Volume 10, Issue 1
February 2021

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X



# Design of Fault Tolerance block acceleration with customization approach for power reduction in WSN applications

**[1]Vilabha Patil and [2]Shraddha Deshpande**

[1]Research Scholar, Walchand College of Engg., Sangli
Asst. Prof., Dept of E&TC, RIT Rajaramnagar, Maharastra, India
[2]Dept. of Eectronics, Walchand college of Engg., Sangli, Maharastra, India
**E-mail:** [1]vilabha.mane@ritindia.edu, [2]shraddhha.deshpande@wachandsangli.ac.in

## ABSTRACT

The current developments in wireless sensor network (WSN) headed to the implementation of fault-tolerant, low power, and low-cost wireless sensor nodes. FPGA based soft-core processor becomes an attractive solution to design the sensor node with the required flexibility. In this paper, we propose a new soft-core processor built an architectural model for WSN nodes based on the idea of custom instructions. At the architectural level power saving is possible by custom instruction design by exploiting parallelism. The custom instruction accelerates the time-critical operations as custom hardware logic block. Thus by considering power saving and fault tolerance factor the effective error correcting code cyclic redundancy check (CRC) is designed as custom instruction and implemented as a proposed work with NIOSII soft-core processor using linear feedback shift register (LFSR) in which the data is processed sequentially. It helps to improve the performance with low power fault tolerance in comparison with the software-only implementation of the respective algorithm. With this customization, the design of CRC is further accelerated with parallelism, while adding as custom instruction. This acceleration with parallelism leads to significant performance improvement w.r.t. custom implementation. The overall power efficiency, reprogrmmability, performance and speed of execution are improved considerably with customization.

**Keywords:** *Fault Tolerance, CRC, NIOSII Processor, Custom Instruction, Acceleration with parallelism*

## 1. INTRODUCTION

The demand for high-performance WSN node is growing and its power consumption has threatened the life of the network. With the recent rapid growth of applications in the WSN domain, there is a need to provide a better performance, high flexibility, and scalability. Extending the life of WSN in each application is significant.
Until now the WSN nodes are based on low power microcontrollers such as MSP430 [3] and ATmega128 [1,2]. These designs lead to less power consumption but do not provide necessary scalability and flexibility as per applications of WSN. Thus the design of sensor nodes with commercial of the shelf devices like microprocessors and microcontrollers do have the static functionality. Soft-core processor [4] based solutions enable preferred scalability and flexibility with complete customizable capability. Hence in this paper, the sensor node with soft-core processor as a processing unit is implemented. Numbers of soft-core processors are available like Microblaze, Picoblaze, Leon, and NIOSII. Amongst these NIOSII processor has a custom instruction facility, i.e. up to 256 custom instructions can be added to NIOSII processor. Though NIOSII is better for customization, NIOSII soft-core processor is chosen for implementation.

Failures will occur predictably because sensor nodes are arbitrarily installed in an unfriendly nature environment. Main work in most of WSN application is to monitor the remote areas and send the collected information to the destination node. In these applications the data transmission is affected by environmental factors like noise, so the data received at the receiver end is different than the original information signal. Therefore, it is necessary to consider faults and power consumption issues while designing WSN sensor nodes [5]. The general idea to overcome this is to add redundancy bits error correction techniques.

Cyclic Redundancy Code (CRC) is most commonly used in wireless communication as an effective method for detection of an error in data to be transmitted. Several software implementations of CRC computations are realized with the help of processors or controllers [6,7,8]. Though these general purpose processor based systems on chip designs are highly expensive with inflexible, inefficient results. It is needed to make this CRC block configurable so that it will be updated as per application. The addition of flexibility through reconfigurability is a method to reduce the cost of design. Field programmable gate array (FPGA) becomes an attractive option to achieve flexibility and configurability. Fully FPGA based CRC computation circuit is implemented in [9]. It helps to reduce the complexity and time of programming. Various hardware CRC implementations on FPGA have been introduced in [10,11,12]. However, the combination of hardware and software leads to improve system performance by the method known as codesign. In the case of WSN embedded systems demand is increasing for hardware-software codesign. Hence we implement the CRC as custom instruction which introduces codesign. The design aspect used for CRC calculation is using the most established method LFSR [13] with shift registers and logic gates. This

Volume 10, Issue 1
February 2021

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X



design performs computation by handling data sequentially. This custom implementation with LFSR is not appropriate for today's high speed WSN applications. The parallel design can perform faster computation than the serial design aspect. Thus customization is further modified for CRC acceleration as custom instruction using parallel circuit design.

Parallel CRC is generated by many methods. One of the methods for implementation of parallel CRC is using recursive formula [14,15] from serial implementation. In this number of bits are processed simultaneously independent on technology. Another method is the calculation of CRC using byte enable [17] method. It is faster than bitwise method. In [16] original message data is systematically decomposed into a set of subsequences using the theory of Galois field. Lookahead technique is employed for parallel CRC computation and to speedup the computation. However, each of the methods has its own advantages and disadvantages. Some are more suitable for high speed designs where area required is more. Some are with compact designs with less speed. The method introduced in [18] overcome the drawbacks of the above mentioned methods. In this method parallel CRC VHDL or Verilog code generation by using practical method is introduced. The algorithm of implementation is explored in section 2.3b.

In this paper, we propose an innovative custom codesign based design mechanism for the fault tolerance block in WSN applications. Initially, customization of fault tolerance block i.e. the design of CRC is done using LFSR. Later the custom block is accelerated and optimized by the addition of parallelism to CRC design using a parallel algorithm mentioned in section 2.3b while adding as custom instruction. The main contribution of this paper is customization with parallelism and optimization for fault tolerance block in WSN applications.

The aim of this design is to decrease the overhead of sensor node processing unit, lessen the cost of algorithm implementation, additionally with increasing the use for different applications modulating flexibility.

This paper is divided into six sections. Section 2 presents the overview of technology and algorithm. Section 3 explores the hardware and software used for the design and implementation. Experimental design flow is illustrated in section 4. The results are discussed in section 5. Finally concluding remarks are given in section 6.

## 2. OVERVIEW

The approach used in this paper to achieve above goals is based on soft-core processor NIOS II of ALTERA FPGA. In this design the fault tolerance i.e. CRC calculation is implemented as hardware on FPGA with two aspects. First aspect used is serial calculation and implementation using LFSR in FPGA and add it as a custom block to NIOS II processor through software macro. Further this hardware design and implementation on FPGA is accelerated using parallelism then, added as custom instruction to NIOS II processor.

In this section some initial information related to design is provided as given below:

### 2.1 NIOS II Processor

NIOS II is 32 bit configurable embedded soft-core processor specially designed for ALTERA family of FPGA. NIOS II's basic functionality can be extended by adding custom peripherals or custom instructions. By the addition of custom instructions, time-critical software operations can be accelerated as a custom logic block. It has three different member families like NIOS II/e (economy), NIOS II/f (fast), and NIOS II/s (standard). Each of these is having a specific price and performance range. It has pipelined RISC architecture with isolated 16bit instruction bus and 32bit databus. The register file is configurable with 128, 256 or 512 registers. Amongst these only 32 registers are accessible as general purpose registers.

The NIOS II processor and many other components such as custom peripherals and standard peripherals are integrated into ALTERA DE2 board to form the total system as shown in fig.1 CYCLONE IVE FPGA device enable the process of interfacing the NIOS II processor and peripherals to the DE2 board. Avalon bus is used to form the interconnection network by connecting these components.

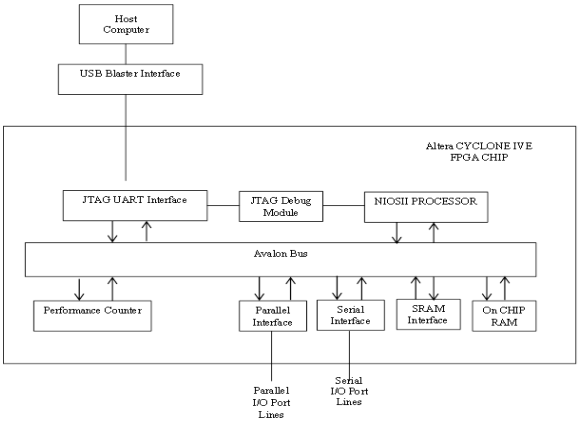


Figure 1. NIOSII implemented on FPGA

### 2.2 NIOSII custom instruction

A custom instruction allows decreasing a complex sequence of standard instructions to a single instruction implemented as custom hardware block. As many as 256 custom instructions are added to NIOSII processor. To map the custom instructions, NIOSII processor uses GNU compiler collection (GCC). The custom instructions are accessed through software macro directly by C or C++ application code. The NIOS II processor has four types of custom instructions like combinational, extended,

Volume 10, Issue 1
February 2021

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X



multicycle and internal register file. While implementation of custom instruction specific ports are used as hardware interface and software interface.

### 2.3 Cyclic Redundancy Check (CRC)

CRC is the most dominant error detecting code. Transmitter T sends a sequence of data with N bits like {b0, b1,….., bn-1} to the receiver. The transmitter generates another data sequence {b'0, b'1,…b'n-1} to permit the receiver for detection of data error. Both data sequences are concatenated and this is divisible by sequence like p = {p0, p1, …., pm}. When transmitter sends the total data sequence to the receiver, then receiver divides the data sequence by p. If the remainder is zero, it is considered as data with no error.

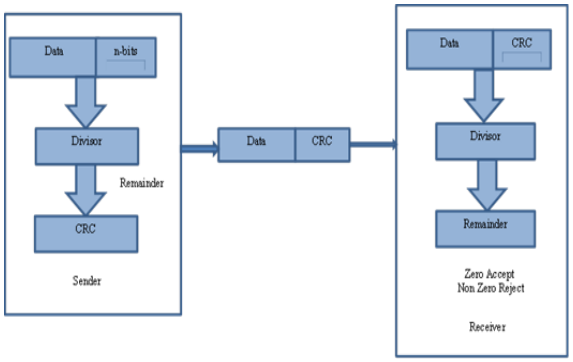


Figure 2. Block Diagram of CRC

### 2.3. a. CRC Algorithm using LFSR

CRC as custom instruction implementation using most established method linear feedback shift register (LFSR) [8]. In this approach, the data is processed sequentially.

1. Build m-bit LFSR
2. Figure out the flip-flops as FF1 on the right side
3. Q output of leftmost flip-flop gives the feedback path.
4. Identify the original polynomial of the form $X^m+\ldots+1$
5. The $X^0$ =1 term links to associating the feedback directly to the D input of FF1.
6. Every term of the form $X^n$ links to connecting to an XOR between FF n and n+1

### 2.3. b. CRC Algorithm with Parallelism

CRC custom instruction implementation using parallel computation approach. In this approach, the data is processed simultaneously.

1. Initialization
N = Data width
M = CRC polynomial width
e.g. CRC8 with – 7-bit data
N = 7, M = 8
2. Serial CRC implementation for a given polynomial
3. Implementation of parallel CRC is a function of N-bit data input and M-bit presents CRC state.
Two matrices are build
H1 = Mout Next CRC state – as a function of input data (Nin) when Min=0
Mout = CRC Parallel [Nin, Min=0]
H2= Mout (Next CRC state as a function of Min current CRC state when Nin=0
So, Mout = CRC parallel [Nin=0, Min]
4. Build Matrix H1 for N= 4
5. Build matrix H2 for M= 5
6. XOR inputs Min[j] and Nin[j] which give result = Mout[i]

### 2.4 Custom Instruction Block Diagram

As shown in fig. 3. the better design can be achieved through custom instruction based paradigm. The sensor data collected is processed by a custom logic block designed with the above mentioned aspects. The custom logic block is designed in hardware adjacent to the arithmetic logic unit between the datapath of the processor. Thus it helps to reduce the sequence of instructions into single custom instruction. These custom instructions are mapped through the GNU compiler. The macro generated is directly used in C or C++ code.

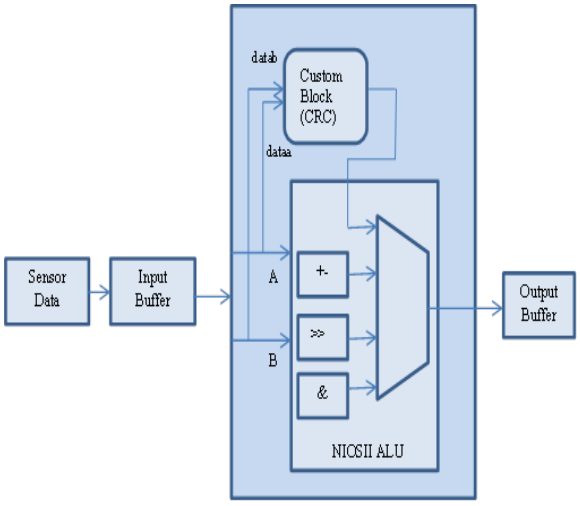


Figure 3. Custom Logic Block Diagram

The proposed method in this paper implements Soft-core processor on FPGA using Quartus tool and Altera Development board. The Quartus II and QSYS integration tools are used to design the CRC custom instruction as per need with less complexity. ALTERA provides various range of Development board [21] with different feature such as Cyclone, Stratix edition, the designer can choose according to the application requirements.

Volume 10, Issue 1
February 2021

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X



## 3. HARDWARE SOFTWARE REQUIREMENTS

The hardware-software required for system generation is as explained below:

### 3.1 Software

a. Altera Quartus II software version 13.1 or later: Development of hardware design files can be done with this software. It is also used for synthesizing netlist for the design and to download the .sof file to target FPGA board.
b. QSYS integration tool: This integration tool is used to design the hardware system with the help of components like processors, I/O interfaces, memories, timers and other components as per the requirement of application. This hardware system designed and generated and compiled in QUARTUSII tool.
c. NIOSII Embedded Design Suite: NIOSII application software is written using this development tool. Two approaches are provided to develop the application software as:

1. The command line interface
2. NIOSII SBT for Eclipse

### 3.2 Hardware platform

NIOSII Development board :

Different board series are present for NIOSII cyclone series, Stratix series, Arriax series. In this paper, it is decided to use Altera DE2115 [21] development and education board with CYCLONE IVE device (fig. 2). This board has many features that allow the designer to implement a varied range of circuit designs as per application.

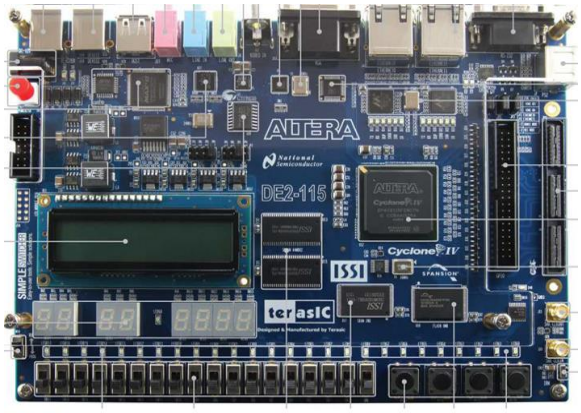


Figure 4. Cyclone IV E Development Board

## 4. EXPERIMENTAL DESIGN FLOW

The experimental system development flow includes hardware- software design and implementation of the application.

### 4.1 Design and develop NIOSII hardware system

The NIOSII is a soft-core processor that could be implemented on Altera FPGA. In this, the total hardware system and software system creation for simple message display has been done. NIOSII hardware system is built to display the simple message “Hello from NIOSII”
Design of NIOS II hardware system requires the following necessary components:

- NIOS II processor core, that’s where the software will be executed
- On-chip memory to store and run the software
- JTAG link for communication between the host computer and target
- Hardware (typically using a USB-Blaster cable)
- LED peripheral I/O (PIO), be used as indicators

### 4.2 Implementation of Custom instruction

There are two important blocks of the custom instruction as a customized logic block and the software macro. The customized logic block is implemented in hardware for the acceleration of standard logic operations. During the implementation of custom logic, the software macro is used to access the accelerated logic through a software program of NIOS II IDE. The algorithm is implemented in hardware to accelerate the performance of algorithm. The NIOS II hardware system is integrated with the custom logic block and recompiled with QUARTUSII software.

### 4.3 Software implementation flow

The custom instruction generated in system.h is used for the realization of algorithm in NIOS IDE. This source program is then compiled using NIOSII build facility to convert it into executable code.
Experimental design flow is as shown in fig.5.

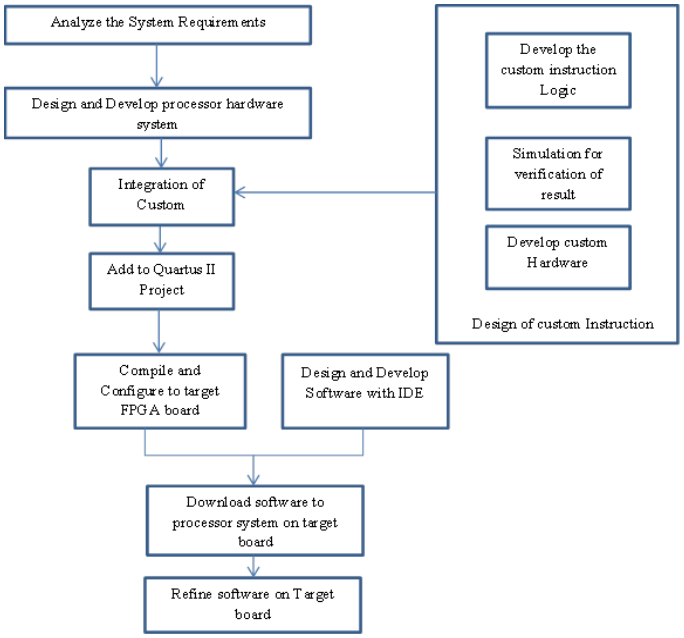


Figure 5. Experimental Design Flow

Volume 10, Issue 1
February 2021

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X



## 5. RESULTS AND DISCUSSIONS

In this work DE2-115 board of ALTERA is used for implementation with Cyclone IV E computational device. The proposed QSYS design consists of a NIOSII processor, On-chip Memory, JTAG UART and performance counter to measure the cycle count.
Additionally, the custom instructions generated CRC8, CRC16, CRC32 are also interconnected in QSYS design. The basic hardware design is as shown in fig.6.

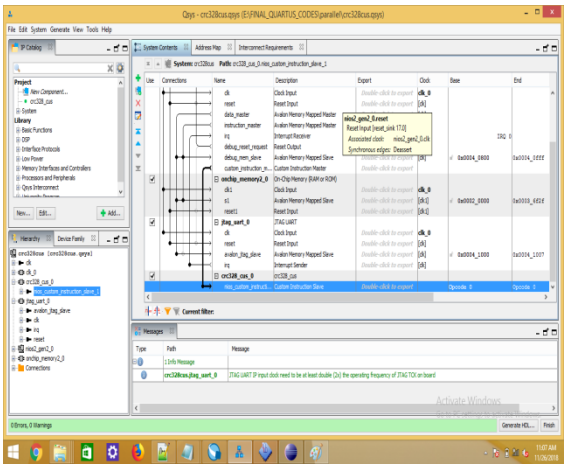

Figure 6. Interconnect Logic in Qsys

After successful compilation of system with custom instruction, hardware generation is done with cyclone IV Altera FPGA. Subsequently generation of system hardware, the algorithm is implemented with the help of custom instruction in NIOS IDE. Hence the project is built with the help of build project command. After a successful build of the project, the C program is run by command run as NIOSII hardware. The result is displayed on the NIOSII console window as shown in fig. 7. The result includes CRC value with performance parameters like clock cycle count and execution time of the respective algorithm.

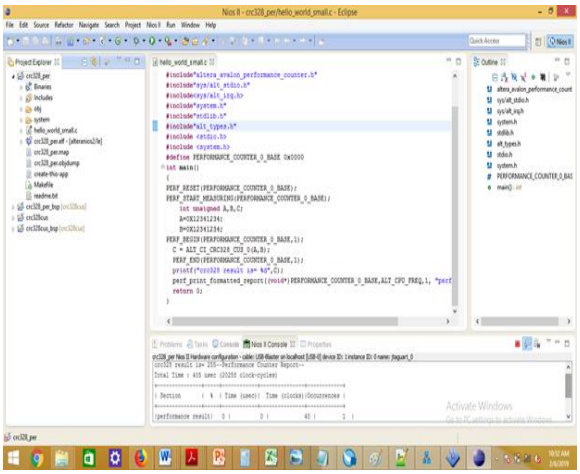

Figure 7. NIOS II IDE

The results of custom instruction based approaches are compared with software only implementation code with NIOS II processor, based on the parameters likes power consumption, clock cycle count and time of execution with performance improvement.
The power analysis of custom instruction added to the NIOS II processor has been done with the help of powerplay analyzer tool of Quartus II.
Total power dissipation is divided into two main parts
$P_{total} = P_{dynamic} + P_{static}$
Static power ($P_{static}$) is power consumed while there is no circuit activity. Dynamic power ($P_{dynamic}$) is the user design power due to input data pattern and logic design internal activity, hence $P_{dynamic}$ power is considered for analysis and comparison.
The power consumption comparison is done for both aspects i.e. custom instruction with the serial aspect and accelerated custom instruction with parallelism. The power analysis with both NIOS II processor versions like NIOSII/e (economy) and NIOSII/f (fast) for different lengths of CRC is as presented in Table 1.

**Table 1.**Comparison of Power consumption

| Custom instruction | Software only Implementation (mw) | | Custom Implementation (mw) | | Custom Implementation with parallelism (mw) | |
|---|---|---|---|---|---|---|
| | E | F | E | F | E | F |
| CRC8 | 0.95 | 1.15 | 0.84 | 1.08 | 0.74 | 1.01 |
| CRC16 | 0.95 | 1.15 | 0.86 | 1.11 | 0.75 | 1.04 |
| CRC32 | 0.95 | 1.15 | 0.90 | 0.97 | 0.76 | 1.00 |

From the power analysis results, it is observed that the power required for execution of CRC without custom instruction i.e. software-only implementation is improved by 11% as compared to CRC with custom instruction i.e. in hardware. As well as the power required for the custom instruction design with parallelism is less as compared to the initial custom instruction approach i.e. the parallel aspect can reduce the power dissipation by 6%. From the analysis, it is observed that customization can save the power of soft-core processor.
fig. 8. and fig.9. compares the power consumption of diverse lengths CRCs computed with both versions of NIOS II processor E and F using the software only implementation, custom implementation and custom implementation with parallelism.

Volume 10, Issue 1
February 2021

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X



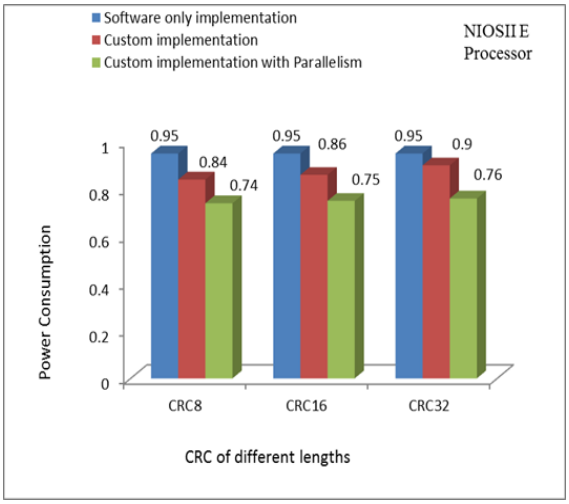


Figure 8. Comparison of power consumption for NIOS II(E) processor

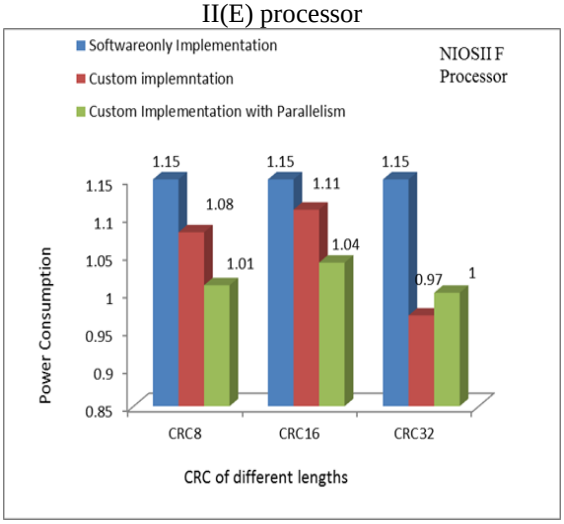


Figure 9. Comparison of power consumption for NIOS II(F) processor

Execution time and clock cycle count required for all implementation i.e. for software only implementation, custom implementation and acceleration of custom block with parallelism approach are used as a performance metric in order to identify the improvement achieved with customization in comparison with software-only implementation. The evaluation parameter results of CRC are as shown in Table 2.

**Table 2.** Comparison of clock cycles and execution time

| Custom Instruction | Clock Cycles | | | | | | Execution Time(µsec) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Software only Implementation | | Custom Implementation | | Custom Implementation with Parallelism | | Software only Implementation | | Custom Implementation | | Custom Implementation with Parallelism | |
| | E | F | E | F | E | F | E | F | E | F | E | F |
| CRC8 | 387 | 32 | 46 | 20 | 40 | 20 | 7 | 1 | 0.92 | 0.4 | 0.8 | 0.4 |
| CRC16 | 739 | 38 | 47 | 27 | 40 | 17 | 14 | 2 | 0.94 | 0.54 | 0.8 | 0.34 |
| CRC32 | 1050 | 57 | 70 | 44 | 40 | 14 | 21 | 3 | 1.4 | 0.88 | 0.8 | 0.28 |

The results from Table 2 shows that acceleration of algorithm as custom instruction is about 84% faster than the software only implementation. The results with custom instruction drastically improve the performance than without custom instruction aspect of implementation. Additionally, the parallel circuit based custom instruction approach of this research has a comparable improvement in execution time and clock cycle count than the serial aspect of custom instruction implementation. Thus acceleration of custom block with parallelism reduces the cycle count by 27% as compared to custom block implementation with serial aspect. Such improvement is possible due to the acceleration of algorithm with hardware-software codesign. fig. 10 and 11 compares the execution time of diverse lengths CRCs computed using the software only implementation, custom implementation and custom implementation with parallelism for both versions of NIOS II processor.

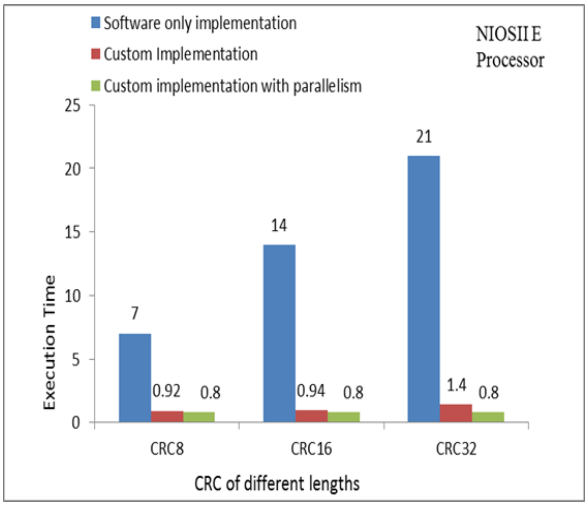


Figure 10. Comparison of execution time for NIOS II(E) processor

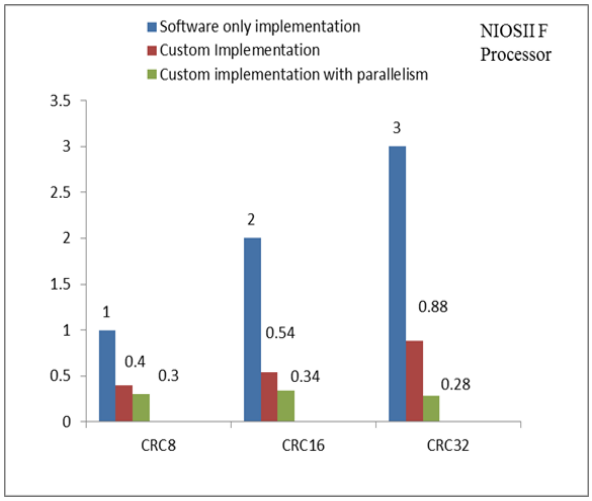


Figure 11. Comparison of execution time for NIOS II(F) processor

To estimate the increase in speed provided by customization, execution cycle count of custom

Volume 10, Issue 1
February 2021

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X



implementation is compared with that of custom implementation accelerated with parallelism.

$$Rise\ in\ speed = \frac{X}{Y}$$

$X$=Number of clock cycles required for custom implementation

$Y$=Number of clock cycles required for custom implementation with parallelism

**Table 3**. Speed Improvement

| Custom Instruction | Rise in Speed |
|---|---|
| CRC8 | 1.15 |
| CRC16 | 1.175 |
| CRC32 | 1.75 |

From Table 3 it is noticed that the speed of custom implementation is accelerated due to parallelism.

Performance Results:

The performance of different lengths of CRCs customization depends on processor clock cycles required for CRC custom implementation. The performance of both custom implementations for different lengths of CRCs can be calculated as,

$$Performance = \frac{Fn}{Cn}$$

$Fn$ = NIOS II processor clock rate (50MHz)

$Cn$ = Number of processor clock cycles required for given length of CRC Computation

From the computation, it is detected that overall performance decreases with rise in length of CRC, however, the performance of custom implementation accelerated with parallelism is much higher than the custom implementation with serial aspect.

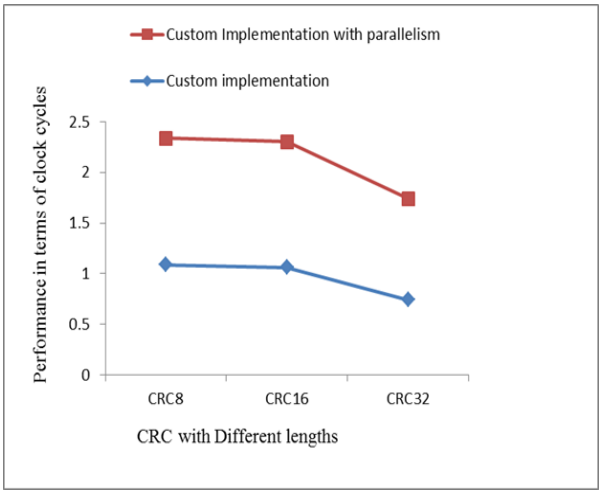


Figure 12. Performance comparison of custom implementation w.r.t. custom implementation with parallelism

From Fig. 12 it is noticed that the proposed acceleration in custom implementation with parallelism drastically improves the performance.

The synthesis details are given in Table 4. As illustrated in this table the custom implementation uses more FPGA area. Utilization of FPGA logic is related to its circuit size. It is reasonable to say that the circuit size of custom implementation is larger than the software only implementation. It is a trade-off between hardware utilization and performance.

**Table 4.** Hardware resource utilization

| | Custom Instruction | Total Logic elements | Total combinational functions |
|---|---|---|---|
| Software only implementation | CRC8 | 2021 | 1104 |
| | CRC16 | 2021 | 1104 |
| | CRC32 | 2021 | 1104 |
| Custom instruction implementation | CRC8 | 2200 | 1173 |
| | CRC16 | 2224 | 1187 |
| | CRC32 | 2217 | 1197 |

## 6. CONCLUSION

In this paper, the problem of fault tolerance and power consumption in WSN node has been resolved with the acceleration of CRC algorithm. The acceleration of CRC algorithm is addressed by custom instruction paradigm. The custom block CRC have been implemented using serial circuits called LFSRs and designed as custom instruction of NIOS II processor. It improves the performance drastically as compared to software-only implementation. Power reduction of 11% is achieved as compared to software only implementation in addition with 78 % of performance improvement in terms of clock cycles. Further, the custom implementation is accelerated and optimized with parallelism in CRC code hardware block. In comparison with an initial custom implementation, experiment and performance analysis shows that the parallelism gives the speed improvement by 11% along with power reduction by 6% w.r.t custom implementation. Overall performance improvement due to this custom design is 89% with 13% power saving. It would be very useful to reduce the overhead of processing unit in WSN nodes and adaptable to any WSN application.

## REFERENCES

[1] Atmel Corporation. (2007) ATmega 103L 8-bit AVR Low-Power Microcontroller. Tech. Report.

[2] Atmel Corporation. (2009) ATmega 128L 8-bit AVR Low-Power MCU. Tech. Report.

[3] Texas Instruments.(2009) MSP430 User Guide. Tech. Report.

[4] V. S. Patil, Y. B. Mane and S. Deshpande,"FPGA Based Power Saving Technique for Sensor Node in Wireless Sensor Network (WSN)", In Computational Intelligence in Sensor Networks (2019),pp. 385-404, Springer, Berlin, Heidelberg.

[5] N.A.Alrajeh, U. Marwat, B. Shams, and S.S.H. Shams, "Error correcting codes in wireless sensor networks: an energy perspective", Applied Mathematics & Information Sciences, (2015), 9(2), p.809.

[6] D.C.Feldmeier, "Fast software implementation of error detection codes", IEEE/ACM Transactions on networking, (1995). 3(6), pp.640-651.

[7] J.R. Engdahl, and D. Chung, "Fast parallel CRC implementation in software", In 14th International Conference on Control, Automation and Systems (ICCAS 2014) , October , pp. 546-550,

[8] Zhisheng, W.Z. L.Y.Y. "Design and implementation of CRC algorithm", Journal of Electronic Measurement Technology, ( 2007) -12.

[9] T. Zhang, and Q. Ding, "Design and Implementation of CRC Based on FPGA", In 2011 Second International Conference on Innovations in Bio-inspired Computing and Applications(2011), December -pp. 160-162, IEEE.

[10] W.M. El-Medany, "FPGA implementation of CRC with Error Correction'', In The Eighth International Conference on Wireless and Mobile Communications,(2012), ICWMC.

[11] C. Anton, L. Ionescu, I. Tutanescu, A. Mazare, and G. Serban, "FPGA-implemented CRC algorithm", In 2009 Applied Electronics , pp. 25-29, IEEE.

[12] C. Toal, K. McLaughlin, S. Sezer, and X. Yang, "Design and implementation of a field programmable CRC circuit architecture", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, ( 2009),17(8), pp.1142-1147.

[13] Tutorial: Linear feedback shift registers, EE times

[14] G.Albertengo, and R., Sisto, "Parallel CRC generation", IEEE Micro, 10(5), 1990. pp.63-71.

[15] G. Campobello, G. Patane, and M. Russo, "Parallel CRC realization", IEEE Transactions on Computers, (2003),52(10), pp.1312-1319.

[16] R.J.Glaise, "A two-step computation of cyclic redundancy code CRC-32 for ATM networks", IBM Journal of Research and Development, (1997), 41(6), pp.705-709.

[17] A.Perez, "Byte-wise CRC calculations" IEEE Micro, (1983), 3(3), pp.40-50.

[18] E. Stavinov, "A practical parallel CRC generation method", Circuit Cellar-The Magazine For Computer Applications, (2010.), 31(234), p.38.

## AUTHOR PROFILES

**Vilabha S. Patil** was born in Kolhapur, India, in 1982. He received the B.E. degree in Electronics engineering and M. Tech. in Electronics Technology from the Shivaji University, Kolhapur in 2003 and 2010 respectively. He is pursuing a Ph.D degree in Electronics Engineering under A.I.C.T.E.Q.I.P. scheme at Walchand College of Engineering (Govt. aided and an Autonomous Institute) affiliated to Shivaji University, Kolhapur, MH India. Since 2010, he has been an Assistant Professor in the Electronics Engineering Department, Rajarambapu Institute of Technology, Rajaramnagar, MH-India.

**Shraddha Deshpande** was born in Miraj, Maharashtra, India in 1962. She received her Bachelor Engineering (B.E.) and Masters (M.E.) degree in Electronics Engg. from Walchand College of Engineering, Sangli, Shivaji University, Kolhapur. She has received a Ph.D. degree in 2010 from Indian Institute of Technology, Bombay. MH India. She has 32 years of experience in teaching. Currently she is working as Professor at Electronics Dept., of Walchand College of Engg.,Sangli. She has received approximately 60 lakhs grant from AICTE in various heads like conference, FDP, RPS, MODROB. She has good number of publications in journal and conferences. (11-international Journals, 2-Book Chapters, 9-International conference, 2- National conference, 3-submitteed for international conference