마이크로프로세서 기능 검증을 위한 바이어스 랜덤 벡터 생성기 설계

권 오 현, 양 훈 모, 이 문 기
연세대학교 전기전자공학과 VLSI&CAD연구실
전화 : 02-2123-4731 / 핸드폰 : 011-9160-7432

Design of A Biased Random Vector Generator for A Functional Verification of Microprocessor

Oh-Hyun Kwon, Hoon-Mo Yang, Moon-Key Lee
Dept. of Electrical and Electronic Engineering, Yonsei University
E-mail : kwon0108@spark.yonsei.ac.kr

Abstract

In this paper, we propose a bias random vector generator which can verify functions of microprocessor effectively. This generator is a pre-processor of assembly program, and defines pre-processor instructions which create random vector only in the part which the designer wants to verify. Therefore, this generator shows higher detection ration than any other generators. And, we can cut down design costs because of shortening a period for verifying function.

I. 서론

시뮬레이션을 통한 기능 검증 방식은 검증 결과의 신뢰성이 워크로드의 수행 량에 비례하는 귀납적인 특성을 지닌다. 따라서 이론적으로 완벽한 신뢰성을 얻기 위해서는 존재 가능한 모든 명령어 조합 유형에 대한 시뮬레이션을 수행해야 한다. 그러나 마이크로프로세서와 같은 복잡한 시스템의 경우, 그런 소모적인 시뮬레이션은 명령어 조합의 유형 수가 명령어 비트 크기에 비례하여 지수적으로 증가하므로 사실상 16비트급 이상만 되어도 적용 불가능하다.

따라서 효율적인 기능 검증을 위해서는 검증 대상 시스템의 모든 corner case를 대표할 수 있는 적절한 양의 표본 테스트 벡터가 요구된다. 기존에는 설계자가 미리 발생 가능한 오류 상황을 예측하여 직접 작성하는 방식을 주로 사용하였으나 이러한 방식은 오류 검출에 유리한 방향으로 명령어 조합을 조절하기 힘들기 때문에 부적절하다. 반면 이에 대한 대안으로 사용되는 랜덤 테스트 벡터는 작성이 매우 간편하며 설계자가 미처 간파하지 못한 부분에 대한 오류 발견에 매우 탁월하고 새로운 오류 유형의 정립 지침을 제공하기도 한다. 그러나 실제 오류를 검출하는 테스트 벡터 표본은 모집합에 대하여 균일하게 분포하는 것이 아니라 특정 유형과 강한 상관 관계를 지니는 경우가 대부분이다. 따라서 순수 랜덤 벡터의 경우 이러한 상관 관계를 제대로 반영하지 못하기 때문에 효율성이 매우 떨어지는 단점이 있다.

따라서 본 논문은 설계자 의도에 따른 테스트 벡터와 랜덤 테스트 벡터의 장점을 포괄한 새로운 형태의 바이어스 랜덤 벡터 생성기를 제안한다.

본 랜덤 벡터 생성기는 어셈블리 언어 수준의 워크로드 소스 코드에 대해 구문 단위로 무작위성을 제어하는 기능을 제공한다. 따라서 설계자는 검증하고자 하는 기능에 대한 상관 관계를 미리 설정하고 부분적으로 무작위성을 부여함으로써 랜덤 벡터의 효율성을 극대화시킨다.

본 논문의 내용은 II 장에서 제안한 바이어스 랜덤 벡터 생성기의 사양과 구조를 살펴보고 III 장에서 랜덤 벡터를 이용한 기능 검증 전략을 제안하며 최종 결론을 제시한다.

표 1. 랜덤 벡터 생성기 전처리 명령어

구문	동작
%DEVAR `identifier *lower* : *upper* : *step*	[*lower*, *upper*] 범위의 *step*에 대한 등차수열 형태를 지닌 정수 랜덤 마크로 변수를 정의
%DEVAR `identifier *string*	*string*을 값으로 하는 문자열 랜덤 마크로 변수 정의
%DEVAR `identifier $`pre_defined_identifier	이미 정의된 마크로 변수를 포함하는 새로운 랜덤 마크로 변수 정의
%DEVAR `identifier (*first*, ..., *last*)	값 집합을 갖는 랜덤 마크로 변수 정의, 값 집합 요소는 정수, 문자열, 그리고 랜덤 마크로 변수를 포함
%DEVAR `identifier [*width*] \ (*first_value* [*first_bit_range*], ... \ *last_value* [*last_bit_range*])	비트열 랜덤 마크로 변수 정의, *width*는 비트열의 길이, *value*, *range* 쌍은 각각 랜덤하지 않은 부분 비트 열의 값과 위치를 나타낸다.
%DEVAR `identifier [UNIQUE]	할당가능한 마크로 변수 정의
%DEFUN `identifier [GUARD]* `statements %ENDFUN	마크로 함수 정의, 함수 인자의 프로토타입 검사를 하지 않는다.
%SELECT `first_statements %OR `second_statements %OR .... %ENDSEL	무작위로 하나의 구역만을 선택하여 출력한다.
%REPEAT *number* [GUARD]* `statements %ENDREP	무작위로 구역을 *number* 만큼 반복하여 출력한다.
$`identifier	정의된 마크로 변수의 쓰기/읽기 접근
$`identifier(*first*, ..., *last*)	정의된 마크로 함수의 호출
$`number	호출된 마크로 함수 내에서의 함수 인자를 지칭, 숫자는 함수 호출 시 인자의 순서를 나타낸다.
%EVAL `identifier `express	마크로 변수와 상수로 이루어진 식을 계산하여 마크로 변수에 할당

II. 바이어스 랜덤 벡터 생성기의 구조

1. 랜덤 벡터 생성기 사양

C/C++과 같은 상위 수준 언어는 구문 합성을 위해 사용되는 기계어의 종류와 조합이 제한되므로 기능 검증용 워크로드로는 어셈블리 언어가 더 적합하다. 따라서 본 랜덤 벡터 처리기는 기본적으로 어셈블리 수준 소스 코드의 전처리기로서 동작하도록 설계하였다.

사용자는 어셈블리 소스 코드에 전처리기 명령어를 삽입함으로써 무작위성을 부여하고 벡터 생성기는 전처리기 명령어를 해석 수행하여 최종적인 어셈블리 소스 코드를 생성한다.

본 랜덤 벡터 생성기가 지원하는 전처리 명령어는 크게 3가지 부류로 분류된다.

첫 번째 유형은 무작위하게 문자열이나 정수, 비트 열을 발생시키는 랜덤 마크로 변수이다. 랜덤 마크로 변수는 일반적인 프로그램 언어의 마크로 변수와 유사하나 차이점은 후자가 한 가지 값을 가지는 반면 전자는 미리 정의된 값 집합에서 무작위로 선택된 요소를 취한다는 점이다.

두 번째 유형은 구문 생성을 제어하는 구문 제어 문으로 특정 구역을 정해진 회수만큼 반복 생성하는 REPEAT 문과 여러 구문 중 무작위로 선택된 하나의 구문을 생성하는 SELECT 문으로 구분된다.

셋째는 마크로 함수 구문으로 일반적인 프로그램 언어의 마크로 함수와 동일하다. 전처리기 명령어의 자세한 사양은 표 1에 정리하였다.

2. 랜덤 벡터 생성기 구현

랜덤 벡터 생성기의 내부 구조는 그림 2와 같다. 랜덤 벡터 생성기는 전처리기 명령어가 포함된 어셈블리 소스 코드를 입력 받아 중간 코드를 합성한다. 중간 코드는 최종 출력을 수행하기 위한 전단계 원시 코드로서 변수 값 생성, 출력 생성을 위한 코드 제어 및 최종 문자열 출력 등을 수행한다. 정의된 중간 코드는 표 2와 같다.

```
// 랜덤 변수 정의 및 치환
%DEVAR OPCODE (LDR, STR)
$OPCODE R0, [R1, R2]      →   LDR R0, [R1, R2]
%DEVAR INDEX 0:15:2
MOVEQ R$INDEX, R3         →   MOVEQ R4, R3

// 반복 구문
%REPEAT 2
LDR R$INDEX, [R4]         →   LDR R0, [R4]
CMP R$INDEX, R5               CMP R0, R5
%ENDREP                       ----------
                              LDR R2, [R4]
                              CMP R2, R4

// 랜덤 선택 구문과 반복 구문
%REPEAT 4
%  SELECT
       ASR R0, R1         →   ASR R0, R1
%  OR                         ----------
       MOV R1, R2             NOP
%  OR                         ----------
       NOP                    MOV R1, R2
%  ENDSEL                     ----------
%ENDREP                       MOV R1, R2

// 마크로 함수 정의 및 호출
%DEFUN F
    MOV $1, $2
%ENDFUN

%REPEAT 2                 →   MOV R0, R2
   $F(R$INDEX, R2)            ----------
%ENDREP                       MOV R8, R2
```

그림 1. 랜덤 벡터 생성기 전처리 명령어 사용 예

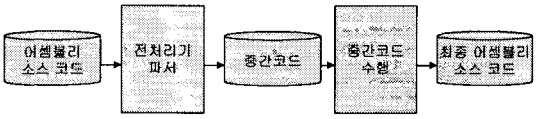

그림 2. 랜덤 벡터 생성기 내부 동작 흐름도

랜덤 벡터 생성기는 REPEAT에 의한 반복 문이나 마크로 함수 호출 수행 시, 동일한 레이블이 반복 출력되는 문제가 발생한다. 이 경우 랜덤 벡터 생성기 자체는 오류가 아니나 이후 어셈블러에 의해 레이블 중복 정의 오류가 발생한다. 본 랜덤 벡터 생성기는 이러한 문제점을 해결하기 위해 두 가지 방법을 제안한다.

첫 번째 방법은 REPEAT 문이나 마크로 함수 정의문에 GUARD라는 키워드를 사용하는 방식이다. 이 방식은 지역 레이블을 지원하는 어셈블러에만 적용 가능하다. GUARD 속성은 구역 경계에 광역 레이블을 삽입한다. 따라서 구역 내부에 동일한 지역 레이블을 사용해도 이름 공간 충돌을 방지할 수 있다.

두 번째 방법은 UNIQUE라는 키워드를 사용하는 방식이다. 마크로 변수에 UNIQUE 속성을 부여할 경우, 접근할 때마다 카운트를 수행하여 고유의 키를 발생시킨다. 따라서 이를 레이블로 이용하여 동일한 이름이 발생하는 것을 방지할 수 있다.

표 2. 랜덤 벡터 생성기의 중간 코드

코드 니모닉	동작
EXIT	종료
JA *L*	L 레이블로 무조건 분기
JC [*C*] *L*	카운트 C가 0이 되면 L로 분기
JR *LA*	레이블 집합 LA 중 하나로 무작위 분기
AC ← CONST	어큐뮬레이터에 상수 저장
AC ← RAND	어큐뮬레이터에 랜덤 변수 저장
ENTER	마크로 함수 프레임 생성
BL *L*	마크로 함수 호출
LEAVE	마크로 함수 프레임 제거 및 리턴
PUSH	마크로 함수 인자 스택에 저장
AC ← ARG	어큐뮬레이터에 함수 인자 저장
OUT RAND	랜덤 변수를 출력
OUT ARG	함수 인자를 출력
OUT BUFFER	버퍼를 출력. 버퍼는 전처리기 명령어를 제외한 구문이 저장됨.

```
// GUARD 키워드, LTEMP는 지역 레이블
%REPEAT 2 GUARD
LTEMP: MOV R0, R1 →  REPEAT_10_0:
%ENDREP                LTEMP: MOV R0, R1
                     ENDREP_10_0:
                     REPEAT_10_1:
                       LTEMP: MOV R0, R1
                     ENDREP_10_1:

// UNIQUE 키워드
%DEVAR X UNIQUE

%REPEAT 4        → X_0: MOV R0, R1
$X: MOV R0, R1     X_1: MOV R0, R1
%ENDREP            X_2: MOV R0, R1
                   X_3: MOV R0, R1
```

그림 3. 반복문이나 함수 호출 시 레이블 중복 방지

III. 랜덤 벡터를 이용한 기능 검증 전략

랜덤 벡터 수행 시 가장 큰 문제점은 시뮬레이션 수행 중 적법하지 않은 메모리 영역을 참조함으로써 수행 도중 오동작을 일으키거나 비정상적인 종료가 발생하는 경우이다. 적법하지 않은 메모리 영역 참조는 명령어의 잘못된 분기나 메모리 로드/스토어 명령어에 의한 잘못된 데이터 주소에 대한 읽기/쓰기로 구분된다. 이와 같은 문제점을 방지하기 위한 방법으로 두 가지 방식이 있다. 첫 번째 방식은 발생한 랜덤 벡터 코드를 실제 명령어 시뮬레이터에서 수행하여 적법하지 않은 메모리 영역을 참조하는 경우, 기존 랜덤 벡터 코드를 변경하거나 새로운 명령어를 첨가 가공하는 방식이다. 이와 같은 방식은 후방(backward) 분기가 많은 워크로드의 경우 용량을 최소화할 수 있는 장점이 있는 반면 별도의 명령어 시뮬레이터 및 명령어 변환기가 필요하다.[1] 두 번째 방식은 매 버스 사이클마다 프로세서에서 인출되는 주소 대신 별도의 카운터에서 발생하는 순차적인 값을 명령어/데이터 참조 주소로 사용하고 프로세서에서 인출되는 주소는 오직 검증 대상으로서만 사용하는 방식이다.[2] 따라서 명령어는 프로세서 내부의 분기 여부에 관계없이 무조건 순차적으로 수행되며 명령어에서 계산한 데이터 주소와 실제 전송되는 데이터 사이에는 아무런 연관성이 없다. 이러한 방식은 후방 분기가 많은 워크로드의 경우, 용량이 매우 커질 수 있다는 단점이 존재한다. 그러나 실제로 후방 분기나 루프 문 처리는 기능 검증보다는 메모리의 지역성에 의거한 성능 평가에 더 중요한 요소이다. 반면 기능 검증은 소수 명령어 조합에 대한 다양한 패턴 수행을 요구하므로 후방 분기나 루프 문의 빈도가 매우 낮다. 따라서 본 랜덤 벡터 생성기는 후자의 경우를 가정하여 설계하였으므로 구조가 매우 간소하며 어셈블러의 전처리기로서 동작하므로 임의의 프로세서에 대해 적용 가능하다.

기능 검증은 대부분 다음과 같은 3단계로 이루어진다.

(1) Compliance Check: 단일 명령어 수준 기능 검증

(2) Corner Check: 다수 명령어 조합 수준 기능 검증

(3) Algorithmic Check: 알고리즘 및 프로그램 수준 기능 검증

본 랜덤 벡터 생성기는 위 3단계를 모두 효과적으로 만족시킬 수 있는 환경을 제공한다. 1 단계의 경우, 오류 유형에 대한 정립이 되지 않은 단계로 명령어 간 상관 관계없이 순수 랜덤 벡터를 생성하며 이후 단계가 높아질수록 정립된 오류 유형에 대해 상관관계를 정의하고 설계자가 관심 갖는 부분에 대해서만 중점적으로 랜덤 벡터를 생성함으로써 효율성을 극대화한다.

IV. 결 론

본 논문은 효율적인 마이크로프로세서의 기능 검증을 위한 바이어스 랜덤 벡터 생성기를 제안한다. 본 랜덤 벡터 생성기는 어셈블리 프로그램의 전처리기로서 동작하며 효율적인 랜덤 벡터 생성을 위한 일련의 전처리기 명령어를 정의하였다. 전처리기 명령어는 설계자가 초점을 두는 부분에 대해서만 중점적으로 랜덤 유형을 생성하게 함으로써 명령어 조합에 일정한 상관관계를 설계자 임의로 제약을 들 수 있고 따라서 순수한 랜덤 벡터에 비해 높은 오류 검출율을 보이는 장점을 지니고 있다. 또한 본 랜덤 벡터 생성기는 하드웨어와 상관없이 어셈블러가 제공되는 임의의 마이크로프로세서에 대한 적용이 가능하다. 따라서 설계 과정 중 많은 비용이 소요되는 기능 검증을 단축시킴으로써 전체적인 설계 비용을 줄일 수 있는 효과가 있다.

참고문헌

[1] Ta-Chung Chang, "A Biased Random Instruction Generation Environment for Architectural Verification of Pipelined Processors", in Journal of Electronic Testing : Theory and Applications 16, pp.13-27, 2000.

[2] Hoon-Mo Yang, Sung-Ho Kwak, Moon-Key Lee, "Efficient Verification Method With Random Vectors for Embedded Control RISC Cores", 대한전자공학회, Vol.38, No.10, pp.65-75, Oct. 2001.

[3] Windley, P.J. "Formal modeling and verification of microprocessors," IEEE Transactions on Computers, Vol44, No1, pp.54-72, Jan. 1995.

[4] M. S. Abadir, J. Ferguson, and T.E. Kirkland, "Logic Design Verification via Test Generation," IEEE Trans. Computer-Aided Design, Vol.7, No.1, pp.138-148, Jan. 1988.

[5] D. C. Lee and D. P. Siewiorek, "Functional Test Generation for Pipelined Computer Implementations.", Proc. Int. Symp. Fault-Tolerant Computing, 1991, pp.60-67.