

# Hardware Software Co-Design for Protein Identification

**THALLADA SANDEEP**

A Dissertation Submitted to  
Indian Institute of Technology Hyderabad  
In Partial Fulfillment of the Requirements for  
The Degree of Master of Technology



भारतीय प्रौद्योगिकी संस्थान हैदराबाद  
Indian Institute of Technology Hyderabad

Department of Electrical Engineering

June, 2016

## Declaration

I declare that this written submission represents my ideas in my own words, and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited, or from whom proper permission has not been taken when needed.



---

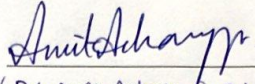
(Signature)


**THALLADA SANDEEP**

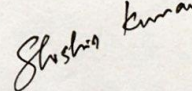
**EE13M1028**

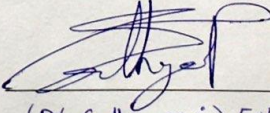
## Approval Sheet

This thesis entitled **Hardware Software Co-design for Protein Identification** by THALLADA SANDEEP is approved for the degree of Master of Technology from IIT Hyderabad.

  
(Dr. Amit Acharya) Advisor  
Dept. Electrical Engineering

  
(Dr. Ashudeb Dutta) Member  
Dept. Electrical Engineering

  
(Dr. Shishir Kumar) Member  
Dept. Electrical Engineering

  
(Dr. Sathya peyi) External  
Dept. Computer Science & Engineering

## **Acknowledgements**

I would like to acknowledge my guide Dr.Amit Acharyya for his guidance and continuous support in my M.Tech thesis. I am grateful for his confidence on me that I could do a good job with my thesis. It has been a great pleasure, in fact, an honor to work with him. Also I would like to acknowledge my Research team members Gudur Venkateshwarlu, Abhinay and Venkata Krishna for their valuable suggestions and Discussions. Also I would like to acknowledge my friends and family for motivating and encouraging me for completion of my M.Tech thesis. Thank you all very much.

Dedicated to

My Parents, Teachers and Friends

## Abstract

Recently new technologies and research in computational bioinformatics have revolutionized the rate of biological data generation. A vast amount of proteomics and genomics data is contributed to the life science society by researchers especially in the domain of high throughput next generation sequencing methods and it is doubling at every 18 months. Protein identification is a fundamental step in protein sequence analysis and it needs efficient solutions to match the data growth. Rapid methods are focused in the quest for faster protein sequence analysis to scan databases and identify a protein accurately. This benefits the discipline of disease biomarker identification and aid disease diagnosis and prognosis.

The problem of identifying a protein is similar to the string matching problem, i.e. the problem of finding a substring in another string; in particular in this case the problem consists in matching a string identifying a peptide of an unknown protein, against a string identifying a whole well known protein. String matching algorithms like Boyer-Moore and Knuth-Morris-Pratt (KMP) search single pattern strings in a larger string. These approaches have a requirement of high computational complexity. Aho Corasick algorithm (ACA) is a widely used multi-pattern string matching algorithm that has a linear computational complexity. Hardware accelerated solutions for protein identification are used to address the bottlenecks in the computational biology pipeline. Hardware Software codesign approach is used for reconfigurable string matching and simultaneously harness the advantages of both hardware and software. Reconfigurable string matching is performed in the disciplines of protein identification and biomarkers discovery. With the generation of plethora of sequenced data and number of biomarkers for several diseases, it is becoming necessary to have an accelerated processing and on-the-fly reconfigurable system design methodology to bring flexibility to its usage in the medical science community without the need of changing the entire hardware every time with the advent of new bio-marker or protein.

In this Thesis on-the-fly reconfigurable hardware-software co-design based reconfigurable solution for protein identification in real-time is presented. We use

on-chip memory based implementation to realize FSM design using ACA algorithm. We demonstrate the way proposed design can be used in real life with plethora of test cases.

The proposed methodology where implementing an accelerated and reconfigurable multi-pattern string matching platform does not require any step of fixed hardware system design when used in an application making it reconfigurable enabling the sequencing with any number of biomarkers for as many diseases as possible.

The proteome database of human at UniProtKB (Proteome ID up000005640) comprising of 20192 reviewed proteins and 42132 canonical, and isoform proteins with variable database-size are used for testing the proposed design and the performance of the proposed system has been found to compare favorably with the state-of-the art approaches with the additional advantage of real-time re-configurability.

# Contents

Declaration.....	ii
Approval Sheet .....	ii
Acknowledgements.....	iv
Abstract.....	vi
<b>Nomenclature .....</b>	<b>viii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Proteomics and Protein sequencing</b>	<b>4</b>
<b>3 ACA Algorithm for Protein Identification</b>	<b>7</b>
3.1 Theoretical Background .....	7
3.2 Pseudo Code for ACA FSM Implementation.....	10
3.3 Proposed Methodology .....	11
3.4 Results and Discussions .....	14
3.5 Conclusion.....	16
<b>4 Bit Split ACA Algorithm for Protein Identification.....</b>	<b>17</b>
4.1 Theoretical Background.....	17
4.2 Pseudo Code for Bit Split ACA FSM Implementation.....	23
4.3 Future work.....	27
<b>5 Conclusion .....</b>	<b>28</b>
<b>References.....</b>	<b>29</b>



# Chapter 1

## Introduction

The ability to quickly identify proteins is a major concern in many medical applications such as cancer monitoring and recognition and pharmaceutical research. Due to the fast increment of data available thanks to technological improvements, we need to take proteins identification a step further, improving its identification speed to match the growth of both proteomics and genomics databases. Because of the fast data growth inside both proteomic and genomic databases, protein identification requires an increasing amount of computation. Many researches will benefit from accurate protein identification and the ability to produce more accurate results at faster rates. For instance, disciplines, such as detecting biomarkers within diagnostic field, in order to recognize and monitor cancer disease with serum proteomic [1], bacterial identification, and pharmaceutical research will take advantage of this possibility.

The problem of identifying a protein is similar to the string matching problem, i.e. the problem of finding a substring in another string; in particular in this case the problem consists in matching a string identifying a peptide of an unknown protein, against a string identifying a whole well known protein. After all the peptides of the unknown protein have been searched a score can be assigned to the proteins in the database to find the one that best matches the unknown protein we are trying to identify. Unfortunately the computational complexity of this problem grows as  $L_{pep} \times L_{pro}$  if we identify the length of the peptide as  $L_{pep}$  and the length of the protein as  $L_{pro}$ . Furthermore as we have to search for all the unknown peptides forming the protein,  $U$  in the whole database composed of  $P$  proteins we have a worst case computational complexity of

$$U \times L_{pep} \times P \times L_{pro}$$

This complexity refers to a single protein identification, and obviously another multiplication factor has to be added to the formula to take in consideration the scenario of multiple proteins identification.

Efficient substring search algorithms such as Boyer-Moore[2] and Knuth-Morris-Pratt [3] that locate single pattern strings within a larger text string can be used in a multipass manner (i.e., one pass for each string in the set of peptides). However, this approach does not scale well with an increasing number of pattern strings. In particular, assuming  $p$  patterns with an average length of  $n$  and a text string of length  $m$ , naïve, multi-pass, approaches have computational complexity of  $O(p(m + n))$ .

The Aho-Corasick algorithm [4] provides a scalable solution to the string set matching problem in that it incorporates the search mechanism for the entire set of patterns into a single finite state machine (FSM). The power of Aho-Corasick stems from the ability of the algorithm to find the location of the strings in the pattern set in the text string in a single pass. The computational complexity of Aho-Corasick search is  $O(m + k)$  where  $k$  is the total number of occurrences of the pattern strings in the text. This linear processing time complexity has resulted in the widespread use of Aho-Corasick in string matching application.

The performance of the Aho-Corasick algorithm can be further enhanced by implementing it in hardware. Tan and Sherwood [5] were the first to describe an area-efficient hardware approach for implementing the Aho-Corasick for network intrusion detection systems implemented in application specific integrated circuits (ASICs). However, the complexity and costs associated with ASIC development is a significant impediment in their adoption in computational biology. Field programmable gate array (FPGA) devices, on the other hand, can be repeatedly reconfigured to create a variety of application-specific processing elements. This reconfigurable nature makes FPGAs a popular low-cost alternative to the development of specialized ASICs for a variety of application domains, including computational biology.

In [6] realized a HW implementation of Aho Corasick that aims at optimizing the utilized area on the FPGA device, while maintaining good performance, by partitioning the Finite State Machine (FSM) that performs the string matching analysing a small number of peptides in parallel. Although it has been demonstrated how Aho Corasick is the fastest string matching algorithm, it suffers from a great limitation for what concern HW implementation. It relies on the

creation of an ad-hoc HW component to match a single peptide, which causes the need to synthesize different HW components for each possible peptide.

Therefore there is need for Reconfigurable string matching is required at bioinformatics disciplines where patterns to be searched are changed, for example, a newly discovered biomarker is added into the database of known biomarkers. Multiple biomarkers can be searched in a given sample and many diseases can be found simultaneously. Pure hardware solution is not feasible in this scenario where patterns to be searched are updated continuously. All the steps of a hardware system design are necessarily run in pure hardware solutions. In a hardware system design steps like writing programs in hardware description language (HDL), synthesis, translation, mapping, place and route, programming file generation and configuring FPGA using bit stream file are run. In the scenario of only hardware solution, these steps are repetitively carried that add substantial amount design time. These require dedicated computer systems with sophisticated proprietary tools. Repetitive running of these steps can be avoided by employing reconfigurable systems with intelligent hardware software partitioning and codesign. We use hardware software codesign approach in our design.

# Chapter 2

## Proteomics and Protein Sequencing

Proteins are complex molecules. They are biochemical molecules consisting of one or more polypeptides, where polypeptide is a macromolecular chain of linked amino acids. Proteins and their interactions regulate the majority of processes in the human body. From mechanical support in skin and bones to enzymatic functions, the operation of the human body can be characterized as a complex set of protein interactions. Despite the efforts of scientists, many proteins and their functions have yet to be discovered. The wealth of information that lies in these unknown proteins may well be the key to uncovering the mysteries that govern life [7].

Proteomics investigates the proteins that make up an organism. Protein identification is a fundamental problem in Proteomics [7]. The ability to identify proteins and to determine their covalent structures has been central to the life sciences. The amino acid sequence of proteins provides a link between proteins and their coding genes via the genetic code, and, in principle, a link between cell physiology and genetics. The identification of proteins provides a window into complex cellular regulatory networks. For the identification of proteins, their sequencing, quantification and other tasks, mass spectrometry is currently the tool of choice.

Over the past 20 to 30 years, the analysis of tandem mass spectrometry data generated from polypeptide fragments has become the dominant method for the identification and classification of unknown protein samples. Tandem mass spectrometry (MS/MS) now plays a very important role in protein identification due to its speed and high sensitivity. It is emerging as the standard method for this important protein identification problem. With wide-ranging application in numerous scientific disciplines such as pharmaceutical research, cancer diagnostics, and bacterial identification, the need for accurate protein identification remains important and the ability to produce more accurate identifications at faster rates would be of great benefit to society as a whole. Protein mixtures are first digested into suitable sized peptides for mass spectrometric analysis using site-specific

proteases (usually Trypsin). Then the peptides are ionized via electro-spray ionization (ESI). Some of the peptides are fragmented by collision-induced dissociation (CID) and their tandem mass (MS/MS) spectra are collected. These peptides can then be analysed computationally to reveal their complete sequence. One way to interpret these MS/MS spectra is to compare the spectra with a protein sequence database to find the peptide whose predicted mass spectrum matches the experimental MS/MS spectra best. The original proteins present in the sample are then inferred based on the list of peptides matched to MS/MS spectra. These are referred as database search algorithms [8].

Protein identification is a fundamental step in protein sequence analysis and it needs efficient solutions to match the data growth. Rapid methods are focused in the quest for faster protein sequence analysis to scan databases and identify a protein accurately [9]. This benefits the discipline of disease biomarker identification and aid disease diagnosis and prognosis [10]. Protein identification using peptide fragments obtained by mass spectrometry involves database searching that is similar to string matching [11]. In string matching a database or text is searched to find locations of one or more strings also called patterns.

String matching algorithms like Boyer-Moore [2] and Knuth-Morris-Pratt (KMP) [3] search single pattern strings in a larger string. These approaches have a requirement of high computational complexity [6]. Aho Corasick algorithm (ACA) is a widely used multi-pattern string matching algorithm that has a linear computational complexity [4][12].

The Aho-Corasick algorithm (ACA) is widely used in computational biology for a variety of pattern matching tasks. For example, Brundo and Morgenstern use a simplified version of ACA to identify anchor points in their CHAOS algorithm for fast alignment of large genomic sequences [13,14]. The TROLL algorithm of Castelo, Martins, and Gao uses ACA to locate occurrences of tandem repeats in genomic sequence [15]. Farre et al. use Aho-Corasick as the search algorithm for predicting transcription binding sites in their tool PROMO v. 3. [16] Hyyro et al. demonstrate that Aho-Corasick outperforms other algorithms for locating unique oligonucleotides in the yeast genome [12]. The SITEBLAST

algorithm [17] employs the Aho-Corasick algorithm to retrieve all motif anchors for a local alignment procedure for genomic sequences that makes use of prior knowledge. Sun and Buhler use Aho Corasick deterministic finite automata (DFA) to design simultaneous seeds for DNA similarity search [18]. The AhoPro software package adapts the Aho-Corasick algorithm to compute the probability of simultaneous motif occurrences [19].

Aho-Corasick is arguably the best and the widest used multiple pattern matching algorithm that searches all occurrences of any of a finite number of keywords in a text string. Dandass et al. have used this algorithm for hardware acceleration of peptide pattern matching for the first chromosome of human genome [6]. We have used this algorithm and proposed a Methodology to make the system reconfigurable by partitioning into Hardware and software with usage of Xilinx Zync FPGA for peptide matching. This algorithm consists of two phases; constructing a finite state machine from keywords and then using these state machines for locating the keywords by processing the text string in a single pass.

In the Next Chapter construction of Finite State Machine for set of peptides using ACA and Hardware Software Co Design Implementation of ACA [26] for locating the peptides in the protein Database is explained.

# Chapter 3

## ACA Algorithm for Protein Identification

### 3.1 Theoretical Background

The Aho-Corasick algorithm consists of an initial pre-processing phase that creates the FSM from the set of pattern strings. The FSM resulting from the pre-processing phase is subsequently used for performing the string set matching.

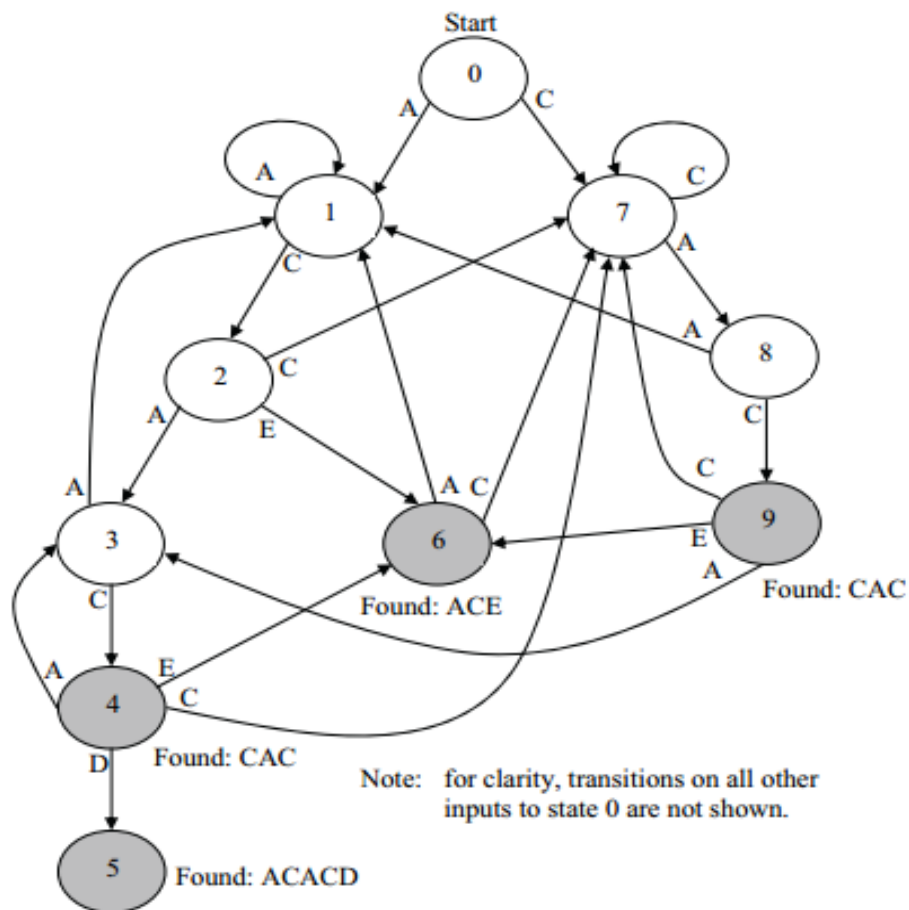
The pre-processing phase has a runtime complexity of  $O(pn)$  and the search phase has a runtime complexity of  $O(m + k)$ . Detailed description and analysis of Aho-Corasick can be found in [4]. A brief description follows below

In the pre-processing phase, the FSM is constructed using two steps. In the first step, a set of target strings is organized into a "keyword" tree. The root of the tree represents the state when no part of any pattern string has been found in the input message. The remaining nodes of the tree represent states where the pattern strings have been partially or fully matched. The edges in the tree represent the transitions resulting from the occurrence of specific symbols in the text string. The path from the root node to any node on the tree represents the subset of pattern strings that are potential matches.

In the second pre-processing step, "failure links" are added to the tree. Failure links lead from nodes on one branch of the tree to nodes on other branches. Failure links are needed because patterns strings can overlap in the text string and when the current branch of the tree fails to produce a match because of the current symbol in the text string, the FSM needs to resume processing from a new branch, without having to rescan input symbols.

In a computer, the FSM state transitions can be represented in the form of a table. Figure 1 illustrates the organization of the FSM for an implementation of

Aho-Corasick that matches the peptide set {ACACD, ACE, and CAC}; Table 1 presents a table-oriented representation used for implementation of the same FSM. In Figure 1, the state 0 is the start state and the shaded states 4, 5, 6, and 9 match peptides CAC, ACACD, ACE, and CAC, respectively.



**Figure1**  
**An FSM for matching peptide set {ACACD, ACE, and CAC}.**

When the FSM is in any state and receives an input symbol not shown in the figure, the FSM transitions to state 0. At runtime, the FSM interpreter reads the row corresponding to the current state from the table, reads the next input symbol from the reading frame, and determines the next state from the row entry corresponding to the input symbol.

When the FSM transitions to a state, it looks at the pattern match column of the table's corresponding row in order to determine if a match has occurred.



A non-null entry in the pattern match entry of row specifies the pattern that has been located by the FSM.

**Table-1. A table oriented representation of the FSM for the peptide set {ACACD, ACE, and CAC}**

Input Text Symbol											
Current State	A	B	C	D	E	F .....	Z	Match Vector			
0	1	0	7	0	0	0	0	0	0	0	0
1	1	0	2	0	0	0	0	0	0	0	0
2	3	0	7	0	6	0	0	0	0	0	0
3	1	0	4	0	0	0	0	0	0	0	0
4	3	0	7	5	6	0	0	0	0	0	2
5	1	0	7	0	0	0	0	0	0	0	1
6	1	0	7	0	0	0	0	0	0	0	4
7	8	0	7	0	0	0	0	0	0	0	0
8	1	0	9	0	0	0	0	0	0	0	0
9	3	0	7	0	6	0	0	0	0	0	2

### 3. 2 Pseudo code for ACA FSM Implementation

#define M=1000. \*\*\* M is Sum of the length of all the peptides

#define N=25 \*\*\* N is the number of Amino Acid Symbols

#define N1=1 \*\*\* N1 is Peptide Match Vector column used for Representing at which state Peptide Matches

Consider a set of peptides

Create ACA FSM for the above Peptides

Extract FSM into Matrix A [M][N]

Copy matrix A [M][N] in to the matrix BRAM\_table [M][N]

For (k=1 to M)

Initialize l1 = 0; p1 = 0;

For (j=0 to N)

If (A [0][j]!= 0)

    If (BRAM\_table[k][j] == 0)

        BRAM\_table[k][j] = A [0][j];

    Else

        l1 = A [0][j];

        p1 = A[k][j];

    If (p1 != 0)

        Recursive (l1, p1);

Void Recursive (int l1, int p1)

Initialize j, l = 0, p = 0;

For (j = 0 to N)

    If (A [l1][j]!= 0)

        If (BRAM\_table[p1][j] == 0)

            BRAM\_table[p1][j] = A[l1][j];

        Else

            l = A[l1][j];

            p = A[p1][j];

        If (p != 0)

        Recursive (l, p);

The Above Algorithm Implementation creates FSM Table for the peptides ACACD, ACE, CAC as shown in table-1 above

The Above Algorithm for FSM Table is implemented in C language which is the software part of our Hardware Software Co-Design. The above table is stored in BRAM of Xilinx ZYNQ FPGA and it is interfaced with generic custom IP such that it forms the FSM logic in Hardware.

The FSM logic Hardware is implemented in PL (Programmable Logic) Section of Xilinx Zed Board and controlled by ZYNQ Processor which is in the PS (Processing System) section of ZYNQ-7000 SOC.

### 3.3 Proposed Methodology

The architectural diagram of the proposed system is depicted in Fig. 3. The architecture has a Zynq-7000 All Programmable SoC (AP SoC) ZYNQ7 processing system as the master. The FPGA device on board is XC7Z020 and it has an ARM Cortex-A9 MP Core CPU. A processor system reset module is used to reset and synchronize all modules in the design by resetting them as per the reset conditions given by master at its input. Advanced extensible Interface (AXI) interconnect is used to interface the memory-mapped ZYNQ7 master with the memory-mapped slaves.

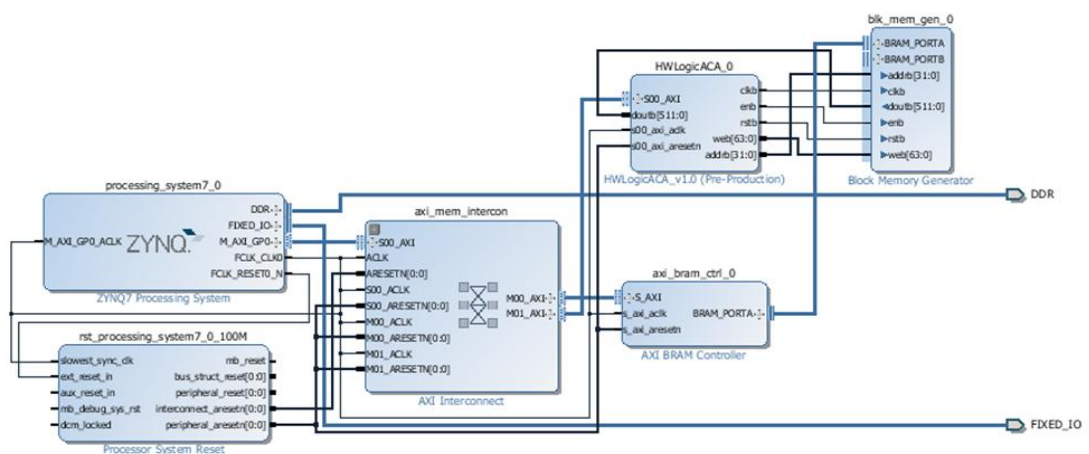


Figure 2. Architecture of the proposed system

Block memory generator module is used for generating block memory. Block memory is used for storing data during program run time. AXI block RAM (BRAM) controller acts as a bridge to Block memory generator module is used for generating block memory. Block memory is used for storing data during program run time. AXI block RAM (BRAM) controller acts as a bridge to communicate between ZYNQ7 via AXI interconnect and BRAM created by block memory generator. Custom hardware module is the necessary hardware logic required to realize FSM using memory communicate between ZYNQ7 via AXI interconnect and BRAM created by block memory generator. Custom hardware module is the necessary hardware logic required to realize FSM using memory. A detailed diagram of the interface between block memory and hardware logic is depicted in Fig. 3 (a). The custom hardware is a generic hardware and along with memory implements a memory based FSM. Block memory is stored with contents that realize Aho-Corasick algorithm with the hardware logic. Depending on an input character value the corresponding input lines at the multiplexer (MUX) are activated and are available at the output of multiplexer. A state register is used to latch the multiplexer output and it feeds block memory generator with the necessary address.

We use Xilinx Vivado Design Suite software tools to design our system. Vivado Design Suite has all the features of a hardware system design along with system on a chip development. We built the design in Vivado as depicted in Fig. 1. We use Creating and Packaging Custom IP utility available in Vivado to design the hardware logic as depicted in Fig. 3 (a) and interface it to the ZYNQ7 processing system. A hardware description file (hdf) is generated in Vivado and exported along with bit stream file to Xilinx Software Development Kit (SDK) tool. Complete hardware of the system is ready before begin SDK. We write a C language application program in SDK to run on the ZYNQ7 processing system. A flowchart describing the application program execution is depicted in Fig.3 (b). A brief idea of all the steps performed in the system and its working is described as follows.

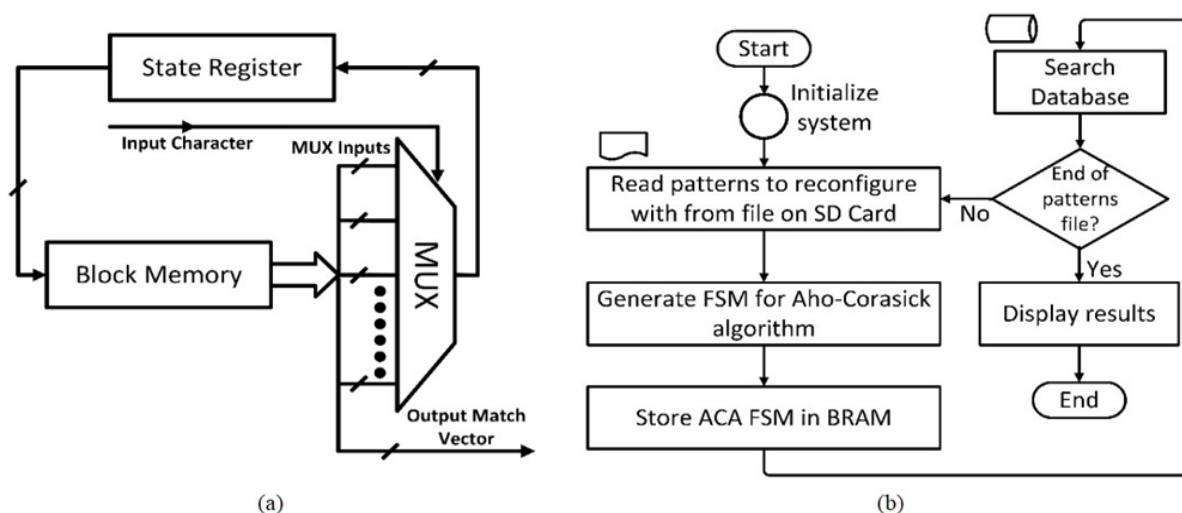


Figure 3. Proposed (a) Memory based FSM design (b) Flow diagram describing steps followed in the application program

A file containing patterns which are to be searched is stored on a portable Secure Digital High Capacity (SDHC) card with FAT32 file system. The database in which search is performed is also residing on the SDHC card. The FPGA device on board is configured with the bit stream file along with the application program. Patterns are read from patterns file and ACA algorithm is run for searching these patterns. An ACA-FSM is designed on the run as per the given patterns. We use the memory based architecture for implementing FSM proposed in [20][21] and generate block RAM contents for the corresponding FSM and store them in block RAM. The database file in which patterns are searched is read from the SDHC card and characters are received by the custom hardware. These characters act as addresses to block memory and data stored at these addresses is made available out of the block memory. Multiplexer inputs are connected to this data and part of this data is selected depending on the character fed to custom hardware. Output match vector is also a part of the data and it gives the information about the pattern found after receiving characters. The system work till the end of the database file. UniProt (Universal Protein Resource) identifier for the protein in which the patterns are found along with their corresponding location in that protein is displayed on a console window.

## 3.4 Results and Discussions

We implemented the proposed system using Avnet Zed board development board. From the available on-chip resources for the XC7Z020 FPGA device 9.18% FF, 12.42% LUT, 1.23% Memory LUT and 35.71% BRAM resources are used for the System. The above figure for BRAM is maximum allotted and actual BRAM Utilization depends on the number of patterns and their size. Total on-chip power Comprising of static and dynamic is 1.891 watt for the complete board.

For testing the system with real world data we use the UniProt Knowledgebase (UniProtKB) [24]. The proteome database of human at UniProtKB (Proteome ID up000005640) is made up of 20192 reviewed proteins. This database, made of 42132 canonical & isoform proteins, is chosen in FASTA format for testing our system. A set of well referred and standard disease biomarker proteins is selected [22]. Peptide Mass, an online tool for enzymatic cleavage of proteins, is used to digest these selected disease biomarker proteins. The peptides obtained after proteins digestion act as patterns and stored in different patterns file with identifier as their file names. The proposed system is run with these peptides and the time taken for searching the complete database is noted.

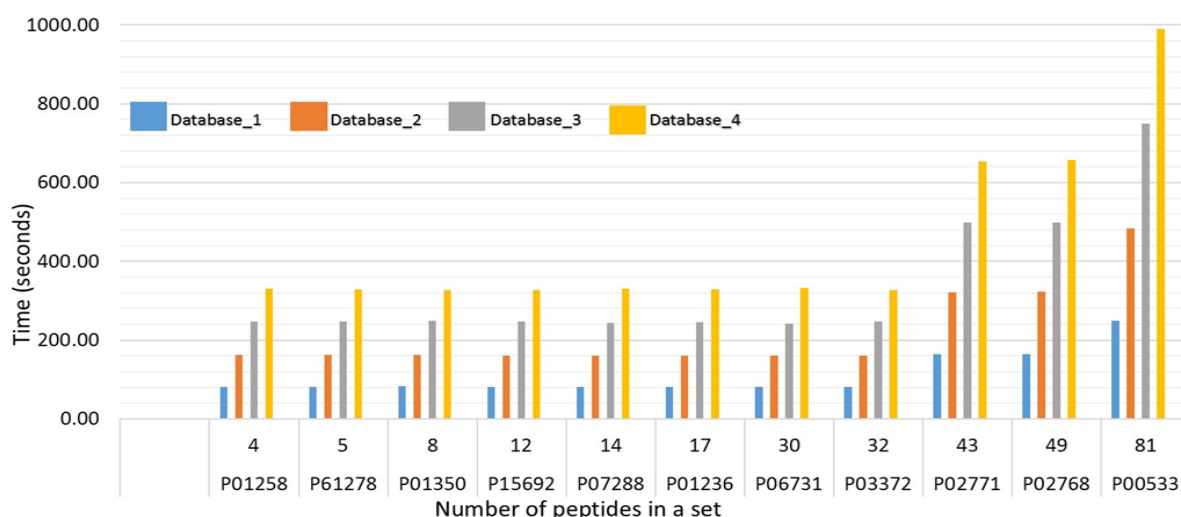
We also studied the effect of database size in searching these proteins. The whole human proteome database is divided into four databases of 8404, 16830, 25256 and 33682 proteins. We perform search for the previously mentioned ten pattern sets. Table- 2 shows both the results. We see that time taken for searching database is independent of the number of patterns. We take a maximum of 32 patterns obtained after protein digestion and perform search. In case of patterns more than 32, the next 32 proteins are selected for searching. As a result of this the time taken for searching patterns greater than 32 and less than 64 in number is nearly doubled and for greater than 64 in number it is tripled. Fig. 3 shows the comparison bar chart. It is evident that time taken for search is linear and is independent of the number of patterns which is a characteristic of Aho-Corasick algorithm [4]. Table-3 compares the features available in the proposed design with few other designs available in literature. The proposed system does not have any limitation on the number of patterns to be searched but at the cost of time. It is a multi-pattern searching system

designed intelligently with hardware-software codesign. Table 2 and Fig. 4 shows that the proposed system is reconfigurable with the desired patterns to be searched. This can be done during run time by selecting the respective pattern file.

**Table-2 Impact of database length on time for search**

UniProt Identifier	Number of peptide	Time taken for searching (seconds)				
		<i>DB (1)</i>	<i>DB (2)</i>	<i>DB (3)</i>	<i>DB (4)</i>	<i>Full DB</i>
P01258	4	81.97	162.62	247.45	330.66	415.30
P61278	5	82.06	162.45	248.24	328.57	415.64
P01350	8	82.48	163.17	249.57	327.66	415.36
P15692	12	81.62	160.91	247.99	327.47	416.25
P07288	14	81.88	161.57	244.55	329.86	416.51
P01236	17	82.27	160.25	245.66	329.47	415.45
P06731	30	81.99	160.92	242.37	332.99	416.02
P03372	32	81.79	161.46	247.57	326.54	416.22
P02771	43	163.81	321.12	497.66	654.24	838.46
P02768	49	163.99	323.67	499.38	656.51	838.81
P00533	81	249.31	484.47	748.90	990.13	1264.61

\*DB: Database



**Figure 4. Effect of varying the length of database**

**Table-3****Features available in proposed system**

<b>System Design</b>	<b>SMV [23]</b>	<b>GB [24]</b>	<b>Proposed design</b>
<b>Scheme of string matching algorithm</b>	<b>ACA</b>	<b>KMP</b>	<b>ACA</b>
<b>Simultaneous single pattern matching</b>	<b>✓</b>	<b>✓</b>	<b>✓</b>
<b>Simultaneous multi-pattern matching</b>	<b>✓</b>	<b>✗</b>	<b>✓</b>
<b>Real world data verification</b>	<b>✗</b>	<b>✓</b>	<b>✓</b>
<b>Hardware modules in system</b>	<b>FSM Logic</b>	<b>KMP and DMA core</b>	<b>Generic custom hardware</b>
<b>Need to run hardware system design flow repeatedly</b>	<b>✗</b>	<b>✓</b>	<b>✓</b>

*KMP: Knuth-Morris-Pratt*

*DMA: direct memory access*

## 3.5 Conclusion

We presented a real-time reconfigurable string matching solution using hardware-software codesign that does not require to carry the steps of hardware system design repeatedly. The proposed method has also been validated against the variable human proteomic data-bases. The proposed system can search multiple strings in a given text in quick-time and can be employed for applications real world proteomic/genomic variable data-size for emerging bioinformatics applications.



# Chapter 4

## Bit Split ACA Algorithm for Protein Identification

### 4.1 Theoretical Background

The branching factor of the Aho-Corasick FSM tree depends on the number of symbols possible in the input text string. For example, the branching factor is 256 when eight bits are used for representing the alphabet of valid symbols in the strings (as is the case with intrusion detection applications such as Snort). However, because only five bits are needed for representing all the 20 amino acids and additional special symbols (*e.g.*, those representing ambiguous amino acids), the size of the table can be reduced significantly by reducing the total number of columns.

Additional savings in storage can be obtained by splitting the FSM into smaller FSMs. Following are two approaches to make an FSM smaller:

1. Reduce the number of peptides in the peptide set. This will reduce the number of states in the FSM, and therefore, will reduce the number of bits required to store the "next state" transition value.
2. Split the FSM into simpler FSMs that are responsible for encoding and operating on individual bit positions of the symbols in the peptides patterns and the text string. For example, the FSM in Table 1 can be split into five separate bit-split FSMs, FSM0, FSM1, FSM2, FSM3, FSM4, one for each of the five bit positions it takes to encode all the peptides. Because the bit-split FSMs operate independently from each other, all of the separate bit-split FSMs must agree on a match before a peptide match is confirmed.

The general bit-split FSM algorithm is described in detail in [25]. The bit split FSM process is described below for the FSM in Table 1, resulting in the five bit-split FSMs designated as FSM0, FSM1, FSM2, FSM3, and FSM4 shown in Tables 5, 6, 7, 8 and 9 respectively.

Consider the construction of FSM 0, the bit-split FSM corresponding to bit position 0; table 4 describes the bitwise encoding of selected amino acids. State  $n$  in the original FSM is designated as FSM:  $n$  and state  $m$  in  $FSM_0$  is designated as  $FSM_0:m$ .

Initially, the root node  $FSM_0:0$  is added to  $FSM_0$ . Next, all states in the original FSM that can be reached from  $FSM:0$  (The root node from the original FSM) when the bit position in the transition is 0 are determined and aggregated into a new bit-split node  $FSM_0:1$ . In the example,  $FSM:1$  and  $FSM:7$  are aggregated to form  $FSM_0:1$ . Because  $FSM_0:1$  does not already exist in  $FSM_0$  (i.e., there is no state in  $FSM_0$  that is aggregated from  $FSM:1$  and  $FSM:7$ ), it is added to  $FSM_0$  with a transition from  $FSM_0:0$  when the input bit is 0. Next, all states in the original FSM that can be reached from  $FSM:0$  when the bit position in the transition is 1 are determined and aggregated; in this example, there are no such states. Therefore, the transition from  $FSM_0:0$  goes back to  $FSM_0:0$  when the input bit is 1. This process is repeated for all newly added states in  $FSM_0$ .

**Table-4 Bit encoding of selected peptides**

<b>Bit Encoding of Selected Peptides</b>					
<b>Peptide</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
<b>A</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>C</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>
<b>D</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>
<b>E</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>
<b>....</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>M</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>1</b>
<b>...</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>Y</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>

FSM0:1 was added previously and is examined next. Note that FSM 0:1 is an aggregate of FSM:1 and FSM:7. Therefore, all states in the original FSM that can be reached from either FSM:1 or FSM:7 when the input at bit position is 0 are aggregated into FSM0:2. In this example, FSM0:2 is created from FSM:2 and FSM:8. Because FSM0:2 does not already exist in FSM0, it is added to FSM0 with a transient of 0 from FSM0:1. Again, there is no transition from FSM0:1 when the input bit is 1, therefore, state FSM0:1 transitions back to FSM0:0 when the input bit is 1. This process is continued until there are no new states added to FSM0. Note that only unique new nodes are added to FSM0. When a new node FSM0:n is created by aggregation but another node, FSM0:k, created by aggregating the same set of nodes already exists in FSM0, then instead of inserting the new node, FSM0:n, a transition to FSM0:k is inserted into FSM0. Peptide matches are also handled using aggregation (i.e., state FSM0:k matches all the peptides that are matched by the states in the original FSM that were aggregated into FSM0:k). This process is repeated for all bit positions resulting in the five separate bit-split FSMs depicted in tabular form in Tables 5, 6, 7, 8 and 9.

**Table-5 The Bit Split FSM corresponding to Bit position 0 (FSM0)**

State	0	1	Match Vector
0	1	0	0 (000)
1	2	0	0 (000)
2	3	0	0 (000)
3	4	0	6 (110)
4	4	5	6 (110)
5	1	0	1 (001)

**Table-6 The Bit Split FSM corresponding to Bit position 1 (FSM 1)**

State	0	1	Match Vector
0	1	2	0 (000)
1	1	3	0 (000)
2	7	2	0 (000)
3	4	2	0 (000)
4	1	5	2 (010)
5	4	6	4 (100)
6	7	2	1 (001)
7	1	8	0 (000)
8	4	2	4 (100)

**Table-7 The Bit Split FSM corresponding to Bit position 2 (FSM 2)**

State	0	1	Match vector
0	1	0	0 (000)
1	2	0	0 (000)
2	3	4	0 (000)
3	5	4	4 (100)
4	1	0	2 (010)
5	6	4	4 (100)
6	6	4	5 (110)

**Table-8 The Bit Split FSM corresponding to Bit position 3 (FSM 3)**

State	0	1	Match Vector
0	1	0	0 (000)
1	2	0	0 (000)
2	3	0	0 (000)
3	4	0	6 (110)
4	5	0	6 (110)
5	5	0	7 (111)

**Table-9 The Bit Split FSM corresponding to Bit position 4 (FSM 4)**

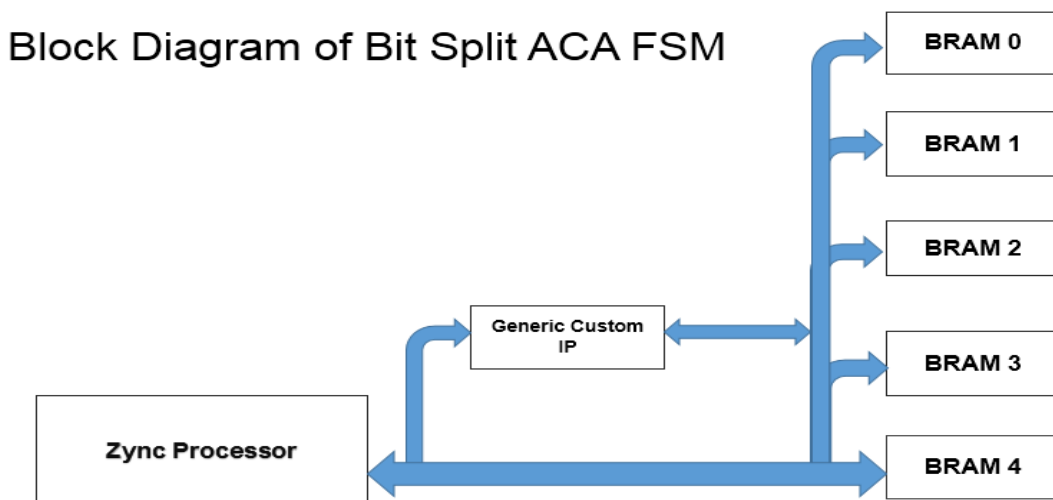
State	0	1	Match Vector
0	1	0	0 (000)
1	2	0	0 (000)
2	3	0	0 (000)
3	4	0	6 (110)
4	5	0	6 (110)
5	5	0	7 (111)

Because several states from the original FSM that match different peptides may be combined into a single state in a bit-split FSM, a mechanism to indicate multiple matches is required. In the bit-split FSM, a vector of bits is used to encode the peptide matching attribute of for each state. For example, state FSM0:3, matches peptides 2 and 4, and therefore, has a peptide matching bit vector containing 110. Using this mechanism, after the various state machines enter their respective new states, a bitwise logical and operation can be used to

determine the peptide match that all five FSMs agree on. For example, assume that at some point in time, the bit-split FSMs in Tables 5, 6, 7,8and 9 are in states FSM0:3, FSM1:5, FSM2:6, FSM3:5, andFSM4:5 with match bit vectors of 110, 100, 101, 111, and111, respectively. The bitwise logical AND of the five match bit vectors results in the bit vector 100 that indicates that peptide 3 is matched in this case. However, if FSM0 is in state FSM0:5, with a matching bit vector of 001, then the result of the logical and will result in 000,indicating that no peptides are currently matched.

The above tables 5,6,7,8 and 9 generated are stored in Bram 0, Bram 1,Bram 2 Bram 3, and Bram 4.The Generic Custom IP along with Bram's forms FSM logic. The FSM table is generated in C language by ZYNC processor and stored in Bram's. The Block diagram of the system is presented below which acceralates the pattern matching of peptides by exploiting bit level parallelism 5 bit split FSM and the memory consumption of the design is low compared to ACA FSM

**Figure-5 Bit Split ACA System Design Block Diagram**



## 4.2 Pseudo Code for Bit Split ACA Implementation

```
#define N=26
```

```
#define N1=1
```

```
#define N2=2
```

```
#define M=1000
```

```
#define N3=3
```

Consider ACA FSM  $A1[M][N+N1]$

Create Structure consist of array "a[N]" and variable "value" for representing state number.

```
create Array FSM0[M][N3]
```

```
create structure states S0[M]
```

```
initialize l1=0 ,input_0[1]=0,match_vector=0,vector[26]
```

```
(S0[0].a)[0] = 0;
```

```
S0[0].value = 1;
```

```
l1 = l1+1;
```

```
create0 (input_0, 1, 0, 0);
```

```
for(i=0 to l1)
```

```
{
```

```
match_vector=0;
```

```
for(j=0 to S0[i].value)
```

```
{
```

```
vector[j]=(S0[i].a)[j];
```

```
match_vector=match_vector|B0 [vector[j]][26];
```

```
}
```

```
FSM0[i][N2]=match_vector;
```

```
}
```

```

Void create0 (int * input_0, int n, int s, int b)
{
initialize i,j,a,kl=0,kr=0,x=0,y=0,Left[N],Right[N],Pl[M],Pr[M],sl,sr,w,w1,Ql=0,Qr=0;
for (i=0 to M)
{
    Pl[i]=0;
    Pr[i]=0;
}

for(j=0 to n)
{
    for(i=0 to 26)
    {
        w=i;
        w=w%2;
        if(Pl[A1[input_0[j]][i]] == 0 && w == 0)
        {
            Left[x] = A1[input_0[j]][i];
            x=x+1;
            Pl[A1[input_0[j]][i]] = 1;
        }
        if(Pr[A1[input_0[j]][i]] == 0 && w == 1)
        {
            Right[y] = A1[input_0[j]][i];
            y=y+1;
            Pr[A1[input_0[j]][i]] = 1;
        }
    }
}
}

```



```

for(i=0 to 11)
{
  if(x == S0[i].value)
  {
    for(j=0 to x)
    {
      if(Left[j]== (S0[i].a)[j])
      {
        kl = kl+1;
      }
    }

    if(kl==x)
    {
      sl=i;
      Ql=x;
    }
  }
  if(y == S0[i].value)
  {
    for(j=0 to y)
    {
      if(Right[j]== (S0[i].a)[j])
      {
        kr = kr+1;
      }
    }
    if(kr==y)
    {
      sr=i;
      Qr=y;
    }
  }
  kl = 0;
  kr = 0;
}

```

```

if(Q1!=x)
{
    for(i=0 to 26)
    {
        (S0[l1].a)[i] = Left[i];
    }
    S0[l1].value = x;
    FSM0[s][0]=l1;
    l1=l1+1;
}
else
{
    FSM0[s][0]=sl;
}
if(Qr != y)
{
    for(i=0 to 26)
    {
        (S0[l1].a)[i] = Right[i];
    }
    S0[l1].value = y;
    FSM0[s][1]=l1;
    l1=l1+1;
}
else
{
    FSM0[s][1]=sr;
}
    if(Q1 !=x)
    {
        create0(Left,x,FSM0[s][0],0);
    }
    if(Qr !=y )
    {
        create0(Right,y,FSM0[s][1],0);
    }
return;
}

```

## **4.3 Future work**

In the above Bit Split Algorithm Implementation we have given the idea about the system design which we envisioned. The above system can be implemented on Real time Xilinx Zync FPGA by introducing the concept of Hardware Software Co-Design with Re-Configurability and Parallel Architecture by exploiting the concept of bit level parallelism with less memory consumption with all these advantages the system can be used to accelerate the Bio-Informatics Research for protein Identification which has numerous Bio medical applications.

## **5. Conclusion**

In this Thesis I have proposed a Reconfigurable Hardware Software Co-Design Methodology for Protein Identification which has real time application for Bio Informatics Research and I also given the idea about the Hardware Software Co-Design of Bit Split Algorithm a System Design Approach which is a parallel Architecture which can further accelerate the Protein Identification with less memory consumption.

# References

- [1] W. Liu, Q. Yang, B. Liu, and Z. Zhu, "Serum proteomics for gastric cancer," *Clinica Chimica Acta*, vol. 431, pp. 179–184, 2014.
- [2] Boyer RS, Moore JS: A Fast String Searching Algorithm. *Communications of the ACM* 1977, 20:762-772.
- [3] Knuth DE, Morris JH, Pratt VB: Fast pattern matching in strings. *SIAM Journal of Computing* 1977, 6:323-350.
- [4] Aho A, Corasick M: Efficient string matching: an aid to bibliographic search. *Communications of the ACM* 1975, 18:333-340.
- [5] Tan L, Sherwood T: A High Throughput String Matching Architecture for Intrusion Detection and Prevention: Madison, Wisconsin US. ; 2005.
- [6] Y.S.Dandass,S.C. Burgess, M. Lawrence, and S. M. Bridges,"Accelerating string set matching in fpga hardware for bioinformatics research," *BMC bioinformatics*, vol. 9, no. 1, p. 197, 2008.
- [7] A. Alex, J. Rose, R. Isserlin-Weinberger and C. Hogue, "Hardware Accelerated Novel Protein Identification", *Field Programmable Logic and Application- Lecture Notes in Computer Science*, vol. 3203, 2004, pp. 13-22.
- [8] Jian Zhang, McQuillan I., Fang Xiang Wu, "Speed improvements of peptide spectrum matching using SIMD instructions", *Bioinformatics and Biomedicine Workshops (BIBMW)*, 83-88, Dec. 2010.
- [9] W.J. Henzela, C. Watanabea and J.T. Stults, "Protein identification: the origins of peptide mass fingerprinting," *Journal of the American Society for Mass Spectrometry*, Vol. 14, no. 9, pp. 931-942. 2003.
- [10] Sahab, Z. J., Semaan, S. M., & Qing-Xiang, A. S. (2007).Methodology and applications of disease biomarker identification inhuman serum. *Biomarker Insights*, 2, 21
- [11] T. A. Anish , M. Dumontier , J. S. Rose and C. W. V. Hogue,"Hardware-accelerated protein identification for mass spectrometry", *Rapid Communi. Mass Spectrom.*, vol. 19, pp. 833-837, 2005
- [12] H. Hyyrö, M. Juhola, M. Vihinen, "On exact string matching of unique oligonucleotides". *Comput Biol Med*, vol. 35, no. 2, pp. 173-81, 2005.
- [13] Brudno M, Morgenstern B: Fast and sensitive alignment of large genomic sequences. *Proc IEEE Comput Soc Bioinform Conf* 2002,1:138-147.

- [14] Brudno M, Steinkamp R, Morgenstern B: The CHAOS/DIALIGNWWW server for multiple alignment of genomic sequences. *Nucleic Acids Res* 2004, 32(Web Server issue):W41-4.
- [15] Castelo AT, Martins W, Gao GR: TROLL--tandem repeat occurrence locator. *Bioinformatics* 2002, 18(4):634-636.
- [16] Farre D, Garcia D, Alba MM, Messeguer X: Prediction of Transcription Factor Binding Sites with PROMO v. 3: Improving the Specificity of Weight Matrices and the Searching Process. In 5th Annual Spanish Bioinformatics Conference Barcelona Spain; 2004
- [17] Michael M, Dieterich C, Vingron M: SITEBLAST--rapid and sensitive local alignment of genomic sequences employing motif anchors. *Bioinformatics* 2005, 21(9):2093-2094.
- [18] Buhler J, Keich U, Sun Y: Designing seeds for similarity search in genomic DNA. *Journal of Computer and System Sciences* 2005, 70(3):342-363.
- [19] Boeva V, Clement J, Regnier M, Roytberg MA, Makeev VJ: Exact pvalue calculation for heterotypic clusters of regulatory motifs and its application in computational annotation of cisregulatory modules. *Algorithms Mol Biol* 2007, 2(1):13.
- [20] I. Garcia-Vargas et al., "Rom-based finite state machine implementation in low cost fpgas," in *Industrial Electronics, 2007ISIE 2007. IEEE International Symposium on*, June 2007, pp. 2342-2347
- [21] JR. Senhadji-Navarro, I. Garcia-Vargas and J. L. Guisado, "Performance evaluation of RAM-based implementation of finite state machines in FPGAs," *Proc. 19th IEEE Int. Conf. Electron. Circuits Syst. (ICECS)* , pp.225 -228.
- [22] "UniProt: a hub for protein information", *Nucleic Acids Res.*, vol. 43, pp. D204-D212, 2015.
- [23] Vidanagamachchi, S.M.; Dewasurendra, S.D.; Ragel, R.G., "Hardware software co-design of the Aho-Corasick algorithm: Scalable for protein identification?," in *Industrial and Information Systems (ICIIS), 2013 8th IEEE International Conference on* , vol., no., pp.321-325, 17-20 Dec. 2013.
- [24] Bianchi, G.; Casasopra, F.; Durelli, G.C.; Santambrogio, M.D., "A hardware approach to protein identification," in *Biomedical Circuits and Systems Conference (BioCAS), 2015 IEEE* , vol., no., pp.1-4, 22-24 Oct. 2015.
- [25] Jung HJ, Baker ZK, Prasanna VK: Performance of FPGA Implementation of Bit-split Architecture for Intrusion Detection Systems. 2006.

[26] Venkateshwarlu Y. Gudur, Sandeep Thallada, Abhinay R. Deevi, Venkata Krishna Gande, Amit Acharyya, Member, IEEE, Vasundhra Bhandari, Paresh Sharma, and Ganesh R. Naik, Senior Member,IEEE, Saqib Khursheed , Reconfigurable Hardware-Software Co-Design Methodology for Protein Identification ,IEEE EMBC 2016