International Journal of Electrical and Electronics Engineers
Volume 16, Issue No. 01, Jan-June 2024
ISSN (O) 2321-2055
ISSN (P) 2321 -2045

FAST FOURIER TRANSFORM AND ITS ARCHITECTURES

Krishna Kumar[1], Kabir Mendiratta[2], Priyanshu Sharma[3], Mr. Shailendra Bisariya[4], Mr. Mudit Saxena[5]

[1,2,3,4,5]*Department of Electronics and Communication Engineering, ABES Engineering College, Ghaziabad*

Abstract

The purpose of this paper is to provide a detailed review of the Fast Fourier Transform and Radix-2 computational algorithm with hardware architecture and for FFT and its comparison on the basis of LUTs used , Power and operating Frequency.

Introduction

The Fast Fourier Transform (FFT) method is a widely used algorithm for computing the discrete Fourier transform (DFT) of a sequence of numbers. The DFT is a mathematical technique that transforms a signal from the time domain to the frequency domain, which is useful in many fields such as signal processing, image processing, and audio processing. The basic principle of the FFT method is to exploit the properties of the DFT to reduce the number of computations required to compute it. The DFT of a sequence of N numbers requires O(N^2) operations, which can be computationally expensive for large values of N. However, the FFT method reduces the number of operations required to O(N log N), making it much faster than the direct computation of the DFT.

The FFT algorithm achieves this by dividing the input sequence of N numbers into smaller sub- sequences of size N/2, and recursively computing their DFTs using a divide-and- conquer approach. The smaller DFTs are then combined to compute the final DFT of the original sequence. The key insight behind the FFT algorithm is that the DFT of an N-point sequence can be expressed as the sum of two N/2-point DFTs, each of which can be computed using the FFT algorithm.

The FFT algorithm is widely used in many applications, including audio and video compression, digital signal processing, and scientific computing. It is a powerful tool foranalyzing and manipulating signals in the frequency domain, and has revolutionized many fields of science and engineering

There are several computational algorithms but we are going to focus on Radix-2 algorithm and for that first we have to understand the Digital Fourier Transform (DFT) .

Discrete Fourier transform (DFT)

Digital Fourier Transform (DFT) is a mathematical procedure used to determine the harmonics or frequency, content of a discrete signal sequence.

Mathematical equation for DFT,

$$x[k] = \sum_{n=0}^{N-1} x[n]e^{\frac{-j2\pi kn}{N}}$$

Eq…..1

X(k)=the kth DFT output, i.e. X(0), X(1)….etc. k= the index of the DFT output in the frequency domain

x(n) = the sequence of input samples, x(0), x(1),x(3) etc.

n = the time domain index of the input samples n,k=0,1,2,3,……,N-1

j=√-1

N = numbers of samples of the input sequence.

For each X(m), the output term is the product of the points for point product between the input sequence and a sinusoidal of form cos(ϴ)

- sinusoidal(ϴ). The frequency of each sinusoidal depends the number of samples N . the N separate DFT analysis frequencies are

$f_{analysis}(m) = mf_s/N$.............Eq.2

The FFT does similar Computation but in a fast manner by reducing redundant groups. Now we are going to discuss about **Radix -2** algorithm.

Radix-2 algorithm is a member of the family of Fast Fourier transform (FFT) algorithms. It computes the DFTs of the even-indexed inputs($x_0,x_2,...,x_{N-2}$) and of the odd-indexed inputs ($x_1,x_3,...,x_{n-1}$),and then add those two results to produce the DFT output.This results in reduction of complexity from O(N^2) to O(N logN).

Mathematical expression:

$$x[k] = \sum_{n=0}^{N-1} x[n]e^{\frac{-j2\pi kn}{N}}$$

$$x[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r]e^{\frac{-j2\pi k(2r)}{N}} \;+\; x[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r+1]e^{\frac{-j2\pi k(2r+1)}{N}}$$

$$x[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r]e^{\frac{-j2\pi k(2r)}{N}} \;+\; x[k] = e^{\frac{-j2\pi k}{N}} \sum_{r=0}^{\frac{N}{2}-1} x[2r+1]e^{\frac{-j2\pi k(2r)}{N}}$$

$$x[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r]e^{\frac{-j2\pi k(r)}{N/2}} \;+\; x[k] = e^{\frac{-j2\pi k}{N}} \sum_{r=0}^{\frac{N}{2}-1} x[2r+1]e^{\frac{-j2\pi k(r)}{N/2}}$$

$$x[k] = x_{even}[k] \;+\; e^{\frac{-j2\pi k}{N}} x_{odd}[k]$$

the upper equation can be written as

X(m) = {even samples} + e$^{-j2\pi k/N}$ ∗ {odd samples}

here e$^{-j2\pi k/N}$ is called **Twiddle Factor** and can be represented by ω^k .

upper equations concludes that there is no need to perform any cosine or sine multiplication to get
N
X(k+N/2. It can be done by changing the sign of the twiddle factor m^n and use the result of the two

summation from X(m) to get X(k+n/2).

in simplified terms equations

^{m}**X(k) = A(k)+** m^{k}_{N} *** B(k)** …… Eq.3

X(k+N/2) = A(k)- m^{k}_{N} *** B(k)** …… Eq.4

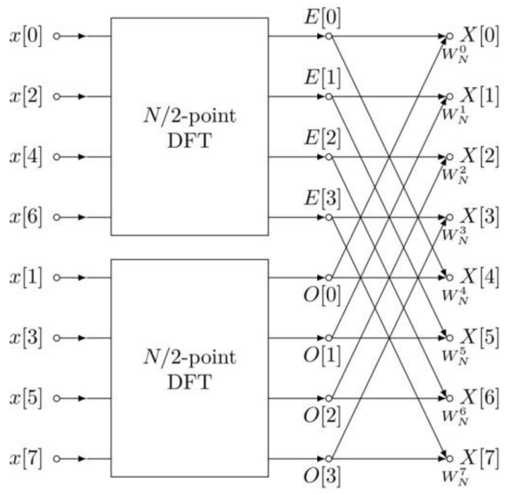

Fig .1

the upper diagram shows how the computation flows in Decimation in time for 8- point FFT and this diagram also called BUTTERFLY diagram.

Decimation is basically means drastic reduction in the strength or effectiveness of something .In FFT there are two type of decimation exist the first is Decimation in time (DIT) and second is Decimation in Frequency (DIF).. If data re-orderd before FFT computation than this reordering is called Decimation in time and If the data gets reorderd after the computation of FFT than this reordering is called Decimation in Frequency.Below is the calculation scheme for a DIT 8 point FFT. Note that the output in the correct order, while the x(n) is pre-ordered.

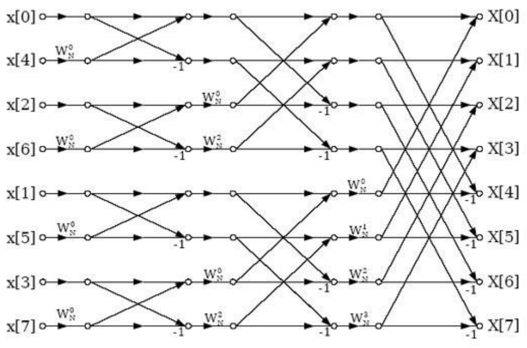

Fig .2 .

And below is the calculation scheme for a DIF 8 point FFT. Note that the x[k] is output in the incorrect order, while the x(n) in ordered manner.

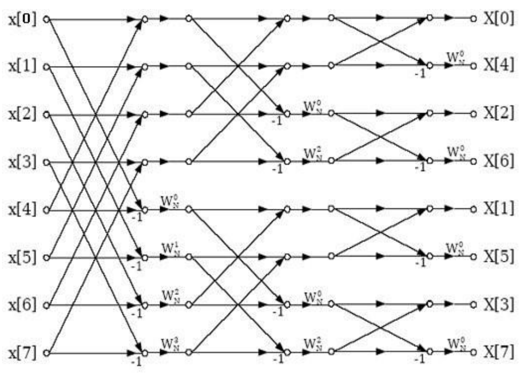

Fig.3

Field Programable Gate Array (FPGA)

FPGA stands for Field-Programmable Gate Array. It is a integrated circuit that can be reconfigured or programmed after manufacturing, making it highly flexible and customizable for various applications.

A matrix of programmable logic blocks coupled by programmable interconnects makes up FPGAs. These logic building blocks can be set up to carry out particular tasks or create specific digital circuits. The routing of signals between the logic blocks is made possible by the interconnects, allowing for intricate connections and data flow.

A hardware description language (HDL), such as VHDL or Verilog, is used to configure an FPGA. The intended circuit or system functionality is described in the HDL code. In order to specify the precise circuitry and interconnections, this code is then synthesized into a configuration bitstream and fed into the FPGA

Literature Review

Author Tarek Belabed [1] has converted the twiddle factor into an integer value for computation and to convert twiddle factor into integer value, the exponential coefficients were multiplied by a fixed factor which is equal to 2^m. Only the integer part is used, and at the output's end the result coefficients gets divided by the same coefficient (2^m) that previously used to convert input exponential coefficients into integer. the factor is chosen in multiples of 2 ,because in digital system multiply by 2 can be done by left shift and divide by 2 is done by right shift . This operation is very fast and less expensive for an FPGA.

To mitigate errors arising from the integerization of twiddle factors, it is recommended to meticulously select the value 'm.' and further to optimize the multiplication strategy involves converting the multiplication operation into a series of left shifts. Therefore, the essence of this approach lies in transforming the multiplication process into a addition of left shifts, ensuring precision while minimizing computational complexity.

left shift by K = multiply 2^K

This operation can only be done by choosing power coefficients of 2(2, 4, 8, 16, etc.). The idea here is to equate the coefficients in equivalence of power of two, which their sum is almost equal to the twiddle factor.

The article also proposes an architecture that proposes to minimize read and write times by dividing the RAM into two parts, each with its own address bus, one for recording the result of the real part of the FFT and the other for the imaginary part. Writing and reading to both RAM memories occurs simultaneously. The read and write counter do addressing of the two RAMs .The demultiplexer and multiplexer unit is used to send and receive the data from two RAMs and to the FFT.

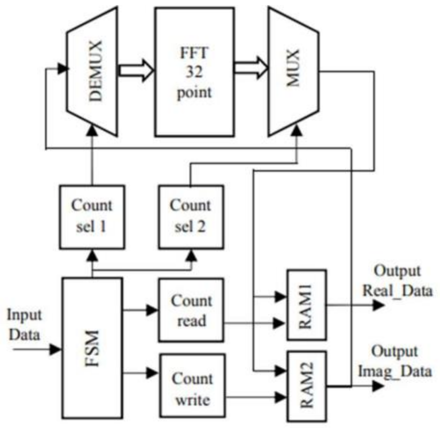

Fig 4. Block diagram of proposed architecture

[2] In paper Kanders, Mario Garrido and others presented The 1-Million Point Fast Fourier Transform has been successfully implemented on a single FPGA, utilizing only the onboard memory and without requiring interconnected FPGAs. This achievement was made possible through the use of a pipelined Single-Delay Feedback (SDF) FFT architecture, which has five interconnected stages .Every stage has a radix-2 butterfly and a rotator. The outputs of the radix -2 butterfly are connected to first-in- first-out (FIFO) buffers in feedback loop. This arrangement enables the system to handle single-path input and process one sample at every clock cycle efficiently.

The FFT algorithm significantly influences the placement of the rotator in the architecture. Given that the W1M rotator is the most complex among all rotators, it is advisable to position it towards the end of the architecture. This is because the numerous angles in the rotator necessitate a larger data word length for accurate calculations. As a result, it is suggested to delay the increase in word length until later in the architecture to minimize the number of stages having large word lengths. This decision will impact the size of the buffers in the SDF stages, as each stage in the FFT reduces buffer size by half. By strategically placing the W1M rotator later in the architecture, the amount of required memory can be reduced.The W1M rotator has been specifically designed to obtain exceptional resolution, and we have carefully selected word lengths throughout the FFT stages to ensure precise computations and a compact data memory size. This comprehensive approach marks the first time these challenges have been thoroughly addressed in a large FFT setting. "Consequently, the suggested design not only entails minimal space, but also attains exceptional precision and a reasonable power usage."

The report shows that the proposed structure fits on a single FPGA and having a large amount of resources left. The block random access memory [BRAM] has highest utilization among other blocks and this is happened because the FFT is processing very large number of points in this proposed architecture. The W16 rotators in the architecture are implemented as shift-and-add using LUTs, which is hardware efficient due having small size.

As a result, the proposed approach achieves the highest performance above all conventional million point FFT. Furthermore, the proposed design obtains a high SQNR and a power consumption of 3.436 W.

[3] Authors Pramod Kumar & others presented an area and energy-efficient architecture for radix-2 DIT for integer valued FFT. The proposed configuration for point in-place Decimation in Time Real Fast Fourier Transform (DIT RFFT) computation is illustrated in Figure 5. This structure comprises an Arithmetic Unit (AU), a Data Storage Unit (DSU), a Twiddle-Factor Storage Unit (TFSU), and a Control Unit (CU). In every cycle, the arithmetic unit receives a packet having four samples from the data storage unit and two twiddle factors from the twiddle factor storage Unit. Over the first 12 clock cycles within a 16-clock cycle period, it executes a 4-point Butterfly operation in each cycle, generating a 4- point intermediate-output data block which is written by into data storage unit. During the final 4 clock cycles of the 16-clock cycle , the FFT coefficients are obtained as the output of the AU.

The proposed structure having significantly less area-delay product and less energy for each sample than the existing folded structures for RFFT.

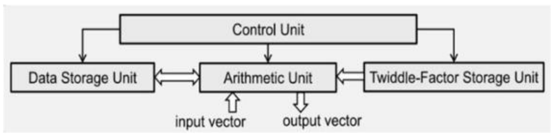

Fig .5. structure of DIT RFFT

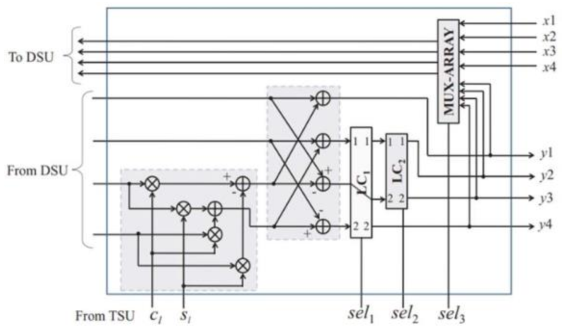

Fig.6. Structure of arithmetic unit (AU)

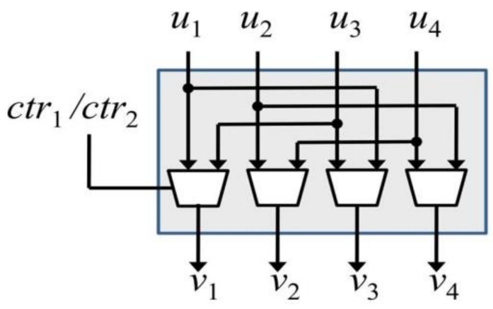

Fig.7. Structure of data-selector (DS)

[4] Author Mario Garrido and Pedro Malagon presents a CM FFT that uses constant multipliers for evaluating the rotations. This achievement is realized through the grouping of data that undergo rotations by identical angles, directing them along a common path within the architecture. The proposed approach serves a dual purpose: firstly, it dissects the rotations at the FFT stages into a combination of constant rotators, and secondly, it ensures that data requiring identical constant rotations follow the same trajectory within the architecture. To facilitate this, supplementary shuffling circuits are incorporated to ensure that each constant rotator gets the relevant data. The deployment of constant multipliers at virtually all stages of the architecture significantly mitigates the complexity associated with rotators. Contrary to W8 and W16 rotators, constant rotators exclusively perform rotations by a single constant angle, eliminating the need for multiplexers for selection among different angles. Even after choosing the constant multiplier, it doesn't guarantee the reduction in resultant area of FFT, despite having less

complexity, the architecture incorporates greater number of shuffling circuits.
the proposed FFT architecture might not good as area wise but it manifests in its throughput. This is attributed to the lower level of complexity of the rotators, which enables deep pipelines which significantly shortens the critical path of the architecture. As a result, there is an increase in clock frequency and throughput.

However, the complex multiplier quadratically gets increases as the number of stages increases. Thus For the large FFTs, good alternatives are radix-2^4 or radix-2^5. Experimental results of a 1024-point 4-parallel radix-2^4 CM FFT provide throughput having 20% higher than the highest throughput among previously developed 1024-point 4-parallel pipelined architectures on FPGA .

[5] Authors Kevin Bowlyn and Sena Hounsinou presented a DA-CBNS based multiplier-less approach for computing Radix-2 FFT. Every butterfly structure in the DA-CBNS approach only requires one complex multiplication, one addition, and two complex additions/subtractions. Achieving this

involves initially converting each input of the butterfly structure to the (-i+j) CBNS. Subsequently, multiplier-less Differential Arithmetic (DA) implementation is used for Multiplication circuit. here authors considered DA structure based in ROM having no dedicated multiplierer and having minimal memory utilization.

In a typical DA structure, binary inputs are customary, and it uses an arithmetic right shift during the accumulation phase for multiplication to preserve the sign of the product. However, the design by Kevin Bowlyn and Sena Hounsinou deviates by utilizing a logical shifter in place of an arithmetic shifter, as the latter lacks a sign bit. Another enhancement involves storing constant twiddle factor using a non-Look-Up Table (LUT) ROM component instead of a ROM memory bank. Due to this modification, the DA-LUT was implemented using logic gates, which led to a small increase in the number of gates. But this modification also results in a 100% decrease in memory usage. All of these changes added up to our suggested multiplier unit, which comprised of a CBNS adder unit, a register that retain the CBNS adder output, a Parallel In Serial Out (PISO) shift register, a non-LUT ROM base to store the twiddle factor required to compute the DA-CBNS FFT structure, and a logical right-shift unit for the accumulation phase.

Throughout each stage of the butterfly structure, these steps are iteratively performed, maintaining the (-1+j) base for the initial input operands, resulting in a product also in the same base. The utilization of CBNS contributes to an overall better performance compared to the

existing Radix-2 FFT structure. Despite the increased number of addition operations due to these modifications, the proposed design exhibits a performance improvement comparison with the FFT implementations having dedicated multipliers, DA-only, or CBNS alone.This is slightly similar as Tarek Belabed, Sabeur Jemmali , Chorkri Souani [1] has proposed in their paper.

[6] Authors Ghattas Akkad & others presented a study dedicated to accelerating signal processing algorithms using High-Level Synthesis (HLS) tools for digital communication applications. In contrast to rudimentary Hardware Description Language (HDL) implementations, HLS brings forth flexibility in adopting diverse directives and constraints to achieve requisite performance while upholding speed, resource utilization, and development efficiency. Notably, HDL provides the possibility of inferring particular architectures using a structural methodology, which makes it easier to implement vendor-specific core modules and customized intellectual property (IP).

The research delves into the effects of variation in frequency, in addition to the application of various coding architectures and styles, input delay, latency, and speed for both HLS and HDL methodologies. Although HDL design approaches produce low-level, optimized implementations by following stringent coding guidelines, HLS tools produce results that are almost identical.

Furthermore, HLS enables the use of various program directives or optimization techniques dependent on compilers, eliminating the necessity for extensive recoding. This comparative exploration underscores the adaptability and efficiency of HLS tools in signal processing algorithm acceleration for digital communication purposes .

Authors Prasanna Kumar Godi, Battula Tirumala Krishna & others introduces a multiplier-free FFT design that innovatively replaces traditional twiddle factors with the UMA method's input. This UMA (Unified Multiplier-less Architecture) method operates through shift and addition operations, eliminating the need for division. The design encompasses six stages of a 64- point FFT, as depicted in Figure 8. Both the real and imaginary parts serve as input values for the FFT, with clock values provided at each stage of the implementation. The input data for the FFT is 16-bit in length.

During the NWT (Number of Wasted Terms) computation, the design calculates power- of-2 operations using carefully crafted shift operators. Control signals, such as shift ctrl0 and shift ctrl1, manage these shift operations, transmitting the necessary twiddle factors. Remarkably, the UMA scheme's output replaces the conventional twiddle factor generation in the FFT. The UMA method executes shift and add operations to produce the required output, obviating the need for complex multiplier blocks typically used in twiddle factor multiplication. Consequently, the UMA- based FFT design operates without the requirement for complex multipliers, offering a streamlined and efficient approach to FFT computation.

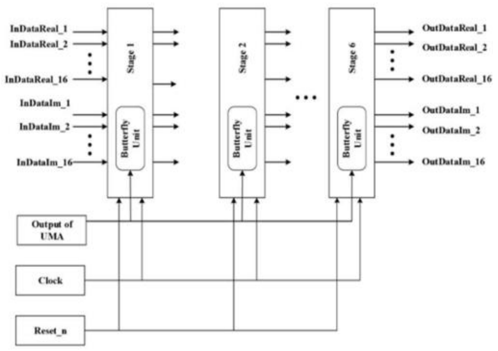

Fig.8.Six stage of radix-2 64 point FFT based on UMA

Comparison Result

This section presents some comparison between the FFT algorithm that is reviewed in upper section. The comparison is done on the basis of hardware required, power and performance.

The comparison is done for Radix-2 FFT.

Here we can see that all the architectures has some kind of drawback in terms of power, speed or LUTs so we cannot suggest any architecture for universal use and all the design are tested on Different FPGA with different configurations so it is hard to point out one.

When speed is concern we would go with the structure [2] because of its Single-Delay Feedback system which provide ability to architecture to handle single-path input and process one sample per clock cycle efficiently which result is higher speed of computation. Although seem like its has very high power consumption actually it is not, it is happens because in this architecture used 1-million points for computation.

But if power is concern structure [3] and structure [7] turns out to be more efficient than others. this is happening due to the multiplier-less approach which both are exhibiting but it comes at the cost of speed. Which make them not suitable for some applications such as astronomical signal processing and complex image processing where data size is very large.

Conclusion

In this work we have presented several FFT computational algorithm with some hardware implementation and optimization. Some of proposed architecture align toward less computation and some of has balance between hardware and its computational power but the structure [2] comes out to be best. Due to these characteristics, the design is appropriate for DSP applications where performance and efficiency are crucial, like imaging in the medical domai.

NAME	LUTS	Frequency (MHz)	Power(w)
Structure of {3] 32 -point	N.A	N.A	0.0029
Structure of [7] for 32-point	998	400	0.003358
Structure of [1] for 32- point	22179	N.A	N.A
Structure of [2]	16827	233	3.436
Structure of [4] for 1024-point	V6-2576 V7-2631	V6-475 V7- 680	N.A 1.68
Structure of [5] 16-point	38072	75.001	0.65244
Structure of [6] 8-point	1680	196.539	N.A

References

[1] Tarek Belabed, Sabeur Jemmali , Chorkri Souani "FFT implementation and optimization on FPGA" 4th International Conference on Advanced Technologies For Signal and Image Processing - ATSIP 2018 March 21-24, 2018

[2] Hans Kanders, Tobias Mellqvist, Mario Garrido, Kent Palmkvist, and Oscar Gustafsson, "A 1-Million Point FFT on a single FPGA" Ieee transactions on circuits and systems–i: regular papers, vol. 66, no. 10, October 2019

[3] Pramod Kumar Meher, Basant Kumar Mohanty Sujit Kumar Patel, Soumya Ganguly, and Thambipillai Srikanthan, "Efficient VLSI Architecture for decimation-in-Time Fast Fourier Transform of real-Valued Data", Ieee transactions on circuits and systems—I: regular papers, vol. 62, no. 12, December 2015

[4] Mario Garrido and Pedro Malagon," The Constant Multiplier FFT", ieee transactions on circuits and systems–I: regular papers, (Volume: 68,Issue:1 January 2021)

[5] Kevin Bowlyn and Sena Hounsinou "An Improved Distributed Multiplier -less Approach for Radix-2 FFT", ieee letters of the computer society, vol. 3, no. 2, july-december 2020

[6] Ghattas Akkad, Ali Mansour , Bachar ElHassan , Frederic Le Roy , Mohamad Najem "FFT Radix-2 and Radix-4 FPGA Acceleration Techniques Using HLS and HDL for Digital Communication Systems", 2018 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET), 06 January 2019.

[7] Prasanna Kumar Godi, Battula Tirumala Krishna , Pushpa Kotipalli, "Design optimisation of multiplier-free parallel pipelined FFT on field programmable gate array", IET Circuits, Devices & Systems ISSN 1751-858X Received on 27th November 2019 Revised 2nd July 2020 Accepted on 6th July 2020 E-First on 22nd October 2020