault coverage and energy consumption applies to informed decisions about the energy usage of parity implementation and error handling techniques.

Salehi et al. [12] generalized this work to multi-core systems that used two-phase low-energy N-modular redundancy. Their solution shows how the redundancy method that can be optimized to be energy-efficient and at the same time fault tolerant. In the case of UART use in multi-core systems, such concepts propose opportunities to distribute communication processing among cores and ensure the data integrity.

Viegas et al. [13] came up with power-efficient anomaly-based embedded system intrusion detection, which tackled the security issues as well as energy consumption. Their paper emphasises the increased need of security in communication interfaces, which implies that future UART implementations might require addition of authentication and encryption capabilities to the bare functionality of data transfer.

Bakhshalipour et al. [14] explored the mechanisms of fast data delivery of many-core processors with an emphasis on the optimization of cache and memory hierarchy. Although their work focuses on processor architecture and not on peripheral interfaces, the concepts of efficient data movement apply to the UART systems installed on complex System-on-Chip designs where the communication latency determines the overall system performance.

Melhem et al. [15] analysed the relationship between power manipulation and fault handling in real time systems. Their discussion on the impact of dynamic voltage scaling on the system recovery mechanisms and reliability of system presented a useful background on the UART implementations that need to be stable under different power conditions.

Zhu et al. [16] designed techniques of scheduling techniques, dynamically adjusting the voltage, with slack reclamation in multiprocessor real-time systems. Such methods make best use of the energy use and address timing requirements and can have potential advantages with UART systems embedded in power-controlled System-on-Chip systems.

It is observed in the literature survey that there are a number of gaps that prompt the current work. Although much has been done on approximate computing, optimization of FPGA, and power management, there has been a paucity of research on the improvement of UART reliability using architectural buffering mechanisms. Moreover, known UART implementations tend to reconsider buffering as an hacked feature and not part of the design. This paper fills these gaps by describing a coherent UART-FIFO design that has been achieved by adhering to the systematic VLSI design process.

A. Limitations Of Conventional UART

Traditional UART applications have been used in the electronics industry over decades with the provision of easy and effective serial communication. Nonetheless, these classical designs have a number of drawbacks that are becoming more problematic in the contemporary high-performance system. The inherent limitation is the lack of buffering between the serial line and the system interface whereby the processors will be involved directly in all the data transfer operations [17].

Under no buffering, each received byte is sent out as an interrupt, and the processor must be interrupted to process it immediately. In case the processor is running on higher priority tasks or interrupt is disabled temporarily, there is a loss of incoming data bytes whose recall cannot be guaranteed. This overrun condition is a critical failure case in real time systems where loss of data is not allowable. Likewise, the processor has to be engaged at all the times in the process of transmission until each and everybyte is packages in a serialization process and they actually cost the system a lot of processing cycles that would otherwise be utilized in other activities in the system [18].

Baud rate synchronization requirement provides further limitations. The two communicating devices need to run at exactly the same rate of bits per second, normally by use of crystal oscillators or ceramic resonators. Any difference in frequency, even within manufacturer tolerance, may add up over a series of bytes, and may ultimately result in framing errors. This is increased by long data streams or continuous transmissions, which restrict the viable duration of error-free communication sessions [19].

This is added by the power consumption considerations that complicate the traditional UART design. This is because of the continuous clock running that is necessary to monitor the serial line to detect the start bits as opposed to deep sleep modes that would otherwise save power. This constant operation in battery-powered systems has a major effect on the lifetime of the system and might require a higher capacity battery or a more rapid charge rate [10].

B. Network-on-Chip Approaches

The shortcomings of conventional bus architectures in System-on-Chip designs have promoted interest in Network-on-Chip solutions that are borrowed ideas of the packet-switched networks. NoC architectures separate the issues of communication and computation with protocol stack implementations allowing the creation of scalable interconnection fabrics in complex systems [20].

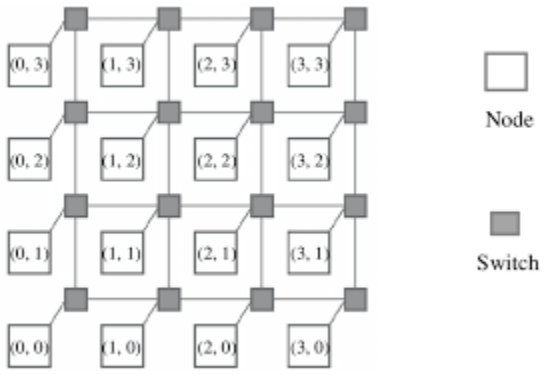

Figure 1 2D Mesh NoC Topology

As shown in Figure 1, a typical 2D mesh NoC topology has switches which route packets between resource nodes such as processors, memories and special accelerators. The architecture is intrinsically scaled and is able to reuse communication infrastructure between designs.

The 2D mesh topology has a number of benefits on on-chip communication such as the regular structure, wiring delays are predictable, and supports multiple transactions at the same time. NoC implementations however come with a major overhead of area and they need advanced routing algorithms to control congestion and quality of service guarantees.

C. Self-Tune Redundancy Techniques

Self-tune redundancy or self-mode redundancy, is a generalization of N-modular redundancy, in that it adds fault tolerance mechanisms that are adaptive. In triple modular redundancy design, three systems are set up and are configured to work on the same data and a majority voter is chosen to make the right choice. This method conceals individual failures and offers uninterrupted proper functioning despite faults in one of the modules [21].

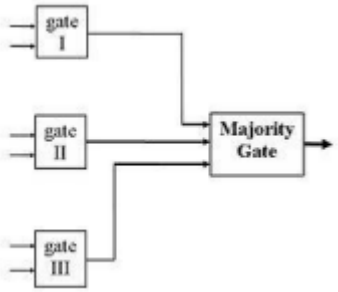

Figure 2Self-Tune Modular Redundancy

In figure 2, a self-tune modular redundancy design is illustrated in which the three logic circuits were fed to a majority vote gate. In case any of the circuits become faulty, the other two correct circuits continue to produce the correct majority output such that the system continues to operate.

The implementation of self-tune redundancy to UART systems has got opportunities to realize high reliability in safety-critical communication systems. Nonetheless, its area and power overhead makes it impractical in embedded systems with a tight cost constraint like triple modular redundancy. Redundancy can be used selectively on critical control logic and not on the entire system, which will provide superior trade-offs between levels of reliability and resource consumption.

D. *Permanent Fault Considerations*

With the CMOS fabrication technology continuing to scale down to smaller feature sizes, permanent faults are becoming an increasing concern to the reliability of the system. Electro-migration and stress induced dielectric breakdown of gates, and permanent circuit failures which eventually accumulate due to system lifetime [22].

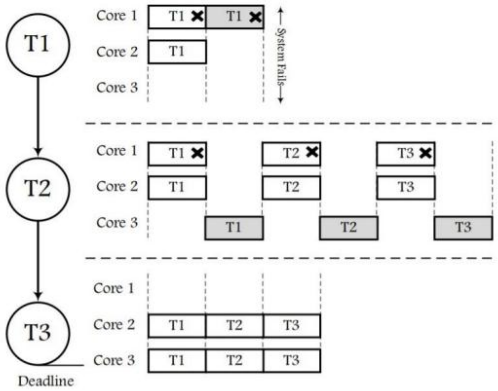

Figure 2 Reconfigurable FIR Tap with NoC

Figure 3 shows an example of a reconfigurable FIR tap design with NoC based communication between processing units. This architecture allows the dynamic reconfiguration of faulty processing units to achieve a computational throughput.

Permanent fault tolerance does not share any approaches with transient fault handling. Whereas extremely short lived faults can be overcome by retrying or error correction codes, permanent faults will require architectural redundancy or reconfiguration. RFIR approach keeps history of executions of various cores and allocates responsibilities to reduce the likelihood of faults based on the observed behavior.

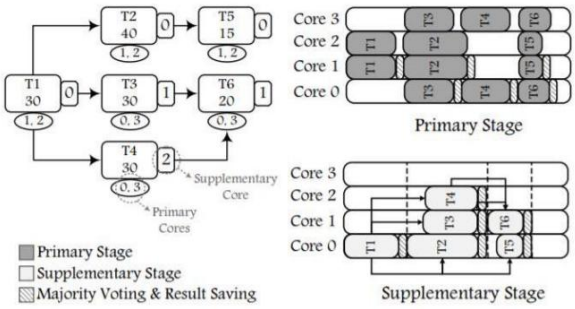

Figure 4 RFIR Scheduling

Figure 4 would show the scheduling of RFIR by including tasks on the cores based on historical fault distribution. In the case of permanent fault in certain cores, the scheduler reassigns the tasks in such a way as to avoid the impacted resources whilst preserving the real-time constraints.

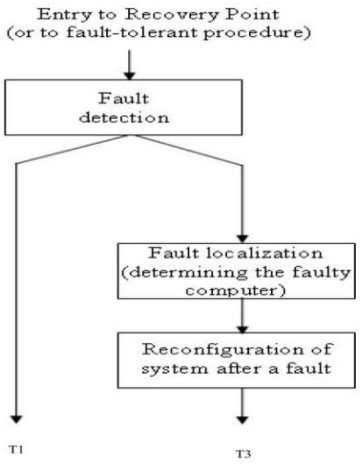

Figure 5Reconfigure Flow of FIR Filter Core

Figure 5 illustrates the reconfiguration flow with the FIR filter cores in response to any variations in faults by modifying their runtime schedules. It is an adaptive design style that ensures it is functional even after acquiring permanent faults, and prolongs system life and minimizes maintenance costs.

III. PROPOSED METHODOLOGY

A. System Architecture Overview

The FIFO system of the proposed UART is a solution to the drawbacks established in the application of traditional implementations that have been overcome in terms of architecture that enables buffering functionality without affecting the adherence to the conventional UART protocols. There are four large functional blocks that make up the entire system which works in concerted manner to realize reliable asynchronous communication.

The top-level architecture is connected to the well-defined interfaces of these blocks and gives independent checking possibilities as well as the design reuses. Key system parameters such as FIFO depth, data width, baid rate divisor and parity options can be configured using parameterization without being changed in the implementations of core modules.

B. Module Descriptions

1) UART Transmitter Module

The transmitter module makes parallel-serial conversion of the data bytes based on the standard UART frame formats. Every transmission frame starts with a start bit (logic low), then data bits which can be configured to any number (usually 8), optional parity bit and lastly one to more stop bits (logic high). The module communicates with the transmit FIFO with control signals of data availability and new data readiness.

The operation of the transmitters is based on a state machine, which controls the sequence of elements in the frames. Upon the signal of the FIFO that there is data available the transmitter reads the parallel-bit into an internal shift register and starts the transmission in time with the baud rate clock. It has a shift register, which is serially written with least significant bits to most significant bits to enable it to be used with the industry-standard UART implementations. Status signals such as transmit complete and transmitter busy indicators produced by the transmitter are available to the higher-level system software to monitor the flow control. These signals also synchronize with the FIFO control logic in order to request new data when a current transmission is over.

2) UART Receiver Module

To assemble parallel data bytes in an incoming stream of serial data, the receiver module monitors bit transitions on start bits and samples the bits that immediately come after that at a controlled frequency specified by the baud rate generator. Constant scan of the serial input line allows the falling edge which marks the start bit initiation to be detected immediately.

When the receiver spots a valid start bit, he waits half a bit period to sample around the middle of each bit to optimally provide timing margin and minimize sensitivity to distortion of the signals. The bits sampled are loaded into a shift register until the entire data frame has been obtained, whereby, the assembled byte is subjected to optional parity checking.

The received data bytes are passed over to the receive FIFO with status data which contains whether parity or framing errors were found during the reception process. This error data can be used by upper level software to add the correct error handling measures such as retransmission requests or logging of errors.

3) FIFO Buffer Design

The FIFO buffer is an important improvement that facilitates sound communications. As a dual port memory, with its own read and write pointers the FIFO preserves the order of data, although it allows access by transmitter or receiver modules and system interface logic simultaneously.

The FIFO control logic produces status flags that represent the conditions of occupancy of buffers such as empty, full, almost empty and almost full. These flags allow push and pull mode flow control in which the system has the ability to control the data flow without polling or interrupt overheads. Almost empty and almost full thresholds are programmable to allow flexibility to meet various application needs.

To support multi-clock domain operation, the FIFO includes synchronization in the design, which avoids metastability in transfer of pointer information across asynchronous clock domains. The pointers are coded by gray code to make sure that each increase by one bit will cause only a single bit change in the coded value, reducing the chances of synchronization errors.

4) Baud Rate Generator

The baud rate generator produces the timing of the tick pulse with a perfect metering of bit timing in the transmitter and the reciever modules. The generator is based on the system clock and is programmably divided in support of the standard baud rates of 9600, 19200, 38400, 115200 and above based on the system clock frequency.

The generator architecture incorporates a counter which has to count the cycle of the system clock and produce an output pulse when the counter reaches the programmed divisor value. This is a very efficient strategy with minimum logic resource cost and gives the correct timing. To simulate it a smaller divisor speeds up the verification process by reducing the bit time without changing timing relationships between bits.

IV. MODULE IMPLEMENTATION

A. Verilog HDL Implementation.

The implementation is in Verilog HDL, which offers suitable levels of abstraction in describing the digital circuits and holds the synthesizability of the physical implementation. The multiple modeling styles that can be supported by Verilog allow structural connectivity and behavioral functionality to be modeled efficiently in a single design environment.

Module interfaces are named in the same way and signal polarities are similar, making integration easier and minimizing chances of connection mistakes. Figure 3 Design reuse with parameterization Designs with different FIFO depth requirements, data width requirements, and baud rate requirements are reused across applications.

B. Target FPGA Platform

Xilinx Spartan-3 FPGA family is used as a target implementation platform, with cost-effective programmable logic that is used in prototyping communication interfaces. The XC3S400 device offers a sufficient logic capacity to the UART-FIFO design and leaves space to add more to the integration of the system.

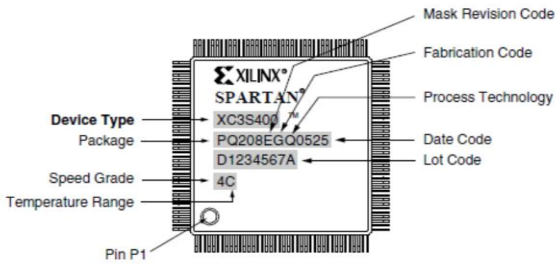

Figure 6Reconfigure Flow of FIR Filter Core

Figure 6 illustrates Spartan-3 package model with pin

assignments and physical attributes that will be important with regard to board level integrations. The 208-pin PQFP package has adequate I/O to connect to UARTs and more interfaces to integrate the system.

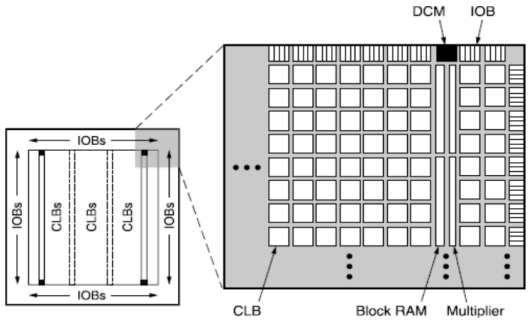

Figure 7Spartan-3 Architecture

Figure 7 illustrates the Spartan-3 architectural organization, which displays the lay out of Configurable Logic Blocks, Block RAM, multiplier blocks, and Digital Clock Managers which offer implementation resources to the UART-FIFO design. The block RAM resources efficiently implement the FIFO memory, and the CLBs are used to provide the flip-flops and look-up tables used to store the control logic.

V. IMPLEMENTATION RESULTS

A. Simulation Waveform Analysis.

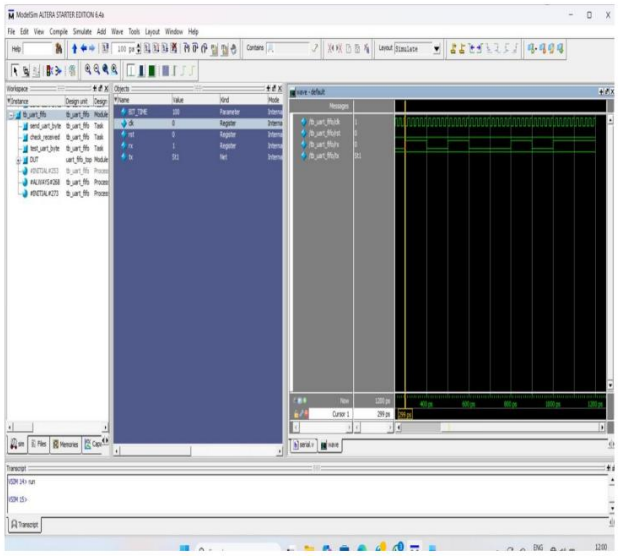

Figure 8Simulation Waveform

The operation of the entire UART-FIFO system using ModelSim, simulation shows that the system functions correctly under a variety of test conditions. Figure 8 illustrates the simulation waveform of some important signals such as system clock, reset, serial data input and output and FIFO status flags and internal control signals.

The waveform will ensure the correctly configured start bit detection by the receiver module, correct data bits sampling, and generation of stop bits.

FIFO write and read actions maintain the order of data as well as status flags are used to show the occupancy of the buffers. Test patterns ensure that various data patterns such as alternating bit patterns and worst-case transitions have been correctly transferred and received.

B. Resource Utilization

The achieved results of synthesis show that the entire UART-FIFO design requires around 350 lookup tables and 220 flip-flops in the Spartan-3 device used. When used as a FIFO memory, block RAM implementation of FIFO memory will use 1 memory block to store 16-byte depth of a FIFO memory; however, depths beyond this limit will be implemented with a linear increase in memory usage. The analysis of timing shows that the maximum operating frequency is more than 100 MHz, which offers a significant range of standard UART baud rates. The baud rate generator and the FIFO pointer comparison logic both have critical paths, which can be pipelined in case higher frequencies are required.

VI. CONCLUSION

This paper has provided design, implementation and verification of a UART with inbuilt FIFO buffering Verilog HDL in the systematic VLSI design. The basic weaknesses of traditional UART implementations have been overcome through the proposed architecture, which offers buffering functionality that removes data loss through different load conditions.

Interaction between transmitters and receivers is further facilitated by the implementation of FIFO buffers to ensure that communication is possible in spite of the unpredictable availability of processors as well as in cases where the rate at which the data is transferred is near the maximum possible system rates. Status flags give convenient flow control command alternatives, and are compatible with conventional UART protocols.

The design approach and the quality of implementation is validated by the ability to get simulation results that ensure proper functionality in various test conditions. Consumption of the resources is low, making it

possible to be integrated into cost-sensitive applications without huge area cost. At standard operating frequencies, timing closure has been shown to be practically viable at real world frequencies.

The design can be followed in a number of ways in future work. The auto-baud rate detection would make those communication rates that are not known adapt automatically simplifying the system configuration. Request-to-send and clear-to-send signals would be used as a hardware flow control to make variety of robustness to long-distance communication.

The connection to the AXI bus interfaces would allow its integration into more complicated System-on-Chip designs. Noise cancellation in the coding of error correction would increase the reliability of the performance in noisy environments at the expense of overhead.

The parameterized nature of the design should provide that these extensions can be added without redesigning fundamental functionality and that the current work will retain its advantages with the future evolution being allowed to proceed to fulfill any new need.

REFERENCES

[1] A.Ejlali, B. M. Al-Hashimi, and P. Eles, "Low-Energy Standby-Sparing for Hard RealTime Systems," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 2012.
[2] M. Salehi, A. Ejlali, and B. M. Al-Hashimi, "Two-Phase Low-Energy N-Modular Redundancy for Hard Real-Time Multi-Core Systems," IEEE Transactions on Paralleland Distributed Systems (TPDS), 2016.
[3] E. Viegas, A. O. Santin, A. Franc¸a, R. Jasinski, V. A. Pedroni, and L. S. Oliveira, "Towards An Energy-Efficient Anomaly-Based Intrusion Detection Engine for Embedded Systems," IEEE Transactions on Computers (TC), 2017.
[4] A. Munir, S. Ranka, and A. Gordon-Ross, "High-Performance EnergyEfficient Multicore Embedded Computing," IEEE Transactions on Parallel and Distributed Systems (TPDS), 2012.
[5] M. Bakhshalipour, P. Lotfi-Kamran, A. Mazloumi, F. Samandi, M. Naderan, M. Modarressi, and H. Sarbazi-Azad, "Fast Data Delivery for Many-Core Processors," IEEE Transactions on Computers (TC), 2018.
[6] R. Melhem, D. Mosse, and E. Elnozahy, "The Interplay of Power Man- ´ agement and Fault Recovery in Real-Time Systems," IEEE Transactions on Computers (TC), 2004.
[7] D. Zhu, R. Melhem, and B. R. Childers, "Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multiprocessor Real-Time Systems," IEEE Transactions on Parallel and Distributed Systems (TPDS), 2003.