TMS320 DSP
DESIGNER'S NOTEBOOK

Random Number Generation on a TMS320C5x

APPLICATION BRIEF: SPRA239

Eric Wilbur
Digital Signal Processing Products
Semiconductor Group

Texas Instruments
July 1994

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Examples

Random Number Generation on a TMS320C5x

Abstract

This document discusses how a random number can be generated on a TMS320C5x. The goal of this application note is to provide a fast, proven, useful random number generator that can be used in various fixed-point applications. The theory is discussed and a code example is provided.

Design Problem

How is a random number generated on a TMS320C5x?

Solution

The philosophy of the term "random" (i.e., how random is random?) has been argued for centuries. I'm sure that there were probably several duels held over the years as a result of disagreements on this topic. Some argue that using a computer (a precise, logical, predictable device) to produce random numbers is quite ironic (but useful!). Purists would state that the only truly random event in nature is the time delay between clicks of a Geiger counter placed near a piece of radioactive material.

The goal of this application note is not to solve the ongoing debate over the issue of randomness and somehow vindicate one side or another, but to provide a fast, proven, useful random number generator that can be used in various fixed-point applications.

Theory and Implementation

Many algorithms exist to generate random or pseudo-random numbers. The design objectives of this algorithm were speed, simplicity, "good" results, and the ease of integrating the code into any application. Based on these criteria, a form of uniform deviate called the linear congruential method (introduced by D. Lehmer in 1951) was used. The advantages of this method are speed, simplicity to code, and ease of use. However, if care is not taken in choosing the multiplier and increment values, the results can quickly become degenerate. This algorithm produces 65,536 unique numbers and the correlation is very good. Only the LSB exhibits a repeatable pattern every 16 calls.

The linear congruential method has the following form:

Rndnum(n) = (Rndnum(n-l) * MULT) + INC (mod M)

Where: Rndnum(n) = current random number
Rndnum (n- 1) = previous random number
Rndnum(l) = SEED value (arbitrary constant)
MULT = multiplier (unique constant)
INC = increment (unique constant)
M = modulus (word width of 'C5x = 16 bits = 64K)

Much research has been done to identify the optimal choices for the constants MULT and INC. The constants used in this implementation are based on this research. If changes are made to these numbers, extreme care must be taken to avoid degeneration.

Following is a more detailed look at the algorithm and the numbers used:

M: M is the modulus value and is typically defined by the word width of the processor. This algorithm will return a random number between 0 and 65,535 and is NOT internally bounded. If the user requires a min/max limit, this must be coded externally to this routine. The result is not actually divided by 65,536. The accumulator is allowed to overflow, thus implementing the modulus.

SEED: The first random number in the sequence is called the seed value. This is an arbitrary constant between 0 and 64K. Zero can be used, but the first two results of the generator will be 0 and 1. This is OK if the code is allowed 3 calls to "warm up" before the numbers are taken seriously. The number 21,845 was used in this implementation because it is $1/3$ of the modulus (65,536).

MULT: Based on random number theory, this number should be chosen such that the last three digits are even-2-1 (such as xx821, x421, etc.). The number 31,821 was used in this implementation. Caution: the generator is extremely sensitive to the choice of this constant!

INC: In general, this constant can be any prime number related to M. Two values were actually tested in this implementation: 1 and 13,849. Research shows that INC should be chosen based on the following formula:

$$INC = \left(\frac{1}{2} - \left(\frac{1}{6} \times \sqrt{3} \right) \right) \times M \quad \text{(Using M = 65,536 and INC = 13,849)}$$

NOTE:
This implementation can be modified to return a 32-bit or 8-bit random number if necessary. For the 32-bit number, simply modify the code to execute a 32×32 multiply instead of 16×16. Remember, your modulus is now 2^32. If an 8-bit result is desired, the low or high byte of the 16-bit result can be used. However, randomness is not guaranteed—duplications will exist.

Example 1. Code Example

```
================================================================
;; RANDOM NUMBER GENERATOR FOR THE TMS320C5x DSPs
;;
;; Title:          Rand16.ASM
;; Author.         Eric Wilbur
;; Date:           October 1993
;; Application:    Random Seeks for Hard Disk Drive
;; Target DSP:     TMS320C51
;;
```

```
;; Usage:          To Initialize:                    Call InitRandl6
;;                 To get the next random number:    Call _Rand16
;;
;; Assumptions:    SXM,OVM = don't care
;;                 SPM = 0 (no shift)
;;
;; Input           None
;; Output          ACCL = 16-bit random number
;;=================================================================
;; MEMORY ALLOCATION
;;=================================================================
;;
Rndnum              .usect   "Variables",1    ;allocate space for random
;;                                            ;number result
;;
;;=================================================================
;;INITIALIZE CONSTANTS
;;=================================================================
SEED                .set     21845            ;arbitrary seed value (65536/3)
MULT                .set     31821            ;multiplier value (last 3
                                              ;digits are even-2-1)
INC                 .set     13849            ;1 and 13849 have been tested
;;
;;=================================================================
;; CODE START
;;=================================================================
;;
                    .text
;;
;;=================================================================
;; INITIALIZE RANDOM NUMBER GENERATOR - Load the SEED value
;;=================================================================
;;
_InitRandl6:        LDP      #Rndnum
                    LACC     #SEED            ;ACC = SEED value
                    SACL     Rndnum           ;Rndnum = SEED
                    RET                       ;return to caller
;;
;;=================================================================
;; GENERATE NEXT RANDOM NUMBER
;;=================================================================
;;
_Randl6:            CLRC     OVM              ;clear overflow - implements
                                              ;MOD 64K
                    LDP      #Rndnum          ;set data page pointer
                    LT       Rndnum           ;TReg = Rndnum
                    MPY      #MULT            ;PReg = Rndnum * MULT
                    PAC                       ;ACC = Preg
                    ADD      #INC             ;ACC = Rndnum * MULT + INC
                    SACL     Rndnum           ;store new random number
                    RET                       ;return to caller
```