Generador Pseudoaleatorio de Many-Core basado en Chudnovsky

Jesús Antonio Álvarez-Cedillo, Teodoro Álvarez-Sánchez, Raúl Junior Sandoval-Gómez

[a,]*Instituto Politécnico Nacional, jaalvarez@ipn.mx, CDMX México.*
[b]*Instituto Politécnico Nacional, talvarezs@ipn.mx, CD Tijuana Baja California México.*
[c]*Instituto Politécnico Nacional, rsandoval@ipn.mx, CDMX México.*

Resumen

El algoritmo de cálculo de Pi de Chudnovsky, concebido por David y Gregory Chudnovsky, destaca como el método más prevalente para estimar el valor de Pi. Con la capacidad de generar hasta 7 mil millones de decimales en sus cálculos, este algoritmo sobresale por su rapidez en comparación con otros métodos. ¡Su complejidad computacional de O(n)! permite una implementación eficiente y lineal, lo que lo convierte en una opción ideal para procesos iterativos. En este estudio, se empleó este algoritmo como base para generar una extensa secuencia de multiplicadores decimales de forma pseudoaleatoria, utilizando un generador pseudoaleatorio implementado en una tarjeta Parallax P8x32a. Los resultados obtenidos se compararon con otros enfoques similares, demostrando la eficacia y versatilidad de esta técnica.

Palabras clave— *procesamiento paralelo; generador pseudoaleatorio; Cálculo de Pi; Chudnovsky; dispositivos embebidos.*

Abstract

The Chudnovsky algorithm for calculating Pi, devised by David and Gregory Chudnovsky, stands out as the most widely used method for estimating the value of Pi. Capable of generating up to 7 billion decimals in its calculations, this algorithm excels in speed compared to other methods. Its computational complexity of O(n)! allows for efficient and linear implementation, making it an ideal choice for iterative processes. In this study, this algorithm was used as the main seed to randomly generate a large sequence of decimal multipliers using a pseudo-random alternator, implemented on a Parallax P8x32a card. The obtained results were compared with others from similar approaches, demonstrating the effectiveness and versatility of this technique.

Keywords— *Parallel processing, pseudorandom generator, Calculation of Pi, Chudnovsky, embedded devices.*

1. INTRODUCCIÓN

Los generadores de números integrados en la electrónica y el desarrollo robótico son omnipresentes, aunque su precisión puede variar. Diversos autores han propuesto múltiples estrategias de implementación, desde el uso del reloj interno o de cronómetros establecidos de manera específica, hasta la utilización de números primos e incluso algoritmos avanzados basados en inteligencia artificial.

Pi, como número irracional que es, representa el cociente entre la circunferencia y su diámetro, y encuentra aplicaciones frecuentes en campos como las matemáticas, la física y la ingeniería para la realización de diversos cálculos. A pesar de su infinita expansión decimal, no se requiere conocer una gran cantidad de dígitos para su uso práctico. Por ejemplo, la NASA emplea 16 lugares decimales para sus cálculos, mientras que el Instituto Nacional de Estándares y Tecnología (NIST) utiliza 32 bits para la verificación de cálculos informáticos. Más allá de estos estándares, la extensión de la precisión decimal se emplea en el desarrollo e investigación de tecnologías de cifrado y criptografía [1].

En la actualidad, el algoritmo predominante para calcular el valor de Pi (π) es el algoritmo de Chudnovsky, creado por David y Gregory Chudnovsky. Este algoritmo estableció récords mundiales en diciembre de 2009 al generar 2.7 mil millones de dígitos de π, en agosto de 2010 al alcanzar los 5 mil millones, y en octubre de 2011 al llegar a los 10 billones de dígitos [2].

En la literatura, se han documentado varios algoritmos para la generación de números aleatorios, entre ellos:

a) Generadores de Congruencia Lineal (GCL): Son los más comunes y conocidos, fundamentados en la tasa de recurrencia [3];

b) Generadores de Desplazamiento de Bits: Estos generadores obtienen cada nuevo entero aleatorio manipulando los bits del número anterior [4];

c) Generadores Fibonacci: Destacan por su alta velocidad y largo período. Se basan en un estímulo teórico diferente al de los GCL [5];

d) Distribuciones no Uniformes: Se aborda el desafío de obtener secuencias con una densidad de probabilidad específica (w(y)) en un intervalo definido (y_min, y_max) a partir de una secuencia de números con distribución uniforme U (0,1) [6];

El principal reto en los generadores radica en producir secuencias de números uniformemente distribuidas en un intervalo dado. Para resolver este problema, se han propuesto varias soluciones [7].

i) Consultar tablas de números aleatorios previamente publicadas;

ii) Observar procesos físicos como la desintegración radiactiva o el ruido eléctrico;

iii) Utilizar funciones integradas en lenguajes de programación y hojas de cálculo;

iv) Emplear algoritmos específicos diseñados para la generación de números aleatorios.

Las principales ventajas de los generadores de números aleatorios son [7]:

1. Velocidad
2. Conveniencia
3. Reproducibilidad
4. Portabilidad

Además, las desventajas son:

1. Las secuencias obtenidas no son verdaderamente aleatorias, ya que se generan mediante operaciones deterministas.
2. En su lugar, se obtiene secuencias pseudoaleatorias que, aunque no son aleatorias en el sentido estricto, cumplen con ciertos criterios de aleatoriedad apropiados.

Los números generados deben cumplir con características específicas para ser considerados válidos, tales como:

1. Distribución uniforme.
2. Estadística independiente.
3. La media debe ser estadísticamente igual a 0.5.
4. Varianza debe ser estadísticamente igual a 0.5.
5. Período o ciclo de vida ampliado.
6. Deben generarse mediante un método rápido.
7. Generación a través de un método que requiera un bajo consumo de capacidad de almacenamiento en la computadora.

Por lo general, los números enteros son preferidos debido a su aritmética precisa y eficiente.

Los enteros se generan N_i entre 0 y M-1, y $x_i = \frac{N_i}{M}$ genera valores absolutos en el intervalo [0, 1].

En términos generales, los algoritmos utilizan relaciones de recurrencia del tipo:

$$N_i = f(N_{i-1}) \quad (1)$$

en el caso de recurrencia simple:

$$N_i = f(N_{i-1}, \dots, N_{i-k}) \quad (2)$$

Para el caso de una recurrencia de orden k.

Se requiere proporcionar un valor inicial para iniciar el algoritmo (valores k para recurrencias de orden k).

2. METODOLOGÍA

Los matemáticos ucranianos David y Gregory Chudnovsky son los creadores del algoritmo que lleva su nombre. Este algoritmo, considerado uno de los más modernos y ampliamente utilizados para calcular π con alta precisión, se fundamenta en una fórmula desarrollada por el matemático hindú Ramanujan e implementa una serie de convergencia rápida mediante una serie hipergeométrica.

La expresión matemática es la siguiente:

$$\frac{1}{\pi} = 12 \sum_{n=0}^{\infty} \left(\frac{(-1)^k (6k)! (13591409 + 545140134k)}{(3k)! (k!)^3 * 640320^{3k+\frac{3}{2}}} \right) \quad (3)$$

Computacionalmente, la implementación de (3) se muestra en el algoritmo 1.

Algoritmo 1: Pi conChudnovsky

precisión de= 14
Límite = (Precisión + 3) / 14

Para k <Límite
Numerador = 0
Denominador = 0

Numerador = $-1^k (6k)! (13591409 + 545140134 * k)$
Denominador = $(3k)! (k!)^3 * 640320^{3k+\frac{3}{2}}$
3/2PINumber = PINumber + (Numerador / Denominador)
k = k + 1
PINumber = 12 * PINumber
PINumber = 1 / PINumber

La arquitectura Propeller Parallax está compuesta por 8 procesadores independientes de 32 bits, conocidos como COGS, etiquetados del 0 al 7. Cada COG cuenta con 2 KB por celda (512 x 32 bits) de RAM, así como dos contadores PPL 1x-16x, un generador de video capaz de manejar señales PAL, NTSC y VGA, además de un registro de entrada y salida, y un registro de direcciones.

Cada COG tiene acceso exclusivo a los recursos compartidos mediante un controlador de bus interno llamado Hub, que opera con un esquema de programación por turnos. El Hub está equipado con 32 KB de RAM para código de programa, datos variables y 32 KB de ROM.

La arquitectura Propeller Parallax distingue entre dos tipos de recursos: mutuamente excluyentes y compartidos. Los recursos compartidos, como ciertos pines de entrada y salida y un contador del sistema, pueden ser accedidos por cualquier COG en cualquier momento. Por otro lado, los demás recursos son mutuamente excluyentes y son gestionados por el Hub.

Los datos pueden definirse como bytes o palabras (8, 16 o 32 bits respectivamente) y almacenarse en formato "little-endian". Aunque la memoria está compuesta por celdas de 32 bits, se puede leer un byte o una palabra para mantener el orden correcto. El lenguaje ensamblador de Propeller permite definir datos como bytes alineados de menor tamaño.

El microcontrolador Propeller Parallax fue diseñado para aplicaciones científicas de propósito general y presenta las

siguientes características:

El sistema aprovecha ocho núcleos independientes (COGS) que operan en paralelo, compartiendo todos los recursos, ya sea memoria o E/S.

La gestión de todos los recursos está a cargo del hardware y está fuera del control del programador, lo que garantiza una latencia predecible.

Cada núcleo o COG cuenta con su propia RAM privada de 4 KB, además de capacidad para generar diversas señales.

No se implementa el concepto de interrupciones en ningún aspecto. En su lugar, se ejecutan tareas simultáneas en diferentes "núcleos".

El lenguaje Spin se desarrolló para facilitar la programación simultánea y multinúcleo, combinando enfoques procedimentales y orientados a objetos.

El Listado 1 presenta un ejemplo de código que resulta muy legible y comprensible.

Listado 1: Ejemplo spin

```
CON
_clkmode = xtal1 + pll16x
_xinfreq = 5_000_000
OBJ
led: "LD.spin"
VAR
byte Contador
pila larga [90]
PUB Main
cognew (flicker (16, clkfreq / 50), @stack [0])
cognew (parpadeo (19, clkfreq / 150), @stack [30])
cognew (parpadeo (22, clkfreq / 100), @stack [60])

PUB parpadeo (PIN, RATE)
repetir
 repetir Contador de 0 a 100
 led. (Contador, PIN)
   waitcnt (RATE + cnt)
  repeat Contador de 100 a 0
   led.LED Brillo (Contador, PIN)
   waitcnt (RATE + cnt)
```

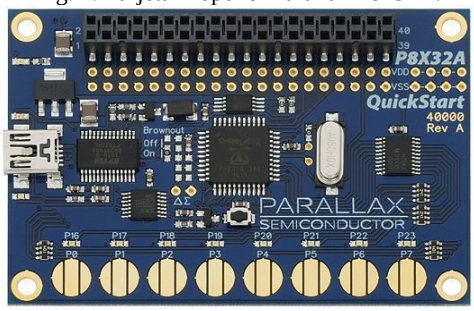
La función cognew activa la función Flicker en tres COGS, cada uno parametrizado con una frecuencia de acceso y un área de memoria específica.

La figura 1 muestra la tarjeta utilizada. En la figura 2 ilustra un diagrama del sistema embebido, mientras que la

2.1 Implementación del algoritmo

La obtención de verdadera aleatoriedad es un desafío considerable que requiere una cantidad significativa de recursos informáticos. Diversos investigadores han concluido que las soluciones presentadas en la literatura no siempre son óptimas, aunque son factibles para su implementación en sistemas embebidos. La estrategia común para generar números pseudoaleatorios implica utilizar una semilla inicial generada por un proceso lo suficientemente aleatorio como para alimentar una función que produzca una secuencia impredecible.

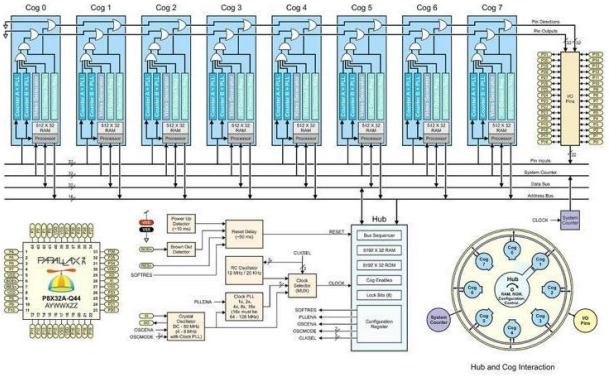
Fig. 1. Tarjeta Propeller Parallax P8x32A.



Fuente: Parallax.com

Dada una clave o semilla, no es necesario prever cómo se transformarán los datos. Por lo tanto, un generador de números aleatorios no es más que una función específica que toma una semilla aleatoria y genera una secuencia simple. El enfoque recomendado se muestra en la figura 3. En este artículo, la semilla consistirá en un número aleatorio básico, el cual será utilizado para alimentar la función de Chudnovsky con el propósito de calcular pi. Al calcular pi con una expansión decimal aleatoria, se seleccionará los últimos cinco números generados por la función y combinación de estos valores. Luego, dividirá el resultado por 1,000,000 para expresarlo en formato decimal, obteniendo así un número aleatorio.

Fig. 2. Sistema embebido de la tarjeta P8x32A.



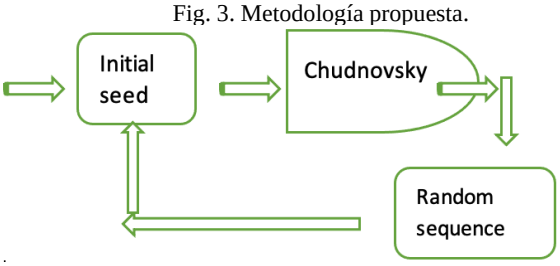
Fuente: the official Guide book Parallax

Dada una semilla s = 1000 obtenida mediante la función aleatoria de C, la función de Chudnovsky (1000) con esta semilla produciría el siguiente valor de π =3.10198.

En este caso, se tomará los últimos cinco dígitos 10198, y se dividirá por 1e + 6, con lo cual se genera el número aleatorio 0.10198,. La nueva semilla se obtiene del tiempo de

ejecución del algoritmo. La implementación de la función de Chudnovsky en la tarjeta de la figura 1, también se muestra en la figura 3 la metodología, así como el código en el Listado 2.

Fig. 3. Metodología propuesta.



Fuente: elaboración propia

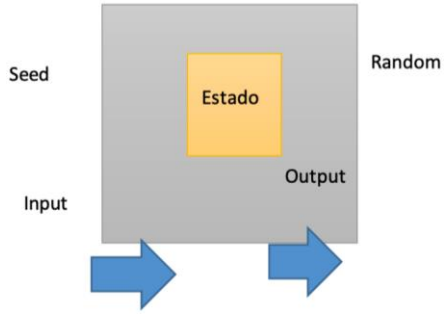
Listado 2: Función de Chudnovsky

```
#include <stdio.h>
const int n = 10000;
const int dim = ((10 * n) / 3);
int i, j, k, resto, digant, nueves, aux;
int pi [dim + 1];
int main (void) {
para (i = 1; i <= dim; i ++) pi [i] = 2;
nueves = 0; digante = 0;
para (i = 1; i <= n; i ++) {
  resto = 0;
  para (j = dim; j> = 1; j -) {
        aux = 10 * pi [j] + resto * j;
        pi [j] = aux% (2 * j-1);
        resto = aux / (2 * j-1);
      }
  pi [1] = resto% 10;
  resto = resto / 10;
  if (resto == 9) nueves ++;
  else if (resto == 10) {
    printf ("% i", digant);
    para (k = 1; k <= nueves; k ++) printf ("0");
   digante = 0;
   nueves = 0;}
     else {
     printf ("% i", digant);
     digante = resto;
     if (nueves! = 0) {
 for (k = 1; k <= nueves; k ++) printf ("9");
 nueves = 0;
}}}
printf ("% i", digant);
scanf ("% i");
}
```

3. IMPLEMENTACIÓN

El generador de hardware implementado en Propeller Parallax enviará un número aleatorio cada vez que otro dispositivo de hardware lo solicite. Además, actualizará la semilla en el dispositivo utilizando el tiempo de generación proporcionado por la función. La estructura generada en el dispositivo se muestra en la figura 4.

Fig. 4. Entidad desarrollada.



Fuente: elaboración propia

Para poner a prueba el generador, se empleó un Arduino uno, en el cual se desarrolló el código mostrado en el Listado 3. Este código permite solicitar un número al dispositivo a través de comunicación serial.

Listado 3: Solicitud al generador Arduino uno

```
void setup (){
Serial.begin (9600);
}

bucle vacío (){
Serial.write (81); // Envía la semilla inicial "
if (Serial.available ()> 0)
// Comprobamos si hay datos en el búfer
{
 int datos = Serial.read ();
// Leer valor aleatorio
Serial.println (datos);
// Imprimimos el valor aleatorio
}
  }
```

4. RESULTADOS

El algoritmo de Chudnovsky implementado en el sistema embebido Propeller ha sido evaluado con respecto a los tiempos registrados, los cuales se detallan en la Tabla 1. Además, en la figura 5 se muestra el comportamiento lineal del algoritmo.

También se observa que el algoritmo funciona con eficiencia en el contexto del generador propuesto. El Arduino uno es invocado para iniciar el rango de operación del generador, que normalmente va desde 0 hasta 10000, estableciendo así nuevos valores de semilla.

Esto indica que el algoritmo es capaz de generar números aleatorios en un tiempo adecuado para las necesidades del sistema embebido propuesto.

La figura 5 muestra la conexión de prueba para determinar si el sistema integrado está funcionando correctamente.

Fig. 5. Ensamble entre Propeller Parallax P8x32 y Arduino uno.

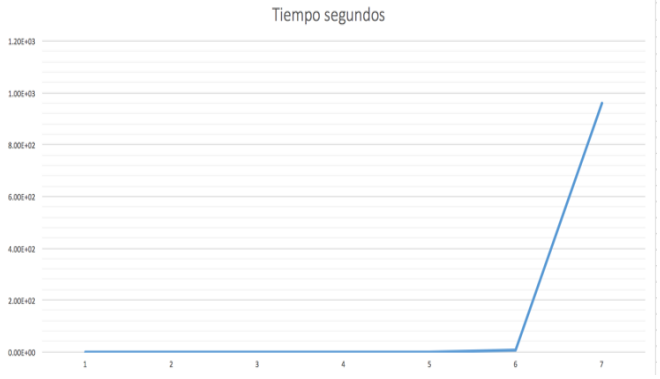
Fuente: Parallax.com.

Tabla 1. Tiempos de ejecución

Decimales	Tiempos segundos
10	3.850E-05
10	2.70E-05
100	0.00216356
1000	0.00307730
10000	0.0768525
100000	9.45337
1000000	100.3

Fuente: elaboración propia.

Fig. 6. Tiempo de ejecución.

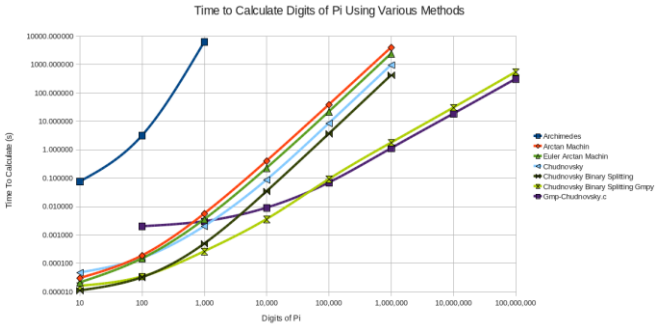


Fuente: elaboración propia

figura 6. El comportamiento del programa implementado en el Propeller Parallax P8x32. La figura 7. muestra la comparación del rendimiento de diferentes métodos para calcular PI.

Como observar, el comportamiento del algoritmo implementado es similar al reportado anteriormente; a pesar de tener recursos limitados en el sistema embebido, los tiempos de ejecución son similares.

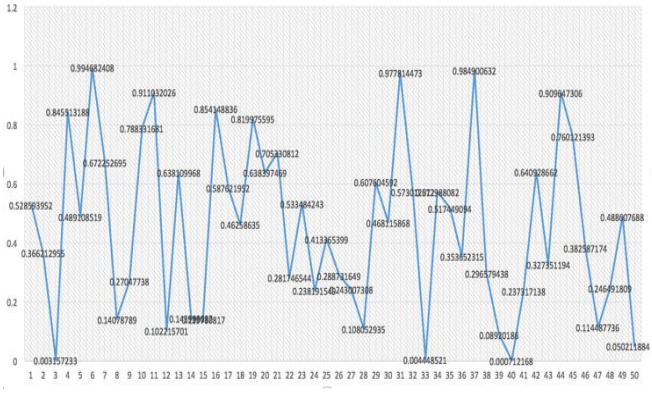
Fig. 7. Tiempos de ejecuciones utilizando varios métodos.



Fuente: elaboración propia

La figura 8 muestra los valores obtenidos para 50 solicitudes del dispositivo.

Fig. 8. Números enviados desde el generador.



Fuente: elaboración propia

Hay algunos métodos disponibles para verificar varios aspectos de la calidad de los números pseudoaleatorios.

5. CONCLUSIONES Y RECOMENDACIONES

Como se puede observar, el uso de Chudnovsky para generar números pseudoaleatorios en un sistema embebido representa un método viable e implementable debido a las características del algoritmo.

Al realizar las pruebas de uniformidad e independencia, los resultados obtenidos fueron sobresalientes, y sus valores indicaron que el generador es robusto, confiable y representó una operación en su complejidad computacional lineal al implementar la tarjeta Propeller Parallax P8x32A.

Este generador ayudará a crear propuestas para el desarrollo de algoritmos genéticos, evolutivos, heurísticos, de Montecarlo y criptográficos.

Recomendaciones

El Propeller Parallax ofrece una plataforma versátil para una amplia gama de aplicaciones embebidas, desde control de hardware hasta desarrollo de juegos y sistemas de automatización. Al seguir las recomendaciones y explorar

ejemplos prácticos, se pueden desarrollar soluciones innovadoras, sistemas de monitoreo y control, sistemas de medición, pruebas control de robots, desarrollo de juegos y robustas

Reconocimiento:
Agradecemos las facilidades otorgadas para la realización de este trabajo al Consejo Nacional de Ciencia y Tecnología (CONACYT), Instituto Politécnico Nacional a través de la Secretaría de Investigación y Posgrado con los proyectos SIP 20230215, SIP 20231468. Además, a la Unidad Interdisciplinaria de Ingeniería y Ciencias Sociales y Administrativas y Centro de Investigación y Desarrollo de Tecnología Digital. Asimismo, el "Programa de estímulos al desempeño de la investigación" (EDI), el "Programa de estímulos al desempeño docente" (EDD) y la "Comisión de Operación y promoción de actividades académicas" (COFAA).

6. REFERENCIAS

[1] Blum, L, Blum, M y Schub M. 1986: Un simple generador de números pseudoaleatorios impredecibles. SIAM Journal of Computing. Vol. 15, no. 2, págs. 364-383.
[2] Chudnovsky, David V.; Chudnovsky, Gregory V. (1989), "The Computation of Classical Constants", Actas de la Academia Nacional de Ciencias de los Estados Unidos de América 86 (21): 8178–8182, DOI: 10.1073 /
[3] Cernák, J. 1996. Generadores digitales del caos. Cartas de física A. vol. 214, págs. 151-160.
Couture, R y L'Ecuyer, P. 1997: Propiedades de distribución de los generadores de números aleatorios de multiplicación con acarreo. Matemáticas de la Computación, vol. 66, pág. 591.
[4] Entacher, Karl. 1998. Subsecuencias erróneas de generadores de números pseudoaleatorios lineales congruentes conocidos. Transacciones ACM sobre Modelado y Simulación por Computadora. Vol. 8, no. 1, págs. 61-70.
[5] Fog, A. 2001. Generadores de números pseudoaleatorios. http://www.agner.org/random.
James, F. 1990. Una revisión de generadores de números pseudoaleatorios. Comunicaciones de Física Informática. Vol. 60, págs. 329-344.
[6] Marsaglia, G., Narasimhan, B. y Zaman, A. 1990. Un generador de números aleatorios para PC. Comunicaciones de Física Informática. Vol. 60, pág. 3. 4. 5.