논문 2010-47SC-3-1

소형마이콤에서의 카오스난수 발생 함수구현

(Chaos Based Random Number Generation In Tiny MCU)

현 재 호*

(JaeHo Hyun)

다수의 노드(node)를 갖는 네트워크 시스템에 있어서 Ethernet 등의 방식을 이용한 LAN 기반의 시스템이 아닌 작은 규모의 제품에서는 RS-485 통신방식을 적용하는 것이 보통이다. 다수의 제품이 RS-485 버스에 결속되어 있을 때 각각의 제품(노드)은 Dip-Switch를 이용하여 개별 Address를 설정 한 후 전체 시스템은 소정의 작동을 하게 된다. 통신의 안정성 때문에 1:n Polling 방식을 사용 할 수 있으나 속도 문제와 Master Node의 부담으로 인하여 다자간 n:n Multi 통신방식을 선호한다. 이 경우 Master 없이 각 Node는 상대 Node의 Address로 직접 송신하게 되는데, 여기서 충돌을 막기 위하여 몇 가지 방법을 사용하긴 하지만 결국 충돌 발생률을 낮출 뿐이지 충돌은 존재 한다. 따라서 재송신 방법에 따라 시스템의 안정성이 좌우 된다. 이는 적정 지연 후 재송신 하는 것인데 이때의 지연시간 결정이 매우 모호하다. 대부분의 경우 유사 난수를 발생하여 이를 해결하곤 하는데 마이콤 기반의 작은 시스템에서의 난수 발생은 의외로 어려운 과제이므로 이 모든 것을 해결하는 방안으로 카오스난수 발생기를 마이콤에서 구현하고자 한다. 카오스 난수를 발생시키면 난수의 안정적인 결과를 기대 할 수 있으며 카오스 난수 지연으로 인하여 시스템의 안정성도 높아진다.

Abstract

RS-485, communication bases from small network system must prepare in collision. The collision is that mean the data transfer breaks. For a stabilized communication chooses 1:N polling methods. But polling is low speed in addition to maybe overload Master device. So, usual N:N Prefers a communication. In this case, must be preparing to avoid collision or some solutions. Generally, to after collision retransmits after short time. It's called delay time for short time. When making a delay time, uses address of each systems. (Address of each node) If the many nodes collided, the each node has different delay time. When making a delay time, uses a usual random number. Making a random number is hard job. So uses a usual pseudorandom number. It is more difficult from small size MCU. The Chaos random number provides stabled value. Finally, when uses the Chaos random number, the stability and reliability of system get better.

Keywords: 난수발생기, 카오스난수, Chaos random number

I. 서 론

일반적으로 공통배선 방식(Multi Drop)에서의 통신 에러 발생 시 재송신은 적정지연시간 후에 이루어진다. 적정시간은 유사난수를 발생시키거나 혹은 자신의 address값을 이용해 지연시간을 얻기도 한다. 하지만 이 모든 방법이 충돌한 Node들 모두에게 동일하게 적용되므로 다수의 Node가 충돌 할 경우, 시스템은 마비(Halt) 상태가 될 수도 있다. 카오스난수를 이용하여 지연함수를 만들면 상당히 개선된 결과를 가져온다. 이는 충돌이 발생하였을 때 빠른 시간 안에 모든 Node가 통신을 완료한다는 의미이다. 실험분석 결과 카오스난수는 유사난수에 비하여 진정한 난수에 가까운 결과를 가져오기 때문에 이러한 결과를 얻을 수 있다고 분석 되었다.

본 논문에서는 난수 발생기를 제작함에 있어 지금의 유사난수발생기의 구현방법과는 다른 카오스이론을 적용함을 그 수단으로 고려하였다. 카오스이론을 사용하게 된 동기는 난수발생기의 결과가 지속적으로 반복되는 지금의 유사난수발생기를 개선하기위해서는 불규칙과 규칙의 논리를 전개한 카오스의 이론이 가장 적합하

* 정회원, 연세대학교
(Yonsei University)
접수일자: 2009년7월22일, 수정완료일: 2010년5월10일

다고 판단되었기 때문이다. 실제로 카오스 난수 발생에 관한 이론은 인터넷 등 관련 사이트에서 많이 접할 수 있다. 이는 이미 카오스난수의 필요성을 알게 되었고 이를 활용하는 단계에 이르렀다고 판단된다.

하지만 그 내용들은 상당한 복잡한 공식과 함수로 나열되어 있어, 상당한 수학적 함수를 계산할 만한 성능의 장치, 이를테면 컴퓨터를 기반으로 전개된 것이 보통이어서 일반 마이컴 등에서의 활용은 불가능한 것이 보통이다. 본 논문에서는 실사용이 매우 빈번한 저가의 소형 마이컴을 기반으로 카오스난수를 발생하고자 하는 것이 본 취지이므로 카오스모델이라는 기본 원칙에 준거하여 연구와 실험을 하였고 이에 따른 결과를 개제함을 목적으로 한다.

이에 카오스 이론의 기반설명이 있을 수 있으나 이미 널리 알려진 이론이므로 본 논문에서는 카오스의 일반적인 이론에 관한 설명은 하지 않기로 한다. 다만 카오스이론에 있어서 카오스모형을 이루는 공식을 인용 및 변형하는 과정을 본론에 기술할 것이다.

Ⅱ. 본 론

1. 기존 방법의 예

엔지니어에 따라 다른 방법을 구사 할 수 있겠으나 대체적으로 원론에 의거한 두 가지 방법이 사용되고 있다.

(1) 씨앗(Seed)를 기준으로 일정 루프(Loop)를 반복 수행하여 값을 얻는 방법.

(2) 레지스터를 지속적으로 회전(보통의 경우 Shift Left 또는 Rotate Left) 시키되 특정 Bit들을 배타논리합(Exclusive Or)하여 그 결과를 최하위비트(Bit 0)에 적용하는 방법.

상기 (1)의 방법에 있어 씨앗의 값에 매번 다른 값을 넣어야 하고 이 결정이 모호하여 현재의 시각정보를 넣기도 하지만 빠른 시간 안에 두 개의 난수를 구하려 하면 당연히 동일한 결과가 나오게 된다. 더욱이 시계(RTC: Real Time Clock) 부품이 내장되어 있지 않은 시스템에서는 이러한 방법조차 불가능함으로 씨앗 값의 설정은 더욱 어려워진다. 상기 (2)의 방법은 (1)항에 비하여 좋을 수 있으나 큰 Loop를 기준으로 보면 반복되는 함수라고 할 수 있으며 이의 구현이 MCU의 구조 또는 명령어(Instruction Set)에 따라 어렵거나 복잡할 수 있다. 가장 큰 문제는 어느 시점에서 반복되는 결과가 얻어진다는 점이다.

무엇보다 이 들의 문제점은 난수를 발생하는데 소요되는 시간이다. 많게는 수백 번의 Loop를 돌아야 하는 이러한 구조는 소형 MCU의 프로그램(Firm ware) 설계에서는 금칙으로 되어있는 불문율이다. 이는 MCU에서의 여타 작업에 영향을 끼칠 만큼 자원이 많이 소모되기 때문이다.

2. 논리 구현

본 논문에서는 카오스모델을 기반으로 난수발생을 Loop를 돌아 생성됨이 아닌, 수식에 의하여 간결하고 빠르게 생성시키자는 취지가 가장 크다. 카오스 난수발생에 있어 카오스모델 공식을 기반으로 하였다.

결국, 지정된 Loop 프로그램을 반복하지 아나하고 단 하나의 루틴을 Call 함으로서 양질의 난수를 얻자는 것이다.

$$X = X^2 + C[1] \quad (1)$$

(C는 상수, X는 결과 값을 계속적으로 대입)

시험의 편의상 결과 값 X 는 Y로 변경하였다.

$$Y = X^2 + C[1] \quad (2)$$

소형 MCU 에서는 결과치의 영역이 8Bit 또는 16Bit에 불과하다. 본 연구와 실험에서는 보다 열악한 상황으로 실험하기위하여 8Bit를 선택 하였다. 256개에 불과한 작은 영역에서의 카오스 구현은 지속적인 중첩 결과 발생으로 인하여 많은 어려움이 있었다. 많은 실험 끝에 본래 함수에서의 C 는 어떤 기준 상수를 의미하지만 본 실험에서는 C를 n을 대치하였다.

$$Y = X^2 + n \quad (3)$$

C를 기본 상수를 가져갈 경우, 256 또는 65,536이라는 작은 범위를 기반인 MCU에서 기본 공식을 적용하며 이 또한 난수의 결과가 반복되어 그 의미를 잃게 된다. 따라서 C를 n으로 대체하고 결과 Y를 다음에 X로 대입할 때는 8Bit 로 Mask 하였으며 n은 난수 발생 할 때마다 증가시키는 인수로 변경하였다. 변수 n을 지속적으로 증가시키는 증분 값을 1 부터 실험 하였으나

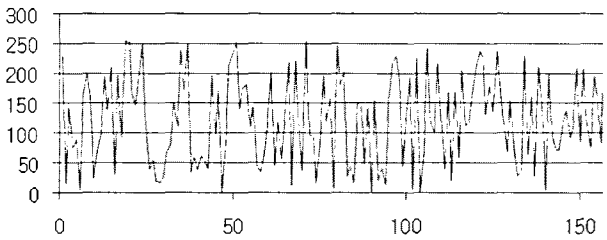

그림 1. Microsoft 社의 Excel을 이용하여 출력한 그래프 본 함수를 적용하여 발생 한 난수를 지속적으로 Plotting 하였다

Fig. 1. Computer added simulation for the Chaos based random number generation. Used Excel (Microsoft).

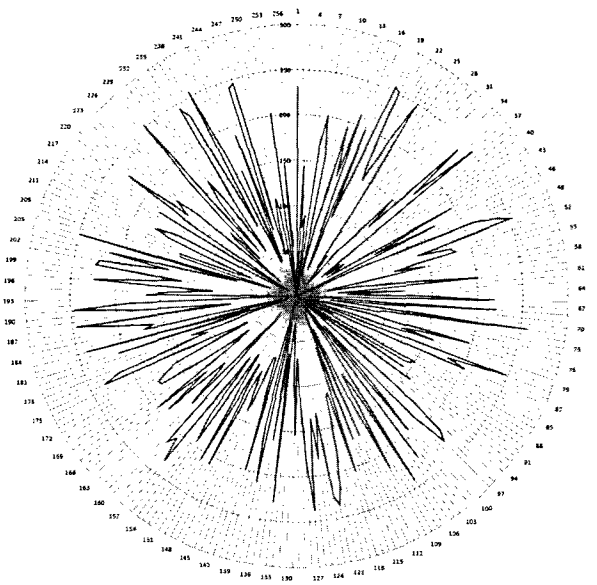

그림 2. 200회 이상 난수를 발생하여 그 결과를 레이더 그래프로 Plotting 하였다.

Fig. 2. Same as Fig.1, With radar graph for over a 200 times generation. Used Excel(Microsoft).

(1.2.3....n) 누차의 실험결과 소수(Prime Number)를 사용함이 최상임을 알게 되었다.

그 증분 값을 예컨대 67로 하였을 때

$$Y= X^{2}+ n, \ (n=n+67). \tag{4}$$

실험은 67로 하였으나, 여타의 임의의수를 넣어도 유사 결과 값을 얻을 수 있다. 이를 Microsoft 社의 Excel에 입력하여 결과를 Simulation 하였으며 그 결과를 그래프로 표현 할 경우 그림 1과 같다.

y축은 8Bit영역의 값을 갖는 chaos_random의 결과값이며 x축은 누차에 거친 chaos_random 함수 call 차수이다. 그에 따른 연속적 결과를 Plotting 한 것이다.

그림 2는 200회 이상 난수를 발생시키고 그 결과를 레이더 그래프로 도시하였다. 그래프 시계 방향은 발생회 차수이고 동심원은 난수의 결과 값인 0에서 255까지의 영역이다. 그림에서 보는 바와 같이 난수의 결과가 전 영역에 거쳐 골고루 발생하였다.

Ⅲ. 실 험

실험은 가장 널리 사용하는 MCS8051 core를 이용하여 이를 구현하였다. 프로그램에는 두 개의 변수(x,n)가 필요하며, 난수를 필요로 할 때에는 해당 서브루틴을 Call 하는 형태로 구성하였다. 그리고 두 개의 변수는 지속적으로 변경되며 마지막 값을 유지하여야한다. 프로그램으로 구현한 서브루틴 chaos_random은 자체적으로 두 개의 변수를 관리하므로 이를 사용하는 응용프로그램에서는 두 개의 변수의 변화와 관리에 대한 노력은 필요 없다.

실무에서 사용할 경우 본 문에 기술된 서브루틴 Chaos_Random을 Call(호출)하고나면 카오스 난수가 얻어진다.

1. 프로그램 구현[2]

변수 x와 n을 각각 var_x, var_n 이라 했을 때,

```
0: Chaos_random:
1 ;n=(n+67),
2       MOV     a,var_n
3       ADD     a,#67
4       MOV     var_n,a
5 ;X²결과= a레지스터:하위8Bit, b레지스터:상위8Bit
6       MOV     a,var_x
7       MOV     b,var_x
8       MUL     ab
9 ;var_x  + var_n
10      ADD     a,var_n
11 ;최종결과를 var_x 에 옮김(공식의 Y 결과값)
12      MOV     var_x,a
13      RET
```

상기와 같이 구현되었으며, 이를 사용할 때는 'chaos_random' 함수를 call만 하면 된다. 그 결과는 var_x 변수에 저장된다. 따라서 난수가 필요 할 때마다 chaos_random 루틴을 call 하면 매번 다른 값의 카오스

난수를 얻을 수 있다.

2. 프로그램 설명

공식에서의 n은 call 할 때 마다 67을 더한 후 그 값을 다시 n에 저장한다. 프로그램의 Line:2~4 부분이며 그 결과는 8bit를 넘을 수 있으나 8bit MCU 이므로 8bit 이상의 값은 자연스럽게 없어지고 var_n 에는 하위 8bit 값만 남게 된다. 결국 8bit 로 Mask 함에 있어서 추가적인 조치는 필요 없다.

공식에서의 X^2을 위하여 Line:6에서 MUL(곱하기) 명령어를 사용한다. var_x를 각각 a와 b 레지스터네 저장한 후 MUL 명령어를 통하여 곱셈을 한다. 그 결과는 각각 b 레지스터에는 상위8bit가, a레지스터에는 하위 8bit가 저장된다. 본 논문에서는 8bit만 사용하므로 상위 8bit 인 b 레지스터는 사용하지 않는다.

공식의 Y를 만들기 위하여 Line:2~4의 결과(var_n)와 Line:6~8의 결과(var_x)를 더해준다. Line:10 결과값 Y 는 다음 연산을 위하여 var_x 에 옮기고 서브루틴은 복귀한다.(Return from subroutine)

따라서 최종 값은 var_x에 저장되며 응용프로그램에서도 var_x의 값을 이용하여 용도에 맞게 사용하면 된다.

Ⅳ. 결 론

본 연구와 실험의 시작은 소형 네트워크 시스템에서의 지연함수의 필요성과 지연함수 발생에 있어서의 난수 사용, 그리고 난수 발생에 있어서의 기존의 난점을 개선하기위하여 카오스난수를 고려하였다.

소형 MCU에서 활용 할 수 있는 방안을 연구와 실험을 통하여 제시하였다. 본 논문에서는 소형 네트워크 시스템에서의 필요성과 사용을 예시 하였으나 난수의 용도는 매우 다양하므로 이의 활용도는 높다고 판단된다. 더욱이 실험결과가 매우 간결하면서도 강력하기 때문에 활용도는 높을 것이라 생각된다.

본 논문에서의 실험은 intel 社의 8051 core를 기준으로 assembly로 구성하였다. 하지만 내우 간단한 결과가 되었으므로 여타 MCU 에서도 충분히 변형, 적용 가능할 것으로 보인다.

참 고 문 헌

[1] 지아우딘사르사르 지음, 이충호 옮김, "카오스", 김영사, 32쪽, 2008.

[2] 차영배 지음, "어셈플리어로 배우는 8051", 동일출판사, 220쪽, 2009.

저 자 소 개

현 재 호(정회원)
2008년 연세대학교
전자공학과 석사 졸업
<주관심분야 : 통신, 신호처리>