논문 2006-43SC-5-6

유전자 알고리즘을 이용한 DNA 서열 생성 시스템의 효율적인 구현에 대한 연구

(Implementation of efficient DNA Sequence Generate System with Genetic Algorithm)

이 은 경*, 이 승 렬*, 김 동 순***, 정 덕 진**

(Eun-Kyung Lee, Sung- Lee, Dong-Soon Kim, and Duck-Jin Chung)

요 약

DNA 컴퓨팅은 컴퓨터의 계산 수준을 분자 수준으로 끌어내려 막대한 병렬성을 확보 하고, 보다 효율적인 정보 처리를 가능케 해 차세대 컴퓨팅 기법으로서의 위치를 확고히 하고 있다. 그러나 DNA 컴퓨팅은 실제 실험을 통해 계산 모델 및 알고리즘을 검증하기 때문에 많은 연산 시간을 필요로 한다. 따라서 빠른 계산 모델 및 알고리즘의 검증을 위해 시뮬레이터인 NACST가 개발되었다. 그러나 NACST에 포함된 서열생성 시스템의 반복적인 연산 특징 때문에 이 또한 많은 연산시간을 필요로 하게 되었다. 따라서 시뮬레이션 시간 단축을 위한 서열생성 시스템의 효율적인 하드웨어 구조가 요구 된다. 이에 본 논문은 DNA 코드 최적화 부분의 연산시간이 NACST 연산시간의 약 95% 이상을 차지한다는 점을 착안하여 DNA 서열 생성 시스템에 병렬 기법과 Pipeline 기법을 적용하였고 적합도 함수 간 연산을 공유시켜 연산의 양을 대폭 줄이고 분배해 시뮬레이션 시간을 크게 줄일 수 있는 하드웨어 구조를 제안하고 검증 하였다. 실험 결과 제안된 하드웨어는 기존 소프트웨어에 비해 약 467배 이상의 연산시간 감소를 보였으며 DNA 서열 생성 성능은 기존과 동일함을 보였다.

Abstract

This paper describes the efficient implementation of DNA sequence generate system with genetic algorithm for reducing computation time of NACST. The proposed processor is based on genetic algorithm with fitness functions which would suit the point of reference for generated sequences. In order to implement efficient hardware structure, we used the pipelined structure. In addition our design was applied the parallelism to achieve even better simulation time than the sequence generator system which is designed on software. In this paper, our hardware is implemented on the FPGA board with xc2v6000 devices. Through experiment, the proposed hardware achieves 467 times speed-up over software on a PC and sequence generate performance of hardware is same with software.

Keywords: Genetic Algorithm, Sequence generator

I. 서 론

최근 들어 DNA 분자가 가지고 있는 막대한 병렬성과 저장성 때문에 DNA 컴퓨팅 기술에 대한 연구가 활발해 지기 시작했다. DNA 컴퓨팅 기술은 DNA 분자 각각을 계산의 도구로 사용하는데, DNA 분자는 1g 정도의 액체에 대략 3×10^{22}개 존재한다. 막대한 양의 DNA 서열 각각은 정보 저장의 매체로 사용 가능하다. 또한 DNA 분자는 일련의 생화학 반응에 의해 단시간에 무수히 많은 수로 확장 가능하고 각각의 계산 과정을 이루는 하나의 실험 또한 무수히 많은 수의 DNA 서열에 동시 적용될 수 있다. 이러한 DNA 컴퓨팅의 저장성과 병렬성은 보통 컴퓨터로는 상상할 수 없는 정도이다. 그러나 DNA 컴퓨팅은 아직 시작 단계에 있기 때문에 해결되지 못한 문제점들이 몇 가지 있는데[1] 첫 번째 문

* 정회원, ** 종신회원, 인하대학교 정보통신공학과
(Department of Electronics Engineering, Graduate School, Inha University)
*** 정회원, 전자부품 연구원 DxB 통신융합 연구센터
(Korean Electronics Technology Institute)
접수일자: 2005년12월21일, 수정완료일: 2006년8월30일

제는 실제 실험을 하는 과정에서 계산 시간이 많이 소요되는 것이다. 따라서 DNA 분자 실험을 효율적으로 진행하고 비용과 시간의 단축을 극대화하기 위해 시뮬레이터(NACST)가 개발되었다[2]. 두 번째 문제는 생물학 실험들은 확률적이고 통계적으로 반응하므로 이러한 실험들을 연산자로 사용하면 DNA 컴퓨팅에서 해를 찾고도 찾지 못했다고 하거나, 해를 발견하지 못했는데도 발견했다고 하는 등의 치명적인 오류가 생길 수 있다. 이러한 오류의 원인으로 연산의 기본이 되는 서열들의 최적화 정도를 꼽을 수 있다. 염기서열의 최적화에 따른 오류는 진화 알고리즘을 이용해 DNA 염기배열을 최적화함으로써 그 가능성을 미리 제거할 수 있다[2]. 그러나 진화 알고리즘을 적용해 개발한 NACST는 반복적이고 수직적인 연산의 특징을 가지고 있어 시뮬레이션 시간이 오래 걸리는 단점이 있다. 소프트웨어로 구현된 NACST의 1회 시뮬레이션 시간은 실험 결과 약15시간 이상이었다. 비록 NACST가 실제 실험을 시뮬레이션으로 대체해 검증시간을 현저히 단축시키기는 했지만 여러 계산 모델을 빠르게 검증하고 그 결과를 토대로 실제 실험을 진행해야 하는 실험자의 입장에서는 더욱 단축된 시뮬레이션 시간이 요구된다. 따라서 NACST의 시뮬레이션 시간의 약 95% 이상을 차지하는 NACST의 코드 최적화 부분의 시뮬레이션 시간 단축을 통해 NACST 전체의 시뮬레이션 시간을 줄여야 한다. 이는 NACST의 DNA 서열 생성 시스템을 하드웨어로 대체함으로써 가능하다. 따라서 NACST의 서열 생성 시스템의 반복적이고 수직적인 연산 구조의 특성을 반영한 하드웨어 구현이 필수적이다.

본 논문은 서열생성 시스템의 빠른 연산을 위해 효율적인 하드웨어 구조를 제안하고 구현하였다. 하드웨어는 반복적인 계산을 병렬적, 연속적으로 수행할 수 있기 때문에 연산시간이 크게 단축될 것으로 기대된다.

실험을 위하여 FPGA device인 xc2v6000이 탑재된 prototype board를 사용하여 실험을 수행하였고, 본 구조의 성능을 평가하기 위해 NACST의 서열생성 시스템과 하드웨어로 구현된 서열 생성 시스템의 연산 시간을 비교하고, 구현된 서열 생성 시스템이 채택한 DNA 서열 생성 기준의 유효성을 검증하기 위하여 Deaton 등이 제시한[12] 'Good codes'와 'Bad codes'에 대해 생성된 DNA 서열들이 어떠한 값을 부여하는가를 관찰하였다.

다음 Ⅱ장에서는 DNA 컴퓨팅 및 NACST, DNA 서열생성 시스템의 개념 및 동작원리, 특성을 설명하고, DNA 서열생성 시스템에 적용된 유전자 알고리즘 및 적합도 함수에 대해 설명한다. Ⅲ장 에서는 제안된 DNA 서열 생성 시스템의 내부 중요 모듈의 구조와 병렬처리 및 pipelining 방식 등의 구조에 대해 설명한다. 이어서 Ⅳ장에서는 제안된 하드웨어의 성능검증을 위해 제작된 prototype과 하드웨어의 시뮬레이션 결과 및 성능비교 결과를 설명하고 마지막으로 Ⅴ장에서는 본 논문의 결론을 맺고자 한다.

Ⅱ. DNA 서열생성 시스템

1. DNA 컴퓨팅

DNA 컴퓨팅 기법은 실제 생체 분자인 DNA를 계산의 도구 및 정보 저장 도구로 사용한다. 즉, 2진수를 사용하는 컴퓨터와는 달리 가상의 DNA 컴퓨터는 A(Adenine), C(Cytosine), G(Guanine), T(Thymine) 4가지 염기로 정보를 표현한다. 그림 1은 DNA 서열의 염기(bases) A,C,G,T 및 DNA 구조를 보여준다[3].

이들 4가지 염기들은 각각 Phosphate, Sugar와 결합하여 Deoxyribonucleotide를 이루게 된다. Sugar의 한 쪽에는 Base가 결합하고, 반대쪽에는 Phosphate가 결합한다. 즉 하나의 Deoxyribonucleotide는 Phostphate, Sugar, Base(A,C,G,T)의 세 부분으로 구성되며 이렇게 만들어진 하나의 Deoxyribonucleotide를 DNA 서열의 최소 단위로 생각해도 무방하다. 하나의 Deoxyribonucleotide는 자신의 Phosphate(Sugar)를 이용하여 다른 Deoxyribonucleotide의 Sugar (Phosphate)와 결합할 수 있는데 여러 개의 Deoxyribonucleotide가 이러한

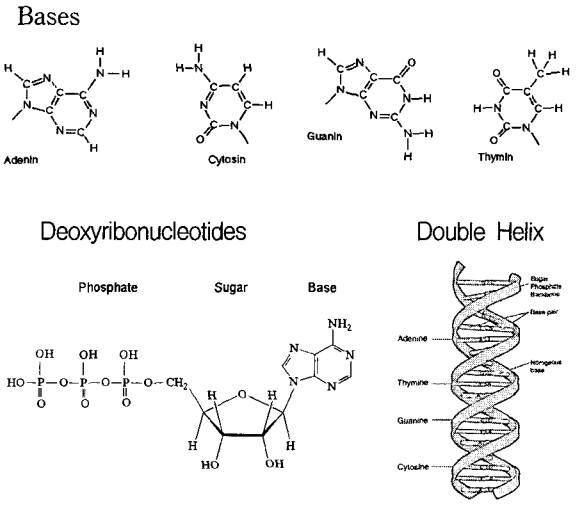

그림 1. DNA의 구조
Fig. 1. Structure of DNA

표 1. DNA 컴퓨팅 연산자와 실험
Table 1. DNA computing operators and experiments.

연산자	실험	대표적인 실험명	비고
길이측정	DNA 서열의 길이 측정	electrophoresis	병렬
찾기	특정 DAN 서열 검출	PCR 응용	병렬
분리하기	double strand 분리	가열, pH 높임	병렬
결합하기	DNA duplex 생성	hybridization	병렬
늘이기	DNA 서열에 특정 base 추가	transferization	병렬
줄이기	DNA 서열에서 특정 base 제거	효소 이용	병렬
자르기	DNA 서열의 특정 위치에서 절단	제한 효소 이용	병렬
연결하기	2개의 서열을 연결시킴	ligation	병렬
수정하기	DNA 서열의 특정 base를 교체	repair 효소 이용	순차
증폭하기	특정 DNA 서열의 수를 증가시킴	PCR	병렬
읽기	DNA 서열의 base를 밝힘	sequencing	순차

Phosphate-Sugar 결합을 통하여 길게 늘어선 것을 일반적으로 DNA 서열이라 부른다. 이 DNA 서열이 DNA 컴퓨팅에서의 피연산자(operand)에 해당하는 것이며 이것을 대상으로 여러 생물학 실험(연산자)을 통해 실제 '컴퓨팅'과정을 이루게 된다[2][4]. 표 1에 DNA 컴퓨팅의 연산자[5]와 해당 생물학 실험을 정리하였다.

2. NACST

NACST(Nucleic Acid Computing Simulation Toolkit)는 DNA 컴퓨팅의 새로운 알고리즘 및 계산 모델을 개발하거나 새로운 문제 표현 방법들을 검증하는데 소모되는 많은 비용과 시간을 단축하고 DNA 분자 실험의 효율을 극대화하기 위해 개발된 생물학 실험 시뮬레이터이다. DNA 컴퓨팅은 실제 생물학 실험 방법들을 연산자로 사용하기 때문에 일반 컴퓨터에 비해 많은 연산 비용과 시간을 필요하게 된다. 실제로 연산을 위한 고가의 실험 장비들이 모두 갖추어져 있다고 해도 한번 계산하는 과정에 필요한 소모성 재료비만 대략 100만원 가량 필요하다. 또한 계산시간을 살펴보면, Adleman이 HPP를 해결하는데 7일 이라는 시간이 걸렸다고 한다[6]. 또한, 연산자인 실험 방법들은 항상 오류의 가능성을 가지고 있기 때문에 실험 방법이 정확했다 하더라도 원하는 결과를 얻지 못할 수 있다[2]. 따라서 NACST를 통해 다양한 환경에서 계산 모델의 상태를 관찰하여 오류가 최소화 되는 모델을 개발하고 소프트웨어적으로 검증한 후 실제 실험을 통해 재검증을 하면 훨씬 효율적으로 DNA 컴퓨팅 알고리즘 및 계산 모델을 검증하고 새로운 문제 표현방법들을 개발할 수 있을 것이다. 그림 2에 NACST의 개념적인 도식이 나타나있다[2]. 그림에서도 알 수 있듯이 NACST는 코드 최적화

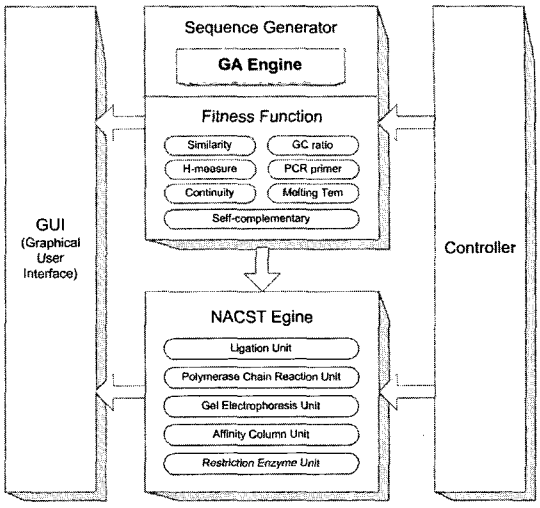

그림 2. NACST의 개념적인 형태
Fig. 2. Conceptual structure of NACST.

부분(DNA sequence generator)과 연산자 시뮬레이터(NACST engine) 그리고 사용자 환경(GUI)와 제어 부분(controller)으로 구성되어 있다. 코드 최적화 부분에서는 실험에 사용되는 코드인 DNA 염기배열을 유전자 알고리즘을 이용해 최적화 한다. 연산자 시뮬레이터는 각종 실험 방법들을 시뮬레이션하며, 사용자 환경 부분을 통해 사용자는 실험 환경 및 방법을 선택해 시뮬레이션을 한 후 그 결과를 확인하도록 되어 있다. 제어 부분은 사용자의 입력에 따라 실험 과정들을 제어한다.

3. 유전자 알고리즘을 이용한 DNA 서열 최적화

DNA 컴퓨팅은 주어진 문제를 DNA 염기배열로 변환하는 효율적인 표현 방법을 가지고 있지 않다. 현재까지의 연구들은 무작위 표현 방법(random encoding method)을 사용하고 있다. 무작위 표현 방법은 DNA 염기배열의 길이를 주어진 문제를 표현하기에 충분하고 오류의 가능성을 최대한 줄일 수 있는 길이로 정한 후 임의의 염기를 배열하여 문제를 표현한다. 그러나 이러한 방법은 표현할 수 있는 문제의 크기에 한계가 있고, 임의로 염기를 표현하는 과정에서 오류가 포함될 가능성이 높아 생물학 실험과정 중 잘못된 결과를 보일 확률이 높다[7]. 따라서 보다 효과적인 문제 표현 방법을 개발하기 위해 Deaton et al.7[2]등의 연구 결과를 바탕으로 한 진화 연산 방법이 NACST의 코드 최적화 모듈에 적용되었다.

가. 유전자 알고리즘

유전자 알고리즘은 생태계의 진화를 주어진 조건에

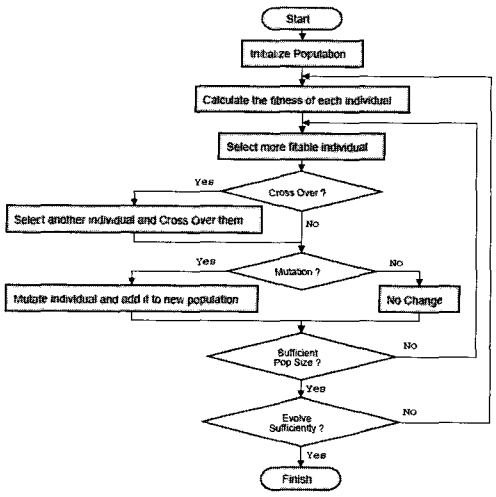

그림 3. 유전자 알고리즘
Fig. 3. Genetic algorithm.

대한 최적화 계산과정으로 보고 이를 컴퓨터상의 계산 모델로 환원한 알고리즘 이다[8]. 유전자 알고리즘은 선택과 교차, 돌연변이 등의 연산자를 이용하여 문제에 대한 후보해 또는 유기체로 구성되는 개체군을 새로운 개체군으로 반복 변형한다. 이러한 과정이 문제해결에 이용될 때, 기본적으로 요구되는 전제는 인위적인 진화 현상을 일으킬 초기집단을 구성하는 것이다. 집단은 문제 상에서 잠정적인 해를 뜻하는 다수의 개체로 형성되는데 이들은 유전자의 역할을 반영 하도록 흔히 비트스트링의 형태로 표현된다. 이 집단은 유성생식과 진화원리를 모방한 유전 연산자에 의해 점진적으로 개선되게 된다. 각 개체는 집단의 구성원으로 더 적합하고 덜 적합한 어떤 특징을 가지고 있기 때문에 적합한 정도가 더 큰 개체들이 더 많이 선택된다. 선택된 개체들 간에는 유전정보가 교환되고, 무작위적인 유전자 변경이 도입됨으로써 다음 세대의 집단을 형성하며 진화를 거듭하게 된다[9, 10, 11]. 그림 3에 유전자 알고리즘의 기본 흐름도를 나타내었다.

한 개체군(population)은 여러 개의 개체(individual)로 구성되며, 주어진 조건에 적합도가 높은 개체들을 선택(selection)하여 정보의 일부분을 상호 교환함으로써 교차(crossover)하여 다음 개체군을 형성한다. 진화 중에는 각 개체의 다양성(diversity)을 높이기 위해 개체가 가진 정보의 일부분을 임의로 변경하는 돌연변이(mutation)를 일으키기도 한다. 이런 방법의 진화가 거듭되면 후세대에 속한 개체들은 주어진 조건에 대해 매우 높은 적합도를 갖게 되며 우리는 이들 중 최적의 개체를 찾으리라 기대하는 것이다. 이러한 기본 틀을 바탕으로 개체의 인코딩(encoding)방법, 선택(selection)방법, 교차 연산(crossover) 혹은 돌연변이 연산(mutation)방법 등에 따라 다양한 형태의 알고리즘이 존재하며, 문제의 성격에 따라 성능을 극대화시키기 위해 여러 방법들이 활발하게 연구되고 있다.

나. NACST에서의 DNA 서열 최적화

오류 발생 가능성이 적은 DNA 서열을 생성하기 위해 이를 판별할 수 있는 기준들을 여러 종류의 적합도 함수를 통해 수치화 하게 되면, 이는 결국 적합도 함수들의 가중합 함수에 대한 최적화 문제로 바뀌게 된다. 따라서 최적화 문제에 많이 사용되는 유전자 알고리즘을 이용해 DNA 서열을 최적화 시킬 수 있게 된다.

DNA 컴퓨팅은 여러 개의 DNA 서열들이 한 묶음으로 실험에 사용되기 때문에 일반적으로 하나의 이진 문자열이 하나의 개체(chromosome)에 대응되는 단순 유전자 알고리즘(simple genetic algorithm)을 그대로 DNA 서열의 최적화에 적용하기는 어렵다. 따라서 자식 개체(sequence)의 개념을 추가하여 실제 DNA 서열 하나는 하나의 Sequence가 되고 여러 개의 DNA 서열이 묶여 하나의 Chromosome이 되도록 하였다. 또한 유전자 알고리즘에서는 Chromosome을 하나의 개체로 간주한다. 그림 4에 DNA 서열의 최적화를 위해 NACST에서 고안된 개체군의 구조를 나타내었다.

NACST는 위와 같은 서열 개체군의 구조를 기반으로 교차 및 돌연변이 연산자를 통해 반복적으로 population을 진화시킴으로써 DNA 서열을 최적화 시킨다. NACST는 여러 교차방법 중 일점 교차(one point crossover) 방법을 사용하며 0.05 이하의 돌연변이 확률 (P_m)을 갖고 스트링에 있는 각각의 비트에 대해 독립적으로 확률을 적용하여 변이를 일으키는 돌연변이 법을 사용한다. 또한 NACST는 DNA 서열의 우수성을 측정하기 위해 여러 적합도 함수를 사용 하는데 이런 적합도 함수들은 몇 가지 특징을 갖는다. 첫 번째 특징은 한 chromosome에 적용되는 적합도 함수가 단일한

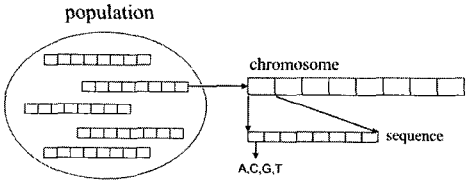

그림 4. DNA 서열 개체군 구조
Fig. 4. DNA sequence population structure.

형태가 아닌 서로 다른 여러 적합도 함수의 가중합 형태라는 점이다. 이런 형태의 적합도 함수에서 나타나는 가장 큰 문제점 중 하나는 적합도 함수들 간에 강한 의존성이 존재하여 최적화에 어려움을 겪게 되는 것인데, 예를 들어 함수 f와 g에 의존성이 존재하여 f를 최소화하면 g의 값이 커지고, g를 최소화하면 f값이 커지는 등의 제약 조건이 발생하는 것이다. NACST의 적합도 함수들은 이런 현상을 피하도록 정의되어 있다[4]. 또 다른 문제점은 함수값들의 변위가 고정되어 있지 않음으로써 특별히 지배적인(dominant) 최적화 조건이 형성되는 경우인데, 예를 들어 함수 f값의 범위가 매우 크고 다른 함수 g, h의 범위가 상대적으로 작은 값을 갖는 다면, 진화 과정 중에 생성되는 개체들은 함수 f만을 최적화하는 경향을 갖게 되는 것이다. NACST는 함수값을 적절히 정규화 하는 방법을 통해 이 문제를 적절한 수준으로 해결하였다[4]. 두 번째 특징은 각 적합도 함수의 적용 대상이 모두 같지는 않다는 점이다. 예를 들면, GC Ratio를 측정 하는 함수는 하나의 Sequence에 적용되는 것인 반면, Similarity에 대한 함수는 한 Chromosome에 속한 두 개의 Sequence를 그 대상으로 한다는 것 등이다. Chromosome을 길게 늘어선 하나의 이진 문자열로 생각해 보면, 하나의 문자열을 여러 개의 단편(segment)들로 쪼개었을 때, 각각의 단편들의 특성만을 수치화하는 함수와 단편들 간의 관계를 수치화하는 함수들이 동시에 적용되는 형식이다. 이런 조건을 고려하여, 한 Chromosome에 속한 모든 Sequence에 대한 적합도 함수값들을 전부 합하여 Chromosome의 적합도로 부여하며, 유전자 알고리즘의 선택 과정은 이 값에 근거하여 이루어지도록 하였다[4]. 다음 절에서 이러한 적합도 함수들의 정의와 그 계산 방법 및 특징들을 자세히 살펴보도록 한다.

4. DNA 서열 생성 기준

DNA 컴퓨팅의 성공 여부는 문제 해결에 사용된 DNA 서열의 종류와 성질에 크게 의존하므로 DNA의 화학적 특성을 수치화 하여 실험에서의 적합성을 고려한 DNA 서열을 생성하는 일이 중요하다. 표 2에서 NACST에서 사용하는 DNA 적합도 함수[4]들을 정리하였다.

표 2. DNA 적합도 함수
Table 2. DNA fitness function.

함수명	비고
Similarity	대상 서열이 다른 서열과 비슷한 정도를 측정
H-measure	다른 서열들과의 상보성 정도를 측정
3'-end Hybridization	서열의 3'-end 부분에서 H-measure 측정
GC Ratio	서열에서 G, C염기가 차지하는 비율 측정
Continuity	특정 염기가 연속으로 나타는 정도 측정

가. Similarity

Similarity는 한 서열이 다른 서열과 다른 고유한 정도를 확보하기 위한 적합도 함수로서 「찾기」, 「결합하기」, 「늘이기」, 「연결하기」 등의 연산자를 안정화 시키는 역할을 한다[4]. Similarity는 임의의 두 개의 서열이 동일한 위치에 같은 종류의 염기를 얼마나 많이 가지고 있는지를 측정한다. NACST는 부가적으로 보다 정확한 고유성을 측정하기 위해 두 서열을 같은 위치에 놓고 같은 위치에서 두 서열 간 일치하는 염기의 수를 계산한 후 한 서열을 한 염기만큼 우측으로 이동시키며 계속적으로 서열의 유사정도를 비교한다. 이때 비교된 길이의 17%를 임계값(threshold)로 하여 이 이상의 것만을 함수값으로 취한다.

그림 5에서 위쪽의 Similarity 값은 2이고, 아래쪽의 경우는 4이다. 그러나 위쪽 그림의 비교된 길이가 15이므로 임계값은 3이 된다. 따라서 위쪽의 경우의 Similarity값은 함수값에 포함되지 않는다. 반면 아래쪽 그림의 경우 비교된 길이가 13으로 임계값이 3이되 Similarity값 4가 적합도 함수값에 포함된다.

두 개의 서열의 Similarity값을 측정하는 함수 SIM은 :

$$SIM(S) = \sum_{i<j} sim(s_i, s_j) \quad \text{Chromosome의 Similarity 값} \quad (1)$$

$$SIM(s_i) = \sum_{1\le j\le n} sim(s_i, s_j) \quad \text{Sequence의 Similarity 값} \quad (2)$$

이때 자기 자신과의 Similarity 비교에서 우측으로의 이동이 없이 비교하는 경우는 함수값을 불필요하게 크게 만들기 때문에 계산에서 제외한다.

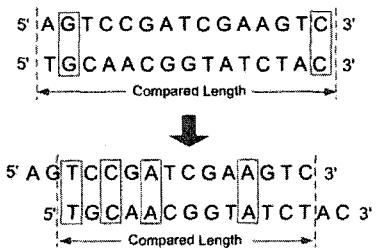

그림 5. Similarity
Fig. 5. Similarity.

나. H-measure

Similarity는 방향이 동일한 서열간의 고유성을 측정하는 기준인 반면 H-measure는 비교하는 서열 간 염기 방향이 상보적이어서 DNA가 duplex를 형성하는 경우 내가 원하는 염기 서열들 간에 상보결합(A-T, G-C)을 형성하는 지를 판단하는 기준이다. H-measure 또한 Similarity와 같이 한 서열이 다른 서열과 다른 고유한 정도를 확보하기 위한 적합도 함수로서 「찾기」, 「결합하기」, 「늘이기」, 「연결하기」 등의 연산자를 안정화 시키는 역할[1]을 하며, 실험과정에서 발생하는 mismatch hybridization 과 shifted hybridization을 사전에 예방하는 효과를 가져다준다. H-measure는 비교하는 두 서열이 서로 상보 서열이라는 가정 하에서 각 염기가 서로 상보 결합을 형성하는 횟수를 계산한다. 이 또한 Similarity와 마찬가지로 한 염기씩 이동하며 계산하고 비교되는 길이의 17%이상의 상보결합 횟수만을 함수값에 추가 시킨다.

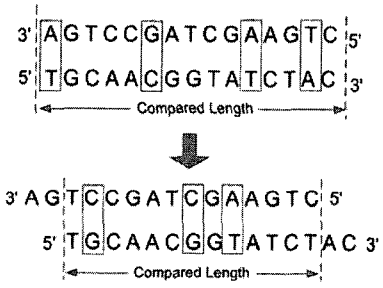

그림 6. H-measure
Fig. 6. H-measure.

그림 6에 H-measure의 계산 과정이 나타나 있다. 아래쪽의 그림은 위쪽의 그림에 비해 2염기 만큼 우측으로 이동시켜 비교한 것이다. 여기에서 좌측 그림의 H-measure 값은 4 이고 임계값 3을 넘어 함수값에 포함된다. 또한 우측 그림의 H-measure 값은 3으로서 임계값 3을 넘어 역시 함수값에 포함된다.

두 서열의 H-measure 값을 측정하는 함수 H는 :

$$H(S)=\sum_{i\le j} h(s_i,s_j) \quad \text{Chromosome의 H-measure 값} \qquad (3)$$

$$H(s_i)=\sum_{1\le j\le n} h(s_i,s_j) \quad \text{Sequence의 H-measure 값} \qquad (4)$$

다. GC-ratio

DNA의 상보결합은 A-T, G-C 염기간의 결합으로 이루어진다. 이중 G-C결합은 A-T결합에 비해 결합강도가 강해 전체 서열의 결합의 안정성은 서열 안에 존재하는 G, C 염기의 비율에 크게 의존하게 된다. 결합의 강도는 구조의 안정성으로 이어지며 이는 컴퓨팅의 안정성을 결정하는 요인이 된다. GC-ratio는 서열 안에 존재하는 전체 염기의 개수에 대한 G, C염기의 개수의 비를 계산한다.

서열의 GC-ratio 값을 측정하는 함수 G는 :

$$G(x)=\frac{G,C\text{염기의 개수}}{\text{전체 염기의 개수}}\times 100 \qquad (5)$$

라. Continuity

같은 종류의 염기가 연속적으로 나타나는 서열은 실제 실험에서 의도하지 않은 결합을 형성하는 경우가 많으며, 그 반응을 제어하기 또한 쉽지 않다. 따라서 실제 실험에서 생길 수 있는 오류의 가능성을 줄이기 위해 서열 내 동일한 염기 2번의 연속은 허용하고 그 이상의 연속은 연속된 횟수를 제곱하여 함수값에 포함 시킨다. 이러한 방법을 통해 같은 염기가 여러 번 연속되는 서열일수록 큰 함수값이 적용되며 이러한 서열은 유전자 알고리즘의 진화 과정에서 자연스럽게 도태된다.

그림 7에 Continuity 계산 예가 나와 있다. 이 서열의 경우 Continuity 값은 $4\times4+6\times6=52$가 된다.

NACST가 유전자 알고리즘을 통해 DNA 서열을 최적화 하는 과정에서 많은 시뮬레이션 시간을 할애 하는 부분이 바로 적합도 함수 부분이다. 유전자 알고리즘의 특성상 매 세대를 거듭할 때마다 적합도 평가 또한 반복되기 때문에 대량의 DNA 정보를 반복적으로 처리하는 적합도 함수 모듈에 많은 양의 연산이 몰리게 된다. 따라서 NACST의 시뮬레이션 시간 단축을 위해서는 적합도함수 모듈의 효율적인 하드웨어의 구현이 필수적이다. 하드웨어는 반복적인 계산을 병렬적으로 수행할 수 있을 뿐만 아니라 모듈 내부의 독립적인 부분들 또한 병렬적으로 처리 할 수 있어 연산시간 단축에 크게 기여 할 것으로 기대한다. 따라서 본 논문은 NACST의 시뮬레이션 시간 단축을 위해 최적화된 적합도함수 모듈을 포함하는 서열생성 시스템의 하드웨어 구조를 제안하고 구현한다.

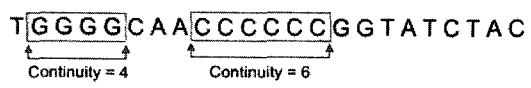

그림 7. Continuity
Fig. 7. Continuity.

Ⅲ. DNA 서열생성 시스템 구현

1. 제안된 서열생성 시스템의 구조

제안된 NACST 서열생성 시스템은 진화 연산 알고리즘을 수행하기 위한 교배(crossover)모듈, 돌연변이(mutation)모듈, 선택(parent selection)모듈, 적합도 함수(fitness function)모듈, 난수 발생기(random number generator), 세대 진화(evaluation, update) 모듈 등으로 구성되어 있다.

연산이 시작되면 Population initialize 모듈이 random number generator로부터 난수를 입력받아 240x5000 메모리에 chromosome을 채워 초기 population을 형성한다. 이러한 초기화 과정이 끝나면 연산의 시작을 알리는 start 신호가 parent select 블록에 전달되고 이 신호를 기준으로 서열 최적화가 시작된다. start 신호가 '0'이 되면 parent select 모듈은 난수 발생기로부터 임의의 값을 받아 부모들의 주소를 계산하는데 이때 chromosome의 크기는 1200bit이고 메모리 크기는 240bit이기 때문에 메모리 안의 240bit 데이터 5개를 하나의 데이터로 인식할 수 있도록 주소 값은 5의 배수가 되어야 한다. 이렇게 주어진 주소를 통해 부모를 읽어와 교차, 돌연변이를 거쳐 새로운 자손을 만들어 내고 새로 생성된 자손은 적합도 평가를 거쳐 새로운 세대로 전달한다. 이렇게 세대가 거듭될 때마다 generation loop controller는 그 횟수를 세고 정해진 세대수가 되면 시스템 전체 연산을 중단 시키고 생성된 서열들을 데이터 전송 컨트롤러를 통해 PC로 전달한다. 본 구조의 parent buffer 및 address buffer, child buffer는 우수한 개체들을 다음 세대로 전달하기 위한 모듈들로서 부모 및 자손의 적합도 평가 후 address buffer에 저장되어 있는 부모 주소로 살아남은 개체들을 전달한다.

서열생성 시스템에서 연산의 기본단위는 보통 Sequence를 의미한다. 따라서 서열생성 시스템의 적합도 평가는 기본단위 인 Sequence를 기준으로 하게 된다. 그러나 유전자 알고리즘의 연산 기본단위는 chromosome이다. 이때 하나의 개체(chromosome)는 약 20개의 Sequence로 이루어지며, 하나의 Sequence는 30개의 염기로 이루어진다. 염기는 A, C, G, T 총 4가지 종류가 있는데 이들을 binary로 표현하기 위해서는 각각에 2bit가 할당 되어야 한다. DNA염기들을 적절히 잘 표현하면 적합도 함수 계산 시 XOR 와 NAND 게이트 혹은 XOR와 NOR게이트만을 가지고 같은 염기 간 비교 및 상보염기 간 비교 결과를 간단히 얻을 수 있다. 표 3에 적합도 함수의 계산을 용이하게 하는 염기의 binary표현법을 나타내었다. 또한 이러한 표현법을 통한 같은 염기 간의 비교 및 상보 염기 간 비교결과를 그림 9에 나타내었다. 그림9를 통해 아래와 같은 염기 표현법을 이용하면 그 결과를 '0'혹은 '1'로 간단히 얻을 수 있음을 알 수 있다.

따라서 하나의 개체는 2bit의 30개 염기로 표현되는 Sequence 20개로 이루어져 1200bit의 데이터 크기를 가진다.

제안된 본 구조에서는 1200bit라는 대량의 데이터를 빠르게 처리하기 위해 pipeline 및 parallel 기법을 적용하였다. 또한 연산이 종료된 후 생성된 최적의 서열들을 실제 실험 연산자들이 구현되어 있는 PC로 전달하여 DNA 분자컴퓨팅 시뮬레이션을 진행하기 위해 PC와 보드 간 interface를 구현하였다. 이를 위해 1Mbyte

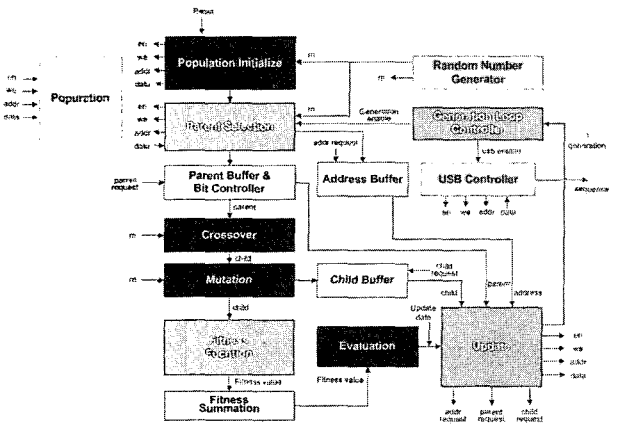

그림 8. 서열생성 시스템 전체 구조
Fig. 8. Sequence generator system architecture.

표 3. DNA 염기의 binary 표현법
Table 3. Binary expression of DNA bases.

DNA 염기	binary 표현법
Adenine(A)	00
Cytosine(C)	01
Guanine(G)	10
Thymine(T)	11

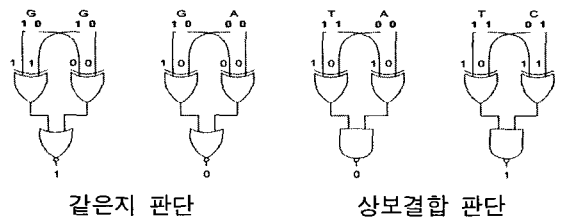

그림 9. 두 염기의 일치여부와 상보결합 여부 판단
Fig. 9. Sequence generator system architecture.

의 전송 능력을 갖는 usb가 사용되었고 usb가 8bit 단위로 데이터를 전송하기 때문에 생성된 서열들을 8bit 단위로 잘라 PC로 전송하고 동기를 맞춰줄 데이터 전송 컨트롤러(usb controller) 모듈을 구현 하였다. 또한 보드 위에 구현되어 동작하기 때문에 보드에 발생하는 열로 인한 기계적 오작동의 염려가 있어 gated clocking의 저전력 설계기법이 적용되었다.

구현된 교차, 돌연변이, 선택, 적합도 함수 모듈 및 pipeline, parallel 기법들은 다음 장에 자세히 설명하도록 한다. 그림 8에 보이는 bit controller는 pipeline 및 parallel 기법을 위해 메모리로부터 읽어온 데이터를 쪼개고 합치는 작업을 하는 모듈로서 이 또한 다음 장에 설명한다.

2. Pipelining 과 Parallelism 설계

가. Parallelism

NACST 서열생성 시스템의 연산 시간을 줄이기 위해 제안된 병렬처리 구조가 그림 10에 나타나 있다.

서열생성 시스템에서 사용하는 chromosome의 크기는 1200bit인데 이 데이터는 240bit의 데이터 5개로 나누어져 메모리에 저장되어 있다. 1200bit의 데이터를 모두 읽어와 한꺼번에 교차 및 돌연변이 적합도 함수 등의 연산을 하려면 데이터를 읽어 오는 데만 최소 5번의 clock이 필요하고 읽혀진 1200bit 데이터를 임시로 저장할 수 있는 buffer가 필요하며 sequence단위로 계산되는 적합도 함수를 위해 즉, 60bit의 데이터를 한번 계산하는데 1140bit의 데이터가 기다려야 한다. 혹은 1200bit 데이터의 적합도 값을 동시에 구하기 위해 1200bit를 동시에 처리할 수 있는 적합도 함수를 구현하여야 한다. 이는 FPGA의 제한된 리소스 내에서는 구현하기 힘들 정도의 큰 모듈이 될 것이다. 따라서 제안된 구조에서는 한번에 240bit씩 읽어와 1200bit의 데이터가 모두 도착할 때까지 기다린 후 연산을 진행하는 대신 NACST 연산의 기본단위인 Sequence 60bit 데이터를 기준으로 병렬처리 한다. 따라서 본 구조는 240bit 데이터를 읽어 옴과 동시에 60bit 데이터 4개로 나누어 동시에 처리하여 연산시간 및 사용하는 하드웨어 리소스를 줄였다. 그림에서는 60bit 단위의 프로세서 20개가 병렬처리 되는 것처럼 보이지만 이는 병렬처리의 개념을 설명하기 위한 그림이고 실제는 60bit 단위의 프로세서 4개가 병렬 처리된다.

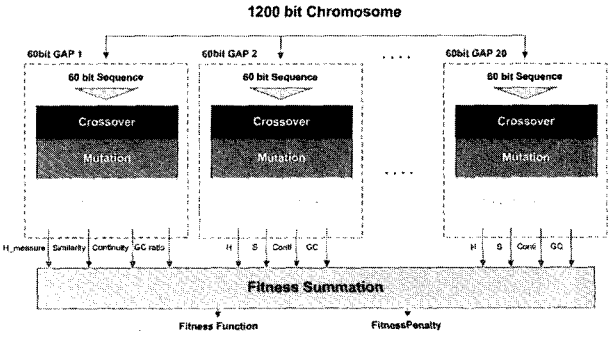

그림 10. Parallelism method
Fig. 10. Parallelism method.

유전자 알고리즘은 chromosome을 하나의 단위로 삼고 교차, 돌연변이, 적합도 평가 등을 거쳐 새로운 세대를 구성한 뒤 새롭게 구성된 세대에서 다시 개체들을 골라 연산을 진행하는 것을 기본으로 한다. 따라서 작은 단위의 데이터로 나누어 계산하는 병렬처리를 통해 이런 유전자 알고리즘의 기본 연산 틀을 벗어나서는 안된다. 따라서 유전자 알고리즘의 기본 연산 방법의 틀을 벗어나지 않는 병렬처리 방법을 제안한다. 240bit의 데이터는 메모리로부터 읽혀지면 그림 8의 bit controller에 의해 60bit 데이터 4개로 나누어져 각각 60bit 단위로 유전자 알고리즘 연산을 하는 프로세서로 할당되게 된다. 이때, 교차는 미리 교차 포인트를 계산해 두었다가 입력되는 60비트의 sequence들을 적절히 위치시키고 돌연변이 확률 값도 미리 계산해 놓은 후 해당사항이 없는 sequence는 통과 시키고 돌연변이가 필요한 sequence는 돌연변이를 일으킨 후 적합도 함수 모듈로 전달하게 된다. 적합도 평가는 sequence 단위로 연산하며 연산된 결과는 fitness summation 블록에서 모두 더해져 최종 1개의 chromosome에 대한 적합도 값을 만들어 낸다. 각 부분들의 자세한 연산 방법은 다음 장에서 설명한다. 이와 같이 제안된 병렬 프로세서는 handshaking protocol 기반의 파이프라인 적용을 통해 프로세서 내부의 연산을 보다 효율적으로 수행하고 연산의 병목 현상이 없도록 설계되었다.

나. Pipelining

제안된 병렬처리 구조에서 60bit 프로세서의 각 모듈은 알고리즘 수행 상태에 따라 하나 혹은 두 개가 동시에 독립적으로 실행된다. 그림 11은 프로세서 내부의 파이프라인 동작을 나타내는 그림으로써 1개의 병렬 그룹과 2개의 단독 모듈이 세 단의 파이프라인 단으로 이루어져 있다.

각 병렬 그룹들은 지연시간이 모두 다르기 때문에 연속적인 sequence의 연산을 위해 handshaking protocol

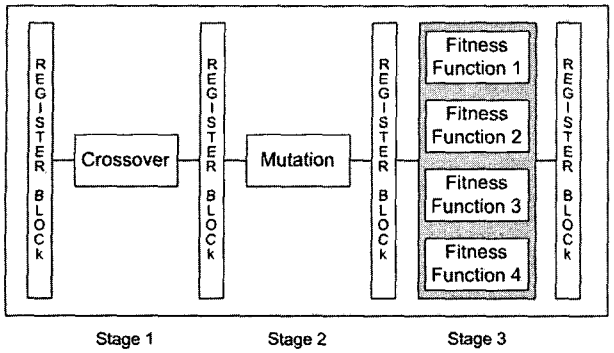

그림 11. 3단 파이프라인 개념도
Fig. 11. 3 state pipeline method.

기반으로 구성되어진 파이프라인을 사용하였다. 각 모듈들은 미리 계산 되어진 교차 지점과 돌연변이 발생 가능성에 따라 연산시간의 차이를 보이게 되는 것이다. 병렬 그룹으로 구현된 적합도 함수 모듈은 clock에 의해 독립적으로 연산이 가능하도록 설계 하였으며 적합도 함수 간 구조적으로 공유가 가능한 부분들이 공유되어 한번 연산으로 두 적합도 함수의 계산에 기여할 수 있게 설계하여 효율적이고 빠른 연산 처리가 가능하도록 하였다.

3. 주요 모듈들의 구조 및 특징

가. Selection

선택된 개체들 중 다음 세대로 진화시킬 대상을 임의로 선택하는 부분으로써 Roulette Wheel 방법 과 elitism 방법을 기반으로 설계되었다. 적합도가 높은 개체들 순으로 선택될 확률을 높게 주고 가장 적합도가 우수한 개체는 무조건 선택하고 그 나머지 개체들 중 난수 발생기로부터 받은 값에 의해 다음 세대로 진화시킬 개체 하나를 추가로 선택한다. 만약 적합도 값이 같아 선택된 개체가 둘이면 하나의 개체를 난수 값에 의해 임의로 선택한다. 적합도가 가장 우수한 개체를 우선 선택함으로써 선택 압력을 높여 가능한 더 빠르게 최적의 해에 수렴하도록 설계하였다.

나. Crossover

제안된 교차(crossover) 모듈은 단순교차(one-point crossover)로 구현되었다. 교차 모듈에 포함된 교차 지점 발생기(crossover point generator)는 parent selection 모듈이 난수 발생기로부터 난수를 넘겨받아 부모가 될 데이터의 주소 값을 계산하는 동안 이 데이터들에 교차가 일어날 지점을 미리 계산한다. 이렇게 미리 계산된 교차 지점은 1200bit의 chromosome의 교

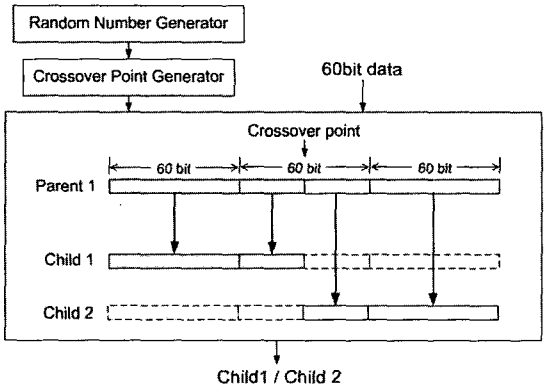

그림 12. 교차
Fig. 12. Crossover.

차 연산이 병렬로 수행되는 도중 변하지 않도록 카운터를 사용해 1200bit 데이터의 연산이 모두 끝날 때 까지 같은 값을 유지하도록 한다. 교차 모듈은 이렇게 미리 계산된 교차 지점을 기준으로 입력되는 60bit 데이터의 단순 전달 및 데이터 교환 유무를 판단하게 된다. 그림 12에 이러한 연산 방법이 나타나 있다.

그림에서 보이듯이 입력되는 60bit 데이터 안에 교차 지점이 포함되어 있지 않다면 새로 생성될 자손으로 바로 위치시키고 교차 지점이 포함된 경우에는 교차 지점에 따라 데이터를 나누어 자손에 위치시킨다. 병렬로 놓인 4개의 60bit 교차 모듈은 데이터가 위치될 자손의 주소 값을 정확히 알고 있기 때문에 교차 지점과 가지고 있는 주소 값을 비교하여 입력된 데이터가 자손1로 위치되어야하는지 자손2로 위치되어야 하는지 혹은 입력된 데이터가 교차에 포함 되 데이터를 나누어 위치시켜야 하는지를 쉽게 판단할 수 있다. 이러한 연산의 방법을 통해 1200bit의 데이터를 나누어 병렬 처리 하지만 1200bit 데이터 전체에 단순 교차 방법을 적용한 것과 같은 결과를 얻을 수 있다.

다. Mutation

돌연변이(mutation)는 다중 돌연변이(multi-point mutation)가 발생하도록 설계하였다. NACST 서열생성 시스템에 적용된 유전자 알고리즘에서는 돌연변이를 Sequence 단위로 일으킨다. Sequence단위 즉, 60bit단위로 그 Sequence에 돌연변이를 발생 시킬지 아닐지의 여부를 돌연변이 비율(mutation rate)에 의해 결정한 후 sequence 내부의 모든 비트에 대해 돌연변이 비율(mutation rate)을 적용하여 돌연변이가 발생할 지점을 결정 한다. 이와 같은 연산의 방법이 그림 13에 나타나 있다.

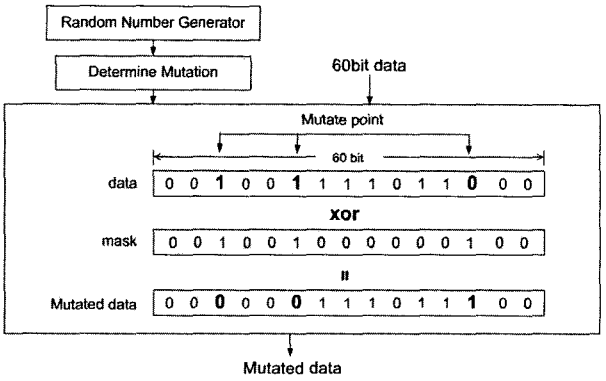

그림 13. 돌연변이
Fig. 13. Mutation.

Determine Mutation 블록에서 입력되는 Sequence에 대한 돌연변이 발생 여부를 결정하고, 입력된 데이터에 대해 돌연변이를 발생시키지 않는다면 데이터를 그대로 출력 단으로 연결시킨다. 반면 돌연변이를 발생시켜야 하는 경우 난수 발생기로부터 난수 값을 받아 데이터 내부의 모든 비트에 대해 돌연변이 비율을 적용하여 돌연변이 발생 지점을 선정한다. 이런 방법으로 그림 13와 같이 돌연변이 지점이 선택되면 모든 비트가 '0'으로 세팅되어 있는 mask에서 선택된 지점과 같은 자리의 비트들을 '1'로 바꾸어 돌연변이 발생을 위한 mask를 생성한다. 이렇게 생성된 mask와 원 데이터를 XOR 하여 최종 데이터를 생성해 낸다.

라. Random Number Generator

난수 발생기는 개체군의 배열로부터 개체의 선택, 교차 지점 선택, 돌연변이의 발생 여부 판단, 돌연변이의 발생위치 결정 등을 위해 사용되었다. 본 논문에 사용된 난수 발생기는 일반적으로 사용되지만 난수의 평균이 피연산자의 반이 되지 않아서 정확한 계산을 기대하기 어려운 LFSR(linear feedback shift register)방법의 단점을 보완한 cellular automata(CA)방법[13]을 이용하여 설계 되었다. 주기 및 초기 값의 문제를 개선하기 위해 4bit counter를 이용하여 CA의 경계조건을 변화시켜 초기 값에 관계없이 난수를 발생하도록 설계되었으며 maximum length cycle을 증가시켰다. 난수 발생기에서 사용한 CA는 시뮬레이션의 성능에서 rule 150(식(7))과 거의 비슷한 rule 90(식(6))에 의해 설계되었다. 또한 각기 다른 두 개의 seed 값을 사용하여 개체군을 생성함으로써 좀 더 다양한 개체군을 생성하도록 하였다.

$$a_i(t+1) = a_{i-1}(t) \oplus a_{i+1}(t) \quad (6)$$

$$a_i(t+1) = a_{i-1}(t) \oplus a_i(t) \oplus a_{i+1}(t) \quad (7)$$

마. Fitness Function

서열생성 모듈 중 적합도함수는 유전 알고리즘 내에서 개체들 간 우열을 판단하는 기준이 된다. 유전 알고리즘은 여러 세대를 반복해 가면서 최적의 해를 찾는데, 한 세대를 거치는 동안 적합도함수는 보통 3~4회 사용된다. 그러나 최적해를 찾는 동안 세대는 약 10,000번 이상 반복 생성 되므로 유전 알고리즘이 연산되는 전체시간 동안 적합도함수 모듈은 약 3~40,000번 이상 실행된다. 서열생성 모듈에서 사용하는 적합도 함수는 특히 그 연산양이 많고 연산 시간도 길어 대량의 데이터를 반복적으로 연산하는데 효과적인 적합도함수 모듈의 하드웨어 구조가 필수적이다.

제안된 적합도 함수 구조는 함수 간 구조적 공유가 가능한 부분들을 공유하여 한번 연산으로 두 적합도 함수의 계산에 기여할 수 있게 설계하였다. H-measure와 Similarity 함수의 경우 두 함수의 측정 목적은 서로 다르지만 두 염기를 비교하는 xor 부분은 공유될 수 있기 때문에 한 모듈 안에 설계되었다. 그림 14에 두 염기를 비교하는 부분이 공유된 Sequence 1개에 대한 H-measure와 Similarity의 연산이 나타나있다.

비교 블록으로 입력된 두 sequence는 sequence를 이루는 염기들 간 상보결합 정도와 일치 정도에 따라 30bit binary를 생성한다. 30bit의 binary는 H-measure의 경우 '0'이 존재하면 그 위치에 해당하는 염기들 간에 상보결합이 가능하다는 의미이고, '1'의 값은 그 위치에 해당하는 염기들 간에 상보결합이 없다는 의미이다. 또한 Similarity의 경우 '1'이 존재하면 그 위치의 염기들이 서로 같다는 의미이고, '0'인 경우는 그렇지 않다는 의미이다. 따라서 H-measure는 '0'을 Similarity는 '1'의 개수를 세어 이 값이 각각의 문턱값을 넘으면 적합도 값에 포함시키게 된다. Continuity와 GC-ratio의 경우 하나의 Sequence에 대해 연산을 수행한다. Continuity는 같은 염기의 연속성을 확인해야 하기 때문에 그림 15에 보이는 비교 블록의 두 xor 입력으로 이웃한 두 염기가 들어가고, GC-ratio는 G, C와 일치하는지 여부를 확인해야 하기 때문에 두 xor의 좌측 입력으로는 염기가 들어가고 나머지 우측 입력으로는 G(01)와 C(10)가 교대로 들어간다.

Continuity와 GC-ratio는 하나의 Sequence에 대해 연산을 수행하므로 모든 sequence 조합에 대한 연산을

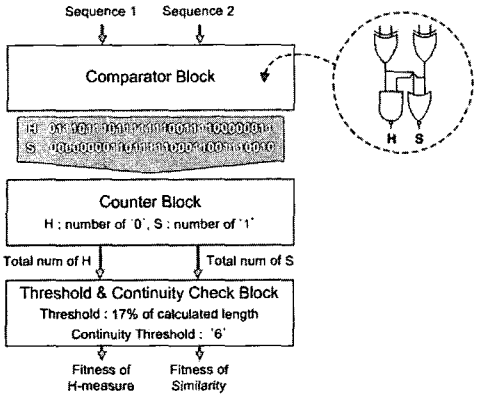

그림 14. H-measure 와 Similarity 구성도
Fig. 14. H-measure & Similarity method.

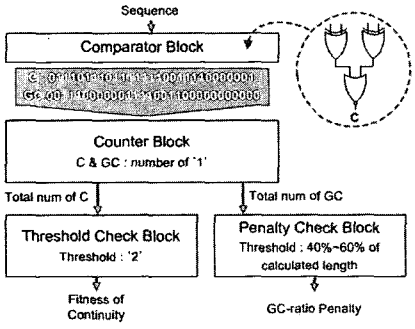

그림 15. Continuity 와 GC-ratio 구성도
Fig. 15. Continuity & GC-ratio method.

해야 하는 H-measure 와 Similarity와는 달리 20개의 Sequence에 대한 연산만을 해주면 된다. 그림 16에 Continuity 와 GC-ratio의 연산이 나타나 있다.

Continuity와 GC-ratio의 연산은 4개의 Sequence를 차례대로 수행하여도 연산이 복잡한 H-measure와 Similarity가 병렬로 처리되는 시간보다 적게 걸리므로 Threshold 와 Penalty를 확인하는 블록을 제외한 나머지 블록은 공유시키고, 입력되는 Sequence들을 차례대로 연산하도록 구현하였다. 이런 설계를 통해 불필요한 하드웨어의 면적을 줄일 수 있다. Continuity 및 GC-ratio 블록은 입력된 4 Sequence의 적합도 및 패널티 값을 차례로 연산한 후 Summation 블록의 문턱값과 패널티 기준 값을 바탕으로 최종 값을 산출한다.

H-measure & Similarity 모듈로 120bit씩 즉, 두 Sequence씩 입력된다. H-measure & Similarity 모듈 안에는 두 Sequence를 비교하는 블록이 두 개가 들어가 있다. 이는 입력된 두 Sequence의 모든 조합이 연산되어야 하는 H-measure 와 Similarity의 특성을 고려해 병렬로 연산을 처리 할 수 있게 설계하여 단 시간 안에 반복적인 연산을 대량으로 처리 할 수 있도록 하였다.

Control블록을 두어 Sequence가 가질 수 있는 조합

표 4. H-measure 와 Similarity 연산을 위한 Sequence 조합 표
Table 4. Sequence combination table for H-measure and Similarity operation.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1/1	1/2	1/3	1/4	1/5	1/6	1/7	1/8	1/9	1/10	1/11	1/12	1/13	1/14	1/15	1/16	1/17	1/18	1/19	1/20
2	2/1	2/2	2/3	2/4	2/5	2/6	2/7	2/8	2/9	2/10	2/11	2/12	2/13	2/14	2/15	2/16	2/17	2/18	2/19	2/20
3	3/1	3/2	3/3	3/4	3/5	3/6	3/7	3/8	3/9	3/10	3/11	3/12	3/13	3/14	3/15	3/16	3/17	3/18	3/19	3/20
4	4/1	4/2	4/3	4/4	4/5	4/6	4/7	4/8	4/9	4/10	4/11	4/12	4/13	4/14	4/15	4/16	4/17	4/18	4/19	4/20
5	5/1	5/2	5/3	5/4	5/5	5/6	5/7	5/8	5/9	5/10	5/11	5/12	5/13	5/14	5/15	5/16	5/17	5/18	5/19	5/20
6	6/1	6/2	6/3	6/4	6/5	6/6	6/7	6/8	6/9	6/10	6/11	6/12	6/13	6/14	6/15	6/16	6/17	6/18	6/19	6/20
7	7/1	7/2	7/3	7/4	7/5	7/6	7/7	7/8	7/9	7/10	7/11	7/12	7/13	7/14	7/15	7/16	7/17	7/18	7/19	7/20
8	8/1	8/2	8/3	8/4	8/5	8/6	8/7	8/8	8/9	8/10	8/11	8/12	8/13	8/14	8/15	8/16	8/17	8/18	8/19	8/20
9	9/1	9/2	9/3	9/4	9/5	9/6	9/7	9/8	9/9	9/10	9/11	9/12	9/13	9/14	9/15	9/16	9/17	9/18	9/19	9/20
10	10/1	10/2	10/3	10/4	10/5	10/6	10/7	10/8	10/9	10/10	10/11	10/12	10/13	10/14	10/15	10/16	10/17	10/18	10/19	10/20
11	11/1	11/2	11/3	11/4	11/5	11/6	11/7	11/8	11/9	11/10	11/11	11/12	11/13	11/14	11/15	11/16	11/17	11/18	11/19	11/20
12	12/1	12/2	12/3	12/4	12/5	12/6	12/7	12/8	12/9	12/10	12/11	12/12	12/13	12/14	12/15	12/16	12/17	12/18	12/19	12/20
13	13/1									13/10	13/11	13/12	13/13	13/14	13/15	13/16	13/17	13/18	13/19	13/20
14	14/1									14/10	14/11	14/12	14/13	14/14	14/15	14/16	14/17	14/18	14/19	14/20
15	15/1	15/2	15/3	15/4	15/5	15/6	15/7	15/8	15/9	15/10	15/11	15/12	15/13	15/14	15/15	15/16	15/17	15/18	15/19	15/20
16	16/1	16/2	16/3	16/4	16/5	16/6	16/7	16/8	16/9	16/10	16/11	16/12	16/13	16/14	16/15	16/16	16/17	16/18	16/19	16/20
17	17/1	17/2	17/3	17/4	17/5	17/6	17/7	17/8	17/9	17/10	17/11	17/12	17/13	17/14	17/15	17/16	17/17	17/18	17/19	17/20
18	18/1	18/2	18/3	18/4	18/5	18/6	18/7	18/8	18/9	18/10	18/11	18/12	18/13	18/14	18/15	18/16	18/17	18/18	18/19	18/20
19	19/1	19/2	19/3	19/4	19/5	19/6	19/7	19/8	19/9	19/10	19/11	19/12	19/13	19/14	19/15	19/16	19/17	19/18	19/19	19/20
20	20/1	20/2	20/3	20/4	20/5	20/6	20/7	20/8	20/9	20/10	20/11	20/12	20/13	20/14	20/15	20/16	20/17	20/18	20/19	20/20

에 따라 다음 블록에 입력을 공급하는 역할을 하며, 중복되는 Sequence 조합의 계산을 피하기 위해 표4 부분의 조합 생성은 피한다. Summation 블록은 계산된 적합도 값을 문턱 값과 연속허용치를 기준으로 값을 누적하며, 표4에서 연산이 생략된 부분의 값을 추가로 더해 최종 적합도 값을 산출해 낸다.

IV. 성능평가 및 Prototype 제작

1. Prototype

제안된 DNA 서열생성 시스템은 Prototype은 그림 17에서 보여 지는 것처럼 Xilinx사의 Virtex2 XCV6000 FPGA chip을 이용해 구현되었다. 사용된 Virtex2 XCV6000 FPGA는 약 6000,000개의 게이트 자세히 말하면 76,032개의 로직 요소와 2,592 kbits의 BRAM으로 구성되어 있다. 제안된 구조는 사용된 FPGA 76,032개의 로직 요소 중 38%를 이용하여 구현되었으며 40MHz로 동작한다. 내부 메모리 중 240x5000을 이용하여 Population에 해당되는 부분을 대체하였다.

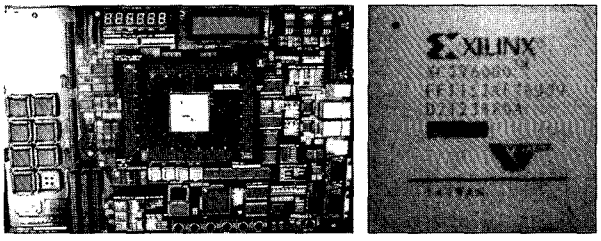

그림 16. 구현된 DNA 서열생성 시스템의 Prototype Board
Fig. 16. DNA sequence generator prototype board.

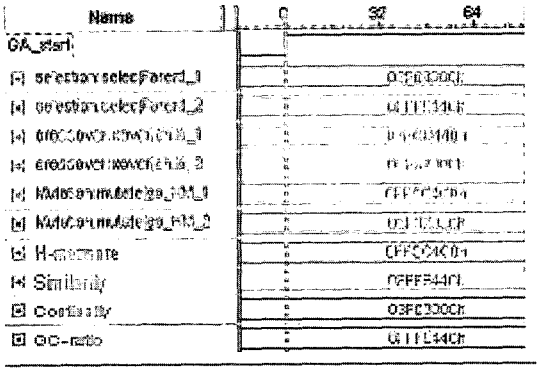

그림 16. FPGA 시뮬레이션 결과
Fig. 16. FPGA Simulation result.

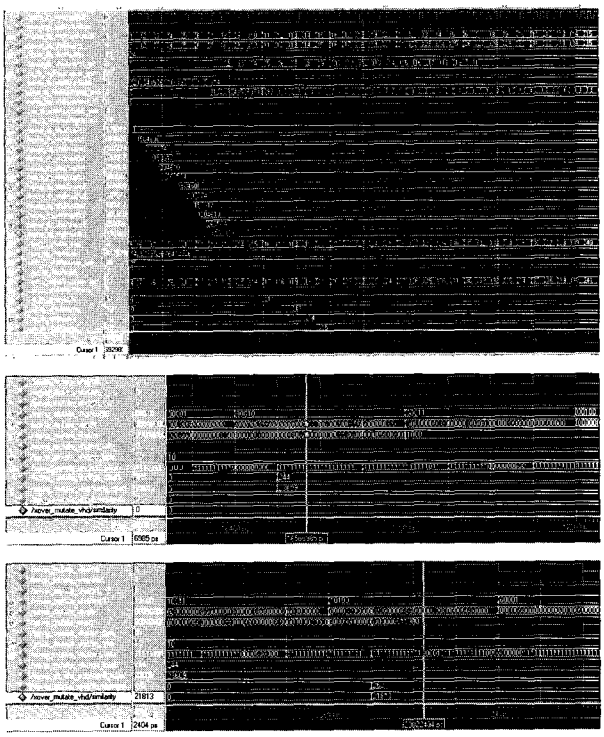

그림 17. 시뮬레이션 결과 (Modelsim)
Fig. 17. Simulation result (Modelsim).

본 논문에서 설계한 구조는 VHDL을 이용하여 설계 및 논리단계 검증을 마쳤으며 SymplicityTM의 Symplify를 이용하여 합성하고 FPGA로 구현되었다. ModelSim을 사용하여 시뮬레이션 Wave를 확인한 결과 모든 값이 정확히 나왔음을 확인할 수 있었고 구현된 prototype의 동작을 확인하기 위해 논리 분석기 (HP1663A)를 사용하였다. 그림 17은 FPGA 시뮬레이션 결과 이고 그림 18는 Functional Simulation 결과 이다. 설계한 전체 적합도함수는 Epson 0.25um Standard Cell 라이브러리로 합성되었고 Timing 검증결과 40.29MHz에서 동작하였다.

2. 시뮬레이션 결과 및 성능비교

본 연구에서 하드웨어로 구현된 DNA 서열생성 시스템의 성능을 평가하기 위해 NACST의 서열생성 시스템과의 연산시간을 비교하였으며 구현된 시스템의 최적 서열을 생성해 내는 성능을 비교하기 위해 Adleman이 Hamiltonian 경로 문제를 해결하기 위해 사용한 DNA 서열 및 NACST로 Hamiltonian 문제를 해결하기 위해 생성한 서열 과 구현된 시스템에 의해 생성된 서열을 비교하였다. 이로써 본 논문은 하드웨어로 구현된 DNA 서열생성 시스템이 NACST의 서열생성 시스템에 비하여 월등히 단축된 연산 시간을 보이며 서열을 생성하는 성능이 기존 소프트웨어와 동등함을 보였다.

가. 연산시간 비교

현재 서열 생성 시스템을 테스트하기 위해 사용하는 조건은 염기개수(sequence length) 30개에 Sequence 20개인 Chromosome과 Population 크기 1000, 세대 생성 회수 10.000 이다. 이러한 조건들을 가지고 NACST의 서열생성 모듈에 대한 연산을 PC에서 10회 이상 진행한 결과 평균 56,080초 약 15시간 정도의 연산시간을 보였다. 같은 조건으로 하드웨어로 구현된 DNA 서열생성 시스템을 40MHz의 클럭 속도를 갖는 자일링스 xcv6000보드에 구현하여 실행한 결과 평균 약 2분의 연산시간을 보였다. 이는 PC에서 동작하는 NACST의 연산시간에 비해 월등히 적은 연산시간을 보였다. 이는 하드웨어의 장점인 병렬 처리 기법 및 pipelining 기법을 통해 가능했던 것으로 생각된다. 이에 대한 테스트 조건, 환경 및 비교 결과가 표 5에 나타나 있다.

이어서 표 6에 NACST 전체의 약 95%이상의 연산시간을 차지하는 서열 생성 시스템의 연산시간의 대부분을 차지하는 적합도 함수 모듈에 대한 개별적 연산시

표 5. 제안된 하드웨어와 소프트웨어 간 연산 시간 비교
Table 5. Operation time comparison of proposed hardware with software.

테스트 조건		수치
염기 개수(Sequence 길이)		30개/Sequence
Sequence 개수		20개/Chromosome
Generation 수		100,000 generation
테스트 환경		
PC	Linux server / Pentium4 3.0GHz / 2G Memory	
FPGA board	Xilinx xcv6000 / 40MHz Clock speed	

표 6. 제안된 하드웨어와 소프트웨어 간 연산 시간 비교

Table 6. Operation time comparison of proposed hardware with software.

구분	S/W (sec)	H/W (sec)
H-measure	0.097	0.2×10^{-6}
Similarity	0.1008	0.2×10^{-6}
Continuity	0.045	0.2×10^{-6}
GC-ratio	0.032	0.2×10^{-6}
total	0.2758	0.2×10^{-6}
30,000번	8274(2hour, 16.9min)	0.6×10^{-2}

간을 비교하였다.

본 구조는 Timing 검증결과 40.29MHz에서 동작하였다. 그러므로 적합도함수 모듈의 성능은 40.29MHz의 주파수를 바탕으로 표과 같이 예측할 수 있다. 시뮬레이션은 1200bit의 개체 하나를 기준으로 30,000번 반복 수행되었으며, 시뮬레이션을 통해 제안된 하드웨어 구조가 병렬로 적합도 연산을 처리해 모든 값을 동시에 산출하는 것을 확인하였다. 개체 하나의 적합도 함수를 구하는데 걸리는 시간을 비교해 보면 위와 같은 PC 환경에서의 NACST 서열생성 시스템은 0.2758초, 구현된 DNA 서열 생성 시스템은 0.2X10-6초의 연산시간을 보였다. 이러한 결과는 적합도 함수간의 연산 공유와 병렬적 계산을 통해 연산시간이 대폭 줄어들었음을 보여준다.

나. 생성된 DNA 서열 비교

실험 1 : Adleman의 DNA 서열

DNA 컴퓨팅의 시초로 여겨지는 Adleman의 논문[6]에는 Adleman이 Hamiltonian 경로 문제를 해결하기 위해 사용한 DNA 서열 중 일부가 제시되어 있다. Adleman은 이 서열들을 임의로 생성하였다고 밝히고 있으나, 그와 같은 방식의 서열 생성은 실험 과정 중에 많은 오류를 발생시킬 가능성을 가지고 있다[12],[14]. 보다 적은 오류 가능성의 DNA 서열 생성을 위하여 Hamiltonian 경로 문제 해결에 요구되는 DNA 서열의 특징은 여러 조건들 중에서도 DAN 서열들의 고유성, '증폭하기' 연산에서의 안정성 이다. DNA 서열들의 고유성이 요구되는 이유는 Hamiltonian 경로 문제에서 그래프 상의 vertex를 표현하게 되는 각각의 DNA 서열들은 먼저 다른 vertex들과 결합을 일으키지 않아야 하며, 또한 다른 vertex와 결합할 서열들과도 결합하지 않아야 하기 때문이다[14]. 한 DNA 서열이 다른 DNA 서열들과 다른 정도는 DNA 서열 생성 기준 중 Similarity를 통해 해결하고 상보 결합에서의 '고유성'은 H-measure를 통해 확보할 수 있다. 다음으로 '증폭하기'연산의 안정성은 문제 해결의 결과 검증에 PCR 실험이 이용된다는 점을 고려하여 H-measure 와 GC-ratio 및 Melting Temperature 등을 통해 해결하고자 하였다[1]. 표 7은 Adleman이 제시한 DNA 서열[6]들과 그들에 대한 적합도 판정 결과이고, 표 8은 NACST에서 생성한 DNA 서열들과 그 적합도 판정 결과 이며, 표 9는 하드웨어로 구현된 서열 생성 모듈에 의해 생성된 DNA 서열들과 그 적합도 판정 결과 이다. 아래의 결과는 공정한 비교를 위하여 NACST에 적용된 조건과 같은 조건인 길이 20인 DNA 서열 6개를 Population의 수 100, 진화 회수 100, 돌연변이 연산 확률 0.05로 설정한 후 측정한 결과 이다.

표 7과 표 8, 표 9를 비교해 보면, H-measure와 Similarity, Continuity의 결과는 NACST와 구현된 서열생성 시스템이 전반적으로 좋은 결과를 보이고 있으나 Adleman의 서열 역시 2번째 서열을 제외 할 경우, 비교적 좋은 결과를 보이는 것으로 판단된다. 이는 Adleman이 임의 생성한 DNA 서열들 역시 어느 정도 Hamiltonian 경로 문제 해결에 맞게 디자인되었음을 의미한다. 반면 GC ratio의 경우 NACST의 서열들과 구현된 DNA 서열 생성 시스템의 경우 50%로 통일된데 비해 Adleman의 서열들은 각각35%에서 55%까지 변하는 GC ratio의 값을 보임을 알 수 있다.

생성 시스템을 통해 생성된 서열들이 Adleman의 서열에 비해 비슷하거나 더 낮은 가능성을 포함할 것으로 판단할 수 있다. 또한 NACST 및 구현된 서열 생성 시스템에 의해 생성된 서열 묶음이 Hamiltonian 경로 문제 해결에 직접 사용 가능한 수준인 것으로 생각할 수 있다. 또한 표 8과 표 9를 비교해 보면 NACST에 의해 생성된 서열들과 구현된 DNA 서열생성 시스템이 만들어 낸 서열들의 문제에 대한 적합도에 큰 차이가 없는 것을 알 수 있다. 따라서 하드웨어로 구현된 DNA 서열 생성 시스템은 NACST의 알고리즘을 그대로 수용해 생성해 내는 서열에 대한 성능은 유지 하면서 연산 시간은 크게 줄였다는 것을 알 수 있다.

따라서, 실험 과정중의 오류 발생 가능성을 고려할 때, NACST가 생성한 DNA 서열 및 구현된 DNA 서열 생성 시스템을 통해 생성된 서열들이 Adleman의 서열에 비해 비슷하거나 더 낮은 가능성을 포함할 것으로 판단할 수 있다. 또한 NACST 및 구현된 서열 생성 시

표 7. Adleman의 DNA 서열 판정 결과
Table 7. Decision result of Adleman's DNA sequence.

Sequence(5′→3′)	H-measure	Similarity	Continuity	GC %
TATCGGATCGGTATATCCGA	73	58	0	45
GCTATTCAAGCTTAAAGCTA	106	79	9	35
GGCTAGGTACCAGCATGCTT	22	23	0	55
GTATATCCGAGCTATTCGAG	57	16	0	45
CTTAAAGCTAGGCTAGGTAC	14	0	9	45
CGATAAGCTCGAATTTCGAT	-	-	9	40

표 8. NACST의 Adleman 실험 서열 생성 결과
Table 8. Ganeration result of Adleman experiment with NACST.

Sequence(5′→3′)	H-measure	Similarity	Continuity	GC %
CCTGTCAACATTGACGCTCA	48	46	0	50
TTATGATTCCACTGGCGCTC	30	51	0	50
ATCGTACTCATGGTCCCTAC	27	23	9	50
CGCTCCATCCTTGATCGTTT	15	4	0	50
CTTCGCTGCTGATAACCTCA	13	4	0	50
GAGTTAGATGTCACGTCACG	-	-	0	50

표 9. 구현된 서열생성 시스템의 Adleman 실험 서열 생성 결과
Table 9. Ganeration result of Adleman experiment with Proposed sequence generator system.

Sequence(5′→3′)	H-measure	Similarity	Continuity	GC %
ATGCATCATCTGTCGCCTCA	43	37	0	50
TTCGCTCGCTAGATACCTAC	28	48	0	50
CTCGTACCAATGTCAGTCCA	23	27	0	50
TAGGATAGTGTACCGCTAGC	16	4	0	50
TTCCGTCCGAGTTACATCCA	14	0	0	50
TTGATATTCACCTGGCCTGC	-	-	0	50

스템에 의해 생성된 서열 묶음이 Hamiltonian 경로 문제 해결에 직접 사용 가능한 수준인 것으로 생각할 수 있다. 또한 표8 과 표9를 비교해 보면 NACST에 의해 생성된 서열들과 구현된 DNA 서열생성 시스템이 만들어 낸 서열들의 문제에 대한 적합도에 큰 차이가 없는 것을 알 수 있다. 따라서 하드웨어로 구현된 DNA 서열 생성 시스템은 NACST의 알고리즘을 그대로 수용해 생성해 내는 서열에 대한 성능은 유지 하면서 연산 시간은 크게 줄였다는 것을 알 수 있다.

V. 결 론

최근 현재의 시스템으로는 해결이 어려운 문제들을 DNA 컴퓨팅을 이용해 해결하려는 노력이 계속되고 있다. 그러나 하나의 문제를 해결하는데 약 일주일 정도의 시간이 걸리는 실제 실험으로는 계산 모델의 빠른 검증 및 개발이 불가능 하다. 지금까지 본 논문에서는 DNA 서열생성 시스템의 빠른 연산시간 확보를 위한 서열생성 시스템의 효율적인 하드웨어 구조를 제안하고 구현하였다. 제안된 하드웨어 구조는 막대한 양의 데이터를 빠르게 처리하기 위해 병렬처리 기법과 pipelining 기법을 적용하였으며, 연산시간 중 많은 부분을 차지하는 적합도 함수 모듈의 효율적인 설계를 위해 함수들의 연산을 분석하여 일부 함수들 간 연산을 공유 시키고 병렬처리 기법을 적용하여 적합도 함수가 소모하는 연산시간의 많은 부분을 줄였다. 결과적으로 제안된 하드웨어는 소프트웨어에 비해 약 467배 정도의 실행시간을 단축시켰으며, NACST 서열생성 시스템의 알고리즘은 그대로 수용하면서도 서열생성의 성능은 그대로 유지시켰다. 또한 제안된 하드웨어는 유전자 알고리즘의 연산자 및 적합도 함수를 독립적으로 설계해 향후 서열생성 시스템에 새로운 유전자 알고리즘을 수용한다고 하더라도 본 논문에서 제안한 병렬 구조 및 pipelining 구조를 쉽게 적용할 수 있도록 하였다. 따라서 본 연구는 근래에 생물 정보학(Bioinformatics) 등에서 이슈가 되고 있는 Hybridization 시뮬레이션 및 기타 생화학 실험의 시뮬레이션에서, 그 과정 중 사용될 DNA 서열들을 빠르게 생성 검증할 수 있게 함으로써 전체 시스템 및 알고리즘 개발에 충분한 도움이 되리라 예상하고, DNA 컴퓨팅의 훌륭한 보조 수단이 되리라 생각한다.

참 고 문 헌

[사전] 생물학사전, 아카데미 서적, 1998.
[1] Maley, C. C.,"DNA Computation: Theory, Practice, and Prospects", Evolutionary Computation 6(3): pp.201-229, 1998.
[2] 신수용,"DNA 컴퓨팅 기법을 이용한 조합 최적화 문제의 해결", 서울대학교 학위논문집 1999, pp. 12-27.
[3] T. A. Brown, GENOMES, John Wiley & Sons (Asia) Pte Ltd, 1999, pp.3.
[4] 김동민, "유전자 알고리즘을 이용한 DNA 서열 생성 시스템", 서울대학교 학위논문집 2001, pp.7-20.
[5] G. Paun, G. RoBenberg, and A. Salomaa, DNA

Computing: New Computing Paradigns, Springer, 1998, pp. 3-40.
[6] Adleman, L. M., "Molecular computation of solutions to combinatorial problems", Science, 266:1021-1024, 1994.
[7] Deaton, R., Murphy, R. C., Garzon, M., Franceschetti, D. R. Stevens, S. E. Jr.., "A DNA based implementation of an evolutionary search for good encodings for DNA computation". Proceedings of IEEE Conference on Evolutionary Computation, IEEE Press, pp. 267-271, 1997.
[8] J.Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, 1975.
[9] 장병탁, "인공 생명과 진화 알고리즘", 전자공학회지, 제24권 제 3호, pp.51-60., 1997년 3월.
[10] D. E. Goldberg, Genetic Algorithm in search, Optimization, and Machine Learning, Addison-Wesley, 1989.
[11] G. Winter et al., Genetic Algorithms in Engineering and Computer Science, John wiley & Sons, 1996.
[12] R. Deaton, R. C. Murphy, M. Garzon, D. R. Franceschetti, and Jr. S. E. Stevens, "Good encodings for DNA-based solutions to combinatorial problems," in Proceedings of the Second Annual Meeting on DNA Based Computers, 1996.
[13] Peter D. hortensius et al., "Parallel Random Number Generation for VLSI System Using Celluar Automata" IEEE Trans. on Computers. Vol.38. No. 10 pp. 1466-1473 October 1989.
[14] R. Deaton, M. Garzon, R. C. Murphy, D. R. Franceschetti, and Jr. S. E. Stevens, "Genetic search of reliable encodings for DNA based computation", in First Conference Genetic Programming, MIT Press. 1996.
[15] N. Yoshida, T. Moriki and T.Yasuoka, "GAP: Genetic VLSI processor for genetic algorithm", Second International ICSC Symp. on Soft Computing, pp.341-345., 1997.
[16] Dan Mihaila, Florin Fagarasan, Mircea Gh. Negoita, "Architectural Implications Of Genetic Algorithms Complexity In Evolvable Hardware Implementation" EUFIT'96, Vol.1, pp.400-404 September 1996.
[16] Melanie Mitchell, An introduction to Genetic Algorithms, The MIT Press, 1997.
[17] T. Fogarty and R. Huang, " Implementing the Genetic Algorithm on Transputer Based Parallel Processing Systems", Parallel Problem Solving from Nature, pp.145-149. Springer-Verlag, 1991.
[18] M. Tommiska, J. Vuori, "Implementation of genetic 미해갸소 一ㄴ with programmable logic devices", Proceeding of the 2NWGA, August 1996.
[19] N. Uoshida, T. Yasuaka and T. Moriki, "Parallel and Distributed Processing in VLSI Implementation of Genetic Algorithms", Third Int'1 ICSC Symp. On Soft Computing, June 1999.
[20] M. Salami, "Multiple genetic algorithm processor for hardware optimization", Proc. First International Conference, IECS96 Evolvable System: From Biology to Hardware, pp. 249-259., October 1996.
[21] Barry Shackleford, Etsuko Okushi et al., "A High-performance Hardware Implementation of a Suvival-based Genetic Algorithm", ICONIP'97, 00. 686-695, Nov, 1997.
[22] N. Jonaska and N. C. Seeman, Eds., DNA Computing, 7th International Workshop on DNA-based Computers, Tampa, U.S.A., 10-13 June 2001. University of South Florida.
[23] Q. Ouyang, P.D. Kaplan, S. Liu, and A. Libchaber, "DNA solution of the maximal clique problem,"Science, vol. 278, pp. 446-449, 1997.

저 자 소 개

이 은 경(정회원)
2004년 가톨릭대학교 반도체 공학과 학사 졸업
2006년 인하대학교 정보통신 공학과 석사 졸업
2006년~삼성전자 반도체총괄 메모리 사업부
<주관심분야 : 반도체, 메모리, VLSI 및 SoC 설계, Embedded System 설계>

이 승 렬(정회원)
2003년 인하대학교 반도체공학과 학사 졸업
2005년 인하대학교 정보통신 공학과 석사 졸업
2005년~현재 인하대학교 정보통신공학과 박사과정
<주관심분야 : 통신 시스템 설계, WPAN, WLAN, VLSI 설계>

김 동 순(정회원)
1997년 인하대학교 전자재료 공학과 학사 졸업
1999년 인하대학교 전자재료 공학과 석사 졸업
2005년 인하대학교 정보통신 공학과 박사 졸업
1999년~전자부품연구원 DxB 통신융합 연구센터
<주관심분야 : 통신, DMB, VLSI 및 SoC 설계

정 덕 진(종신회원)
1970년 서울대학교 전기공학과 학사 졸업
1984년 미국 Utah State University 석사 졸업 (Electrical Eng.)
1988년 미국 Utah State University 박사 졸업 (Electrical Eng.)
1989년~현재 인하대학교 정보통신공학교 교수
<주관심분야 : VLSI 및 SoC 설계, 컴퓨터 구조 및 Embedded System 설계>