International Journal of Electronics and Communication Engineering.
ISSN 0974-2166 Volume 5, Number 1 (2012), pp. 99-112
© International Research Publication House
http://www.irphouse.com

A Smith-Waterman Algorithm Accelerator Based on Residue Number System

Kwame O. Boateng[1] and Edward Y. Baagyere[1,2]

[1]Kwame Nkrumah University of Science and Technology, Kumasi, Ghana
E- mail: boat.soe@knust.edu.gh
[2]University of Development Studies, Navrongo, Ghana
E-mail: eddiesyoung2000@yahoo.com.

Abstract

One of the biggest challenges confronting the bioinformatics community is fast and accurate sequence alignment. The Smith-Waterman algorithm (SWA) is one of the several algorithms used in addressing some of these challenges. Though very sensitive in doing sequence alignment, SWA is not used in real life applications due to the computational cost associated with the software implementation. Heuristics methods such as BLAST and FASTA are used, though they do not guarantee accurate sequence alignments. In this paper, we proposed a novel accelerator for addressing the challenge using Residue Number System (RNS). RNS is an integer system with properties that support parallel computation, carry-free addition, borrow-free subtraction, and single-step multiplication (without partial product). Based on some of these properties, the design of a hardware accelerator for SWA is presented on the assumption that two long strings of DNA can be compared in a divide-and-conquer manner. Simulation of a sample design indicates modest hardware consumption and much improved overall speed of the SWA.

Introduction

The origin of Residue Number System (RNS) can be traced to the puzzle given by Sun Tzu, a Chinese Mathematician and is illustrated as follows: How can we determine a number that has the remainders 2, 3, and 2 when divided by the numbers 7, 5, and 3, respectively? This puzzle, written in the form of a verse in the third century book, *Suan -ching* by the Chinese scholar Sun Tsu, is perhaps the first documented use of number representation using multiple residues. The answer to this puzzle, *23*, is outlined in Sun Tzu's historic work. The puzzle essentially asked us to convert the residues $(2|3|2)_{\text{RNS}(7|5|3)}$ into its decimal equivalent. Sun Tsu formulated a method for manipulating these remainders (also known as residues), into integers. This method is regarded today as the Chinese Remainder Theorem (CRT). The CRT, as well as the theory of residue numbers, was set forth in the 19th century by Carl Friedrich Gauss in his celebrated *Disquisitiones* Arithmetical [1].

This over 1700 – year - old number system is making waves in computing recently. Digital systems implemented on residue arithmetic units may play an important role in ultra – speed, dedicated, real – time systems that support pure parallel processing of integer – value data due to its inherent features such as carry free addition, borrow free subtraction, single step multiplication without partial product, parallelisms, and fault tolerant. These interesting properties of RNS have lead to its widespread usage in Digital Signal Processing (DSP) applications such as digital filtering [2], [3], [12], convolution, correlation, Fast Fourier Transform (FFT)[2], [18],[19]. Discrete Cosine Transform (DCT)[4], [5], image processing, cryptography, digital communications[16], [17] and other highly intensive arithmetic applications[8], [18]. However, RNS has not found wide spread usage in general purpose processors due to difficulties associated with magnitude comparison, sign representation, overflow detection, data conversion, moduli selection, division, and other complex arithmetic operations[6], [10], [11], [13].

RNS is defined in terms of a relatively – prime moduli set $\{m_1, m_2, m_3 ..., m_n\}$, that is *GCD* $(m_i, m_j) = 1$ for $i \neq j$, where *GCD* means greatest common divisor. A binary number *X* can be represented by the residues $(x_1, x_2, x_3, ..., x_n)$, where $x_i = X \bmod m_i$, $0 \leq x_i < m_i$. Such a representation is unique for any integer $X \in [0, M-1]$, where $M = \prod_{i=0}^{n-1} m_i$ is the dynamic range of the system. For a signed number system, any integer in $(-M/2, M/2)$, has a RNS n – tuple representation where $x_i = X \bmod m_i$ if *X > 0,* and $(M - |X|) \bmod m_i$ otherwise. The signed RNS system is often referred to as a symmetric system [7], [9].

Addition, subtraction, and multiplication in RNS are very efficient since digit by digit computations are allowed. Additionally, there is no ordering significance between the digits. However, division in RNS is rather complex since it is not a closed operation. For example, given that X*, Y, and Z* have RNS representations:

$$\left.\begin{array}{l} X \xrightarrow{RNS} (x_1, x_2 ..., x_n) \\ Y \xrightarrow{RNS} (y_1, y_2 ..., y_n) \\ Z \xrightarrow{RNS} (z_1, z_2, ..., z_n) \end{array}\right\} \qquad (1)$$

and supposing that # denotes the operation +, -, or *, then *Z=X # Y,* means

$z_i = \left(x_i \# y_i\right) \bmod m_i$ ....(2)

if Z belongs to Z_M

This means that no carry information need be communicated between residue digits. This explains why RNS is applicable in high performance computing and thus widely used in highly intensive DSP applications. In order to fully exploit these RNS parallelisms, arithmetic units that efficiently implement the modular statement $\left(X_i \# Y_i\right) \bmod m_i$ must be found [9].

Smith-Waterman Algorithm

Bioinformatics is becoming an increasingly important field of research. With the ability to rapidly sequence DNA information, biologists have the tools to, among other things, study the structure and function of DNA, study evolutionary trends; and correlate DNA information with disease. For example, two genes were identified to be involved in the origins of breast cancer in 1994 [20]. Such a research is only possible through the help of high speed sequence comparison.

A human genome contains approximately 3 trillion DNA base pairs. In order to discover which amino acids are produced by each part of a DNA sequence, it is necessary to find the similarity between two sequences. This is done by finding the minimum string edit distance between the two sequences and the process is known as sequence alignment.

There are many algorithms for doing sequence alignment. The most commonly used ones are FASTA [21] and BLAST [22]. BLAST and FASTA are fast algorithms which prune the search involved in a sequence alignment using heuristic methods. These methods are extremely very fast but at the expense of accuracy. The most accurate sequence alignment algorithm available is the Smith-Waterman Algorithm (SWA) [23]. However, the SWA is computationally very expensive for particularly long sequences.

The SWA is an optimal method for homology searches and sequence alignment in genetic databases and makes all pair wise comparisons between two strings of DNA. It achieves high sensitivity as all the matched and near-matched pairs are detected; however, the computation time required strongly limits its use. We believe RNS as a tool, can be used to address the computational time limitation of SWA. The description of SWA has been extensively presented in [24], [25], [26].

In calculating the local alignment, the matrix H(i ,j) is used to keep track of the degree of similarity between the two sequences that are aligned.

The elements of H(i, j) are calculated using the following equation:

$$H(i,j) = \max\begin{cases} 0 \\ H(i-1,j-1)+S(i,j) \\ H(i-1,j)-d \\ H(i,j-1)-d \end{cases} \qquad (3)$$

Where
H(i, j) is the maximum similarity score between the two sequences compared
S(i, j) is the similarity score in comparing sequence A_i to sequence B_j and
d is the gap penalty for a mismatch in the comparison.

The SWA is executed in the following three main steps.
The initialization step:-Matrix H is initialized by setting H(0,j) = 0 and H(i,0) = 0 for all i and j.

Matrix fill step:- This step is done to fill in all the entries of the matrix using equation (1). This step in particular is computationally intensive, and several attempts have been made in literature to accelerate this step in hardware [24, 25].

Trace back step: - The matrix scores entered are trace back to inspect optimal score local alignment. The trace back starts at the cell in the matrix with the highest score and then continues up the entire matrix until the score fall down to a minimum predefined value.

The time complexity of the initialization step is O(M + N), where N and M are the sizes of the two sequences. During the matrix fill step, the entire matrix needs to be filled using equation 1, making its time complexity equal to the number of cells in the matrix or O(MN). The time complexity of the trace back is also O(MN), as the entire matrix needs to be traversed during this step. Thus the total time complexity of the SWA is O(M + N) + O(MN) + O(MN) = O(MN). The total footprint of the SWA is also O(MN), as it fills a single matrix size MN. In order to reduce the O(MN) complexity of the matrix fill stage, multiple entries of the H(i, j) are calculated in parallel [24, 25].

The inherent carry-free addition property of RNS is exploited in addressing the computational challenge associated with the SWA.

Hardware Acceleration of the Matrix Fill Step

In this section, we present the proposed novel method of using RNS to address the computational challenge associated with the SWA. This method exploits the arithmetic advantages and the modular nature of RNS to accelerate the SWA. In the next paragraph we present the architecture and organization of the acceleration logic.

The acceleration logic of the SWA implementation is made up of three major building blocks. These are as follows.

- The Binary/Decimal-to–RNS conversion,
- RNS Processor,
- RNS Reverse Converter/Comparator.

These blocks are depicted in Figure 1.

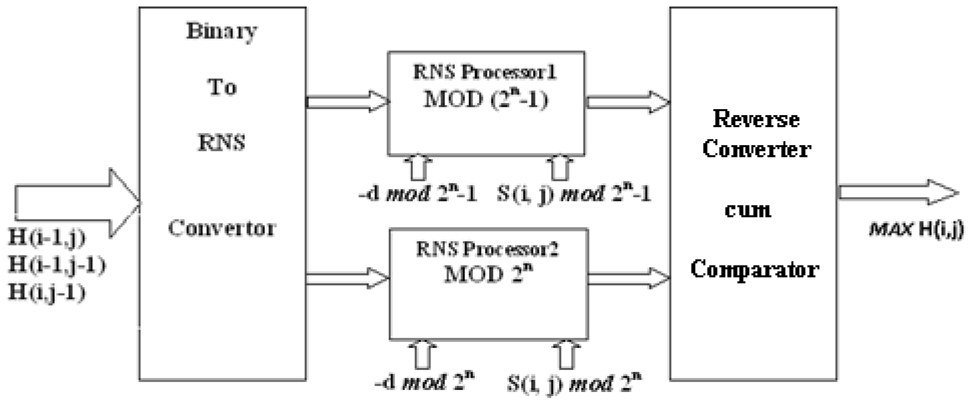


Figure 1: The proposed RNS-SWA accelerator architecture

The unit that performs the conversion of the SWA inputs into their residue equivalents is termed as RNS Forward Converter. In the next subsection we present an instance of a customized memory-less RNS forward converter. The converter does not need any memory or processing elements (PEs) in its residue computation. It works for both signed two's-complement and unsigned numbers. The moduli set used in the implementation is m = {2^n, 2^n-1}. The dynamic range gives the number of unique representation of the decimal numbers in the RNS system. In order to be able to use moduli set with small dynamic range, matrix partitioning is assumed based on the fact that the comparison of two long strings of DNA can be done in a divide–and–conquer manner.

The RNS forward converter with m = {16, 15}

Using the moduli set m = {16, 15} yields a dynamic range M = 240. This means sub-matrices should have elements whose values are restricted within the range [-120, +119]. This is useful information in determining the sizes of sub-matrices and for that matter the number of sub-matrices vis-à-vis the lengths of the strings being compared.

The decimal number, D, which is an *8-bit number,* is partitioned into *two nibbles* namely *U* and *V*. *U* (i.e. *U0, U1, U2, U3*) is the high order nibble of the binary representation of D and *V* (i.e. *V0, V1, V2, V3*) is the low order nibble. For modulus 2^n, the residue is simply the lowest order n bits. However for modulus 2^n-1, the forward conversion is not as simple.

In the case of modulus 15 the high order nibble is added to the low order nibble by a parallel adder which is made up of one half adder and three full adders, labeled *Parallel Adder1* in Figure 2. The sum from *Parallel Adder1,* namely *R1, R2, R3, R4,* forms an operand for a second stage of addition. The second operand for this stage is derived from U3, the carry-out and the sum from *Parallel Adder1* as per the intervening logic shown between *Parallel Adder1* and *Parallel Adder2* in Figure 2. The sum from this stage (without the carry-out) constitutes the D *mod* 15 representation. D *mod* 16 is simply the nibble V.

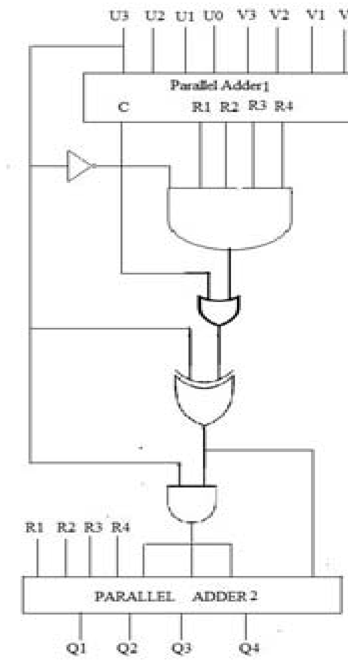


Figure 2: The Modulus 15 architecture

The RNS Processor

The next step after the binary/decimal conversion to RNS phase is the RNS Processing stage. This stage exploits the inherent properties of RNS to do carry-free arithmetic. The design consists of two sets of four bit-sliced 2-to-1 multiplexers, two modulus 15 parallel adders, one modulus 16 parallel adder and a control unit.

Carry free additions and borrow free subtractions operations are done on the residues produced by the mRNS forward converter in accordance with Equation 3 shown above. The Sequence of these arithmetic operations is as follows:

- Components of H(i-1,j-1) are added to the components of S(i, j), (this is called the Diagonal addition),
- Components of H(i-1, j) are added to the components of (- d), (this is called the Upper addition) and
- Components of H(i, j - 1) are added to the components of (-d), (this is called the Left addition).

The logic in the control unit controls the sequencing of these additions. In this implementation, d = -2, since this is the constant value mostly used in SWA calculation in literature.

The RNS Reverse Converter/Comparator Implementation

The last stage is the RNS reverse conversion/comparison. The block diagram in Figure 3 shows the RNS reverse converter/comparator. It performs reverse conversion of the residue results of the arithmetic operation by the RNS processor to twos complement binary representation and compares them with zero and with each other. The comparisons ends with the outputting of the maximum value for assignment to H(i, j) as the matrix score.

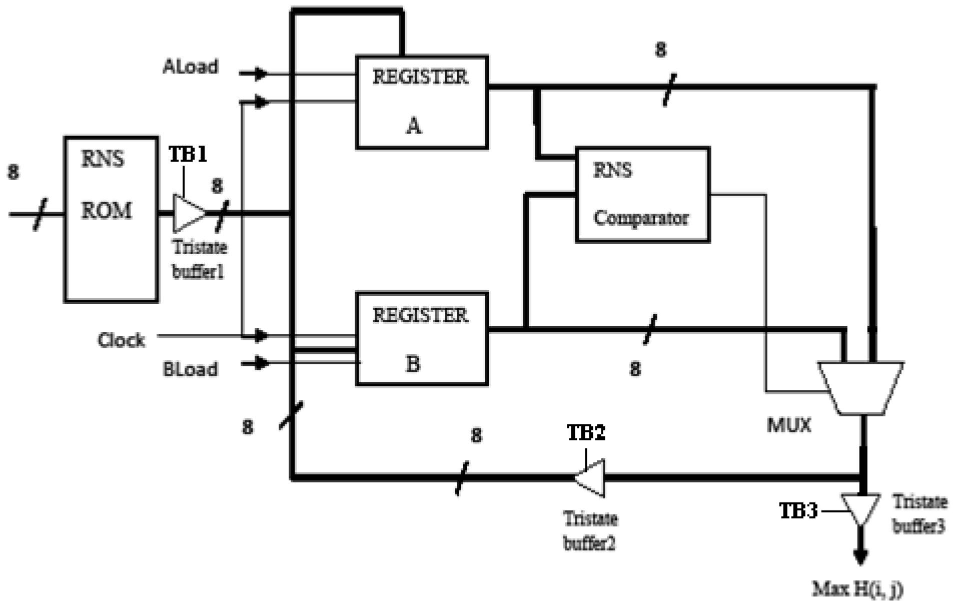


Figure 3: Schematic Diagram of the RNS Reverse Converter/Comparator

The unit comprises a 256×8-bit ROM that contains the twos-complement values of all the numbers within the dynamic range. The two residue nibbles are concatenated into an 8-bit address of the location where the corresponding twos-complement value of the decimal is stored. The decimal values corresponding to the four values {H(i-1, j-1) + S(i, j), H(i -1, j) –d, H(i, j-1) –d and 0} being compared are read into two different registers in various clock cycles and then compared by a binary comparator. Only two comparisons are actually done: H(i-1, j-1) + S(i, j) is compared with H(i -1, j) –d to get the first maximum value. This initial maximum value is compared with H(i, j-1) –d to obtain the overall maximum of the three values. The value 0 is placed in the Register when the retrieved twos-complement value is negative. Figure 2 shows the datapath of the reverse converter/comparator.

Performance Evaluation of the Accelerator Implementation

Laiq Hasan and Zaid Al - Ars in 2007 [25], used the GNU profiler, *gprof*, to profile software implementation of SWA in order to get the function that consumes most of the computation time. Table 1 shows the profiling result that was obtained. The GNU profiler gave information about the number of times, each function is called and the number of Clocks Ticks consumed by each function. The code was run on the Intel

Pentium - IV (3.2 GHz) processor, for which the time period of the clock is

$$\frac{1}{3.2GHz} = 0.312ns$$

The matrix fill function, labeled *fill_matrix_2* in Table 1 was identify as the most called function and consumed 72.33% of the total runtime, making it the right candidate to be implemented in hardware. In the table, *fill_matrix_2* function took 5.23µs of the total time for running 100 times. So the actual time consumed by one run of the matrix fill stage function is

$$\frac{5.23}{100}\mu s = 0.05232\mu s$$

In Laiq Hasan and Zaid Al-Ars, 2007 [25], the post place and route simulation showed that the total delay of their hardware implementation was 0.0146µs, whereas the time consumed by its software equivalent was 52.32µs. Their runtime improvement was calculated to be 3582% = 35.82 times.

Specifications of the proposed accelerator entered into a *Quartus II version 4.0* VHDL application software using the graphic entry or schematic capture tool embedded in the software. Figure 4 is the schematic of the RNS forward converter as represented using the design tool.

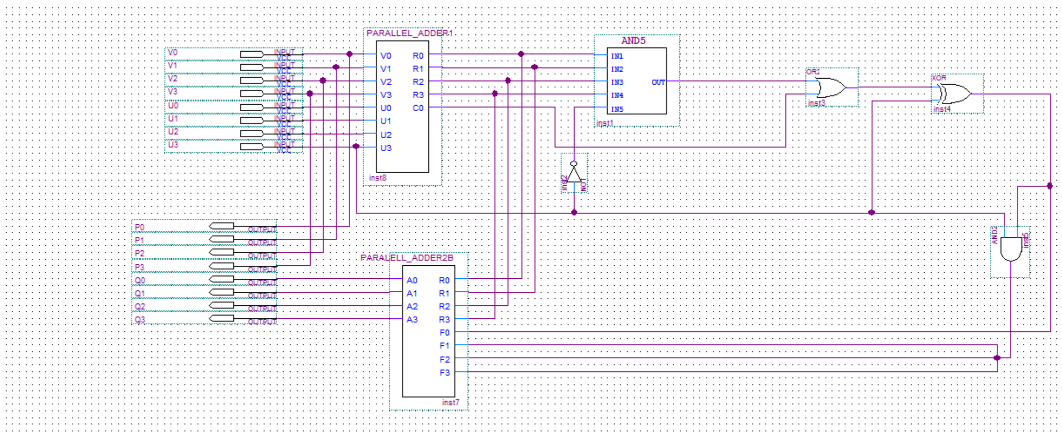
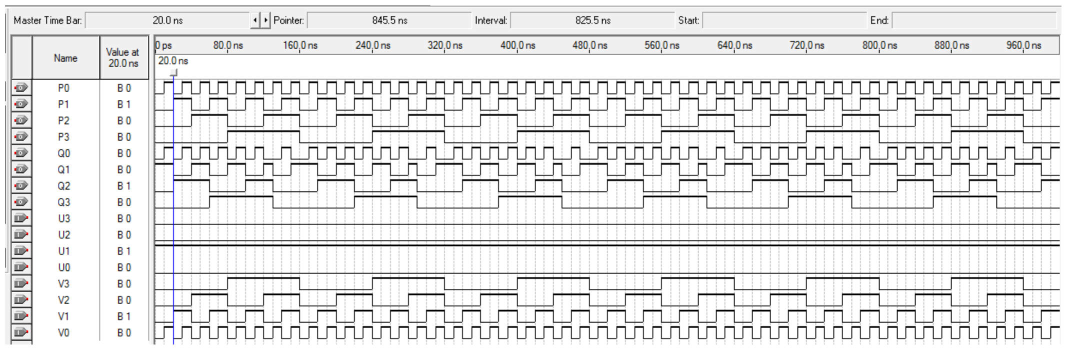


Figure 4: Schematic diagram of the RNS Forward Converter

After the design entry is completed, it is compiled, in order to translate the source code into object code in format that can be logically tested or downloaded to a FPGA target device. Next to the compilation process is the functional simulation. This is done by the software to confirm that the logic circuit functions as expected. Simulation was done block-by-block and finally for the entire accelerator in order to verify that the correct outputs are produced for a specified set of inputs. A waveform editor (a device independent software tool) is what was used for the verifications. Figure 5 shows a snapshot of simulation results of the RNS forward converter.


Master Time Bar: 20.0 ns Pointer: 845.5 ns Interval: 825.5 ns Start: End:

Name	Value at 20.0 ns
P0	B 0
P1	B 1
P2	B 0
P3	B 0
Q0	B 0
Q1	B 0
Q2	B 1
Q3	B 0
U3	B 0
U2	B 0
U1	B 1
U0	B 0
V3	B 0
V2	B 0
V1	B 1
V0	B 0

Figure 5: A snapshot of results of simulation of the RNS Forward Converter

As shown in Figure 5, at 20 ns from the start of simulation $V = v_3v_2v_1v_0 = 0010$ and $U = u_3u_2u_1u_0 = 0010$, which implies that the representation of the number in twos complement is 0010 0010 (i.e. 34_{10}). At the time in question $P = V = 34\, mod\ 16 = 2 = p_3p_2p_1p_0 = 0010$ and $Q = 34\, mod\ 15 = 4 = q_3q_2q_1q_0 = 0100$. This can be confirmed by result of the simulation. NOTE: *The simulation tool displays 0 and 1 as B0 and B1, respectively.*

Finally, timing simulation was done to verify that the circuit works at the design frequency and that there are no propagation delays or other timing problems that will affect the overall operation of the circuit when implemented on the hardware device.

The performance of the proposed accelerator was evaluated in terms of speed and hardware cost. Note that our work sought to improve upon the computational cost associated with the matrix fill stage (labeled as fill_matrix_2 in the Table 1). Table 2 shows the device utilization summary for the proposed approach. The timing simulation of the proposed accelerator shows that the total delay is equal to 6.006ns at a clock speed of 185.53 MHz.

Comparison between the processing runtime of the pure software version and the delay of hardware version of the implementation as formulated in L. Hasan and Z. Al – Ars work [25] gives the percentage runtime improvement achieved. Mathematically, the percentage runtime improvement of a hardware accelerator implementation of the fill_matrix_2 is calculated as follows:

$$\text{Runtime Improvement} = \left[\frac{\dfrac{1}{Hardware\_time} - \dfrac{1}{Software\_Runtime}}{\dfrac{1}{Software\_Runtime}}\right] * 100\% \qquad (4)$$

Using a similar formulation for performance comparison between the delay of hardware version by L. Hasan and Z. Al – Ars (*Hardware_time(old)*) and the delay of our hardware version (*Hardware_time(new)*) of the implementation, the runtime improvement achieved by our design is as follows:

$$\text{Runtime Improvement} = \left[\frac{\frac{1}{Hardware\_time(new)} - \frac{1}{Hardware\_time(old)}}{\frac{1}{Hardware\_time(old)}}\right] * 100\% \qquad (5)$$

Substituting the software runtime value from [25] and the propagation delay of the proposed accelerator into Equation 5 gives

$$\text{Runtime Improvement} = \left[\frac{\frac{1}{6.006x10^{-9}} - \frac{1}{14.6x10^{-9}}}{\frac{1}{14.6x10^{-9}}}\right] * 100\% = 143\%$$

Thus the total time that will be needed to perform the *Fill_Matrix_2* function during alignment two strings of DNA using the RNS - SWA architecture will be 2.43 times shorter than that needed by the hardware implementation reported in [25] to perform the same function. Also, relative to the software implementation of the *Fill_Matrix_2* function, a runtime improvement of 871,029% has been achieved as compared to 3,582% runtime improvement achieved by L. Hasan *et al.*[25]. This achievement has been accomplished using modest amount of hardware as shown in Table 2.

It can therefore be inferred that RNS provides a good platform to implement SWA, since it has a high prospect of improving the overall computational cost of the algorithm. Table 2 also provides hardware cost information, which appears to be modest.

Conclusions

The major challenge faced by the bioinformatics community is fast and accurate sequence alignment. The Smith-Waterman Algorithm (SWA), though sensitive in aligning sequences of DNA or RNA, is computationally intensive. Several attempts have been made to accelerate this algorithm in hardware to fully explore its advantages. In this paper, we proposed a novel approach in solving this computational challenge by the exploitation of the inherent properties of the Residue Number System. Our solution is based on the assumption that one can use moduli set with a small dynamic range with matrix partitioning to compare two long strings of DNA in a divide-and-conquer fashion.

This proposed hardware architecture is made up of a forward converter, two concurrent channels—a MODULUS 2^n and MODULUS 2^n-1 channels that do parallel computations in the most computation intensive stage of SWA, and a reverse converter cum comparator unit. We have simulated an instance of the proposed RNS-SWA accelerator (for n=4). The worst-case propagation delay (t_{pd}) between the specified source and destination points is 6.006 ns. The accelerator is both area and time efficient. The percentage runtime performance improvement as compared with the hardware accelerator in [25] shows a 143% improvement in speed. This improvement in speed is attributable to the fact that RNS offers a good platform to accelerate the SWA.

Acknowledgements

We gratefully acknowledge Dr. K. A. Gbolagade for introducing Mr. E. Y. Baagyere to the concepts of Residue Number System (RNS).

Table 1: L. Hasan and Z. Al –Ars Profiling Results of the software implementation of the SWA.

Function	No. Of Calls	No. Of Clock Ticks	No. Of Clock Cycles Consumed	Total Time Consumed (µs)	% Time Consumed
Init_Matrix	100	71944	2302208	0.718	9.93
Fill_Matrix_1	100	32392	1036544	0.323	4.47
Fill_Matrix_2	4800	524040	16769280	5.23	72.33
Trace_back_1	100	31232	999424	0.312	4.31
Trace_back_2	500	64944	2078208	0.648	8.96

Table 2: Hardware resource utilization table of the proposed accelerator hardware

Flow Status	Successful - Nov 26 2010
Revision Name	FINAL_SWA_PROCESSOR
Top-level Entity Name	FINAL_SWA_PROCESSOR
Family	Stratix II
Total combinational functions	143
Total registers	46
Total pins	32 / 343 (9 %)
Total memory bits	0 / 419,328 (0 %)
DSP block 9-bit elements	0 / 96 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)
Device	EP2S15F484C3
Total ALUTs	189 / 12,480 (1 %)

References

[1] Soderstrand, M. A., Jenkins, W. K., Jullien, G.A., and Taylor, F. J., 1986, Residue Number System Arithmetic: Modern Applications in Digital Signal Processing, New York: IEEE Press.

[2] Conway, R., and Nelson, J., 2004, “Improved RNS FIR Filter Architectures. IEEE Trans. on Circuits and System – II,” Express briefs, Vol. 51, No. 1, pp. 26 – 28.

[3] Jenkins, W.K., and Leon, B. J., 1977, ”The use of Residue Number Systems in the design of finite Impulse Response Digital Filters, IEEE Trans. on Circuit and Systems, vol. 24: pp 191 – 200.

[4] Fernandez, P. G., Garcia, A., Ramirez, J., and, Lloris, A., 2000, “Fast RNS – based 2D – DCT computation on Field _ Programmable devices,” Proceedings of IEEE Signal Processing Systems Workshop, pp. 365 – 373, LA, USA.

[5] Fernandez, P. G., Garcia, A., Ramirez, J., and, Lloris, A., 2000, “A RNS – based Matrix – vector – multiply FCT Architecture for DCT computation,” Proceedings of 43rd IEEE Midwest symposium on Circuits and Systems, pp. 350 – 353, Lansing, MI.

[6] Diauro, G., Impedovo, S., and Pirlos, G., 1993, “A new technique for fast comparison in Residue Number System,” IEEE Trans., on Computers, Vol. 42, No. 5, pp. 608 – 612.

[7] Parhami, B., 2000, Computer Arithmetic and Hardware designs, New York, Oxford University press.

[8] Baraiecka, A. Z., and Jullien, G. A., 1980, “Residue Number System Implementation of number theoretic transform in complex residue rings,”. IEEE Trans. Acoustics, Speech, Signal Processing, Vol. ASSP – 28, pp. 285 – 291.

[9] Taylor, F.J., 1984, “Residue Arithmetic: A Tutorial with examples,” IEEE comp. magazine, vol. 17, No. 5, pp. 50 – 62.

[10] Bi, S., and Gross, W. J., 2008, “The Mixed – Radix Chinese Remainder Theorem and its applications to residue comparison,” IEEE Trans. on computers, Vol. 57, No. 12, pp. 1624 – 1632.

[11] Al – radadi, E., and Siy, P., 2001, “A new technique for fast number comparison in the residue number system based on Chinese Remainder Theorem II,” Proceedings of the MUG 18th International Conference, Denver.

[12] Tenkins, W. J., 1978, “Techniques for residue – to – analog conversion for residue – encoded digital filters,” IEEE Trans. on Circuits and Syst. Vol. CAS – 25, pp. 555 – 562.

[13] Jullien, G. A., 1978, “Residue Number Scaling and other operation using ROM arrays,” IEEE Trans. Computers, Vol. C. 27, pp. 325 – 336, April, 1978.

[14] Huang, C. H., and Taylor, F. J., 1980, “High speed DFTs using Residue Numbers,” In Proceedings IEEE 1980 conference Acoust., speech, signal processing, Denver, Co, pp. 238 – 241.

[15] Taylor, F. J., 1990, “An RNS Discrete Fourier Transform Implementation,” IEEE Trans. Acoust. Speech, Signal Process, Vol. 38, No. 8. Pp. 1386 – 1394.

[16] Madhukumar, A. S., and Chin, F., 2002, “Performance of a Residue Number system based CDMA system over wireless personal communication”, Springer. Netherlands, Vol. 22, No. 1, pp. 89 – 102.

[17] Madhukumar, A. S., Chin, F., and Premkumar, A. B., 2000, “Residue Number System based multicarrier CDMA for broadband mobile communication

systems," Proceedings of 43rd IEEE Midwest symposium on circuits and systems. pp. 536 – 539, Lansing, MI.

[18] Merill, R.D., 1964, "Improving digital Computer performance using Residue Number Theory. Trans. on Electronic Computers". Vol. 13, Issues 2, pp. 93 – 101.

[19] Taylor, F. J., and Huang, C. H., 1982, "A comparison of DFT algorithms using a Residue Arithmetic Architecture. International Journal of Computer and Electronic Engineering.

[20] Smith, T.F., and Waterman, M. S., 1981, "Identification of common molecular subsequences," In journal of molecular biology, vol. 147, pp 195 – 197.

[21] Miki, Y., et al., 1994, "A Strong Candidate for the Breast Cancer and Ovarian Cancer Susceptibility Gene, BRCA1," Science, 266:6-71.

[22] Altschul, S. F., et al., 1990, "A Basic Local Alignment Search Tools," In Journal of Molecular Biology, vol 215, pp. 403 – 410.

[23] Pearson, W. R., and Lipman, D.J., 1985, "Rapid and Sensitive Protein Similarity Searches," In Science, vol 227, pp 1435 – 1441.

[24] Hasan, L, et al, September 2 – 5, 2007, "Hardware Acceleration of sequence alignment algorithms – an overview," Proceedings of International conference on Design and Technology of Integrated Systems in Nano cell Era (DTIS'07), pp 96 – 101, Rabat, Morocco.

[25] Hasan, L., and Al-Ars, Z., November 29 – 30, 2007, "Performance Improvement of the Smith–Waterman algorithm," Annual workshop on circuits, systems and signal processing (ProRISC). Veldhoven, The Netherlands.

[26] Yu, C. W., Kwong, K. H., Lee, K. H., and Leong, P. H. W., 2003, "A Smith-Waterman systolic cell," In Proceedings of the 13th International Workshop on Field Programmable Logic and Applications.

Biographies

KWAME OSEI BOATENG is a senior lecturer in the Department of Computer Engineering at Kwame Nkrumah University of Science and Technology in Kumasi, Ghana, and heads the KVCIT of IDL of the same university. He earned a B.S. in Electrical/Electronic Engineering in 1991 from KNUST, an M.S. in Computer Science in 1997 from Ehime University, Japan, and a Doctoral degree (in Information Systems Engineering, 2000) from Ehime University, Japan. Dr. Boateng has worked as a researcher at Fujitsu Laboratories Ltd., Japan and was an ICT consultant for KNUST. His research interests include, design and test of logic circuits, reconfigurable instrumentation and residue number system applications.

EDWARD YELLAKUOR BAAGYERE received his BSc.degree (Hons) in Mathematical Sciences (Computer Science option) from the University for Development Studies, Tamale, Ghana in 2006. He joined the Faculty of Applied Sciences of the same University as a Senior Research Assistant in 2007 where he assisted in Teaching and Researching. He is currently pursuing an MPhil programme in Computer Engineering at the Kwame Nkrumah University of Science and Technology, Kumasi, Ghana. His research interests include Computer Arithmetic, Residue Number System, Bioinformatics, Digital Logic Design, Parallel Computing and Numerical Computing.