고속 메모리 테스트를 위한 파이프라인 ALPG

A Pipelined ALPG for Ultra-High Speed Memory Testing

윤현준, 양명훈, 김용준, 박영규, 이대열, 강성호
연세대학교 전기전자공학과
hjyoon@soc.yonsei.ac.kr shkang@yonsei.ac.kr

Abstract

In order to perform effective testing for ultra-high speed memory devices, a new pipelined Algorithmic Pattern Generator (ALPG) architecture is developed. The proposed ALPG provides the efficiency using the pipelined architecture of the instruction analyzer. This pipelined ALPG can generate patterns for memory devices at high speed.

I. 서론

최근 반도체 기술의 급속한 발전에 따라, 고속 메모리가 등장하고 있다. 이에 따라 이러한 메모리를 테스트할 수 있도록 빠르게 패턴을 생성하는 ALPG의 필요성이 급증하고 있다. 또한, 메모리 회로의 복잡도가 증가함에 따라 보다 다양하고 복잡한 고장을 잡아낼 수 있는 다양한 종류의 패턴이 요구된다.

본 논문은 고속 메모리를 테스트하는 동시에, 사용자가 원하는 알고리즘을 사용할 수 있는 ALPG를 소개한다. 이 ALPG는 기존에 소개된 것과는 달리, 미리 명령어 리스트를 만들어 합성하지 않고, 사용자가 원하는 명령어를 그대로 사용할 수 있는 하드웨어 구조를 가지고 있다.

II. 기존의 연구

고속 메모리를 테스트하기 위한 기존의 연구 중 가장 광범위하게 사용되었던 연구로, 네 개의 Pattern Generator(PG)를 병렬로 연결하여 만든 [1]과 [2]의 연구를 꼽을 수 있다. [2]의 경우, 각 PG는 미리 합성된 명령어를 수행함으로써 네 배의 속도를 내는 ALPG를 만들어낸다. 즉, 첫 번째 PG는 하나의 명령어를 수행하고, 두 번째, 세 번째, 네 번째 PG는 각각 두 개, 세 개, 네 개의 명령어가 미리 합성된 명령어를 한 사이클 내에 수행한다. 예를 들어, X를 하나의 레지스터라고 하고 X의 값에 1을 더하여 X에 저장하는 명령어인 X < X + 1를 네 번 수행하는 과정이라고 할 때, 네 번째 PG에서는 X < X + 4의 명령어를 수행해야 한다.

하지만, 이러한 방법으로 미리 명령어를 합성하여 만들 경우, 합성해야 하는 명령어의 개수가 셀 수 없을 정도로 많다는 문제점이 있다. 즉, 모든 경우의 수를 따져서 합성된 명령어를 미리 정해놓는다는 것은 한계가 있다. 많이 사용하는 MARCH 알고리즘 등의 알고리즘에서 주로 쓰이는 명령어의 리스트를 만들어서 미리 명령어를 합성해 놓으면 어느 정도의 명령어는 적용할 수 있다. 하지만, 사용자가 직접 만든 명령어를 사용할 경우, 명령어 리스트 이외의 다른 명령어를 합성해야 되므로 이 방법은 사용할 수 없다는 문제점이 있다.

III. 제안하는 ALPG 구조

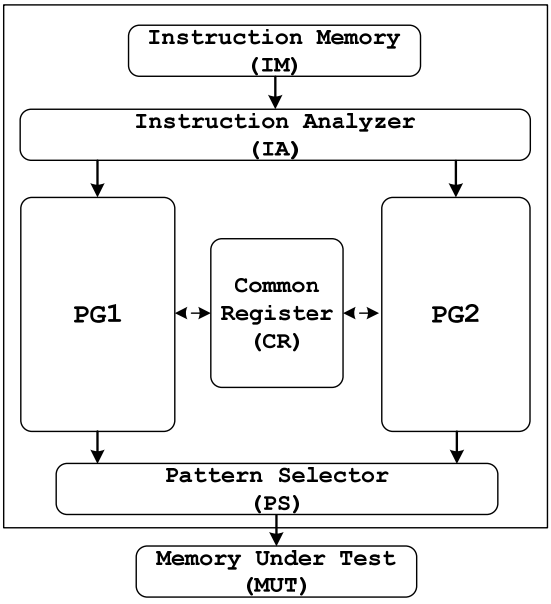

그림 1. 파이프라인 ALPG의 구조

사용자가 원하는 새로운 명령어를 사용하기 위한 ALPG 구조는 다음과 같다. 이 구조는 그림 1과 같이 크게 Instruction Memory(IM), Instruction Analyzer(IA), 2개의 PG, Common Register (CR), Pattern Selector(PS)로 구성되어 있다.

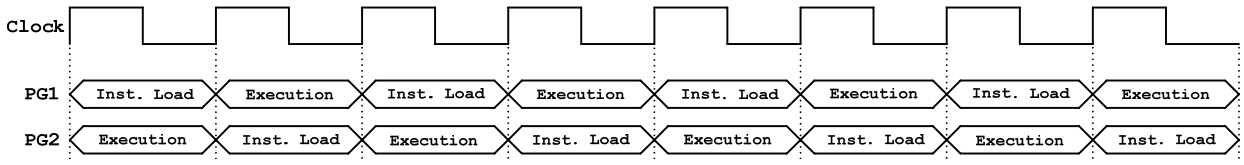

그림 3. ALPG의 파이프라인 구조

IM에는 사용자가 원하는 명령어가 저장되어 있고, 각 명령어는 [2]에서 제시한 바와 같이 sequential 부분, address 부분, data 부분, control 부분으로 구성되어 있다. 하나의 명령어가 IA로 입력되면 IA는 명령어의 sequential 부분을 분석하여, No operation(NOP), Loop, Jump, End 여부를 가려낸다. IA는 Loop, Jump 등이 섞여있는 모든 명령어 리스트를 NOP으로만 이루어진 명령어 리스트로 만들어, 명령어를 정렬하여 주는 역할을 하게 된다. 그림 2는 Loop 명령이 있는 경우, IA가 어떻게 명령어 리스트를 만드는지 보여준다. 이 그림에서 Loop Number는 Loop 횟수를, NOP는 NOP을, Loop_A는 A에서 Loop_A까지의 Loop를 의미한다.

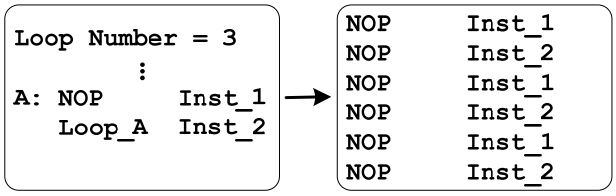

그림 2. IA에 의한 명령어 리스트 생성

IA가 만드는 각 명령어는 하나씩, 두 개의 PG에 번갈아 가며 순서대로 입력된다. 즉, n=0,1,2,3... 이라고 할 때, IA에서 만들어지는 명령어 리스트의 $2n$, $2n+1$번째 명령어는 각각 첫 번째, 두 번째 PG로 입력된다. 각 PG는 입력되는 패턴의 address 부분, data 부분, control 부분을 파싱하고 필요한 값과 컨트롤 신호를 생성하게 된다. 이때 sequential 부분을 IA가 NOP로 만들었으므로 PG는 sequential 부분을 파싱할 필요가 없다. PG의 동작은 두 가지 과정으로 나눠진다. 첫 번째는 Instruction Load이며 IA에서 해당 PG로 명령어가 입력되는 과정이다. 두 번째는 Execution으로 입력된 명령어가 실행되는 과정이다. 이때 첫 번째 PG에서 Instruction Load가 수행되는 동안 두 번째 PG에서는 Execution이 수행되며, 첫 번째 PG에서 Execution이 수행되는 동안 두 번째 PG에서는 Instruction Load가 수행된다. 이와 같이 두 PG에서 Instruction Load와 Execution이 번갈아 실행되므로 두 배의 성능 향상을 기대할 수 있다.

각 PG가 생성하는 패턴은 CR에 저장되었다가 PS에 의해 선택되어 다음 클럭에 Memory Under Unit(MUT)으로 입력된다. 이때, 각 클럭 마다 두 PG가 교대로 CR에 저장하되 address 부분과 data 부분에서 필요한 경우에는, 이미 CR에 저장되어 있는 값과 더하여 CR에 저장하게 된다. 이러한 과정으로, 예상되는 명령어를 미리 합성하여 저장해야만 하는 기존의 문제점을 간단하게 해결할 수 있다.

IV. MARCH Algorithm의 실행 예

```
START #0
    A  : NOP        XB<0 YB<0 TP<TPH
         Loop .  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX
    B  : NOP     R  X<XB Y<YB
         Loop_B  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX /D
    C  : NOP     R  X<XB Y<YB /X /Y                    /D
         Loop_C  W  X<XB Y<YB /X /Y XB<XB+1 YB<YB+1^BX

         GoTo&Inverse_A
STOP
END
```

그림 4. MARCH 알고리즘 명령어

파이프라인 ALPG를 사용하여 MARCH 알고리즘을 실행했을 때의 예를 살펴보자. 그림 4는 MARCH 알고리즘을 구현하기 위한 명령어로 [3]에서 쓰인 형식과 비슷하다. 간단히 설명하면, X는 메모리 열의 주소이고 Y는 메모리 행의 주소이다. XB와 YB는 각각 X, Y 주소 레지스터이며, TP는 데이터 레지스터이다. 그리고 TPH는 TP의 초기값으로 0값을 가진다. XB<XB+1은 X값을 1만큼 증가시키는 명령어이고, YB<YB+1^BX는 X값이 한 행의 마지막 열의 주소가 되었을 때 그 다음 행의 주소가 Y값, 첫 번째 열의 주소가 X값이 되도록 하는 명령어이다. /X, /Y, /D는 각각 X, Y, D의 반전된 값이고, Goto&Inverse_A는 A의 명령어부터 다시 시작하되, TP값을 반전시킨다. 그러므로, 그림 4와 같은 명령어를 사용하여 MARCH 알고리즘을 구현할 수 있다.

그림 4의 명령어를 위에서 제안한 ALPG 구조에 적용해보자. 먼저, MARCH 알고리즘 명령어는 IM에 저장되었다가, 한 명령어씩 IA로 입력된다. 위에서 말한 것과 같이 IA는 그림 4의 명령어를 NOP으로만 이루어진 명령어 리스트로 만든다. 그림 5는 만들어진 명령어 리스트이며, 각 사각형은 미리 지정된 수, 즉

모든 cell의 개수만큼만 반복된 명령어의 집합이다.

```
START #0
NOP     XB<0 YB<0 TP<0
NOP  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX
                  ⋮
NOP  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX

NOP  R  X<XB Y<YB
NOP  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX /D
                  ⋮
NOP  R  X<XB Y<YB
NOP  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX /D

NOP  R  X<XB Y<YB /X /Y                    /D
NOP  W  X<XB Y<YB /X /Y XB<XB+1 YB<YB+1^BX
                  ⋮
NOP  R  X<XB Y<YB /X /Y                    /D
NOP  W  X<XB Y<YB /X /Y XB<XB+1 YB<YB+1^BX

NOP     XB<0 YB<0 TP<1
NOP  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX
                  ⋮
NOP  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX

NOP  R  X<XB Y<YB
NOP  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX /D
                  ⋮
NOP  R  X<XB Y<YB
NOP  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX /D

NOP  R  X<XB Y<YB /X /Y                    /D
NOP  W  X<XB Y<YB /X /Y XB<XB+1 YB<YB+1^BX
                  ⋮
NOP  R  X<XB Y<YB /X /Y                    /D
NOP  W  X<XB Y<YB /X /Y XB<XB+1 YB<YB+1^BX

STOP
END
```

그림 5. IA에 의해 만들어진 명령어 리스트

그림 5와 같이 만들어진 명령어는 IA에 의해 한 클럭에 한 명령어가 한 개의 PG에 들어가며, 연속적으로 모든 명령어가 두 개의 PG에 차례대로 들어간다. 모든 cell에 0을 쓰는 명령어의 집합인 그림 5의 첫 번째 사각형은 그림 6과 같이 합성되지 않은 NOP 명령어의 형태로 각 PG에 입력된다. 만약 열의 주소의 크기가 8 이상일 경우, X, Y 주소는 그림 6의 오른쪽에 나타낸 것과 같이 바뀌게 되며, 이는 두 PG가 서로 다른 클럭에서 CR에 접근하기 때문에 가능하다. 각 클럭에서의 두 PG의 Execution 과정에 의한 X, Y 레지스터 값의 변화는 그림 7과 같이 나타나는 것을 알 수 있다. 이와 같이 CR을 사용함으로써 명령어를 합성하지 않고 파이프라인 ALPG의 두 PG에 NOP 명령어를 입력하여 패턴을 생성하는 것을 확인할 수 있다.

```
NOP  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX ⟶ PG1 ⇢ X=1, Y=0
NOP  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX ⟶ PG2 ⇢ X=2, Y=0
NOP  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX ⟶ PG1 ⇢ X=3, Y=0
NOP  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX ⟶ PG2 ⇢ X=4, Y=0
NOP  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX ⟶ PG1 ⇢ X=5, Y=0
NOP  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX ⟶ PG2 ⇢ X=6, Y=0
NOP  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX ⟶ PG1 ⇢ X=7, Y=0
NOP  W  X<XB Y<YB       XB<XB+1 YB<YB+1^BX ⟶ PG2 ⇢ X=8, Y=0
                  ⋮                          ⋮          ⋮
```

그림 6. 명령어 리스트의 PG 입력

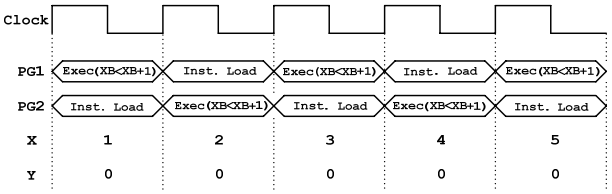

그림 7. Execution 과정에 따른 X, Y 레지스터 값 변화

V. 결론

본 논문에서는 두 개의 PG에 순차적으로 명령어를 입력하는 파이프라인 ALPG 방법을 제안한다. 제안하는 방법은 명령어를 처음부터 끝까지 순서대로 두 개의 PG에 교대로 입력하고, 각 PG의 결과 값과 컨트롤 신호를 공통의 레지스터에 입력하였다가 MUT에 넣는 방법이다. 두 PG는 파이프라인 구조로 동작하고, sequential 부분을 제외한 명령어만 파싱하여 실행하면 된다. 따라서 본 논문에서 제안하는 방법은 속도를 두 배로 증가시켜 메모리를 테스트하는 동시에, 수많은 명령어를 합성하지 않고 쉽게 명령어를 입력하여 원하는 패턴을 만들어낼 수 있다는 장점이 있다.

Acknowledgment

본 논문은 IDEC(IC Design Education Center)의 CAD tool 지원을 받은 것임.

참고문헌

[1] S, Kikuchi, Y. Hayashi, T. Matsumoto, R. Yoshino, R. Takagi, “A 250 MHz shared-resource VLSI test system with high pin count and memory test capability”, Proceedings of IEEE International Test Conference, pp. 558-566, 1989.

[2] S. Kikuchi, Y. Hayashi, T. Matsumoto, R. Yoshino, R. Takagi, “Generation technique of 500 MHz ultra-high speed algorithmic pattern”, Proceedings of IEEE International Test Conference, pp. 677-684, 1996.

[3] ADVANTEST CORPORATION, “ATL-51 Pattern Program Reference Manual”, 2004.