A Random Number Generator That is
Better Than the One in the C6x Compiler
Library

The function `rand()` in the C6x compiler library generates
integers uniformly distributed over the set \{1, 2, \ldots, RAND_MAX\} where \(\text{RAND}\_\text{MAX} = 32767 = 2^{15} - 1\). The C code for this random number generator is shown in Slide 3 and is included in the class web site.

The random number generator for the C3x/C4x compiler generates uniformly distributed integers from the set \{1, 2, \ldots, RAND_MAX\} where \(\text{RAND}\_\text{MAX} = 2^{31} - 2 = 2147483646\). The C code for this random number generator is shown starting with Slide 4 and is included in the class web site. You can find the theory behind this generator in the following references:


Both of the above references can be found on the web by doing a Google search.

A very good random number generator is required when simulating the performance of a system at high SNR’s where errors are caused by very infrequent large noise values. In Chapter 10, Gaussian random variables were generated by first forming the Rayleigh random variable

\[ R = \sqrt{-2\sigma^2 \log_e (1 - V)} \]

where \( V \) is a random variable uniformly distributed over \([0, 1)\). Let \( M = \text{RAND\_MAX} + 1 \) and that the integers from the C3x or C6x generator are normalized to be between 0 and 1 by dividing by \( M \). Then the largest possible result is \( \frac{M-1}{M} = 1 - \frac{1}{M} \). This means the largest value for \( R \) is \( R_{\max} = \sqrt{2\sigma^2 \log_e M} \) so a large \( M \) is required to get large Gaussian noise excursions. The ratio of the maximum \( R \)'s for the C3x and C6x generators is

\[
\frac{R_{\max}(C3x)}{R_{\max}(C6x)} = \sqrt{\frac{\log_e (2^{31} - 1)}{\log_e 2^{15}}} \sim \sqrt{\frac{31}{15}} = 1.438
\]

Thus the C3x generator is better, although, not dramatically better.
Random Number Generator in the C6x C Compiler Library rts.src

/***************************************************************************/
/* rand.c */
/***************************************************************************/
#include <stdlib.h>
#include <_lock.h>

static _DATA_ACCESS unsigned long next = 1;

_CODE_ACCESS int rand(void)
{
    int r;
    _lock();
    next = next * 1103515245 + 12345;
    r=(int)((next/65536) % ((unsigned long)RAND_MAX+1));
    _unlock();
    return r;
}

_CODE_ACCESS void srand(unsigned seed)
{
    _lock();
    next = seed;
    _unlock();
}
Random Number Generator from the C30 C Compiler Library rts.src

/***/
/* rand.c V5.11 for TMS3203x/4x */
/* */
/* NOTE: This file should be compiled with the -mm (short multiply) */
/* switch for best results. */
/*******************************
#include <stdlib.h>

static unsigned next = 1;

/*******************************
/* RAND() - COMPUTE THE NEXT VALUE IN THE RANDOM NUMBER SEQUENCE. */
/* */
/* The sequence used is x' = (A*x) mod M, (A = 16807, M = 2^31 - 1). */
/* This is the "minimal standard" generator from CACM Oct 1988, p. 1192. */
/* The implementation is based on an algorithm using 2 31-bit registers */
/* to represent the product (A*x), from CACM Jan 1990, p. 87. */
/* */
*******************************/
#define A 16807u /* MULTIPLIER VALUE */

int rand()
{
    unsigned x0 = (next << 16) >> 16; /* 16 LSBs OF SEED */
    unsigned x1 = next >> 16; /* 16 MSBs OF SEED */
    unsigned p, q; /* MSW, LSW OF PRODUCT */

    /* 16-BIT HALVES OF THE INPUT VALUES. THE RESULT IS REPRESENTED AS 2 */
    /* 31-BIT VALUES. SINCE 'A' FITS IN 15 BITS, ITS UPPER HALF CAN BE */
    /* DISREGARDED. USING THE NOTATION val[m::n] TO MEAN "BITS n THROUGH */
    /* m OF val", THE PRODUCT IS COMPUTED AS:
    */
    /* q = (A * x)[0::30] = (A * x1)[0::14] << 16) + (A * x0)[0::30] */
    /* p = (A * x)[31::60] = (A * x1)[15::30] + (A * x0)[31] + C */
    /* WHERE C = q[31] (CARRY BIT FROM q). NOTE THAT BECAUSE A < 2^-15, */
    /* (A * x0)[31] IS ALWAYS 0. */
/*---------------------------------------------------------------------*/
    q = ((A * x1) << 17 >> 1) + (A * x0);
    p = ((A * x1) >> 15) + (q >> 31);
    q = q << 1 >> 1;  /* CLEAR CARRY */

/* IF (p + q) < 2^31, RESULT IS (p + q). OTHERWISE, RESULT IS */
/* (p + q) - 2^31 + 1. (SEE REFERENCES). */
/*---------------------------------------------------------------------*/
    p += q;
    return next = ((p + (p >> 31)) << 1) >> 1;  /* ADD CARRY, THEN CLEAR IT */
}

/***************************************************************************/
/* SRAND() - SET THE INITIAL SEED FOR RAND(). */
/***************************************************************************/
void srand(unsigned seed)
{
    next = seed;
}