Design and Analysis of DNA Fragment Assembly's Converter and Checker Modules using De Bruijin Graph

Mohd Shafiq B Mohd Helmi
Faculty of Electrical Engineering
Universiti Teknologi MARA,
40450 Shah Alam, Selangor, Malaysia
Email: shafiqhelmi@aol.com

Abstract:- **This paper presents the Design and Analysis of DNA Fragment Assembly's Converter and Checker Modules using De Bruijin Graph. There are several objectives of this paper. The first objective is to design and analyze edges and vertices converter. Secondly, this paper will also discuss about designing and analyzing common edge and branch checker. In addition, all the modules will be integrated in one top module. Lastly, the integrated module will be simulated in Xilinx ISE software. Based on the objectives, this paper will discuss on constructing a DNA Fragment Assembly module that consist of four submodules which is Edges Converter, Vertices Converter, Common Edge Checker and Branch Checker. The sequencing method used in this paper is Hybridization method. Hybridization method is an option to shotgun sequencing. Hybridization uses the array identifying algorithm to identify the sequence. The algorithm used to construct the module is Bruijn Graph. De Bruijn Graph represents sequences of symbols from an alphabet, and edges that indicate where the sequence may overlap. First, the set of DNA will inserted in to the module, then the module will do all the converting and checking process of the input before produce the output of vertices and the branch of edges. All the algorithm and theory then been simulated in Xilinx to test the functionality.**

Keyword: Deoxyribonucleic Acid (DNA), Hybridization, Shotgun, Edge, Vertices, Common Edge and Branch Edges.

I. INTRODUCTION

Fragment assembly is a typical approach to sequencing long DNA molecules to sample and then sequence fragments from them. The objective of DNA Fragment assembly to obtain the DNA sequence using certain hints, which are (approximate) substrings of the row [1].

What is DNA? DNA is a long polymer made from repeating units called nucleotides [2]. From a chemical point of view DNA consist of variable side groups of Adenine (A), Cytosine (C), Guanine (G), and Thymine (T) [3].

For the last two decades, DNA fragment has followed the concept of overlap-layout-consensus [2].

7. **Overlap**: matching all possible find any overlapping
8. **Layout**: reads along DNA and putting them together
9. **Consensus**: deriving how sequence will appear based on layout

There will be some problem of doing the DNA sequencing. First, there could be multiple ways to reconstruct the original strand out of the fragment pieces, or "snippets," and only one of which is correct [1-2]. In addition, the overlap stage finds pair wise similarities that do not always provide true information on whether the fragments (sequencing reads) overlap [2]. Furthermore, if a repeating sequence is larger than the size of the viewable reads, it would make construction of the genome almost impossible [1].

The best solution that researcher has come into is to do in graph theory, specifically Eulerian Paths and de Bruijn Graphs, help most of the researchers to see the possible conclusions about the problem regarding reassembled strands of DNA.

As mentioned in abstract, de Bruijn Graph represents sequence. Dutch mathematician Nicolaas de Bruijn found that a cyclic sequence of letters taken from a given alphabet for every word of a certain length (k) appears in the cyclic sequence exactly once [3].

Bruijn Graph's idea is like a stair. For an example taken from [1], the input sequence is 0110101. The length set to be 3 (k = 3). The first output sequence to be 011 since the length set to be 3 which are 0, 1 and 1. The next output sequence will start at the second input, so the output sequence is 110. By looking at all the output sequence at Figure 1, de Bruijn Graph will create outputs sequence like a stair. Figure 2 shows the de Bruijn Graph that coincide with the sequence [1].

```
0   1   1
    1   1   0
        1   0   1
            0   1   0
                1   0   1
```

Figure 1: The Output Stair Concept of De Bruijn Graph For A Sequence '0110101'

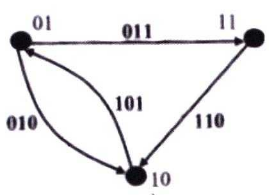


Figure 2: The De Bruijn Graph for the Sequence '0110101' With Fragment of Length 3

The concept of Bruijn Graph also can be applied in DNA Fragment Assembly. Take the DNA group of Adenine (A), Cytosine (C), Guanine (G), and Thymine (T) as the input [4]. The group of ACGT can be the input sequence of the de Bruijn Graph.

For an example taken from [5], the input sequence of the de Bruijn Graph is ATGGAAGTCGCGGAATC. The length is set to be 7 (k = 7). The output sequence of the input refers at Figure 3.

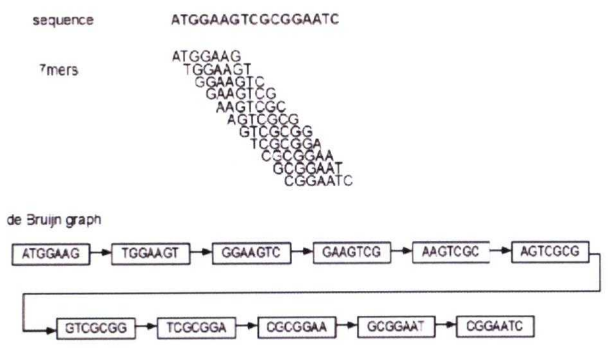


Figure 3: The De Bruijn Graph for the Sequence 'ATGGAAGTCGCGGAATC' With Fragment of Length 7

Base on the concept of de Bruijn Graph, there are 11 output sequences of the input ATGGAAGTCGCGGAATC consist of ATGGAAG, TGGAAGT, GGAAGTC, GAAGTCG, AAGTCGC, AGTCGCG, GTCGCGG, TCGCGGA, CGCGGAA, GCGGAAT and CGGAATC.

The studies of de Bruijn Graph is very important because the concept of de Bruijn Graph will be use before the sequencing part in the DNA Fragment Assembly. During the sequencing part, the Euler's technique will be use to rearrange the edges and the vertices that has been created during the converter's and the checker's part based on the de Bruijn Graph [6].

Eulerian Paths is created by Leonhard Euler while solving the famous Seven Bridges of Königsberg problem in 1736 and the first complete proof of this idea was published in 1873 by Carl Hierholzer [7].

A path is said to be open if the starting vertex is different from the ending vertex and a path is said to be closed if the starting and ending vertices are the same vertex [8]. Both open and closed Eulerian path has the same condition. The condition is to visits each edge in a graph exactly once [1]. Figure 4 and Figure 5 show the open and the closed Eulerian Path [9].

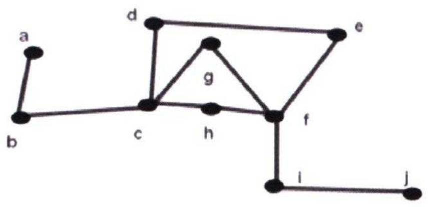


Figure 4: Open Eulerian Circuit of the input 'a-b-c-d-e-f-g-c-h-f-i-j'

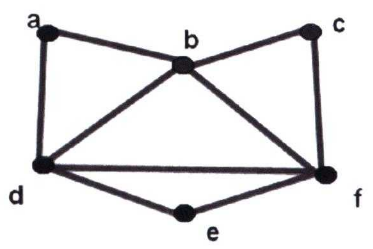


Figure 5: Close Eulerian Circuit of the input 'a-b-c-d-e-f'

II. METHODOLOGY

DNA Fragment Assembly consists of 3 parts. The first part is the converter, the second part is to check and the last part is to sequence the input. The converter part consists of two modules, edges converter and vertices converter. The second part is the checker. The checker is including common edge checker and branch checker. The last part is the sequencer. The sequencer will sequence all input from branch checker and vertices converter. Refer Figure 6.

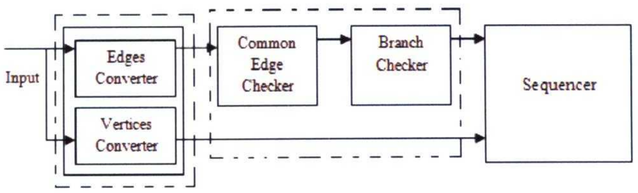


Figure 6: The Block Diagram of Complete DNA Fragment Assembly

This paper will only discuss on the first and the second part of DNA Fragment Assembly that consists of Edge Converter, Vertices Converter, Common Edge Checker and Branch Checker. Refer Figure 7.

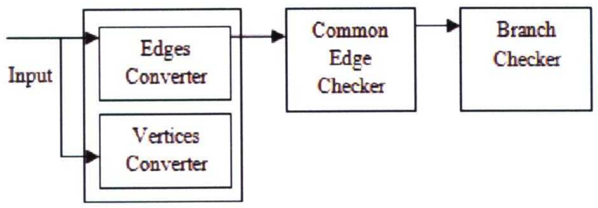


Figure 7: The Block Diagram of the Converter and Checker of DNA Fragment Assembly

A set of input strand will be injected into the first block that consists of Edge Converter and Vertices Converter. Then, the output of edge converter, the edges, will go in to the common edge checker. In the common edge checker module, the edges will be check side by side for the similarity. The last module is the branch checker. What branch checker do is to check for the branch among the edges. So the output of this paper will be the output from the vertices converter and the output from the branch checker. Further information will be discussed later on this paper.

The input A, C, G and T will be transfer into binary to make the coding easier. Each of the DNA elements will represented by 3 bits binary as the Table 1 below,

TABLE 1. BINARY REPRESENTATIVE OF THE DNA STRAND

DNA Element	3 Bits Binary
A	000
C	001
G	010
T	011

The four modules of DNA Fragment Assembly for this paper clearly will do different task in order to complete the system requirement.

There are some limitations in this project design. The first limitation is the maximum number of strands that can be accepted by the module is 5. For an example, it's only accepting input strands such as AACTG. Furthermore, the branch checker will only check for the existence of branch but it not put the branch in the place it's should be which is between the edge. Next, the paper will discuss a little more about the modules exist in the Converter and Checker of DNA Fragment Assembly.

A. Edge Converter

Edge Converter is the first module that the input will go through. The input from the DNA is a very long sequence. Before further steps, the 5 inputs first been converted in to binary. For an example if the input sequence (strand) is S = {A, T, A, T, G}, then the binary input will be,

A – 000
T – 011
A – 000
T – 011
G – 010

So, the full sequence is **S = 000011000011010.**

Edge Converter will convert the input S into several edges base on Bruijn Graph. The length of the edges has been set to be 2 (k = 2). What the statement means that, the stair (refer to the stair concept in the introduction) will stop after 2 inputs have been met. Refer Figure 8.

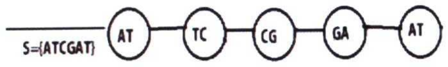


Figure 8: The Set of Edges for the Input of 'ATCGAT'

For the input strand, S = {A, T, A, T, G}, the edges are AT, TA, AT and TG based on the de Bruijn Graph algorithm with the length of 2. Refer Figure 9. The figure shows that how de Bruijn Graph with the lengths of 2 works. Every 2 inputs, it will produce one output then change to the next input to convert the second output.

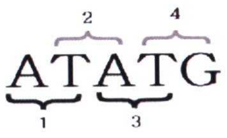


Figure 9: The Edges Converter Concept Base On De Bruijn Graph

The Edge Converter will convert whatever input, S, inserted in to the system. From 5 inputs strands, it will detect and covert all of edges available. For example we did before, the Edge Converter will convert into 4 edges, E3, E2, E1 and E0 then convert it into binary numbers as Table 2 below.

TABLE 2. BINARY EDGES

Output	Edge	Binary Edge
E3	AT	000011
E2	TA	011000
E1	AT	000011
E0	TG	011010

B. Vertices Converter

Vertices are the DNA element that connects the two edges. Same as Edge Converter, Vertices Converter will convert input strand (S) into several vertices in term of binary number. The edges will have the length of 2 snippets while the vertices will have the length of 3 snippets.

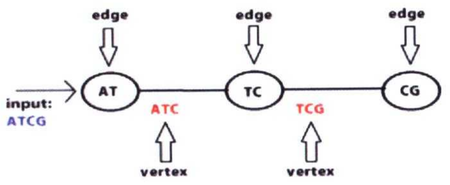


Figure 10: Example of Vertices of the Input Sequence 'ATCG'

Figure 10 shows that an example of the vertices. The figure shows that the vertices are the combination of 2 edges. The edges are AT, TC and CG. From the edges AT and TC, the vertex is ATC since it is the combination from the two edges. It is the same thing happen at the edges TC and CG.

From the previous example on the Edge Converter, S = {A, T, A, T, G}, the vertices will be as Table 3.

TABLE 3. BINARY VERTICES

Output	Edge	Vertex	Binary Vertices
E3	AT		
V2		ATA	000011000
E2	TA		
V1		TAT	011000011
E1	AT		
V0		ATG	000011010
E0	TG		

The same concept applied to the vertices converter. Same as edge converter, de Bruijn Graph also applied to this module. The only different to the edge converter is that, it has the length of 3 but the edge converter only has the length of 2.

C. Common Edge Checker

Common Edge Checker is the module that will check the same edges side by side. It means that, it will check between E3 and E2, E2 and E1 and lastly E1 and E0. If there is the similarity of the side edge, one of the edges will be eliminate and go into the common edge output, CE.

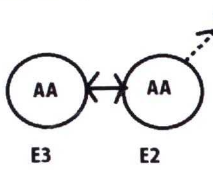


Figure 11: Common Edge

From Figure 11, E3 and E2 from Edges Converter output, are comparing each other to see whether they are the same or not. If they are the same, E2 will go into CE2. Figure 12 shows flowchart of this module.

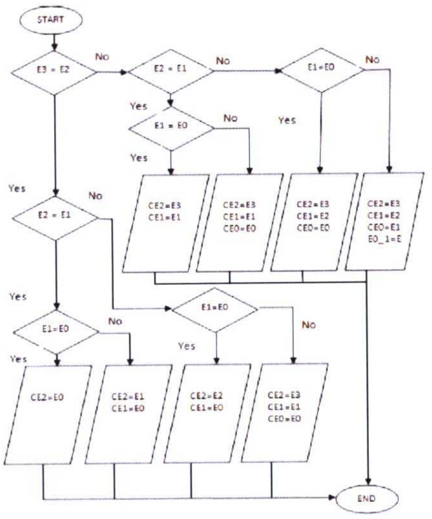


Figure 12: Flowchart of A Common Edge

D. Branch Checker

The last module of the DNA Fragment Assembly is the Branch Checker. Branch Checker get the input from the output of Common Edge Checker which is CE2, CE1, CE0 and E0. What branch checker do is that it will detect the branch in the input strand.

What is branch? Branch is the edge between the two same edges.

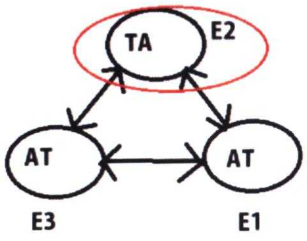


Figure 13: Branch

Branch Checker will compare two edges. It's different from Common Edge Checker because it will compare two edges that is not side by side to it. Figure 13 shows that, edge E3 will compare to edge E1. If E3 and E1 are the same, edge that is in between E3 and E1, which is E2, will become the branch. Figure 14 shows the flowchart of branch checker.

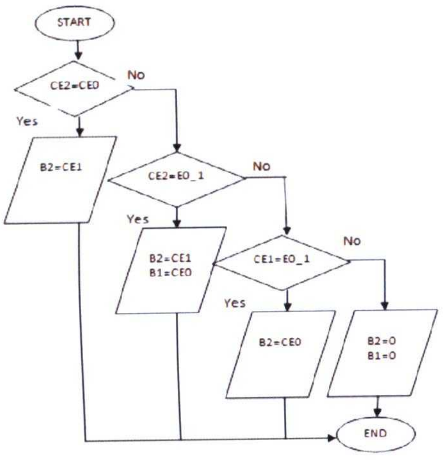


Figure 14: Flowchart of A Branch Checker

III. RESULTS AND DISCUSSION

All the modules have successfully integrated and simulated using Verilog coding in Xilinx ISE Design Suite [10-14]. All of the modules behave as it should. Figure 15 shows the RTL Schematic of DNA Fragment Assembly.

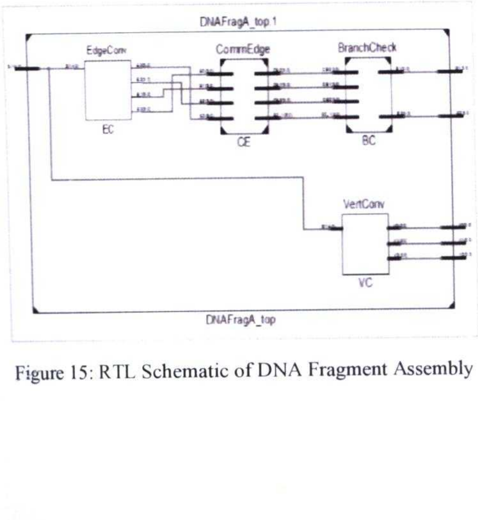


Figure 15: RTL Schematic of DNA Fragment Assembly

Before the test has been done to the overall module (complete Converter and Checker for DNA Fragment Assembly), the test had been done to the submodule in it. The first test has been done to the Edge Converter. Figure 16 shows the Technology Schematic of the Edge Converter. Edges Converter will get the input with 5 strands which is 15 bits (3 bits per strand). The input named as S. The output of the Edges Converter is E3, E2, E1 and E0.

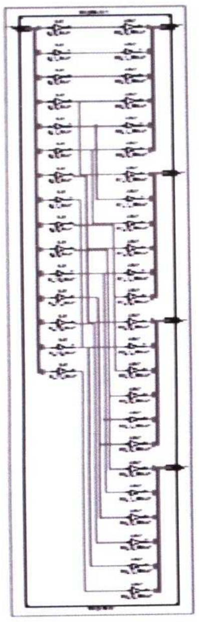


Figure 16: Technology Schematic of the Edges Converter

To test the functionality of the module, several sets of 15 bits inputs has been entered into the test fixture of Xilinx ISE. Below is the figure of one set of input that has been entered into the module which is S = 000011000011001. Base on this input, we can that, the DNA elements that has been entered are ATATC.

From the theory of an Edge Converter at the methodology part, Edges Converter will convert the input into parts with the length of 2 based on de Bruijn Graph. From the input ATATC above, what we should get in the Edges Converter are AT, TA, AT and TC. Table 4 shows the result that should been get by the simulations and Figure 20 shows the result of the input and the output of the Edge Converter of the simulation.

TABLE 4. BINARY EDGES OF THE INPUT ATATC

Output	Edge	Binary Edge
E3	AT	000011
E2	TA	011000
E1	AT	000011
E0	TC	011001

The functionality of the Edges Converter has been proved because the theoretical values have the same value with the simulation values.

Next, the input strands S, will go through the Vertices Converter. Figure 17 shows the Technology Schematic of the Vertices Converter. Vertices Converter will get the input with 5 strands which are 15 bits (3 bits per strand). The input named as S. The output of the Vertices Converter is V2, V1 and V0. Vertices Converter will convert the input into parts with the length of 3 based on de Bruijn Graph. From the input ATATC above, what we should get in the Vertices Converter are ATA, TAT and ATC.

Table 5 shows the result that should been get by the simulations and Figure 20 shows the result of the input and the output of the Vertices Converter of the simulation.

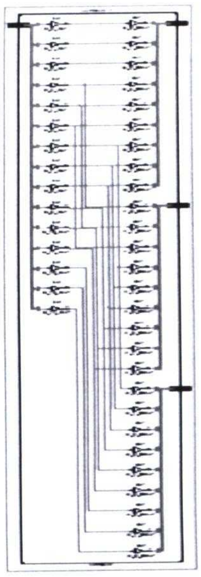


Figure 17: Technology Schematic of a Vertices Converter

TABLE 5. BINARY VERTICES OF THE INPUT 'ATATC'

Output	Vertex	Binary Vertices
V2	ATA	000011000
V1	TAT	011000011
V0	ATC	000011001

The output of Edges Converter then will undergo the checker module consist of Common Edges Checker and Branch Checker. The output of Edges Converter first will go to Common Edges Checker. Figure 18 shows the technology schematic of a Common Edges Checker.

Common Edges Checker will check for the same edges side by side with the other edge. For an example there are two edges AA and AA side by side. Common Edge Checker will detect the second edge and delete in from the module.

The input of Common Edge Checker is E3, E2, E1 and E0 from the output of the Edges Converter. The outputs of this module are CE2, CE1, CE0 and E0_1. Since the edges from the Edges Converter are AT, TA, AT and TC, there is no common edge since AT is not the same as TA, TA is not the same as AT and AT is not the same as TC. So, the result of the common edges is the same as the result from the Edges Converter since there is no edge has been eliminate. Table 6 shows the result that should been get by the simulations and Figure 20 shows the result of the input and the output of the Common Edges Checker by the simulation.

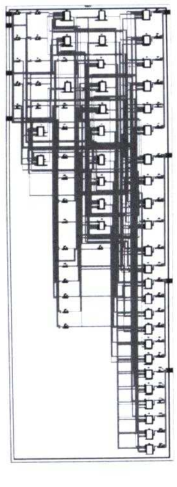


Figure 18: Technology Schematic of a Common Edge Checker

TABLE 6. BINARY COMMON EDGES OF THE INPUT 'ATATC'

Output	Edge	Binary Edge
CE2	AT	000011
CE1	TA	011000
CE0	AT	000011
E0_1	TC	011000

The output of Common Edges Checker then will undergo the Branch Checker's module. The module will check for the branch. As has been explain before at the methodology part, branch is the edge between the two same edges. The input of the Branch Checker will be CE2, CE1, CE0 and E0_1 as there are from the output of Common Edges Checker and the output is B1 and B0. Figure 19 shows the technology schematic of a Common Edges Checker.

The input of the Branch checker are AT, TA, AT and TC. Since the first edge, AT, is the same with the third edge, AT, so TA (that exist in the middle between the two AT) will be the branch.

Table 7 shows the result that should been get by the simulations and Figure 20 shows the result of the input and the output of the Branch Checker of the simulation.

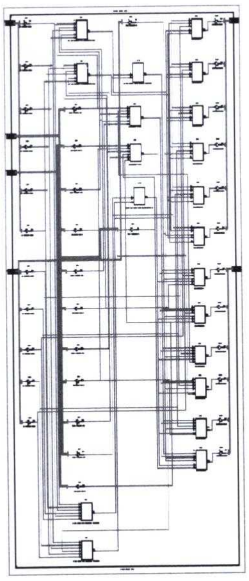


Figure 19: Technology Schematic of a Branch Checker

TABLE 7. BINARY REPRESENTATIVE OF BRANCH OF THE INPUT 'ATATC'

Output	Edge	Binary Edge
B2	TA	011000
B1	0	0

Figure 21 shows the result of the input and the output of the whole Converter and Checker of DNA Fragment Assembly of the simulation.

IV. CONCLUSIONS

From the theoretical and simulation results, it was found that the checkers and the converters of DNA Fragment Assembly that have been designed and analyzed are working correctly. Edges and Vertices converter have been converted the 15 bits input strands to small segment with the length of 2 for the edges and length of 3 for the vertices. The Common Edges Checker and also Branch checker have been checked for any common edge existence and the branch existence. The entire modules have been integrated in one big module called DNA Fragment Assembly. All of the modules have successfully been simulated in verilog coding using Xilinx ISE software.

V. RECOMMENDATIONS FOR FUTURE WORK

De Bruijn Graph is highly recommended in the DNA Fragment Assembly since it has precise in cutting the long strands in to small segment. It is also easy to understand. For the further analysis to improve the design, it is recommended to use a longer input strands as the input because the real DNA has a very long sequence to be tested. Furthermore, it is also recommended that to combine both Common Edge Checker and Branch Checker in to one module to reduce the size of the design.

VI. ACKNOWLEDGMENT

I am thankful to my supervisors, Puan Tuan Norjihan Tuan Yaakub and Encik Abdul Karimi Halim, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding and achieving the goal of this project. Next to my friends, Aqilah Razali and Fadzilatul Husna whose have help me in understanding and completing the project. Lastly, I offer my regards and blessings to my parents and all of those who have supported me in any respect during the completion of this project.

VII. REFERENCES

[1] J. Kaptcianos. ***Graph Theory Aiding DNA Fragment Assembly***, St. Michael's College Colchester, Vermont, USA Volume 7, September 19 2008.

[2] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, & P. Walter, ***Molecular Biology of the Cell, 4th edition***, New York: Garland Science; 2002.

[3] P. A. Pevzner, H. Tang & M. S. Waterman. ***A New Approach to Fragment Assembly in DNA Sequencing***, Los Angeles, CA, USA, 2001.

[4] C. Sugnet, ***DNA: Structure and Function***, Internet: http://users.soe.ucsc.edu/~sugnet/documentation/biology_starter/DNA.html, University Of California Santa Cruz.

[5] Homologus Frontier, ***De Bruijn Graphs***, Internet: http://www.homolog.us/blogs/2011/07/28/de-bruijn-graphs-i/, July 28th, 2011.

[6] H. Fleischner, ***Eulerian Graphs and Related Topics***, Elsevier Science, London, 1990.

[7] L. B. H. Victor, ***Eulerian Path And Circuit***, January 24, 2010.

[8] P. E. C. Compeau, P. A. Pevzner & G. Tesler, ***How To Apply De Bruijn Graphs To Genome Assembly***, Nature Biotechnology 29, 987–991 (2011)

[9] N. L. Biggs, E. K. Lloyd and R. J. Wilson, ***Graph Theory***, Clarendon Press, Oxford, 1976.

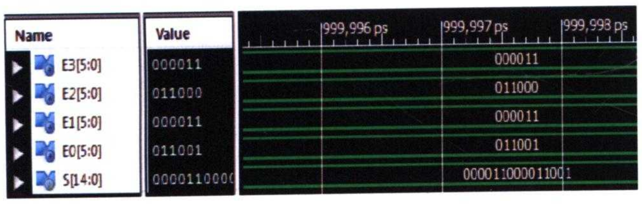
[10] F. Vahid (University of California, Riverside) & Roman Lysecky (University of Arizona), ***Verilog for Digital Design***, Wiley Bicentennial, 2007.

[11] T. R. Padamanabhan & B. B. T. Sundari, ***Digital Through Verilog HDL***, Wiley Interscience, 2003.

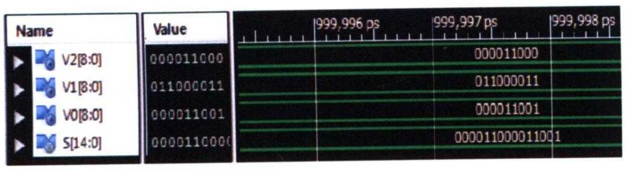
[12] D. E. Thomas & P. R Moorby, ***The Verilog Hardware Description Language***, Kluwer Academic, 1998.

[13] S. Golson, Carlisle MA, ***State Machine Design Techniques For Verilog and VHDL***, Carlisle MA, USA, 1994.

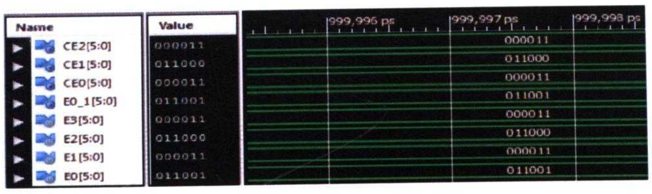
[14] D. M. Harris & S. Harris, ***Digital Design & Computer Architecture***, Morgan Kaufmann, Mar 2, 2007.


```
Name       Value        999,996 ps   999,997 ps   999,998 ps
E3[5:0]    000011       000011
E2[5:0]    011000       011000
E1[5:0]    000011       000011
E0[5:0]    011001       011001
S[14:0]    0000110000   000011000011001
```

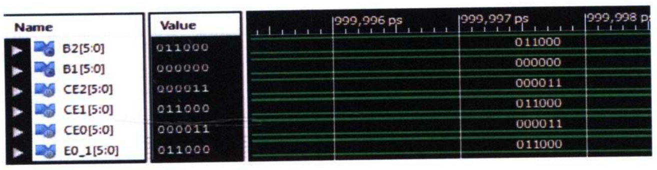
(a)


```
Name       Value        999,996 ps   999,997 ps   999,998 ps
V2[8:0]    000011000    000011000
V1[8:0]    011000011    011000011
V0[8:0]    000011001    000011001
S[14:0]    0000110000   000011000011001
```

(b)


```
Name        Value     999,996 ps   999,997 ps   999,998 ps
CE2[5:0]    000011    000011
CE1[5:0]    011000    011000
CE0[5:0]    000011    000011
E0_1[5:0]   011001    011001
E3[5:0]     000011    000011
E2[5:0]     011000    011000
E1[5:0]     000011    000011
E0[5:0]     011001    011001
```

(c)


```
Name        Value     999,996 ps   999,997 ps   999,998 ps
B2[5:0]     011000    011000
B1[5:0]     000000    000000
CE2[5:0]    000011    000011
CE1[5:0]    011000    011000
CE0[5:0]    000011    000011
E0_1[5:0]   011000    011000
```

(d)

Figure 20: The Simulations Results Of The Modules Of The Checker And The Converters Of DNA Fragment Assembly With The Input Of 'ATATC': (a) The Result of the Edges Converter. (b) The Result of the Vertices Converter. (c) The Result of the Common Edges Checker. (d) The Result of the Branch Checker.

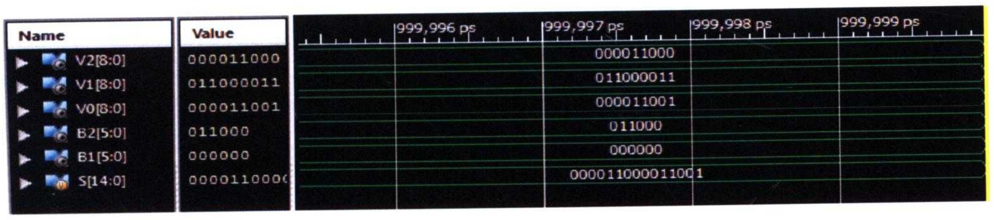

```
Name       Value        999,996 ps   999,997 ps   999,998 ps   999,999 ps
V2[8:0]    000011000    000011000
V1[8:0]    011000011    011000011
V0[8:0]    000011001    000011001
B2[5:0]    011000       011000
B1[5:0]    000000       000000
S[14:0]    0000110000   000011000011001
```

Figure 21: The Result of the Input and the Output of the Whole Converter and Checker of DNA Fragment Assembly