
Universidad Nacional del Sur

Tesis de Maestría en Ingeniería

# Diseño e implementación de arquitecturas para estructuras paralelas

Ing. Alejandro Pasciaroni

Bahía Blanca Argentina

2015

Impreso en Argentina.
Septiembre de 2014

# Prefacio

Esta tesis se presenta como parte de los requisitos para optar al grado académico de Magister en Ingeniería de la la Universidad Nacional del Sur y no ha sido presentada previamente para la obtención de otro título en esta Universidad u otra. La misma contiene los resultados obtenidos en investigaciones realizadas en el ámbito del Departamento de Ingeniería Eléctrica y Computadoras en el período comprendido entre Abril del 2012 y Agosto del 2014, bajo la dirección del Dr. Pedro Julian y del Dr. Pablo Mandolesi.

Bahía Blanca, 29 de Septiembre de 2015.

Alejandro Pasciaroni
Departamento de Ingeniería Eléctrica y de Computadoras
Universidad Nacional del Sur


Universidad Nacional del Sur
Secretaría General de Posgrado y Educación
Continua

La presente tesis ha sido aprobada el ..../..../..... , mereciendo la calificación de .....(.....)

# Agradecimientos

Me gustaría brindar mis más sinceros agradecimientos al Dr. Pedro Julian y al Dr. Pablo Mandolesi por brindarme su dirección y guía en la exploración de diferentes aspectos del problema y al Dr. Favio Masson por sus constantes aportes brindados al trabajo. También quiero agradecer al Dr. Juan Agustín Rodriguez por la motivación, por los consejos y conocimientos en el diseño e implementación VLSI de las arquitecturas desarrolladas. También quiero agradecer a Martín Di Federico por su ayuda y sus aportes brindados.

Quiero agradecer a todos mis compañeros de oficina que me han brindado su apoyo y con los cuales también he compartido lindas charlas y encuentros.

Quiero agradecer a mi novia María Pia Cardona, mis padres Liliana Fernandez y José Pasciaroni por el apoyo que me brindaron durante el transcurso de la Maestría.

Por último agradezco a la Universidad Nacional del Sur por brindarme la oportunidad de realizar el posgrado y al Consejo Nacional de Investigaciones Científicas y Técnicas por haberme otorgado la beca sin la cual este trabajo no hubiera sido posible.

# Resumen

Este trabajo de investigación explora el diseño e implementación de arquitecturas paralelas que permiten el procesamiento en paralelo de datos. Se consideró, como caso de estudio, el procesamiento en tiempo real del algoritmo del filtro de partículas para aquellas aplicaciones que requieren miles de ellas. En estos casos el algoritmo presenta un cuello de botella en el tiempo de ejecución debido al remuestreo, la única operación del algoritmo cuyo procesamiento no puede ser paralelizado en forma directa. El estudio tuvo como objetivos la revisión bibliográfica sobre los algoritmos de remuestreo e implentación del filtro de partículas y por último la proposición de arquitecturas digitales para un elemento de procesamiento para luego considerar arquitecturas con procesamiento distribuido.

Se revisionaron las estrategias de paralelización del algoritmo de remuestreo y se llevó acabo una evaluación cualitativa y cuantitativa del comportamiento de las mismas. La estrategia seleccionada para las arquitecturas propuestas es el remuestreo distribuido que se basa en la distribución del remuestreo en grupos de partículas. De la evaluación se concluye que si se aumenta la cantidad de partículas por grupo se reduce el error en la estimación pero no sucede lo mismo si se aumenta la cantidad de grupos de igual cantidad de partículas.

Se propusieron tres arquitecturas digitales basadas en el remuestreo distribuido. Las dos primeras arquitecturas se basan en el modelo computacional *Dataflow* y la tercera arquitectura es un arreglo de procesadores de propósito general que integran una arquitectura *Single Instruction Multiple Data* (SIMD). El primer diseño prioriza la tasa de procesamiento mientras que los otros dos el área de silicio requerida. Para reducir el área del elemento de procesamiento se recurrió a la multiplexación en tiempo de ciertos recursos computacionales.

Se realizó un análisis comparativo en términos de tiempo de ejecución y área de silicio de las arquitecturas propuestas. Se observa que el multiplexado en tiempo de recursos resulta exitosa en la reducción del área total. Por otra parte a igual número de grupos de procesamiento instanciados resultará conveniente el Diseño 1 si se prioriza la tasa de procesamiento y el Diseño 2 si la prioridad es minimizar el área de silicio. El Diseño 3 no presenta ventaja respecto al Diseño 1 a pesar de disponer de un diseño regular y un elemento de procesamiento más versátil.

# Abstract

This research work explores the design and implentation of digital architectures that allows parallel data processing. The particle filtering in real time is considered as case study specially for those applications that requires thousands of particles. In those cases the algorithm presents a bottle neck in the execution time of the filter due to the resampling operation which can not be parallelized in a straight way. The study had as objectives the bibliographic revision of resampling algorithms and particle filter implementation and the proposition of digital architectures for processing elements that integrate a distributed processing architecture.

The bibliographic revision of strategies to parallelize resampling algorithms was carried out. Further a quantitative and qualitative evaluation of the strategies was made. The distributed resampling strategy was choosen for the architecture implementations. This strategy is based on the distribution of the resampling operation into groups of particles. From the evalution it is concluded that: the estimation error of the filter is improved by increasing the number of particles per group. However, increasing the number of groups with equal quantity of particles does not reduce the error estimation.

Three digital architectures were proposed based on distributed resampling. The two first architectures are based on the dataflow computational model and the third one is an array of general purpose processors that conforms a Single Instruction Multiple Data architecture (SIMD). First design is focused on maximizing the data processing rate meanwhile the two other designs are focused on reducing the required silicon area. In order to reduce the silicon area a time multiplexing of hardware resources was implemented.

A comparison in terms of execution time and silicon area was carried out for the three proposed architectures. From this analysis is possible to observe taht the time multiplexing of hardware resources was successful in reducing the silicon area. Comparing Design 1 and Design 2 it is concluded that: for an equal number of processing groups instantiated Design 1 results more appropiate when data processing rate is important meanwhile Design 2 is the best option when the design goal is to reduce the silicon area. Finally Design 3 does not presents any advantage compared to Design 1 despite its more versatile processing element and its regular design.

# Índice general

# Capítulo 1

# Introducción

## 1.1. Motivación y contribuciones

El filtro de partículas es un algoritmo para estimación dinámica de estados y permite obtener una aproximación de la función de densidad de probabilidad del estado a partir de partículas pesadas. Para aplicaciones que requieren del filtrado de miles de partículas, el procesamiento en tiempo real es un requisito díficil de lograr dado que el algoritmo es computacionalmente caro y una de sus operaciones, llamada remuestreo, se ejecuta en forma secuencial y se convierte en un cuello de botella en el tiempo de ejecución del filtro.

Existen aplicaciones que requieren de estimaciones en tiempo real de sistemas no lineales y no gaussianos como localización de robots, seguimiento visual [1] [2] [3], localización de fuentes acústicas [4] y seguimiento de personas [5]. Estas aplicaciones utilizan el filtro de partículas pero se requiere de un gran número de partículas para proveer estimaciones precisas.

En la bibliografía actual, existen estrategias de paralelización del remuestreo y arquitecturas de múltiples elementos de procesamiento que tienen como objetivo explotar el paralelismo del algoritmo y permitir el filtrado en tiempo real. Sin embargo estas arquitecturas no resultan del todo escalables con el número de partículas.

En el trabajo de investigación aquí presentado se analizaron las distintas estrategias de paralelización del algoritmo de remuestreo y luego a partir de este análisis se desarrollaron distintas arquitecturas para las tecnologías *Very Large Scale Integrated* (VLSI). Estas arquitecturas escalables con el número de partículas tienen como objetivo el filtrado de partículas en tiempo real.

## 1.2. Metodología de trabajo

La metodología de trabajo adoptada consiste en generar los modelos en punto flotante del algoritmo de filtro de partículas y las estrategias de paralelización del remuestreo en un lenguaje de programación técnico como Matlab. Luego, se desarrollan los mismos modelos en punto fijo y se simulan exhaustivamente para una configuración dada de número de bits. Una vez que se verifica el correcto funcionamiento del modelo en punto fijo se mapean uno a uno las operaciones del algoritmo a módulos de hardware a nivel RTL (del inglés *Register Transfer Level*). Por último, una vez finalizado el modelo RTL, se simula y se verifica que los datos procesados son idénticos a los provistos por el modelo en punto fijo.

Como caso de estudio se consideró el seguimiento de un móvil en dos dimensiones. Por lo tanto se desarrollaron escenarios de simulación en Matlab y se simularon junto con los modelos en punto flotante y fijo con el fin de observar el desempeño de los modelos anteriormente descriptos y asegurar que las estimaciones del filtro no diverjan.

## 1.3. Organización de la memoria

Este documento está organizado en cinco capítulos. En el Capítulo 2 se describe el algoritmo del filtro de partículas, sus aplicaciones más frecuentes y las operaciones involucradas. Se describe también el filtro de partículas para el seguimiento de un móvil en dos dimensiones.

En el Capítulo 3 se introduce la revisión y análisis de las estrategias de paralelización de los algoritmos de remuestreo y su combinación. Se presenta también un modelo de alto nivel para los filtros de partículas paralelos que permite realizar una evaluación de los compromisos entre tiempo de ejecución y recursos de harware de las estrategias.

Las arquitecturas VLSI para el filtrado de partículas en tiempo real que tienen en cuenta una de las estrategias de paralelización del remuestreo se presentan en el Capítulo 4. Se describen los módulos de procesamiento de cada una de las arquitecturas y se presentan también los resultados de la verificación del modelo RTL y el modelo en punto fijo en Matlab. Un análisis basado en la comparación, en términos de área y tiempo de ejecución, de las tres arquitecturas propuestas se presenta al final del capítulo, lo que permite tener conocimiento del alcance de cada una de las arquitecturas.

Las conclusiones y el trabajo futuro se resumen en el Capítulo 5.

## 1.4. Trabajos publicados

- A. Pasciaroni, S. Sañudo, J. A. Rodríguez, F. Masson y P. Julián “Modelling and Analysis of Parallel Particle Filters” en la XV Reunión de Trabajo en Procesamiento de la Información y Control, vol. 1, nro. 1, 2013, pp. 1-6.
- A. Pasciaroni, J. A. Rodríguez, F. Masson, P. Julián y E. Nebot “VLSI Architecture Design for Particle Filtering in Real-Time” en Escuela de Microelectrónica Tecnología y Aplicaciones, vol. 1, nro. 1, 2014, pp. 1 -7.

# Capítulo 2

# Filtro de partículas

En este capítulo se realiza una revisión del algoritmo del filtro de partículas y los algoritmos de remuestreo. El capítulo comienza con la descripción del algoritmo para el filtro de partículas y luego los algoritmos de remuestreo presentes en la bibliografía. Por último, se presenta el caso de estudio adoptado en este trabajo.

## 2.1. Algoritmo del filtro de partículas

El filtro de partículas [6] es un método para realizar estimación estadística y dinámica de estados. La función de densidad de probabilidad de un estado dado se representa por un conjunto de partículas pesadas, las cuales son actualizadas en forma iterativa de acuerdo a mediciones de sensores y un modelo dinámico del sistema.

Si las distribuciones son gaussianas y los modelos son lineales, el filtro de Kalman [7] realiza estimaciones de la media y covarianza de la función de densidad de probabilidad en forma óptima. Sin embargo, en todos aquellos casos en que los modelos son no lineales y las distribuciones de probabilidad son no gaussianas, sus estimaciones ya no son óptimas e incluso pueden diverger. De hecho, en estos casos el filtro de partículas genera una aproximación a la función de densidad de probabilidad con un mejor desempeño que el filtro de Kalman. Existen otras soluciones aproximadas como el filtro de sumas de gaussianas [8], o la evaluación de la función de densidad sobre una grilla en el espacio de estados [9].

El foco de este trabajo se sitúa en el filtro de partículas, el cual se basa en la generación recursiva y posterior evaluación de muestras de la distribución de los estados no conocidos del sistema. El algoritmo de partículas básico llamado « Sequential Importance Sampling» (SIS) se compone de dos operaciones fundamentales: muestreo y actualización de pesos. En la operación de muestreo cada partícula se procesa a través del modelo dinámico, el cual describe la evolución

del sistema en el tiempo. La operación de actualización de pesos modifica el peso asociado a cada partícula tomando en consideración la medición y el modelo del sensor, el cual describe la relación entre la medición y el estado del sistema. Una vez que estos dos pasos fueron realizados, la aproximación de la función de densidad de probabilidad es actualizada y se pueden computar nuevas estimaciones.

El principal problema del algoritmo SIS es la degeneración de los pesos. Luego de algunas iteraciones del algoritmo, sólo una partícula tiene peso importante y el resto tiene peso casi nulo, lo que produce una representación pobre de la función de densidad de probabilidad. Una forma de reducir este efecto es introducir el paso de remuestreo cuya función es reemplazar las partículas con peso pequeño por aquellas con peso mayor. A este nuevo filtro se lo llama «Sampling Importance Resampling» [10]. Sin embargo, los algoritmos de remuestreo actuales [11] [12] pueden deteriorar la diversidad de partículas y desde el punto de vista de implementación imponen un obstáculo a la paralelización del filtro.

A continuación se realiza una descripción del problema de seguimiento utilizando el filtro de partículas.

Un sistema cuyo estado dinámico se representa por un conjunto de variables de estados $x_{k-1} \in \mathbb{R}^n$ para un instante de tiempo $k-1$, evoluciona en el tiempo de acuerdo al modelo dinámico del sistema

$$x_k = f_k(x_{k-1}, v_{k-1}) \tag{2.1}$$

donde $f_k : \mathbb{R}^n \times \mathbb{R}^l \to \mathbb{R}^n$ es el modelo dinámico del sistema que puede ser una función no lineal y $v_{k-1} \in R^l$ es un vector aleatorio cuya función de densidad de probabilidad se asume conocida.

El objetivo del seguimiento de un objeto, entendiendo como objeto un móvil, determinados pixeles de una imágen o cualquier parámetro dentro de un sistema, de acuerdo a las reglas bayesianas, es estimar en forma recursiva el estado $x_k$ teniendo en cuenta todas las mediciones obtenidas hasta el instante $k$. La ecuación que modela las mediciones es

$$z_k = h_k(x_k, n_k), \tag{2.2}$$

donde $h_k : \mathbb{R}^p \times \mathbb{R}^r \to \mathbb{R}^p$ puede ser una función no lineal, $z_k \in \mathbb{R}^p$ es el vector de mediciones y $n_k \in R^r$ es un vector de componentes aleatorias que pueden ser muestras de una distribución no gaussiana. Los vectores $v_k$ y $n_k$ representan respectivamente las incertidumbres del modelo dinámico y de las mediciones.

El conjunto de todas las mediciones obtenidas hasta el instante $k$ se nota como $z_{1:k} = \{z_1, z_2, \ldots, z_k\}$. Entonces el objetivo del seguimiento según las reglas bayesianas es la función de densidad de probabilidad del estado del sistema $x_k$ dado todas las mediciones obtenidas hasta

el instante $k$, es decir $p(x_k \mid z_{1:k})$.

Tal como se dijo al inicio de esta sección el filtro de partículas permite aproximar la función de densidad de probabilidad $p(x_k \mid z_{1:k})$ a través de un conjunto de partículas pesadas, las cuales son actualizadas en forma iterativa.

El algoritmo realiza tres operaciones básicas por iteración: muestreo, actualización de pesos y remuestreo. Para un instante de tiempo $k-1$ el filtro trabaja con un conjunto de $N$ valores $\{x_{k-1}(i), w_{k-1}(i)\}$, $i = 1, \ldots, N$, donde $x_{k-1}(i)$ es un vector probable de estados del sistema con índice $i$, o lo que es lo mismo es una muestra del espacio de estados y $w_{k-1}(i)$ es su peso asociado. Estas muestras, que también se denominan partículas, se encuentran aproximadamente distribuidas de acuerdo a la función de densidad de probabilidad $p(x_{k-1} \mid z_{1:k-1})$.

Las operaciones de muestreo y actualización de pesos se basan en el modelo dinámico del sistema y de las mediciones.

Cuando una medición $z_k$ está disponible, se computan las operaciones del filtro de partículas. La operación de muestreo procesa las partículas utilizando el modelo dinámico. Como resultado se obtienen muestras de la función de densidad de probabilidad $p(x_k \mid x_{k-1})$

$$x_k(i) = f_k(x_{k-1}(i), v_{k-1}(i)), i = 1, \ldots, N. \tag{2.3}$$

Luego, la operación de actualización de pesos computa la siguiente ecuación

$$w_k^*(i) = w_{k-1}(i) \cdot p(z_k \mid x_k(i)), i = 1, \ldots, N, \tag{2.4}$$

la cual involucra la evaluación de la función de verosimilitud $p(z_k \mid x_k)$ que expresa la probabilidad de $z_k$ dado $x_k$, para cada muestra obtenida de la operación de muestreo. Dicha función está perfectamente definida por el modelo de mediciones $h_k$.

Para representar la función de densidad de probabilidad se utilizan los pesos normalizados por lo que se requiere de la operación de normalización

$$w_k(i) = \frac{w_k^*(i)}{\sum_1^N w_k^*(i)}, i = 1, \ldots, N. \tag{2.5}$$

Los algoritmos de remuestreo replican las partículas con peso grande y descartan aquellas que tienen peso pequeño. Al final de la operación de remuestreo el número total de partículas permanece constante y los pesos toman todos el mismo valor:

$$w_k(i) = 1/N, i = 1, \ldots, N. \tag{2.6}$$

La función de densidad de probabilidad $p(x_k \mid z_{1:k})$, se aproxima entonces como

$$p(x_k \mid z_{1:k}) \approx \sum_{1}^{N} w_k(i) \cdot \delta(x - x_k(i)), \tag{2.7}$$

donde $\delta(.)$ es la función Delta de Dirac y $w_k(i)$ corresponde a la ecuación 2.5. A medida que el número de partículas $N$ crece la aproximación se acerca a la función real.

## 2.2. Algoritmos de remuestreo

La función de los algoritmos de remuestreo es eliminar partículas que no contribuyen en forma significativa a la representación de la función de densidad de probabilidad y potenciar aquellas que si lo hacen. Entonces el remuestreo evita la degeneración del filtro en el sentido de que haya sólo una partícula representativa. Sin embargo, puede resultar también que luego de varias iteraciones, el filtro pierda diversidad en sus partículas debido a que existen muchas copias de una sola partícula.

Las operaciones del algoritmo del filtro de partículas se pueden representar gráficamente, lo cual resulta útil para lograr un mayor entendimiento de las mismas. La Fig. 2.1 muestra como las operaciones modifican a las partículas y sus pesos. Se considera un espacio de estados de una dimensión, $x_{k-1} \in \mathbb{R}$ y un total de diez partículas. Las partículas se representan como círculos y su peso el diámetro. Al inicio del algoritmo las partículas se inicializan de acuerdo a una distribución conocida. En esta instancia todas las partículas poseen el mismo peso y es igual a $1/N$. A continuación se realiza la operación de muestreo por lo que las partículas se desplazan de acuerdo al modelo dinámico y su incertidumbre. Luego, se realiza la actualización de pesos teniendo en cuenta la función de verosimilitud. Dependiendo del valor de la función algunas partículas adquieren peso significativo o lo contrario. El remuestreo se encarga de descartar estas últimas y replicar aquellas con mayor peso. Notar que una vez realizado el remuestreo varias partículas se encuentran en la misma posición y su peso se iguala a $1/N$. Al comenzar una segunda iteración, las partículas son desplazadas nuevamente y se distribuyen de acuerdo a la función de densidad de probabilidad $p(x_k \mid x_{k-1})$.

En lo que sigue se revisaran diferentes algoritmos de remuestreo, algunos de ellos modificados para operar con pesos no normalizados. Esto presenta una ventaja desde el punto de vista de implementación ya que se evita el cómputo de la suma acumulada de pesos $W = \sum_{i=1}^{N} w_k^*(i)$ y también $N$ operaciones de división, donde $N$ es el número total de partículas.

Existen otros algoritmos de remuestreo tales como el remuestreo parcial [13] y selección local [14]. Sin embargo los aquí descriptos son los más utilizados en la bibliografía actual.

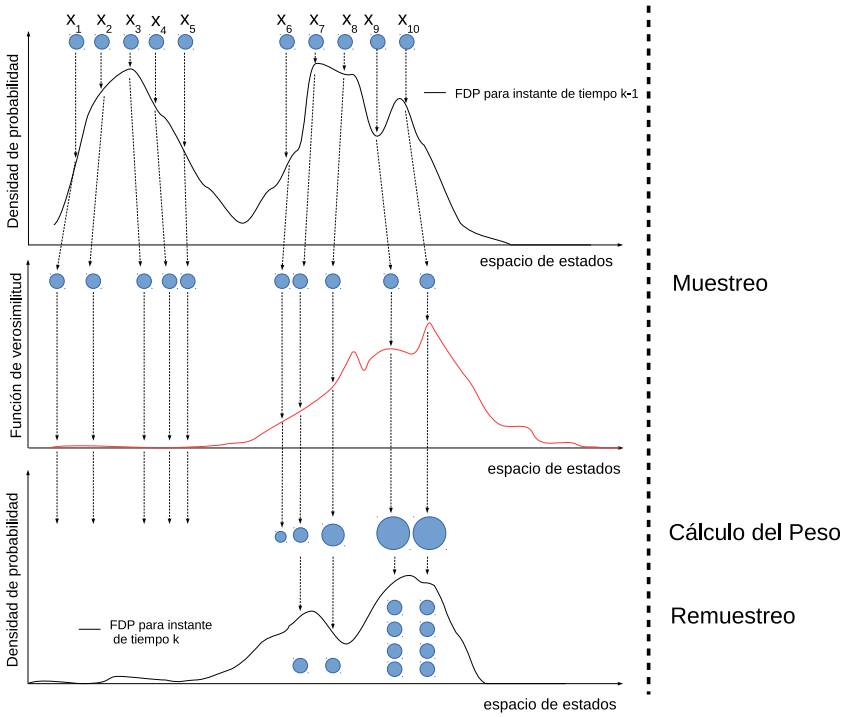


Figura 2.1: Representación gráfica de las operaciones del filtro de partículas.

### 2.2.1. Remuestreo sistemático

El algoritmo de remuestreo sistemático (RS) [11] se muestra en el siguiente pseudo-código:

$u \sim U(0, \frac{1}{N}]$;
$c = 0$;
**for** $j \leftarrow 1$ ***to*** $N$ **do**
  $k = 0$;
  $c(j) = c(j) + w_k(j)$;
  **while** $(c(j) > u)$ **do**
    $k = k + 1$;
    $u = u + \frac{1}{N}$;
  **end**
  $r(j) = k$;
**end**

donde $w_k(j)$ es el peso normalizado de la partícula $x_k(j)$. El algoritmo inicializa el valor de $u$ con una muestra aleatoria de una distribución uniforme $U(0, \frac{1}{N}]$ de manera tal de introducir incertidumbre. Se computa la suma acumulada de pesos $c(j) = \sum_{i=1}^{j} w_k(i)$ y se compara con $u$. Mientras $c(j)$ sea mayor que $u$ se incrementa el factor de replicación $k$ en 1 y el valor de $u$

se actualiza como $u = u + \frac{1}{N}$. A la salida del ciclo *while* se obtiene como resultado el factor de replicación $k$ de la partícula $x_k(j)$. Este factor se define como la cantidad de veces que $u$ está contenido en el intervalo $(c(j-1), c(j))$. El valor de $k$ se almacena en la posición $j$ del arreglo de replicación $r$.

Las partículas deben ser replicadas de acuerdo a los valores almacenados en el arreglo $r$, el pseudo-código del algoritmo para tal fin se muestra a continuación.

```
l = 0;
for j ← 1 to N do
    if r(j) ≠ 0 then
        for i ← 1 to r(j) do
            x̃_k(l) = x_k(j);
            l = l + 1;
        end
    end
end
```

donde $\tilde{x}_k(l)$ es la partícula remuestreada. Si el factor de replicación $r(j)$ de la partícula $x_k(j)$ es distinto de cero, entonces la misma se copia $r(j)$ veces en el arreglo de partículas remuestreadas.

La principal ventaja de este algoritmo es su buen desempeño. Sin embargo, desde el punto de vista de implementación presenta algunas dificultades. En principio, el algoritmo cuenta con dos ciclos anidados, siendo la cantidad de iteraciones del ciclo *while* variable lo cual dificulta su implementación tipo *pipeline* dado que requiere de lógica de control compleja.

### 2.2.2. Remuestreo sistemático residual

El remuestreo sistemático residual (RSR) [13] posee ventajas con respecto al remuestreo sistemático, dado que posee un solo ciclo y la cantidad de iteraciones es fija y conocida. El pseudo código se muestra a continuación para un conjunto de $N$ partículas y pesos no normalizados.

```
u ~ U(0, 1];
K = N/W;
for j ← 1 to N do
    temp = (w*_k(j) · K) − u;
    r(j) = ⌈temp⌉;
    u = temp − r(j);
end
```

Al inicio el valor de $u$ se inicializa con una muestra aleatoria de una distribución uniforme $U(0, 1]$, lo que al igual que el remuestreo sistematico, introduce incertidumbre en la ejecución

```
l = 0;
for j ← 1 to N do
    if r(j) ≠ 0 then
        for i ← 1 to r(j) do
            x̃_k(l) = x_k(j);
            l = l + 1;
        end
    end
end
```

del algoritmo. La constante $K$ es igual a $N/W$ donde $W = \sum_{j=1}^{N} w_k^*(i)$. En cada iteración del ciclo $for$ el algoritmo calcula el factor de replicación de la partícula $x_k(j)$. Para tal fin, calcula la variable temporal $temp$ como la diferencia entre peso no normalizado $w_k^*(j)$, escalado por la constante $K$, y el valor de $u$. La función $\lceil x \rceil$ se define como $min\{k \in \mathbb{Z} | x \leq k\}$, por lo que el factor de replicación $r(j)$ es el mínimo entero superior a la variable $temp$. Se calcula un nuevo valor de $u$ como la diferencia entre el valor real $temp$ y el entero $r(j)$. Luego se realiza la copia de las partículas en función de los valores almacenados en el arreglo $r$ al igual que el algoritmo anterior.

Un ejemplo de la ejecución del remuestreo sistemático residual se muestra en la Tabla 2.1. La primer columna identifica la iteración del algoritmo sobre la partícula a remuestrear $x_k(j)$ y su peso no normalizado $w_k^*(j)$. El resultado de la iteración, es decir la partícula remuestreada $\tilde{x}_k$, se muestra en la última columna. La replicación de las partículas 1 y 5 es previsible dado que son las que poseen mayor peso.

Tabla 2.1: Ejemplo de ejecución del remuestreo sistemático residual.

| j | $x_k(j)$ | $w_k^*(j)$ | $u$ | $temp$ | $r$ | $\tilde{x}_k(j)$ |
|---|---|---|---|---|---|---|
| 1 | 1 | 6 | 0.0357 | 2.6916 | 3 | 1 |
| 2 | 3 | 0.1 | 0.3084 | -0.2630 | 0 | 1 |
| 3 | 2 | 0.5 | 0.2630 | -0.0357 | 0 | 1 |
| 4 | 5 | 3 | 0.0357 | 1.3279 | 2 | 5 |
| 5 | 10 | 1.4 | 0.6721 | -0.0357 | 0 | 5 |

### 2.2.3. Remuestreo IMH

El pseudo código del algoritmo de remuestreo « Independent Metropolis Hastings» [12] requiere de la generación de $N$ números pseudo aleatorios y también de la operación de división y de mínimo. Sin embargo, si se encuentra una simplificación en la generación de la variable $\alpha$ puede resultar en un algoritmo muy beneficioso desde el punto de vista de su implementación. Una versión modificada del algoritmo que logra reducir su costo computacional se describe en

el Capítulo 4.

$\tilde{x}_k(1) = x_k(1)$;
$\tilde{w}_k(1) = w_k^*(1)$;
**for** $j \leftarrow 2$ ***to*** $N$ **do**
  $u \sim U[0,1]$;
  $\alpha(\tilde{x}_k(j-1), x_k(j)) = \text{mín}\{1, \frac{w_k(j)}{\tilde{w}_k(j-1)}\}$;
  **if** $u \leq \alpha(\tilde{x}_k(j-1), x_k(j))$ **then**
    $\tilde{x}_k(j) = x_k(j)$; $\tilde{w}_k(j) = w_k^*(j)$ ;
  **else**
    $\tilde{x}_k(j) = \tilde{x}_k(j-1)$; $\tilde{w}_k(j) = \tilde{w}_k(j-1)$;
  **end**
**end**

Al comienzo del algoritmo, la primer partícula $x_k(1)$ se copia en $\tilde{x}_k(1)$. También se copia el peso de la partícula remuestrada $w_k^*(1)$ en $\tilde{w}_k(1)$. En cada iteración el valor de $u$ se obtiene de una muestra aleatoria de una distribución uniforme $U[0,1]$. Se calcula $\alpha$ como el valor mínimo entre dos valores: 1 y el resultado de la división entre el peso de la partícula a remuestrear $w_k^*(j)$ y de la última partícula remuestreada $\tilde{w}_k(j-1)$. Si el valor de $u$ es menor o igual a $\alpha(\tilde{x}_k(j-1), x_k(j))$ entonces se copia en $\tilde{x}_k(j)$ y $\tilde{w}_k(j)$ la partícula $x_k(j)$ y su peso $w_k(j)$ respectivamente. En caso contrario se copia la partícula previamente remuestreada $\tilde{x}_k(j-1)$ junto con su peso $\tilde{w}_k(j-1)$.

### 2.2.4. Complejidad computacional de los algoritmos de remuestreo

La Tabla 2.2 muestra la complejidad computacional de los algoritmos de remuestreo descriptos. Se debe notar que para el análisis se consideró la operación comparación y mínimo como una resta.

Tabla 2.2: Costo Computacional de los algoritmos de remuestreo.

| *Algoritmo* | *Generación de números pseudo aleatorios* | *Multiplicaciones* | *Sumas* | *Divisiones* | *Operación con pesos no normalizados* |
|---|---|---|---|---|---|
| RS | 1 | 0 | $4 \cdot N - 1$ | 0 | *No* |
| RSR | 1 | $N$ | $3 \cdot N - 1$ | 1 | *Si* |
| IMH | $N$ | 0 | $2 \cdot N - 1$ | $N$ | *Si* |

En la tabla se detalla el peor caso en la cantidad de operaciones suma para el remuestreo sistemático, en el que una sola partícula es remuestreada $N$ veces.

De los tres algoritmos presentados el remuestreo sistemático residual posee el menor costo computacional. A pesar de que requiere $N$ multiplicaciones, las cuales tienen un costo en área y tiempo de ejecución mayor que la operación suma, el remuestreo sistemático opera con pesos normalizados, lo que requiere de $N-1$ sumas y $N$ divisiones. Lo mismo sucede con el algoritmo

IMH, si bien tiene un costo computacional menor que el algoritmo sistemático, su principal desventaja se debe a las $N$ operaciones de divisón.

Los algoritmos mencionados en este trabajo son de ejecución secuencial. Si el problema requiere una gran cantidad de partículas para poder lograr estimaciones precisas la operación de remuestreo resultará en un cuello de botella en el tiempo de ejecución del filtro dado que no es posible paralelizarla en forma directa. Esto último se debe a la dependencia de datos que presentan todos los algoritmos de remuestreo presentes en la bibliografía actual. En cada iteración la actualización de sus variables depende del valor que poseían en la iteración anterior. En el caso del remuestreo sistemático, la actualización de la suma acumulada de pesos $c$ depende del valor que adquiere en la iteración anterior. Por otra lado, en el remuestreo sistemático residual la variable *temp* depende del valor de $u$ adquirido en la iteración previa. Por último, en el remuestreo IMH el valor de $\alpha$ para la iteración $j$ es dependiente de $\tilde{w}_k(j-1)$. A raíz de esta problemática, en el Capítulo 3 se analizan distintas estrategias para la paralelización del remuestreo.

## 2.3. Aplicación al problema de seguimiento

Como caso de estudio se consideró el seguimiento de un móvil en dos dimensiones, por lo que a continuación se describe el algoritmo de filtro de partículas para esta problemática.

El modelo dinámico utilizado es de velocidad del móvil casi constante. Las ecuaciones que lo definen para un instante de tiempo $k$ son:

$$x_k = \Phi \cdot x_{k-1} + \Gamma \cdot \begin{bmatrix} v_x \\ v_y \end{bmatrix}, \tag{2.8}$$

donde $\mathrm{x}_k = \begin{bmatrix} x & \dot{x} & y & \dot{y} \end{bmatrix}'$ siendo $x$ e $y$ las coordenadas de la posición en el plano y $\dot{x}$ y $\dot{y}$ la velocidad del móvil en cada eje.

$$\Phi = \begin{bmatrix} 1 & \Delta T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta T \\ 0 & 0 & 0 & 1 \end{bmatrix}, \Gamma = \begin{bmatrix} 0{,}5 \cdot \Delta T^2 & 0 \\ \Delta T & 0 \\ 0 & 0{,}5 \cdot \Delta T^2 \\ 0 & \Delta T \end{bmatrix}, \tag{2.9}$$

donde $v_x$ y $v_y$ son muestras de una distribución dada.

El modelo de las mediciones $h_k$ depende del tipo de sensor que se utilice. A lo largo del trabajo se utilizaron distintos escenarios de simulación donde la principal diferencia reside en el modelo de las mediciones utilizado. En general se utilizaron modelos de sensores que permiten

medir el rango de un móvil por lo que se requiere del cómputo de la distancia euclidiana de la partícula.

Si se considera que el modelo probabilístico del sensor es gaussiano con varianza ${\sigma_k}^2$ y $p_k^i$ es la $i$-ésima partícula, $p_k^i = \begin{bmatrix} x^i & \dot{x}^i & y^i & \dot{y}^i \end{bmatrix}'$, es decir un posible valor de $x_k$, entonces el algoritmo del filtro de partículas se describe como:

*Inicialización aleatoria de las partículas;*
**for** $i \leftarrow 1$ ***to*** $N$ **do**
  $p_k^i = f(p_{k-1}^i, v_x, v_y)$;
  $d^i = sqrt((x^i)^2 + (y^i)^2)$;
  $pred_k = h_k(d^i)$;
  $w^i = \frac{1}{\sqrt{2\pi \cdot \sigma_k^2}} \cdot \exp(\frac{-(pred_k - z_k)^2}{2 \cdot \sigma_k^2})$
**end**
$[\hat{w}, \hat{p}] = remuestreo(w, p_k)$;

El algoritmo realiza el muestreo de la partícula $p_{k-1}^i$ dando como resultado $p_k^i$. Se computa la distancia eucludiana $d^i$ de la partícula $p_k^i$ al orígen del sistema de coordenadas. Luego se utiliza el modelo de medición $h_k$ para obtener una medición aproximada $pred_k$ que provería el sensor de encontrarse el móvil en la posición indicada por la partícula $p_k^i$. Por último, se computa el peso de la partícula $w^i$ utilizando $pred_k$ y la medición real provista por el sensor $z_k$. Como se considera que el modelo probabilístico del sensor es gaussiano se computa la función normal $\frac{1}{\sqrt{2\pi \cdot \sigma_k^2}} \cdot \exp(\frac{-(pred_k - z_k)^2}{2 \cdot \sigma_k^2})$.

En la operación de *remuestreo* se utiliza cualquiera de los algoritmos previamente descriptos sobre el conjunto $w = \{w^j : j = 1, \ldots, N\}$ y $p_k = \{p_k^j : j = 1, \ldots, N\}$ dando como resultado el conjunto de partículas remuestreadas $\hat{p}$ y sus pesos $\hat{w}$.

## 2.4. Resumen

Se realizó una revisón del algoritmo de filtro de partículas y sus principales operaciones. Luego se describieron los algoritmos de remuestreo presentes en la bibliografía y por último se presentó el caso de estudio adoptado para este trabajo.

# Capítulo 3

# Análisis y modelado del filtro de partículas

En este capítulo se realiza una revisión de las estrategias de paralelización de los algoritmos de remuestreo y su combinación. Primero se describen las estrategias más utilizadas en la bibliografía. Luego se presenta un modelado de las estrategias, lo cual permite una evaluación cualitativa. Por último se realiza un análisis cuantitativo por medio de simulaciones. Este análisis permite visualizar el alcance de las mismas en cuanto a reducción en el tiempo de ejecución.

## 3.1. Revisión de las estrategias de paralelización de la operación de remuestreo

El problema de la paralelización del filtro de partículas (FP) ha sido investigado por varios autores [15],[16] que recurren a dos estrategias fundamentales para acelerar la operación de remuestreo: 1) distribuir la operación entre grupos de partículas permitiendo una evaluación paralela; 2) realizar una distribución local y un remuestreo localizado para cada partícula permitiendo una reducción en el número total de partículas.

A partir de estos trabajos es difícil evaluar claramente la relevancia y efectividad de cada estrategia dado que si bien se utiliza un escenario en común [6], no se observan configuraciones en común en términos del número total de partículas, el agrupamiento y la cantidad de partículas generadas en las distribuciones locales. Esta situación se agrava aún más por las diferentes plataformas de harware utilizadas [17] [18] [19] [20].

En [15] se introducen varios métodos para distribuir la operación de remuestreo en elementos de procesamiento (EP) idénticos. Si $N$ es el número total de partículas, cada EP procesa $N/P$ partículas donde $P$ es el número de elementos. Luego, aproximaciones locales (provistas por

cada EP) de la función de densidad de probabilidad a posteriori se combinan para formar una aproximación global. Sin embargo, dado que estas aproximaciones pueden estar alejadas de la verdadera solución, existe una pérdida de desempeño si se compara con el algoritmo de filtro de partícula estándar. Con el fin de reducir esta pérdida se introduce un intercambio de partículas entre EPs. Los autores proponen dos estrategias para realizar el remuestreo distribuido llamadas RPA y RNA.

La estrategia RPA requiere de una comunicación no determinística y consiste en realizar el remuestreo en dos etapas. Primero, se calcula la suma acumulada de pesos de cada EP. Una vez conocido el valor de las sumas, se realiza un remuestreo entre EPs con el fin de definir la cantidad de partículas que cada uno debe replicar. Por último, en base a esas cantidades se realiza un remuestreo dentro de cada elemento de procesamiento. Finalizada esta última operación, puede resultar que algunos EPs contengan más partículas que otros, por lo que una comunicación entre ellos debe realizarse a fin de igualar el número de partículas que cada uno contiene. El desempeño de este algoritmo es igual al del remuestreo secuencial pero su principal desventaja es que requiere de una comunicación global entre EPs lo cual afecta la escalabilidad de la arquitectura.

La estrategia RNA si bien implica un control de comunicación más simple y escalable, introduce una degradación en el desempeño del filtro. La estrategia consiste en agrupar EPs y realizar el remuestreo distribuido dentro de cada grupo. Por último se realiza una comunicación determínistica entre grupos. Los autores desarrollan tres maneras de realizar el intercambio de partículas: agrupamiento fijo, agrupamiento adaptivo y RNA con intercambio local. Tal como su nombre lo sugiere el agrupamiento fijo implica que los grupos ya están fijos y predeterminados y el remuestreo dentro de cada grupo se realiza con el algoritmo RPA. El agrupamiento adaptivo se diferencia del anterior en la manera de generar los grupos, los cuales se forman considerando los pesos relativos de cada EP. Por último, en el algoritmo RNA con intercambio local, el remuestreo se realiza localmente en cada EP y luego se realiza un intercambio fijo y predeterminado de partículas entre EPs vecinos. Desde el punto de vista de implementación, esta última manera de realizar el intercambio resulta la más simple y escalable con el número de EPs.

En [21] se presenta una optimización en el procedimiento de intercambio de partículas. Aplicando la divergencia Kullback-Leibler los autores demuestran que el intercambio de partículas de mayor peso entre EPs adyacentes, en el esquema RNA, resulta en un rendimiento mayor que el obtenido al aplicar un intercambio aleatorio.

En [22] se presentan varios procedimientos de intercambio. En particular el esquema llamado «Full Random Mixing» provee una precisión en la tarea de seguimiento comparable con el FP estándar pero su implentación es compleja.

En [23] se presenta una técnica llamada «Finite-Redraw-Importance» (FRIM) que se deriva

del procedimiento propuesto en [6] para mejorar la operación de muestreo. Esta técnica realiza una prueba de aceptación para cada partícula muestreada con el fin de identificar si se encuentra cerca de alguna región de probabilidad alta de la función de densidad. En caso de que la prueba de resultado negativo, la partícula se muestrea nuevamente. Cuando esta técnica se aplica al FP se obtienen estimaciones más precisas. La combinación de esta técnica con el esquema RNA también se presenta pero no se aplica algún intercambio de datos entre EPs. Al no existir comunicación es esperable una pérdida en la diversidad de partículas y además existe la posibilidad de que ciertos EPs evaluen partículas sin impacto en la aproximación de la función de densidad de probabilidad.

En [16] se presenta el algoritmo filtro de partículas multi-predicción (MP-PF por el inglés *Multi Prediction Particle Filter*). Siguiendo el mismo concepto que el procedimiento de [6] y la técnica FRIM, este algoritmo mejora la operación de muestreo. Esto permite una reducción en el número total de partículas lo cual resulta en una aceleración de la operación de remuestreo pero conservando precisión. El incremento en la complejidad de la operación de muestreo no introduce grandes costos adicionales en el tiempo de ejecución dado que es una operación que no presenta dependencia de datos y puede ser evaluada en forma paralela. Sin embargo, la operación de remuestreo se realiza en forma centralizada lo cual limita el paralelismo alcanzable.

## 3.2. Análisis de modelos de filtros de partículas paralelos

La implementación de FP paralelos recurren a dos estrategias fundamentales para acelerar la operación de remuestreo: (1) distribuir la operación entre grupos de partículas permitiendo una evaluación en paralelo, y (2) realizar la operación de muestreo mejorada lo cual es equivalente a un remuestreo localizado por cada partícula, permitiendo una reducción en el número total de partículas.

La primera estrategia acelera el algoritmo de remuestreo al distribuir el número total de partículas en grupos. Sin embargo, existe un número mínimo de partículas que cada grupo debe tener por lo que el número de grupos tiene un límite. Un grupo pequeño puede resultar en una pérdida de diversidad en el conjunto de partículas y su estimación individual puede no contribuir significativamente a la estimación global.

Una manera de reducir este efecto es realizar un intercambio de partículas entre grupos [15] [22]. Este intercambio puede ser determinístico o no en términos de la cantidad de partículas que se envían de un EP a otro y en términos de la cantidad de EPs involucrados. Los intercambios determinísticos [21] presentan un menor tiempo de ejecución y requisitos de implementación más simples dado que el número de partículas a intercambiarse y sus destinos están fijados. Su principal desventaja es que puede que no sea del todo exitoso en la tarea de evitar la pérdida

de diversidad en el conjunto de partículas. Los métodos no determinísticos [22] toman en consideración el peso relativo de las partículas entre grupos y el rendimiento logrado es similar al FP secuencial. El número de partículas y la dirección de la comunicación varía de iteración en iteración. La complejidad en la selección de datos y los requisitos de transferencias se incrementa afectando la escabilidad de este tipo de intercambio. Siguiendo el caso de estudio descripto en la Sección 2.3, el pseudo código del filtro de partículas con remuestreo distribuido se muestra a continuación.

*Inicialización aleatoria de las partículas;*
**for** $i \leftarrow 1$ ***to*** $P$ **do**
  $limInf = S * (i-1) + 1$;
  $limSup = i * S$;
  **for** $j \leftarrow limInf$ ***to*** $limSup$ **do**
    $p_k^j = f(p_{k-1}^j, v_x, v_y)$;
    $d^j = sqrt((x^j)^2 + (y^j)^2)$;
    $pred_k = h_k(d^j)$;
    $w^j = \frac{1}{\sqrt{2\pi \cdot \sigma_k^2}} \cdot \exp(\frac{-(pred_k - z_k)^2}{2 \cdot \sigma_k^2})$
  **end**
  $[\hat{w}, \hat{p}] = remuestreo(w, p_k)$;
**end**
*comunicación entre grupos;*

donde $S$ es el número de partículas por grupo y $P$ es el número de grupos. Este algoritmo primero calcula el límite inferior $limInf$ y el límite superior $limSup$ del índice de las partículas para cada grupo. Luego, para cada partícula de grupo computa el muestreo y cálculo de peso. Por último, se realiza el remuestreo sobre el conjunto $w = \{w^j : j = limInf \dots limSup\}$ y $p_k = \{p_k^j : j = limInf \dots limSup\}$. Esta última operación da como resultado el conjunto de partículas de grupo remuestreadas $\hat{p}$ y sus pesos $\hat{w}$. Una vez que se finaliza el procesamiento descripto para todos los grupos se procede el intercambio de partículas.

La diferencia entre este algoritmo y el descripto en la Sección 2.3 es el hecho de que las operaciones del filtro de partículas se realizan por grupos y no en el conjunto total de partículas. Lo que lleva también a añadir el intercambio de partículas por las razones previamente descriptas en esta sección.

La versión aquí presentada tiene dos ciclos *for* anidados. Sin embargo, otra posibilidad es dos ciclos *for* no anidados, uno para las operaciones de muestreo y actualización de pesos y otro para el remuestreo distribuido. Sin embargo, la implementación de esta segunda versión tiene un costo en recursos computacionales mucho mayor.

La segunda estrategia se basa en realizar un remuestreo localizado (RL) de cada partícula. Es

equivalente a generar una distribución local de $M$ partículas nuevas alrededor de cada partícula original y luego conservar la más representativa. Como resultado, el conjunto total de partículas genera una aproximación más cercana a la función de densidad de probabilidad y las estimaciones resultan más precisas. Es por esto que esta estrategia permite una reducción en el número total de partículas con la consecuente reducción en el tiempo de ejecución de la operación de remuestreo, conservando la misma precisión. Sin embargo, cuando el valor de $M$ crece el costo computacional es alto. Existe también un incremento en los recursos de hardware dado que es necesario computar en paralelo un remuestreo local en cada partícula para reducir el tiempo de ejecución total del filtro, de otra manera la operación muestreo y remuestreo local dominarían el tiempo de ejecución. El pseudo código para el caso de estudio descripto en la Sección 2.3 se presenta a continuación donde $M$ es el número de partículas generadas en la distribución local.

*Inicialización aleatoria de las partículas;*
**for** $i \leftarrow 1$ ***to*** $N$ **do**
  $p_k^i = f(p_{k-1}^i, v_x, v_y)$;
  $d^i = sqrt((x^i)^2 + (y^i)^2)$;
  $pred_k = h_k(d^i)$;
  $w^i = \frac{1}{\sqrt{2\pi \cdot \sigma_k^2}} \cdot \exp(\frac{-(pred_k - z_k)^2}{2 \cdot \sigma_k^2})$;
  **for** $j \leftarrow 2$ ***to*** $M$ **do**
    $p_{temp} = f(p_{k-1}^i, {v_x}^j, {v_y}^j)$;
    $d_{temp} = sqrt((x^{temp})^2 + (y^{temp})^2)$;
    $pred_{temp} = h_k(d_{temp})$;
    $w_{temp} = \frac{1}{\sqrt{2\pi \cdot \sigma_k^2}} \cdot \exp(\frac{-(pred_{temp} - z_k)^2}{2 \cdot \sigma_k^2})$;
    $[p_k^i, w^i] = RL(w_{temp}, w^i, p_{temp}, p_k^i)$;
  **end**
**end**
$[\hat{w}, \hat{p}] = remuestreo(w, p_k)$;

Este algoritmo realiza para cada partícula $p_{k-1}^i$ la operación de muestreo y el cálculo del peso dando como resultado la partícula $p_k^i$ y $w^i$. Luego, se computa un muestreo adicional dando como resultado $p_{temp}$ donde $x^{temp}$ e $y^{temp}$ son las posiciones en cada eje. Se calcula su peso $w_{temp}$ y a continuación se realiza una selección o remuestreo local entre las partículas $p_k^i$ y $p_{temp}$. El remuestreo local sobre escribe la partícula $p_k^i$ y peso $w^i$ por los que hayan sido seleccionados. Este remuestreo local se repite $M$ veces. Una vez finalizado el procesamiento descripto para todas las partículas se procede al remuestreo del conjunto total $w = \{w^j : j = 1 \dots N\}$ y $p_k = \{p_k^j : j = 1 \dots N\}$.

El trabajo presentado en [15] sigue la primer estrategia descripta mientras que el trabajo presentado en [16] sigue la última. Finalmente el trabajo presentado en [23] introduce la combinación de ambas estrategias.

Las Figuras 3.1 y 3.2 presentan un modelo gráfico para las estrategias descriptas. La primera muestra el FP estándar y el FP utilizando remuestreo distribuido y el esquema de comunicación asociado. La segunda representa la técnica de remuestreo localizado y la combinación de ambas estrategias, remuestreo localizado y distribuido. Esta representación gráfica es útil cuando se consideran diferentes compromisos para la implementación del filtro.

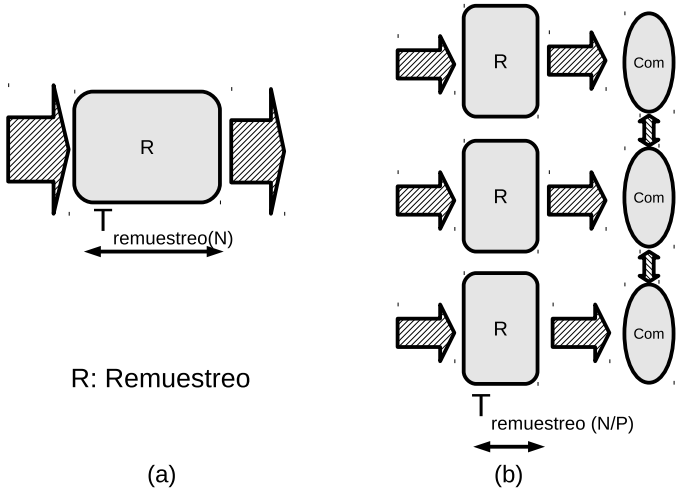


Figura 3.1: Modelo para el (a) FP estándar y (b) FP con remuestreo distruibuido.

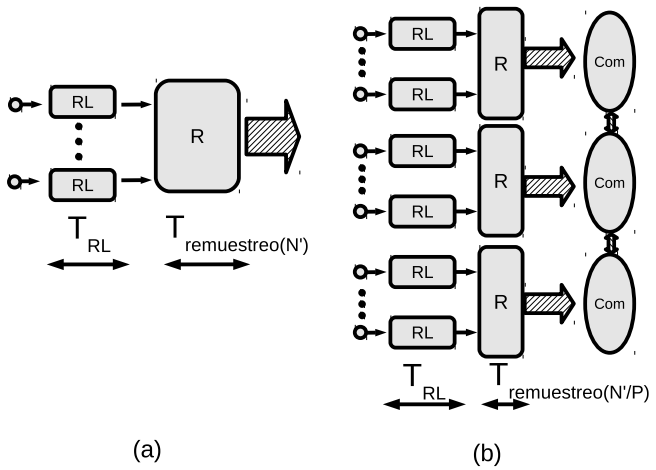


Figura 3.2: Modelo para el (a) FP con remuestreo local y (b) FP con remuestreo local y distribuido.

El ancho de cada caja representa el tiempo de ejecución para el cómputo y la altura el costo en recursos de hardware requeridos para lograr ese rendimiento. Se considera un esquema de comunicación determinístico con un número de partículas de intercambio constante como es el

caso de RNA con comunicación local. Por lo tanto, el tiempo de ejecución requerido para la comunicación es constante y se asume insignificante comparado con el tiempo total requerido por el filtro. Bajo esta suposición, se pueden analizar diferentes compromisos en el tiempo de ejecución sin considerar la comunicación.

Como los métodos de remuestreo son secuenciales, cada caja de remuestreo representa el mismo recurso de hardware y su altura es la misma en todos los casos. Por lo tanto, múltiples cajas de remuestreo indican múltiples recursos de hardware iguales. El ancho de la misma varía de acuerdo al valor de $P$ y $M$. Las cajas de remuestreo local también conservan la altura y su ancho varía de acuerdo al valor de $M$.

Un ejemplo es el filtro de la Fig. 3.1 .b) el cual es más costoso en términos de hardware pero su tiempo de ejecución es menor que en el FP secuencial de la Fig. 3.1.a). Otro caso es el filtro de la Fig. 3.2.a): como se aplica el remuestreo localizado el número de partículas $N'$ es menor que $N$ y el ancho de la caja de remuestreo es más angosto que el de la Fig. 3.1.a). Finalmente, el filtro de la Fig. 3.2.b) es el más costoso en hardware pero su tiempo de ejecución es el menor de los cuatro filtros considerados aquí.

## 3.3. Análisis de precisión para diferentes configuraciones *N-P-M*

Además del comportamiento cualitativo de los filtros paralelos que se describe utilizando los modelos descriptos en la sección anterior, se realizó un análisis cuantitativo. Se implementó el algoritmo de filtro de partículas con remuestreo distribuido y local en Matlab. Se introdujo el protocolo de comunicación RNA con comunicación local presentado en [21], con la diferencia que el intercambio se realiza luego del remuestreo y el número de partículas intercambiadas es 2. El método de remuestreo considerado es el «remuestreo sistemático residual» modificado para operar con pesos no normalizados [13]. Se realizaron simulaciones Monte Carlo para 100 corridas diferentes de 25 instantes de tiempo cada una. Como métrica de precisión se tomó el error absoluto relativo medio entre la posición real del móvil y la estimada.

El filtro de partículas implementado se aplicó a un escenario de seguimiento en 2 dimensiones. El objetivo es estimar la posición de un móvil moviéndose a una velocidad casi constante. Se consideró un sensor que provee mediciones de ángulo y rango, como puede ser un sensor laser. Los parámetros del filtro $P$ (números de EPs), $N$ (número total de partículas) y $M$ (número de partículas destinadas a la distribución local) tiene un impacto directo en la precisión del filtro. Debido a la relevancia de estos parámetros se nombró el conjunto $\{N\text{-}M\text{-}P\}$ como configuración y se realizaron varias simulaciones para distintas configuraciones.

El sistema dinámico está dado por las ecuaciones 2.8 y 2.9 cuyas componentes $v_x$ y $v_y$ son muestras de una distribución gaussiana con matriz de covarianza $Cov_{dynamic} = q \cdot I_2$, donde $I_2$ es la matriz identidad de dimensión $2x2$ y $q = 2$ $[m^2]$. El estado inicial del móvil es $\mathbf{x}_0 = [0.2$ m, 1 m/s, 60 m, $-3$ m/s] y $\Delta T$ es 1 $s$.

El sensor se localiza en el origen del sistema de coordenadas y el modelo se describe por la ecuación:

$$z_k = \begin{bmatrix} \arctan(y_k/x_k) \\ \sqrt[2]{x_k^2 + y_k^2} \end{bmatrix} + \begin{bmatrix} v_t^{angulo} \\ v_t^{rango} \end{bmatrix}, \tag{3.1}$$

donde $v_t^{angulo}$ and $v_t^{rango}$ son muestras de un ruido gaussiano con media cero y matriz de covarianza

$$Cov_{sensor} = \begin{bmatrix} r_b & 0 \\ 0 & r_r \end{bmatrix}. \tag{3.2}$$

Las varianzas del sensor fueron definidas con los valores: $r_b = 0{,}08726^2$ $[rad^2]$ y $r_r = 0{,}4^2$ $[m^2]$ y el rango de las mediciones puede ir hasta 80 $[m]$.

Las Figs. 3.3, 3.4 y 3.5 muestran resultados comparativos entre el filtro de partículas paralelo (FPP) con y sin remuestreo local para diferentes configuraciones y el filtro secuencial. La precisión del filtro estándar permanece constante dado que ninguno de los parámetros afectan su desempeño. El filtro paralelo con o sin remuestreo local presenta una degradación en su desempeño a medida que el valor de $P$ se incrementa. Además el filtro paralelo sin la técnica de remuestreo local tiene un mayor error que el filtro estándar lo cual es consistente con el análisis previo. Si se aplica la técnica de remuestreo local se obtienen estimaciones más precisas y el error se reduce a medida que el valor $M$ crece.

La Fig. 3.3 muestra que con 1024 partículas el filtro paralelo con remuestreo local es más preciso quel el filtro estándar para todos los valores de $P$. Por otro lado las Figs. 3.4 y 3.5 muestran que para $P > 16$, el filtro paralelo con remuestreo local y $M = 2, 4, 8$ es menos preciso que el filtro estándar. Este comportamiento es diferente que el observado en la Fig. 3.3 y se explica por el pobre desempeño del filtro estándar para 1024 partículas.

Comparando las figuras para un valor fijo de $P$ se observa un decremento en el error cuando el número de partículas total crece, lo cual es razonable dado que cada EP procesa más partículas. Sin embargo, para igual número de partículas por grupo, por ejemplo $P = 64$ en la Fig. 3.4 y $P = 128$ en la Fig. 3.5 existe una pequeña mejora lo cual demuestra que incrementar el número de EPs de características iguales no representa una ganancia desde el punto de vista de precisión.

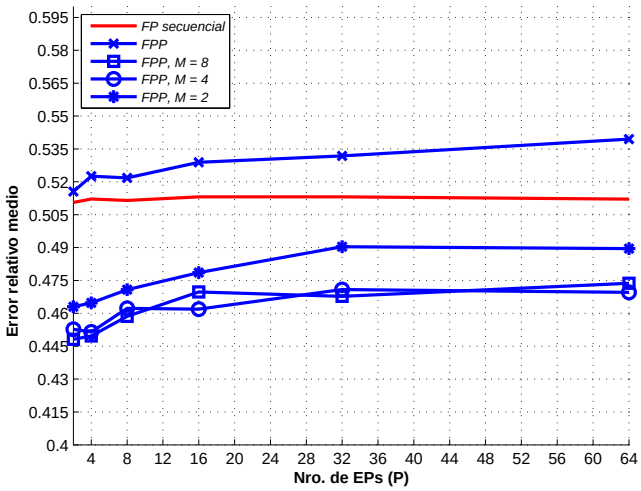


Figura 3.3: Error en la estimación para FP estándar y FP paralelo con y sin remuestreo localizado para $N = 1024$ en función del número de EPs (P).

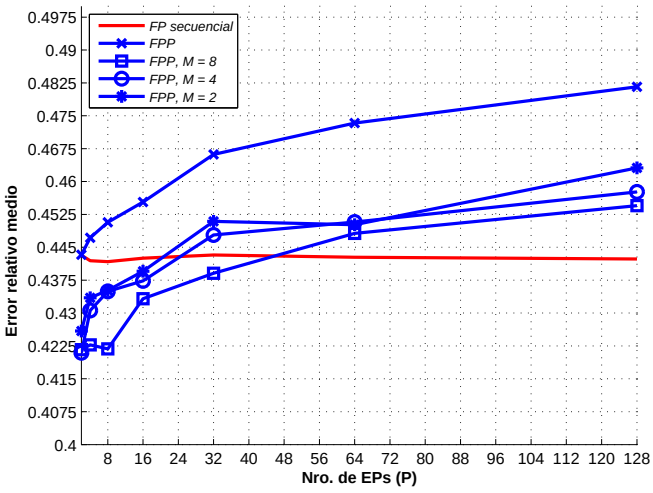


Figura 3.4: Error en la estimación para FP estándar y FP paralelo con y sin remuestreo localizado para $N = 2048$ en función del número de EPs (P).

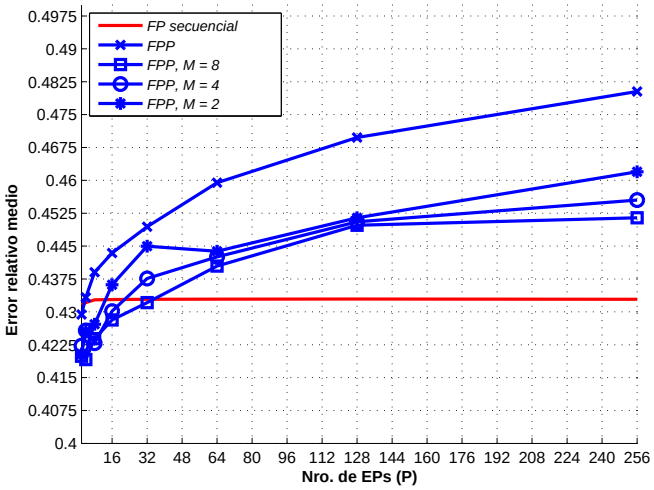


Figura 3.5: Error en la estimación para FP estándar y FP paralelo con y sin remuestreo localizado para $N = 4096$ en función del número de EPs (P).

La Tabla 3.1 muestra el tiempo de ejecución de la operación de muestreo junto con la técnica de remuestreo local y también del remuestreo. Esta informacón es útil para analizar el balance entre $M$ y $P$. La idea es equilibrar el tiempo de ejecución de las operaciones muestreo y remuestreo tomando en consideración la precisión. Basados en el valor de la tabla y el error máximo permitido, se puede obtener una configuración para el filtro paralelo que se aproxime a la óptima. Por ejemplo, asumiendo que el número de partículas es fijo e igual a 10240 y se toma como primera aproximación $M = 4$ y $P = 128$. Para esta configuración el tiempo de ejecución del muestreo es un 33 % mayor que el de remuestreo. Una opción mejor sería tomar $M = 4$ y $P = 256$, el error es similar pero el tiempo de ejecución es menor.

Tabla 3.1: Error y tiempo de ejecución para filtro paralelo con remuestreo local y $N = 10240$.

| *Filtro* | $M$ | $P$ | $T_{muestreo}$ | $T_{remuestreo}$ | $EMC$ |
|---|---|---|---|---|---|
| FP estándar | - | - | 0.024 ms | 20.55 ms | 0.415 |
| FP paralelo | 2 | 4 | 0.107 ms | 5.45 ms | 0.415 |
| | | 8 | | 2.80 ms | 0.417 |
| | | 16 | | 1.47 ms | 0.417 |
| | | 32 | | 0.81 ms | 0.424 |
| | 4 | 64 | 0.172 ms | 0.43 ms | 0.432 |
| | | 128 | | 0.23 ms | 0.442 |
| | | 256 | | 0.13 ms | 0.450 |
| | 8 | 512 | 0.302 ms | 0.08 ms | 0.457 |

## 3.4. Resumen

Se abordó un análisis de dos estrategias fundamentales para reducir el tiempo de ejecución de la operación de remuestreo. Ambas estrategias se basan en el mismo concepto: si se reduce el número de partículas, se logra también una reducción en el tiempo de ejecución de los algoritmos de remuestreo. La primera estrategia consiste en distribuir la operación de remuestreo en grupos de partículas. La segunda estrategia consiste en una mejora de la operación de muestreo permitiendo el remuestreo de un número reducido de partículas pero a la vez más representativas de la función de densidad de probabilidad sin afectar la precisión del filtro. Como desventaja presenta una complejidad computacional mayor en la operación de muestreo. Es también posible una combinación de las estrategias de manera de obtener un filtro de partículas con menor degradación.

De modo de poder conocer el alcance de ambas estrategias y su combinación se llevó a cabo una evaluación cualitativa y cuantitativa del comportamiento de las mismas. De la evaluación se concluye que efectivamente existe una degradación en el filtro al aplicar el remuestreo distribuido. A medida que se aumenta la cantidad de EPs aumenta la degradación. Por otro lado al aplicar el remuestreo local se reduce el error en la estimación por lo que se podría reducir el número total de partículas. Se observa también que si se aumenta la cantidad de partículas por grupo se logra reducir el error pero no sucede lo mismo si se aumenta la cantidad de grupos de igual cantidad de partículas.

# Capítulo 4

# Arquitecturas VLSI para el filtrado de partículas en tiempo real

En este capítulo se describen tres arquitecturas para el filtrado de partículas en tiempo real. Estas arquitecturas utilizan la estrategia de remuestreo distribuido y buscan explotar el paralelismo intrínseco del algoritmo del filtro de partículas.

En lo siguiente, primero se presenta una revisión de los trabajos previos que más han contribuido en la implentación del algoritmo. A continuación se describe el escenario de prueba que se desarrolló para verificar los tres diseños propuestos siguiendo la metodología descripta en la introducción. Se presentan por separado cada uno de los diseños, su arquitectura y los resultados de simulación y verificación. Finalmente se realiza una comparación entre las arquitecturas en términos de su tiempo de ejecución y el área de silicio requerida.

## 4.1. Revisión de trabajos previos

En la bibliografía existen trabajos previos que se enfocan en la implementación del filtro de partículas para aplicaciones en tiempo real. En [24] se presenta una arquitectura dedicada al FP para el seguimiento de un móvil compuesta de múltiples elementos de procesamiento y una unidad central. Las operaciones del algoritmo son realizadas en forma local en cada elemento de procesamiento (EP). Luego del remuestreo, la unidad central se encargar del intercambio de partículas entre los EPs de forma tal de reducir la degradación del desempeño. Se introducen varios esquemas de comunicación, incluyendo un intercambio fijo.

En [25] se presenta un diseño VLSI del elemento de procesamiento, el cual incluye un *pipeline* que permite la utilización de bloques de procesamiento con diferentes latencias. En [26] se presenta el diseño y la implementación VLSI de una unidad central, introducida en [24],

que realiza varios esquemas de comunicación para una arquitectura de múltiples elementos de procesamiento.

En [27] se introduce el diseño de una arquitectura con *pipeline* que permite el procesamiento paralelo de partículas. El número de etapas del *pipeline* replicadas es variable. Dado que la tasa de procesamiento de cada etapa depende de la cantidad de veces que está replicada y su latencia, se introduce un análisis para determina la cantidad óptima de veces que cada etapa debe replicarse de forma tal de obtener el tiempo de procesamiento total mínimo.

## 4.2. Descripción del escenario para prueba de los diseños propuestos

Se tomó como caso ilustrativo la localización de un móvil utilizando la tecnología de Indentificación por Radio Frecuencia (RFID por las siglas del inglés *Radio-Frequency identification*). En redes de sensores, la localización basada en radio frecuencia ha ganado importancia en ambientes donde el Sistema Global de Posicionamiento (GPS por las siglas del inglés *Global Positioning based system*) no tiene un buen funcionamiento debido a disponibilidad de satélites reducida o problemas relacionados con múltiples caminos de la señal de recepción [28] [29]. La aplicación elegida es la localización basada en el indicador de potencia de la señal recibida (RSSI por las siglas del inglés *Received Signal Strength Indicator*) de camiones en minas a cielo abierto.

La tecnología RFID se compone de receptores, antenas y etiquetas. Las etiquetas envian su número de identificación a los receptores. Haciendo uso de RSSI es posible estimar la distancia entre una etiqueta y un receptor dado que su valor decrece con la distancia. Debido a varios factores que afectan la propagación de las ondas electromagnéticas en un medio (refracciones, reflexiones, dispersión) la potencia de la señal recibida varía con los obstáculos en el ambiente, la altura y dirección de la antena y también la potencia de la señal transmitida. Como resultado la función del sensor, *potencia recibida* vs *distancia*, resulta en una función suryectiva y por lo tanto en una distribución de probabilidad multi-modal.

La Fig. 4.1 muestra el modelo de propagación de señales de radio frecuencia (RF) que incluye dos rayos [30] para un ambiente rural y una frecuencia de comunicación de 433 Mhz y una altura para la antena de recepción y de la etiqueta de 2.5 m. La figura muestra la potencia media de la señal recibida versus distancia. Para un valor de distancia dado, la distribución de la señal de RF se considera gaussiana y su varianza varía con la potencia de la señal [28]. Es posible observar que para una potencia recibida de $-70$ dBm se corresponden múltiples valores de distancia: 8 m, 8.6 m, 15.8 m, 21 m and 43.1 m, siendo uno de ellos el verdadero valor de la distancia de la etiqueta a la antena. Cada uno de estos valores es un modo de la función de densidad de

probabilidad, lo que resulta en una función multi-modal.

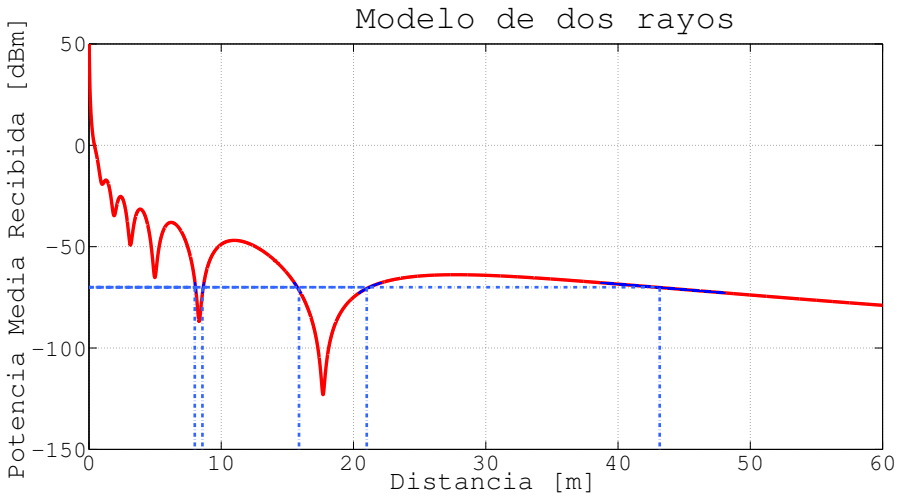


Figura 4.1: Modelo de dos rayos para un enlace de comunicación de 433 Mhz en un ambiente rural.

La tarea de localización basada en RSSI puede ser realizada utilizando el algoritmo de filtro de partículas. El escenario hipótetico es un móvil con una etiqueta RFID moviendose en el plano y una antena localizada en el origen de coordenadas. Sea $p_k^i$ la $i$-ésima partícula, $p_k^i = \begin{bmatrix} x^i & \dot{x}^i & y^i & \dot{y}^i \end{bmatrix}'$, donde $x^i$ e $y^i$ es la posición en cada eje y $\dot{x}^i$ e $\dot{y}^i$ las velocidades. La evolución en el tiempo de la posición y velocidad del móvil viene dada por:

$$f(p_{k-1}^i, v_x, v_y) = \begin{bmatrix} 1 & \Delta T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta T \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot p_{k-1}^i + \begin{bmatrix} 0{,}5 \cdot \Delta T^2 & 0 \\ \Delta T & 0 \\ 0 & 0{,}5 \cdot \Delta T^2 \\ 0 & \Delta T \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \end{bmatrix}, \qquad (4.1)$$

donde $v_x$ y $v_y$ son muestras de un ruido aleatorio con distribución uniforme $U[0, Q]$.

El pseudo código del algoritmo de Filtro de Partículas adaptado a la aplicación elegida y para un conjunto de $N$ partículas se describe a continuación.

Este algoritmo realiza para cada partícula $p_{k-1}^i$ la operación de muestreo utilizando el modelo dinámico descripto en la ecuación 4.1. Luego, calcula la distancia euclidiana de la partícula $p_k^i$ al origen. A partir de la distancia $d^i$ se computa la potencia $Pot^i$ utilizando el modelo de propagación de dos rayos cuya característica se muestra en la Fig. 4.1. Esta indicación de potencia modela la señal que arrojaría el sensor RF de encontrarse el móvil en la ubicación senãlada por la partícula $p_k^i$. Dependiendo de los obstáculos presentes en el ambiente puede que se requiera de un modelo de propagación más complejo y detallado que el aquí utilizado. Por último, se

*inicialización aleatoria de las partículas;*
**for** $i \leftarrow 1$ ***to*** $N$ **do**
  $p_k^i = f(p_{k-1}^i, v_x, v_y)$;
  $d^i = sqrt({x^i}^2 + {y^i}^2)$;
  $Pot^i = F_{sensor}(d^i)$;
  $w^i = \frac{1}{\sqrt{2\pi \cdot \sigma^2}} \cdot \exp(\frac{-(Pot^i - Pot_{medida})^2}{2 \cdot \sigma^2})$
**end**
$[\tilde{w}, \tilde{p}] = remuestreo(w, p_k)$;

calcula el peso de la partícula $w^i$ asumiendo que el modelo probabilístico del sensor $F_{sensor}(d)$ es gaussiano con varianza $\sigma^2$. La variable $Pot_{medida}$ es la medición de potencia de la señal recibida.

El remuestreo se computa sobre el conjunto total de partículas $p_k = \{p_k^i : i =, \ldots, N\}$ y pesos $w = \{w^i : i = 1, \ldots, N\}$. Se pueden utilizar algunos de los algoritmos de remuestreo descriptos en el Capítulo 2. Como resultado se obtiene un nuevo conjunto de partículas remuestreadas $\tilde{p}$ junto con sus pesos $\tilde{w}$.

La posición estimada se computa utilizando la ecuación 4.2.

$$\tilde{x} = \sum_{i=1}^{N} \tilde{p}^i \cdot \tilde{w}^i. \tag{4.2}$$

El algoritmo del filtro de partículas aquí descripto es similar al de la Sección 2.3 excepto que se define $h_k$ como el modelo de propagación de dos rayos y $z_k$ es una medición de potencia.

El escenario de simulación consiste en una antena ubicada en el origen del sistema de coordenadas y cuyo modelo de propagación es el descripto anteriormente y un móvil moviendose a velocidad casi constante.

El estado inicial del móvil es $\mathbf{x}_0 = [0.2 \text{ m}, 1 \text{ m/s}, 60 \text{ m}, -3 \text{ m/s}]$ y los parámetros $\Delta T$ y $Q$ es 0,1 s y 10 m/s$^2$.

## 4.3. Diseños basado en arquitecturas *Dataflow*

En esta sección se describen dos diseños basados en el modelo computacional *Dataflow* [31]. Para ambos diseños se presenta la arquitectura y también la micro-arquitectura de los módulos de cómputo. Se implementó el modelo RTL y en Matlab, en este caso en punto flotante y punto fijo. Luego, se presentan los resultados de simulación del modelo RTL, los cuales son verificados con los obtenidos a partir del modelo Matlab.

### 4.3.1. Diseño 1

El diseño propuesto es un elemento de procesamiento cuya arquitectura está basada en el modelo de computación *Dataflow* y cuyo propósito específico es el procesamiento de datos acorde al algoritmo del Filtro de Partículas. Luego, el elemento de procesamiento se integra a una arquitectura con múltiples procesadores idénticos para asi explotar la estrategia de remuestreo distribuido.

La Fig. 4.2 muestra el gráfico de flujo de datos para el algoritmo del filtro de partículas. Las operaciones del algoritmo son mapeadas en los siguientes nodos: *muestreo*, *unidad cuadrática*, *unidad raíz cuadrada*, *predicción potencia*, *gaussiana*, *remuestreo*. El modo de ejecución seleccionado es data-driven por lo que la operación de cada nodo estará sujeta a la disponibilidad de sus operandos. Claro ejemplo de este control del flujo de datos es el nodo *gaussiana* el cual comenzará su procesamiento una vez que los operandos: *observación*, *varianza* y la salida del nodo *predicción potencia* son actualizados.

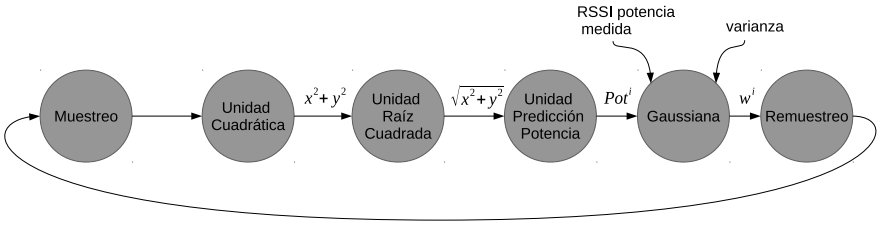


Figura 4.2: Diagrama de flujo de datos para el filtro de partículas.

La Fig. 4.3 muestra la arquitectura del elemento de procesamiento. Se mapean los nodos del gráfico de flujo de datos de la Fig. 4.2 en modulos de procesamiento y además se adiciona el módulo $palabra - a - memoria$. Este último registra la partícula remuestreada y la partícula trasladada. La estructura propuesta cuenta con un pipeline para lograr un diseño de alto rendimiento. El *pipeline* está compuesto por los modulos de procesamiento donde cada uno de ellos es una etapa del *pipeline*.

Por otro lado el control del flujo de datos se realiza localmente en cada módulo. En otras palabras, los módulos adyacentes realizan un intercambio de señales de control que permite la ejecución *data driven*. El hecho de contar con un control distribuido permite una flexibilidad mayor en la división del procesamiento en modulos. En lo que sigue los terminos «etapa» y «módulo de procesamiento» se utilizan de manera indistinta.

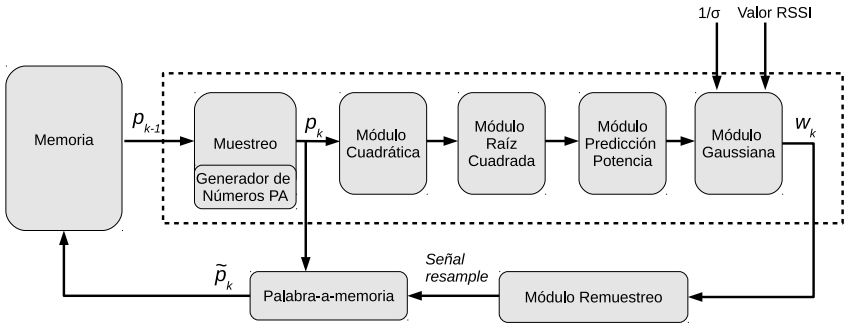


Figura 4.3: Arquitectura del elemento de procesamiento.

#### 4.3.1.1. Operación

El módulo *muestreo* recibe el dato partícula $p_{k-1}$ de la memoria principal. Una vez que el valor $p_k$ se actualiza, es registrado por el módulo $palabra - a - memoria$ que a su vez guarda también el dato previo escrito a memoria $\tilde{p}_{k-1}$. El dato que se escribe a memoria será uno de los dos: $p_k$ ó $\tilde{p}_{k-1}$ dependiendo del estado de la señal *resample* cuyo valor lo establece el módulo *remuestreo* una vez que recibe el valor actualizado de $w_k$.

La memoria principal es de doble puerto, uno de lectura y otro de escritura. Por lo tanto una vez que el *pipeline* está lleno, ambas operaciones se realizan en simultáneo.

#### 4.3.1.2. Módulos Muestreo y Cuadrática

El módulo *muestreo* procesa los datos de memoria principal. El ancho de palabra de la memoria principal es de 48 bits donde cada componente del vector partícula posee 12 bits. El rango de las variables posición y velocidad es de $[-40, 40]$ m y $[-25, 25]$ m/s. Este modelo realiza una traslación en el plano de la partícula utilizando una versión simplificada del modelo dinámico detallado en la ecuación 4.1. La posición y velocidad son computadas de la siguiente manera:

$$p_k^i(1) = p_{k-1}^i(1) + p_{k-1}^i(2) \cdot \triangle T + \frac{1}{2} \cdot n_x, \tag{4.3}$$

$$p_k^i(3) = p_{k-1}^i(3) + p_{k-1}^i(4) \cdot \triangle T + \frac{1}{2} \cdot n_y, \tag{4.4}$$

$$p_k^i(2) = p_{k-1}^i(2) + n_x, \tag{4.5}$$

$$p_k^i(4) = p_{k-1}^i(4) + n_y, \tag{4.6}$$

donde $n_x$ y $n_y$ son muestras de ruido cuya distribución es uniforme $U[0, W]$, $p^i_{k-1}(1) = x$, $p^i_{k-1}(3) = y$, $p^i_{k-1}(2) = \dot{x}$ y $p^i_{k-1}(4) = \dot{y}$. Dependiendo del valor asignado al parámetro $\triangle T$, el valor de $W$ debe ser ajustado de manera tal de asegurar un rango dinámico de aceleración similar al del modelo original. Por otra parte la muestra de ruido se implementa mediante un registro de desplazamientos con retroalimentación lineal [32] cuya semilla es reconfigurable. La Fig. 4.4 muestra el gráfico de flujo de datos que se implementó para ambos modulos.

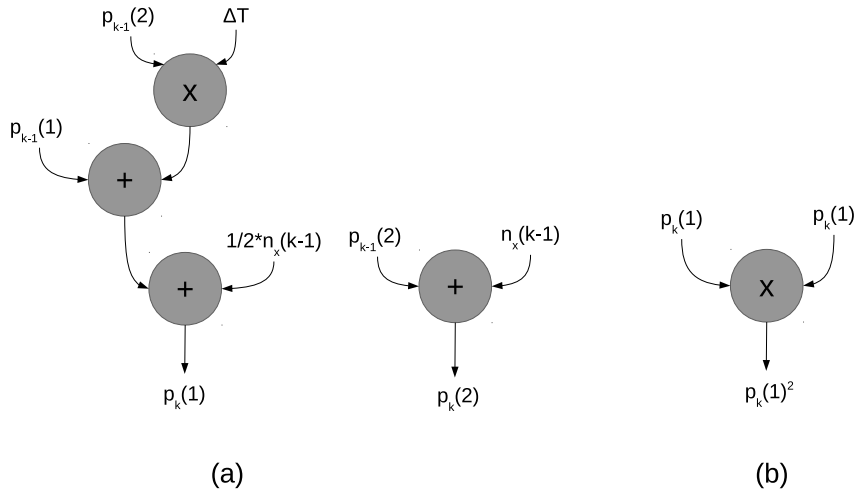


Figura 4.4: Diagrama de flujo de datos: (a) módulo muestreo; (b) módulo cuadrática sólo para componente $x$.

En cuanto al módulo *Unidad Cuadrática* se implementaron dos multiplicadores de 12 bits de forma tal de realizar las multiplicaciones: $p^i_k(1)^2$ y $p^i_k(3)^2$ en forma concurrente.

#### 4.3.1.3. Módulo Raíz Cuadrada, Predicción Potencia y Gaussiana

Las funciones no lineales: raíz cuadrada, el modelo de propagación de señales de RF descripto en la Sección 4.2 y la distribución normal son implementadas mediante tablas. Todas se evaluan mediante una aproximación lineal a tramos con segmentación uniforme. Además se implementó una interpolación lineal para reducir su tamaño.

Para cualquier punto $x \in [a\ , b]$, la interpolación lineal se calcula de la siguiente manera:

$$\tilde{f}(x) = \frac{f(b) - f(a)}{b - a} \cdot (x - a) + f(a). \tag{4.7}$$

Supongamos que la tabla posee un tamaño de $N \cdot Q$ bits, donde $N$ es la cantidad de lugares de memoria y $Q$ es el ancho de palabra y $M$ los bits dedicados a la interpolación. Esto implica que entre dos lugares de memoria consecutivos habrá $2^M$ divisiones. La Fig. 4.5 muestra la

segmentación uniforme y las subdivisiones dentro de cada intervalo. Desde el punto de vista de implementación, los $N$ bits más significativos de la palabra $x$ son utilizados para direccionar la tabla y los siguientes M bits proveen la posición dentro del intervalo.

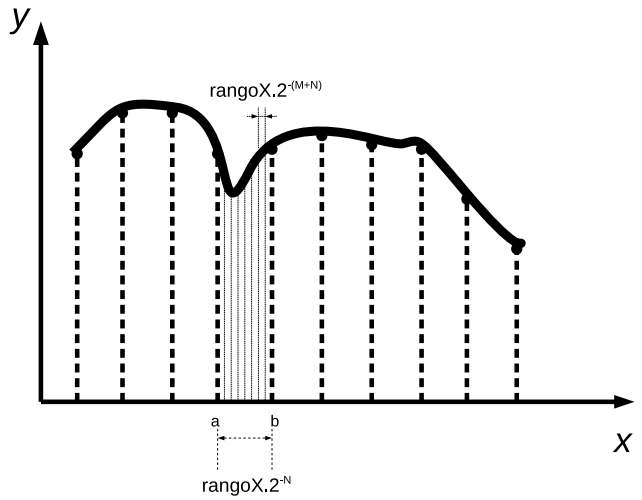


Figura 4.5: Segmentación uniforme y subdivisiones para la interpolación.

Si la variable $x$ tiene una resolución de $(N+M)$ bits y rango $rangoX$, su valor en punto fijo viene dado por:

$$x_{PF} = \lceil \frac{x \cdot 2^{(N+M)}}{rangoX} \rceil, \tag{4.8}$$

donde tal como se dijo en la Sección 2.2 la función $\lceil x \rceil$ se define como $min\{k \in \mathbb{Z} | x \leq k\}$.

Entonces la aproximación en punto fijo de $\tilde{f}(x)$ se expresa de la siguiente manera:

$$\tilde{f}(x) \approx \frac{k_3}{k_2 \cdot k_1} \cdot ((f(b)_{PF} - f(a)_{PF}) \cdot (x_{PF} - a_{PF}) + f(a)_{PF} \cdot \frac{k_1}{k_3}), \tag{4.9}$$

donde

$$k_1 = \frac{rangoX}{2^{(N+M)}}, \tag{4.10}$$

$$k_2 = \frac{rangoY}{2^Q}, \tag{4.11}$$

$$k_3 = \frac{rangoX}{2^N}. \tag{4.12}$$

El factor $\frac{k_1}{k_3}$ es en realidad igual $2^{-M}$, por lo que no es necesario computar la operación división.

La Fig. 4.6 muestra un esquema del hardware implementado. Al considerar memorias de doble puerto es posible leer los dos valores $f(b)$ y $f(a)$ en un solo ciclo de reloj. La resolución $R$ del dato interpolado es igual a $Q + M$.

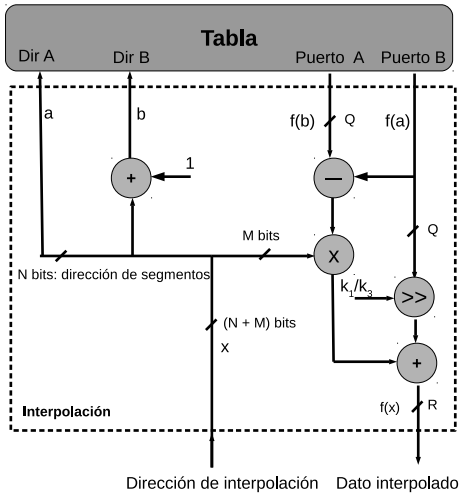


Figura 4.6: Arquitectura para los módulos Raíz Cuadrada, Predicción Potencia y Gaussiana.

La ecuación 4.9 y el hardware de la Fig. 4.6 es genérico para la implementación de cualquier función no lineal.

La implementación de la distribución normal requiere evaluar distribuciones normales con diferentes valores de varianza y media. Es posible obtener cualquier distribución normal a partir de la distribución normal estándar $N(0, 1)$ a partir de las siguientes ecuaciones. Si una distribución con media $\mu$ y varianza $\sigma^2$ debe ser evaluada para un valor particular $t$:

$$z = \frac{t - \mu}{\sigma}, \tag{4.13}$$

$$p_{Normal} = \frac{1}{\sigma} \cdot p_{StandardNormal}(z), \tag{4.14}$$

donde

$$P_{StandardNormal}(z) = \frac{1}{\sqrt{2 \cdot \pi}} \cdot \exp(\frac{-z^2}{2}). \tag{4.15}$$

Además, como la función es simétrica alrededor de la media, sólo es necesario almacenar los valores correspondientes a una mitad del intervalo de evaluación reduciendo aún más el tamaño

de la tabla.

El flujo de operaciones presentado posee tres funciones tabuladas e interpolaciones en cascada. Por ejemplo el valor interpolado correspondiente a la función raíz cuadrada se convierte en la dirección de interpolación para la función correspondiente al modelo de dos rayos. Dada esta situación es deseable encontrar un ancho de palabra apropiado para el direccionamiento de la tabla, la interpolación y cuantización de los valores tabulados. Estos anchos de palabra deberían igualar a uno la razón entre el ancho de palabra de la dirección de interpolación y del valor interpolado de la función anterior. Al mismo tiempo los errores de aproximación deberían ser minimizados dado que se propagan a través el flujo de datos. En este sentido, se adoptó el análisis de precisión para implementaciones prácticas de funciones lineales a tramos presentado en [33]. Dicho análisis establece la relación de compromiso entre $M$ y $Q$ en relación al error de aproximación para un valor dado de $N$.

El error de aproximación o de interpolación se calcula como el promedio del error relativo sobre un conjunto de mil muestras del intervalo de evaluación y se exceptuan los valores donde $f(x) \approx 0$:

$$error(x) = mean(|\ \frac{f(x) - f_{interp}(x))}{f(x)}\ |). \tag{4.16}$$

Para el caso aquí tratado se definió el valor de $N$ para cada tabla en base a simulaciones y luego se realizaron las gráficas de error relativo tal como se muestra en la Fig. 4.7. En dicha figura se observan curvas del error en función de $Q$ y parametrizadas en $M$. Se puede observar que para $Q$ igual 8 el error es prácticamente independiente del valor que tome $M$. En el otro extremo donde $Q$ es igual a 15, se observa que las curvas de error se vuelven asintóticas para ciertos valores de $M$ y que a medida que se incrementa el valor de este parámetro se observa una reducción en el error. Sin embargo, para valores de $M$ superiores a 4 no se evidencia una reducción significativa. En el caso de la función raíz cuadrada se estableció $N$ igual a 10 y un error de interpolación menor a $3{,}5 \cdot 10^{-4}$. De la figura se observa que para $Q$ igual a 11 y $M$ igual 2 se obtiene un error de $3{,}3 \cdot 10^{-4}$. Además para $M$ mayor a 2 no se obtiene un reducción en el error. Por lo tanto, la implementación seleccionada es $N = 10$, $M = 2$ y $Q = 11$. Este mismo método se realizó para las tablas restantes. Es importante notar que la cota de error se estableció en forma aproximada para cada tabla en base a simulaciones, siendo $4 \cdot 10^{-4}$ para la función del sensor y $5 \cdot 10^{-3}$ para la función normal.

La Tabla 4.1 muestra la configuración elegida para cada implementación de las funciones lineales a tramos donde $N$,$M$,$Q$ son el número de bits asignados para la segmentación, interpolación y cuantización de los valores tabulados. Además $R$ es la cantidad de bits asignados para el valor interpolado y $S$ la cantidad de bits que son descartados para el direccionamiento de la

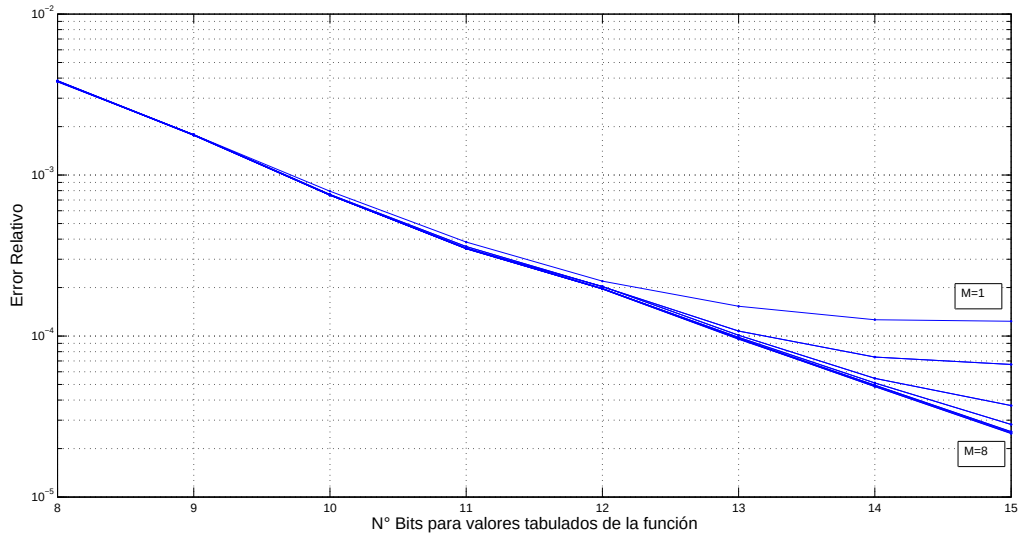


Figura 4.7: Gráficas de error en función de $Q$ y parametrizadas en $M$.

Tabla 4.1: Configuración de las funciones lineales a tramos.

| *Función* | $N$ | $M$ | $Q$ | $R$ | $S$ | *Tamaño Kbits* | *Rango de evaluación* | *Error de Interpolación* |
|---|---|---|---|---|---|---|---|---|
| Raíz Cuadrada | 10 | 2 | 11 | 13 | 11 | 11 | [0,3200] | $3{,}3 \cdot 10^{-4}$ |
| Sensor | 10 | 2 | 11 | 13 | 1 | 10 | [0,113] | $2 \cdot 10^{-4}$ |
| Normal | 9 | 1 | 10 | 11 | 3 | 5 | [0,5] | $4{,}1 \cdot 10^{-3}$ |

próxima tabla.

#### 4.3.1.4. Módulo Remuestreo y palabra-a-memoria

El algoritmo de remuestreo implementado es el Independent Metropolis Hasting (IMH) modificado [12] en el que se elimina la operación mínimo y división por una multiplicación lo cual resulta muy beneficioso desde el punto de vista de la implementación. Su pseudo código se presenta a continuación.

$\tilde{p_k}^1 = p_k^1$;
$w_{prev} = {w_k}^1$;
**for** $i \leftarrow 2$ ***to*** $NUMPARTICLES$ **do**
  $u \sim U[0,1]$;
  **if** $(\ u \cdot w_{prev} > {w_k}^i\ )$ **then**
    $w_{prev} = w_{prev}$;
    $\tilde{p_k}^i = \tilde{p_k}^{i-1}$; $resample = 1$;
  **else**
    $w_{prev} = {w_k}^i$;
    $\tilde{p_k}^i = {p_k}^i$; $resample = 0$;
  **end**
**end**

Al inicio del algoritmo, la primera partícula del conjunto es remuestreada, por lo que $\tilde{p_k}^1 = p_k^1$ y el valor inicial que toma la variable $w_{prev}$ es el peso $w_k^1$. Luego, por cada iteración del ciclo $for$ se computa $u$ que es una muestra aleatoria de una distribución uniforme $U[0,1]$. La partícula $p_k^i$ será remuestreada si ${w_k}^i$ es mayor a $u \cdot w_{prev}$. En tal caso, $w_{prev}$ se iguala al peso $w_k^i$, $\tilde{p_k}^i = {p_k}^i$ y la variable $resample$ toma el valor uno. Caso contrario, $w_{prev}$ no cambia su valor, $resample$ se iguala a cero y se remuestrea nuevamente la partícula anterior por lo que $\tilde{p_k}^i = {p_k}^{i-1}$.

Al igual que los algoritmos descriptos en el Capítulo 2, el propósito de agregar la muestra de un ruido con distribución uniforme es introducir cierta aleatoriedad en la selección de la partícula a remuestrear.

El hardware asociado al algoritmo descripto y la arquitectura para el módulo $palabra - a - memoria$ se muestra en la Fig. 4.7.

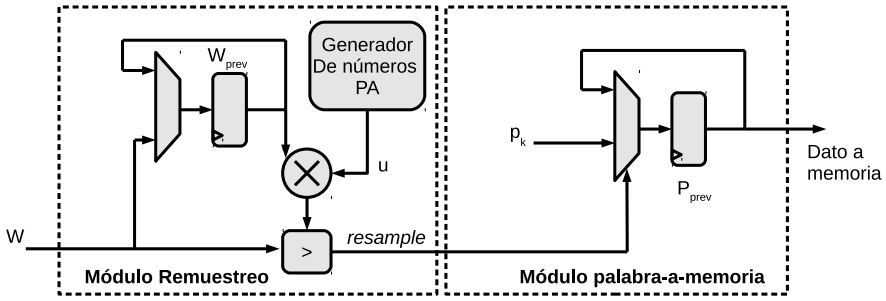


Figura 4.8: Arquitectura para módulo remuestreo y palabra-a-memoria.

La señal $resample$ controla el dato que se almacena en memoria. Si el valor de $resample$ es 1, el dato presente en el registro $P_{prev}$ se escribe a memoria, caso contrario, la partícula de entrada al módulo y el registro $w_{prev}$ son actualizados.

El generador de números pseudo-aleatorios se implementa con un registro de desplazamiento con retroalimentación lineal de 16 bits con semilla configurable.

Para sincronizar el dato $w_k^i$ en el módulo *remuestreo* y ${p_k}^i$ en la unidad *palabra-a-memoria*, este último debe ser introducido en un *pipeline* de datos cuyo número de etapas iguala al *pipeline* principal. De no tenerse en cuenta en el diseño este *pipeline* paralelo, el dato que se registre en la unidad *palabra-a-memoria* no corresponderá al peso $w_k^i$. En otras palabras, el retraso que introduce el *pipeline* de procesamiento debe tenerse en cuenta para la unidad *palabra-a-memoria* dado que esta unidad posee como entrada a ${p_k}^i$ que cronológicamente se encuentra al comienzo de dicho *pipeline*.

#### 4.3.1.5. Tiempo de ejecución del filtro

El rendimiento del diseño es un dato cada dos ciclos. La Fig. 4.9 muestra el diagrama de tiempos del *pipeline* implementado.

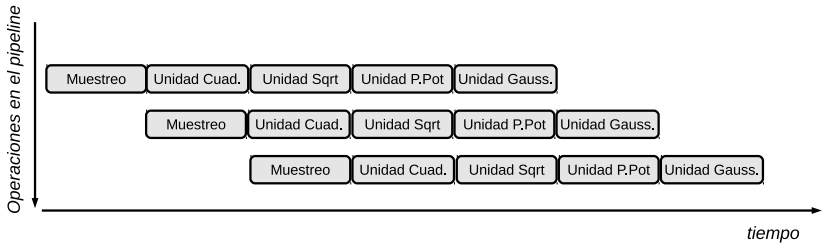


Figura 4.9: Tiempo de ejecución del filtro.

#### 4.3.1.6. Resultados de simulación

Se desarolló el modelo RTL del elemento de procesamiento y de los módulos de *remuestreo*, *palabra-a-memoria*, las tablas y los generadores de números pseudo-aleatorios. Se simuló el modelo Matlab en punto flotante y el modelo RTL para una potencia de medición de $-82{,}5$ dBm, $\sigma = 2{,}5$ dBm y un total de 2048 partículas. Para este nivel de potencia se corresponden tres valores de distancia: 8.3 m, 16.8 m, 19 m de acuerdo al modelo de propagación de dos rayos de la Sección 4.2. La función de distribución de probabilidad $p(x_k|z_{1:k})$ equivale a la suma de tres gaussianas cuyas medias son los valores mencionados. La Fig. 4.10 muestra la distribución de *pesos* vs *distancia* de ambos ambos modelos. Se observa que el peso de las partículas aproxima en forma correcta a la función de distribución de probabilidad. Puede notarse que el modelo RTL provee resultados similares al modelo en punto flotante. Notar además que dado que la medición está cuantizada en 8 bits, la distribución normal está también cuantizada.

Se simularon además los modelos Matlab en punto flotante y en punto fijo, este último es equivalente al modelo RTL. Se utilizó el escenario presentado al principio de este capítulo para un total de 16384 partículas. En la Fig. 4.11 se muestra la aproximación de la función de

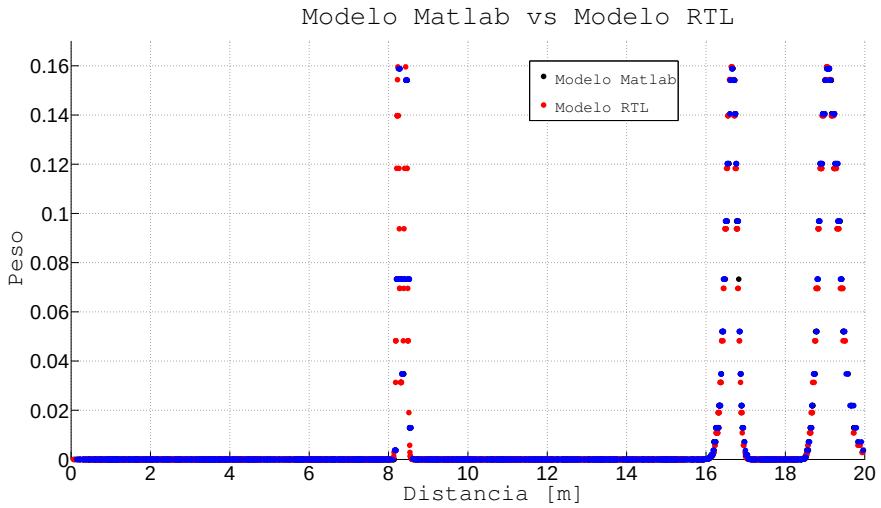


Figura 4.10: Pesos vs. distancia.

densidad de probabilidad que ambos modelos proveen. El instante de simulación corresponde a 0.4 $s$. Como se puede observar, la función posee tres modos que corresponden a anillos radiales concéntricos a la posición de la antena y cuyos radios son aproximadamente 19 $m$, 16.5 $m$ y 8.3 $m$. La posición del móvil [-14.4 m, 7.6 m] se indica con un círculo verde cuya distancia a la antena es de 16.3 $m$.

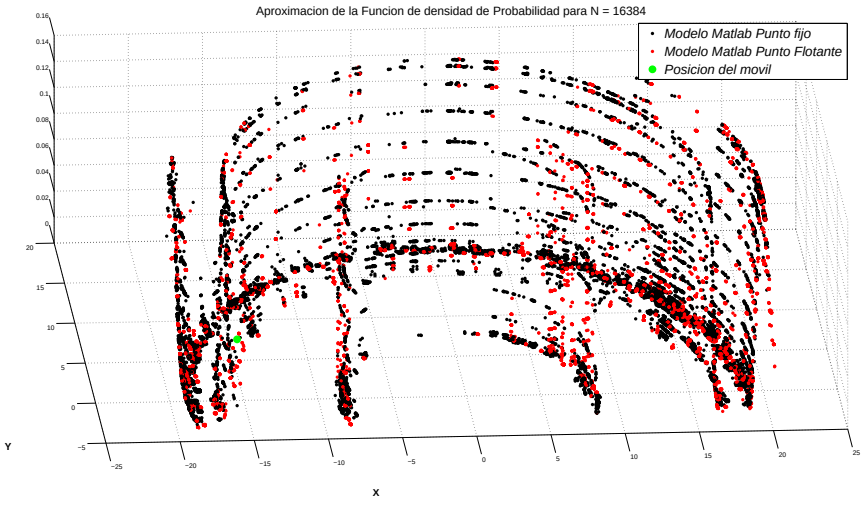


Figura 4.11: Simulación de Matlab del modelo en punto fijo y flotante.

En el instante de tiempo indicado la potencia de la señal recibida es de -76.7 $dBm$. Para esta potencia existen 4 hipótesis cuyas distancias radiales son: 8.15 $m$, 16 $m$ ,19.8 $m$, 55 $m$ de

acuerdo al modelo de propagación de dos rayos. Sin embargo, debido al filtrado iterativo de las hipótesis se conservan solo algunas.

La Fig. 4.12 muestra una ventana de tiempo de simulación del modelo RTL utilizando la herramienta Modelsim. Es posible observar que en el extremo superior se encuentra la señal de reloj. Inmediatamente debajo de ésta se encuentra la dirección de memoria de las partículas leídas. Luego, el resto de las señales continuan hacia abajo y se encuentran agrupadas según la unidad a la que pertenecen. Se puede observar la operación del *pipeline* tomando como referencia el cursor ubicado en el instante de tiempo en que la direccón de memoria de la partícula es 2. El módulo *muestreo* toma dos ciclos para su procesamiento. Luego comienza el procesamiento de la tercer partícula y así sucesivamente. Por otra parte, una vez que se registra el dato de salida del módulo *muestreo*, un ciclo después del cursor, comienza el procesamiento de la módulo *cuadrática* que también le toma dos ciclos completarlo. Lo mismo sucede con el resto de las etapas del *pipeline*.

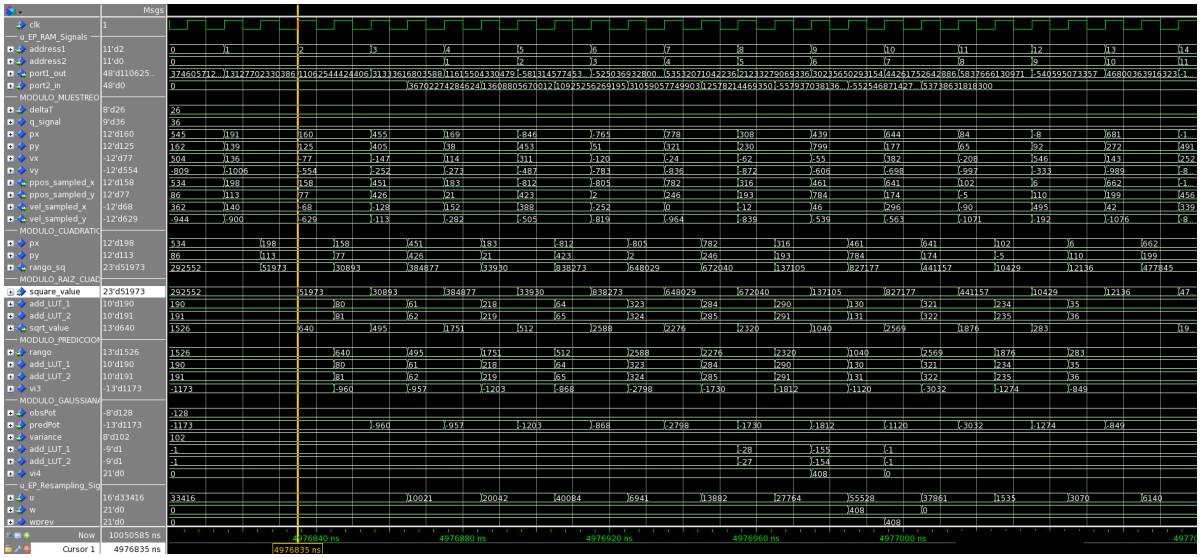

Figura 4.12: Simulación de señales del *pipeline* con la herramienta Modelsim.

#### 4.3.1.7. Resultados de síntesis lógica y estimación de área

Se realizó una síntesis lógica del modelo RTL del elemento de procesamiento utilizando Synopsis DC Compiler y la tecnología CMOS 0.13 $\mu$m. La Tabla 4.2 muestra el área requerida por el elemento de procesamiento y sus módulos. El conjunto de celdas básicas no incluye las celdas de memoria SRAM, por lo que se procede a estimar el área de memoria requerida utilizando el modelo de una celda de memoria establecida en [34] con una eficiencia de 0.7. Se

considera que la memoria principal tiene una capacidad de almacenamiento de 1024 partículas de 48 bits de ancho de palabra.

Tabla 4.2: Resultado de síntesis lógica y estimaciones del elemento de procesamiento.

| **Módulo** | **Área** [$\mu$**m**$^2$] |
|---|---|
| Muestreo | 14331 |
| Módulo Cuadrática | 4654 |
| Módulo Raíz Cuadrada | 2787 |
| Tabla Raíz Cuadrada | 67986 |
| Módulo Predicción Potencia | 2799 |
| Tabla Función Sensor | 61806 |
| Módulo Gaussiana | 10267 |
| Tabla Función Gaussiana | 30903 |
| GNPA Muestreo | 923 |
| GNPA Remuestreo | 926 |
| Remuestreo | 11515 |
| Palabra-a-memoria | 1883 |
| Registros del *pipeline* | 14002 |
| Memoria Local | 296670 |
| Área total EP | 521452 |

#### 4.3.1.8. Análisis de tiempo de Ejecución y Área

Para analizar el tiempo de ejecución del arreglo se considera la cantidad de ciclos que toma el procesamiento de una partícula y la cantidad de partículas procesadas. En este análisis se considera que cada EP procesa 1024 partículas.

La cantidad de ciclos viene dada por la siguiente expresión:

$$\#ciclos = 1024 \cdot 2, \tag{4.17}$$

La cantidad de datos viene dada por la cantidad de EPs $P$ y la cantidad de partículas que almacena la memoria de grupo:

$$\#datos = 1024 \cdot P, \tag{4.18}$$

Entonces la razón $\#ciclos \setminus \#datos$ viene dada por la expresión:

$$\frac{\#ciclos}{\#datos} = \frac{2}{P}. \tag{4.19}$$

La Fig.4.14 muestra la cantidad de ciclos por dato en función de la cantidad de elementos de procesamiento. Se puede observar que a medida que crece el número de EPs, la cantidad total de partículas procesadas también lo hace y se logra una relación cantidad de ciclos vs cantidad de

partículas procesadas menor a uno. Esto sucede a partir de 2 EPs, lo cual es razonable teniendo en cuenta que cada EP procesa un dato cada 2 ciclos de reloj.

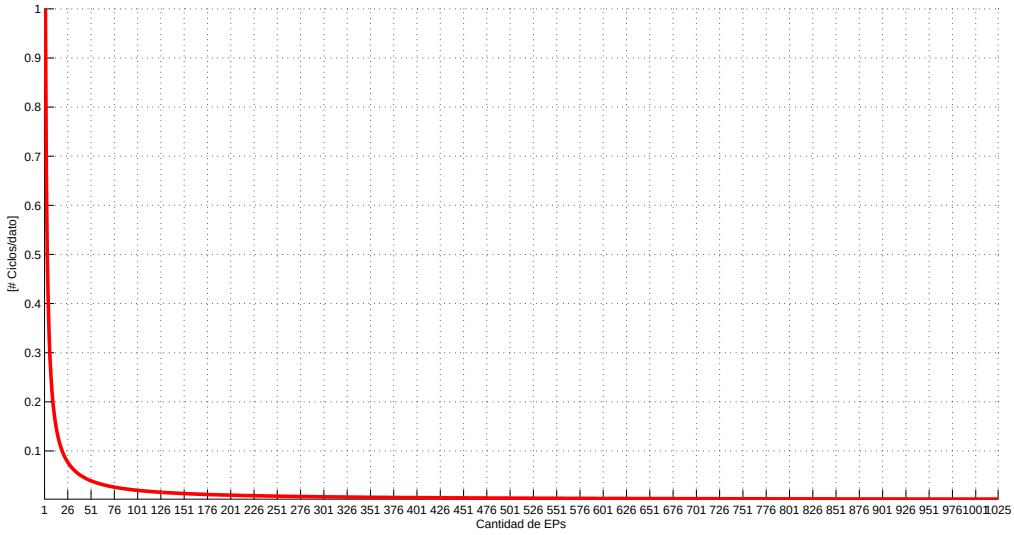


Figura 4.13: Cantidad de ciclos por dato para un arreglo de EPs.

Para realizar un análisis de primer orden se considera la estimación y síntesis lógica del elemento de procesamiento presentada en la Subsección 4.3.1.7. Considerando que el área ocupada es un cuadrado, la Fig. 4.14 muestra la longitud del lado del área estimada de un arreglo de procesadores para la tecnología CMOS 0.13 $\mu$m.

En el caso de que se requiera un procesamiento de 1048576 partículas con la tasa de procesamiento más alta posible, entonces se requieren 1024 EPs, cuyo costo en área es de 23 mm de longitud de lado. El total de las partículas se procesará a una tasa de 0.002 [ciclos/dato].

### 4.3.2. Diseño 2

En esta sección se describe el segundo diseño propuesto, también basado en el modelo de computación *Dataflow*. El diseño de la Sección 4.3.1 si bien permite una tasa de procesamiento alta, también tiene un costo alto en área de silicio, más aún si se considera una arquitectura de múltiples procesadores. El diseño que se presenta en esta sección implementa una fracción del procesamiento como recursos globales. De esta manera se reduce el área que requiere el elemento de procesamiento permitiendo instanciar un número mayor de EPs en un área de silicio dada.

Con el fin de procesar miles de partículas en paralelo en tiempo real, la arquitectura debe explotar el nivel de paralelismo de datos y al mismo tiempo tener en cuenta la estrategia de

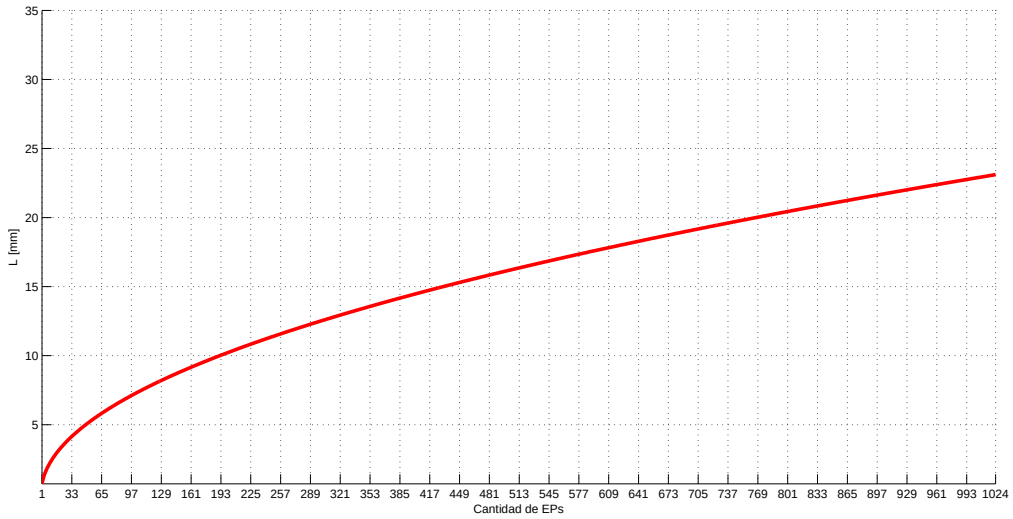


Figura 4.14: Área estimada de un arreglo de EPs para la tecnología CMOS 0.13 $\mu$m.

remuestreo en grupo descripta en el Capítulo 3. En este sentido, se adopta una jerarquía en los niveles de paralelismo. El primer nivel consiste en introducir múltiples elementos de procesamiento (EP) cada uno realizando los pasos del algoritmo que no presentan dependecia de datos. El segundo nivel consiste en agrupar EPs de manera tal que la entrada para el remuestreo está compuesta de la partícula y peso procesados por cada EP dentro de un grupo. Una vez que la iteración del filtro finalizó, el estimado de cada grupo se combina para formar un estimado global y se realiza un intercambio de partículas entre grupos.

La arquitectura propuesta implementa las operaciones más costosas en área en tablas externas (por fuera del arreglo de EPs). Por cada tabla existe un módulo emisor que lee secuencialmente la tabla y realiza una interpolación tal como se describió en la Sección 4.3.1.3. El valor interpolado y la dirección de interpolación son emitidos a todos los arreglos a través de canales. Cada EP computa localmente la dirección de interpolación y compara ésta con la presente en el canal. Si encuentra una equivalencia, toma el dato correspondiente del canal.

La Fig. 4.15 muestra la arquitectura en detalle. En este caso, hay 4 grupos compuestos de 4 EPs. La medición y el recíproco de su varianza $1/\sigma^2$ son comunicados a todos los EPs. Se introducen cuatro recursos globales: tabla cuadrática, tabla raíz cuadrada, tabla sensor y normal. Cada módulo emisor posee 2 buses: *dirección de interpolación* y *dato interpolado*. Además se introducen dentro de cada grupo un módulo de remuestreo, un generador de números pseudoaleatorios y $palabra-a-memoria$. La comunicación entre grupos no se muestra para simplificar la figura.

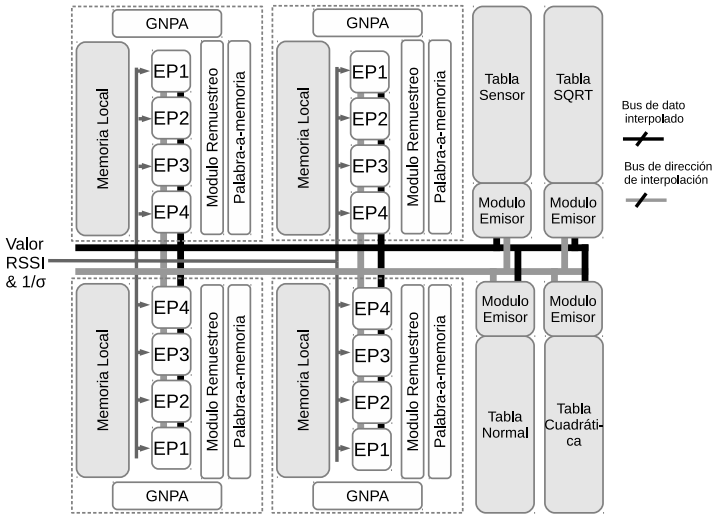


Figura 4.15: Arquitectura VLSI de procesamiento.

Cada grupo tiene su propia memoria local y trabaja sin dependencia de datos con otros grupos excepto cuando se realiza el intercambio de partículas. Por lo tanto, los elementos de procesamiento pertenecientes a un grupo comparten la memoria de datos.

Respecto a la lógica de control, cada grupo de procesamiento tiene su propia lógica que controla la escritura y lectura de memoria. Además cada EP tiene una lógica de control local distribuida en las etapas del *pipeline* similar al diseño descripto en la Sección 4.3.1.

#### 4.3.2.1. Operación del grupo de procesamiento

En modo ejecución, cada EP dentro de un grupo recibe una partícula de memoria principal. Cada módulo emisor emite secuencialmente y sin interrupción la dirección de interpolación y el valor interpolado hacia todos los EPs. Estos datos emitidos permiten que los EPs completen el procesamiento no lineal. Dado que el flujo de operaciones de los elementos de procesamiento trabaja en modo *pipeline*, se puede utilizar una lectura completa de las tablas para procesar varias partículas.

Los datos de salida de cada elemento de procesamiento, partículas trasladadas y sus pesos son almacenados en dos arreglos. Una vez que ambos arreglos son actualizados en su totalidad, sus registros son procesados en forma secuencial por el módulo remuestreo y $palabra-a-memoria$. A medida que el valor de cada registro es procesado, éste es inmediatamente actualizado por el correspondiente EP. La operación de ambos modulos se describe en la Sección 4.3.2.4. La Fig. 4.16 muestra más en detalle la arquitectura del grupo de procesamiento.

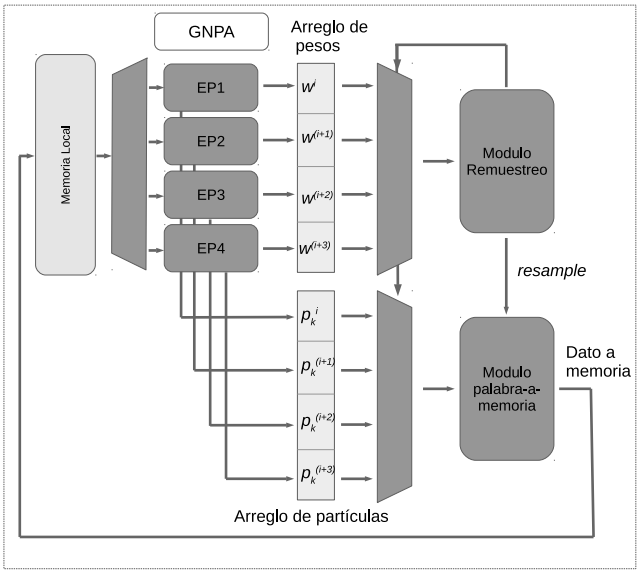


Figura 4.16: Arquitectura del grupo de procesamiento.

Tabla 4.3: Configuración de las funciones lineales a tramos.

| *Función* | $N$ | $M$ | $Q$ | $R$ | $S$ | *Tamaño Kbits* | *Rango de evaluación* | *Error de Interpolación* |
|---|---|---|---|---|---|---|---|---|
| Cuadrática | 9 | 2 | 14 | 17 | - | 7 | [0,40] | $5 \cdot 10^{-4}$ |
| Raíz Cuadrada | 10 | 2 | 11 | 13 | 5 | 11 | [0,3200] | $3 \cdot 10^{-4}$ |
| Sensor | 10 | 2 | 10 | 12 | 1 | 10 | [0,113] | $4 \cdot 10^{-4}$ |
| Normal | 9 | 1 | 10 | 11 | 3 | 5 | [0,5] | $5 \cdot 10^{-3}$ |

#### 4.3.2.2. Diseño de las funciones tabuladas

Al igual que en la Sección 4.3.1.3 las funciones tabuladas se aproximan con una función lineal a tramos junto con la operación de interpolación. Por lo tanto, el módulo emisor implementa la ecuación 4.9. Su arquitectura es similar a la mostrada en la Fig. 4.6. Para este nuevo diseño la palabra de bits que se utiliza tanto para direccionar la memoria como para realizar la interpolación proviene de un contador, de forma tal de recorrer la tabla en forma completa. Además, el dato interpolado y la dirección de interpolación forman un canal de datos. La Fig. 4.17 muestra la arquitectura del módulo emisor.

Se aplicó el análisis de precisión presentado en la Sección 4.3.1 para la implementación de las funciones tabuladas. La Tabla 4.3 muestra la configuración elegida para cada función, donde $N, M, Q$ son el número de bits asignados para la segmentación, interpolación y cuantización de los valores tabulados. Por otra parte, $R$ es la cantidad de bits asignados para el valor interpolado mientras que $S$ es la cantidad de bits que son descartados para el direccionamiento de la próxima tabla.

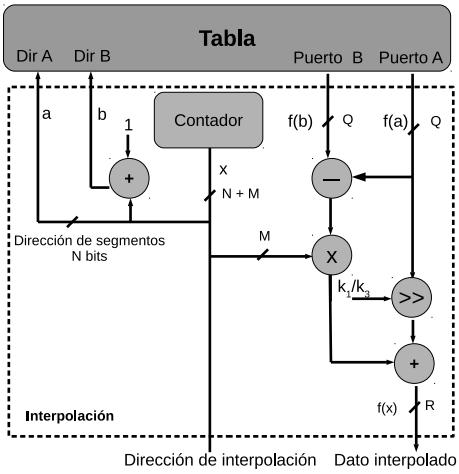


Figura 4.17: Arquitectura para los módulos Raíz Cuadrada, Predicción Potencia y Gaussiana.

#### 4.3.2.3. Micro arquitectura del elemento de procesamiento

Cada elemento de procesamiento realiza los pasos que no presentan dependencia de datos. El procesamiento se divide en varios módulos de manera tal de implementar un *pipeline*. Estos son: *muestreo*, *unidad adquisidora cuadrática*, *unidad adquisidora raíz*, *unidad adquisidora sensor*, *unidad adquisidora normal*. La arquitectura del *pipeline* se muestra en la Fig. 4.18.

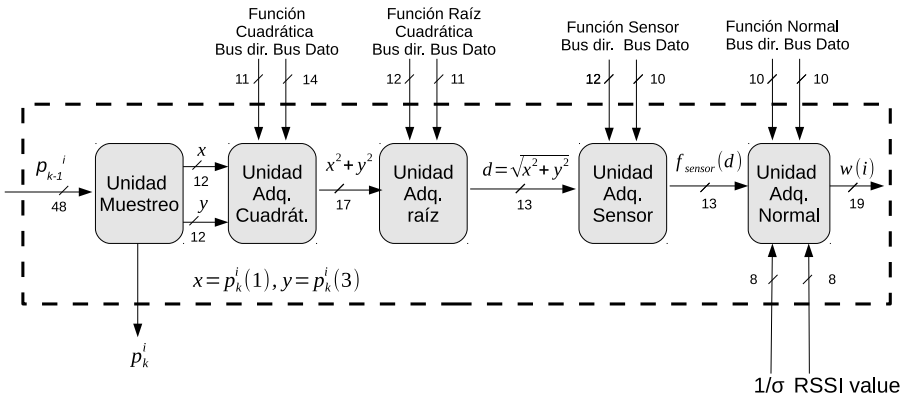


Figura 4.18: Micro-arquitectura del elemento de procesamiento.

La unidad de muestreo recibe datos de la memoria principal. El ancho de palabra de memoria es de 48 bits, donde cada componente del vector partícula tiene 12 bits. Se implementa la misma simplificación del modelo dinámico original descripta en la Sección 4.3.1. El generador de números pseudo-aleatorios se implementa con un registro de desplazamiento con retroalimentación lineal con semilla reconfigurable. El generador es un recurso compartido dentro del

grupo de procesamiento. Cada EP toma un número en su turno correspondiente. Los ochos bits más significativos son usados para la componente $n_x$ y los ocho bits menos significativos para $n_y$. Cada componente del ruido se multiplica por el valor de varianza $W$. Tanto $W$ y $\triangle T$ son registros programables de 8 bits.

Todas las unidades adquisidoras detectan cuando su dato de entrada es igual al presente en el canal de datos. Esta operación se realiza con una operación XOR bit a bit. Cuando se detecta la equivalencia se captura el dato presente en el canal.

La unidad *adquisidora cuadrática* provee como salida la suma de sus entradas al cuadrado. Cuando el valor de $x$ o $y$ es negativo, se computa el complemento a dos. Tanto $|x|$ como $|y|$ tienen 11-bit de ancho de palabra y son comparados con el valor presente en el canal de dirección de interpolación. Una vez que el valor cuadrático es capturado para ambos componentes se computa la suma con un ancho de palabra de 17 bits. El módulo emisor para la función raíz cuadrada posee un canal de dirección de interpolación de 12 bits por lo que los 5 bits menos significativos de $x^2 + y^2$ son descartados por la unidad *adquisidora raíz*. Lo mismo ocurre para el resto de las unidades adquisidoras.

La unidad *adquisidora Normal* genera la palabra a comparar utilizando (4.13) con $\mu$ igual al valor RSSI medido por el receptor de RFID. Una vez que se da la equivalencia, el valor capturado es multiplicado por el recíproco de la desviación estandar como se muestra en (4.14). El recíproco de la desviación estandar y el valor de medición de potencia tienen un ancho de palabra de 8 bits. Por lo tanto los 5 bits menos significativos del dato de entrada son descartados. El dato de salida tiene un ancho de palabra de 19 bits.

#### 4.3.2.4. Módulo Remuestreo

Al igual que en el Diseño 1 se implementó el algoritmo IMH modificado. La Fig. 4.19 muestra la arquitectura del módulo de *remuestreo* y *palabra-a-memoria*. El *arreglo_particulas* se completa con las partículas procesadas por la unidad de muestreo. Ambos arreglos *arreglo_particulas* y *arreglo_pesos* deben estar completos para habilitarse la operación de remuestreo.

La operación de ambos módulos es similar al del Diseño 1. El módulo remuestreo compara el peso de la última partícula almacenada en memoria $W_{prev}$ (previamente multiplicado por $u$) y el peso presente a la salida del multiplexor $muxW$. En base al resultado de esta comparación la partícula a almacenar a memoria puede ser o bien el valor almacenado en $P_{prev}$ o el dato presente a la salida del mux $muxP$.

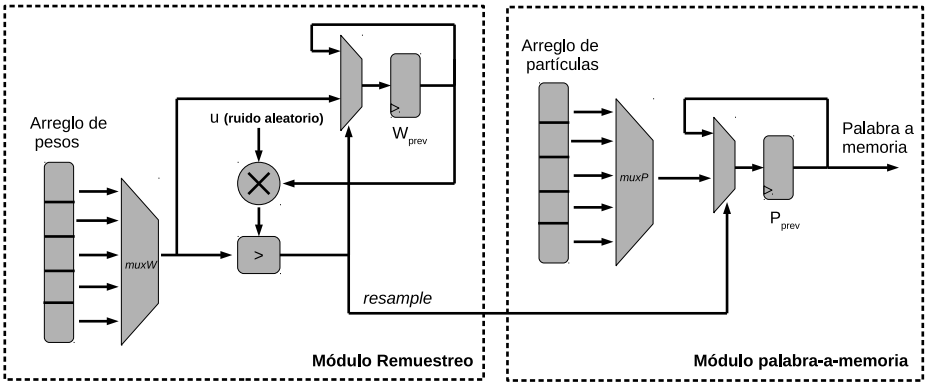


Figura 4.19: Arquitectura del módulo remuestreo y palabra-a-memoria.

#### 4.3.2.5. Tiempo de ejecución

Dado que el tiempo de ejecución de cada módulo es variable, cada EP completará el procesamiento en instantes de tiempo diferentes. La Fig. 4.20 muestra el tiempo de ejecución del flujo de operaciones para un grupo de procesamiento compuesto por dos elementos. En colores diferentes se muestran las operaciones de cada EP que se sincronizan en la operación de remuestreo.

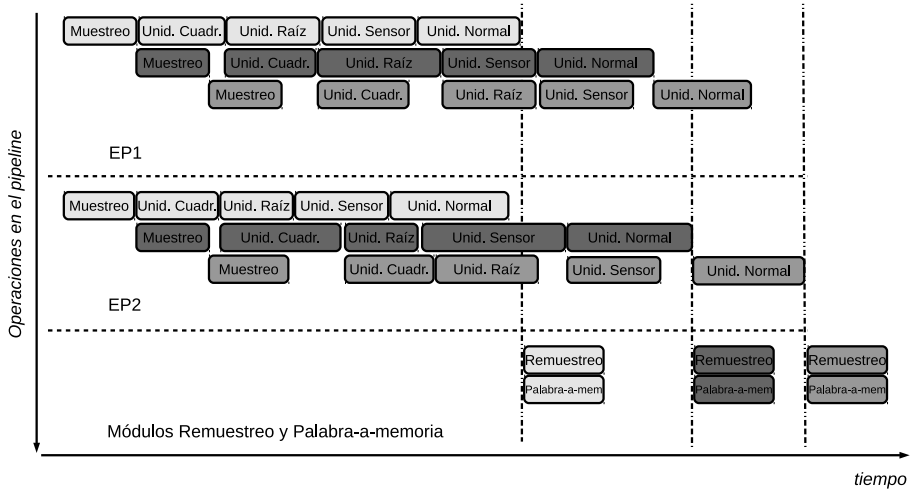


Figura 4.20: Tiempo de ejecución del filtro.

El retardo de un *pipeline* está dado por la etapa que toma más ciclos para completar su procesamiento. En el diseño propuesto la cantidad de ciclos que requieren las etapas es variable por lo tanto el número de ciclos entre datos va a ser variable también.

La cantidad de ciclos que lleva realizar la lectura de una tabla en su totalidad y la interpolación es $2^{N+M}$ donde $N$ es la cantidad de lugares de memoria de la tabla y $M$ los bits dedicados

a la interpolación. Volviendo al retardo del *pipeline* y tomando el peor caso, la unidad adquisidora raíz puede tomar 4096 ciclos para su procesamiento dado que para la tabla asociada $N$ es 9 y $M$ es 2, por lo que se podría considerar que el *pipeline* procesa una partícula cada 4096 ciclos. Sin embargo, en base a simulaciones con conjuntos de datos diferentes, se obtuvo un valor experimental promedio de 1200 ciclos de procesamiento.

La operación de remuestreo requiere de un ciclo por partícula por lo que el número de ciclos para finalizar esta operación dependerá de la cantidad de EPs que se agrupen. Por lo tanto, la tasa de procesamiento total es de $1200 + P$ ciclos por partícula, donde $P$ es el número de EPs en el grupo.

#### 4.3.2.6. Resultados de simulación

Se desarollaron los modelos RTL del elemento de procesamiento y de los módulos *remuestreo*, *palabra-a-memoria*, las tablas, módulo emisión y los generadores de números pseudo-aleatorios. Se simuló el modelo Matlab en punto flotante y el modelo RTL para una potencia de medición de $-55{,}42$ dBm y $\sigma = 2{,}5$ dBm. Según el modelo de propagación de dos rayos para este valor de potencia se corresponden los siguientes valores de distancia: 4,8 m, 5,17 m, 7,7 m, 9,15 m y 14,2 m. La función de densidad de probabilidad tendrá entonces 4 modos. La media de cada uno de estos modos corresponde a uno de los valores mencionados. La Fig. 4.21 muestra la distribución de *pesos* vs *distancia*. Se puede observar que ambos modelos aproximan proveen resultados similares y aproximan correctamente a la función de densidad de probabilidad.

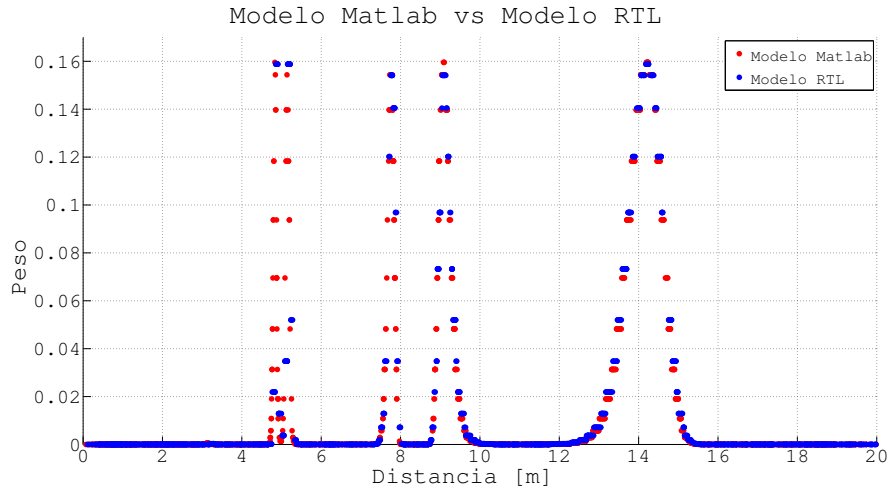


Figura 4.21: Pesos vs. distancia.

Por otra parte se utilizó el escenario 2-D presentado en la Sección 4.2 para mostrar el desem-

peño dinámico de la arquitectura propuesta. En este caso, se utiliza un arreglo de tres antenas RF ubicadas en las posiciones $[0, 0]$, $[20, 0]$, $[0, -20]$. Se desarrolló un modelo Matlab equivalente en punto fijo y se utilizó este mismo en vez de su homólogo RTL para reducir el tiempo de simulación.

Se utilizó un total de 4096 partículas, las cuales son uniformemente distribuidas en una región delimitada por los intervalos $[-20$ m $, 20$ m $]$ y $[0, \pi]$ radianes al comienzo de la simulación. La velocidad de las partículas se inicializó en forma aleatoria con distribución uniforme en el intervalo $[17, 7]$ m/s para $\dot{x}$ and $[7, -3]$ m/s para $\dot{y}$. El estado inicial del móvil a seguir es $\mathbf{x}_0 = [-8$ m $, 12$ m/s $, 10$ m $, -2$ m/s $]$ y $\Delta T = 0{,}1$ s.

La Fig. 4.22 muestra la trayectoria del objetivo a seguir en linea verde y los resultados de la simulación para el modelo Matlab en punto flotante y fijo en lineas de color rojo y negro respectivamente. Ambos modelos proveen resultados similares. El error que comete el modelo en punto fijo respecto al modelo en punto flotante es de 0.4 m. Lo cual

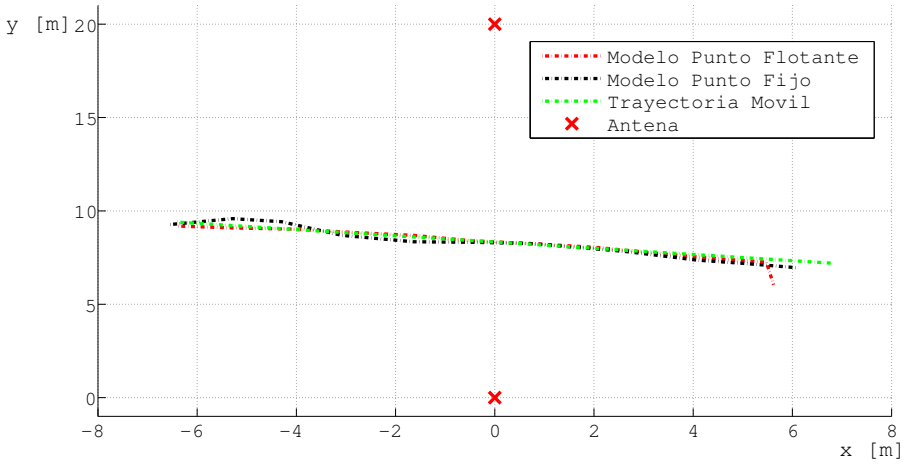


Figura 4.22: Seguimiento de un móvil con tres antenas RF.

El error es aproximadamente del orden de 0.5 [m]. La principal fuente de error en la estimación al comienzo de la simulación es la existencia de múltiples hipótesis. Dado que la estimación es la media de la función de densidad de probabilidad el hecho de que existan múltiples hipótesis ocasiona un corrimiento en la estimación respecto a la ubicación del móvil. A medida que se van realizando las iteraciones estas son filtradas y la estimación es más precisa.

Al igual que en el Diseño 1 se simularon los modelos Matlab en punto flotante y en punto fijo del diseño propuesto con el fin de demostrar el filtrado iterativo de las hipótesis. Se utilizó el

escenario presentado al principio de este capítulo para un total de 16384 partículas. En la Fig. 4.23 se muestra la aproximación de la función de densidad de probabilidad para ambos modelos. El instante de simulación corresponde a 0.5 s. La función posee tres modos cuya distancia radial es aproximadamente 15 m, 8 m y 8.6 m. La posición del móvil [−14.2 m, 6.8 m] se indica con un círculo verde cuya distancia a la antena es de 15.74 m.

En el instante de tiempo indicado la potencia de la señal recibida es de -67 dBm. Para esta potencia recibida existen 4 hipótesis cuyas distancias radiales son: 8 m, 8.6 m ,15 m, 21 m, 38 m. Sin embargo a causa del filtrado iterativo de las hipótesis sólo se conservan algunas de ellas.

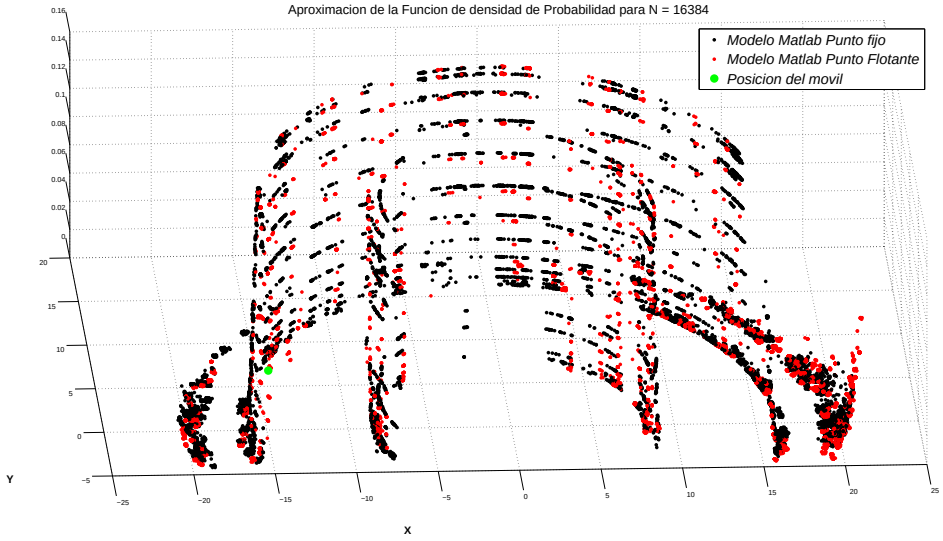


Figura 4.23: Simulación de Matlab del modelo en punto fijo y flotante.

El modelo RTL fue simulado utilizando la herramienta Modelsim con el fin de verificar el correcto funcionamiento del hardware descripto. La Fig. 4.24 muestra una ventana de tiempo de simulación suficiente para ilustrar el comportamiento del *pipeline*. En la figura aparece el cursor trazado con una línea amarilla para el instante 12225 ns. Para dicho instante de tiempo el valor de los registros *p_x* y *p_y* de la *unidad adquisidora cuadrática* son 193 y 252 respectivamente. Estos valores se corresponden con los procesados anteriormente por el módulo *muestreo* siguiendo la operación del *pipeline*. Cuando la simulación llega a 27000 ns aproximadamente, las señales *add_equal_x* y *add_equal_y* indican que se encontró igualdad con el dato presente en el canal de direcciones de la tabla cuadrática, por lo que se actualizan los valores almacenados en los registros *x_squared*, *y_squared* y la suma *rango_sq* a 582, 992 y 1574 respectivamente. Luego, una vez que la *unidad adquisidora raíz* terminó de procesar su dato actual, evento que sucede cerca de 45000 ns de simulación, está habilitada a procesar el nuevo dato de entrada que resulta ser

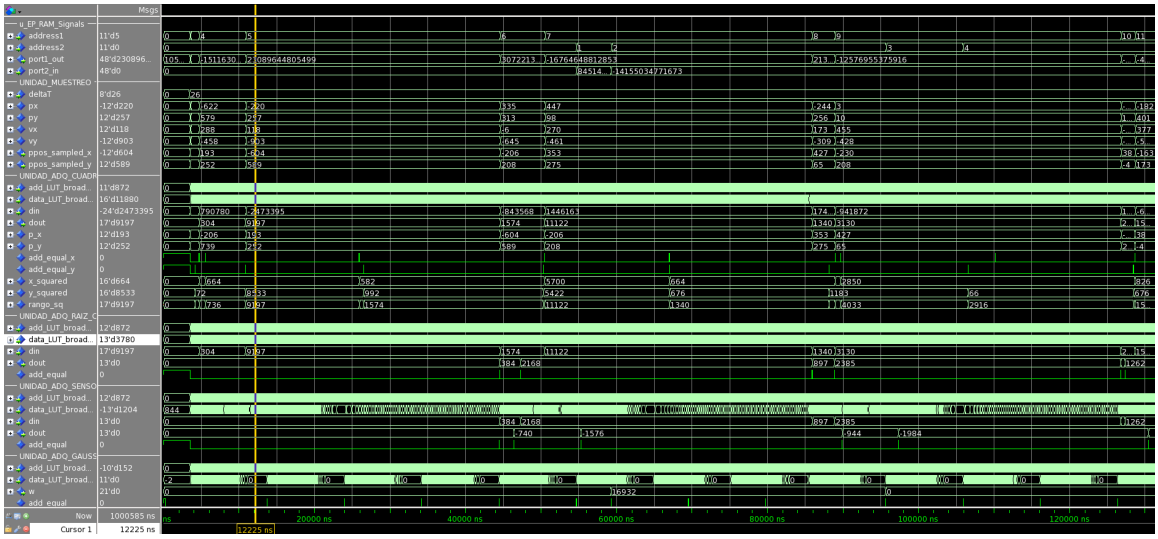

Figura 4.24: Simulación de señales del *pipeline* con la herramienta Modelsim.

1574. Lo mismo sucede para la *unidad adquisidora cuadrática* que toma los valores −604 y 589. Esta misma operación sucede con el resto de las etapas.

#### 4.3.2.7. Resultado de síntesis lógica y estimación de área

Se generó una síntesis lógica del modelo RTL del elemento de procesamiento utilizando Synopsis DC Compiler y la tecnología CMOS 0.13 $\mu$m. Dado que el arreglo está compuesto de múltiples elementos de procesamiento es deseable conocer el área que requiere esta unidad básica. La Tabla 4.4 muestra el área requerida por el elemento de procesamiento y sus módulos.

Tabla 4.4: Resultado de síntesis lógica del elemento de procesamiento.

| **Módulo** | **Área [$\mu$m$^2$]** |
|---|---|
| Muestreo | 11985 |
| Acq. Cuadrática | 2296 |
| Acq. raíz | 750 |
| Acq. Sensor | 637 |
| Acq. Normal | 6589 |
| Registros del *pipeline* | 21378 |
| **Área total EP** | **43635** |

Por otra parte, la Tabla 4.5 muestra el área destinada a los recursos compartidos del grupo. Al igual que el diseño anterior no se dispone de celdas de memoria SRAM para la síntesis lógica por lo que la estimación se logra utilizando el área de una celda de memoria establecida en [34] con una eficiencia de 0.7. La memoria local del grupo de procesamiento posee 1024 direcciones

con un ancho de palabra de 48 bits.

Tabla 4.5: Resultado de síntesis lógica y estimaciones de área de recursos compartidos del grupo de procesamiento.

| **Módulo** | **Área** [$\mu$**m**$^2$] |
|---|---|
| GNPA Muestreo | 957 |
| GNPA Remuestreo | 926 |
| Remuestreo | Depende del número de EPs |
| Palabra-a-memoria | Depende del número de EPs |
| Memoria Local | 296670 |

El área de los módulos *Palabra-a-memoria* y *Remuestreo* depende del número de EPs que se instancien por grupo. La Tabla 4.6 muestra el área que ocupan estos módulos para distinto número de EPs.

Tabla 4.6: Resultado de síntesis lógica de módulo remuestreo y palabra-a-memoria.

| **Número de EPs** | **Área Remuestreo** [$\mu$**m**$^2$] | **Área Palabra-a-memoria** [$\mu$**m**$^2$] |
|---|---|---|
| 1 | 11515 | 1883 |
| 2 | 14582 | 5109 |
| 4 | 16345 | 8927 |
| 8 | 19637 | 16617 |
| 16 | 26232 | 31690 |
| 32 | 40019 | 59300 |
| 64 | 65033 | 121154 |
| 128 | 116850 | 240415 |

Por último la Tabla 4.7 muestra la estimación de área para los recursos compartidos globales, es decir compartidos a todos los grupos de procesamiento.

Tabla 4.7: Estimaciones de área de los recursos compartidos globales.

| **Módulo** | **Área** [$\mu$**m**$^2$] |
|---|---|
| Tabla Función Cuadrática | 43264 |
| Tabla Función Raíz Cuadrada | 67986 |
| Tabla Función Sensor | 61806 |
| Tabla Función Gaussiana | 30903 |

#### 4.3.2.8. Análisis de tiempo de ejecución y área

Para este análisis se considera que cada grupo tiene una memoria local de 1024 partículas y un tiempo de ejecución promedio del *pipeline* de 1200 ciclos por partícula, valor que se obtuvo en base a simulaciones. Por lo tanto la cantidad de ciclos promedio que toma el procesamiento de una partícula viene dada por la siguiente expresión:

$$\#ciclos = \frac{1024 \cdot (1200 + P)}{P}, \tag{4.20}$$

donde $P$ es el número de EPs en el grupo.

La cantidad de datos viene dada por la cantidad de grupos $N$ y la cantidad de partículas que almacena la memoria de grupo:

$$\#datos = 1024 \cdot N. \tag{4.21}$$

Entonces la razón $\#ciclos \setminus \#datos$ viene dada por la expresión:

$$\frac{\#ciclos}{\#datos} = \frac{(1200 + P)}{N \cdot P}. \tag{4.22}$$

La Fig. 4.25 muestra la cantidad de ciclos por dato en función de la cantidad de grupos y elementos de procesamiento. Se puede observar que a medida que el número de grupos crece, la cantidad total de partículas procesadas también lo hace y se requiere una menor cantidad de EPs por grupo para lograr una misma relación de cantidad de ciclos vs cantidad de partículas.

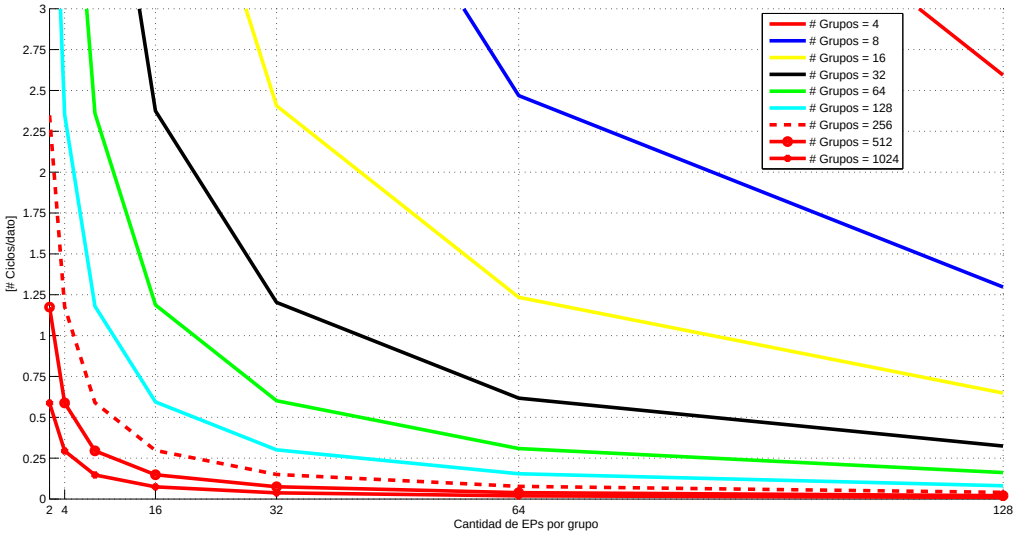


Figura 4.25: Cantidad de ciclos por dato del arreglo considerando 1200 ciclos por partícula.

En cuanto al análisis de área se utiliza la estimación de área y síntesis lógica del elemento de procesamiento y los recursos compartidos presentada en la Subsección 4.3.2.7. Estos resultados permiten conocer el área del grupo de procesamiento para distintos números de EPs.

Considerando que el área ocupada es un cuadrado, la Fig. 4.26 muestra la longitud del lado

del área estimada del arreglo de procesadores para distinto número de grupos de procesamiento y EPs para la tecnología CMOS 0.13 $\mu$m.

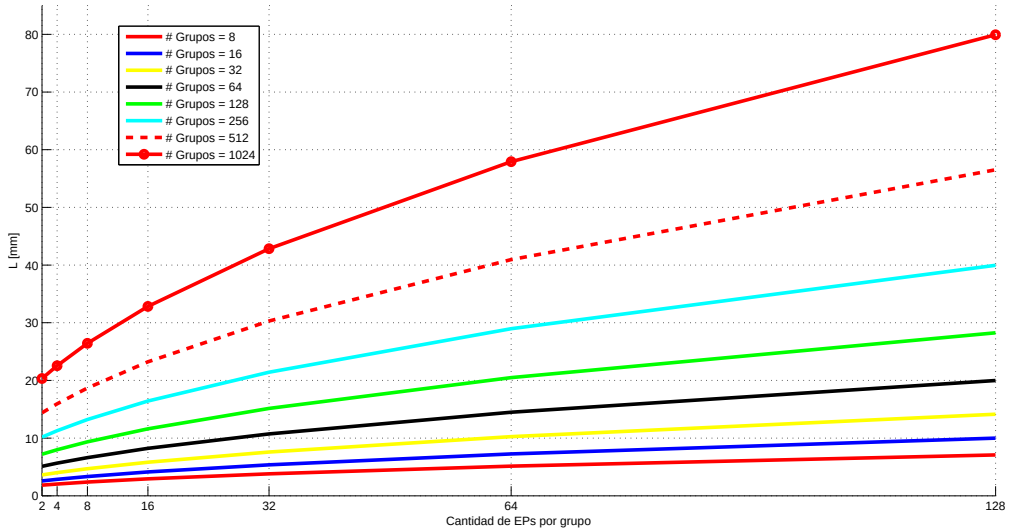


Figura 4.26: Área estimada del arreglo para la tecnología CMOS 0.13 $\mu$m.

Para el procesamiento de 1048576 partículas se elije una arquitectura de 1024 grupos con 2 EPs cada uno de ellos. El costo en área de esta arquitectura es de una longitud del lado de 20.3 mm para la tecnología CMOS 0.13 $\mu$m y la tasa de procesamiento es 0.29 ciclos por dato. En comparación con un diseño de 1024 EPs del Diseño 1 el área de silicio requerida es un 28 % menor y la tasa de procesamiento es 145 veces menor. En aplicaciones donde el área es un parámetro crítico de diseño y no así la velocidad de procesamiento esta arquitectura resulta beneficiosa.

Existe un límite en el número de EPs a instanciar por grupo. Por encima del mismo, la arquitectura ya no provee una mejora en área. A medida que se incrementa el número de EPs por grupo, se reduce el tiempo de ejecución y aumenta el área. Sin embargo la reducción en la tasa de procesamiento no es lo suficiente para justificar el aumento de área. Ejemplo de esta situación es el caso de una arquitectura de 1024 grupos de 128 EPs cada uno. El área tiene 79 mm de lado y una tasa de 0.01 ciclos por dato. En la misma área se podrían instanciar 6984 EPs del Diseño 1, lo que resultaría en el procesamiento de 6 millones de partículas a una tasa de 0.0003 [ciclos/dato].

## 4.4. Diseño basado en Arquitecturas SIMD

El diseño propuesto en este trabajo si bien se basa en el concepto *Single Instruction Multiple Data* (SIMD) [35] y en procesadores de propósito general, también cuenta con harware dedicado para el filtro de partículas. A continuación se describe la arquitectura, el conjunto de instrucciones, las unidades de procesamiento, la lógica de control y los módulos dedicados. Por último se presentan los resultados de simulación.

### 4.4.1. Diseño 3

El diseño propuesto se muestra en la Fig. 4.27. La arquitectura cuenta con un arreglo de procesadores de propósito general y de baja complejidad donde cada uno de ellos computa sobre un vector partícula diferente. El ancho de palabra de las componentes del vector es de 10 bits por lo que cada vector es de 40 bits.

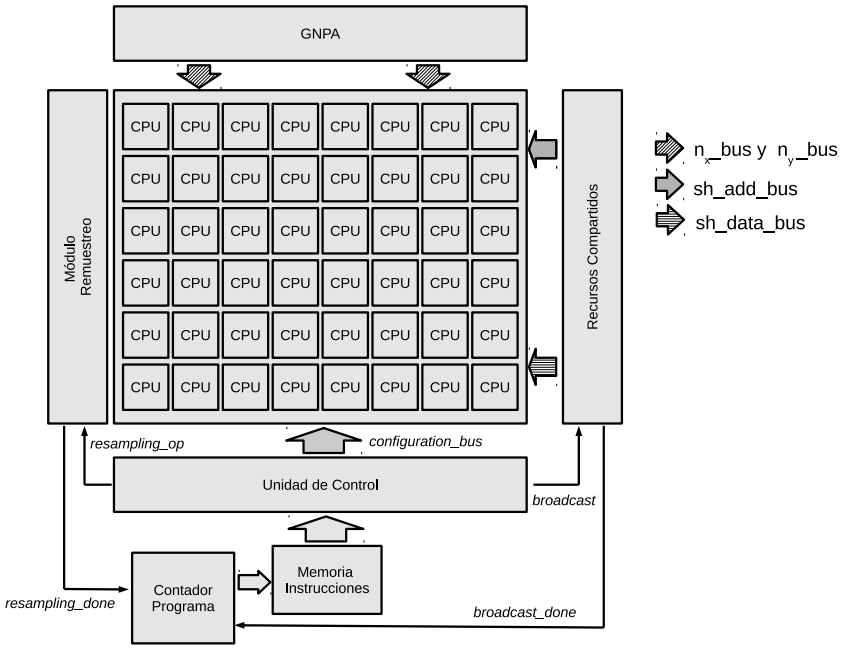


Figura 4.27: Arquitectura SIMD propuesta.

Al igual que en el Diseño 2, se introdujeron funciones tabuladas. Estas tablas se definen como recursos globales multiplexados en tiempo de forma tal de reducir el área dedicada al procesador. Además, para reducir la cantidad de canales, se utiliza un multiplexor de datos interpolados por lo que un único canal de direcciones *sh_address_bus* y de datos *sh_data_bus* llega a todas las unidades de procesamiento. Existen 6 canales adicionales: *data_bus_wr*, *data_bus_rd*,

, *-obs*, $n_x$*_bus*, $n_y$*_bus*, *configuration_bus* cuya función se explica más adelante. Todos tienen un ancho de bits de 10 bits excepto por *data_bus_wr*, *-obs* y *configuration_bus* que tienen 40, 8 y 20 bits respectivamente.

La arquitectura también cuenta con generadores de números pseudo-aleatorios, incluidos en el módulo GNPA, y un módulo de remuestreo.

El control se soporta sobre una memoria de instrucciones y una unidad de control que se encarga de decodificar la intrucción y generar las señales de control apropiadas qué definen que operación debe ejecutar el arreglo de procesadores. Estas señales son transmitidas a través del canal *configuration_bus*.

#### 4.4.1.1. Operación de la arquitectura

La operación de esta arquitectura es similar a cualquier procesador excepto por la operación de remuestreo. La memoria de instrucciones es leída secuencialmente por el contador de programa. La instrucción obtenida de la memoria es decodificada por la unidad de control y luego transmitida, a través del canal de configuración, a todos los procesadores los cuales proceden a la ejecución de la misma.

Se dispone de instrucciones especiales para la operación de remuestreo y lectura de tablas. Cuando se requiere leer una tabla la lógica de control habilita el módulo Recursos Compartidos con la señal *broadcast* y el contador de programa no se incrementa hasta que la lectura de la tabla no se haya completado en su totalidad. Dicho evento es indicado con la señal *broadcast_done*. En cuanto al remuestreo, la lógica de control habilita el módulo remuestreo con la señal *resampling_op*. El módulo se encarga de realizar la operación por fila y cuando finaliza indica su estado al contador de programa. Este último no se incrementa hasta que la señal *resampling_done* no cambia de estado.

#### 4.4.1.2. Conjunto de instrucciones

El diseño del procesador está basado en la arquitectura de procesadores con juego de instrucciones reducidas como es el caso de MIPS [35]. Sin embargo a diferencia de este tipo de procesador no cuenta con un *pipeline*. El conjunto de instrucciones diseñados incluye:

- Instrucciones de registros (tipo-R),
- Instrucción de emisión (tipo-E),
- Instrucción de remuestreo (tipo-RM).

El ancho de palabra de la instrucción es de 22 bits. Las instrucciones de registros operan con registros internos del procesador. La instrucción de emisión selecciona cuál de las funciones tabuladas se va a utilizar y controla el multiplexado en tiempo de las mismas. El valor interpolado y la dirección de interpolación son enviados a través de los canales *sh_data_bus* y *sh_address_bus*. Cada procesador del arreglo realiza una comparación entre el dato presente en el canal de direcciones y su dato interno. Cuando se detecta una equivalencia se registra el dato presente en el canal de datos. La instrucción de remuestreo habilita el módulo asignado a dicha operación y coloca en estado inactivo a los procesadores.

El formato de las instrucciones se detalla en la Tabla 4.8 donde la función del campo *Opcode* y *Función* se describe en la Sección 4.4.1.4. Los campos *RS* y *RT* definen los registros fuente de la operación. El campo *RC* define el recurso global que será emitido por el canal. Por último *RD* define el registro destino.

Tabla 4.8: Formato de las instrucciones.

| Opcode | RS | RT | RD | RC | Función |
|---|---|---|---|---|---|
| 2 bits | 3 bits | 3bits | 7 bits | 3 bits | 4 bits |

Con el juego de instrucciones diseñado es posible computar el modelo dinámico descripto en la Sección 4.3.1.2 y todas las demás operaciones involucradas en el algoritmo del filtro de partículas. En el apéndice se detalla el conjunto total de instrucciones y el código ensamblador del algoritmo del filtro de partículas.

#### 4.4.1.3. Procesador

El procesador está compuesto por una unidad arítmetica lógica (UAL), registros y un árbol de compuertas OR para detectar resultado nulo. La UAL permite realizar la operación de suma, resta, complemento a 2, desplazamientos aritméticos hacia izquierda y derecha y también movimiento de datos entre registros. La arquitectura del procesador se muestra en la Fig. 4.28.

Las entradas de datos del procesador son el canal de escritura de datos *data_bus_wr*, los canales $n_x$*_bus* y $n_y$*_bus*, el valor negativo del dato medición $-obs$ y el canal de dirección y de datos de interpolación *sh_address_bus* y *sh_data_bus*. A través del canal *data_bus_wr* se transfiere el vector partícula que es almacenado en los registros internos del procesador $p_x$, $p_y$, $v_x$, $v_y$. Por otro lado los números pseudo-aleatorios se almacenan en los registros $n_x$ y $n_y$. Los registros $reg1$, $reg2$ y $reg3$ son de propósito general.

Las entradas de control se dividen en dos grupos: entradas de control de la UAL y de los multiplexores de los registros internos. Este último está compuesto por las señales $CtrlMux1$ y $CtrlMux2$. El primer grupo está compuesto por las señales: $C_{in}$, $enSubs$, $S_5$, $S_4$, $S_3$, $S_2$,

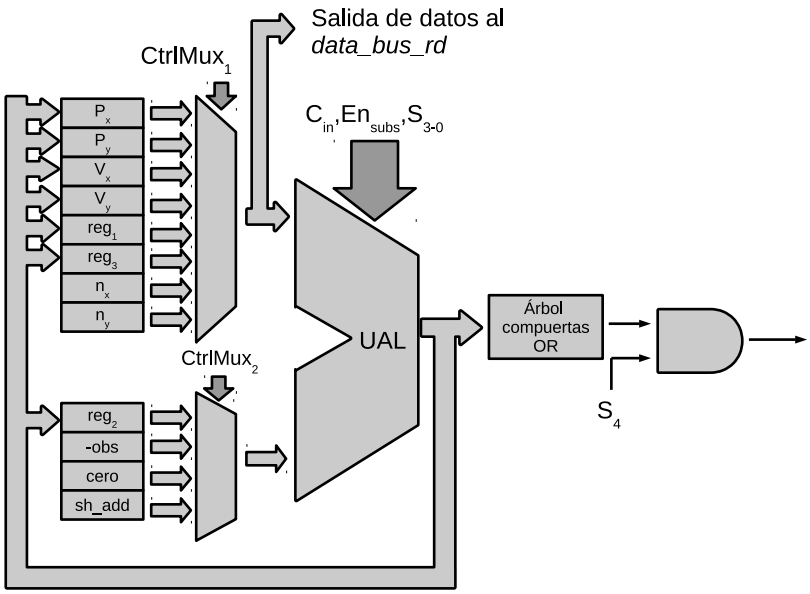


Figura 4.28: Arquitectura del Procesador.

$S_1$, $S_0$. La señal $C_{in}$ y *enSubs* asigna el valor al bit de acarreo y habilita la operación resta respectivamente. La función de las entradas $S_{5-0}$ se detalla a continuación:

- $S_5$ : controla la operación complemento a dos,
- $S_4$ : habilita la salida del árbol de compuertas OR,
- $S_3$ : control del desplazamiento a izquierda o derecha,
- $S_2$ : selección de operando A ó B para operación desplazamiento.
- $S_1$ : selección de la salida: salida aritmético/lógica o entrada A/B.
- $S_0$ : selecciona la salida del módulo lógico o aritmético de la UAL.

La unidad de control es la encargada de decodificar la instrucción y establecer un valor para todas estas señales a través del canal *configuration_bus*.

La arquitectura de la UAL se muestra en la Fig. 4.29. El módulo aritmético y lógico se muestran en la Fig. 4.30 donde $G_{9-0}$ es el resultado de la suma o resta y el módulo lógico se diseñó para trabajar con datos en complemento a dos.

El hardware asociado a la escritura de las componentes del vector partícula y de los registros de propósito general se muestra en la Fig. 4.31. El campo *RD* está compuesto por las líneas de selección: *sel_p*$_x$, *sel_p*$_y$, *sel_v*$_x$, *sel_v*$_y$, *sel_reg*$_1$, *sel_reg*$_2$ y *sel_reg*$_3$. Si en la instrucción se selecciona

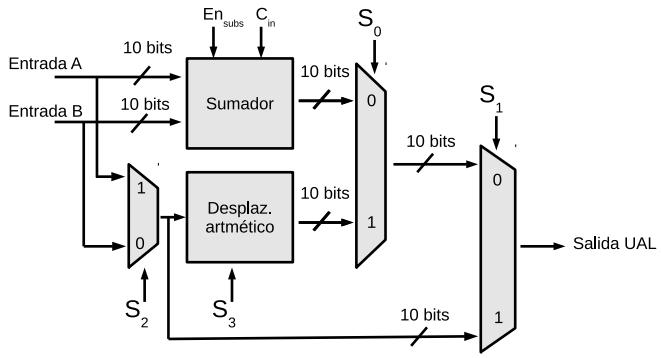


Figura 4.29: Arquitectura de la UAL.

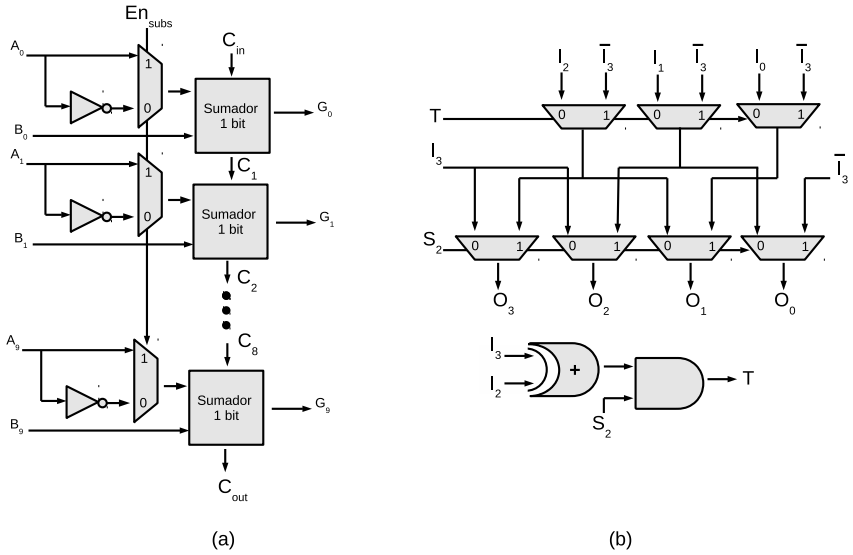


Figura 4.30: Micro-arquitectura: a) Módulo Aritmético y b) Lógico para un ancho de palabra de 4 bits.

alguno de los registros como destino, entonces la salida de la UAL se almacena en el mismo. Para los registros: $p_x$, $p_y$, $v_x$, $v_y$ se adiciona un multiplexor para inicializar el vector partícula a través del canal *data_bus_wr*.

El registro $reg_2$ soporta a la instrucción de emisión. El procesador realiza la operación resta entre el dato presente en el canal de direcciones *sh_address_bus* y un dato interno definido por el multiplexor cuyo control viene dado por $CtrlMux1$. Cuando se detecta el resultado cero (utilizando el árbol de compuertas OR) se registra el dato presente en el canal *sh_data_bus*.

La salida del multiplexor controlado por la señal $CtrlMux1$ está conectada al canal de lectura *data_bus_rd*. De esta manera es posible transmitir el vector partícula y el valor del peso que estará almacenado en alguno de los registros de propósito general: $reg_1$ y $reg_3$.

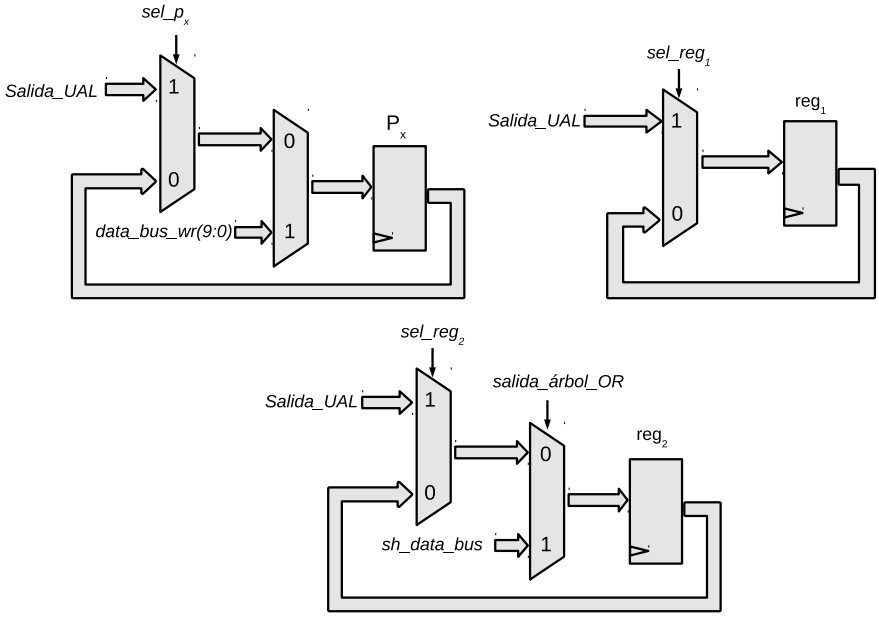


Figura 4.31: Escritura de los registros internos del procesador.

#### 4.4.1.4. Diseño de la unidad de control

El diseño de la unidad de control se realizó siguiendo la metodología propuesta en [35]. La relación entre las operaciones de la UAL y sus señales de control se muestran en la Tabla 4.9.

Tabla 4.9: Relación entre las señales de control de la UAL y sus operaciones.

| *Acción UAL* | $C_{in}$ | $En_{subs}$ | $S_{5-0}$ |
|---|---|---|---|
| Inactivo | 0 | 0 | 000000 |
| sumar | 0 | 0 | 000000 |
| restar | 1 | 1 | 000000 |
| Desplz. Izq. entrada A | 0 | 0 | 001001 |
| Desplz. Izq. entrada B | 0 | 0 | 001101 |
| Desplz. Der. entrada A | 0 | 0 | 000001 |
| Desplz. Der. entrada B | 0 | 0 | 000101 |
| Mover entrada A | 0 | 0 | 000010 |
| Mover entrada B | 0 | 0 | 000110 |
| Compl. a 2 | 1 | 1 | 100000 |

La unidad de control genera las señales de control del procesador teniendo como entrada el campo *Opcode* junto con *Función* de la palabra instrucción. Si bien la práctica más común es utilizar varios niveles de control, en este caso se utilizó uno solo al ser un conjunto de instrucciones bastante reducido.

En la Tabla 4.10 se muestra la relación entre el campo *Opcode* junto con *Función* y las señales

de control de la UAL.

Tabla 4.10: Relación entre los campos *Opcode* y *Función* y las señales de control de la UAL.

| *Instrucción Opcode* | *Opcode* | *Operación Instrucción* | *Campo Función* | *Acción UAL deseada* | *Entrada Control UAL* |
|---|---|---|---|---|---|
| inactivo | 00 | inactivo | 0000 | idle | 00000000 |
| tipo-R | 00 | sumar | 0001 | sumar | 00000000 |
| tipo-R | 00 | restar | 0010 | restar | 11000000 |
| tipo-R | 00 | desplz. izq. UAL A | 0011 | desplz. izq. UAL A | 00001001 |
| tipo-R | 00 | desplz. izq. UAL B | 0100 | desplz. izq. UAL B | 00001101 |
| tipo-R | 00 | desplz. der. UAL A | 0101 | desplz. der. UAL A | 00000001 |
| tipo-R | 00 | desplz. der. UAL B | 0110 | desplz. der. UAL B | 00000101 |
| tipo-R | 00 | mover UAL A | 0111 | mover UAL A | 00000010 |
| tipo-R | 00 | mover UAL B | 1000 | mover UAL B | 00000110 |
| tipo-R | 00 | compl. a 2 | 1001 | compl. a 2 | 11100000 |
| tipo-E | 01 | emisión de un recurso compartido | 0010 | restar | 11010000 |
| tipo-RM | 10 | remuestrear | 0000 | inactivo | 00000000 |

Por otro lado los campos $RS$ y $RT$ se mapean uno a uno con las señales de control del procesador $CtrlMux1$ y $CtrlMux2$ respectivamente. El campo $RD$ no tiene codificación y controla la escritura de los registros internos tal como se describió previamente.

La unidad de control también recibe como entrada al campo $RC$ a partir de la cual genera señales de control destinadas al manejo de los recursos compartidos y del multiplexor del canal de direcciones y de datos. La Tabla 4.11 resume la composición del canal de configuración.

Tabla 4.11: Canal de configuración.

| $CtrlMux2$ | $CtrlMux1$ | $C_{in}$ | $En_{subs}$ | $S_{5-0}$ | $RD$ |
|---|---|---|---|---|---|
| 2 bits | 3 bits | 1 bits | 1 bits | 6 bits | 7 bits |

La unidad de control también genera las siguientes señales:

- *broadcast* : habilita el contador de lectura de las tablas compartidas,
- *resampling_op* : inicia operación de remuestreo.
- *shr_id* : controla el multiplexor del *sh_data_bus*.
- $shr_{5-0}$ : señales decodificadas que controlan la lectura del módulo recursos compartidos.

#### 4.4.1.5. Módulo Remuestreo

Al igual que en los dos diseños descriptos previamente, el algoritmo de remuestreo implementado es el IMH modificado. El flujo de datos se muestra en la Fig. 4.32. Es posible disponer de un módulo remuestreo por fila, para una cantidad dada de filas o para el arreglo completo.

Para explicar el funcionamiento se supone un módulo de remuestreo por fila. Cuando se decodifica la instrucción de remuestreo, se genera la señal *resampling_op* que inicia la operación y se deshabilita el incremento del contador de programa. A continuación se realiza la lectura del peso $w$ y dato partícula del primer procesador de la fila a través del canal de lectura *data_bus_rd* y se almacenan en $w_g$ y $P_g$ respectivamente.

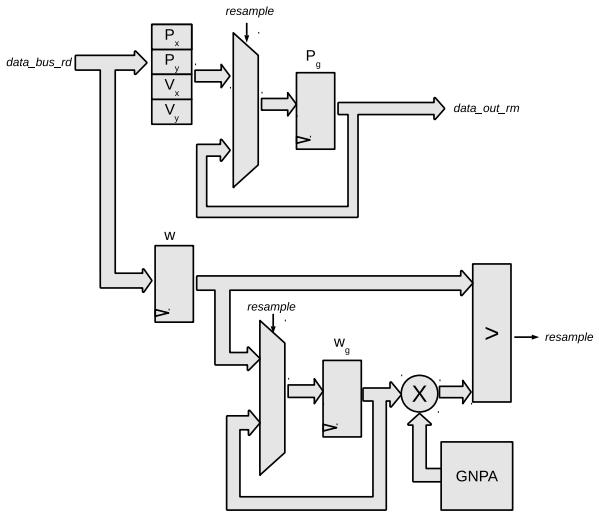


Figura 4.32: Flujo de datos para el módulo de remuestreo.

Luego se procede a la lectura del peso del siguiente procesador y se actualiza el valor de la señal *resample* de acuerdo al algoritmo de remuestreo de la Sección 4.3.1. Si el valor actualizado de *resample* es 0, se realiza la lectura del vector partícula y se almacena en el registro $P_g$. El valor de $w$ se almacena en $W_g$ y finalmente el dato remuestreado se escribe en el mismo procesador a través del canal escritura *data_bus_wr*. Si el valor asignado a la señal *resample* es 1 entonces no se lee el dato partícula y se escribe en el procesador el dato almacenado en el registro $P_g$.

La máquina de estados se muestra en la Fig. 4.33. Como el canal *data_bus_rd* es de 10 bits se introducen los estados *lectura_W*, *lectura_*$P_x$, *lectura_*$P_y$, *lectura_*$V_x$ y *lectura_*$V_y$ para realizar la lectura de la partícula y su peso en 5 ciclos. Una vez que se finalizó con el remuestreo de la fila se asigna el valor 1 a la señal *resampling_done* cuyo propósito es habilitar nuevamente el incremento del contador de programa.

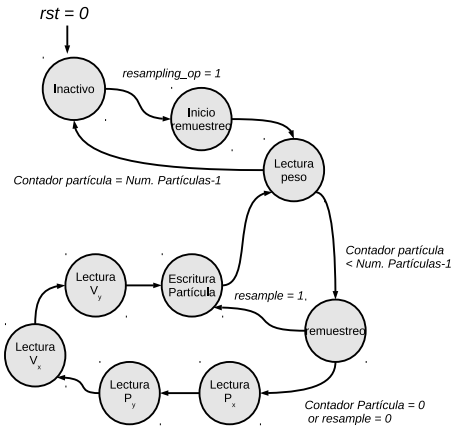


Figura 4.33: Máquina de estados para módulo de remuestreo.

#### 4.4.1.6. Módulos como recursos compartidos

Se introdujeron dos tipos de recursos compartidos: funciones tabuladas y multiplicadores. Las funciones se muestran en la Tabla 4.12. Su implementación está basada en el análisis de precisión presentado en la Sección 4.3.1. Recordar que $N, M, Q$ son el número de bits asignados para la segmentación, interpolación y cuantización de los valores tabulados. Por otra parte, $R$ es la cantidad de bits asignados para el valor interpolado mientras que $S$ es la cantidad de bits que son descartados para el direccionamiento de la próxima tabla.

Tabla 4.12: Configuración de las funciones lineales a tramos para arquitectura SIMD.

| *Función* | $N$ | $M$ | $Q$ | $R$ | $S$ | *Size Kbits* | *Rango de evaluación* | *Interp. Error* |
|---|---|---|---|---|---|---|---|---|
| Square | 9 | 0 | 10 | 11 | 1 | 5 | [0,40] | $2 \cdot 10^{-3}$ |
| Sqrt | 9 | 1 | 9 | 10 | 0 | 4.5 | [0,3200] | $1{,}5 \cdot 10^{-3}$ |
| Sensor | 9 | 1 | 9 | 10 | 1 | 4.5 | [0,113] | $1 \cdot 10^{-3}$ |
| Normal | 9 | 0 | 10 | 10 | 0 | 5 | [0,5] | $9 \cdot 10^{-3}$ |

Además de las funciones tabuladas se multiplexaron en tiempo las multiplicaciones de una variable por una constante. Para el algoritmo del filtro de partícula estas operaciones son:

- $V \cdot \Delta T$ para el cómputo modelo dinámico,
- $(Pot_{medicion} - Pot_{observacion}) \cdot 1/\sigma$ para la evaluación de la distribución normal.

La constante $\Delta T$ se representa con 8 bits de resolución y 4 bits para $1/\sigma$. Por lo tanto el resultado de las multiplicaciones es de 18 bits y 12 bits. Se genera el vector de entrada para todos los valores posibles de la variable utilizando un contador. Luego se transmiten por el canal

de direcciones el vector generado y por el canal de datos el resultado de la multiplicación. Para transmitir este último por el canal *sh_data_bus* cuyo ancho es de 10 bits es necesario descartar los bits menos significativos.

#### 4.4.1.7. Memoria de instrucciones y contador de programa

En modo ejecución el contador de programa se incrementa de manera automática y la ejecución de cada instrucción toma un ciclo de reloj. Sin embargo, si la instrucción que se está ejecutando es del tipo-E o tipo-RM el contador de programa queda inhabilitado a continuar incrementando su valor hasta que la operación de lectura de la tabla compartida o el remuestreo hayan finalizado.

#### 4.4.1.8. Operación de entrada y salida de datos

Los datos son extraídos del arreglo a través del canal de lectura *data_ bus_rd* que tiene como destino el módulo remuestreo y un puerto de salida del diseño. Una señal de habilitación recorre el arreglo y permite que cada procesador coloque en el canal el vector partícula y peso en 5 ciclos de reloj.

Para ingresar datos partículas en el arreglo se utiliza el canal de escritura *data_ bus_wr*. Un multiplexor selecciona qué dato se transmite: el presente en el puerto de entrada *data_in* o la salida del módulo remuestro *data_out_rm*. De esta manera el canal se utiliza tanto para inicializar el arreglo como para transmitir las partículas remuestreadas.

#### 4.4.1.9. Carga de números pseudo aleatorios

La escritura de los registros internos $n_x$ y $n_y$ se realiza utilizando el mismo mecanismo que para la entrada de datos partícula. Por otra parte, se solapan la carga de los números pseudo aleatorios con la operación de remuestreo evitando así el costo en tiempo.

#### 4.4.1.10. Resultados de simulación

Al igual que en los dos diseños anteriores se simularon los modelos Matlab en punto flotante y en punto fijo para un total de 16384 partículas. En la Fig. 4.34 se muestra la aproximación de la función de densidad de probabilidad que ambos modelos proveen para el instante 0.12 s. La función posee dos modos que corresponden a dos anillos radiales concéntricos a la posición de la antena de radio 11 m y 7 m aproximadamente. Se indica también la posición del móvil [-12.8 m, 1.2 m] con un círculo verde cuya distancia a la antena es de 12.85 m.

Para el instante de tiempo seleccionado, la potencia de la señal recibida es de $-47.69$ dBm. Si se observa la figura del modelo de dos rayos Fig. 4.1, para esta potencia existen 5 hipótesis correspondientes a las distancias radiales: 11 m ,7 m, 5.5 m, 5 m y 4.3 m. Sin embargo, debido al filtrado iterativo de las hipótesis sólo se conservan las de mayor radio.

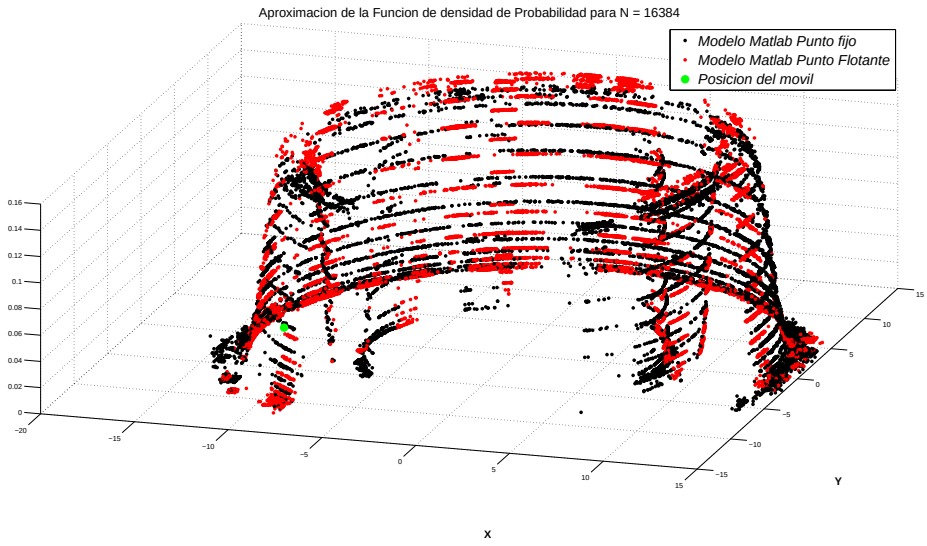


Figura 4.34: Simulación de Matlab del modelo en punto fijo y flotante.

Se desarrolló un modelo RTL del diseño propuesto con un arreglo de procesadores configurable. Se desarrolló también en Matlab un ensamblador que mapea el mnemónico de las instrucciones a código binario. Se describió el algoritmo del filtro de partículas con el juego de instrucciones diseñado y se verificó cada una de las instrucciones involucradas.

La Fig. 4.35 muestra la simulación del modelo RTL utilizando la herramienta ModelSim. La primer señal que se observa es el contador de programa. El resto de las señales se dividen por pertenecia al elemento de procesamiento. A medida que avanza el contador así también lo hace la ejecución de las instrucciones por parte de los procesadores. Por ejemplo, cuando el contador de programa tiene el valor 4, la instrucción corresponde a la suma de *reg2* con $p_x$ y el resultado se almacena en el registro *reg1*. Además también es posible observar que la instrucción correspondiente al valor 14 del contador de programa es del tipo emisión ya que la señal *broadcast_signal* está en uno y se detiene el incremento del contador de programa.

#### 4.4.1.11. Resultado de síntesis lógica

Se realizó una síntesis lógica del modelo RTL de un arreglo de 64 elementos de procesamiento utilizando Synopsis DC Compiler y la tecnología CMOS 0.13$\mu$m. La Tabla 4.13 muestra el área

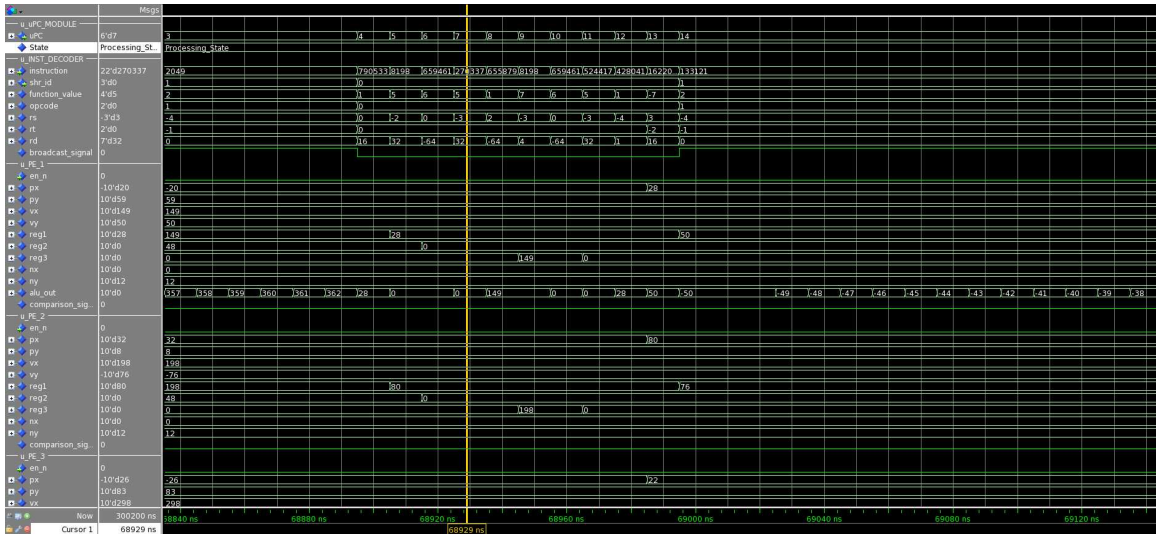

Figura 4.35: Simulación de señales del *pipeline* con la herramienta Modelsim.

que ocupa el elemento de procesamiento y el área total que ocupa un arreglo de 64 EPs.

Tabla 4.13: Resultado de síntesis del elemento de procesamiento.

| **Módulos del Elemento de Procesamiento** | **Área [$\mu$m$^2$]** |
|---|---|
| UAL | 1000 |
| Registros | 2826 |
| Multiplexores y Lógica Adicional | 1887 |
| **Área total EP** | **5713** |
| **Área total Arreglo de EPs** | **365632** |

La Tabla 4.14 muestra el área que ocupa los recursos compartidos del arreglo.

Tabla 4.14: Resultado de síntesis y estimaciones de área de los recursos compartidos.

| **Módulos del Arreglo** | **Área** [$\mu$**m**$^2$] |
|---|---|
| Contador de Programa | 390 |
| Decodificador Instrucciones | 999 |
| Generador de Números Pseudo-Aleatorios | 3000 |
| Generador de Números Pseudo-Aleatorios Remuestreo | 763 |
| Remuestreo | 10366 |
| Lógica de control de la programación | 301 |
| Tabla Función Cuadrática | 30903 |
| Tabla Función Raíz Cuadrada | 27812 |
| Tabla Función Sensor | 27812 |
| Tabla Función Gaussiana | 30903 |
| Memoria de Instrucciones | 10198 |
| **Área total Recursos Compartidos** | **143448** |

#### 4.4.1.12. Análisis de tiempo de ejecución y área

Para analizar el tiempo de ejecución del arreglo se considera la cantidad de ciclos que consumen las instrucciones que conforman el algoritmo de filtro de partículas incluido el remuestreo.

La cantidad de ciclos para procesar una partícula está se expresa como:

$$\#ciclos = 3871 + (5 \cdot P), \tag{4.23}$$

donde $P$ es la cantidad de EPs. Las instrucciones toman 3871 ciclos para su ejecución mientras que $(5 \cdot P)$ es la cantidad de ciclos que toma la operación de remuestreo por fila.

Por otro lado la cantidad de datos viene dada por el tamaño del arreglo, es decir $P \cdot P$. La razón $\#ciclos/\#datos$ se expresa en la ecuación

$$\frac{\#ciclos}{\#datos} = \frac{3871 + (5 \cdot P)}{P \cdot P}. \tag{4.24}$$

La Fig.4.36 muestra la cantidad de ciclos por dato en función de la cantidad de procesadores por fila de la matriz de EPs. Se puede observar que a medida que crece el número de procesadores, la cantidad total de partículas procesadas también lo hace y se logra una relación cantidad de ciclos vs cantidad de partículas procesadas menor a uno.

Para realizar un análisis de primer orden del área requerida por el arreglo se consideran los resultados presentados en la sección 4.4.1.11. Para estimar el área de arreglos de mayor tamaño se realiza una extrapolación.

La Fig. 4.37 muestra el área estimada del arreglo de procesadores para distinto número de elementos de procesamiento para la tecnología CMOS 0.13$\mu$m. Entonces para 1024 EPs por fila se requiere un área de 100 mm de longitud de lado a una tasa de 0.0085 ciclos por dato. El diseño propuesto requiere de un lado de área de silicio 3.3 veces mayor que el Diseño 1 y procesa

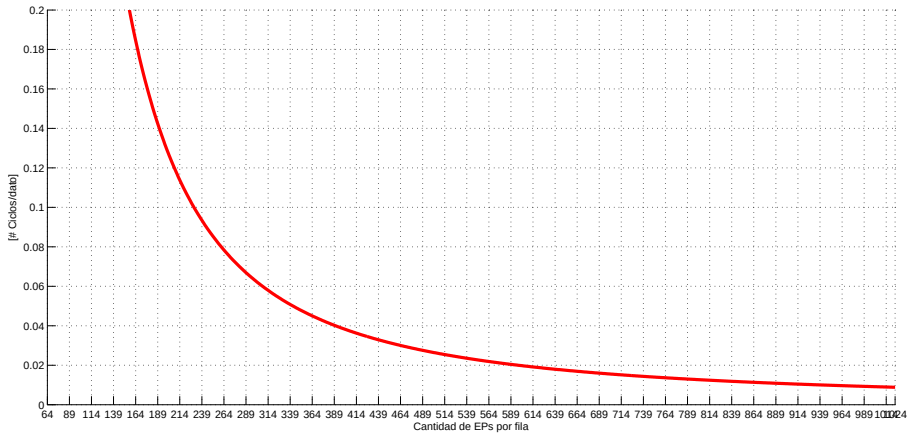


Figura 4.36: Cantidad de ciclos por dato vs cantidad de procesadores por fila del arreglo.

el total de las partículas 4.4 veces más lento.

Si bien este diseño tiene como ventaja su estructura modular y regular y que su elemento de procesamiento tiene una versatilidad mayor en cuanto a capacidad de cómputo, no provee una mejora en la tasa de procesamiento y el área.

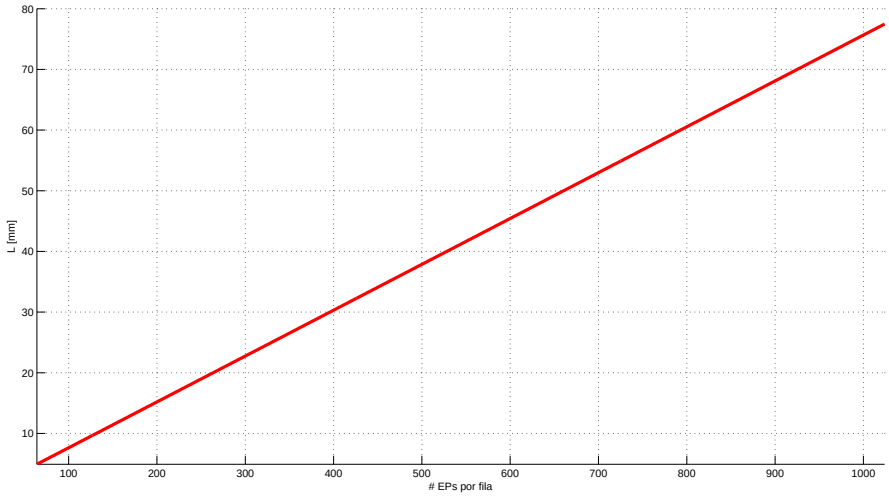


Figura 4.37: Área estimada del arreglo para la tecnología CMOS 0.13$\mu m$.

## 4.5. Comparación de las arquitecturas diseñadas

En esta sección se realiza una comparación de las tres arquitecturas diseñadas para el filtrado de partículas en tiempo real. La comparación se realiza en base a dos parámetros: tiempo de ejecución y área.

### 4.5.1. Comparación del Diseño 2 respecto al Diseño 1

En primer instancia se comparan los Diseños 1 y 2 sin contabilizar el área destinada a los recursos compartidos de forma tal de poder observar el impacto de la reducción de área efectuada en el elemento de procesamiento del Diseño 2.

La Tabla 4.15 muestra los resultados del elemento de procesamiento del Diseño 1 para una memoria de 1024 partículas, donde $n$ es la cantidad de EPs instanciados. La columna *Área Proc.* muestra el área destinada a elementos de procesamiento sin contabilizar la memoria.

Tabla 4.15: Área y Tiempo de Ejecución del EP del Diseño 1.

| **n** | **Área Memoria 1024 Part.** [$\mu^2$] | **Área EP D1** [$\mu^2$] | **Area Procesamiento** [$\mu^2$] | **Tiempo de Ejecución** |
|---|---|---|---|---|
| 1 | 296670 | 224782 | 224782 | 2048 |

Tomando como punto de referencia los resultados de la Tabla 4.15, la Tabla 4.16 muestra la relación de área y tiempo de ejecución del Diseño 2 respecto al Diseño 1, donde $n$ es la cantidad

de EPs instanciados del Diseño 2 dentro del grupo y $p$ es la cantidad de partículas procesadas. Se considera también un área de memoria de 1024 partículas. La columna *Área Proc. D1/D2* muestra la relación de área de procesamiento sin considerar la memoria. Por último la columna *Área D1/D2* muestran la relación de área considerando las memorias de partículas.

Tabla 4.16: Tabla comparativa del Diseño 2 respecto al Diseño 1.

| **n** | **p** | **Área Mem. 1024 Part.** [$\mu^2$] | **Área EP D2** [$\mu^2$] | **Área Proc.** [$\mu^2$] | **Tiempo de Ejec.** | **Tiempo D2/D1** | **Área Proc. D1/D2** | **Área D1/D2** |
|---|---|---|---|---|---|---|---|---|
| 1 | 1024 | 296670 | 43635 | 43635 | 1229824 | 600.5 | 5.15 | 1.53 |
| 2 | 1024 | 296670 | 43635 | 87270 | 615424 | 300.5 | 2.58 | 1.36 |
| 4 | 1024 | 296670 | 43635 | 174540 | 308224 | 150.5 | 1.29 | 1.11 |
| 8 | 1024 | 296670 | 43635 | 349080 | 154624 | 75.5 | 0.64 | 0.81 |
| 16 | 1024 | 296670 | 43635 | 698160 | 77824 | 38 | 0.32 | 0.52 |
| 32 | 1024 | 296670 | 43635 | 1396320 | 39424 | 19.25 | 0.16 | 0.31 |
| 64 | 1024 | 296670 | 43635 | 2792640 | 20224 | 9.88 | 0.08 | 0.17 |
| 128 | 1024 | 296670 | 43635 | 5585280 | 10624 | 5.19 | 0.04 | 0.09 |
| 256 | 1024 | 296670 | 43635 | 11170560 | 5824 | 2.84 | 0.02 | 0.05 |
| 512 | 1024 | 296670 | 43635 | 22341120 | 3424 | 1.67 | 0.01 | 0.02 |
| 1024 | 1024 | 296670 | 43635 | 44682240 | 2224 | 1.09 | 0.01 | 0.01 |

En principio se observa que el área del EP del Diseño 2 es aproximadamente 5 veces más pequeña con respecto al EP del Diseño 1, lo que implica que la estrategia de multiplexar en tiempo ciertos recursos de hardware resultó exitosa. Analizando ahora la relación de tiempos y área para distinto número de EPs instanciados se puede observar que en el caso del Diseño 2 un grupo de procesamiento con mas de 4 EPs instanciados ya no provee ninguna ventaja en términos de área respecto a un sólo EP del Diseño 1. Para el caso mencionado la relación de área de procesamiento es 1.29 y considerando la memoria la relación es de 1.11. Por otra parte, para $n$ igual a 8 las relaciones mencionadas poseen un valor menor a 1 lo cual indica que no hay reducción en área.

En cuanto al tiempo de ejecución el Diseño 2 resulta más lento respecto al Diseño 1 debido al multiplexado en tiempo de los recursos compartido y se acerca a uno a medida que se instancian más elementos de procesamiento.

La Tabla 4.17 muestra la comparación entre ambos diseños, ahora considerando igual número de veces que se instancia el EP del Diseño 1 y el grupo de procesamiento del Diseño 2 conformado por 2 EPs. Aquí si se considera el área destinada a los recursos compartidos dentro del grupo y los globales, que son aquellos compartidos por todos los grupos. El parámetro $n$ indica cantidad de veces que se instancian ambos diseños y el parámetro $p$ indica la cantidad de partículas procesadas. La relación de área total evidencia una mejora en el área de silicio requerida para 4 o más grupos instanciados del Diseño 2. Además dicha mejora presenta un comportamiento asintótico para el valor de 1.29.

Tabla 4.17: Tabla comparativa para distinto número de grupos de 2 EPs.

| **n** | **p** | **Área Proc.** [$\mu^2$] | **Área Mem. 1024 Part.**[$\mu^2$] | **Área Rec. Comp. del grupo** [$\mu^2$] | **Área Rec. Comp. Globales** [$\mu^2$] | **Área Total D2** [$\mu^2$] | **Área Total D1** [$\mu^2$] | **Área Total D1/D2** | **Tiempo D2/D1** |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1024 | 87270 | 296670 | 21574 | 203959 | 609473 | 521452 | 0.85 | 300.5 |
| 2 | 2048 | 87270 | 296670 | 21574 | 203959 | 1014987 | 1042904 | 1.02 | 300.5 |
| 4 | 4096 | 87270 | 296670 | 21574 | 203959 | 1826015 | 2085808 | 1.14 | 300.5 |
| 8 | 8192 | 87270 | 296670 | 21574 | 203959 | 3448071 | 4171616 | 1.20 | 300.5 |
| 16 | 16384 | 87270 | 296670 | 21574 | 203959 | 6692183 | 8343232 | 1.24 | 300.5 |
| 32 | 32768 | 87270 | 296670 | 21574 | 203959 | 13180407 | 16686464 | 1.26 | 300.5 |
| 64 | 65536 | 87270 | 296670 | 21574 | 203959 | 26156855 | 33372928 | 1.27 | 300.5 |
| 128 | 131072 | 87270 | 296670 | 21574 | 203959 | 52109751 | 66745856 | 1.28 | 300.5 |
| 256 | 262144 | 87270 | 296670 | 21574 | 203959 | 10401554 | 133491712 | 1.28 | 300.5 |
| 512 | 524288 | 87270 | 296670 | 21574 | 203959 | 207827127 | 266983424 | 1.28 | 300.5 |
| 1024 | 1048576 | 87270 | 296670 | 21574 | 203959 | 415450295 | 533966848 | 1.28 | 300.5 |

La Tabla 4.18 muestra los resultados de comparar, en tiempo y área, el grupo de procesamiento del Diseño 2 conformado por 4 EPs y el EP del Diseño 1, teniendo en cuenta que se instancian igual cantidad de veces ambos diseños. Observando las relaciones de área se concluye que no se logra una reducción de área utilizando este grupo de procesamiento. En la Tabla 4.16 se observó que se logra una reducción en el área de procesamiento instanciando como máximo 4 EPs por grupo. Sin embargo, cuando se contabilizan los recursos compartidos se observa que no existe tal reducción.

Tabla 4.18: Tabla comparativa para distinto número de grupos de 4 EPs.

| **n** | **p** | **Área Proc.** [$\mu^2$] | **Área Mem. 1024 Part.** [$\mu^2$] | **Área Rec. Comp. del grupo** [$\mu^2$] | **Área Rec. Comp. Globales** [$\mu^2$] | **Área Total D2** [$\mu^2$] | **Área Total D1** [$\mu^2$] | **Área Total D1/D2** | **Tiempo D2/D1** |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1024 | 174540 | 296670 | 27155 | 203959 | 702324 | 521452 | 0.74 | 150.5 |
| 2 | 2048 | 174540 | 296670 | 27155 | 203959 | 1200689 | 1042904 | 0.86 | 150.5 |
| 4 | 4096 | 174540 | 296670 | 27155 | 203959 | 2197419 | 2085808 | 0.94 | 150.5 |
| 8 | 8192 | 174540 | 296670 | 27155 | 203959 | 4190879 | 4171616 | 0.99 | 150.5 |
| 16 | 16384 | 174540 | 296670 | 27155 | 203959 | 8177799 | 8343232 | 1.02 | 150.5 |
| 32 | 32768 | 174540 | 296670 | 27155 | 203959 | 16151639 | 16686464 | 1.03 | 150.5 |
| 64 | 65536 | 174540 | 296670 | 27155 | 203959 | 32099319 | 33372928 | 1.04 | 150.5 |
| 128 | 131072 | 174540 | 296670 | 27155 | 203959 | 63994679 | 66745856 | 1.04 | 150.5 |
| 256 | 262144 | 174540 | 296670 | 27155 | 203959 | 127785399 | 133491712 | 1.04 | 150.5 |
| 512 | 524288 | 174540 | 296670 | 27155 | 203959 | 255366839 | 266983424 | 1.04 | 150.5 |
| 1024 | 1048576 | 174540 | 296670 | 27155 | 203959 | 510529719 | 533966848 | 1.04 | 150.5 |

De los resultados mostrados se concluye que dependiendo de cuál sea el objetivo que se persigue ya sea reducir el área o el tiempo de ejecución, resultará conveniente el elemento de procesamiento del Diseño 2 en el primer caso (teniendo en cuenta el límite de 2 EPs por grupo) y el del Diseño 1 para el segundo.

### 4.5.2. Comparación del Diseño 3 respecto al Diseño 1

En este caso nuevamente se considera como punto de referencia el EP correspondiente al Diseño 1. La Tabla 4.19 muestra los resultados para distintos tamaños de la matriz de elementos de procesamiento, donde $n$ es la cantidad de EPs por fila y $p$ es la cantidad de partículas procesadas que es igual a $n^2$ dado que se considera una matriz cuadrada. El área de cada matriz se compara con el área que requiere la cantidad de elementos de procesamiento del Diseño 1 para obtener igual cantidad de partículas procesadas.

Tabla 4.19: Tabla comparativa del Diseño 3 respecto al Diseño 1.

| **n** | **p** | **Área EP D3** [$\mu^2$] | **Área Proc.** [$\mu^2$] | **Tiempo de Ejec.** | **Tiempo D3/D1** | **Área de Proc. D1/D3** | **Área D1/D3** |
|---|---|---|---|---|---|---|---|
| 32 | 1024 | 5713 | 5850112 | 4031 | 1.97 | 0.04 | 0.0870 |
| 64 | 4096 | 5713 | 23400448 | 4191 | 2.05 | 0.04 | 0.0886 |
| 128 | 16384 | 5713 | 93601792 | 4511 | 2.20 | 0.04 | 0.0889 |
| 256 | 65536 | 5713 | 374407168 | 5151 | 2.52 | 0.04 | 0.0891 |
| 512 | 262144 | 5713 | 1497628672 | 6431 | 3.14 | 0.04 | 0.0891 |
| 1024 | 1048576 | 5713 | 5990514688 | 8991 | 4.39 | 0.04 | 0.0891 |

En principio se observa que el EP del Diseño 3 resulta aproximadamente 39 veces más pequeño en comparación con el del Diseño 1. Por lo tanto la estrategia de utilizar el multiplexado en tiempo de los recursos y considerar una arquitectura de un procesador estándar en vez de una arquitectura del tipo *Dataflow* es adecuada para reducir el área. Considerando ahora una matriz compuesta de estos elementos se puede observar que no provee una mejora en área ni en tiempo de ejecución en comparación con el Diseño 1. Esta afirmación se mantiene para matrices de diversos tamaños. Se observa también que a medida que crece la cantidad de EPs por fila aumenta el tiempo de ejecución dado que el remuestreo es secuencial y se realiza por fila.

## 4.6. Resumen

En este capítulo se describieron tres diseños que abordan la problemática de filtrado de partículas en tiempo real. Los primeros dos diseños se basan en el modelo computacional *Data-flow* mientras que el tercero en el modelo SIMD. Para cada uno de ellos, se describió su arquitectura global y la micro-arquitectura de los módulos que la integran. Se desarrolló el modelo RTL para cada diseño y se presentan resultados de simulación que verifican el funcionamiento del mismo.

Con el objetivo de conocer el alcance de cada arquitectura se realizaron comparaciones de los Diseños 2 y 3 respecto al 1 teniendo como parámetros el área y el tiempo de ejecución. Se observa que el Diseño 2, en comparación con el EP del Diseño 1, resulta más lento pero presenta una ventaja en el área que requiere. Dicha ventaja se mantiene siempre y cuando la cantidad de EPs a instanciar por grupo no sea mayor a 2. Todo esto implica que a la hora de diseñar una arquitectura para el filtrado de partículas se debe tener en cuenta que si la restricción de diseño es el tiempo de ejecución entonces convendrá implementar una arquitectura basada en el EP del Diseño 1. Por otro lado, si la restriccón es el área, una arquitectura basada en grupos de procesamiento del Diseño 2 resultará la opción adecuada.

Por último se compará el EP del Diseño 3 con respecto al Diseño 1 y se observa que no presenta ventaja respecto al área y el tiempo de ejecución.

En cuanto a la complejidad en la implemantación de los diseños. El Diseño 2 es el que presenta mayor complejidad debido a la cantidad de canales que incluye y la cantidad de líneas de datos. Dicha complejidad aumenta a medida que se incrementa la cantidad de grupos de procesamiento. El Diseño 3, si bien también dispone de recursos compartidos, presenta menor complejidad debido a que la cantidad de canales para estos recursos se redujo a dos y su diseño regular facilita la síntesis física. Por último, el Diseño 1 es el que presenta menor complejidad de los tres: una vez realizada la sítesis física de un EP se instancia el mismo las veces que sea necesario.

En cuanto al soporte y accesibilidad de las celdas básicas de la librería de diseño CMOS 0.13 $\mu$m, la misma fue obtenida a partir del servicio que provee MOSIS siendo el fabricanta IBM. El mayor obstáculo encontrado fue la no disponibilidad de las celdas de memoria SRAM.

# Capítulo 5

# Conclusiones

Este capítulo resume las principales ideas y contribuciones de esta memoria. Además también se describe la dirección del trabajo futuro.

En esta tesis se ha abordado la problemática existente en el filtrado de partículas en tiempo real. Cuando el número de partículas es del orden de los miles o más, se presenta un cuello de botella en el tiempo de ejecución de la operación de remuestreo debido a su ejecución secuencial. Debido a ello surge como una necesidad la exploración e implementación de arquitecturas que permitan el procesamiento del algoritmo en tiempo real. Con este objetivo en mente, en primer lugar se implementó una revisión de los algoritmos de remuestreo y las estrategias desarrolladas para su paralelización.

Un análisis de dos estrategias fundamentales para la aceleración de los algoritmos de remuestreo se presentó en el Capítulo 3. Ambas estrategias se basan en el mismo concepto: si se reduce la cantidad de partículas, se logra una reducción en el tiempo de ejecución de los algoritmos de remuestreo secuenciales. La primer estrategia propone la distribución de la operación de remuestreo en múltiples elementos de procesamiento permitiendo una evaluación paralela. La segunda estrategia propone una versión mejorada de la operación muestreo que permite una reducción en el número total de partículas sin afectar la precisión del filtro.

La desventaja del remuestreo distribuido es la degradación en su desempeño debido a su evaluación paralela lo que puede llevar a una pérdida de diversidad en cada grupo de partículas y estimaciones locales inexactas. Esta problemática se ve aliviada por la introducción de esquemas de comunicación. En este trabajo, los esquemas de comunicación se clasifican como determinísticos y no determinísticos. Los esquemas determinísticos presentan requisitos de implementación más simples y un tiempo de ejecución menor pero su principal desventaja es que puede perder diversidad en el conjunto de partículas. Los esquemas no determinísticos presentan un desempenño similar al filtro de partículas estandar pero su implementación se vuelve cada

vez más compleja a medida que el número de elementos de procesamiento crece.

Ambas estrategias pueden combinarse para obtener un filtro de partículas paralelo con menor degradación en su desempeño. En este sentido, se desarrolló un modelo de alto nivel el cual es útil cuando se consideran diferentes compromisos entre tiempo de ejecución y recursos computacionales. Además se presentaron simulaciones de diferentes filtros de partículas paralelos que confirman el análisis previo.

En el Capítulo 4 se introdujeron tres arquitecturas para el filtrado de partículas en tiempo real. Estos diseños se basan en la estrategia de remuestreo distribuido y consideran un esquema de comunicación determínistico en el que cada elemento de procesamiento le transfiere un número fijo de partículas al elemento de procesamiento adyacente. El hardware dedicado al intercambio de partículas no se describió para ninguno de los tres diseños porque el trabajo se enfocó en el procesamiento principal del filtro. El desarrollo de este hardware se considera como trabajo a futuro.

Las arquitecturas digitales pueden ir desde las arquitecturas de propósito específico con recursos computacionales optimizados hasta las arquitecturas de propósito general que permiten el procesamiento de algoritmos diversos. Entre estos paradigmas, se encuentran los procesadores de aplicación de dominio específico y los procesadores de aplicación con conjunto de instrucciones específico. En esta tesis, se desarrollaron tres esquemas. Los dos primeros diseños, que pueden clasificarse como arquitecturas de propósito específico, están optimizadas para el propósito para el cual fueron diseñadas, es decir para el filtrado de partículas. El tercer diseño se acerca más a una arquitectura de aplicación con conjunto de instrucciones específico.

El primer elemento de procesamiento descripto se basa en el modelo computacional Dataflow con modo de ejecución tipo «data-driven». Posee un pipeline a nivel de módulos con control distribuido en cada etapa. El rendimiento alcanzado es un dato procesado cada dos ciclos de reloj. Este elemento luego se integra a una arquitectura de múltiples elementos.

El segundo diseño propuesto es similar al primero en cuanto a que se basa en el mismo modelo computacional. Su principal diferencia reside en el multiplexado en el tiempo de una parte del procesamiento. La arquitectura se compone de grupos de procesamiento y de recursos compartidos. Un grupo de procesamiento está compuesto por una memoria local, un arreglo de EPs, generadores de números pseudoaleatorios y un módulo de remuestreo. Esta disposición de los recursos permite dos niveles de paralelismo. El primer nivel consiste en introducir elementos de procesamiento que procesen los pasos del algoritmo que no presentan dependencia entre datos. Cuanto más EPs se instancien más partículas pueden ser procesadas en paralelo. El segundo nivel consiste en el remuestreo que se realiza por grupo de acuerdo a la estrategia de remuestreo distribuido. El multiplexado en tiempo de parte de su procesamiento permite reducir el área requerida por un EP y así poder instanciar más EPs en una área de silicio dada.

El tercer diseño propuesto es una arquitectura basada en el modelo computacional SIMD, por lo que se acerca más a una arquitectura del tipo de propósito general aunque sigue disponiendo de módulos de procesamiento dedicado. Además, el conjunto de instrucciones, si bien permite el procesamiento de otros algoritmos, está optimizado para el procesamiento del filtro de partículas. La arquitectura cuenta con un arreglo de procesadores de baja complejidad con la particularidad que todos ejecutan la misma instrucción al mismo tiempo. Cada procesador está asociado a solo una partícula, por lo que no poseen una memoria de múltiples datos como es el caso de los diseños anteriores. El diseño también cuenta con el multiplexado en tiempo de una parte del procesamiento y realiza el remuestreo por fila del arreglo. Del análisis de área y tiempo de ejecución se concluye que no provee una mejora respecto a estos parámetros. Sin embargo, a partir del elemento de procesamiento se pueden desarrollar nuevos diseños con distintas prestaciones. Por ejemplo, un diseño a desarrollar es un procesador que tenga un pipeline y cuente con un módulo de remuestreo. Este diseño probablemente tenga un área menor que el Diseño 1 y una tasa de procesamiento similar.

Se realizó una comparación de las tres arquitecturas propuestas y se concluye que dependiendo de las restricciones que se tengan, ya sea en el tiempo de ejecución o en el área convendrá una arquitectura basada en el EP del Diseño 1 o en grupos de procesamiento y recursos compartidos del Diseño 2. Se observa que existe un límite de no más de dos elementos de procesamiento por grupo a instanciar de manera de obtener una mejora en el área. Por otra parte, como se mencionó en el párrafo anterior, la arquitectura correspondiente al tercer diseño no presenta ventaja en cuanto al área y el tiempo de ejecución.

Para la implementación se empleó una metodología de trabajo que comprende el desarollo de modelos matemáticos en Matlab, en punto flotante y punto fijo, y el posterior desarrollo del modelo RTL de las arquitecturas propuestas. Esto permite realizar una verificación del funcionamiento de la implementación, ya que los resultados de las simulaciones se utilizan para comparar la aproximación de la función de densidad de probabilidad que cada modelo provee. En el caso del primer diseño, éste se encuentra en proceso de fabricación en la tecnología CMOS $0.13\mu$m.

En el futuro se realizará la implementación física de las arquitecturas propuestas en una tecnología VLSI. Luego, se podrán realizar comparaciones de los tres diseños en cuanto a área, consumo y rendimiento sobre la base de resultados experimentales sobre silicio. Esa comparación permitirá una evaluación más detallada de las ventajas y desventajas de cada uno de los diseños propuestos.

# Bibliografía


[1] Isard, Michael y Andrew Blake: *CONDENSATION - conditional density propagation for visual tracking*. International Journal of Computer Vision, 29(1):5–28, 1998.

[2] Fox, Dieter: *KLD-Sampling: Adaptive Particle Filters and Mobile Robot Localization*. En *Advances in Neural Information Processing Systems 14*, volumen 2, páginas 713–720, 2001.

[3] C Kwok, D Fox y M Meila: *Real-time Particle Filters*. Proceedings of the IEEE, 92(3):469–484, Mar 2004.

[4] Fallon, M.F. y S.J. Godsill: *Acoustic Source Localization and Tracking of a Time-Varying Number of Speakers*. Audio, Speech, and Language Processing, IEEE Transactions on, 20(4):1409–1415, May 2012, ISSN 1558-7916.

[5] Montemerlo, D., S. Thrun y W. Whittaker: *Conditional particle filters for simultaneous mobile robot localization and people-tracking*. En *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volumen 1, páginas 695–701 vol.1, 2002.

[6] Gordon, N.J., D.J. Salmond y A. F M Smith: *Novel approach to nonlinear/non-Gaussian Bayesian state estimation*. IEE Proc. Of Radar and Signal Processing, 140(2):107–113, 1993, ISSN 0956-375X.

[7] Kalman, R. E.: *A New Approach to Linear Filtering and Prediction Problems*. 1960. `http://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf`.

[8] Alspach, D.L. y H.W. Sorenson: *Nonlinear Bayesian estimation using Gaussian sum approximations*. Automatic Control, IEEE Transactions on, 17(4):439–448, Aug 1972, ISSN 0018-9286.

[9] Liu, Jun S. y Rong Chen: *Sequential Monte Carlo Methods for Dynamic Systems*. Journal of the American Statistical Association, 93:1032–1044, 1998.

[10] Arulampalam, M. Sanjeev, Simon Maskell y Neil Gordon: *A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking.* IEEE Transactions on Signal Processing, 50:174–188, 2002.

[11] Bolic, Miodrag, Petar M. Djuric y Sangjin Hong: *Resampling Algorithms for Particle Filters: A Computational Complexity Perspective.* EURASIP J. Adv. Sig. Proc., 2004(15):2267–2277, 2004.

[12] Miao, Lifeng, Jun Jason Zhang, Chaitali Chakrabarti y Antonia Papandreou-Suppappola: *Algorithm and Parallel Implementation of Particle Filtering and its Use in Waveform-Agile Sensing.* Signal Processing Systems, 65(2):211–227.

[13] Bolic, Miodrag: *Architectures for Efficient Implementation of Particle Filters.* Tesis de Doctorado, Stony Brook University, 2004.

[14] Bao, Ying, Junfeng Chen, Zhiguo Shi y Kangsheng Chen: *New real-time resampling algorithm for particle filters.* En *Wireless Communications Signal Processing, 2009. WCSP 2009. International Conference on*, páginas 1–5, Nov 2009.

[15] Bolic, Miodrag, Petar M. Djuric y Sangjin Hong: *Resampling Algorithms and Architectures for Distributed Particle Filters.* IEEE Transactions on Signal Processing, 53:2442–2450, 2004.

[16] Chu, Chun Yuan, Chih Hao Chao, Min An Chao y An Yeu Wu: *Multi-prediction particle filter for efficient parallelized implementation.* EURASIP J. Adv. Sig. Proc., 2011:53, 2011.

[17] Hong, Sangjin, Jinseok Lee, A. Athalye, P.M. Djuric y We Duke Cho: *Design Methodology for Domain Specific Parameterizable Particle Filter Realizations.* Circuits and Systems I: Regular Papers, IEEE Transactions on, 54(9):1987–2000, 2007, ISSN 1549-8328.

[18] Hong, Sangjin, Xiaoyao Liang y P.M. Djuric: *Reconfigurable particle filter design using dataflow structure translation.* En *Signal Processing Systems, 2004. SIPS 2004. IEEE Workshop on*, páginas 325–330, 2004.

[19] Fross, D., J. Langer, A. Fross, M. Rossler y U. Heinkel: *Hardware implementation of a Particle Filter for location estimation.* En *Indoor Positioning and Indoor Navigation (IPIN), 2010 International Conference on*, páginas 1–6, 2010.

[20] Par, K. y O. Tosun: *Parallelization of particle filter based localization and map matching algorithms on multicore/manycore architectures.* En *Intelligent Vehicles Symposium (IV), 2011 IEEE*, páginas 820–826, 2011.

[21] Balasingam, Balakumar, Miodrag Bolic, P.M. Djuric y Joaquin Miguez: *Efficient distributed resampling for particle filters.* En *IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, páginas 3772–3775, 2011.

[22] Zhang, Yi Qiao, Thuraiappah Sathyan, Mark Hedley, Philip Heng Wai Leong y Ahmed Pasha: *Hardware efficient parallel particle filter for tracking in wireless networks.* En *PIMRC*, páginas 1734–1739. IEEE, 2012.

[23] Chao, Min An, Chun Yuan Chu, Chih Hao Chao y An Yeu Wu: *Efficient parallelized particle filter design on CUDA.* En *Signal Processing Systems (SIPS), 2010 IEEE Workshop on*, páginas 299–304, 2010.

[24] Bolic, Miodrag, Petar M. Djuric y Sangjin Hong: *Resampling Algorithms and Architectures for Distributed Particle Filters.* IEEE Transactions on Signal Processing, 53(7):2442–2450, July 2005.

[25] Chin, S S y S Hong: *VLSI design of high-throughput processing element for real-time particle filtering.* En *Signals, Circuits and Systems*, volumen 2, páginas 617–620, 2003.

[26] Hong, S, S S Chin, M Bolić y P M Djuric: *Design and Implementation of Flexible Resampling Mechanism for High-Speed Parallel Particle Filters.* Journal of VLSI signal processing systems for signal, image and video technology, 44:47–62, 2006.

[27] Sankaranarayanan, Aswin C., Ankur Srivastava y Rama Chellappa: *Algorithmic and Architectural Optimizations for Computationally Efficient Particle Filtering.* IEEE transcactions on Image Processing, 17(5):737–748, May 2008.

[28] Kloos, Gerold, Jose E. Guivant, Eduardo M. Nebot y Favio Masson: *Range Based Localisation Using RF and the Application to Mining Safety.* En *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, páginas 1304–1311, Oct 2006.

[29] Sanudo, S. y F. R. Masson: *Desempeño del Filtro de Partículas Acotado en una aplicación de localización y seguimiento de camiones en una explotación minera.* En *XIV Reunion de Trabajo en Procesamiento de la Informacion y Control*, volumen 1, páginas 712–717, 2011.

[30] Xia, H., Henry L. Bertoni, L.R. Maciel, A. Lindsay-Stewart y R. Rowe: *Radio propagation characteristics for line-of-sight microcellular and personal communications.* IEEE Transactions on Antennas and Propagation, 41(10):1439–1447, Oct 1993, ISSN 0018-926X.

[31] Veen, Arthur H.: *Dataflow Machine Architecture.* ACM Comput. Surv., 18(4):365–396, Diciembre 1986, ISSN 0360-0300. `http://doi.acm.org/10.1145/27633.28055`.

[32] Barzilai, Z., Don Coppersmith y Arnold L. Rosenberg: *Exhaustive Generation of Bit Patterns with Applications to VLSI Self-Testing*. IEEE Transactions on Computers, C-32(2):190–194, Feb 1983, ISSN 0018-9340.

[33] Lischitz, O., P. Julian, J. Rodriguez y O. Agamennoni: *Accuracy Analysis for an On-Chip Digital PWL Realization*. En *XIV Reunion de Trabajo en Procesamiento de la Informacion y Control*, páginas 429–434, 2011.

[34] Weste, Neil y David Harris: *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publishing Company, USA, 4th edición, 2010, ISBN 0321547748, 9780321547743.

[35] Patterson, David A. y John L. Hennessy: *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edición, 2007, ISBN 0123706068, 9780123706065.