DOI: https://doi.org/10.67897/njma.2022.gc4d3wil



Nigerian Journal of Mathematics and Applications
Volume **32**, (2022), 136 – 151. *Printed by Unilorin press*
©*Nig. J. Math. Appl.* *http : //www.njmaman.com*

Systolic Array Residue Number System Based Smith-Waterman Algorithm

Hassan Kehinde Bello

Abstract

Smith-waterman algorithm (SWA) is widely used in computational biology. It is regarded as the most accurate sequence alignment algorithm on state-of-the-art (SOA). However, the algorithm is devoid of highspeed in term of performance. Evidence showed that various platforms have been used to implement the algorithm in order to improve the poor speed of operation, such as systolic array (SA) implemented on Field Programmable Gate Array (FPGA), acceleration on CPU-GPU architecture, acceleration based on residue number system (RNS) and so on. Evidence on SOA also showed that implementation of the algorithm on FPGA platform recorded a better result than any other platform. In this paper, Systolic array is proposed on SWA taking advantages of inherent and unique properties of RNS and implemented on FPGA (Spartan-III, 64-Bit version (Xilinx family)). The metrics used for evaluation was processing time. The results were finally compared with existing SOA systems. The SA method vs RNS based implementation on FPGA gave a credible result with 372.4 Giga Cell Update Per Second (GCUPS) with 875 PE.

Received: 10/07/2022, Accepted: 25/07/2022, Revised: 05/08/2022. * Corresponding author.
2015 *Mathematics Subject Classification.* 68W01 & 05B15.
Keywords and phrases. Computational biology, Systolic array, Residue number system, Giga cell update per second, Smith-Waterman algorithm
[1]Department of Computer Science, Federal Polytechnic, Offa, Kwara State, Nigeria
E-mails of the corresponding author: hassan.bello@fedpoffaonline.edu.ng; belohk72@gmail.com
ORCID of the corresponding author: 0000-0001-7488-8741

1. Introduction

The importance of processing speed of algorithm in computational biology can not be over emphasized. SWA, a popular sequence alignment algorithm in bioinformatics which is relatively slower due to number of computations involved in the search [5]. It is therefore, desirable to optimize its performance with regard to computational time.
Maximum alignment between any two sequences is given by SWA below.

$$M(i,j) = \begin{cases} 0 \\ M(i-1,j-1) + S(x_i, y_j) \ match/mismatch \\ Max \ M(i-1,j) + g \\ M(i,j-1) + g \end{cases} \tag{1}$$

At $M(0,0) = 0$, $M(0,j) = g \times j$ and $M(i,0) = g \times i$, where $1 \leq i \leq n$, $1 \leq j \leq m$, g is the penalty for gap insertion in any of the sequence and $M(i,j)$ is the score for match or mismatch, depending upon whether $X(i) = Y(j)$ or $X(i) \neq Y(j)$.
Systolic array is the rectangular arrangement of processors in an array ([11] & [12]) so that data can flow synchronously between them. It is used to determine the trace-back and simplifies alignment sequence by aligning one input sequence against different sequences in the database at the same time. In computational biology, the two types of alignments are shown in Figure 1. SW is used along with systolic array to compute the best local alignment of two sequences while RNS was also used to enhance the acceleration.
This paper focuses on designing a smart systolic cell using Residue Number System and carry out implementation on FPGA.

1.1. Technical Background and Related Work.

The building block of DNA are nucleotides which are adenine, cytosine, guanine and thymine abbreviated as A, C, G and T respectively. Series of nucleotide collection are known as genome which are stored as biological data in DNA database. The two popular alignment methods on SOA are Local and Global alignments. Local alignment (Figure 1a) searches some residue of DNA sequences [16].

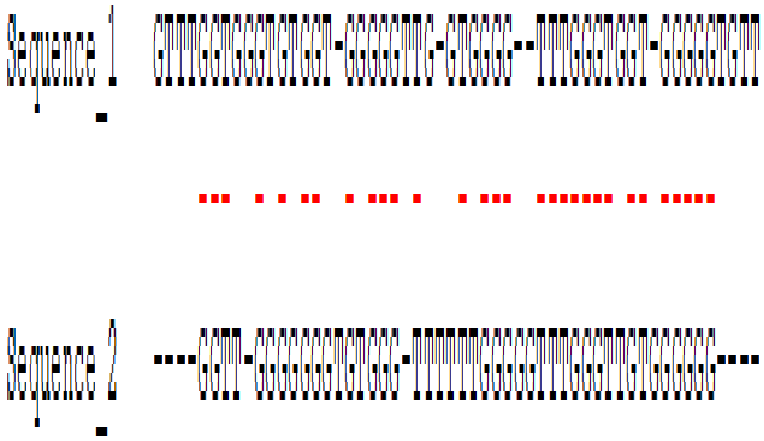


Figure 1a: Local alignment

Global alignment (Figure 1b) searches all residues in the sequences to obtain the similarity between the DNA sequences.

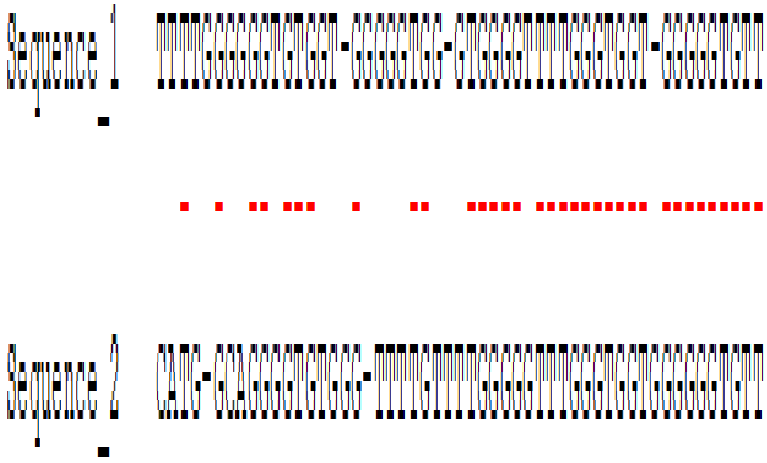


Figure 1b: Global alignment

Smith-Waterman algorithm (SWA) is a well-known dynamic algorithm popularly used in computational biology and considered to be the best local sequence alignment algorithm on SOA, [2]. Its main challenge is the low-speed processing. Several approaches on SOA have been used to address the issue such as systolic array application, Recursive variable expansion application, RNS application, FPGA and so on. This approach intends to use Systolic array on SWA and apply RNS as accelerator then implement on FPGA. In RNS the addition of two large numbers can be computed by adding their corresponding residue values in parallel, this approach is faster with less hardware requirement and free of power consumption than conventional arithmetic [9]. Also in systolic array architecture, parallel computation is used to determine each of the array elements. Implementing these two methods on FPGA using SWA is our objective, to reduce the time and space complexity in SWA. The algorithms address performance or execution time; therefore, it is a trade-off between time and performance ([3] & [4]).

In addition, SWA is a widely used algorithm in computational biology, many researchers attempted to improve the speed of the algorithm on various platforms. [8] used systolic array architecture to accelerate Smith-Waterman Algorithm with cell design. The result of comparing implementation of systolic array architecture with software implementation indicates that systolic array architecture speed up to 652 times the software implementation, which is above twice the best case on the-state-of-the-art. The proposal used 3 adders, 7 logic units per cell, 4 comparators. It was considered to be efficient in resource utilization because it used between 3.69 and 6.36 less resource than other designs in literature.

[7] presented a paper on the efficient design of residue to binary converter, using CRT on the moduli $2^n, 2^{2n}+1, 2^n+1, 2^n-1$. A comparative analysis was carried out between the previous and the proposed design. The results showed that the proposed design is having 15.02% time complexity less than the previous design. Also in [13], systolic array architecture and pipe-lining techniques to achieve a high-speed performance design. The comparative results analysis confirmed that the architecture is 1.2 times faster compare to other FPGA designs.

2. Methodology

The method applied in the proposed system are itemized below:
SWA processing element cell design; SWA-RNS accelerator design; Forward converter design for the three moduli; Processor design and Reverse converter design.

2.1. **Step one: SWA Processing Element Cell Design.**

In designing systolic cell, Processing Element (PE) is designed (Figure 4) which is the building block. Each PE refers to number of hardware required to calculate the content of a cell. It calculates the alignment of elements in a matrix simultaneously, when systolic array is introduced to SWA, the full column is computed at the end of the final

pass of the entire operation. The corresponding block diagram is illustrated in Figure 5.
The performance of our systolic array is highly depended on how better our cell is designed. Figure 5 shows the design of $M_{i,j}$ matrix in a block diagram.
Comparison between two sequences (Q_r and D_b) of DNA nucleotides to be aligned (Figure 5 and Figure 7) determine whether the value of match or mismatch will be added (adder1) to the value in the diagonal cell ($M_{i-1,j-1}$). The output of the addition is compared with 0 and the higher is given as Max1. Gap is added to both values in the left cell and upper cell (adder2 and adder3 respectively) and compares the results of the two additions where the higher one is Max2. Then Max1 is converted to RNS followed by Max2. The maximum of the two values is the optimal value for $M_{i,j}$. The block diagram in Figure 6 was implemented on FPGA.

2.2. **Step two: Design of SWA-RNS based accelerator.** The moduli $2^{2n+1}-1, 2^{n-1}, 2^{2n}-1$ set is proposed to be used in the design of SWA-RNS based accelerator to carry out addition of processing element (PE) in SWA and implement in FPGA to accelerate the filling of all cells in the PE. Since RNS does not involve in carry propagation, the results of addition, subtraction and multiplication operations are very fast. This has been implemented in Digital Signal Processing (DSP), Cryptography, Image processing and so on. In the light of this, RNS takes full advantages of its carry free properties on SW algorithm to fill up the cells in the table. This process basically goes through three stages (Figure 2) as clearly itemized below:
1. The conversion of conventional number to residue number (FC) which produces a non-weighted integer number ([1], [14] & [15]).
2. Modular addition; this involves using of the proposed moduli set to carry out addition on the variables of SW algorithm.
3. Reverse conversion which finally returns the numbers in RNS to binary/decimal number.

2.3. **Step three: Design of forward converter (FC) for moduli set $\{2^{2n+1}-1, 2^{n-1}, 2^{2n}-1\}$.** If X is a binary number and n is even integer and our moduli set $m = \{2^{2n+1}-1,\ 2^{n-1},\ 2^{2n}-1\}$ such that $m_1 = 2^{2n+1}-1$, $m_2 = 2^{n-1}$ and $m_3 = 2^{2n}-1$.
$M(m_1^* m_2^* m_3)$ gives the dynamic range value as $2^{5n}-2^{3n}-2^{3n-1}+2^{n-1}$ for integer X (5n-bit), where the residues set $\{r_1, r_2$ and $r_3\}$ is exclusively defined within the range $[0, 2^{5n}-2^{3n}-2^{3n-1}+2^{n-1}-1]$ with $r_i = |X|_{mi}$.

$$(2) \qquad X = x_{5n-1}x_{5n-2}x_{5n-3}\cdots x_{3n-1}x_{3n-2}x_{3n-3}\cdots x_{2n-1}\cdots x_n x_{n-1}\cdots x_0$$

$$X = \sum_{j=0}^{5n-1} |x_j 2^j| \tag{3}$$

$$|X|_m = |\sum_{j=0}^{5n-1} |x_j 2^j|_m| \tag{4}$$

$$|X|_{mi} = |\sum_{j=0}^{5n-1} |x_j 2^j|_{mi}| \tag{5}$$

$X = x_{5n-1}x_{5n-2}x_{5n-3}\cdots x_0$.
In order to determine the residue r_1, r_2 and r_3, X is partitioned into three blocks P_1, P_2 and P_3.

$$X = \begin{cases} x_3, 5_n - 1x_3, 5_n - 2x_3, 5_n - 3x_3, \cdots, 3n :\equiv 5n \\ x_2, 3_n - 1x_2, 3_n - 2x_2, \cdots, 2_n :\equiv 3n \\ x_1, 2_n - 1x_1, n_2x_1, n_3x_1, \cdots, 0 :\equiv 2n \end{cases} \tag{6}$$

$$\begin{cases} P_1 = \sum_{j=3n}^{5n-1} |x_j 2^{j-3n}| \\ P_2 = \sum_{j=2n}^{3n-1} |x_j 2^{j-2n}| \\ P_3 = \sum_{j=0}^{2n-1} |x_j 2^j| \end{cases} \tag{7}$$

$X = P_1 2^{3n} + P_2 2^{2n} + P_3$
$|X|_m = |P_1 2^{3n} + P_2 2^{2n} + P_3|_m$
$|X|_{RNS(2^{2n+1}-1|2^{n-1}|2^{2n}-1)} = |P_1 2^{3n} + P_2 2^{2n} + P_3 2^{2n+1} - 1|, 2^{n-1}, 2^{2n} - 1$
$= |P_1 2^{3n} + P_2 2^{2n} + P_3 2^{2n+1}| + |P_1 2^{3n} + P_2 2^{2n} + P_3 2^{n-1}| + |P_1 2^{3n} + P_2 2^{2n} + P_3 2^{2n} - 1|$

2.3.1. *Forward Converter for moduli* $2^{2n+1} - 1$*:* For the bit number

$2^{2n+1} - 1$ is $2n + 1$ bit number which can also be represented by
$r_1 = x_{1,2n}x_{1,2n-1}x_{1,2n-2}x_{1,2n-3}\cdots x_{1,0} \equiv 2n + 1$
$r_1 = |X|_m = |P_1 2^{3n} + P_2 2^{2n} + P_3|_m$
$= |X|_{2^{2n+1}-1} = |P_1 2^{3n} + P_2 2^{2n} + P_3|_{2^{2n+1}-1}$
$= |P_1 2^{3n}|_{2^{2n+1}-1} + |P_2 2^{2n}|_{2^{2n+1}-1} + |P_3 2^{2n+1} - 1|_{2^{2n+1}-1}$

$$= P_1(2^{n-1}) + P_2 + P_3 \tag{8}$$

2.3.2. *Forward Converter for moduli* 2^{n-1}. The residue r_2 with $m_2 = 2^{n-1}$ is very simple to compute, it is $(n-1)$ left shift from last digit of binary number X.

2.3.3. *Forward Converter for moduli* 2^{2n-1}. For the bit number $2^{2n}-1$ is a $2n$ bit number which can also be represented by

$$r_3 = x_{3,2n-2}x_{3,2n-2}x_{3,2n-3}\cdots x_{3,0} \equiv 2n-1$$

$$r_3 = |X|_{2^{2n-1}} = |P_1 2^{3n} + P_2 2^{2n} + P_3 2^{2n-1}|_{2^{2n-1}}$$

$$= |P_1 2^{3n}|_{2^{2n-1}} + |P_2 2^{2n}|_{2^{2n-1}} + |P_3 2^{n}|_{2^{2n-1}}$$

$$= P_1(2^{n}) + P_2 + P_3 \tag{9}$$

where r_1, r_2 and r_3 are the residues of $|X|_{2^{2n+1}-1}$, $|X|_{2^{n-1}}$ and $|X|_{2^{2n}-1}$ respectively.
Forward Conversion with $n = 2$ in equation (3) gives the values of m_1, m_2 and m_3 as $31, 2$ and 15 respectively. The dynamic range (DR) will be 930 and all the values for match, mis-match and gap in equation (2) will be within 0 to 929 inclusive. It can be used to represent 10 bits where n is even. Implementation of the forward conversion was carried out on Xilinx (Spartan-III: XC3S250) FPGA device.
Forward conversion of the weighted number 42 with respect to the moduli set $\{2^{2n+1}\text{-}1,\ 2^{n-1},\ 2^{2n}\text{-}1\}$ with $n = 2$ is computed as follows:

$$r_1 = 42|_{2^{2n+1}-1} = (101010)_2|_{2^5-1}$$

$$= 42|_2^5 - 1 \Rightarrow (101010)_2|_2^5 - 1 \ \textit{partition I into three blocks of 5 bit}$$

$$= (000000000101010)_2|_{31}$$

$$= 0 + 1 + 10|_{31}$$

$$= 11|_{31} = 11$$

$$r_2 = 42|_{2^n-1}$$

which is $(n-1)$ left shift from last digit of binary number $101010 = 0$

$$r_3 = 42|_{2^{2n}-1} = (101010)_2|_{2^4-1}$$

$$= (101010)_2|_{15}$$

Partitioned into 3 blocks of 4 bits

$$42|_{2^4-1} = (101010)2|_{15}$$

$$= (000000101010)_2|_{15}$$

$$= 0 + 2 + 10|_{15}$$

$$= 12|_{15} = 12$$

Therefore, $11, 0, 12$ are the residue number representation of the weighted number 42 with respect to moduli $31, 2, 15$ respectively.

2.4. **Step four: Processor Design.** The next stage is the residue number system processor design for the moduli set $2^{2n+1}-1$, 2^{n-1}, $2^{2n}-1$. The RTL block diagram is displayed in Figure 2. RNS moduli addition is carried out using $\{\mathbf{2^{2n+1}\text{-}1,\ 2^{n-1},\ 2^{2n}\text{-}1}\}$ on the SW algorithm.

2.4.1. *Modulo-M Adder:* The addition of two residues X and Y in Modulo-M where $(0 \leq X, Y < M)$ is defined as:

$$SUM = |X+Y|_M = \begin{cases} \mathrm{X+Y,\ if\quad X\ +\ Y\ <\ M} \\ \mathrm{X+Y\ -\ M,\ \ if\ X+Y>M} \\ \mathrm{0,\ \ if\ X+Y=M} \end{cases}$$

where M is either 31, 2 15

Modulo 31 adder:

$$SUM = |X+Y|_{31} = \begin{cases} \mathrm{X+Y,\ if\quad X\ +\ Y\ <\ 31} \\ \mathrm{X+Y\ -\ M,\ \ if\ X+Y>31} \\ \mathrm{0,\ \ if\ X+Y=31} \end{cases}$$

Modulo 2 adder:

$$SUM = |X+Y|_2 = \begin{cases} \mathrm{X+Y,\ if\quad X\ +\ Y\ <\ 2} \\ \mathrm{X+Y\ -\ M,\ \ if\ X+Y>2} \\ \mathrm{0,\ \ if\ X+Y=2} \end{cases}$$

Modulo 15 adder:

$$SUM = |X+Y|_{15} = \begin{cases} \mathrm{X+Y,\ if\ \ X\ +\ Y\ <\ 15} \\ \mathrm{X+Y\ -\ M,\ \ if\ X+Y>15} \\ \mathrm{0,\ if\ X+Y=15} \end{cases}$$

2.5. **Step five: Reverse Converter (RC) Design.** This is the last stage of the design, it converts residue-based number to binary based number, which is based on $\{\mathbf{2^{2n+1}\text{-}1,\ 2^{n-1},\ 2^{2n}\text{-}1}\}$ moduli set. It was implemented on FPGA using Xilinx (Spartan-III; XC3S250) FPGA, which was used to convert the residue $(11, 0, 12)$ to the appropriate binary number.

2.5.1. *SWA-RNS based architecture.* The architecture involves three stages (Figure 2). At the FC stage, moduli conversion of all variables in SW algorithm is performed and at the processing stage, moduli addition takes place. The addition of gap value to the values in the upper cell and left side cell; addition of match or mismatch values to the value in the diagonal. RC is the final stage; it converts from RNS to decimal number.

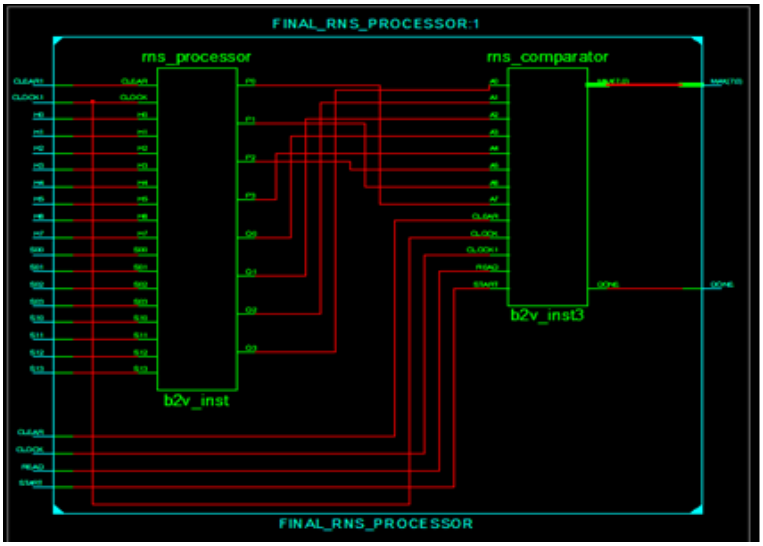


Figure 2: RTL Internal block diagram of SWA-RNS based Processor

2.5.2. *Linear Smith-Waterman Algorithm Computing Array (LSWCA).* Figure 3 illustrates step in parallel computing of anti-diagonal in matrix M with a linear processing element array. It starts with clock T_1. After initialization, the query sequence (Q_r) is sent to the PE array, where each PE holds a character in its register. The first character "A" in the database is sent into PE1 at clock of T_2 where it compares the input character with its content and computes the alignment score. The previous entry is sent to PE2 at T_3 clock. The characters sequence in database (D_b) flow through the PEs step by step. At the end, the synchronous computation of matrix M is completed.

If D_b: ACTTGCA and Q_r: CATG

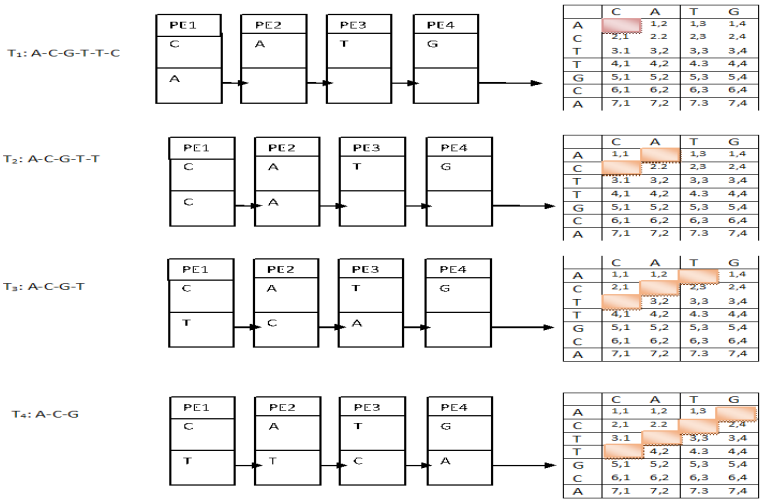


Figure 3: RTL Internal block diagram of SWA-RNS based Processor

2.5.3. *Linear Systolic Array Processing Element (LSAPE).* Figure 5 is a systolic array that computes alignments of two sequences D_b and Q_r. where
D_b: ACTTGCA and Q_r: CATG with scoring parameter of match= 2, mismatch=1 and gap = -1.
The PE shows the nucleotides and the direction of maximum and optimal values. The SW processing element (PE) data path in [12] was slightly modified. Each PE executes the cell. The processing elements in Figure 4 are used to perform dynamic matrix filling. In the design, there are four processing elements, two maximum blocks, three adders and one block computing match/mismatch value between the two nucleotides of the two sequences clearly illustrated in Figure 4. Diagram of the design and Optimized SW-RNS based PE are illustrated in Figure 5 and Figure 7 respectively.
n a complete pass, the total number of clock cycle is equal to 1 less the sum of the lengths of the two sequences. That is, the number of clock cycle in the complete pass is $(4 + 7 - 1) = 10$ cycles. In the Optimized circuit designed (Figure 6), the adders and RNS processors permit the PE to take full advantages of natural properties of RNS like as carry-free, parallelism and fault tolerance.

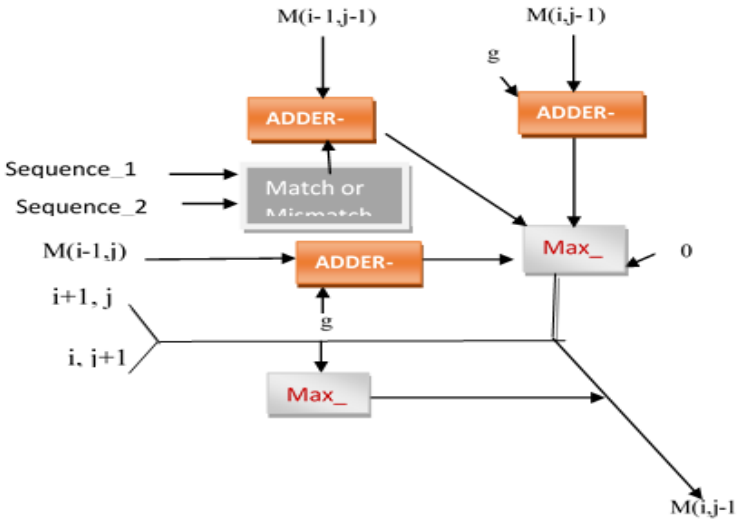


Figure 4: Circuit diagram of PE

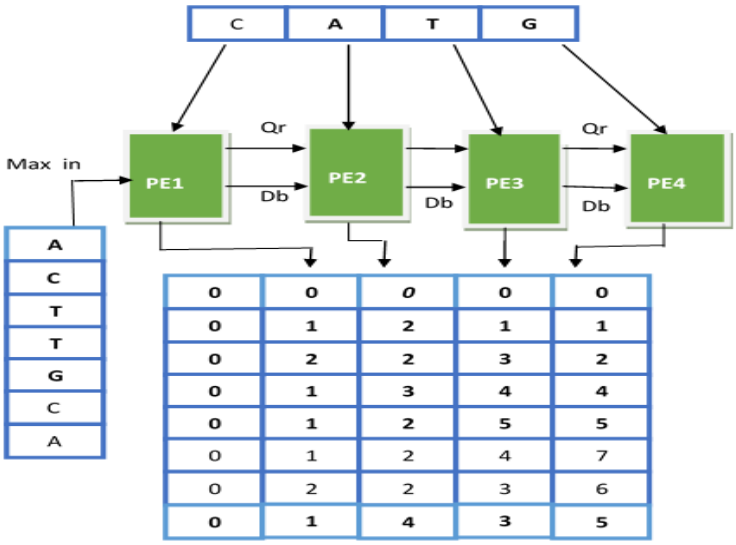


Figure 5: A block diagram of systolic array architecture

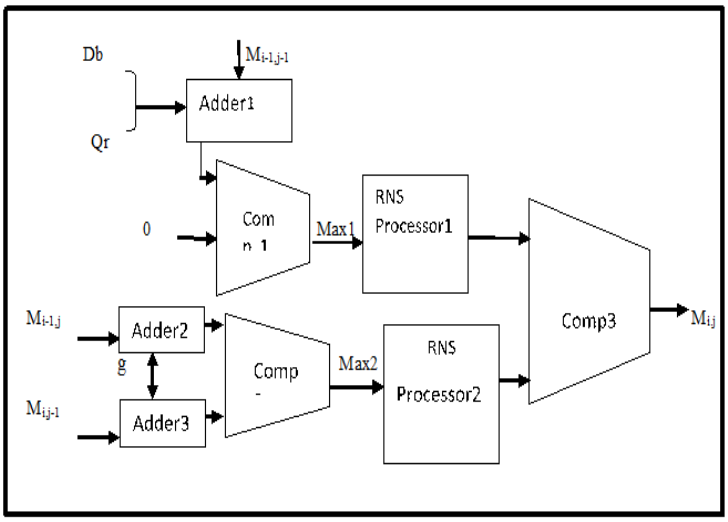


Figure 6: PE circuit design for SWA-RNS based optimization

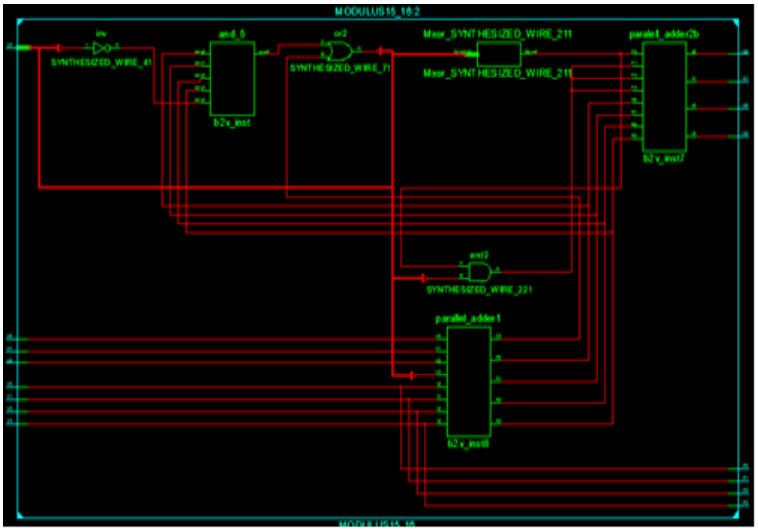


Figure 7: RTL Internal block diagram for Optimized SWA-RNS based PE

2.6. **Summary.** The metric often used to measure performance evaluation in computational biology is the total number of cell update per second (CUPS). The total number of hardware (such as flip-flop, look up table, comparator and so on) needed to compute the element of a cell in SW matrix is termed as processing element where slices are the combination of Flip-Flop (FF) and Look Up Table. Our design with Xilinx (Spartan-III; XC3S250) FPGA stated 10,500 slices where 875 PEs are implemented with a 12 slices per PE at the frequency of 425.6 MHz for the sequence alignment in Figure 6.
Performance Evaluation (CUPS) = Processing elements * Operating frequency
= 875 * 425.6
= 372400 (CUPS)
= 372.4 (GCUPS)
Comparison with others FPGA platform is shown in Table 1. The design is implemented in a low-cost Spartan-III FPGA which is a low priced of Xilinx, this is due to the fact that it does not contain many slices to realize a high-speed performance.

Table 1: Comparison of Performance (systolic array)

Authors	No. of Slices	Slice per PE	Total No. of PE	Operating frequency (MHZ)	Performance evaluation (GCUPS)	Implementation
[10]	55,000	110	500	80.0	40.0	Systolic array and FPGA
[12]	75,968	1,187	64	57.94	3.71	Systolic array and FPGA
[13]	23,136	96	241	98.7	23.79	Systolic array and FPGA
[4]	N.a	N.a	131	193	214	Systolic array and FPGA
[6]	11,264	22	512	200	102.4	Systolic array and FPGA
Proposed Design	10,500	12	875	425.6	372.4	Systolic array, FPGA and RNS

From Table 1 and Figure 8, the proposed design has the smallest slices as compared to others. Since smaller core area can implement more PEs, the speed performance is faster than any other design with higher GCUPS.

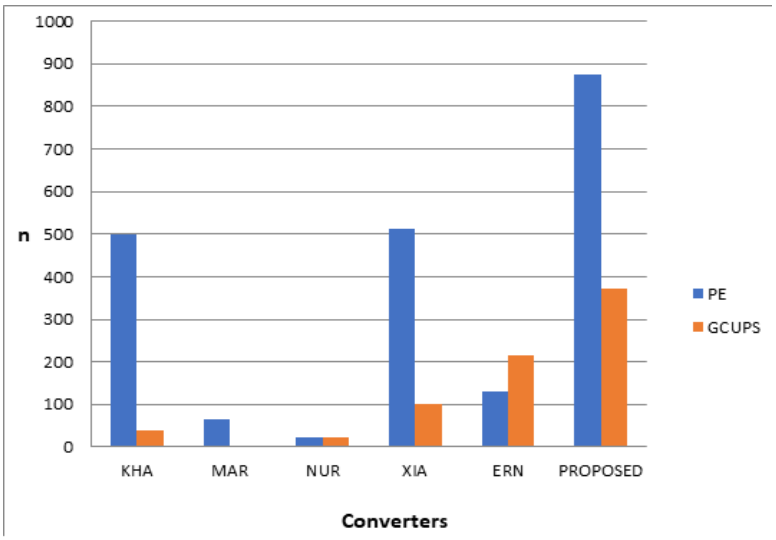


Figure 8: Processing Elements vs Cell Update Per Seconds

Xilinx (Spartan-III; XC3S250) device is used so that the cost of acceleration incurred is minimized through FPGA since this device contains limited numbers of logic slices which inclined to the realization of high performance of our design.

Discussion of Results: This paper presents a first-rate systolic array-FPGA model for SWA sequence alignment using a Residue Number System. Two sequences Q_r and D_b were aligned using Smith-Waterman algorithm. A cell for systolic array was designed and implemented on Xilinx (Spartan-III; XC3S250) FPGA device. Appropriate choice of RNS moduli set contributed immensely to the realization of our objectives. The performance was computed in GCUPS. Performance of our design was 372.4 GCUPS. This performance was compared with other authors in the state-of-the-art with 40GCUPS, 3.71GCUPS, 23.79GCUPS, 102.4GCUPS and 214 GCUPS respectively. Furthermore; the proposed design has the smallest slices which can implement more PEs. All the authors used systolic array architecture on FPGA. The main difference of our architecture with other authors is that, RNS was introduced to accelerate the whole SWA. The proposed design produced the best results in terms of less space consumption and speed of execution as shown in Table 1. It is therefore, enough to say RNS is a hopeful candidate in the acceleration and implementation of Smith-Waterman Algorithm.

Acknowledgement. The author is grateful to Federal Polytechnic, Offa, Kwara State, Nigeria for the supports received during the compilation of this work.

Competing interests: The manuscript was read and approved by the author and declare that there is no conflicts of interest.

Funding: The author received no financial support for the research, authorship, and/or publication of this article.

References

[1] Ahmad H. & Leonel S. (2018). On the Design of RNS Inter-Modulo Processing Units for the Arithmetic-Friendly Moduli Sets $\{2^{n+k}, 2^n - 1, 2^{n+1} - 1\}$. *British Journal of Computer Society. Computer and Communications Networks Systems.* DOI:10.1093/comjnl/bxy119.

[2] Bello H. K. (2020). *Optimizing Smith-Waterman Algorithm Using Residue Number System and Recursive Variable Expansion.* Doctoral dissertation, Kwara State University, Nigeria.

[3] Bello H. K. & Gbolagade K. A. (2018). Acceleration of Algorithm of Smith-Waterman Using Recursive Variable Expansion. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET).* **7** (5).

[4] Ernst J. H., Vlad-Mihai S. & Zaid A. (2017). High Performance Streaming Smith-Waterman Implementation with Implicit Synchronization on Intel FPGA using OpenCL. *IEEE 17th International Conference on Bioinformatics and Bioengineering.* 492-496. DOI: 10.1109/BIBE. 2017. 00089.

[5] Farrar M. (2006). Striped Smith–Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics.* **23** (2), 156-161.

[6] Fei X., Dan Z., Lina L., Xin M. & Chunlei Z. (2018). Fpgasw: Accelerating large-scale smith-waterman sequence alignment application with backtracking on fpga linear systolic array. *Interdisciplinary Sciences: Computational Life Sciences.* **10** (1), 176-188.

[7] Fereshteh B. G. & Shiva T. (2016). An Efficient New CRT Based Reverse Converter for the 4-moduli set $\{2^n, 2^{2n}+1, 2^n+1, 2^n-1\}$. *International Journal of Computer Science and Information Security (IJCSIS).* **14** (12), 226-233.

[8] Hasan L., Khawaja Y. M. & Bais A. (2008). A Systolic Array Architecture For The Smith- Waterman Algorithm With High-Performance Cell Design. *IADIS European Conf. Data Mining.*

[9] Hemantha G. R., Varadarajan S. & Giri P. (2019). *FPGA Implementation of Speculative Pre?x Accumulation-Driven RNS for High-Performance FIR Filter.* Springer Nature Singapore Pte Ltd. https://doi.org/10.1007/978-981-13-365-9_38. 365-375.

[10] Khaled B., Ali A., Cheng L., Yang S., Ying L. & Xiang T. (2012). Review Article High Performance Biological Pairwise Sequence Alignment: FPGA versus GPU versus Cell BE versus GPP. Hindawi Publishing Corporation. *International Journal of Recon?gurable Computing.* Article ID 752910. **15**, 1-15. DOI:10.1155/2012/752910.

[11] Laiq H., Yahya M. K. & Abdul B. (2008). A Systolic Array Architecture for the Smith-Waterman Algghorithm with High Performance Cell Design. *Conference: IADIS European Conference on Data Mining 2008, Amsterdam. The Netherlands, Proceedings.*

[12] Marmolejo-Tejada J. M., Trujillo-Olaya V., Rentería-Mejía C. P. & Velasco-Medina J. (2014). Hardware Implementation of the Smith-Waterman Algorithm using a Systolic Architecture. *IEEE 5^{th} Latin American Symposium on Circuits and Systems.*

[13] Nurdin D. S., Isa M. N. & Goh S. H. (2016). DNA sequence alignment: A review of hardware accelerators and a new core architecture in 2016. *IEEE 3rd International Conference on Electronic Design (ICED)*, 264-268.

[14] Patrick K. M., Edem K. B. & Mohammed M. I. (2018). RNS Smith-Waterman Accelerator based on the moduli set 2^n, 2^n-1, 2^{n-1}-1. *Conference: IEEE 7th International Conference on Adaptive Science & Technology (ICAST)*. DOI: 10.1109/ ICASTECH.

[15] Pemmaraju V. & Ananda M. (2007). RNS-To-Binary Converter for a New Three-Moduli Set 2n+1 1; 2n; 2n -1. *IEEE Transactions On Circuits And Systems II: Express Briefs*. **54** (9), 775- 779.

[16] Phong H. P., & Duck H. N. (2011). Applying GPUs for Smith-Waterman Sequence Alignment Acceleration. *GSTF International Journal on Computing*. **1** (2), 00-00. DOI:10.5176_2010-2283_1.2.56.