High-Throughput Processors for KBand-Based Global PSA of DNA

Duverney Corrales Mendoza, Jaime Velasco Medina, Senior Member, IEEE

**Abstract**— Pairwise Sequence Alignment (PSA) is an essential task in bioinformatics for identifying regions of similarity between two biological sequences (DNA, RNA or proteins). Optimal PSA is carried out by two processes: forward process (FWP) and traceback process (TBP). FWP is performed by using algorithms based on Dynamic Programming (DP) technique. In the optimal global PSA case, Needleman-Wunsch Algorithm (NWA) is used to perform FWP. Nevertheless, NWA process matrices with score values that increase according to the sequence length. Therefore, NWA is usually performed by high-performance systems with a tradeoff between base-pair sequence length and number of Processing Elements (PEs). Fortunately, hardware-oriented algorithms and PEs for performing NWA based on arithmetic mod-2N have been proposed recently, allowing designing hardware accelerators with high PE density and processing base-pair sequences of any length.

In this paper, we present the design of two “high-throughput processors for KBand-based PSA” (KBP) of DNA, using NWA with linear gap penalty (KBPL) or NWA with affine gap penalty of two matrices (KBPA2). Each KBPs use a Systolic Array Architecture (SAA) based on arithmetic mod 2N, which allows processing base-pair sequences of any length and achieving a high PE density. KBPs were synthesized on several FPGA platforms, achieving a PE density of 3-bit of: 2048 PEL and 1024 PEA2 on a CycloneV SoC-FPGA, 4096 PEL and 2048 PEA2 on a Cyclone10 GX FPGA, and 32768 PEL and 16384 PEA2 on a Stratix 10 FPGA. KBPs use two clock domains for increasing the output throughput. KBPs were verified on a co-design embedded system for global PSA, where KBPs were used to accelerate FWP, whereas TBP was performed fully in software. In these cases, KBPs were configured on a Cyclone-V SoC-FPGA with 1024 PEs of 4-bit, 50MHz internally and 100MHz externally, achieving PSA processing rates of 84s/Mbp, that correspond to speed-up 58 times the global PSAL case, and speed-up 427 times the global PSAA2 case, respect to software implementation. KBPs were tested with DNA base-pair sequences of similar lengths up 31-Mbp.

**Index Terms**— Large DNA sequence alignment, KBand-based global alignment, PSA, Systolic array, modular arithmetic, hardware accelerator, co-design embedded system, FPGA.

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

————————————————

1. D. Corrales-Mendoza is with the Electrical and Electronic School, Universidad del Valle, Cali, CO calle13 #100-00. E-mail: duverney.corrales@correounivalle.edu.co.
2. J. Velasco-Medina is with the Electrical and Electronic School, Universidad del Valle, Cali, CO calle13 #100-00. E-mail: Jaime.velasco@correounivalle.edu.co.

*Manuscript received 9 Dec. 2014; revised 2 June 2015; accepted 8 June 2015.  
Date of publication 10 June 2015; date of current version 13 Apr. 2016.  
Recommended for acceptance by R. Cumplido.  
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identiﬁer below.  
Digital Object Identiﬁer no. 10.1109/TPDS.2015.2444376.*

—————————— ◆ ——————————

# 1 Introduction

P

SA is used to find the similarity between biological sequences (DNA, RNA or proteins), in order to know functional, structural and evolutionary relationships [\*] and it is essential to perform more complex task as Sequence Assembly (e.g. Next Generation Sequencing) [\*], Multiple Sequence Alignment (MSA) [\*], build of Phylogenetic Trees [\*] and Databases Searching [\*].

Optimal global PSA is performed using DP-based algorithms. The main DP-based PSA algorithms are Needleman-Wunsch Algorithm (NWA) for global PSA [\*] and Smith-Waterman Algorithm (SWA) for local PSA [\*]. These algorithms have a space and time complexity O(n2) from the view point of software implementation. Therefore, heterogeneous platforms (CPU more GPU, FPGA or ASIC devices) are mainly used to increase the PSA performance [\*].

Specifically, ASIC and FPGA based platforms must address an issue few mentioned: “the number of bits required for the arithmetic units for processing the matrix H”, which could rise when the filling of the similarity matrix H go forward, increasing the required number of bits N. This issue increases the PE area and reduces the PE density of the SAA. For example, the processing of DNA pair-sequences of 1Mbp (220 bases) using a scoring scheme {match, mismatch, gap} = {3, -1, -2} requires of PEs that process 23-bits.

The above issue has a high impact in the performance, since the PE density and PE frequency of the SAA are highly reduced for very length sequences, being necessary to find a tradeoff between sequence length and number of PEs. In [\*], the authors present a SAA for pair-sequences up 80kbp using 80-PEs of 20-bits. In [\*], the authors present a SAA for lengths up 10kbp using 128-PEs of 12-bits. In [\*], the authors present an efficient SAA over FPGA for SWA using KBand method, used for accelerating the Mercury BLASTP; However, this design is limited to sequences of 2-kbp and 75-PEs of 11-bits.

Arithmetic mod-2N is an adequate solution in order to address the above issue. To the best of our knowledge, there are no reported hardware accelerators for PSA using arithmetic mod 2N. Some works as the presented in [\* \*\*] are based on the Lipton Lopresti Simpliﬁcation (LLS), reaching a high PE density; however, those works are restricted to the specific linear gap score scheme {match, mismatch, gap} = {0,2,1}.

Then, we desig KBand-based high-throughput processors (KBPs) for global PSA, considering the algorithms proposed in [\*] and the SAA proposed in [\*].The main contributions of this work are: 1) the design of two KBPs for performing the FWP global PSA with linear gap or affine gap penalties, using the NWA(mod 2N) and packet processing method. The above algorithms and methods maximize the PE density of the KBP, and allow performing PSA of any sequence length, verified up 31-Mbp. 2) a new algorithm for performing the TBP from the arrows-output generated by the designed KBPs. 3) the design of a SoC-FPGA embedded system for performing the FWP by a KBP-based co-design application and the TBP by software, obtaining a completely functional, high performance and a low-cost FPGA-based platform for global PSA.

The rest of this paper is organized as follow: In section II, a brief theoretical background for global PSA and optimization methods is presented. In section III, the operation principles of the KBPs and the conceptual design details of the SAA are described. In section IV, the designed embedded system, and the algorithms for FWP and TBP are presented. In section V, the results of synthesis, performance and verification are presented. Finally, the conclusions are presented in section V.

# 2 Hardware-oriented algorithms and Speed-up methods for global PSA

In this section, firstly we present generalities of global PSA algorithms: FWP, TBP, gap penalties, similarity matrix, arrow matrix and gap penalty matrices. Then, we present the hardware-oriented algorithms, PE designs and speed-up methods used for developing the proposed KBPs.

DP-based global PSA is implemented by using two process: FWP and TBP. FWP performs progressively the building of the similarity’s matrix H and the arrow’s matrix A using the NWA, where the matrix A is used to indicate the source path associated to maximum selected value of each cell of the matrix H. TBP allows to find the optimal global PSA by following the path indicated by the arrows (best score path), starting in the last A-matrix cell and finishing in the first A-matrix’s cell, as can be seen in Figure 1.

Different optimal PSA can be obtained according to used score scheme, which is selected by considering the statistical properties of the pair-sequences to align. For the DP-based global PSA case, the score scheme is composed by a score matrix s(xi , yj) and linear or affine gap penalties. The score matrix match/mismatch is generally used for DNA sequences, where the value “match” is assigned when and the value “mismatch” when . Linear gap penalty considers the the same biological cost g for all gaps. Unlike, affine gap penalty considers different biological costs for open gaps (d) and extended gaps (e) in order to improve the gap distribution [12].

Global PSA with linear gap penalty is performed using only one matrix: the similarity matrix H. However, global PSA with Affine gap penalty is performed using the three matrices {H, GX, GY}, where H is the similarity matrix, and GX and GY are the gap matrices. However, the matrices GX and GY can be reduced to one matrix G, obtaining two matrices {H, G} for the case when mismatch≤2e [13].

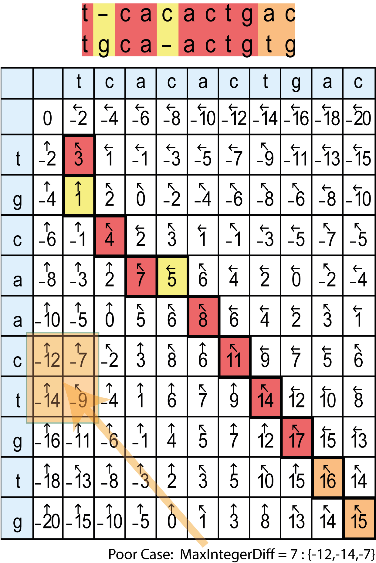
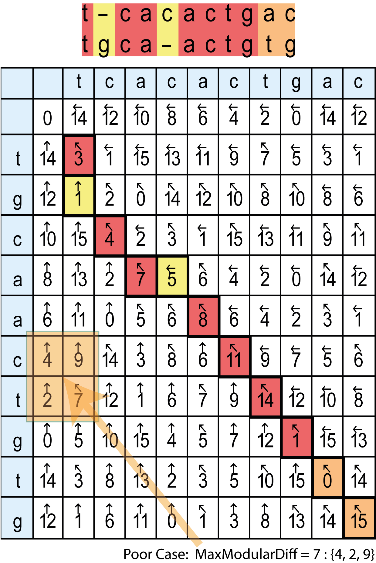
 

Fig. . Matrix H for global PSA with linear gap penalty using integer or mod2N arithmetic.

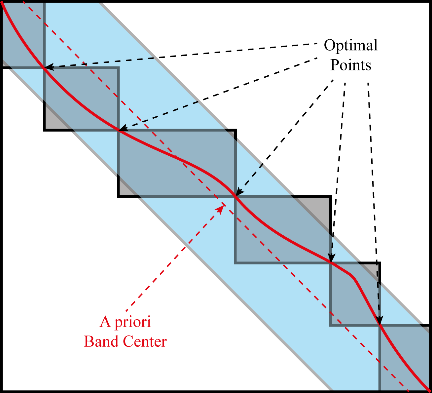


Fig. . Optimization methods for global PSA. 1) wavefront symmetries. 2) Divide & Conquer. 3) KBand.

|  |
| --- |
| **Algorithm 1** NWA(mod-2N) with linear gap penalty |
|  |

|  |
| --- |
| **Algorithm 2** NWA(mod-2N) with affine gap penalty of two matrices |
|  |

Hardware-oriented algorithms for global PSA with linear gap and global PSA with affine gap of two matrices and their associated PE designs were proposed in []. The algorithms are described in Algorithms 1 and 2, and the PE architectures are show in Figures 2 and 3, respectively. These algorithms and PE designs using arithmetic mod-2N allow performing sequences of any length, and reduce and fix the number of bits N of the PEs, considering the maximum difference D between all compared data {HD, HU, HL} by the function max [\*].

Finally, three methods are commonly used to speed up the time of global PSA algorithms and reduce the storage requeriments. 1) Wavefront algorithms for parallel processing of all cells of a same diagonal, considering only the last-processed two diagonal rows. 2) “Divide and Conquer” method (D&C) for performing global PSA between optimal points [3,6]. 3) KBand method for computing only a diagonal band segment of K diagonal rows of the fully matrices, assuming that the optimal PSA is located within this band [7]. Then, the KBand-based PSA is optimal if the optimal PSA is within the band. KBand method reduce the space and time complexity from O(n2) to O(n).

# 3 KBand-Based Global PSA of DNA using NWA(mod-2N)

In this section, we firstly present the codesign algorithm for performing the global PSA’s FWP accelerated by a KBP. Next, we explain the operation principles of the KBP. Finally, we expose the software algorithm for performing the global PSA’s TBP.

## 3.1 Codesign for Forward Process

The algorithm of the codesign application is illustrated in Figure 5 using eigth steps:

**Inicialization**

1. Sequences S1 and S2, with difference between their sequence lengths m=Length[S1] and n=Length[S2] less than the number of PEs K, are loaded from files in FASTA single-line format.

|  |
| --- |
| **Algorithm 3**. Forward process on co-design |
| **Input**: S1, S2, noPEs, dimP  **Output**: ARF[1], ARF[2]… ARF[noARF] |

Fig. 3. Fig. 5 (update). Graphic representation of Algorithm 3

1. Zeros ZL and ZR are added to the shortest sequence to make equal their lengths (ZR+ZL = |m-n|), where ZL are the left-zeros and ZR are the right-zeros. Then, the KBand center is determined by the zero’s distribution. By default, ZL≅ZR for alingning the KBand center with the a priori alignment center (See Figure 2).
2. Sequences with zeros S1’ and S2’ are divided in packets with p sequence symbols for processing them, considering that the hardware logic resources are finites and p≥(K+L)/2, where K the latency of the KBP for loading of the sequences and L=6 is an additional latency of the KBP for generating the arrows. Then, the length of the last packet is n mod p, where n is the length of the sequences S1’ and S2’. In this case, sequences S1’ and S2’ are divided in 3 packets.
3. A packet PZ with zeros and length K+L is also added to the end of both sequences because the latency of the KBP, for allowing the processing of the last symbols of the sequences S1 and S2. Therefore, the number of input packets increases to 4.

**Loop: Repeat steps 5 to 8**

1. The packet of sequences PS[i] is sent to KBP through dedicated DMA streaming interfaces of 8-bit, one for each sequence, considering sequences symbols in ASCII.
2. KBP performs the NWA(mod-2N) progressively, processing the PS[i] as if they were one. Then, the processing of each PS[i] starts from the last processing state of the PS[i-1].
3. The packet of arrows PA[i] is read from KBP throught a dedicate DMA streaming interface of 128-bit. Each PA[i] is sorted in rows, where each row of arrows RA[j] corresponds to an antidiagonal row of arrows generated by the KBP in the same clock cycle. Then, each RA[j] is read in 2K/128 clocks, considering arrows codified with 2-bit and a number of PEs K=2N, where N>7. The number of RA[j] in each PA[i] is the twice to the correspond packet length, except the first and last PA[i], which is 2P-K-L for the PA[1] and K+L for the PA[4], in this case.
4. All PA[i] are concatenated as if they were one, then they are stored in files on a SD card or a HDD to perform the TBP later, where the files have maximum 100MB to allow portability.

## 3.2 Operation Principles of the KBP

TABLE I

PE’s Shift Registers. Start Of Alignment   
S1=ACTGT…GTCAAT and S2=CTGAT…ATCACT

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Clock | PE1 | PE2 | PE3 | PE4 | PE5 | PE6 | PE7 |
| **START** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | C |
| **2** | A | 0 | 0 | 0 | 0 | 0 | C |
| **3** | A | 0 | 0 | 0 | 0 | C | T |
| **4** | C | A | 0 | 0 | 0 | C | T |
| **5** | C | A | 0 | 0 | C | T | G |
| **6** | T | C | A | 0 | C | T | G |
| **7** | T | C | A | **C** | T | G | A |
| **8(begin)** | G | T | C | **AC** | T | G | A |
| **9** | G | T | **CC** | **AT** | G | A | T |
| **10** | T | G | **TC** | **CT** | **AG** | A | T |

TABLE II

PE’s Shift Registers. End of alignment   
S1=ACTGT…GTCAAT and S2=CTGAT…ATCACT

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Clock | PE1 | PE2 | PE3 | PE4 | PE5 | PE6 | PE7 |
| **-5** | A | **TT** | **AC** | **AA** | **CC** | **TT** | G |
| **-4** | A | T | **TC** | **AA** | **AC** | **CT** | T |
| **-3** | T | C | **TA** | **AC** | **AT** | C | T |
| **-2** | T | C | A | **TC** | **AT** | A | C |
| **-1** | C | A | C | **TT** | A | A | C |
| **STOP** | C | A | C | T | T | A | A |

TABLE III

Initialization Function for linear gap  
S1=ACTGT…GTCAAT and S2=CTGAT…ATCACT

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| #Cycle | PE1 | PE2 | PE3 | PE4 | PE5 | PE6 | PE7 |
| **6** | w | w | w | w | w | w | w |
| **7** | w | w | w | **W** | **W** | w | w |
| **8** | 2w | 2w | **2W** | AC | **2W** | 2w | 2w |
| **9** | 3w | 3w | **3W** | CC | AT | **3W** | 3w |
| **10** | 4w | **4W** | TC | CT | AG | **4W** | 4w |

Operation principles of the KBP are illustrated in Figure 6 for the global PSA with linear gap penalty case, which they are equivalent for the global PSA with affine gap penalty case. From Figure 6, it is important to mention the following:

1. KBP is represented by a black diagonal row of K=7 PEs, considering a SAA as the core of the KBP that processes simultaneously an anti-diagonal row for each clock cycle.
2. DNA Symbols of the sequences S1’ and S2’ (ASCII, 8-bit) are codified to 3-bit, obtaining the sequences bS1 and bS2, as it is shown in Table X.

|  |  |
| --- | --- |
|  |  |

Fig. 4. SAA-based Processor. Components and interrelations.

1. Sequences bS1 and bS2 are loaded in the SAA through shift registers with opposite dataflow, alternating between horizontal shifts (H-shift) and vertical shifts (V-shift), in order to generate a diagonal wavefront effect, as it is shown in Figure 6. Then, H-shifts are obtained by left-load of bS1 symbols and V-shifts by right-load of bS2 symbols, as it is presented in Table 1.
2. Zeros are loaded in the shift registers after sequences bS1 and bS2 are already loaded, in order to KBP processes the last symbols of the sequences bS1 and bS2.
3. The examples presented in Figure 6 and Tables 1, 2 and 3 consider the case ZL=0, where the PE that processes the FMC (PEFMC) is the most central PE of the KBP. However, the examples for the case ZL≠0 can be obtained from the above figures and tables by adding ZL zeros to the left (or replacing with zeros the first ZL symbols) of the sequence S1 or S2, which the position of the PEFMC is shifted-left or shifted-rigth ZL positions when ZL≠0 for the sequence S1 or S2, respectively.
4. The initialization function is extended to all cells outside of the matrix H (See Figure 6) for simplifying its hardware implementation. Then, PEs with S1 and S2 valid symbol process the NWA for their associed matrix cell, whereas PEs with S1 or S2 invalid symbols process the initialization function, adding the value g (gap penalty) to the previous value H, as it is shown in Table 3.

TABLE IV

Initialization Function for affine gap  
S1=ACTGT…GTCAAT and S2=CTGAT…ATCACT

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| #Cycle | PE1 | PE2 | PE3 | PE4 | PE5 | PE6 | PE7 |
| **6** | d | d | d | d | d | d | D |
| **7** | d | d | d | **D** | **D** | d | D |
| **8** | d+e | d+e | **D+E** | AC | **D+E** | d+e | d+e |
| **9** | d+2e | d+2e | **D+2E** | CC | AT | **D+2E** | d+2e |
| **10** | d+3e | **D+3E** | TC | CT | AG | **D+3E** | d+3e |

|  |
| --- |
| **Algorithm 1**. Traceback process on software |
| **Input**: S1, S2, noPEs, noARF, ARF[ ]  **Output**: nS1, nS2, simil, dist   1. Load[S1], Load[S2], Load[ARF[noARF] 2. i = Length[S1], j = Length[S2], CF =noARF 3. pR = (8\*noBytes[ ARF[CF] ])/(2\*noPEs) 4. sD = ShiftD [ ARF[CF][pR] ] 5. pA = ArrowPos[ ARF[CF][pR-1] ]; pR=pR-1;   **while** i != 0 or j != 0 **do**  **if** pR<=0 **then**   1. a) Close ARF[CF] and Load ARF[--CF]   b) pR = pR + (8\*noBytes[ ARF[CF] ])/(2\*noPEs)  **end if**   1. lastA=Arrow; Arrow=GetArrow[pR, pA, ARF[CF]] 2. Arrow=DecodeArrow[Arrow, i, j, S1[i], S2[j], lastA] 3. {sym1,sym2} = GetAlignSymbols[S1[i], S2[j]] 4. AddAlignSymbols[nS1, nS2, sym1, sym2] 5. {simil,dist}=UpdateSimilarityDistance[sym1,sym2] 6. {i,j}=GetNewAlignIndexes[Arrow,i,j] 7. {pR,pA,sD}=GetNewArrowPosSD[Arrow,pR,pA,sD]   **end while** |

1. PE interconnections are determined by the data dependency, which is shown in Figure 3-a. Considering a PE[i] with four inputs {HD, HU, HL, Init} and one output H, then: 1) HD[i] is generated with 2-delay clock by H[i]. 2) HU[i] is generated with 1-delay clock by H[i+1] for H-shifts and by H[i] for V-shifts. 3) HL[i] is generated with 1-delay clock by H[i] for H-shifts and by H[i-1] for V-shifts.4) Init[i] is generated with 1-delay clock by H[i].
2. PE interconections H[0]=H[1]-1 and H[n+1]=H[n]-1 are used in the first and last PEs, PE[1] and PE[n] respectively, for these PEs only choose signals from whitin the band.
3. The alignment zone (AZ) corresponds to all processed rows between the first matrix cell (FMC) and the last matrix cell (LMC), where at least one PE is processing a matrix cell. In this zone are updated the registers associed to matrices H and A in each clock cycle.
4. The AZ is detected dynamically by the flag FAZ using the flag Valid of each PE, independent of the pair-sequence length, considering that the all PEs have S1 or S2 invalid symbols when the processing is before or after of the AZ.
5. The FMC is processed later of K+ZL clock cycles, where it is the first time when the aligned sequences are intercepted (See Table 1, PE4, eight-clock cycle). The LMC is processed later of n+m-1 clock cycles counted from the processing of the FMC (See Table 1, PE4, minus-one-clock cycle), where m=Length[S1] and n=Length[S2]. These number of clocks must be taking into account by the codesign application.
6. KBP processes the sequences by packets. When a packet is finished, KBP is stoped and waits for the next packet, which is processed by KBP from the last processing state.
7. Arrow’s row files (ARFs) are used to store the arrow’s rows for being processed by TBP. Several ARFs of maximum 50MB are used in order to optimize the storage and portability in different file systems, and they are managed as a single continuous file. In this case, ARF[1], ARF[2] and ARF[3] are used to store arrows from 100MB to 150MB.
8. The first arrow’s row (FAR) of the first ARF and the last arrow’s row (LAR) of the last ARF have only one arrow, which are the arrows of the FMC and LMC, respectively.
9. The last shift direction (LSD) used in the FWP to process the LMC, is stored after the last arrow’s row for synchronizing the TBP.

## 3.3 Software for Traceback Process

Reverse order. TBP is presented in the Algorithm 1 and illustrated in Figure 3-b. TBP is performing using the arrows generated by FWP and the processed pair-sequences S1 and S2, considering the number of PEs used by KBP in FWP (noPEs) and the number or ARFs generated by KBP in FWP (noARF). The first five instructions correspond to initialization tasks, whereas the other instructions (Loop) are performed until the full alignment is completed, that is, until variables I and J are both equal to zero.

**Initialization**

Arrows are stored in several ARFs for PSAs of very large sequences for optimizing the storage and portability, since the ARFs are managed as a single continuous file. In this case, arrows are stored on 3 ARFs and sorted by rows, where each arrow’s row corresponds to one processed diagonal row.

1. Sequences S1 and S2 are loaded from FASTA-singleline files, just like the last ARF (ARF[3] for this case).
2. Variables I and J indicate the indexes of the current aligned base-pair sequence symbols, and they are initialized with the sequence length of the S1 and S2, that correspond to the indexes of the last symbols of S1 and S2. Variable CF indicates the index of the current processed ARF, and it is initialized with the index of the last ARF (CF=noARF=3 for this case).
3. Variable pR indicates the index of the ARF’s row of the current processed arrow, and it is initialized with the index of ARF’s row with LSD, that corresponds to the number of rows of the last ARF, which is calculated considering the number of bytes of the last ARF and noPEs.
4. Variable sD indicates the shift direction associed to the current processed arrow, and it is initialized with LSD of FWP (V-Shift for this case), which is determinated by function ShiftD.
5. Variable pA indicates the arrow’s position in the ARF’s row of the current processed arrow, and it is initialized with the position of the LMC’s arrow (unique arrow in the LAR), which is determinated by function ArrowPos.

**While Loop**

1. pR<=0 indicates that the arrow’s row is in the next ARF. Then, the current ARF is closed, the next ARF is loaded and pR is increased whit the number of arrow’s rows in the next ARF, when pR<=0.
2. Function GetArrow update the variable Arrow with the current arrow (2-bit), which location is determined by pR and pA.
3. Function DecodeArrow replace the arrow codification when its code is “00”, considering five cases: 1) If index I=0, then Arrow=up, since S1 is fully processed. 2) If index J=0, then Arrow=left, since S2 is fully processed. 3) If I≠0 and J≠0 and S1[I]=S2[J], then Arrow=diag, giving priority to Arrow=Diag case when pair-sequence symbols are matched. 4) Otherwise, Arrow=Arrow. Additionally, If lastA=Diag (Previous value of Arrow) and S1[I]=S2[J], then Arrow=Diag, extending the matched-block aligned when the arrows indicate gaps incorrectly, which is an issue presented in PSA with affine gap penalties.
4. Function GetAlignSymbols determines the current-pair-symbols {sym1,sym2} to be aligned, which are {S1[i],S2[j]} when Arrow=diag, {S1[i],\_} when Arrow=left or {\_,S2[j]} when Arrow=up.
5. Function AddAlignSymbols add the symbol sym1 to the aligned sequence nS1 and the symbol sym2 to the aligned sequence nS2.

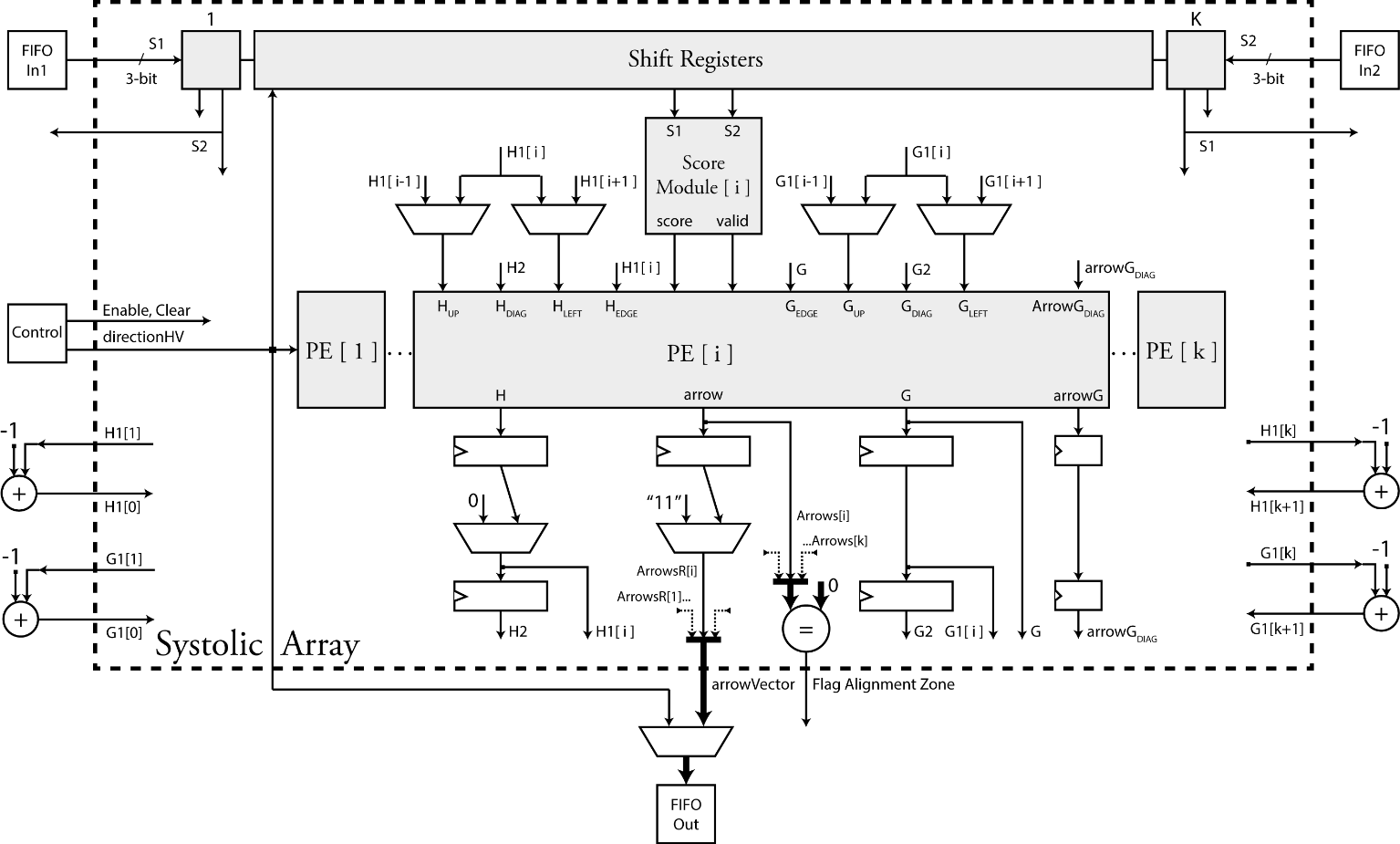


Fig. 5. SAA-based Processor. Components and interrelations.

1. Function UpdateSimilarityDistance calculates gradually the similarity and the edit-distance between the sequences S1 and S2, for obtaining the aligned sequences nS1 and nS2. Similarity is calculated by increasing one the variable simil when sym1=sym2 and decreasing one it when sym1≠sym2. Edit-distance is calculated by increasing one the variable dist when sym1≠sym2.
2. Function GetNewAlignIndexes decrease the indexes i and j when Arrow=diag, only decrease the index i when Arrow=left or only decrease the index j when Arrow=up.
3. Function GetNewArrowPosSD update the variables pR, pA and sD with the location of the next arrow and its associated shift direction, taking into account the data dependency presented in Figure 3-a, which is different for both H-shifts and V-shifts. In this case, the first arrow is LEFT, it is located in ARF3[i,j] and its shift direction is vertical (V). Then, the second arrow is LEFT, it is located in ARF3[i-1,j-1] and its shift direction is horizontal (H).

# 4 Design of Embedded System for KBand-Based global PSA’s

In this section, we present the hardware architecture of the KBP and the co-design embedded system used to process the KBand-based global PSA of very large pair-sequences by the co-design application for performing FWP, and by the software application for performing TBP.

## 4.1 KBP

KBP is designed to perform the global PSA’s FWP. KBP architecture for performing global PSA’s FWP with affine gap penalty is shown in Figure 5. It is composed by two input buffers, one output buffer (Bottleneck), a control unit, a SAA (Core) and edge components. KBP architecture for performing global PSA’s FWP with linear gap is similar, but the hardware design for processing the signals associed to matrix G is omitted because its PEs do not process the matrix G.

**Input Buffers**

The input buffers synchronize the data-load of the ASCII symbols of the sequences S1 and S2 on the KBP’s from the DMA input interfaces through of two streaming inputs of 8-bit. Firstly, these 8-bit symbols are codified on 3-bit (See Figure 4), then they are written on a FIFO memory of 3-bit, where the writing operates with the HFC and the reading operates with the LFC.

**Output Buffer**



Fig 6. FSM core of the streaming processor.

The output buffer synchronizes the KBP’s data-read of the arrow’s row with the DMA output interface through of a streaming output of 128-bit. Firstly, the data-output of 2K-bit (Arrow’s row) is written on a FIFO memory, then the arrow’s row is read by segments through the data output of 128-bit, where the writing operates with the LFC and the reading operate with the HFC. Therefore, this FIFO memory is designed with four-stages of metastability, considering that it works with two clock domains and different number of bits for input and outputs. The shift-direction also is stored through the output buffer.

**KBP’s Bottleneck**



Fig 7. PE for NWA with linear gap penalty.



Fig 8. PE for NWA with affine gap penalty of two matrices.

Output buffer is the bottleneck of the KBP, because the output throughput of the SAA is very high according to the bandwidth of the level-system interfaces. For example, a KBP with 1024-PEs generates 2048-bit by clock, while the bandwith of the HPS-to-FPGA AXI interface is 128-bit, then are required 16-clock to read one arrow’s row. Therefore, KBP was designed to work with two clock domains: the low-frequency clock (LFC) to operate the SAA and the high-frequency clock (HFC) to operate the input/output interfaces, increasing the throughput at the system level, with external components as processors, memory among others. For example, using HFC=100MHz and LFC=50MHz, the KBP’s bottleneck is reduced to 50%.

**Control Unit**

The control unit synchronizes the LFC domain of the input/output buffers, and the SAA. For achieving it, the control unit uses the FSM of 7-state presented in Figure X. This FSM allows to KBP the performing of PSA’s FWP using packets, which fixes the hardware requeriments.

1) IDLE. In this state, the KBP wait for packets when the first valid alignment has not been detected (Clocks 1 to 7 on Table 1).

2) WAIT. In this state, the KBP also wait for packets, but after the first valid alignment has been detected (After clock 8 on Table 1).

3) RUN. In this state, the symbols of the sequences S1 and S2 are progressively charged in the shift registers of the SAA (See Figure 2) when the first valid alignment has not been detected. Therefore, the values of matrix-H and matrix-A are not stored.

4) RUN&SAVE. In this state, the symbols of the sequences S1 and S2 also are progressively charged in the shift registers of the SAA, but after the first valid alignment has been detected. Therefore, the values of matrix-H and matrix-A are stored.

5) PAUSE. In this state, the KBP is paused when the output buffer is full for avoiding the loss of data.

6) SAVEDIR. In this state, the shift direction is stored on the output buffer, considering that the PSA’s FWP has already been completed.

7) RESET. In this state, the registers of the KBP are reset in order to perform the next PSA.

**SAA**

SAA is described in Section 4.2

**Edge Components**

The SAA can be implemented on several FPGAs by connecting with other itself instances, allowing to distribute the complete SAA for increasing the width of the KBand.

According to literature [\*], missing connections presented at the edges of the KBand (First and last PEs) should be connected to INF (impractical value in modular arithmetic) to prevent that these cases will be chosen, limiting the processing space to the KBand. In our design, the missing connections are connected as indicate the Figure 4, generating dynamic values very close to the other score input values (HD, HU, HL, HEDGE), minimizing the difference between them to optimize the use of modular arithmetic.

## 4.2 SAA

SAA is the core of the of the KBP for performing the FWP of the KBand-based global PSA with linear gap, in this case. SAA architecture can be classified in: shift registers, PEs, interconnection logic for data dependency, flag FAZ, and the muxes of control muxH and muxA.

**Shift Registers**

Two Shift Registers are used for loading alternately the sequences S1 and S2, as is described in Section 3.1, in order to change the shift direction between horizontal and vertical shift for generating a diagonal wavefront effect. The outputs oS1 and oS2 of each each module SReg[i] have the pair-symbols to align.

**PEs**

PE architectures for the linear gap and affine gap (this case) penalties are shown in Figures 7 and 8., and them functional details are presented in [].

**Interconnection logic for data dependency**

The interconnection logic for data dependency is designed according the numeral 7 of Section 3.2. For the PE[i] case, delays are generated for the signals H, G and arrowG, and muxes are used for selecting the data source for the inputs HUP and HLEFT with 1-delay. Inputs HDIAG and Init are connected from output H with 2-delay and 1-delay, respectively.

**Flag FAZ**

Flag FAZ=False when all flags Valid[i] of all PE[i] have the value False, that is, FAZ=OR[Valid[1]…,Valid[k]].

**MuxH**

MuxH is used for replacing the output of the register H1 whit the value zero, because the register H1 is enabled for all PEs before the AZ for performing the initialization functions of the matrices H and G for the affine gap case. However, the muxH is not required for the linear gap case because the register H1 is disabled for all PEs before the AZ for the linear gap case.

**MuxA**

MuxA is used to replace the arrow-code “00” (diag-up or diag-left) with the arrow-code “11” (diag) for the arrow of the LMC (If ALMC=“00” then ALMC=“11”), because ALMC is the unique valid arrow in the LAR of the AZ, then the ALMC only can be found if ALMC≠“00”.

## 4.3 Co-design Embedded System

A codesign embedded system was developed for processing the codesign application for FWP of KBand global PSA, where KBPLINEAR is used to speed-up the NWA with linear gap penalty, whereas KBPAFFINE is used to speed-up the NWA with affine gap penalty of two matrices. Software application for TBP of KBand global PSA also can be processed in the codesign embedded system.

In order to verify the designed processors, the KBPs were integrated on a low-cost SoC-FPGA embedded system, where the system-level hardware-design was developed using Qsys tool, the RTL hardware design using the hardware description languages VHDL and Verilog, and the codesign and software applications were developed in C language. The block diagram of the codesign embedded system is shown in Figure 9, where the KBP block can be KBPLINEAR or KBPAFFINE.

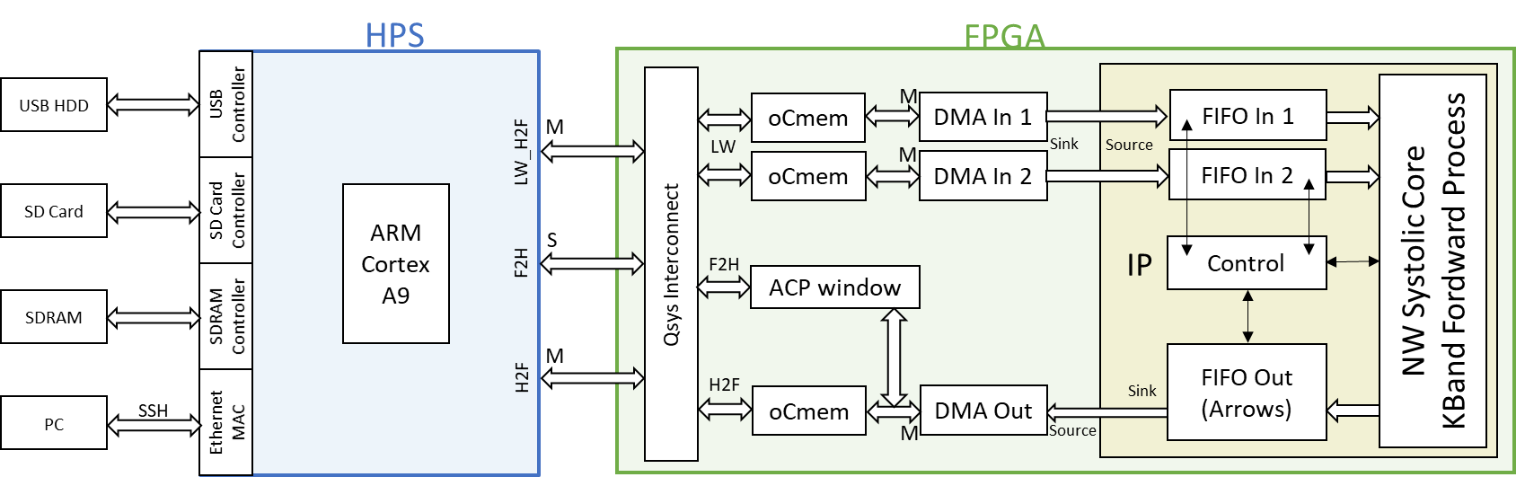


Fig. 9. The same Global PSA using integers (Left) and arithmetic mod-2N (Rigth). For the case {match, mismatch, gap} = {3,-1,-2}, then D=7 and N =4 [\*].

The SoC-FPGA embedded system consists of two parts: A Hard Processor System (HPS) and an FPGA, interconnected them through HPS AXI interfaces. The main HPS component is a general-purpose processor ARM Cortex A9 of two-cores, which allows to run software applications under Linux OS. The main FPGA component is the KBP, and it is integrated to HPS through AXI interfaces. In this case:

1) HPS to FPGA interface (H2F) allows sending each packet of arrows from FPGA on-chip memories oCmem3 to HPS memory.

2) HPS to FPGA LightWeight interface (H2F LW) allows sending each packet of sequences PS[i] from HPS memory to oCmem1 and oCmem2, where symbols of sequences S1 and S2 are stored in oCmem1 and oCmem2, respectively.

3) H2F LW also allows controlling the DMA controllers DMAin1 and DMAin2 for sending the packets of sequences PS[i] to KBP from oCmem1 and oCmem2 through dedicated DMA streaming interfaces of 8-bit, and allow controlling DMAin3 for sending the packets of arrows PA[i] from KBP to oCmem3 through a dedicated DMA streaming interface of 128-bit.

4) FPGA to HPS interface (F2H) allows direct access to HPS memory space from the FPGA through an ACP window, in order to allow futures optimizations in management memory by the codesign application.

…………………………………………………………….

# 5 Results

## 5.1 Synthesis Results of designed processor only

Table IV. Designed processors on CycloneV. Maximum number of H-bits (#bitsH) by number of PEs.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| #PEs | #BitsH | ALMs  Linear | Freq  (Mhz) | #PEs | ALMs  Affine | Freq  (Mhz) |
| 2048 | 3 | 40996 | 123/206 | - | - | - |
| 1024 | 5 | 34829 | 83/241 | 4 | 40077 | 88/229 |
| 512 | 12 | 35897 | 76/275 | 8 | 34528 | 78/241 |
| 256 | 28 | 37099 | 68/275 | 16 | 32385 | 74/256 |

Table V. Designed processors on Cyclone10 GX. Maximum number of H-bits (#bitsH) by number of PEs.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| #PEs | #BitsH | ALMs  Linear | Freq  (Mhz) | #BitsH | ALMs  Affine | Freq  (Mhz) |
| 4096 | 3 | 73139 | 71/307 | - | - | - |
| 2048 | 7 | 77555 | 68/304 | 4 | 78413 | 80/306 |
| 1024 | 16 | 77592 | 57/440 | 9 | 61068 | 80/452 |
| 512 | 32 | 74829 | 61/520 | 20 | 74037 | 76/524 |
| 256 | 64 | 71455 | 78/599 | 40 | 71455 | 78/599 |

Table VI. Comparison with other similar SAA-based designs.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Design** | **Device** | **#PEs (K)** | **Frequency** | **#BitsH (word)** | **Length** | **KBand** |
| [11], [26] | Virtex-II 6000-6 | 75 | 66-MHz | 11 | 2-K | SW Affine |
| [27] | EP2S180F1508C5 | 350 | 42-MHz | - | 350 | No |
| [28] | EP1S30 | 80 | 82-MHz | 20 | 80-K | No |
| [29] | XC5VLX110 | 195 | 162-MHz | 11 | 192 | No |
| [12] | EP4SGX230KF40C2 | 128 | 125MHz | 12 | 10-K | SW Affine |
| Our | 5CSEBA6U23I7NDK | 2048 | 123/206-MHz | 3 | Any | NW Linear |
| Our | 5CSEBA6U23I7NDK | 1024 | 88/229-MHz | 4 | Any | NW Affine |
| Our | 10CX220YF780E5G | 4096 | 71/307-MHz | 3 | Any | NW Linear |
| Our | 10CX220YF780E5G | 2048 | 80/306-MHz | 4 | Any | NW Affine |

Designed processors were synthetized using two FPGA-based platform: DE10-nano and Cyclone 10 GX. DE10-nano is based on a low-cost Intel Cyclone-V SoC-FPGA (5CSEBA6U23I7NDK), with 42K-ALMs operating to maximum 300 MHz, and an ARM dual-core hard processor operating to maximum 800MHz. Cyclone 10 GX is based on an Intel FPGA Cyclone-10 (10CX220YF780E5G), which has more logic resources (80K-ALMs) and performance (max 600MHz).

Tables 4 and 5 present the synthesis results for configurations with 2N PEs and the scoring schemes {match, mismatch, gap} = {1, -1, -2} and {match, mismatch, gapO, gapE} = {1, -1, -2, -1}. For a CycloneV SoC-FPGA, 2048 PEs of 3-bits for linear gap and 1024 PEs of 4-bits for affine gap are reached. For a Cyclone 10 GX, 4096 PEs of 3-bits for linear gap and 2048 PEs of 4-bits for affine gap are reached. These synthesis results only correspond to the designed processors; the results can change when these components are integrated on an embedded system, mainly by changes presented in the fitter stage.

Table 6 compares our synthesis results with other reported works, which no use arithmetic mod 2N, highlighting the increment in the PE density and the maximum sequence’s length (Virtually infinite because the arithmetic mod 2N and the processing by packets).

## 5.2 Co-design Embedded System vs Simulation

Designed processors were integrated on a low-cost SoC-FPGA embedded system, as it is shown in Figure 7.

Designed processors have been verified on a DE10-nano and configured with 1024-PEs of 4-bits. For this case, the design uses 29K-ALMs (70%) for linear gap and 38K-ALMs (90%) for affine gap, operating to a maximum internal/external frequency of 57MHz/104MHz for linear gap and 56MHz/111MHz for affine gap. Therefore, the embedded system was configurated to 50MHz internal frequency (SAA) and 100MHz external frequency (System level), reducing the bottleneck of the SAA to 50%. 320KB (47%) of cache on-chip memory was used, distributed by 32KB to each input sequence and 256KB to the arrows. FPGA component was developed using Quartus, Qsys and SignalTap Intel tools. Software component was developed in C using the Intel SoC FPGA Embedded Development Suite.

Designed processors are used for accelerating FWP of the KBand-based global PSA and they are managed by the same co-design application for both linear and affine gap, because the processor interfaces are identicals. TBP does not need accelerate, so it is performing by using a full-software application. Results were compared with the obtained by software implementations developed in Wolfram Mathematica running on a workstation Dell 7910 which it has two 2.2GHz Intel® Xeon® E5-2630 v4 processors of 10 cores each one and 64GB of RAM. Both embedded and PC platforms was tested using sequences-pairs taken from NCBI as proteins, genomes and chromosomes, distributed logarithmically in a range from 1-Kbp to 30-Mbp (Larger sequences is possible), which are reported in the Table 7.

Table 8 reports the performance comparison between the co-design embedded systems and their software simulations, which only were performed until 1-Mbp because the high memory requeriments for storing the arrows on RAM only. In average, FWP is performing 58x faster to linear gap and 427x to affine gap using the co-design solution, and TBP is performing 6.9x faster.

# 6 Conclusions

Table IX. Performance comparison between software and SoC-FPGA implementations

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Software [s] | | | SoC FPGA [s] | | Ratio Soft/SoC | | |
| Seqs  A-B | **Forward**  **Linear** | **Forward**  **Affine** | **Traceback** | **Forward**  **Both** | **Traceback** | **Forward**  **Linear** | **Forward**  **Affine** | **Traceback** |
| 1 | 6.718 e0 | 4.211 e1 | 0.078 | 1.07 e-1 | 1.11 e-2 | 62.78 | 393.5 | 7.03 |
| 2 | 1.587 e1 | 1.223 e2 | 0.110 | 2.84 e-1 | 2.85 e-2 | 55.88 | 430.6 | 3.86 |
| 3 | 4.446 e1 | 3.627 e2 | 0.312 | 7.86 e-1 | 8.16 e-2 | 56.56 | 461.4 | 3.82 |
| 4 | 1.337 e2 | 8.212 e2 | 1.797 | 2.311 e0 | 2.50 e-1 | 57.85 | 355.3 | 7.19 |
| 5 | 4.543 e2 | 3.056 e3 | 6.953 | 7.591 e0 | 8.33 e-1 | 59.85 | 402.6 | 8.35 |
| 6 | 1.351 e3 | 1.075 e4 | 20.265 | 2.290 e1 | 2.266 e0 | 59.00 | 469.4 | 8.94 |
| 7 | 4.219 e3 | 3.676 e4 | 71.324 | 7.693 e1 | 7.864 e0 | 54.84 | 477.8 | 9.07 |
| 8 | - | - | - | 2.236 e2 | 2.566 e1 | - | - | - |
| 9 | - | - | - | 7.725 e2 | 8.723 e1 | - | - | - |
| 10 | - | - | - | 2.312 e3 | 2.625 e2 | - | - | - |

Table VII

Pair of sequences used to test the design. Uncoded symbols (“N”) have been omitted in the alignments.

|  |  |  |
| --- | --- | --- |
| ID | Accession Number (NCBI) | Length |
| 1-A | NM\_001354943.1 | 1,236 |
| 1-B | NM\_001354944.1 | 1,062 |
| 2-A | NM\_001354925.1 | 3,368 |
| 2-B | NM\_001354924.1 | 3,431 |
| 3-A | NM\_001011645.3 | 10,257 |
| 3-B | NM\_000044.4 | 10,070 |
| 4-A | PSXO01000091.1 | 30,485 |
| 4-B | PSXO01000097.1 | 30,316 |
| 5-A | PSWL01000013.1 | 100,451 |
| 5-B | PSWJ01000010.1 | 100,053 |
| 6-A | NZ\_PQKM01000006.1 | 300,261 |
| 6-B | NZ\_PQKN01000004.1 | 300,339 |
| 7-A | NZ\_KN849241.1 | 1,000,480 |
| 7-B | NZ\_LJXA01000025.1 | 1,000,381 |
| 8-A | NZ\_CP007019.1 | 3,000,273 |
| 8-B | NC\_017545.1 | 3,000,464 |
| 9-A | CM000111.5 | 10,323,212 |
| 9-B | KZ626786.1 | 10,323,212 |
| 10-A | NW\_004929415.1 | 30,985,389 |
| 10-B | KE150199.1 | 30,985,389 |

In this work, we designed two processors for global PSA using the KBand method and arithmetic mod 2N, one for linear gap and one for affine gap. Processor’s PEs were designed using modular comparators and perform dynamically the initialization functions. Score scheme (Match, mismatch and gaps) can be changed by FPGA reconfiguration (instead of soft reprogramming) for reducing the area requeriments.

Taking into account the results obtained, we can conclude that arithmetic mod 2N maximize the PE density (2048-PEs of 3-bits on CycloneV and 4096-PEs of 3-bits on Cyclone10 GX) because the reduced PE size, and allows to process similar sequence’s pairs of any length, verified up 31Mbp. The use of substractors for performing the modular comparisons and very close dynamic values in the edge of the band was key to achieve it.

The co-design application used to verify the processors on a SoC-FPGA embedded system reaches average speed-up of 58x for linear gap and 427x for affine gap respect to software simulation. Considering that the performance of the processors is 200 times faster than the co-design solution based on it, the global performance could increase by using a more efficient co-design application and high-end platforms (More performance and density) with faster interfaces (USB 3.0, PCIe, gigaE) for transfer the big among of arrows, because the arrow storage is the main bottleneck of the co-desing application.

Finally, the pointer-oriented structure of the arrow’s packets allows to process these packets efficiently by the traceback process software-algorithm, being unnecessary a hardware accelerator for performing it.

**Acknowledgment**

The authors wish to thank Claudia, Oscar, Carlos and Miguel for they are best friends 😊.

**References**

1. J.S. Bridle, “Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition,” *Neurocomputing—Algorithms, Architectures and Applications,* F. Fogelman-Soulie and J. Herault, eds., NATO ASI Series F68, Berlin: Springer-Verlag, pp. 227-236, 1989. (Book style with paper title and editor)
2. W.-K. Chen, *Linear Networks and Systems.* Belmont, Calif.: Wadsworth, pp. 123-135, 1993. (Book style)
3. H. Poor, “A Hypertext History of Multiuser Dimensions,” *MUD History,* http://www.ccs.neu.edu/home/pb/mud-history.html. 1986. (URL link \*include year)
4. K. Elissa, “An Overview of Decision Theory,"unpublished. (Unpublished manuscript)
5. R. Nicole, "The Last Word on Decision Theory," *J. Computer Vision,* submitted for publication. (Pending publication)
6. C. J. Kaufman, Rocky Mountain Research Laboratories, Boulder, Colo., personal communication, 1992. (Personal communication)
7. D.S. Coming and O.G. Staadt, "Velocity-Aligned Discrete Oriented Polytopes for Dynamic Collision Detection," *IEEE Trans. Visualization and Computer Graphics*, vol. 14,  no. 1,  pp. 1-12,  Jan/Feb  2008, doi:10.1109/TVCG.2007.70405. (IEEE Transactions )
8. S.P. Bingulac, “On the Compatibility of Adaptive Controllers,” *Proc. Fourth Ann. Allerton Conf. Circuits and Systems Theory*, pp. 8-16, 1994. (Conference proceedings)
9. H. Goto, Y. Hasegawa, and M. Tanaka, “Efficient Scheduling Focusing on the Duality of MPL Representation,” *Proc. IEEE Symp. Computational Intelligence in Scheduling* *(SCIS ’07)*, pp. 57-64, Apr. 2007, doi:10.1109/SCIS.2007.367670. (Conference proceedings)
10. J. Williams, “Narrow-Band Analyzer,” PhD dissertation, Dept. of Electrical Eng., Harvard Univ., Cambridge, Mass., 1993. (Thesis or dissertation)
11. E.E. Reber, R.L. Michell, and C.J. Carter, “Oxygen Absorption in the Earth’s Atmosphere,” Technical Report TR-0200 (420-46)-3, Aerospace Corp., Los Angeles, Calif., Nov. 1988. (Technical report with report number)
12. L. Hubert and P. Arabie, “Comparing Partitions,” *J. Classification,* vol. 2, no. 4, pp. 193-218, Apr. 1985. (Journal or magazine citation)
13. R.J. Vidmar, “On the Use of Atmospheric Plasmas as Electromagnetic Reflectors,” *IEEE Trans. Plasma Science*, vol. 21, no. 3, pp. 876-880, available at http://www.halcyon.com/pub/journals/21ps03-vidmar, Aug. 1992. (URL for Transaction, journal, or magzine)
14. J.M.P. Martinez, R.B. Llavori, M.J.A. Cabo, and T.B. Pedersen, "Integrating Data Warehouses with Web Data: A Survey," *IEEE Trans. Knowledge and Data Eng.*, preprint, 21 Dec. 2007, doi:10.1109/TKDE.2007.190746.(PrePrint)

**Duverney Corrales-Mendoza** received the B.S. degree in electronics engineering from the Universidad del Valle, Cali, Colombia, in 2015. He is currently pursuing the M.Eng. degree with the Bionanoelectronics Research Group, Universidad del Valle. His research interests include high performance hardware accelerators, co-design and heterogenous platforms and high-performance computer mainly on bioinformatics topic. He is a great chess-player and love play UnderWaterHockey.

**Jaime Velasco-Medina** (SM’96) received the BS degree in electrical engineering from the Universidad del Valle, Cali, Colombia, in 1985, and the M.Sc. and Ph.D. degrees in microelectronics from the Institute National Polytechnic of Grenoble, Joseph Fourier University, Grenoble, France, in 1995 and 1999, respectively. He held an internship position, for six months, as a technical staff member at AT&amp;T Bell Labs in Allentown, PA, USA, in 1988. He was the pioneer of the current-based testing for analog and mixed signal circuits, and on-line testing of operational amplifiers. He is currently a Faculty Professor with the School of Electrical and Electronics Engineering, Universidad del Valle, where he is the Director with the Bionanoelectronics Research Group. He has authored more than 40 IEEE papers and 50 peer-reviewed papers in other scientific events and journals. His current research interests include digital systems design for computer arithmetic and digital signal processing, test of analog circuits, and hardware architectures for cryptography, quantum computing, wireless communications, citocomputation, modeling of biological systems, design of graphene-based digital circuits, and computational design of bionanosensors and bionanomachines for nano drug delivery systems. Dr. Velasco Medina is a Reviewer for the JETTA, the IEEE LATW, the IEEE SPL, the IBERCHIP, the IEEE LASCAS, the IEEE Transactions on VLSI System, the IEEE Transactions on Signal Processing, the IEEE Transactions on Circuits and Systems I, and many other international publications and conferences.

Be sure that the symbols in your equation have been defined before the equation appears or immediately following. Italicize symbols (*T* might refer to temperature, but T is the unit tesla). Per IEEE Computer Society, please refer to “(1),” not “Eq. (1)” or “equation (1),” except at the beginning of a sentence: “Equation (1) shows ... .” Also see *The Handbook of Writing for the Mathematical Sciences*, 1993. Published by the Society for Industrial and Applied Mathematics, this handbook provides some helpful information about math typography and other stylistic matters. For further information about typesetting mathematical equations, please visit the IEEE Computer Society style guide: <http://www.computer.org/portal/web/publications/style_math>.

Please note that math equations might need to be reformatted from the original submission for page layout reasons. This includes the possibility that some in-line equations will be made display equations to create better flow in a paragraph. If display equations do not fit in the two-column format, they will also be reformatted. Authors are strongly encouraged to ensure that equations fit in the given column width.

# 6 Helpful Hints

## 6.1 Figures and Tables

Because IEEE Computer Society staff will do the final formatting of your paper, some figures may have to be moved from where they appeared in the original submission. Figures and tables should be sized as they are to appear in print. Figures or tables not correctly sized will be returned to the author for reformatting.

Detailed information about the creation and submission of images for articles can be found at <http://www.computer.org/portal/web/peerreviewjournals/author#figures> where you can [View transactions art\_guide.pdf (PDF, 4.69MB)](http://www.computer.org/cms/Computer.org/Journal%20templates/transactions_art_guide.pdf) . We strongly encourage authors to carefully review the material posted here to avoid problems with incorrect files or poorly formatted graphics.

Place figure captions below the figures; place table titles above the tables. Figure captions appear as left justified. Table captions are restricted to one sentence and are formatted as title case. Any additional sentence in a table caption will be formatted as a footnote below the table (see Table 1 in this document). If your figure has two parts, include the labels “(a)” and “(b)” as part of the artwork. Please verify that the figures and tables you mention in the text actually exist. Figures and tables should be called out in sequential order, as this is how they will be placed in your paper. For example, avoid referring to figure “8” in the first paragraph of the article unless figure 8 will again be referred to after the reference to figure 7. **Please do not include figure captions as part of the figure. Do not put captions in “text boxes” linked to the figures. Do not put borders around the outside of your figures.** Per IEEE Computer Society, please use the abbreviation “Fig.” even at the beginning of a sentence. Do not abbreviate “Table.” Tables are numbered numerically.

For journals that use print for publication, please verify with IEEE Computer Society that the journal you are submitting to does indeed accept color before submitting final materials. **Do not use color unless it is necessary for the proper interpretation of your figures.**

Figures (graphs, charts, drawing or tables) should be named fig1.eps, fig2.ps, etc. If your figure has multiple parts, please submit as a single figure. Please do not give them descriptive names. Author photograph files should be named after the author’s LAST name. Please avoid naming files with the author’s first name or an abbreviated version of either name to avoid confusion. If a graphic is to appear in print as black and white, it should be saved and submitted as a black and white file (grayscale or bitmap.) If a graphic is to appear in color, it should be submitted as an RGB color file.

Figure axis labels are often a source of confusion. Use words rather than symbols. As an example, write the quantity “Magnetization,” or “Magnetization *M*,” not just “*M*.” Put units in parentheses. Do not label axes only with units. As in Fig. 1, for example, write “Magnetization (A/m)” or “Magnetization (Am−1),” not just “A/m.” Do not label axes with a ratio of quantities and units. For example, write “Temperature (K),” not “Temperature/K.” Table 1 shows some examples of units of measure.

Multipliers can be especially confusing. Write “Magnetization (kA/m)” or “Magnetization (103 A/m).” Do not write “Magnetization (A/m) × 1,000” because the reader would not know whether the top axis label in Fig. 1 meant 16,000 A/m or 0.016 A/m. Figure labels should be legible, approximately 8 to 12 point type. When creating your graphics, especially in complex graphs and charts, please ensure that line weights are thick enough that when reproduced at print size, they will still be legible. We suggest at least 1 point.

## 6.3 Footnotes

Number footnotes separately in superscripts (Insert | Footnote)[[1]](#footnote-1). Place the actual footnote at the bottom of the column in which it is cited; do not put footnotes in the reference list (endnotes). Use letters for table footnotes (see Table 1). Please do not include footnotes in the abstract and avoid using a footnote in the first column of the article. This will cause it to appear above the affiliation box, making the layout look confusing.

## 6.4 Lists

The IEEE Computer Society style is to create displayed lists if the number of items in the list is longer than three. For example, within the text lists would appear 1) using a number, 2) followed by a close parenthesis. However, longer lists will be formatted so that:

1. Items will be set outside of the paragraphs.
2. Items will be punctuated as sentences where it is appropriate.
3. Items will be numbered, followed by a period.

## 6.5 Theorems and Proofs

Theorems and related structures, such as axioms corollaries, and lemmas, are formatted using a hanging indent paragraph. They begin with a title and are followed by the text, in italics.

**Theorem 1.** *Theorems, corollaries, lemmas, and related structures follow this format. They do not need to be numbered, but are generally numbered sequentially.*

Proofs are formatted using the same hanging indent format. However, they are not italicized.

**Proof.** The same format should be used for structures such as remarks, examples, and solutions (though these would not have a Q.E.D. box at the end as a proof does). 

# 7 End Sections

## Appendices

Appendices, if present, appear online as supplemental material. In the event multiple appendices are required, they will be labeled “Appendix A,” “Appendix B, “ etc.

IEEE Computer Society Transactions accepts supplemental materials for review with regular paper submissions. These materials may be published on our Digital Library with the electronic version of the paper and are available for free to Digital Library visitors. Please see our guidelines below for file specifications and information. Any submitted materials that do not follow these specifications will not be accepted. All materials must follow US copyright guidelines and may not include material previously copyrighted by another author, organization or company. More information can be found at <http://www.computer.org/portal/web/peerreviewjournals/author#supplemental>.

## 7.2 Acknowledgments

The preferred spelling of the word “acknowledgment” in American English is without an “e” after the “g.” Use the singular heading even if you have many acknowledgments. Avoid expressions such as “One of us (S.B.A.) would like to thank ... .” Instead, write “F. A. Author thanks ... .” Sponsor and financial support acknowledgments are included in the acknowledgment section. For example: This work was supported in part by the US Department of Commerce under Grant BS123456 (sponsor and financial support acknowledgment goes here). Researchers that contributed information or assistance to the article should also be acknowledged in this section. Also, if corresponding authorship is noted in your paper it will be placed in the acknowledgment section. Note that the acknowledgment section is placed at the end of the paper before the reference section.

## 7.3 References

Unfortunately, the Computer Society document translator cannot handle automatic endnotes in Word; therefore, type the reference list at the end of the paper using the “References” style. See the IEEE Computer Society’s style for reference formatting at: <http://www.computer.org/portal/web/publications/style_refs>. The order in which the references are submitted in the manuscript is the order they will appear in the final paper, i.e., references submitted nonalphabetized will remain that way.

Please note that the references at the end of this document are in the preferred referencing style. Within the text, use “et al.” when referencing a source with more than three authors. In the reference section, give all authors’ names; do not use “et al.” Do not place a space between an authors' initials. Papers that have not been published should be cited as “unpublished” [4]. Papers that have been submitted or accepted for publication should be cited as “submitted for publication” [5]. Please give affiliations and addresses for personal communications [6].

Capitalize all the words in a paper title. For papers published in journals not published in English, please give the English citation first, followed by the original foreign-language citation [7].

## 7.4 Additional Formatting and Style Resources

Additional information on formatting and style issues can be obtained in the IEEE Computer Society Style Guide, which is posted online at: <http://www.computer.org/portal/web/publications/styleguide>. Click on the appropriate topic under the Special Sections link.

# Conclusion

Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions. Authors are strongly encouraged not to reference multiple figures or tables in the conclusion—these should be referenced in the body of the paper.

1. It is recommended that footnotes be avoided (except for the unnumbered footnote with the receipt date on the first page). Instead, try to integrate the footnote information into the text. [↑](#footnote-ref-1)