Hardware Design of a High Speed VLSI Comparator Framework for DNA Sequence Matching Using High Level Synthesis

Dr. Seema Verma [1], Sanjeev Kumar [2] and Dibyayan Das Sharma [2]
[1] Dept. of Electronics, Banasthali University,
Rajasthan, India
[2] Dept. of Electronics & Communication Engineering, Delhi College of Technology & Management,
Palwal, Haryana, India

Corresponding Author Email: dibyayandsharma@gmail.com

Abstract

DNA sequence matching has assumed a very important role in molecular biology and bioinformatics. A typical DNA sequence represents a massively huge set of data and hence sequence matching must be made feasible and within practical limits by use of technology and innovation. Here in this paper, a high speed pipelined comparator framework has been designed which can be used for rapid sequence matching. The design is created following a new methodology using GAUT II high level synthesis (HLS) tool which allows developing hardware with optimized performance metrics. The designed hardware has the potential to be a low cost solution to real-time DNA matching and verification processes with significantly low latency.

Keywords: *DNA sequence matching, VLSI, High Level Synthesis, Computer Architecture, High Speed Comparator*

1. Introduction

DNA or Deoxyribonucleic Acid contains the genetic information required for development and functioning of all known living organisms. The specific structure of DNA molecule conveys specific information and hence the sequence of its base pairs is very important [1][2]. Recently there has been tremendous interest in knowing the DNA structure of various living organism and is an area of active research in biology.

Apart from the interest in this field from an academic research point of view, study of DNA is also important from a technological point of view. It is known that the DNA of each and every individual is unique such that one can identify the person by his/her DNA. This can be essentially used to track down criminals. Furthermore, correlation of DNA structure and genetic information with specific ailments can be used to precisely diagnose patients. Study of DNA also has widespread applications in fields like genetic engineering, forensics and bioinformatics.

The typical sequence of a DNA contains millions of base pairs and hence ideally speaking an exact matching process between two DNA samples of same size would require millions of comparisons. To solve this problem, the matching is done between a sample DNA sequence and

a much larger DNA sequence in database. Several algorithms have been designed commonly known as Exact String Matching Algorithms (ESMA) which aims to bring down the computational complexity involved in exact matching process and have been applied to DNA sequencing. They work by finding the number of occurrences of the sample DNA (called a string or a pattern) in the target DNA stored in the database (called text). Yet another class of algorithms is called Inexact or Approximate String Matching Algorithms that aims to find the closest possible match between two DNA sequences.

Whatever be the class of algorithm used, pattern matching algorithms have two main objectives viz. 1) reduction of number of character comparisons required in the worst and average case analysis and 2) reduction of time required in the worst and average case analysis [3]. Considering the complexity involved in DNA pattern matching comprising of millions of base pairs, most of these algorithms are implemented in software. Depending on the amount of investment that can be put in by interested parties, either these softwares are run on general purpose hardware like desktop computers or in highly specialized supercomputers. Both these alternatives are two extremes wherein a software run on desktop computers would be a low cost solution but will take time to compute a typically long sequence while computing the same sequence with a supercomputer would take the least time but will prove to be a costly affair.

In this work, a sort of middle path has been chosen wherein a high speed pipelined comparator framework has been designed which can be used with the aforementioned algorithm and thereby enhance the computational ability of these algorithms while reducing their running time. The integration of this comparator framework with the software based algorithm can prove to be a low cost yet sufficiently high speed solution for DNA sequencing needs and hence has the potential to be widely adopted by parties who may not have access to supercomputing infrastructure.

2. High Level Synthesis Methodology

High level synthesis is the latest technological evolution of logic synthesis for VLSI design. The basic goal of high level synthesis tools is to generate register transfer level (RTL) code in some hardware description language (HDL) like VHDL or Verilog, from an input high level language like C/C++ or MATLAB while satisfying a given set of design constraints and optimizing the given cost function.

Although research in high level synthesis started a long time back (in 1970s), success in this field has been seen only recently [4] in specific domain areas like DSP. However, it is quite interesting to note the various technological advancements that this methodology offers and hence becomes important for researchers to develop a tool that can be used across all the domains of VLSI chip designing. One of the major advantages of high level synthesis tool is that it allows the designer to think about their designs at a higher level of abstraction. This implies that the designer can now concentrate on the underlying algorithm to solve a problem rather than worrying about problems pertaining to gate level synthesis. The design time will reduce drastically and allow the designer to explore multiple algorithmic solutions.

Despite these advantages that make high level synthesis methodology seem like a dream come true for design engineers, there are several loop holes that needs to be sorted out. It is believed that syntheses performed by these tools are not at par with hand coded designs in terms of certain design constraints. At high levels of abstraction, estimates of key hardware constraints

like area, clock cycles, frequency and power are usually prone to error which may propagate to the final implementation [5]. Furthermore, high level synthesis tools do not provide satisfactory results when it comes to synthesis of control-dominated branching logic. These tools, as it turns out, are much more suitable for data intensive algorithm synthesis.

However, with the advent of present tools and cutting-edge research in this field, the situation seems promising for the future. Most tools today employ various techniques like parallelism extraction, loop unrolling, optimized allocation strategies, compiler pragmas etc. to enhance the quality of the synthesis results. At present these tools have largely succeeded in being quite efficient in generating optimized RTL code for FPGA implementation. With enough research in optimization strategies, high level synthesis tools can complement traditional RTL design methodologies if not completely replace them for ASIC designs.

In this work, we chose to use a high level design methodology primarily to assess the efficacy of this important technological paradigm and also to demonstrate the benefit this methodology can bring to the over-all design methodology by providing a fast convergence from idea to hardware implementation.

3. Design Approach

To design the hardware, a novel design approach has been followed based on high level synthesis methodology. The complete flow of our approach is shown in Figure 1. Starting with the formulation of specification, various constraints were ascertained which the proposed hardware was required to satisfy. These constraints and specifications mainly pertained to the high speed operation of the circuit and included parameters like throughput, latency, maximum data rates. Specifications and constraints determined at this stage were still rough estimates. However, these rough estimates are not the ones used by the synthesis tool as the starting point of the optimization. The synthesis tool has its own library for this purpose.

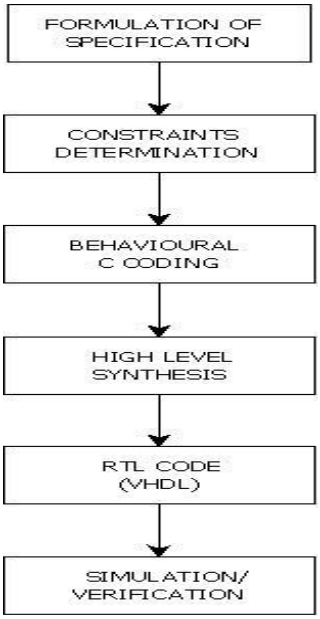

Fig. 1: Complete Flow of Design Methodology

Once the constraints are determined, the C code that shows the behaviour of the hardware is written. Usually the C code contains the operations that the hardware is intended to perform which in this case consists of only comparisons. However, these comparisons are needed to be performed on two input vectors in parallel using pipelined stages. The C code is written keeping this in mind.

The tool that we have used for this work is GAUT II. The GUI of this software can be seen in Figure 2. It is an open source high level synthesis tool developed at LESTER Lab at France. It accepts behavioural VHDL or C code as input and generates IEEE P1076 compliant RTL VHDL code for synthesis as output [6]. The code generated can then be used for synthesis and verification by virtue of simulation which are the last steps of the design approach that has been followed. For more details on GAUT, readers can see [7] and [8].

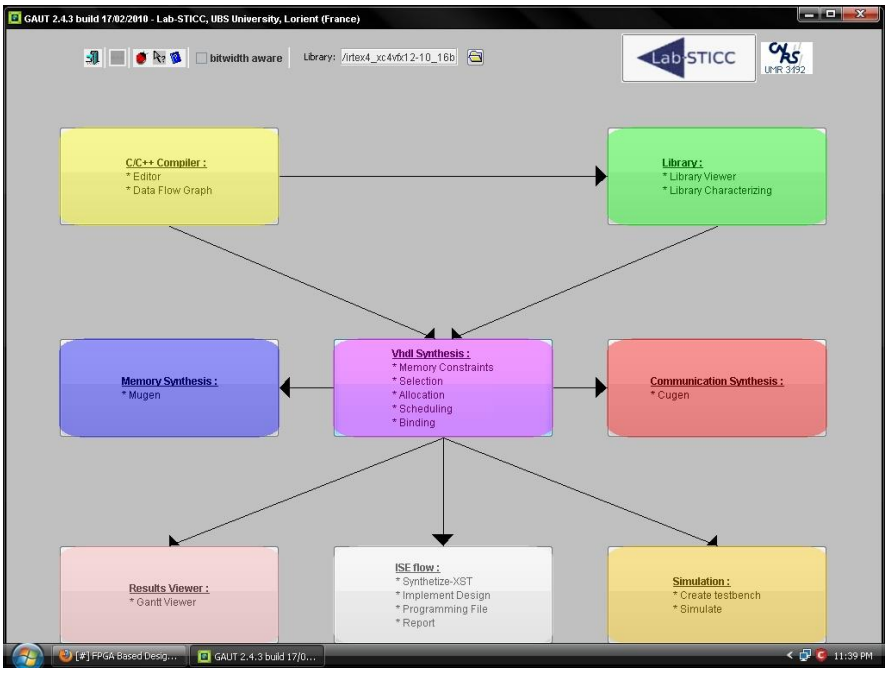

Fig. 2: GUI of the GAUT II High Level Synthesis Tool

4. Proposed Architecture of the Hardware

The hardware has been designed keeping few assumptions in mind. The first assumption is the bit encoding of the four nucleobases A, T, C and G as shown in Table 1. Having done this bit encoding, the input stream of data will simply become a long string of 0s and 1s. However, one cannot simply do a bit by bit comparison as then it will produce wrong results which will not make any sense. We have designed the hardware keeping this in mind. The hardware accepts data serially which are then placed in input registers in groups of two to preserve the bit encoding of the nucleobases. The comparator array then compares these 4 bits (2 for sample base and 2 for target base) and produces a 1 for valid match or a 0 otherwise. This extensive requirement is reflected in our C code by using bit-accurate variables from the Algorithmic C class library from Mentor Graphics which is supported by GAUT tool.

Table 1: Bit Encoding of the Nucleobases

Nucleobase	Bit Representation
A	00
T	01
C	10
G	11

Another assumption that we have made in our design is that the bit encoding is not performed in our hardware but by a software routine run on a general purpose computer with which this hardware is integrated. The string matching algorithms as discussed in the beginning of the paper can be written in a high level language and may include a function that does this bit encoding of input data streams. As the software is executed, our hardware will receive this bit encoded data and then function as intended.

The RTL code as generated by GAUT II from the input C code gives rise to architecture of the form as shown in Figure 3.

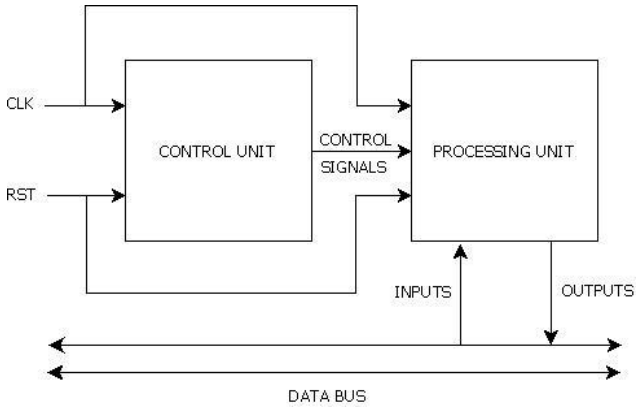

Fig. 3: Proposed General Architecture of the Hardware

The control unit comprises of an FSM that controls and schedules all the operations carried out by the processing unit. The processing unit comprises of pipelined comparator arrays that carries out comparisons between input data which are stored in input registers. The top level dataflow graph of the processing unit is shown in Figure 4 and the generated RTL VHDL code opened from within GAUT GUI is shown in Figure 5.

The 24 green points on the left show the two sets of input data (each a vector of 12 data points). The two orange points are constants, one being 0 and the other being 1. All these inputs are connected with comparator arrays (blue points in the middle) and produce the output which are stored in output registers and shown by 12 yellow points on the right.

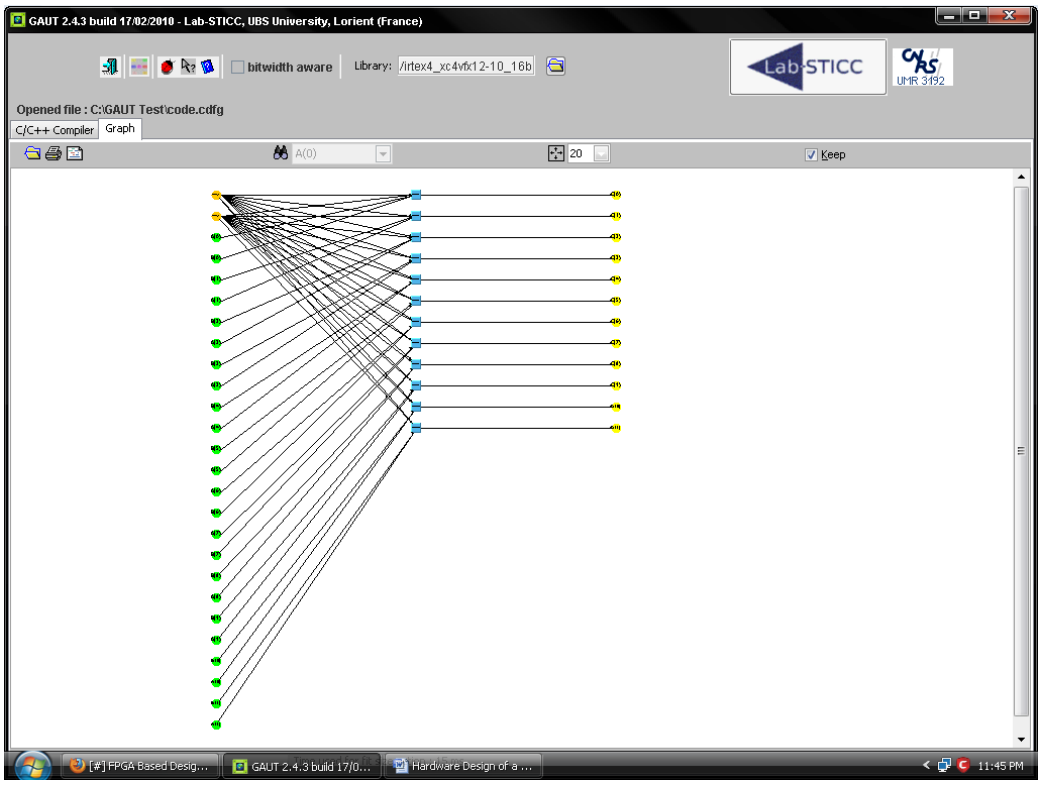

Fig. 4: Top level Dataflow Graph of the Processing Unit

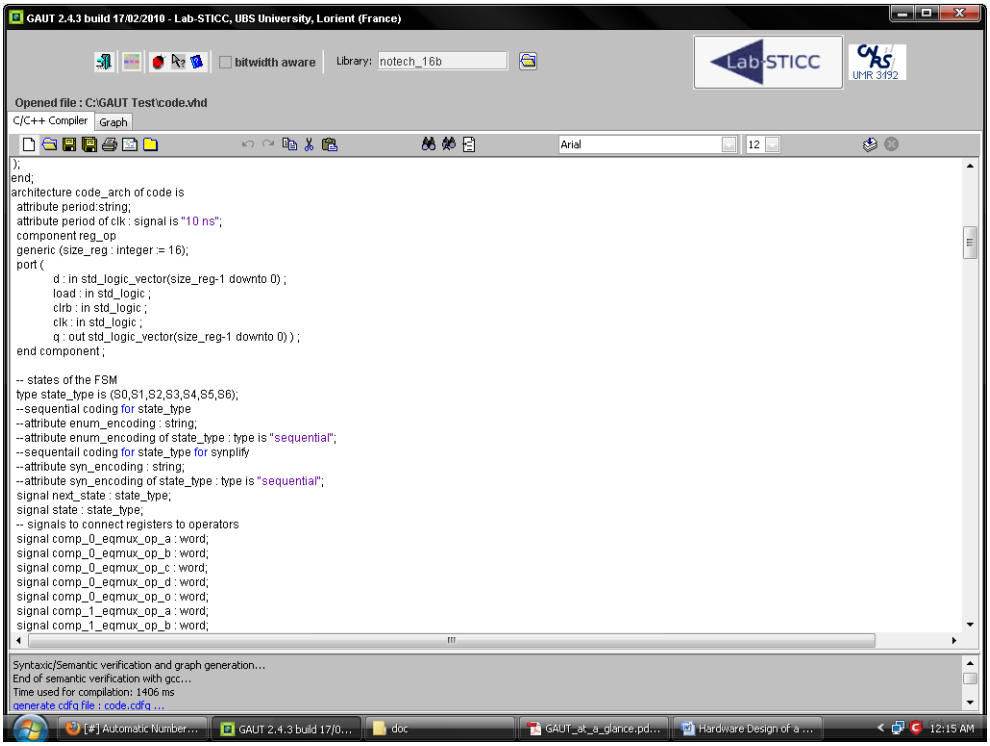

Fig. 5: Generated RTL VHDL code opened from within GAUT GUI

The Gantt chart shows the exact scheduling of the data through various registers and is shown in Figure 6. From here it is evident that the number of pipeline stages is 2. The choice of the number of pipeline stage is determined to satisfy design metrics like speed, area, power etc. For a DNA sample containing approximately 1 million base pairs, the estimated time the hardware takes to compare it is 0.006666666 seconds excluding of course the time it takes for the software routine to run whose running time will depend on the length of the underlying code and the processor on which it is run.

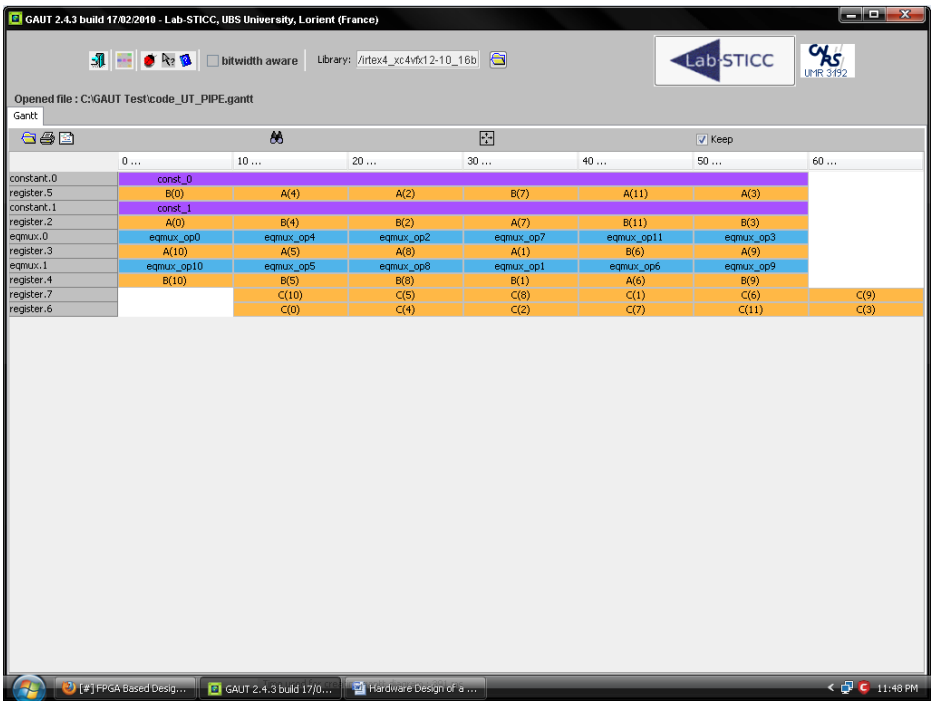

Fig. 6: Gantt chart of the Operation

5. Hardware Characteristics

The characteristics of the designed hardware are shown in Table 2. As can be seen from the Gantt chart, it takes 7 clock cycles to get the output. For clock duration of 10 ns, the output is received after 70 ns which is the latency of the hardware. The hardware synthesis result in GAUT tool is shown in Figure 7.

Table 2: Hardware Characteristics

Parameters	Value
Clock	10 ns
Latency	70 ns
Pipeline Stages	2
Internal data registers	6

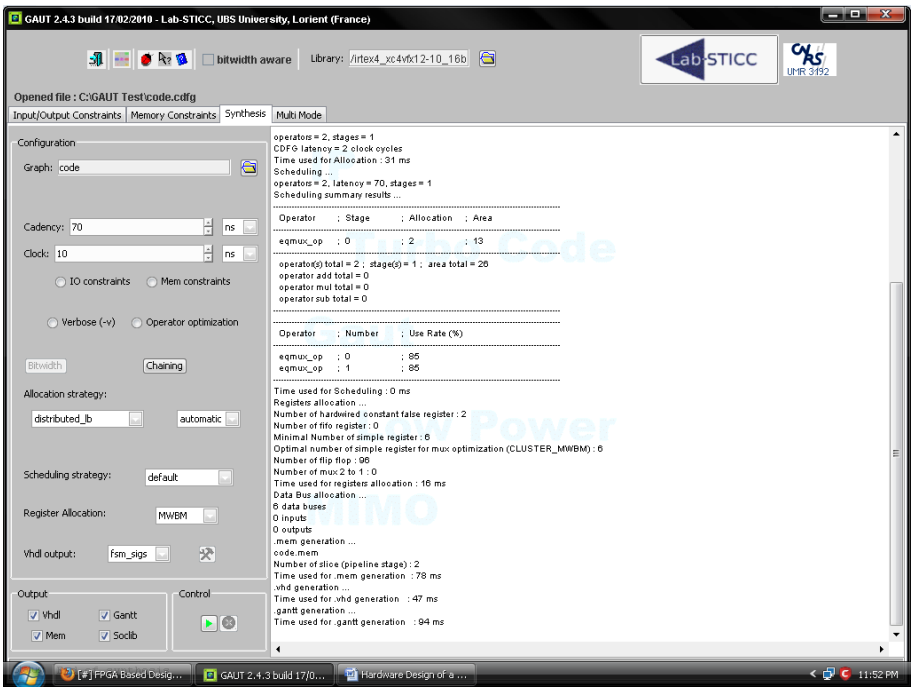

Fig. 7: Hardware Synthesis Result in GAUT Tool

Keeping fixed the clock period at 10 ns, we compared the latency of the design with the area estimate as provided by the GAUT II tool. The details are shown in Figure 8 below. The x-axis represents the latency while the y-axis represents the area estimate provided by GAUT tool.

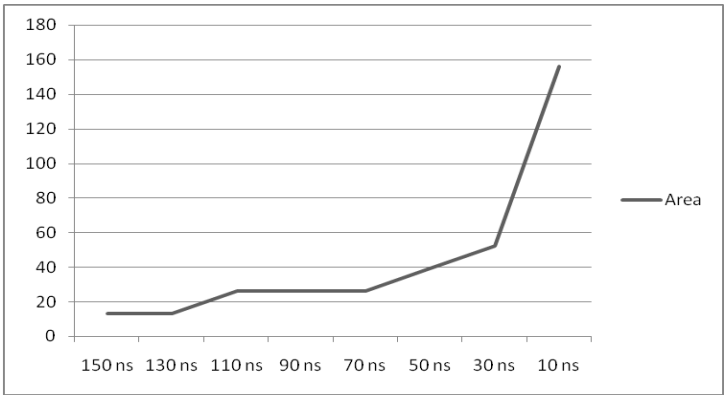

Fig. 8: Area v/s Latency graph

From the graph above, it can be seen that the area increases drastically for the latency period equal to clock period which is 10 ns. For all other latencies the corresponding area estimates are given. This validates the fact that lower the latency is, higher is the logic circuit size. The graph above also helped us decide the particular hardware implementation which had a latency of 70 ns which is quite optimum both in terms of area and speed. This could not have been known so easily with traditional RTL based design methodology where it would require one to develop RTL code multiple times to find out the best possible implementation of the hardware not feasible with stringent time-to-market deadlines prevalent in the industry.

6. Hardware Implementation

The generated RTL description can be used as an input to various synthesis tools for gate-level synthesis. It may be used for implementation on an FPGA also. However, prior to RTL generation one needs to characterize the operators in appropriate library files for the target device (in case of FPGAs) or target processes in case of ASIC. By default, GAUT tool provides library characterization for Xilinx and Altera FPGA families.

The total area or resources consumed on specific FPGA device will be more than what GAUT indicates for the processing unit. This is because other interfacing circuitry will also be synthesized like communication unit, GALS/LIS interface, memory unit etc. With GAUT tool it is possible to have a memory aware synthesis methodology [9] that can further allow us to design fast memory access based comparison framework by incorporating certain string matching algorithms directly in hardware. This research avenue is currently being pursued by the authors.

GAUT tool is also capable of generating SystemC cycle accurate simulation model which is SystemC 2.1 compliant. Hence these simulation models can also be used for virtual prototyping using SoCLib platform.

7. Conclusion

Here in this paper we have shown how high level synthesis methodology can be used to obtain an optimized and efficient hardware implementation in the form of a high speed comparator framework targeted for DNA sequencing needs. High level synthesis allowed us to make several important choices pertaining to hardware performance metrics at a fairly early stage than was possible with traditional RTL based methodologies and allowed us to design the hardware satisfying all design constraints. The hardware has the potential to be a fast and cost-effective solution that can be integrated with software routines to develop a high speed DNA sequencing tool.

References

[1] J. D. Watson and F.H.C Crick, “A Structure for Deoxyribose Nucleic Acid", Nature, 171, 1953, pp. 737 – 738.

[2] J. D. Watson and F.H.C Crick, “Genetical Implications of the Structure of Deoxyribonucleic Acid", Nature, 171, 1953, pp. 964 – 967.

[3] R. Bhukya and DVLN Somayajulu, “2-Jump DNA Search Multiple Pattern Matching Algorithm”, International Journal of Computer Science Issues (IJCSI), Vol. 8, Issue 3, No. 1, 2011, pp. 320 – 329.

[4] G. Martin and G. Smith, “High Level Synthesis: Past, Present and Future”, IEEE Design and Test of Computers, 2009, pp. 18 – 23.

[5] David C. Zaretsky et. al., “Balanced Scheduling and Operation Chaining in High Level Synthesis for FPGA Designs”, Proceedings of the 8th International Symposium on Quality Electronic Design, 2007, pp. 595 – 601.

[6] Official website of GAUT -- http://www-labsticc.univ-ubs.fr/www-gaut/

[7] E. Martin et al., “GAUT: An Architecture Synthesis Tool for Dedicated Signal Processors”, Proceedings of EURODAC, 1993, pp. 14 – 19.

[8] Julien et al., “Low Power Synthesis Methodology with Data Format Optimization Applied on a DWT”, Kluwer Academic Journal of VLSI Signal Processing, 2003, pp. 195 – 211.

[9] G. Corre et al., “Memory Aware High Level Synthesis for Embedded Systems”, IADIS International Conference on Applied Computing, 2004.