논문 2009-46SD-8-13

임베디드 시스템에 적합한 듀얼 모드 의사 난수 생성 확장 모듈의 설계

(Dual-mode Pseudorandom Number Generator Extension for Embedded System)

이 석 한*, 허 원*, 이 용 석**

(Suk Han Lee, Won Hur, and Yong Surk Lee)

요 약

난수 생성 함수는 소프트웨어를 사용한 시뮬레이션 테스트나, 통신 프로토콜 검증 등 수많은 어플리케이션에 사용되어진다. 이런 상황에서 난수의 randomness는 사용 어플리케이션에 따라서 다르게 필요할 수 있다. 반드시 randomness가 보장된 랜덤 함수를 통한 고품질의 난수를 생성해야 할 때가 있고, 단지 난수와 비슷한 형태를 가진, randomness가 보장되지 않은 난수가 필요할 때도 있다. 본 논문에서는 고속으로 동작하는 임베디드 시스템을 위한 듀얼 모드로 동작하는 하드웨어 난수 생성기를 제안하였다. 모드 1 에서는 높은 randomness를 가지는 난수를 6사이클마다 한 번씩 생성하게 되며, 모드 2 에서는 낮은 randomness를 가지는 난수를 매 사이클마다 생성할 수 있다. 테스트를 위해, ASIP(Application Specific Instruction set Processor)를 설계하였으며, 각 모드에 맞는 명령어 세트를 설계하였다. ASIP은 LISA언어를 사용하여, 5 stage MIPS architecture를 기반으로 설계되었고, CoWare 사의 Processor Generator를 통해서 HDL코드를 생성하였으며, HDL 모델은 동부 0.18um 공정으로 Synopsys사의 Design Compiler를 통해서 합성되었다. 설계되어진 ASIP으로 난수를 생성한 결과, 하드웨어 모듈을 추가하기 전에 비해 2.0%의 면적 증가 및 239%의 성능 향상을 보였다.

Abstract

Random numbers are used in many sorts of applications. Some applications, like simple software simulation tests, communication protocol verifications, cryptography verification and so forth, need various levels of randomness with various process speeds. In this paper, we propose a fast pseudorandom generator module for embedded systems. The generator module is implemented in hardware which can run in two modes, one of which can generate random numbers with higher randomness but which requires six cycles, the other providing its result within one cycle but with less randomness. An ASIP (Application Specific Instruction set Processor) was designed to implement the proposed pseudorandom generator instruction sets. We designed a processor based on the MIPS architecture,, by using LISA, and have run statistical tests passing the sequence of the Diehard test suite. The HDL models of the processor were generated using CoWare's Processor Designer and synthesized into the Dong-bu 0.18um CMOS cell library using the Synopsys Design Compiler. With the proposed pseudorandom generator module, random number generation performance was 239% faster than software model, but the area increased only 2.0% of the proposed ASIP.

Keywords: pseudo random number generator, ASIP, embedded system

* 학생회원, ** 평생회원, 연세대학교 전기전자공학과
(Department of Electrical and Electronic Engineering, Yonsei University)

※ 본 연구는 IDEC (IC Design Education Center)에 의해 지원되는 EDA툴이 사용되었습니다.
※ 본 연구는 정보통신 진흥원 과제로 한국정보통신연구원(ETRI,2009-8-0569)과 연세대학교 산학협력단의 지원 및 관리로 수행 되었습니다.
접수일자: 2008년9월24일, 수정완료일: 2009년8월3일

Ⅰ. 서 론

최근 컴퓨팅 환경에서의 난수의 용도는 매우 다양하다[1]. RF-ID 에서의 RN16 생성[2]이나, 무선 통신에서의 encryption, 캐시 컨트롤러에서의 random replacement policy[3], 기상 환경 시뮬레이션 등의 과학 분야로부터 슬롯머신 기계 등의 유흥 분야까지 우리 생활에 깊게 관여하고 있다. 이러한 난수를 생성하기 위해서 수많은 연구들이 선행되었다. 하지만 디지털 환경에서 완벽한 난수를 생성하는 것은 하드웨어적으로 많은 오버헤드를 필요로 한다. 이를 이유로, 유한한 수의 집합을 생성하여 난수처럼 사용하는 것을 의사난수 (pseudorandom number)라 한다. 의사 난수 생성기는 처음 seed 값을 사용자로부터 받은 후, 여러 연산을 거쳐 다음 난수 값을 계산하는 형태로 구성된다. 현대의 거의 모든 컴퓨터 알고리즘의 난수는 위와 같이 의사 난수를 난수로 간주하고 사용하고 있다.

의사 난수 생성기의 경우, 소프트웨어적으로 난수 알고리즘을 선언하고, 그를 검증하는 것이 중요하다. 그래서 난수를 각 개체 단위로 분석하지 않고, 난수 전후의 상관관계를 계산하여 얼마나 난수로서의 의미가 있는지를 측정하는 척도를 randomness라 한다. randomness를 테스트하는 기준으로는 여러 논문에서 Diehard randomness test를 사용하고 있다[4]. 16개의 세분화된 테스트로 이루어져 있으며, 결과 값인 p에 따라서 테스트의 통과 여부가 결정된다.

Diehard test를 통해서 구분되어지는 randomness의 중요성이 application마다 다르게 생각되어질 수 있다는 사실을 전제로 하고, 비교적 randomness가 중요하지 않은 application 에서는 계산을 적게 수행하여 임베디드 시스템에서 빠른 속도로 의사 난수를 제공하고, randomness가 critical하게 작용할 수 있는 application 에서는 충분한 iteration을 거쳐서 높은 randomness를 가지는 의사 난수를 제공하는 하드웨어 듀얼 모드 의사 난수 생성기를 제안한다. 그리고 Verilog HDL로 구현된 의사 난수 생성기를 ASIP에 집적하여 소프트웨어에 기반을 둔 이전 알고리즘에 비해 높은 속도로 난수를 제공함을 증명한다.

본 논문에서는 Ⅱ장에서 기존에 제안된 의사 난수 생성 알고리즘을 살펴본다. Ⅲ장에서는 본 논문에서 제시한 의사 난수 생성기의 세부적인 구조에 대해서 설명한다. Ⅳ장에서는 하드웨어로 구현하면서 기본적인 실험 환경을 설정하고, Ⅴ장에서는 설계된 의사 난수 생성기의 성능을 randomness 및 속도 측면에서 평가한다. 그리고 Ⅵ장에서 결론을 맺는다.

Ⅱ. 의사 난수 생성 알고리즘의 개요

현존하는 컴퓨팅 시스템에 있어서, 의사 난수 생성 알고리즘은 여러 수식에 의해서 결정된다. 의사 생성 알고리즘은 수많은 시뮬레이션과 검증을 통해서 이루어지는 귀납적 방식으로 이루어진다[1, 5~9]. 기본적인 개념은 그림 1과 같이 해석될 수 있다.

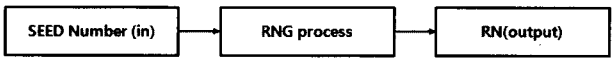

그림 1. 일반적인 의사 난수 생성 알고리즘
Fig. 1. General pseudo random generation algorithm.

이러한 구조의 난수 생성기는 현재 일반적으로 사용하는 난수 생성 함수의 구조이다. 이 난수 생성 알고리즘은 소프트웨어적으로 입력 값으로 들어온 seed값을 기본으로, 여러 프로세스를 거쳐 output값을 뽑아내게 된다. 그리고 식 (1)에서처럼 다음 사이클에 out값이 in 값으로 들어가게 되면서 다음 사용될 난수를 생성하게 된다.

$$\begin{aligned} out &= process(in) \\ in' &= out \\ out &= process(in') \end{aligned} \tag{1}$$

식 (1)의 process는 복잡한 ALU연산을 통해 seed 값을 연산하게 된다. 이 과정에서 많은 cycle이 소모되게 되고, 중간에 계산되어지는 값을 난수로 사용하기는 어렵다. 즉, 난수가 즉시 필요한 경우에도, 전체 process를 수행하는 시간을 대기해야 하는 단점이 존재한다.

Ⅲ. 임베디드 시스템을 위한 의사난수 알고리즘 및 생성기

1. 임베디드 시스템을 위한 의사난수 생성 알고리즘

의사 난수 생성 알고리즘에 대해서는 많은 연구가 진행되어 왔다[1, 5~9]. 그 중에서 하드웨어 구현을 고려한 임베디드 시스템에 맞는 난수 생성 알고리즘을 찾기 위해서는 면적과 속도 및 소모 전력이 가장 중요한 요인이다. 참고 논문 [1]에서는 많은 종류의 임베디드 시스

템을 위한 의사 난수 생성 알고리즘에 대해 다루고 있다. 그 중에서 높은 수준의 의사 난수를 구현할 수 있으며 하드웨어로 구현하였을 때 소모되는 시간이 합리적이며 process의 구조가 간단한 형태를 골라 연구한 결과, 32bit로 동작하는 1-stage counter mode generator를 선택하게 되었다.

$$
\begin{aligned}
&x = k{+}{+};\\
&x = (x \hat{} rot(x,L) \hat{} rot(x,R)) + A;\\
&x = (x \hat{} rot(x,L) \hat{} rot(x,R)) + A;\\
&x = (x \hat{} rot(x,L) \hat{} rot(x,R)) + A;\\
&x = (x \hat{} rot(x,L) \hat{} rot(x,R));\\
&x = (x \hat{} rot(x,L) \hat{} rot(x,R));
\end{aligned}
\tag{2}
$$

식 (2)는 1-stage counter mode 의사 난수 생성식이다. 위 식에서 볼 수 있는 것처럼, 의사 난수 생성 process는 덧셈이 포함된 연산 및 포함되지 않은 연산 2개가 존재하며, 각 process는 간단한 로테이션 시프트 연산 및 덧셈으로 이루어져 있다. 하드웨어를 설계할 때 간단한 iteration으로 설계할 수 있으며, 초기의 x는 카운터 값인 k에 의존하고 최종 결과 값에 의존적이지 않다. 위의 rot(x,L)은 L 만큼 왼쪽으로 x를 로테이션 시프트하는 의미이고, rot(x,R)은 R 만큼 오른쪽으로 x를 로테이션 시프트하는 것이다. 본 알고리즘은 일반적인 시스템에서 소프트웨어 상으로 구현되었을 때, 23cycle 이상이 소요된다[1]. 이는 사용되어진 컴파일러 및 환경에 따라서 달라질 수 있다.

(L,R,A)의 값을 (5, 9, 0x49A805B3)로 입력하고[1] 소프트웨어 시뮬레이션을 돌린 결과, k값이 1씩 증가함에 따라서 x의 값이 32bit xor 연산과 rot연산을 거쳐 덧셈

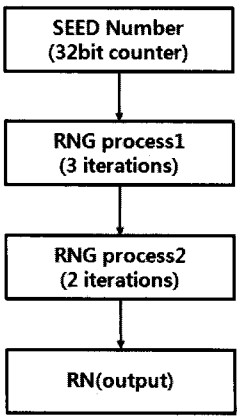

그림 2. 참고 논문에 제안된 의사 난수 생성 알고리즘
Fig. 2. Proposed pseudo random generation algorithm.

기로 들어가게 된다. 그림 2에서와 같이 process1을 3회 거치고, 덧셈을 제외하고는 같은 연산을 수행하는 process2를 2회 수행하여 최종적으로 의사 난수가 생성된다. 1장에서 서술된 이러한 형태의 의사 난수 생성기가 가지는 단점을 해결하기 위해, 각 프로세스가 끝나는 순간의 값도 의미를 갖도록 했다. 그래서 ASIP 내부에 구현이 되었을 때, 프로세스 지연 시간에 관계없이 난수 값을 사용할 수 있다.

2.제안된 하드웨어 의사 난수 생성기의 구조

위에서 설명한, 소프트웨어적으로 많은 이상이 소요되는 형태의 일반적인 난수 생성 함수와 달리, 하드웨어적으로 구현된 난수 생성기는 난수를 생성할 때 소요되는 cycle을 획기적으로 줄일 수 있다. 특히, 난수가 연속적으로 필요하거나, 난수의 생성 패턴 알고리즘이 중요하게 지켜져야 하는 어플리케이션에서 뛰어난 성능을 보일 수 있다.

식 (2)에서 볼 수 있듯이, 제안된 의사 난수 알고리즘의 구조는, 동일한 프로세싱이 6cycle 주기로 계속됨을 관찰할 수 있다. 하드웨어로 구현하였을 때, MUX를 활용하여 면적을 최소화 할 수 있다. 그래서 그림 3과 같

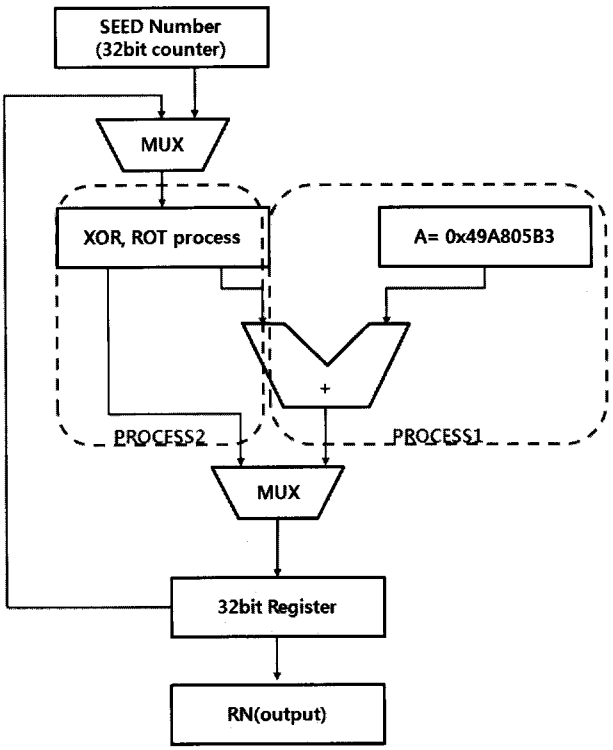

그림 3. 하드웨어로 구현된 의사 난수 생성 알고리즘
Fig. 3. Hardware implemented pseudo random generation algorithm.

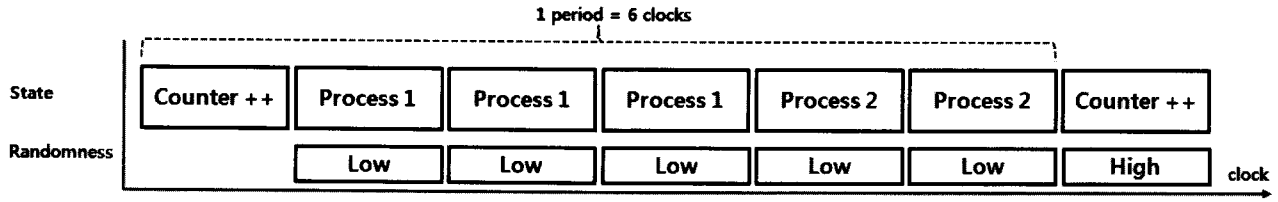

그림 4. 하드웨어 의사 난수 생성기의 스테이트
Fig. 4. State of hardware implemented pseudo random generator.

은 하드웨어 구성을 기대해 볼 수 있다. 그림 3에서는 32비트 카운트 레지스터 1개, 32비트 2:1 MUX 2기, 32비트 xor 및 rotation shift를 수행하는 로직 부분과, 32비트 덧셈기 및 데이터 값을 최종적으로 저장하는 32비트 레지스터가 존재한다. 그림 3의 MUX1은 첫 번째 cycle에서 수행되는 seed값을 1씩 증가시키는 역할을 한다. 이 동작은 난수 값이 필요한 최초에 1회 수행된다. MUX2는 cycle을 보고, 2,3,4번째 cycle때 식 (2)에서 덧셈 부분이 필요한 부분만을 선택하여, addition 연산을 수행하여 준다. 중앙의 xor, rot process 부분은 식 (2)의 $(x \hat{} rot(x,L) \hat{} rot(x,R))$ 부분을 연산하여 준다.

II장에서 언급한대로, 구현된 하드웨어 알고리즘은 매 cycle마다 register에 중간 단계의 의사 난수가 저장이 된다. 중간 단계의 의사 난수이기 때문에, randomness가 떨어질 것이라고 예측하여 볼 수 있다. 하지만 application에 따라서는 randomness가 떨어지더라도 값을 사용하는 것에 영향을 미치지 않는 경우가 있다. 특히 시뮬레이션 내부에서 필요한 난수의 생성이나, RF-ID 내부에서의 난수의 경우에는 단지 이전 숫자와 다른 숫자만 나오면 되는 경우가 있다. 그래서 randomness가 떨어지는 경우에도 사용이 가능하다고 판단했다. 이를 사용하기 위해 2가지의 모드를 구현하여, 의사 난수 생성기 전체 state를 그림 4와 같이 결정하였다. 최종적으로 정해진 사양을 순차 회로로 Verilog HDL을 통해서 구현하였다.

IV. 테스트 환경 설정 및 구현

설계된 하드웨어 연산기의 성능을 측정하기 위해서 ASIP을 설계하였다. 설계된 ASIP은 LISA(Language for Instruction Architecture Set)를 사용하여 구현되었다. LISA는 CoWare사에 의해서 개발되어진 ADL (Architecture Description Language)이다[11]. C 언어에 기반을 둔 LISA언어를 사용해서 CoWare 사의 Processor Designer를 통해, Abstraction Level이 높은 ASIP을 설계할 수 있다[12]. 또한, 생성되는 SDK 를 통해서, assembler, compiler, simulator, linker, debugger를 사용할 수 있다. 그리고 합성이 가능한 HDL Code로 출력이 가능하다. 이를 통해 프로세서를 설계함에 있어서 RTL 설계에 비해, time-to-market 을 비약적으로 감소시킬 수 있다.

표 1. 난수 생성기의 동작을 위해 추가된 명령어
Table 1. Added instruction set for proposed random number generator.

명령어	설명
rand1 <rd>	6 cycle을 소비하여 randomness가 높은 난수를 생성해서 레지스터 <rd>에 저장한다.
rand2 <rd>	1 cycle을 소비하여 randomness가 낮은 난수를 생성해서 레지스터 <rd>에 저장한다.
srand <rs>	seed를 <rs>로부터 가져와, 카운터의 초기값으로 사용한다.

우선 실험을 위해서, 기본이 되는 프로세서를 기본적인 MIPS 구조의 5단 파이프라인(Fetch - Decode - Execute - Memory Operation - Writeback)으로 구현하였다. 기본적인 MIPS 구조의 프로세서가 가지는 기능을 모두 가지면서, 난수 생성기를 위한 명령어를 추가하였다. 추가된 명령어는 표 1과 같다. 결과적으로 execute 단에서 제안된 난수 생성기 extension을 추가하고 실험을 진행하였다.

V. 실 험

본 논문에서 설계한 하드웨어 의사 난수 생성기의 동작을 실험하기 위해서, 우선 C 언어로 알고리즘을 구현한 결과 값과 난수 생성기의 결과 값을 1만개의 test vector를 만들어 seed값을 바꾸면서 비교하였다. 그리고 설계된 ASIP에 집적하였을 때의 결과 값을 비교하였다. 마지막으로, randomness를 비교하기 위해서 앞 장에서 언급된 Diehard test suite를 사용하였다. 그림 5는

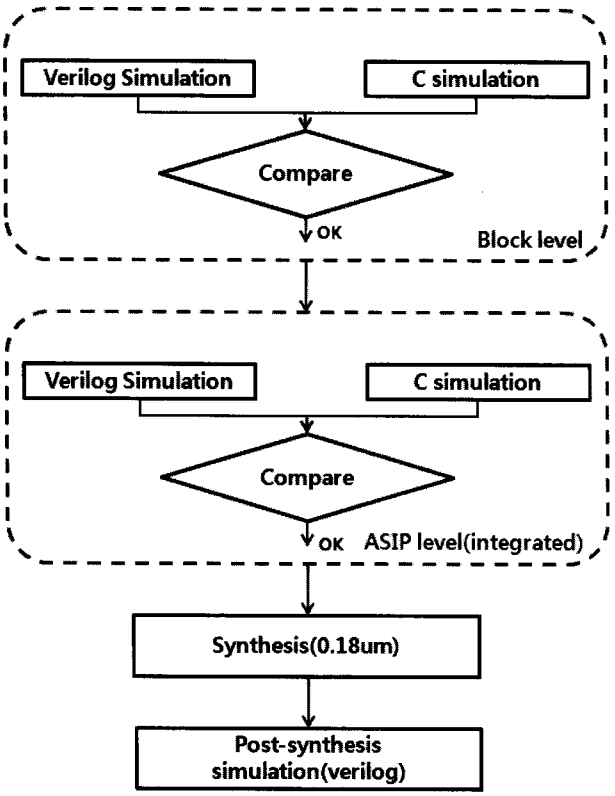

그림 5. 난수 생성기의 설계 흐름도
Fig. 5. Design flow of random generator.

이러한 과정을 설계 흐름도로 나타낸 것이다.

모든 test vector에 대해서 random generator block 및 ASIP으로 집적하였을 때의 결과 값이 모두 동일하였다. Randomness 테스트는 rand1 및 rand2 모드에서의 randomness가 각각 다르므로, 각 모드에서의 결과 값마다 Diehard test suite를 따로 수행하였다. 16개의 일반적인 난수가 적용되었을 때의 결과 값을 분석해서 합격, 불합격의 여부로 randomness를 결정한다. 직접 Diehard test 상에서 합격여부를 결정짓지 않고, 각 테스트별로 결과 값으로 p 값을 출력한다. p값이 0이거나, 0.999이상의 값을 가질 때, 해당 테스트는 실패하였다고 간주한다[1]. 각 테스트의 결과 값은 표 2에 정리하였다. 한 개의 테스트만 실패한 rand1은 비교적 높은 randomness를 가지고 있음이 증명되었다. 반면에, rand2 의 경우에는 올바른 randomness를 가진 난수라고 보기 어렵다.

비록 randomness가 낮은 난수이지만 정밀함을 요하지 않으면서 빠른 속도로 난수를 필요로 하는 어플리케이션에서는 난수를 필요할 때 즉시 사용할 수 있는 장점을 가진다.

설계된 난수 생성기는 그림 5의 설계 흐름도에 따라,

표 2. 각 모드별 Diehard test 결과
Table 2. Result of Diehard test suite.

테스트명	Mode1	Mode 2
birthday Spacing test	합격	불합격
overlapping 5-permutaion test	합격	불합격
binary rank test for 31x31 matrices test	합격	불합격
binary rank test for 32x32 matrices test	합격	불합격
binary rank test for 6x8 matrices test	합격	불합격
bitstream test	불합격	불합격
OPSO, OQSO, and DNA	합격	불합격
COUNT-THE-1's test on a stream of bytes	합격	불합격
COUNT-THE-1's for specific bytes	합격	합격
parking lot test	합격	불합격
minimum distance test	합격	불합격
3DSPHERES test	합격	불합격
SQEEZE test	합격	불합격
OVERLAPPING SUMS test	합격	불합격
RUNS test	합격	불합격
CRAPS test	합격	불합격

synopsys사의 Design Compiler를 통해서 합성하였다. 동부 하이텍 0.18um 공정으로 합성하였을 때, 난수 생성기는 1287게이트(2-input nand)를 차지하며, 최악 조건에서 511Mhz로 동작을 보증하였다. 그리고 설계된 ASIP은 난수 생성기를 포함하지 않고 64002게이트(2-input nand)를 차지하며, 최악 조건에서 125Mhz로 동작을 보증하였다. 결과적으로, 기본 ASIP에 설계된 난수 생성기를 추가하였을 때 65289게이트(2-input nand)를 차지하며, 주파수는 기본 ASIP의 동작주파수인 124.6Mhz로 동작하였다. 합성 후 시뮬레이션을 수행한 결과, 미리 수행한 시뮬레이션 결과 값과 같게 나옴을 확인할 수 있었다. 어셈블러를 통해서 레지스터에 난수를 저장하기까지 걸린 시간은, 표 3과 같다. 모드 1과 모드 2를 비교하면 파이프라인 페널티에 의해서 600% 성능은 확보하지 못한다. 하지만, 연속적인 난수 생성 시에는 600%에 수렴하는 값을 가짐을 시뮬레이션으로 확인할 수 있었다. 난수 생성 지연시간의 경우 소프트웨어적으로 구현한 의사 난수에 비해 약 59% 감소하였고, 모드2의 경우 81% 감소하였음을 확인할 수 있다.

표 3. 난수 생성에 걸리는 지연시간 비교
Table 3. Random number generation latency.

환경	난수 생성 지연시간
기본 ASIP	193ns
ASIP+난수 생성기(모드1)	80.6ns(239% 빠름)
ASIP+난수 생성기(모드2)	40.3ns(479% 빠름)

또한, 소프트웨어 방식은 연산 시에 다른 레지스터들을 점유함으로써, random 함수를 call할 때, stack에 쌓는 과정이 필요하다. 이러한 준비 과정에 의해 소모되는 시간까지 고려한다면, 제안된 의사 난수 생성기가 가지는 이점은 더 크다고 할 수 있다.

VI. 결 론

본 논문에서는 임베디드 시스템에 적합한, ASIP에 집적한 의사 난수 생성기 확장 블록을 설계하였다. 32비트 카운터 레지스터 한 개, 32비트 2:1 MUX 2개, 32비트 xor 및 rotation shift 를 수행하는 로직 부분, 32비트 덧셈기 1개 및 데이터 값을 최종적으로 저장하는 32비트 레지스터 1개를 내장하여, 임베디드 시스템에 적합한 면적과 속도를 가진다. 기존의 ASIP에 쉽게 집적할 수 있도록 하였으며, randomness에 따라서 2개의 모드를 지원함으로써 높은 randomness가 필요한 encryption, 슬롯 머신의 난수 생성기, 기상환경 시뮬레이션 등에 적용할 수 있고, 낮은 randomness의 경우에는 RF-ID통신의 RN16 생성용도로 사용할 수 있다.

의사 난수 생성기는 C 시뮬레이션을 통해서 구조를 확정하고, Verilog HDL을 사용하여 하드웨어를 설계하였다. 그리고 이를 동부 하이텍 0.18um 표준 셀 라이브러리를 통해 합성하였다. 최악 조건에서 511MHz의 동작을 보이고, 6cycle에 한번 높은 수준의 randomness를 가지는 의사 난수를 생성한다. 또한 매 cycle마다 낮은 수준의 randomness를 가지는 의사 난수를 생성한다. 설계된 난수 생성기를 같은 조건에서 124.6MHz로 동작하는 ASIP을 설계하여 집적한 결과, 기본 ASIP 대비 2.0%의 면적 추가로 같은 결과 값을 가지는 어셈블러 코드가 수행되는 시간보다, 난수 생성기를 통해 239% 빠른 속도로 난수를 생성할 수 있었다. 제안된 randomness를 낮게 가지는 모드에서는 479% 빠른 속도로 난수를 생성할 수 있었다. 파이프라인 페널티가 고려된 형태이기 때문에, 더 빠른 속도로 난수를 생성하는 것도 가능하며, 만약 생성하는 난수의 개수가 많을 때에는 600% 빠른 속도로 난수를 생성할 수 있다. 또한 독자적으로 난수를 생성시키기 때문에 프로세스 중의 일반 레지스터 자원을 효율적으로 사용하는 것도 가능하다.

난수를 생성함에 있어서 생성 알고리즘이 여러 가지가 있으므로, 필요한 요소에 따라서 난수 생성기 확장 블록을 바꾸어 줄 수 있는 방법론에 대한 연구가 더 진행된다면, 하드웨어 방식의 의사 난수 생성기의 장점을 더욱 극대화 할 수 있을 것이다.

참 고 문 헌

[1] Laszlo Hars and Gyorgy Petruska, "Pseudorandom Recursions: Small and Fast Pseudorandom Number Generators for Embedded Applications," EURASIP Journal on Embedded Systems, Vol. 2007, No. 1 (2007), pp. 5-5.
[2] ISO/ISE 18000-6C : Parameter for air interface communications at 860Mhz to 960Mhz, 2005.
[3] Willick, D. L. et al, ▯ Disk Cache Replacement Policies for Network Fileservers,▯ Distributed Computing Systems, 1993 Int'l Conf. pp. 2-11.
[4] G. Marsaglia, "DIEHARD: a battery of tests of randomness,"1996, http://stat.fsu.edu/pub/diehard/.
[5] G. Fishmann and L. R. Moore III, "An exhaustive analysis of multiplicative congruential random number generators with modulus ," SIAM Journal of Scientific and tatistical Computing, Vol. 7, No. 1 (1985),pp. 24-45.
[6] F. James, "A review of pseudorandom number generators,"in Computer Physics Communication, North Holland, Amsterdam, The Netherlands, 1990, Vol. 60, pp.329-344.
[7] P. L'Ecuyer, "Maximally equidistributed combined Tausworthe generators," Mathematics of Computation, Vol. 65, No. 213 (1996), pp. 203-213.
[8] S. K. Park and K. W. Miller, "Random number generators: good ones are hard to find," Communications of the ACM, Vol. 31, No. 10 (1988), pp. 1192-1201.
[9] S. L. Anderson, "Random number generators on vector super computers and other advanced architectures," SIAM Review, Vol. 32, No. 2 (1990), pp. 221-251.
[10] R. C. Tausworthe, "Random numbers generated by linear recurrence module two," Mathematics of

Computation, Vol. 19, No. 90 (1965), pp. 201-209.
[11] CoWare Inc., "LISA 2.0 Language Tutorial," http://www.coware.com
[12] T. Glokler and H. Meyr, "Design of Energy-Efficient Application Specific Instruction-Set Processors (ASIPs)," Kluwer Academic Publishers (2003), pp. 117-143.

저 자 소 개

이 석 한(학생회원)
2007년 연세대학교 전기전자 공학과 학사 졸업.
2009년 연세대학교 전기전자 공학과 석사 졸업.
2009년~현재 삼성테크윈 재직
<주관심분야 : 영상처리, 신호처리, SoC, ASIC>

허 원(학생회원)
2007년 2월 경희대학교 전자공학과 학사
2009년 8월 연세대학교 전기전자 공학과 석사
2009년 8월~ 현재 디지털 스트림 테크놀로지
<주관심분야 : 마이크로프로세서, ASIC, SoC>

이 용 석(정회원)
1973년 2월 연세대학교 전기공학과 학사
1977년 2월 University of Michigan, Ann Arbor 석사
1981년 2월 University of Michigan, Ann Arbor 박사
1993년~현재 연세대학교 전기전자공학과 교수
<주관심분야 : 마이크로프로세서, 네트워크 프로세서, 암호화 프로세서, SoC>