응용통계연구 제 16권 2호, 2003년, pp. 427-440

저전력을 소모하는 난수발생기의 성능 평가

윤정민 [1] 김지홍 [2] 김진효 [3]

요 약

휴대전화, PDA와 같은 이동 단말기와 무선 통신의 발전으로 인하여, 이동 단말기를 이용한 전자 메일, 게임, 주식거래 등이 가능하게 되었다. 무선 단말기를 통한 주식거래나 게임 등을 위하여서는 난수발생기(Random Number Generator)의 사용이 필수적이다. 그런데 최근까지의 난수발생기는 우수한 난수성에 중점을 두어 개발되었으며, 이동 단말기에서의 에너지 소비량에 대한 연구는 없었다. 이동 단말기는 무게 및 크기의 한계 때문에 배터리의 용량에 제한이 있게되므로, 되도록 에너지 소비량을 줄여서 주어진 배터리를 오랫 동안 사용하기를 원하게 된다. 본 논문에서는 이동 단말기에서 많이 사용되는 여러 난수발생기들을 살펴보고, 저전력 에너지 측정도구인 SES(SNU Energy Scanner)를 이용하여 각 난수발생기의 에너지 소비량을 측정하여 이들을 비교한다. 이를 바탕으로 이동 단말기 환경에서 저전력을 소모하는 난수발생기를 제안하였다.

주요용어: 이동 단말기, 저전력, 난수발생기

1. 서론

최근의 컴퓨터와 통신 기술의 급속한 발달과 더불어, 들고 다닐수 있는 작고 가벼운 이동 단말기들이 놀라운 속도로 증가하고 있다. 이동 단말기들과 무선 통신 기술의 발달에 의하여 휴대 전화기나 PDA등을 통하여 인터넷 쇼핑몰에서 쇼핑을 하거나 계좌 이체 및 전자 메일을 보내고 받는 등의 일은 흔한 일이 되어 버렸다. 많은 사람들이 PDA나 휴대 전화기를 이용하여 전자 메일을 보내고 받거나, 주식매매 등을 한 경험이 있을 것이다. 이러한 과정을 자세히 기억해보면, 그러한 활동의 중간에는 보안을 위하여 인증(authentication)을 받는 단계가 반드시 있었을 것이다. 이러한 인증단계에서, 일반적으로 비밀번호의 안전한 전송을 위하여 암호화 방법을 사용한다. 우리가 알고 있듯이 이러한 암호화 과정에서는 난수발생기가 필요하게 되고, 이러한 난수발생기가 암호화의 중요한 부분을 차지한다. 그러므로 우리가 이동 단말기에서 주식매매, 계좌이체 등의 활동을 하기 위해서 난수발생기의 사용은 불가피하다. 또한 이동 단말기를 이용하여 게임을 하는 경우에도 게임 프로그램에서는 어느 특별한 상황에서 결정을 내리기 위하여 난수발생기를 사용한다. 이렇듯 우리가

1) (151-742) 서울시 관악구 신림9동 산 56-1번지 서울대학교 전기 · 컴퓨터 공학부, 석사과정졸업
E-mail : twingo@davinci.snu.ac.kr
2) (151-742) 서울시 관악구 신림9동 산 56-1번지 서울대학교 전기 · 컴퓨터 공학부, 부교수
E-mail : jihong@davinci.snu.ac.kr
3) (690-756) 제주도 제주시 아라1동 1번지 제주대학교 전산통계학과, 조교수
E-mail : jinkim@cheju.ac.kr

의식하지 못하는 사이에 난수발생기는 이동 단말기의 많은 프로그램에서 사용되어 지고 있다. 그런데 이동 단말기에서는 이전보다 훨씬 작고 가벼운 배터리가 필요하며, 이렇게 작은 배터리에 의존하는 이동 단말기의 사용시간을 늘리기 위해서는 배터리를 효율적으로 사용하여야 한다. 이를 위하여 이동 단말기에서 많이 사용되는 난수발생기도 전력 소모가 적은 것의 선택이 필요하다.

배터리 사용시간을 늘리는 방법은 크게 두 가지 방법으로 나누어 질 수 있다. 하나는 배터리 자체의 용량을 늘리는 방법이고, 다른 하나는 이동 단말기에서 실행되는 프로그램들의 에너지 소비량을 줄이는 방법이다. 그런데 배터리 용량을 늘리는 기술은 지난 수십년 동안 우리의 요구에 부응할 만큼의 빠른 발전 속도를 보이지 못했다. 그러므로 이동 단말기에서 배터리 사용시간을 늘리기 위해서는 에너지 소비량을 줄이는 연구의 필요성이 최근에 높아졌다.

이동 단말기의 에너지 소비량을 줄이는 방법은 하드웨어적인 접근 방법과 소프트웨어적인 접근 방법의 두 가지 접근 방법이 있다. 하드웨어적인 접근방법의 대표적인 예는 어떤 기기가 낮은 공급 전압에서 동작하도록 설계하는 것이 될 수 있다. 이는 어떤 기기에서의 전력 소비는 이 기기에 공급되는 공급 전압에 비례하기 때문이다. 또 다른 예로써 회로에서 데이타 패스를 이루는 전선과 전기 라인의 길이를 줄이는 것을 들 수 있다. 왜냐하면 전기 저항은 길이가 길면 커지게 되고, 저항이 크면 에너지 손실이 크기 때문이다. 이와는 다르게 소프트웨어적인 접근 방법을 이용하여 전력 소비량을 줄일 수 있는 방법들이 새롭게 연구되어 지고 있다. (Tiwari, 1994, Laramie, 1990) 소프트웨어적인 접근 방법의 대표적인 예는 전기 회로에서 일어나는 이진수 값의 변화를 의미하는 스위칭 활동이 줄어들도록 소프트웨어를 재구성하는 방법을 들 수 있다. 디지털 시스템 환경에서 모든 값들은 0과 1로 표현되며, 이렇게 표현되어 저장된 값들은 새로 만들어 지고 갱신되고 삭제되는 과정에서 0과 1로 표현된 값들의 각각의 자리에서 0은 1로 1은 0으로 바뀌는 스위칭 활동이 일어나게 된다. 디지털 시스템에서는 이러한 이진수 값을 0은 낮은 전압으로 1은 높은 전압으로 나타내기 때문에 회로는 충전과 방전을 반복하게 되며, 이러한 과정에서 에너지가 소비된다. 이렇게 스위칭 활동을 줄이는 연구의 대표적인 연구가 프로그램의 올바른 수행을 보장하는 범위 내에서 CPU의 기본 연산 단위인 명령어의 순서를 바꾸어 스위칭 활동이 줄어들도록 하는 연구이다. 명령어의 순서를 스위칭이 적게 일어나도록 바꾸어 이를 통하여 에너지 소비를 줄이는 것이다. 이 방법은 컴파일러 및 간단한 도구의 도움으로 훨씬 싸고 쉽게 프로그램의 소비전력을 낮출 수 있다.

이렇게 하나의 프로그램에서 소비전력을 낮추는 방법이외에도 여러 가지 가능한 프로그램을 개발하여 선택한 후 그 중에서 에너지 소비량이 가장 적은 소프트웨어를 선택하여 사용함으로 좀 더 긴 배터리 사용시간을 보장할 수 있는 방법도 가능하다. 말하자면 같은 기능을 하는 몇 개의 알고리즘 중에서 가장 에너지 소비량이 적은 알고리즘을 선택하면 에너지 소비량이 줄어드므로 배터리 사용시간이 길어진다.

본 논문에서는 여러가지 난수발생기 중에서 잘 알려져 있고 이동 단말기에서의 구현 및 사용이 적합한 난수발생기들을 선택하였다. 이렇게 선택된 난수발생기들은 여러 면에서 분석이 잘 되어 있고, 널리 사용되어 지고 있으므로 중요하다고 할 수 있다. 컴퓨터 구현을 위

한 난수발생기는 Marsaglia(1985)에서 찾을 수 있는데, 그 당시에는 이동 단말기의 환경이 일반적이지 않았기 때문에 에너지 소비량에 대한 언급은 없었다. 이동 단말기가 일반적인 환경으로 자리잡은 현재의 환경에서는 난수발생기들의 에너지 소비량에 대한 고찰이 필요하다고 본다. 본 논문에서는 많이 사용되고 잘 알려진 5개의 난수발생기를 선택하여 이들의 장단점을 간단히 살펴보고, 이들이 하나의 난수를 만들어 내는데 소비하는 에너지의 양을 비교하고, 이와 더불어 각 난수발생기의 중요한 성능이라고 할 수 있는 난수성을 알아보았다. 2절에서는 5개의 선택된 난수발생기들의 상대적인 장단점을 비교하고, 3절에서는 각 난수발생기의 에너지 소비량의 측정방법을 서술하며, 4절에서는 각 실험의 결과를 비교한다. 그리고 5절에서는 난수발생기 성능 측정의 중요한 요소인 난수성(randomness)을 알아보기 위해 Kolmogorov-Smirnov 검정과 Pearson's Chi-square 검정을 하였다. 여기서 얻은 결론을 6절에 모아 보았다.

2. 선택된 난수발생기들

여러가지 의사(擬似) 난수발생기(pseudo Random Number Generator, 앞으로 본 논문에서는 난수 발생기를 RNG로 표기한다)중에서 많이 쓰이는 5가지를 선택하였고, 이러한 의사 난수발생기중 Linear Congruential Generator, Minimal Standard Generator, Shuffle Generator는 von Neumann이 제안한 다음의 재귀적인(recursive) 알고리즘을 사용한다. (Anderson, 1990, Gentle, 1998, Press, 1993)

$$I_{j+1} = (aI_j + c) \bmod m \tag{2.1}$$

여기에서 I_j, a, c, m은 각각 j번째 생성된 난수, 승수(multiplier), 가수(increment) 그리고 법수(modulus)를 나타낸다. 법수연산(mod)은 0과 $m-1$ 사이의 일양분포를 갖는 숫자를 발생시켜 준다. 식 (2.1)을 기반으로 만들어진 다섯 개의 난수발생기를 살펴보면 다음과 같다.

Linear Congruential Generator(LCG)는 ANSI C 표준으로 제정되어 있을뿐만 아니라 많은 컴퓨터에서 사용되고 있는 난수발생기이다. ANSI C 표준 협의회에서는 이 발생기의 반환 값을 정수형으로 정의하였다. 그런데 일반적으로 거의 모든 컴퓨터에서 정수값은 상대적으로 적기 때문에, LCG는 긴 난수 순서를 가지기 못하는 단점과 또한 각각의 컴퓨터에서 사용되는 정수형의 바이트 크기가 다르기 때문에 컴퓨터 머신에 따라 다른 결과를 가져올 수 있는 단점을 지녔다. 예를 들어 ARM 프로세스(3절참조)와 같이 32bit 기반의 프로세스를 사용하는 컴퓨터에서는 식 (2.1)에 상수로서 $a = 11035152454$, $c = 12345$, and $m = 2^{31}$를 사용한다. 이 난수발생기의 장, 단점에 대하여서는 표 (2.1)에 나타나 있다. 이 난수발생기가 난수 순서가 길지 못하다는 단점을 가지고 있음에도 불구하고, 빠른 연산 속도 때문에 널리 사용되어 지고 있다. 실제로 많은 이동형 단말기의 소프트웨어를 개발하는 개발자 도구들은 GNU의 소프트웨어 툴들을 기반으로 하고 있으며, 이 GNU 소프트웨어 툴에서 사용하고 있는 기본적인 난수발생기도 이 LCG이다. 그래서 많은 소프트웨어 개발자들이 특별한 용도의 난수발생기가 필요하지 않은 한 이 LCG를 그대로 사용하고 있어 가장 많이 사용되고 있는 난수발생기라고 할 수 있다.

Park and Miller(1988)는 지난 30년 동안 사용되어진 여러가지 난수발생기에 대하여 연구한 후, 식 (2.1)에서 $a = 7^5 = 16807$, $m = 2^{31} - 1 = 2147483647$, $c = 0$의 상수 값을 가지는 'Minimal Standard Generator'(MSG)를 제안하였다. 이러한 MSG는 위의 LCG의 컴퓨터마다 사용하는 정수형의 byte 수에 따라 정수형의 최대값이 달라지는 의존성과 짧은 난수 발생 주기의 단점을 보완하여 준다. 그렇기 때문에 LCG에서 발생된 난수들은 같은 프로그램이라 하더라도 다른 컴퓨터에서는 다른 난수의 순서를 발생시키게 되어 이식성(portability)이 떨어지는 문제가 있으며, 이러한 약점이 MSG를 통하여 극복될 수 있다. MSG도 식 (2.1)을 기반으로 하는 식을 사용하고 있고, 차이는 $c = 0$인 상수를 사용한다. 또한 식 (2.1)을 사용하여 MSG를 구현하면 오버 플로우의 문제는 발생하는데, 이러한 오버 플로우를 피하기 위하여 m의 대략적인 factorization에 기반한 Schrage의 알고리즘이 제안되었다. (Schrage, 1979) Schrage의 난수의 주기는 $2^{31} - 2 \approx 2.1 \times 10^9$로 알려져 있다. 표 (2.1)에 MSG의 장, 단점에 대하여 요약되어 있다.

Shuffle Generator(SG)는 MSG의 단점으로 지적되는 낮은 차수 비트들이 가지는 직렬 자기상관(serial autocorrelation)을 극복하려는 노력으로 제안되었다. 이 난수발생기는 MSG와 같은 식을 사용하지만, 이 난수발생기는 새로 생성된 난수를 다음 단계에서 바로 사용하지 않는다. 새롭게 생성된 난수를 사용하는 대신, 그 이전에 생성된 난수들을 저장하고 있는 배열에서 랜덤하게 선택하여 이를 사용하게 된다. 새롭게 생성된 난수는 사용된 난수의 자리에 채워지게 된다. 이 알고리즘을 사용하면, 일반적으로 j번째에 생성된 난수는 $j+32$번째에 사용되어지게 된다(물론 생성된 난수가 사용되어 지는 시기는 난수를 저장하고 있는 배열의 크기에 따라 달라질 수 있다.). MSG의 단점인 낮은 차수 비트들의 직렬 자기상관을 SG는 극복할 수 있다. 그러나 이전에 생성된 32개의 난수들을 저장할 여분의 메모리 공간이 필요하게 된다. 이는 메모리의 제약이 많은 이동 단말기에서는 단점이 될 수 있다. 식 (2.1)을 기반으로 한 난수발생기 중에서 좀 더 긴 난수 순서와 난수성을 높이기 위하여, 본 소고에서 언급된 2, 3개의 난수발생기를 섞어서 새로운 난수발생기를 만들기도 한다. (L'Ecuyer, 1988) 이렇게 서로 다르게 생성되는 두 개의 난수 순서를 가지고 SG에서와 마찬가지로 하나의 더 큰 배열에 합쳐 넣어 난수 순서를 만들어 낼 수 있다. 보통 이와 같이 여러 난수발생기를 섞어서 사용하는 경우에는 좀 더 긴 난수 주기를 만들어 내기 위하여 사용되어 진다.

피보나치(Fibonacci) 수열을 사용하여 난수를 발생시킬 수도 있다. 일반적인 피보나치 수열에서처럼 2개의 연속적인 항을 묶는 대신에 좀 더 나은 난수성을 위하여 약간 어느 정도의 거리가 떨어진 두개의 항을 묶는다. Lagged Fibonacci Generator(LFG)는 다음과 같은 재귀 알고리즘으로 표현된다.

$$I_j = I_{j-p} \odot I_{j-q} \tag{2.2}$$

위의 식 (2.2)에서 $\odot$는 $+$, $-$, $\times$, $\otimes$(배타적 논리곱)과 같은 이항 연산자를 의미하며, 이에 따라 ALFG, SLFG, MLFG, XLFG로 구분된다. 그러므로 이러한 이항 연산자에 따른 각각의 난수발생기가 존재할 수 있다. 일반적으로 LFG가 난수를 발생하는 방법은 다음과 같다. 56개 정도의 초기 난수 순서를 만들어 하나의 배열로 저장한다. 이렇게 저장된 난수 배열에서 현재의 위치를 나타내는 지표에서 어느 정도 거리가 떨어진 두 개의 난수를 고른다. 그

	장 점	단 점
LCG	연산을 위하여 많은 저장공간을 필요로 하지 않음	발생되는 난수가 짧은 주기를 갖음
	호출시마다 적은 연산이 필요	낮은 차수의 비트에서 난수성이 떨어짐
	빠르게 생성됨	
MSG	기계에 비의존적 (machine independent)	잘못된 a,m의 선택으로 동일한 난수들이 생성될 수 있음
	긴 난수 주기	이전 단계에서 생성된 난수를 저장
	메모리 요구량이 적음	serial correlation이 있을 가능성
SG	MSG보다 serial correlation이 적음	32개의 난수를 저장하기 위하여 여분의 32개의 저장장소가 필요
	난수 순서의 긴 주기성	초기값의 생성 비용이 큼
LFG	lag 변화에 따른 난수 주기성이 좋음	이전 단계에서 생성된 난수들을 저장하기 위한 많은 메모리가 필요
	성능이 좋은 난수발생기	기계 의존적인 난수 생성 속도
SRG	상대적으로 빠른 생성 속도	비트들이 오버랩되어 난수성이 떨어짐
	많은 양의 메모리가 필요하지 않음	
	하드웨어 구현이 쉬움	

표 2.1: <선택된 난수발생기들의 비교>

리고 이 두개의 난수를 가지고 해당되는 피보나치 이항 연산을 실시한다. 이렇게 하여 새롭게 생성된 난수를 사용하고, 사용된 난수는 현재의 색인(index)이 나타내고 있는 배열의 위치에 저장한다. 그리고 색인의 값을 증가시키며, 이러한 과정을 LFG는 반복한다. 이러한 방법으로 작동되는 LFG의 장, 단점은 표 (2.1)에 기술되어 있다. 일반적으로 위에서 언급된 4개의 LFG중에서 $\times$를 이용한 난수발생기가 가장 좋은 난수성을 보인다고 알려져 있다.

마지막으로 LFG의 특별한 형태인 Shift Register Generator(SRG)가 있는데 이 난수발생기는 난수를 나타내는 이진수 표기에서 최하위 비트와 그 왼쪽 옆에 있는 비트를 가지고 배타적 논리곱(exclusive-OR)을 실시한다. 그리고 이진수로 표기된 원래의 난수를 오른쪽으로 한 비트씩 이동시킨다. 이렇게 이동된 숫자의 이진 표기의 최상위 bit에 위에서 배타적 논리곱으로 만들어진 비트를 채워 넣는다(그림 (2.1) 참조). n개의 비트를 가지고 이 알고리즘을 사용한다면 우리는 $2^n - 1$개의 주기를 가지는 난수 순서를 만들어 낼 수 있다. 이 난수발생기의 장단점은 표 (2.1)에 나타나 있다.

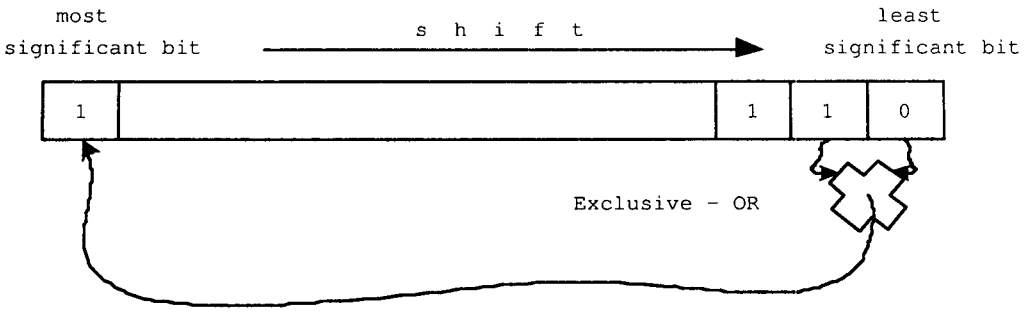

그림 2.1: SRG의 동작방법

3. 난수발생기의 에너지 소비량 측정

일반적으로 난수 1개를 만들어 내는데 걸리는 시간이 적은 난수발생기일수록 사용하는 에너지 소비량이 적을 것으로 보인다. 이런 생각은 일반적으로 긴 수행시간(execution time)은 좀 더 많은 일을 했다는 뜻이고, 이는 곧 많은 에너지 소비를 의미하게 되므로 어느 정도 맞다고 할 수 있다. 그러나 만약에 두개의 프로그램의 수행시간이 같은 상황이라면, 이 상황에서도 두 프로그램의 에너지 소비량이 같을 것인가? 이 물음에 대한 대답은 명백한 '아니다' 이다. 이것은 각 명령어의 소비전력 차이가 다양하기 때문에 수행시간이 같더라도 다른 에너지 소비량을 보일 수 있다. (Laramie, 1999, Tiwari, 1994) 어떤 명령어들은 다른 종류의 명령어들 보다 훨씬 많은 전력을 소비한다. 만약 수행시간이 같더라도 전력 소비가 다르다면, 전력 소비량과 수행시간의 곱으로 나타내어지는 에너지 소비량은 다른 값을 나타내게 된다. 그러므로 에너지 소비량을 판단하는 기준으로 수행 시간을 잡는 것은 올바른 기준이 아니다. 극단적인 경우에는 수행시간이 반 밖에 되지 않더라도 에너지 소비량은 2배가 될수 있다. 이러한 에너지 소비량의 경향은 그 프로그램에서 사용된 명령어의 특징에 의존하고 있다. 대부분의 컴퓨터 시스템에서의 명령어는 그들의 특징에 따라 크게 3 그룹으로 분류되며, 이 세 그룹은 ALU(Arithmetic Logic Unit), Branch, Memory Access이다. Branch 그룹은 프로그램의 조건적인 반복문, 이동 등에 대한 명령어들의 집합이다. ALU 그룹은 계산에 관계된 명령어들의 그룹이다. 마지막으로 Memory Access 그룹은 프로세서와 메모리 사이에서 자료를 저장하거나 읽어 오는 것에 연관된 명령어들의 그룹이다. 이 3 그룹을 전력 소비 측면에서 보면 서로 다른 전력 소비 경향을 보인다. 일반적으로 Memory Access 그룹의 명령어들이 다른 그룹의 명령어들보다 많은 양의 전력을 소비한다. (Laramie, 1999, Tiwari, 1994) 그러므로 각 난수발생기에서 하나의 난수를 생성해 내는데 필요한 에너지 소비량을 정확히 분석하기 위해서는 명령어 수준(instruction level)의 분석이 필요하다.

각 난수발생기의 명령어 수준에서의 에너지 소비량 분석을 위하여, 두 가지 접근 방법

이 시도될 수 있다. 하나의 방법은 시뮬레이션을 이용하는 방법이고, 다른 하나의 방법은 주어진 난수발생기를 실행시키면서 소비하는 전력을 실제로 측정을 하는 방법이다. '시뮬레이션 방법'은 실험의 결과에 영향을 미칠 수 있는 여러 가지 요소들을 조절할 수 있는 장점이 있으나, 시뮬레이션에 사용된 자료값이 정확하지 않다면 결과를 신뢰할 수 없는 단점이 있다. 한편 '실제측정방법'은 측정된 값은 신뢰할 수 있으나 결과에 영향을 미칠 수 있는 요인들의 정확한 통제가 어렵고 명령어 수준의 분석을 하기에는 어려운 단점이 있었서 명령어 수준의 전력소비에서는 많이 사용되지 않았다. 본 논문에서는 명령어 수준의 소비 전력을 실측하기 위하여 개발된 도구인 SES(SNU Energy Scanner) 보드를 이용하여 각 난수발생기의 에너지 소비량을 명령어 수준에서 측정하였다. (Shin, 2002) 한편 실측의 결과에 크게 영향을 주는 요인의 하나는 프로그램이 실행되는 프로세서의 종류이다. 다른 프로세서 사이에서는 같은 프로그램이라고 하더라도 다른 소비전력의 결과가 나올 수 있기 때문에 어떤 프로세서에서 실측을 할 것인가를 결정하는 것이 중요하다. 현재 Nitendo, Sega와 같은 이동 단말 오락기, PDA, 그리고 디지털 카메라등에서 가장 널리 사용되고 있는 프로세서는 ARM 7TDMI 프로세서이므로, (ARM Ltd.) 이동 단말기에서 난수발생기의 에너지 소비량을 실측하기 위해서 ARM 7TDMI에서의 소비전력을 측정하는 것이 결과의 신뢰성을 높일 수 있다. ARM 7TDMI 프로세서는 컴퓨터를 위한 인텔의 펜티엄 프로세서와는 달리 핸드폰, PDA와 같은 내장형 시스템을 위하여 설계된 32비트 범용 프로세서로서 현재 개발되는 있는 많은 수의 내장형 시스템에 사용되어 지고 있는 프로세서이다. SES 보드는 원래 내장형 시스템의 정밀한 에너지 소비량을 모니터링 하기 위하여 고안된 측정 도구로서 이를 이용한 다양한 내장형 시스템 환경의 에너지 측정이 실시되었다. (Shin, 2002) SES 보드는 이 ARM 7TDMI를 바탕으로 하여 구성된 에너지 측정 도구며, SES를 이용하여 각 난수발생기의 에너지 소비량을 측정하였다. SES는 ARM 7TDMI를 프로세서로 탑재하고 있으며, 이 프로세서에서 수행되는 프로그램의 전력 소비를 각 싸이클 별로 정확하게 측정하게 된다. SES 보드는 PCI 인터페이스를 통하여 RedHat Linux 6.2로 운영되는 호스트 PC와 연결하였다. 이 호스트 PC를 통하여 SES 보드의 실행, 정지등을 통제할 수 있고, 측정된 각 싸이클의 전력 소비량을 호스트 PC에 화일로 저장하였다. 각 싸이클 별의 전력 소비량, 그 싸이클에 수행된 명령어 및 프로세서의 상태 등의 결과를 볼 수 있다. 위와 같은 방법으로 우리는 각 난수발생기가 하나의 난수를 발생시키는데 필요한 에너지 소비량을 계산하였다.

SES 보드를 이용하여 각 난수발생기의 에너지 소비량을 측정하기 위하여 다음의 순서에 의하여 실험을 실시하였다. 우선 우리는 위에서 언급된 각 난수발생기를 C언어를 이용하여 구현하였다. (윤정민, 2002) 작성된 C언어를 실행 가능한 바이너리 파일로 만들어 주는 compiler는 SES가 리눅스 호스트에서 작동하는 측정도구이므로 GNU compiler인 gcc를 이용하였다. 일반적으로 실제 프로그램에서 사용되는 난수발생기들은 좀 더 난수성을 높이고 또한 좀 더 긴 난수의 순서를 만들기 위하여 좀 더 복잡하고 많은 연산을 실시한다. 그러나 우리는 기본적인 각 난수발생기의 에너지 소비량을 알아보기 위한 것이므로 되도록 간단하게 난수발생기를 작성하였다. 다음 단계로는 각 난수발생기에서 소비되는 에너지를 측정하기 위하여 SES 보드에서 실행가능한 실행파일을 만들었다. 이 과정에서 우리는 난

수의 순서를 만들어 내는 프로그램을 두 개의 구성 요소로 분리하여 생각하였다. 일반적으로 난수발생기를 사용할 때 초기값을 가지고 난수의 순서를 만들어 내므로, 초기값을 만드는 과정과 주어진 초기값을 이용하여 난수를 만들어 내는 과정의 두 구성 요소로 분리하였다. (본 소고에서는 난수발생기의 초기값을 만들어 내는 함수는 srand()로, 주어진 초기값을 이용하여 난수를 만들어 내는 함수는 rand()로 표기되었다.) 어떤 난수발생기는 초기값이 하나만 필요하기도 하고, 어떤 난수발생기는 여러개의 초기값들이 필요하기도 하다. 이렇게 다양한 갯수를 가지게 되는 초기값을 만드는 srand()의 에너지 소비량이 우리가 만든 난수 발생 프로그램에 포함되어 있기 때문에, 정확하게 하나의 난수를 만드는데 필요한 에너지 소비량을 측정하려면 이 srand()의 에너지 소비량을 빼줘야 한다. 그래서 srand()만으로 이루어진 실행파일을 만들고 이를 이용하여 초기값을 만드는 데 필요한 에너지 소비량을 구했다. 그리고 srand()와 난수를 만들어 내는 rand()로 이루어진 실행파일을 이용하여 난수를 만드는데 필요한 에너지 소비량을 측정하였다. 그런데 각 함수의 에너지 소비량은 함수의 인자 및 각 함수가 불린 환경에 따라 각기 다른 값을 가질 수 있기 때문에 이에 의한 영향을 줄이기 위하여 srand()와 rand()를 1000번 연속하여 부르고, 이 1000번 호출의 에너지 소비량 값을 1000으로 나누어 이를 srand()와 rand()의 에너지 소비량으로 결정하는 방법을 사용하였다. 이러한 방법을 사용하여 환경에 따라 다른 값을 보이는 srand()와 rand()의 에너지 소비량의 평균적인 값을 측정할 수 있다.

4. 각 난수발생기의 에너지 소비량 측정 결과

위에서 언급하였듯이 실험은 초기값을 만드는데 사용되어지는 함수인 srand()의 에너지 소비량과 난수발생기에서 난수를 만드는 필요한 함수인 rand()의 에너지 소비량을 측정하는 두 가지로 나누어 실시 되었다. 먼저 초기값을 만드는 데 필요한 에너지 소비량을 측정하였다. 일반적으로 우리가 사용하는 난수발생기에서 초기값을 사용자가 입력하는 경우도 있기는 하지만, 대부분의 경우는 머신들의 현재 시간을 초기값으로 사용한다. (실제로 많은 컴퓨터에서 사용되는 난수발생기들이 현재 시간에서 얻은 값을 바로 초기값으로 사용한다.) 그러나 일부 컴퓨터에서는 좀 더 질좋은 난수를 얻기 위하여 현재 시간으로 부터 얻은 값을 다른 연산으로 변환하여 초기값을 만들어 사용하기도 한다. 이렇게 서로 다른 초기값 생성 방법은 서로 다른 에너지 소비량을 나타내게 되므로, 이를 체계적으로 분석하기 위하여 많이 사용하는 초기값 생성 방법을 3 그룹으로 분리하였다. 첫번째 그룹은 시간으로부터 얻은 값을 그대로 사용하는 'Direct 그룹'이고, 두번째 그룹은 시간으로부터 얻어진 몇개의 값들의 각 비트에 논리적 배타 곱을 실시하는 'Bit Masking 그룹'이고, 마지막 세번재 그룹은 Bit Masking으로 부터 얻은 값들을 LCG 알고리즘에 의하여 난수화 시켜 초기값으로 사용하는 'Randomizing 그룹'이다. 'Randomizing 그룹'에서 초기값을 얻기 위하여 LCG 외에 다른 난수발생 알고리즘을 사용할 수도 있다. 그러나 각 난수 발생기의 소비 전력 측정 실험결과에서 각 난수발생기의 상대적 에너지 소비량을 알 수 있으므로, 'Randomizing 그룹'에서 다른 난수발생 알고리즘을 사용하였을 경우에 상대적인 에너지 소비량의 유추가 가능하기 때문에 본 논문에서는 LCG만으로 실험하였다. 또한 어떤 난수발생기들은 하

Method	Direct	Bit masking	Randomizing
Energy (pJ)	19321.92	35673.78	65411.45

표 4.1: 초기값 생성의 에너지 소비량

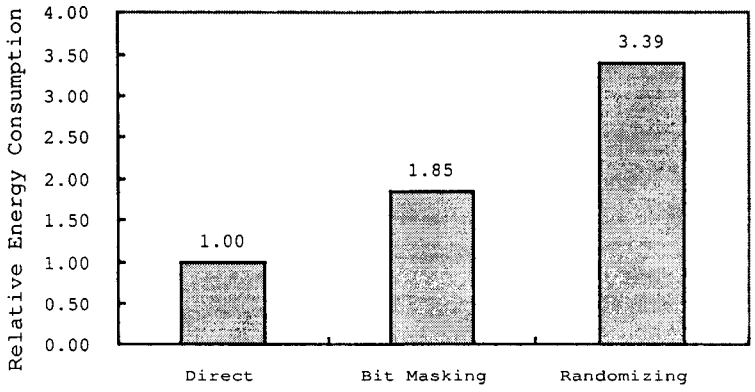

그림 4.1: 초기값 생성의 상대적 에너지 소비량

나의 초기값만이 필요한 반면, 다른 난수발생기들은 여러개의 초기값들이 필요함을 유의해야 한다. 이러한 서로 다른 개수의 초기값을 만드는데 필요한 에너지 소비량을 비교하는 것은 의미가 없으므로 이 실험에서는 난수발생기의 요구와 무관하게 단순히 위의 3가지 방법에 의하여 초기값 1개가 만들어지는 에너지 소비량만을 측정하였다. 이 값을 `srand()`의 에너지 소비량이라고 한다. 이렇게 측정된 `srand()`의 에너지 소비량을 이용하여 각 난수발생기에서 초기값을 만드는 데 필요한 에너지 소비량을 알아낼 수 있다. 위의 초기값을 만드는 3 가지 방법의 에너지 소비량은 표 (4.1)과 그림 (4.1)에 나타나 있다. (표 (4.1)에서 단위 pJ은 10^{-12}J을 나타내며, J은 에너지 단위인 줄을 의미한다.) 난수 1개를 발생시키는 것에 비하여 초기값 하나를 발생시키는 필요한 에너지량은 상대적으로 크지 않기 때문에, 초기값의 갯수가 너무 많지만 않다면 난수 발생 프로그램에서 초기값을 생성시키는 에너지 소비량은 그렇게 큰 영향을 미치지는 않는다는 것을 알 수 있다. 3가지 방법의 에너지 소비량의 상대적인 차이를 알기 위하여 각 방법의 에너지 소비량을 Direct 방법에 표준화 시킨 결과가 그림 (4.1)에 나타나 있다. 이 그림에서 알 수 있듯이 Direct가 가장 적은 에너지를 소비하고, Bit masking 또한 그런대로 좋은 결과를 보이고 있으나 Randomizing의 경우는 Direct 보다 무려 3.39배나 많은 에너지 소비를 보이고 있다.

다음으로는 우리가 본 논문에서 살펴보기로 했던 각 난수발생기에서 난수를 1개 발생시키는데 필요한 에너지 소비량을 측정하였다. 앞에서 언급하였지만, 우리가 작성한 난수

RNG	LCG	MSG	SG	ALFG	SLFG	MLFG	XLFG	SRG
Energy(nJ)	55.46	321.24	321.68	85.16	85.03	91.13	85.64	39.04

표 4.2: 각 난수발생기의 에너지 소비량

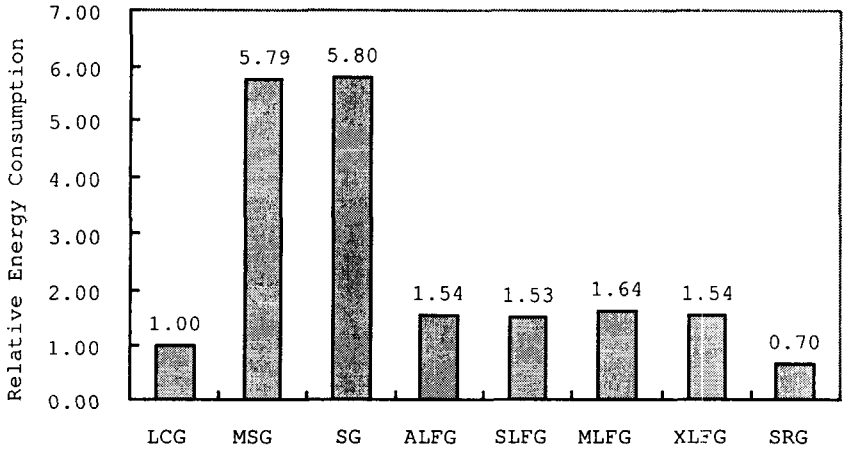

그림 4.2: 각 난수발생기의 상대적 에너지 소비량

발생 프로그램은 한 번의 srand() 함수의 호출과 1000번의 rand() 함수를 호출하도록 되어 있다. 그러므로 이 프로그램의 전체에서 소비된 에너지양에서 위에서 구한 srand()의 에너지 소비량을 이용하여 초기값을 만드는 소비된 에너지 값을 빼고 남은 에너지 값이 난수를 발생시키는데 필요한 에너지 소비량이 되며, 표 (4.2)의 실험결과를 보면 알수 있듯이 한개의 난수를 발생시키는 데 필요한 에너지의 양이 39.04nJ부터 321.68nJ까지 다양하다. (단위 nJ은 10^{-9}J을 나타낸다.) 일반적으로 프로세서에서 곱하기 연산이 다른 연산보다 시간도 훨씬 오래 걸리고 소비되는 에너지의 양도 많다. 그러므로 다른 난수발생기보다 곱셈의 빈도가 많은 MSG나 SG의 경우는 에너지 소비량이 다른 난수발생기에 비하여 많다. 각 난수발생기의 상대적인 에너지 소비량의 비교를 위하여 각 난수발생기의 에너지 소비량과 LCG 에너지 소비량을 비교한 결과를 그림 (4.2)에 나타냈다.

그림(4.2)에서 볼 수 있듯이 에너지 소비가 가장 적은 난수발생기는 SRG이다. 이 난수발생기는 LCG보다 약 30% 나 적은 에너지 소비량을 나타낸다. 서로 다른 이항 연산자들을 가지는 LFG들은 거의 서로 비슷한 양의 에너지 소비량을 보인다. 이중에서는 물론 MLFG가 곱하기 연산을 이용하므로 가장 많은 에너지 소비량을 나타내고 있다. MSG와 SG는 LCG에 비하여 거의 6배나 많은 에너지 소비량을 보이고 있다. 이것은 MSG와 SG에서는 곱하

기 연산이 많은 부분을 차지하고 있기 때문이다. 앞에서도 언급하였지만 곱하기 연산은 다른 연산에 비하여 전력 소모가 상당히 많은 연산이기 때문이다.

그리고 본 논문에서 결과가 실리지는 않았지만 식 (2.1)과 식 (2.2)에 사용된 상수 c값의 변화에 따른 에너지 소비량도 조사하였다. 그러나 값에 의한 에너지 소비량의 차이가 그리 크지가 않고 일정한 규칙성도 없었다. 즉 에너지 소비량의 측면에서 상수 c는 크게 영향을 미치지 않는다는 결론이다.

5. 난수성 검정결과

4절에서 우리는 각 난수발생기의 에너지 소비량을 조사하였는데, 난수성(randomness)을 난수발생기에서 우선 고려해야 한다. 에너지 소비량이 아무리 적다고 하더라고 난수발생기의 난수성이 떨어진다면, 그 난수발생기는 사용하기가 어려울 것이다. 저전력을 소모하는 난수생성자의 선택에서 독립성(independency)과 일양성(uniformity)를 알아보기 위해, 여러 가지의 검정법 중에서 카이제곱 검정과 Kolmogorov-Smirnov 검정 두가지를 골라서 5개의 난수생성자에서 만들어진 난수 $(u_1, \ldots, u_n)$을 이용하여 유의성검정을 하였다. $(n = 2000)$

생성된 난수는 일양분포 $Unif(0,1)$을 따른다는 귀무가설 하에서 정의역 $[0,1)$을 동일한 폭을 갖는 k개의 구간으로 나눈다. i번째 구간에 들어가는 난수 갯수의 관측치(Observed)를 O_i $(i = 1, \ldots, k)$라고 하고 기대(Expected)돗수를 $E_i = n(1/k)$라고 할 때, 카이제곱 통계량

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

는 귀무가설 하에서 자유도 $k-1$인 χ^2-분포를 점근적으로 따른다. 이때 임계값 $\chi^2(k-1;\alpha)$는 자유도 $k-1$인 χ^2-분포에서 $100(1-\alpha)$ 백분위수이다. Splus의 `chisq.gof()`를 이용한 5개 난수생성자의 검정결과가 표 (5.1)에 있는데, ALFG와 XLFG는 귀무가설이 기각된다(사용을 피해야 한다.).

귀무가설하에서 분포함수는 $F_0(x) = x$, $0 < x < 1$이므로, 경험적 분포함수(empirical distribution function)를 $F_N(u) = \sum I_{(-\infty,x)}(u_i)$라고 할 때, KS검정통계량은

$$D_N = \max_u |F_n(u) - F_0(u)|$$

이며 이 검정통계량의 임계값은 흔히 별도의 수표로 주어진다. Splus의 `ks.gof()`를 이용하여 검정한 결과가 표(5.1)에 나타나 있다.

6. 결론

이동 단말기에서 사용하는 대다수의 프로그램들은 LCG를 사용하고 있다. 이는 LCG가 ANSI C 표준이기 때문에, 난수발생기가 필요한 경우 C 언어의 표준 라이브러리에 있는 LCG를 이용한 `rand()`함수를 소프트웨어 개발자들이 관행적으로 사용하기 때문이다. 그러나 표 (4.2)의 결과를 보면 SRG가 다른 난수발생기들보다 적은 전력 소모를 보임을 알 수

				LFG				
	LCG	MSG	SG	ALFG	SLFG	MLFG	XLFG	SRG
Pearson's χ^2	12.16	11.82	19.47	39.36	12.67	12.33	28.65	16.24
(p-value)	0.7329	0.7563	0.245	0.001*	0.6967	0.721	0.0264*	0.4363
KS	0.0596	0.0438	0.0662	0.0608	0.0467	0.0582	0.0714	0.0367
(p-value)	0.4756	0.838	0.3454	0.4501	0.7754	0.5081	0.2595	0.9504

표 5.1: 난수성 검정 결과

있으므로, PDA나 휴대폰과 같이 에너지 소비량이 매우 중요한 이동 단말기의 프로그램에서는 LCG를 사용하기 보다는 SRG를 사용함으로 인하여 30% 정도의 에너지 절약 효과를 볼 수 있으며, 전력소모면이 큰 MSG와 SG의 사용은 피해야 한다. 그러나 이동 단말기에서 사용되는 프로그램이라 하더라도 암호화와 같이 에너지 소비량 뿐만 아니라 난수성도 매우 중요한 요인으로 작용하는 경우에는 난수성이 의심되는 ALFG와 XLFG의 사용은 피해야 된다. 또한 전력소모를 고려한 초기값 생성방법 중에서는 4절에서의 'direct 방법'을 사용하여 에너지 소비량을 줄이는 것이 이동 단말기에서 현명한 판단이 될 것이다.

참고문헌

[1] 윤정민(2002), 난수 발생 프로그램. `http://davinci.snu.ac.kr/jungmin/random`.

[2] Anderson, S.(1990), "Random number generators on vector supercomputers and other advanced architecture," *SIAM Review.* **32**(2):221-251.

[3] ARM Ltd., *ARM7 Thumb Family.* `http://www.arm.com/armtech/ARM7_Thumb`.

[4] Gentle, J.(1998), *Random number generation and Monte Carlo methods.* Springer-Verlag, New York.

[5] Knuth, D.(1981), *The Art of Computer Programming: Seminumerical Algorithms.* Addison Wesley. volume 2, pages 1-93.

[6] Laramie, P.(1999), "Instruction level power analysis and low power design methodology of a microprocessor," Master's thesis, Electrical Engineering and Computer Science, University of California at Berkeley. pages 38-57.

[7] L'Ecuyer, P.(1988), "Efficient and portable combined random number generators," *Communications of the ACM.* **31**(6):742-774.

[8] Marsaglia, G.(1985), "A current view of random number generators," *Computer Sciences*

and Statistics: 16th Symposium on the Interface.(edited by L. Billard), North Holland, Amsterdam. pages 3-10.

[9] Park, S. and Miller, K.(1988), "Random number generators: good ones are hard to find," *Communications of the ACM.* **31**(10):1192-1201.

[10] Press, W., Teukolsky, S., Vetterling, W., and Flannery, B.(1993), *Numerical Recipes in C : The Art of Scientific Computing.* Cambridge University Press. second edition. pages 274-300.

[11] Schrage, L.(1979), "A more portable fortran random number generator," *ACM Transactions on Mathematical Software.* **5**(2):132-138.

[12] Shin, D., Shim, H., Joo, Y., Yun, H., Kim, J. and Chang, N.(2002), "Energy-monitoring tool for low-power embedded programs," *IEEE Design and Test of Computers.* **19**(4):7 17.

[13] Tiwari, V., Malik, S. and Wolfe, A.(1994), "Power analysis of embedded software: a first step towards software power minimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems.* **2**(4):437-445.

[2002년 12월 접수, 2003년 6월 채택]

Energy-Efficiency Evaluation of Low-Power Random Number Generators

Jungmin Yoon[1] Jihong Kim[2] Jinhyo Kim[3]

ABSTRACT

Many mobile applications, such as games, security software and mathematical applications, use a random number generator(RNG). Since mobile devices operate under a limited battery capacity, the low energy consumption is one of key system requirements. For mobile applications based on an RNG, it is important to use low-power RNGs. In this article, we evaluate the energy efficiency of several well-known RNG algorithms and suggest guidelines for selecting RNGs suitable for mobile application.

Keywords: Mobile device; Low Power; Random Number Generator

1) Graduated with Master's degree in Computer Science and Engineering, Seoul National University, Korea
E-mail : twingo@davinci.snu.ac.kr
2) Associate professor, School of Computer Science and Engineering, Seoul National University, Korea
E-mail : jihong@davinci.snu.ac.kr
3) Assistant professor, Department of Computer Sciences and Statistics, Cheju National University, Korea
E-mail : jinkim@cheju.ac.kr