Universidad Nacional del Sur

Tesis Doctoral en Ingeniería

Arquitecturas Eficientes en Energía para Procesamiento No Lineal en Circuitos Integrados

Ing. Mg. Alejandro Pasciaroni

Bahía Blanca Argentina

Noviembre, 2018

Agradecimientos

Quiero agradecer a mi familia que siempre me apoyó y ayudó en todas mis emprendimientos. Siempre intentaron brindarme todo lo que pudieron.

Además quiero brindar mis sinceros agradecimientos al Dr. Pedro Julián y al Dr. Andreas Andreou que me brindaron la posibilidad de realizar mi doctorado, me han guiado en mis investigaciones e implementaciones y me han brindado la oportunidad de trabajar en tecnologías modernas. Por último también quiero agradecerle al Dr. Andreou por su calurosa hospitalidad durante mi estadía en el laboratorio del Dr. Andreou en la Universidad de Johns Hopkins.

Prefacio

Esta tesis se presenta como parte de los requisitos para acceder al grado académico de Doctor en Ingeniería de 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 en investigaciones llevadas a cabo en el Departamento de Ingeniería Eléctrica y Computadoras (DIEC) de la Universidad Nacional del Sur, durante el período comprendido entre el 08 de Marzo de 2016 al 03 de Diciembre de 2018, bajo la dirección del Dr. Pedro Julián y co-dirección de Dr. Andreas Andreou.

Mg. Ing. Alejandro Pasciaroni

UNIVERSIDAD NACIONAL DEL SUR

Secretaría General de Posgrado y Educación

Continua

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

Resumen

En esta tesis se presenta el análisis de paralelismo en sus diferentes niveles para una Sistema en Chip que consta de múltiples procesadores y una memoria de almacenamiento de datos de alta densidad. El objetivo es utilizar el paralelismo como una estrategia para reducir el consumo de energía de las arquitecturas de cómputo VLSI. En particular, se describe la aplicación de técnicas de paralelismo en una arquitectura de reconocimiento automático de voz y su integración en el sistema mencionado implementado en una tecnología CMOS de 55nm. Se describe la aplicación del paralelismo a nivel micro-arquitectura y a nivel de Sistema y se analiza el punto óptimo de paralelismo para obtener una arquitectura de cómputo eficiente desde el punto de vista de tiempo de procesamiento y consumo de energía.

Abstract

In this thesis an analysis of data parallelism implemented in a System on Chip that integrates multiple processing cores and a high density memory is presented. The aim of this work is to optimally utilize different levels of spatial parallelism as a strategy to reduce energy consumption of the whole architecture. The core chosen for this work is an automatic speech recognition architecture integrated in the mentioned System and implemented in a technology CMOS node of 55 nm. Parallelism is included at the micro-architecture level and also at the multiple core chip level. An analysis of the optimal point of the applied parallelism that provides an architecture that minimizes both the energy consumption and the processing time simultaneously is presented.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La voz es el método de comunicación más utilizado entre los humanos. A pesar de todos los avances que se han realizado en términos de algoritmos y capacidad de cómputo, la comunicación de voz con máquinas sigue siendo un desafío. De modo tal de conseguir una alta precisión en el reconocimiento de voz, los algoritmos son computacionalmente intensivos. Esto resulta en implementaciones costosas en términos de área y consumo.

Para implementaciones embebidas tales como el asistente inteligente de los sistemas operativos de los móviles [2], transcripción de reportes médicos [3], entretenimiento del hogar manejado por voz [4], los requerimientos previamente mencionados resultan prohibitivos y para implementaciones a gran escala como el minado de audio [5] estos factores impactan negativamente en el costo de operación. Además dichas aplicaciones requieren respuesta en tiempo real y bajo consumo. En la actualidad, el reconocomiento de voz se realiza en un server remoto para proveer respuestas en tiempo real en vez de realizarse en el dispositivo móvil. Ejemplo de esto es Google Voice Search y

Siri que realizan búsquedas en la red a partir de una pregunta que el usuario transmite con el habla. Esto limita el campo de aplicación del reconocimiento de voz ya que las interfaces basadas en audio podrían diversificarse a robots, aparatos móviles para uso cotidiano sin necesidad de contar con una conexión a la nube. Sin embargo, debido al creciente poder de cómputo es posible decodificar modelos estadísticos mayores. Esto implica trabajar con un tamaño de vocabulario mayor lo que conlleva a un sistema más robusto. Esto se lleva a cabo de diversas maneras tales como: utilización de umbrales de corte de hipótesis más bajos, listas de N-mejores hipótesis mayores.

Si bien aumentó el poder de cómputo, el rendimiento de las computadoras está limitado por el consumo de energía a diferencia de años anteriores donde el límite estaba impuesto por el tamaño del transistor [6]. En aplicaciones para dispositivos móviles el requerimiento de bajo consumo resulta tan importante como la tasa de procesamiento y el área de silicio. El mismo obstáculo se presenta también para las máquinas de escritorio debido a los inconvenientes que existen en la disipación de potencia. La densidad de transistores en una pastilla de silicio se duplica cada dos años según la ley de Moore y por lo tanto también la frecuencia de operación [6]. Estos dos hechos impactan en la disipación y consumo de energía de los circuitos integrados. Es por esto que resulta imprescindible el diseño de arquitecturas eficientes en el consumo de energía y que a su vez, puedan mantener una tasa alta de procesamiento de datos.

La realización de arquitecturas paralelas en sus diferentes niveles de cómputo, siendo el último un arreglo masivo de elementos de procesamiento, permite reducir la velocidad de operación de los diseños pero a la vez manteniendo la

tasa de procesamiento requerida. Esta reducción en la velocidad de operación impacta drásticamente en el consumo de energía de los circuitos integrados [6]. Además, el avance de las tecnologías de fabricación permite instanciar un número cada vez mayor de elementos de procesamiento aumentando así el paralelismo.

Esta tesis estudia la aplicación de técnicas de paralelismo para lograr diseños de bajo consumo con alta capacidad de cómputo en una arquitectura de reconocimiento automático de voz embebida en un Sistema en Chip (SoC) fabricado en una tecnología CMOS de 55nm. El sistema implementado consta de múltiples cores que se comunican entre ellos y con unidades de memoria y otros bloques de procesamiento a través de una red en chip (NoC). El sistema fue diseñado en el marco de un proyecto conjunto entre la Universidad de Johns Hopkins y la Universidad Nacional del Sur.

En particular, se presenta la integración de una arquitectura digital del tipo Dataflow [7] con múltiples procesadores. Se muestra que el aumento del paralelismo permite reducir la velocidad de operación. A su vez, esto se logra con la disminución de la tensión de operación que produce una reducción cuadrática en el consumo. La arquitectura cuenta con paralelismo temporal a través de la utilización de pipelines de datos y solapamiento de operaciones. El sistema integrado cuenta con los siguientes subsistemas:

- una unidad aritmética para computar funciones Gaussianas multivariables.
- unidades de ordenamiento que cuentan con almacenamiento propio.
- sistema de evaluación de distribuciones Gaussianas en formato tipo

árbol.

- memorias caché para reducir la latencia de memoria e incrementar la tasa de procesamiento.

La arquitectura de cómputo procesa las muestras de audio correspondientes a una ventana temporal y devuelve como resultado un listado de sonidos más probables. Para ello se computa el algoritmo de Viterbi [8] utilizando estructuras de datos llamadas fonemas. Los fonemas representan sonidos típicos correspondientes a la fonética de un idioma. La arquitectura realiza en primer instancia el procesamiento de los vectores de características cepstrales [9]. Mediante dicho procesamiento se obtienen los vectores que caracterizan las ventanas temporales de muestras de audio. Luego utilizando dichos vectores se computan los modelos de las distribuciones de probabilidad de cada fonema. Por último, se computa el algoritmo de Viterbi utilizando los modelos de probabilidad previamente computados. Luego de un cierto número de ventanas temporales la arquitectura devuelve el listado de fonemas más probables.

La arquitectura se integra en una red en chip, NoC (del inglés Network on Chip). Las redes en chip permiten explorar arquitecturas de cómputo distribuido con múltiples elementos de procesamiento simples que exploten distintos niveles de paralelismo y que puedan conformar una red de procesadores. Las ventajas que presenta la arquitectura NoC son numerosas: alta escalabilidad, versatilidad y alto rendimiento con buena eficiencia en el consumo. En lo que se refiere a integración, NoC es ampliamente escalable, porque el número de nodos se puede incrementar para proveer un mayor rendimiento

sin que ello degrade la performance de la red. La arquitectura NoC junto con los protocolos de comunicación basados en intercambio de paquetes, proveen posibilidades de comunicación dinámicas y versátiles.

A nivel de integración, esta tesis presenta también la solución de múltiples desafios como la utilización de la memoria de manera eficiente, la entrada y salida de datos, su configuración y la comunicación con otras unidades. La arquitectura diseñada es escalable, de manera que las unidades pueden ser instanciadas varias veces dentro de la NoC, ampliando la capacidad de cómputo del sistema total.

1.1. Organización

La tesis está organizada de la siguiente manera. El Cap. 2 presenta el método de reconocimiento de voz basado en el algoritmo de Viterbi, un método frecuentemente utilizado en la literatura y resultados preliminares de implementaciones digitales para la aplicación en cuestión. En el Cap. 3 se presenta la arquitectura del sistema de múltiples procesadores en la cual se integran las unidades de procesamiento. Se describe la arquitectura del bloque de procesamiento para el reconocimiento de voz automática que fue integrada en el sistema, su implementación y rendimiento. En el Cap 4. se presenta un análisis de paralelismo para uno de los bloques de procesamiento involucrado en el reconocimiento de voz automática en dos niveles diferentes: a nivel de micro-arquitectura y a nivel de CMP. En el Cap. 5 se presentan diseños realizados para procesamiento de propósito general que fueron optimizados para obtener tasas de rendimiento elevadas y que fueron integrados

en el sistema de múltiples procesadores. En el Cap 6. se presentan las conclusiones del trabajo.

Capítulo 2

Reconocimiento de voz automático.

2.1. Teoría

El método de reconocimiento de voz basado en el algoritmo de Viterbi utilizando cadenas de Markov ocultas es apliamente utilizado en la literatura [10] [11] [12]. Dicho método se detalla a continuación. Para un conjunto de palabras $W = w_1, w_2, w_3, ..., w_j$ y un conjunto de símbolos acústicos observados $A = a_1, a_2, a_3, ..., a_k$, la probabilidad de que el conjunto W fue hablado dado el conjunto de observaciones A se denota $P(W|A)$. El conjunto de palabras que maximiza la probabilidad mencionada se describe como:

$$\hat{W} = \arg\max_{W} P(W|A) \tag{2.1}$$

De acuerdo a la regla de Bayes, esto es equivalente a :

$$\hat{W} = \arg\max_{W} \frac{P(A|W)P(W)}{P(A)} \tag{2.2}$$

Dado que $P(A)$ es una constante de normalización, se puede simplificar la ecuación quedando:

$$\hat{W} = \arg\max_{W} P(A|W)P(W) \tag{2.3}$$

De modo que para realizar el reconocimiento de voz automático es necesario computar la Ecuación 2.3. Ésta consta de un modelo de lenguaje $P(W)$, y un modelo acústico $P(A|W)$. Este último es la probabilidad de las observaciones A dado el conjunto de palabras W. En este trabajo se computa el mismo utilizando cadenas de Markov ocultas. El término $P(W)$ es la probabilidad del conjunto de palabras. Un ejemplo de este modelo es el modelo n-gram que predice la próxima palabra en función de una secuencia de palabras anteriores. En este trabajo no se implementó el modelo de lenguaje aunque puede ser facilmente integrado a la arquitectura actual.

2.1.1. Cadenas de Markov ocultas

Las cadenas de Markov ocultas es un método para obtener la secuencia de eventos más probable dada una secuencia de eventos observados. Una cadena de Markov típica se muestra en la Fig. 2.1. Comenzando en el estado s_1, hay dos posibles transiciones, hacia sí mismo o hacia el estado s_2. Estas transiciones ocurren con cierta probabilidad expresada como $p(s_i|s_j)$. La sumatoria de las probabilidades de las transiciones posibles de un estado deben sumar 1. Además cada estado tiene una probabilidad de observar el evento

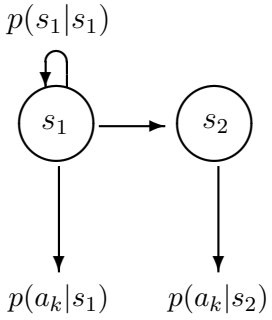


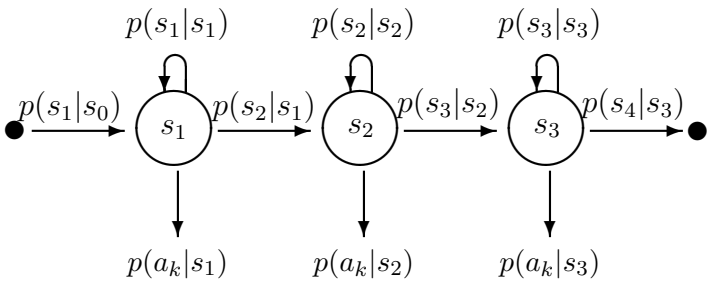
Figura 2.1: Cadena de markov oculta básica.

a_k expresada como $p(a_k|s_i)$.

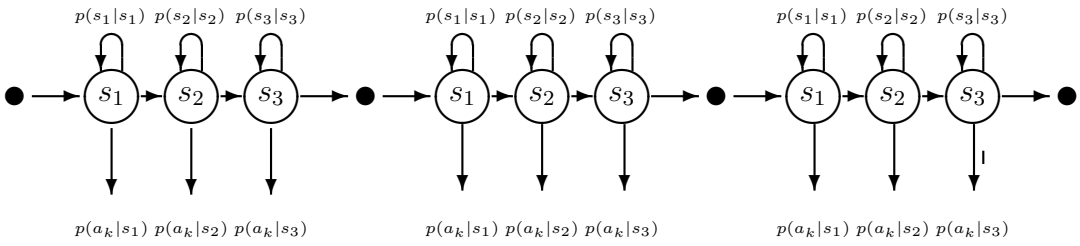
El algoritmo de reconocimiento de voz se basa en unidades fonéticas llamadas fonemas las cuales se modelan con una cadena de Markov de tres estados. Las palabras se componen de una sucesión de estas unidades. En la Fig. 2.2 se muestra la estructura de un fonema y la sucesión de fonemas “p”, “e”, “s”que componen la palabra pez.

El fonema se compone de tres estados más los estados nulos que se utilizan para enlazar fonemas [13]. Cada estado s_i del fonema tiene dos posibles transiciones: hacia si mismo o hacia el próximo estado a la derecha. Tal como se mencionó previamente el modelo está descripto por las probabilidades de transición y observación: $p(s_j|s_i)$ y $p(a_k|s_i)$. En cada intervalo de tiempo, la probabilidad de transicionar a un estado en particular depende de la probabilidad acumulada del estado previo, la probabilidad de transición y la probadilidad de observación.

Es muy común representar la secuencia de transiciones en el tiempo. Este tipo de diagrama se muestra en la Fig. 2.3 y se denomina diagrama Trellis. El eje horizontal es el tiempo y el eje vertical son los estados. Como se puede observar, múltiples caminos tienen un mismo destino, por lo que el diagrama



(a)



"p" "e" "s"

(b)

Figura 2.2: (a) Estructura del fonema. (b) palabra "pez"modelada por una secuencia de fonemas.

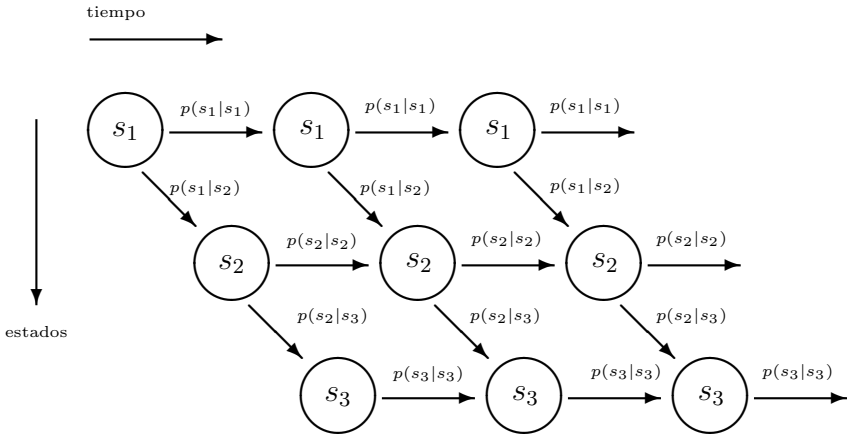


Figura 2.3: Diagrama de Trellis para una secuencia de fonemas.

de Trellis permite visualizar todas las secuencias de estados posibles.

La probabilidad conjunta de una secuencia de estados $S = s_1, s_2, ..., s_N$ y una secuencia de eventos acústicos obervados $A = a_1, a_2, a_3, ..., a_K$, dado un modelo θ viene dada por :

$$P(A|\theta) = \sum_{S} P(A, S|\theta) \tag{2.4}$$

Si usamos la letra l como índice de notación del tiempo entonces la probabilidad del estado s_l de la secuencia mencionada queda expresada, en forma recursiva, por el producto:

$$p(s_l) = \sum_{\forall s_{l-1} \to s_l} p(s_l|s_{l-1}) \cdot p(a_k|s_l) \cdot p(s_{l-1}) \tag{2.5}$$

la sumatoria tiene en cuenta todos los estados s_{l-1} en el intervalo de tiempo previo que transicionan al estado s_l. Dado que la probabilidad es un número menor o igual a 1, la multiplicación sucesiva de las probabilidades puede resultar en números muy cercanos a cero difíciles de discernir, más aún

en aritmética de punto fijo. Para evitar este inconveniente el cómputo (2.5) se realiza en el dominio logarítmico. Dicho esto, la ecuación queda:

$$log(p(s_l)) = log\left(\sum_{\forall s_{l-1}\to s_l} p(s_l|s_{l-1}) \cdot p(a_k|s_l) \cdot p(s_{l-1})\right) \tag{2.6}$$

Entonces debido a la recursividad, la probabilidad $p(s_l)$ depende fuertemente de la secuencia de estados que tiene como destino el estado que se está analizando.

2.1.2. Algoritmo de Viterbi

El algoritmo de Viterbi permite encontrar la secuencia de estados que maximiza la probabilidad de un estado particular. Desde el punto de vista del diagrama de Trellis, el algoritmo encuentra aquella secuencia o camino que maximiza dicha probabilidad. El algoritmo selecciona en cada paso de recursividad el estado cuya probabilidad es máxima y luego computa la ecuación recursiva a partir de dicho estado.

Expresado matemáticamente el algoritmo en cada paso recursivo computa:

$$p(s_l^{max}) = \underset{s_l}{max}\left(p(s_l)\right) \tag{2.7}$$

Esto reduce significativamente el número de caminos a evaluar dado que sólo se tiene en cuenta todos los caminos que tienen como punto de partida el estado de mayor probabilidad de la iteración anterior $p(s_{l-1}^{max})$.

En el dominio logarítmico el algoritmo de Viterbi se expresa como:

$$
\begin{aligned}
log(p(s_l^{max})) &= log\left(\max_{s_{l-1}\to s_l}(p(s_l|s_{l-1})\cdot p(a_k|s_l)\cdot p(s_{l-1}))\right) \\
&= \max_{s_{l-1}\to s_l}(log(p(s_l|s_{l-1})) + log(p(a_k|s_l)) + log(p(s_{l-1})))
\end{aligned}
\tag{2.8}
$$

lo cual simplifica significativamente (2.6).

Es posible que al considerar sólo el estado de mayor probabilidad en cada paso del cómputo recursivo se descarte el camino que eventualmente sea el mejor en cuanto a la inferencia de la palabra hablada. Una estrategia para reducir el riesgo de que esto suceda es evaluar los N mejores estados es decir aquellos que tienen las N mayores probabilidades del paso anterior. Entonces en cada paso se computan múltiples cáminos, se ordenan de mayor a menor en base a su valor de probabilidad y se conservan los N mejores mientras que el resto se descarta.

La Fig. 2.4 muestra el flujo de procesamiento del algoritmo de Viterbi considerando los 64 mejores caminos. Para cada iteración se representa una lista de estados ordenada en función de la probabilidad acumulada. Inicialmente, el módulo tiene 64 estados f_{i_e1}, todos ellos el primer estado de la estructura del fonema como detalla el subíndice *e1* y con una probabilidad inicial uniforme entre todos ellos. Por cada estado se generan dos nuevos de modo tal de cubrir las dos posibles transiciones: hacia al próximo estado o si mismo. Estos se señalan con flechas en la Fig. 2.4. Se actualiza la probabilidad de los estados generados utilizando la probabilidad de observación, la probabilidad de transición y la probabilidad acumulada del estado. Finalmente, estos se

ordenan en función de la probabilidad y se conservan sólo los 64 estados más probables. En la segunda iteración, se repite el mismo procedimiento pero sobre la lista ordenada de estados generada en la iteración anterior, siendo el estado con mayor probabilidad el primero de la parte superior. En este caso es f_{56_e2} que es el estado 2 del fonema 56^{to}. En este ejemplo, para la tercer iteración, en la lista algunos de los fonemas ya se encuentran en el tercer estado, de manera que se evalua la transición hacia si mismo o hacia el estado nulo. El fonema que transiciona al estado nulo, implica que en base a los eventos observados, resulta en un fonema con alta probabilidad y es por lo tanto un resultado del algoritmo. Luego por cada resultado se inserta en la lista un juego nuevo de 64 estado iniciales dado que se debe inferir el siguiente fonema de la estructura de la palabra.

En el ejemplo de la Fig. 2.4 los fonemas 56 y 22 son resultados dado que han transicionado a través de todos los estados. Como se detalló en el párrafo anterior, por cada uno de ellos se insertaron nuevas hipótesis en el flujo de manera de que se puedan inferir nuevos fonemas. En la cuarta iteración se comienza a inferir un nuevo fonema, siendo los estados f_{15_e1} y f_{54_e1} los más probables. En la lista también aparecen fonemas en el segundo y tercer estado dado que aún continuan hipótesis anteriores en el flujo de procesamiento.

Volviendo al ejemplo de la Fig. 2.2 el fonema 56 o 22 puede representar el sonido *"p"*. Y el juego nuevo de estados iniciales tiene como propósito final inferir el sonido *"e"*. Al final de este procesamiento, si el truncamiento de la lista en 64 estados no descartó el camino más probable, se habrán inferido los tres fonemas que componen la palabra *pez*.

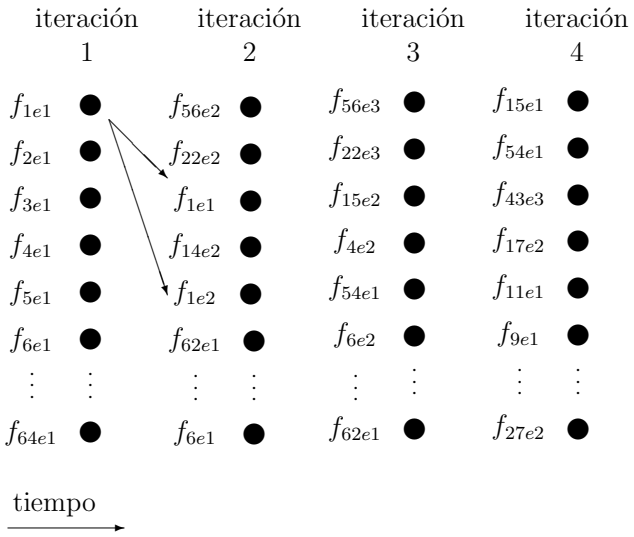


Figura 2.4: Flujo de procesamiento de los fonemas.

2.1.3. Modelo de la probabilidad de observación

El evento acústico del modelo a_k se representa a través de un vector cuyas componentes definen las características del evento. La probabilidad de observación del evento dado el estado s_l, $p(a_k|s_l)$, se puede modelar con una sumatoria de distribuciones Gaussianas. Dado que el evento es un vector de múltiples dimensiones, las distribuciones Gaussianas son multivariables. La Ecuación 2.9 expresa en forma matemática la suma de distribuciones.

$$p(a_k|s_l) = \sum_{i=1}^{N} \alpha_i \cdot \frac{1}{(2\pi)^{P/2}|\sum|^{1/2}} \cdot e^{-\frac{1}{2}(x-\mu)^T \sum^{-1}(x-\mu)} \tag{2.9}$$

donde $x, \mu \in R^3 9$ y $\sum$ es la matriz de covarianza $\in R^{39\times 39}$. Esta última se asume diagonal lo cual simplifica considerablemente el cómputo. Esta simplificación se compensa al considerar un mayor número de distribuciones en la sumatoria. Por otra parte, el procesamiento que realiza la interfaz acústica

reduce la correlación entre las componentes.

Por último α_i es el peso que tiene cada distribución en la sumatoria y está sujeto a la siguiente condición

$$\sum_{i=1}^{N} \alpha_i = 1 \tag{2.10}$$

Es muy común que las distribuciones Gaussianas se computen en el dominio logarítmico dado que provee mayor estabilidad computacional. La sumatoria de distribuciones en este dominio queda expresada como:

$$log\left(p(a_k|s_l)\right) = log\left(\sum_{i=1}^{N} \alpha_i \cdot \frac{1}{(2\pi)^{P/2}|\sum|^{1/2}} \cdot e^{-\frac{1}{2}(x-\mu)^T \sum^{-1}(x-\mu)}\right) \tag{2.11}$$

Esta ecuación puede ser simplificada considerando el máximo elemento de la sumatoria. Aplicando dicha simplificación la ecuación anterior se expresa como:

$$\begin{aligned} log\left(p(a_k|s_l)\right) &\approx log\left(\max_i \left(\alpha_i \cdot \frac{1}{(2\pi)^{P/2}|\sum|^{1/2}} \cdot e^{-\frac{1}{2}(x-\mu)^T \sum^{-1}(x-\mu)}\right)\right) \\ &\approx \max_i \left(log(\alpha_i) + log\left(\frac{1}{(2\pi)^{P/2}|\sum|^{1/2}}\right) - \frac{1}{2}(x-\mu)^T \sum^{-1}(x-\mu)\right) \end{aligned} \tag{2.12}$$

Si se agrupan los dos primeros términos de la Ecuación 2.12 en una sola constante K_i el logaritmo de una suma de distribuciones Gaussianas queda

entonces expresado como:

$$log\left(p(a_k|s_l)\right) \approx \max_i \left(log(K_i) - \frac{1}{2}(x-\mu)^T \sum^{-1} (x-\mu) \right) \tag{2.13}$$

2.2. Resultados Preliminares

El reconocimiento automático de voz ha sido estudiado durante las últimas tres décadas. Sin embargo a pesar de los enormes progresos realizados en algoritmos y en capacidad de cómputo aún continúa siendo un desafío lograr el reconocimiento automático de voz en las computadoras. De manera de poder lograr una gran precisión en el reconocimiento, los algoritmos precisan de grandes modelos y son computacionalmente intensivos. Esto resulta en soluciones costosas en términos de tiempo de procesamiento, área de silicio y también en consumo de energía.

En cuanto al procesamiento de la voz comunmente se realiza en tres etapas de acuerdo a la teoría descripta en la Sección 2.1: extracción de características, evaluación acústica e inferencia. A continuación se explicará cada una de estas etapas.

Una vez que se muestrea digitalmente la voz a través de un micrófono, las muestras se organizan en bloques llamados marcos en donde cada uno de ellos contienen muestras correspondientes a un intervalo de 10 ms. Por cada marco se obtiene un vector de características acústicas. El método más utilizado para la extracción es el que produce los coeficientes Cepstrales en las frecuencias de Mel (MFCC del inglés Mel Frequency Cepstral Coefficients)

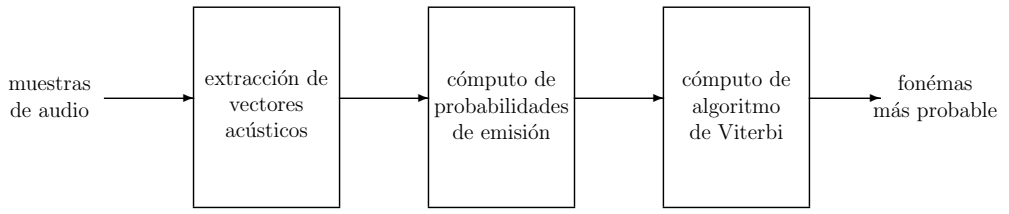


Figura 2.5: Flujo de procesamiento del reconocimiento de voz automático.

[14]. Al aplicar dicho método se obtiene entonces un vector de 39 dimensiones, llamado vector MFCC.

Luego, en la evaluación acústica se evalua la probabilidad de que el vector MFCC sea similar a alguno de los estados que integran los fonemas del idioma elegido. La probabilidad de dicha similitud se modela a través de una sumatoria de distribuciones Gaussianas multi-variables.

Por último utilizando un modelo de palabra y las probabilidades de la etapa anterior se infiere una secuencia de estados que conforma el fonema producido por la voz y en un segundo nivel superior se infiere una secuencia de fonemas que conforma la palabra hablada. En general se utiliza el algoritmo de Viterbi con cadenas ocultas de Markov tal como se detalló previamente. La Fig. 2.5 muestra el flujo de procesamiento más comunmente utilizado.

Desde el punto de vista del diseño de arquitecturas digitales para el reconocimiento de voz los desafíos que se presentan son múltiples: la implementación de operaciones complejas no lineales, operación con estructuras multidimensionales, la elección del nivel de pipeline para cada bloque de hardware y nivel de distribución del cómputo. Dado que ciertos algoritmos permiten un mayor nivel de paralelismo y concurrencia entre tareas, nue-

vas arquitecturas que exploten estos puntos son factibles de ser diseñadas e implementadas.

Existen diferentes técnicas de reducción del consumo de energía en circuitos integrados a nivel algoritmo y a nivel de circuito lógico. A nivel algoritmo se pueden considerar simplificaciones en la aritmética, reducción de las operaciones, incremento del paralelismo, etc. A nivel de circuito lógico se lleva a cabo la implementación de múltiples unidades de cómputo, pipelining, mapeo de algoritmos en arquitecturas SIMD [15], apagado de ciertas partes del circuito, reducción del ancho de palabra, multiplexación en el tiempo de recursos de hardware, etc. Otra técnica utilizada es la reducción del voltaje de operación lo que a su vez limita la frecuencia de operación máxima. Esto afecta de forma negativa a la tasa de procesamiento lo cual puede mitigarse recurriendo a estrategias de paralelismo [16].

En [17] se presenta un diseño para el cual el cuello de botella es el acceso a memoria. El diseño computa las etapas de evaluación acústica e inferencia y utiliza una memoria DRAM para el almacenamiento de los parámetros de las distribuciones Gaussianas y de las cadenas de Markov ocultas del algoritmo de Viterbi. De modo tal de minimizar el efecto de la latencia de la memoria DRAM, los autores hacen uso de una arquitectura con pipeline de modo tal de poder utilizar de una manera eficiente el modo ráfaga de la memoria DRAM, solapando además el acceso a memoria con el tiempo de cómputo.

Con el fin de reducir el ancho de banda de memoria requerido los autores hacen uso en primer lugar de una reducción del ancho de palabra de los parámetros requeridos para el cómputo de las probabilidades de observación. En segundo lugar, paralelizan el cómputo de estas probabilidades para 4

marcos consecutivos y de este modo pueden reutilizar los parámetros leídos de memoria. Por último, se realiza una selección optimizada de la próxima palabra a ser inferida lo cual reduce accesos a memoria innecesarios. Los autores también incorporan memoria local para reducir la cantidad de accesos a la memoria DRAM. La memoria local almacena datos que son accedidos frecuentemente: las distribuciones Gaussianas evaluadas, la lista de estados de fonema activos, etc. Sin embargo la utilización de los recursos de la FPGA es alta y el ancho de banda sigue siendo un factor limitante del diseño.

En [18] se presenta una arquitectura digital implementada en una tecnología CMOS de 40 nm. La arquitectura se basa en el algoritmo descripto previamente y está compuesta por tres unidades: una unidad GMM (del inglés Gaussian Mixture Model), una unidad Viterbi y una unidad de ordenamiento. La extracción del vector de características acústicas MFCC se realiza fuera del chip. Además el sistema utiliza la memoria DRAM de una FPGA externa al chip para almacenar los parámetros de las Gaussianas y de los fonemas.

La unidad GMM realiza el computo de la probabilidad de observación que es una suma de Gaussianas mixtas. De modo tal de maximizar la tasa de procesamiento de esta unidad, los autores recurren a un paralelismo espacial. Se computa la misma distribución Gaussiana para un conjunto de 20 vectores MFCC consecutivos en el tiempo. De este modo se reutilizan los parámetros de una distribución para 20 vectores distintos y se reduce el ancho de banda de la memoria externa por un factor de 1/20. Además la unidad Viterbi es capaz de evaluar 8 transiciones distintas en paralelo.

El principal cuello de botella del diseño es la latencia de la memoria

DRAM. El chip cuenta con una jerarquía de memoria para reducir el efecto negativo de la latencia en la tasa de procesamiento. Los autores diseñaron una memoria caché en el chip y también incorporaron un segundo nivel de memoria caché en la FPGA externa. De esta manera, los datos que no se encuentren almacenados en el primer nivel se buscan en el segundo nivel. En caso de que en este último nivel tampoco esté almacenado el dato finalmente se busca a memoria DRAM.

En [19] se presenta un diseño que computa la etapas de inferencia mientras que la extracción de las características y la evaluación acústica se computa por fuera del diseño. Los autores realizan un análisis del tiempo de procesamiento de cada etapa. La conclusión de dicho análisis revela que la etapa que más tiempo consume es la inferencia dado la cantidad de estados que se actualizan, número que depende del tamaño del vocabulario. Aquí es preciso tener en cuenta que cuando un fonema es completado en el algoritmo de Viterbi, se pueden activar nuevos fonemas a ser inferidos de acuerdo a un diccionario. Esto introduce variabilidad en la cantidad de estados a ser actualizados de marco a marco. Dicha variabilidad implica que el patrón de acceso a memoria es también variable e irregular. Desde el punto de vista de hardware, la última etapa de procesamiento puede beneficiarse de una jerarquía de memoria adecuada y un flujo de procesamiento optimizado. Debido a esto los autores se focalizan en optimizar dicha etapa.

Los autores utilizan una memoria DRAM para el almacenamiento de los parámetros necesarios tanto para el computo de las distribuciones Gaussianas y el algoritmo de Viterbi. Sin embargo, dado que el ancho de banda de memoria tiene un gran impacto en la tasa de procesamiento, los autores incor-

poraron memorias caché y buffers de modo tal de acelerar el diseño. Además se implementó un pipeline en la arquitectura y también se replicaron algunas operaciones de modo tal de incrementar aún más la tasa de procesamiento.

La principal desventaja es que no resulta escalable, ya que sólo para el diseño presentado fue necesaria la utilización de dos FPGA. De ser necesario un diccionario o un modelo de lenguaje de mayor tamaño se necesitará un número mayor de FPGAs.

En [20] los autores se focalizan en un sistema para tamaños de vocabulario medio y diseños de bajo consumo para el cómputo de las distribuciones Gaussianas ya que consideran que dicha etapa de procesamiento es la más intensiva computacionalmente. El resto del procesamiento se realiza por fuera del chip en un microcontrolador por lo que el diseño resulta en un coprocesador. Los autores presentan una implementación apropiada para VLSI de múltiples procesadores trabajando en paralelo. Se presentan resultados de la síntesis y ruteado del diseño en tecnología CMOS 130 nm.

El arreglo de procesadores permite computar distribuciones para 8 estados de fonemas en paralelo para un vector de características acústicas de 27 elementos. Además cuenta con memoria local para el almacenamiento de 79 vectores acústicos de modo que los parámetros almacenados de las distribuciones pueden ser re utilizados. Una vez que los procesadores terminaron la evaluación almacenan los resultados en otra memoria local para que luego puedan ser leídos por el microcontrolador. Los procesadores trabajan en modo pipeline lo que permite solapar la evaluación de las distribuciones de dos marcos distintos.

El diseño presenta un consumo de potencia bajo debido a que opera a una

frecuencia relativamente baja y cuenta con la aritmética optimizada para la tarea. Sin embargo el diseño no resulta apropiado para vocabularios mayores a 1000 palabras desde el punto de vista del consumo de energía y tiempo de procesamiento. El diseño tampoco resulta escalable si se quiere trabajar con un número mayor de Gaussianas por mixtura dado que en ese caso sería necesario un número mayor de procesadores y memoria local para mantener el mismo tiempo de procesamiento.

En [10] se presenta un diseño que realiza las tres etapas de procesamiento. La inferencia se realiza utilizando cadenas de Markov ocultas pero a nivel palabra en vez de a nivel fonema. Esto resulta en una limitante en el tamaño del vocabulario a inferir, siendo sólo apropiado para vocabularios no mayores a 1000 palabras. El diseño paraleliza el cómputo de las distribuciones Gaussianas y el cómputo del algoritmo de Viterbi para varios marcos de datos. Los autores implementan un elemento de procesamiento que realiza el cómputo de las tres etapas. Es posible realizar un arreglo de los mismos en donde uno de ellos se establece como maestro y el resto son esclavos. El maestro se encarga de realizar la extracción de las características acústicas y luego las transmite a los esclavos que junto con el maestro trabajan en paralelo para computar la evaluación acústica e inferencia.

El diseño utiliza memoria SRAM para el almacenamiento de los parámetros necesarios para el cómputo de las distribuciones Gaussianas y el algoritmo de Viterbi y también de los resultados de la etapa de inferencia. Por lo tanto, en el caso de que se quiera ampliar la capacidad del sistema, el ancho de banda no es una limitante pero si el área de silicio. Además el sistema computa probabilidades de observación como distribuciones Gaussianas sim-

ples y para poder computar mixturas de distribuciones se deben llevar a cabo modificaciones en la arquitectura.

En [21] se presenta una arquitectura VLSI que utiliza el modelo de transductores de estados finitos pesados que integra en una sola máquina de estados el modelo de fonemas y del lenguaje. En cuanto al procesamiento es similar al del modelo con cadenas de Markov ocultas descripto anteriormente. De hecho la búsqueda de la palabra más probable de haber sido hablada se realiza utilizando el algoritmo de Viterbi. La arquitectura que se presenta realiza las tres etapas de procesamiento. La misma utiliza una memoria externa para el almacenamiento de los parámetros de los modelos por lo que el principal cuello de botella del diseño es el acceso a memoria y el ancho de banda disponible.

Los autores recurren a ciertas técnicas para reducir la cantidad de accessos a memoria y el ancho de banda de memoria requerido como son: optimización de los datos almacenados en la memoria caché, reducción del ancho de palabra de los parámetros de las distribuciones Gaussianas e inserción de una memoria caché para el almacenamiento de las distribuciones evaluadas. Dado que las mismas distribuciones son a veces requeridas múltiples veces en un marco de datos, para evitar la evaluación repetida se almacenan en la memoria lo que reduce el pedido de parámetros a memoria externa.

No sólo el paralelismo resulta importante para reducir el consumo de energía sino también la implementación de ciertas operaciones en circuitos analógicos que luego son instanciados en la arquitectura global obteniendo así un circuito de señales mixtas analógico-digitales. En [22] se presenta un diseño analógico para la extracción de características acústicas. El diseño

comprende un flujo de procesamiento de 4 etapas : un filtrado de pre enfásis que básicamente es un filtro pasa-altos; un banco de filtros pasa-banda de segundo orden, para el cual las frecuencias centrales de los filtros están distribuidas de acuerdo a la escala de Mel; una etapa rectificadora que extrae la amplitud de la señal para cada banda de frecuencia; y por último un banco de filtros pasabanda que puede o no tener una compresión logarítmica previa. Los autores citan como ventajas un consumo de energía bajo y más robustez frente al ruido de la señal de audio en comparación con una implementación digital. El diseño fue implementado en tecnología CMOS 0.5 μm y testeado experimentalmente.

Por último, la arquitectura planteada en [1] utiliza una tecnología 3D. El diseño cuenta con dos niveles de silicio particionados de la siguiente manera: el nivel superior tiene una memoria DRAM y su controlador que permite transferencias a tasa doble y la interfaz acústica que realiza la extracción de las características acústicas. El nivel inferior tiene dos procesadores dedicados al cómputo de las distribuciones Gaussianas mixtas y a la búsqueda de los fonemas más probables utilizando el algoritmo de Viterbi. Estos procesadores se denominan GMM y HMM (del inglés Gaussian Mixture Model y Hidden Markov Model). Además este nivel cuenta con una memoria caché en la cual se almacenan las evaluaciones de las mixturas de distribuciones Gaussianas. La integración se realizó a nivel bloque, es decir que los bloques de procesamiento completos fueron ubicados en diferentes niveles de obleas. La conexión de estos niveles entre si se realiza a través de vías 3D, que conectan obleas distintas y permiten lograr anchos de buses superiores a 128 bits [23].

Según el autor los procesadores GMM y HMM pueden operar hasta una frecuencia de 125 MHz. Permiten el procesamiento de 2048 distribuciones de Gaussianas y 64 fonemas con una aceleración respecto al tiempo real de 300 para el caso del procesador GMM y del orden de mil para el procesador HMM. El rendimiento mencionado está basado en simulaciones realizadas por el autor. Esta arquitectura permite obtener estos números de aceleración gracias a la memoria principal la cual opera a una frecuencia de 1GHz y se encuentra completamente dedicada al reconocimiento de voz. La interfaz de memoria opera a 250 MHz, permite operaciones a tasa doble de procesamiento de datos y lecturas y escrituras en modo ráfaga. En cuanto a la memoria local RAM, destinada a colas, registros internos y una memoria caché que actua como buffer entre los procesadores GMM y HMM, utiliza 66 KBytes en total. Para una tecnología CMOS de 130 $[\mu m]$ el autor reporta un área requerida de 25 $[mm^2]$.

Capítulo 3

Sistema de Multi-Procesadores.

3.1. Introducción

En este capítulo se describe el sistema en chip y sus bloques principales, la implementación del módulo de reconocimiento de voz automático y su integración en dicho sistema. El sistema en chip se enmarca dentro del proyecto UPSIDE del cual participan la Universidad de Johns Hopkins, la Universidad de Santa Barbara y la Universidad Nacional del Sur. Este proyecto plantea la adquisición y procesamiento de imágenes de alta resolución y el procesamiento de audio con alta eficiencia energética [24] a través de un sistema complejo de bajo consumo. Las imágenes son tomadas desde un avión con un arreglo de cámaras y pueden tener hasta 400 Mpixels. Las tareas de procesamiento visual requeridas comprenden filtrado de la imágen e interpolación de Bayer [25], correción no uniforme, operación de "*dewarping*"[26], clasificación de fondo y objetos en primer plano, operaciones morfológicas sobre los objetos segmentados [27] y por último el seguimiento de dichos ob-

jetos [28]. En cuanto al procesamiento de audio, se implementó un sistema de reconocimiento automático de voz, compuesto por una interfaz acústica, una unidad de cómputo de distribuciones Gaussianas y una unidad para clasificación utilizando cadenas de Markov ocultas. El sistema también incluye estructuras auxiliares de cómputo como multiplicaciones matriz-vector utilizando técnicas de computación estocástica, circuitos de señales mixtas con almacenamiento local de parámetros [29] y procesadores morfológicos[30].

3.1.1. Aquitectura del Sistema en Chip

El proyecto se implementó en cuatro sistemas en chip con diferentes capacidades de procesamiento interconectados a través de un interposer de silicio ([31] [32]). El interposer es un substrato de silicio con varios niveles de metal sobre el cual se ubican e interconectan 3 sistemas en chip, una memoria DRAM DDR 3D y una FPGA Xilinx Zynq-7. Los sistemas leen y escriben datos desde la memoria que posee una capacidad de almacenamiento de 281 Tbits. La memoria cuenta con 4 puertos independientes que abastecen a los 3 sistemas en chip y la FPGA simultáneamente. Tanto los sistemas en chip como la FPGA cuentan con conexiones $\mu Bumps$ para conectarse con el interposer. Este a su vez cuenta con 1130 pads (para realizar wire bonding) para el conexionado con un PCB. La FPGA se añadió para brindar más flexibilidad, capacidad de testeo y búsqueda de fallas; la misma se conecta a cada sistema en chip a través de un puerto dedicado. El área total del interposer es de 6.4 x 5.2 cm.

Cada uno de los sistemas en chip se implementó en la tecnología CMOS

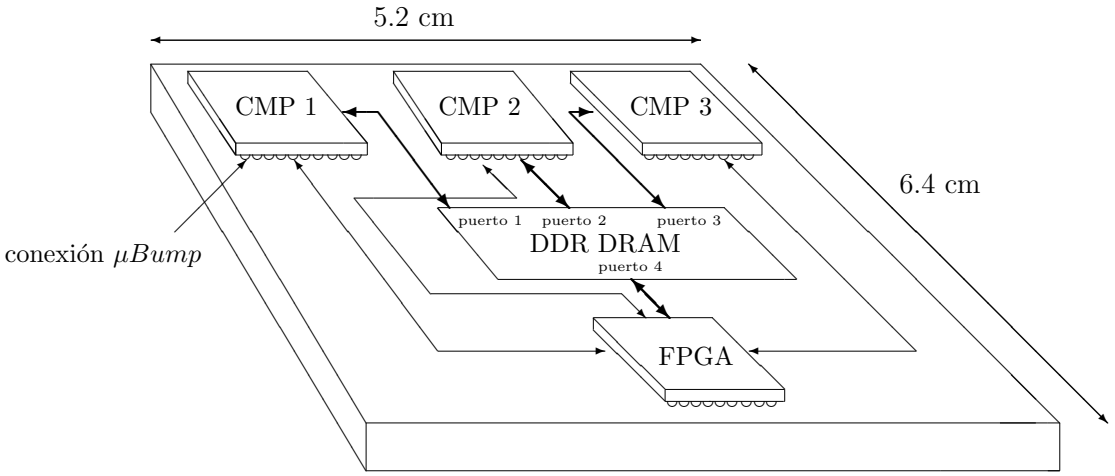


Figura 3.1: Diagrama en bloque del interposer.

55 nm mientras que el interposer se realizó en un proceso de 1 μm. Para la implementación del interposer se tuvieron en cuenta todas las señales de datos entre la memoria 3D, los sistemas en chip y la FPGA. Además se realizó el ruteo de todas las señales de alimentación y tensiones de referencia que son utilizadas por los sistemas (no se muestran los pads para lograr mayor claridad en el diagrama).

La red N1(ver Fig. 3.2) esta destinada para la comunicación entre los procesadores y la memoria principal DDR, con una frecuencia de operación de 300 MHz. La red es del tipo *token ring* y consiste en ocho anillos que operan en forma independiente en cada fila. Posee canales de comunicación separados para los comandos y para los datos. La red de comunicación N2 está destinada para la comunicación entre unidades de procesamiento y también opera a una frecuencia de 300MHz. Es una malla en donde cada slot de las filas tiene acceso a los cuatro vecinos más cercanos (dirección este, oeste, norte y sur). La malla es una red del tipo destino. Esto implica que las

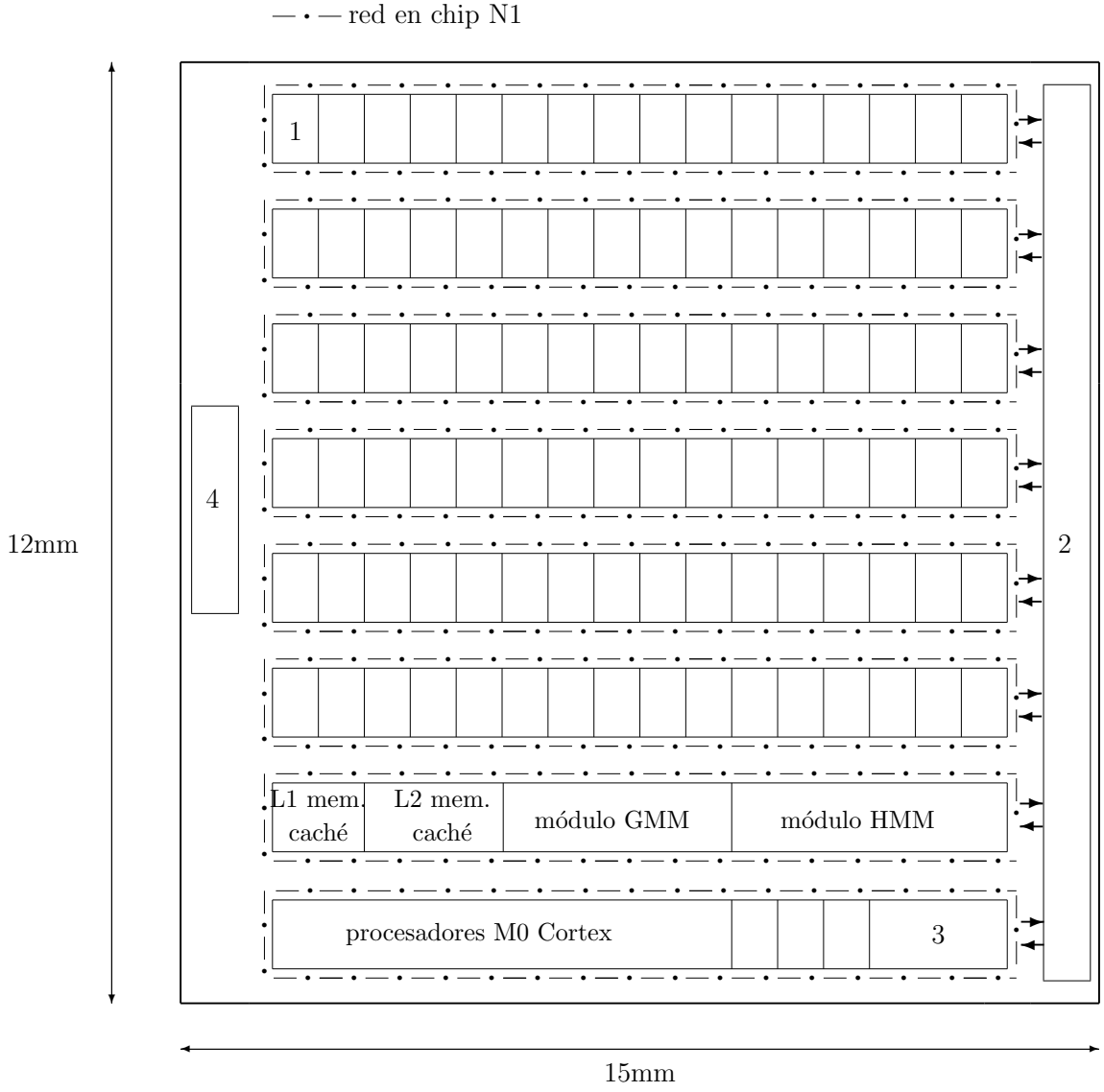


Figura 3.2: Diagrama en bloque del sistema red en chip N1. Referencias: 1) unidad de procesamiento; 2) interfaz memoria DDR; 3) nodo de reloj; 4) nodo FPGA.

unidades de procesamiento reciben paquetes sin conocimiento de la fuente de envío. El Apéndice .2 describe con mayor detalles las redes en chip del sistema.

La memoria principal es una memoria DRAM implementada en tecnología 3D y opera a una frecuencia de 1.6 GHz. La interfaz de la memoria posee 8 puertos independientes, uno por cada fila, que se conectan a los anillos de la red N1. Entre cada puerto y la memoria hay un buffer de pedidos de lectura y/o escritura de forma tal de realizar el cruce de dominios de reloj. Desde el punto de vista de una unidad de procesamiento, para operar la red N1 de manera eficiente se deben realizar varios pedidos en forma secuencial. Estos últimos son almacenados en el buffer y luego la memoria principal va a atender cada uno de ellos también en forma secuencial pero a una frecuencia de operación mayor. Esta operación de la red también ayuda a reducir la latencia de la memoria principal dado que ésta trabaja con páginas de memoria. Esto implica que si los pedidos a memoria corresponden a una página diferente a la que esté trabajando la memoria, la latencia se incrementa considerablemente dado que la memoria no puede cambiar de página hasta que no atiende todos los pedidos de la página actual.

El sistema de reloj incluye dos unidades phase locked loop, cuatro osciladores y una fuente de voltaje de referencia que provee los voltajes de bias para aquellas unidades de procesamiento con circuitos de señal mixta. Este nodo puede ser configurado por el procesador M0 a través de la red N2. Existe además un nodo de comunicación adicional para la red N2 que permite conectar una FPGA externa utilizando los puertos de entrada y salida del sistema en chip (de ahora en más CMP, del inglés *Chip MultiProcessor*). Esta

FPGA puede eventualmente llevar a cabo las tareas asignadas al procesador M0, proveyendo triple redundancia.

Las unidades que se instancian en el CMP son unidades de procesamiento para multiplicación vector-vector [29], unidades de procesamiento morfológico [30] y arreglos de dispositivos basados en inyección de carga [33]. Todas las unidades previamente mencionadas tienen en común que los datos de parámetros que utilizan para su procesamiento se encuentran almacenados localmente en vez de estar localizados en una memoria remota. Esto implica un ahorro en términos de tiempo de cómputo y energía.

3.2. Sistema de reconocimiento automático de voz

3.2.1. Arquitectura

La Fig. 3.3 muestra el diagrama en bloques del diseño. La entrada de la interfaz acústica son muestras de audio digitales para un marco de datos y su salida es un vector de 39 dimensiones de características acústicas CCFM. El procesador GMM recibe dicho vector y evalua la sumatoria de distribuciones Gaussianas para cada fonema. El resultado de dicha evaluación se almacena en una memoria caché. El procesador HMM implementa el algoritmo de Viterbi y utilizando las evaluaciones almacenadas en la memoria caché y los parámetros del modelo brinda como resultado la secuencia de fonemas más probable.

La interfaz acústica convierte una señal variante en el tiempo a una re-

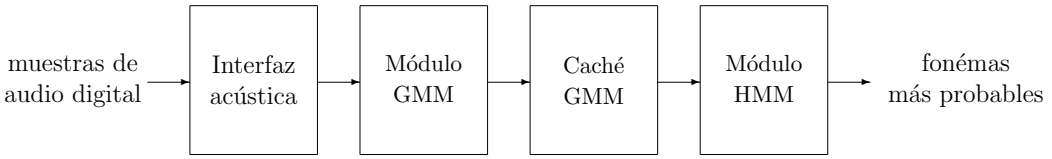


Figura 3.3: Diagrama en bloques de la arquitectura.

presentación en el dominio frecuencia [14]. La características que la interfaz extrae se denominan CCFM como se detalló anteriormente. Los detalles del flujo de procesamiento de la interfaz acústica se pueden encontrar en el Sección .1.

El módulo GMM computa las probabilidades de observación de cada vector de coeficientes CCFM. El módulo recibe un vector de entrada de 39 dimensiones y evalua distribuciones Gaussianas de múltiples variables tomando como argumento el vector de entrada. El bloque requiere de parámetros para realizar el procesamiento: la matriz de covarianza y la media de las distribuciones. Dado que la matriz se considera diagonal, la cantidad de parámetros requeridos son 78 de 16 bits cada uno lo que equivale a 1280 bits por distribución. Dichos parámetros son almacenados en la memoria principal por lo que el módulo requiere de múltiples lecturas a memoria.

La Fig. 3.4 muestra el diagrama en bloques del módulo que está compuesto de: un árbol aritmético, una unidad de ordenamiento, un módulo de control de parámetros y una unidad de control que sincroniza las transferencias de memoria y el cómputo que se lleva a cabo. En la Fig. 3.4 se considera que el bus de datos es de 128 bits y 5 puertos de memoria disponibles por lo que se requieren de 2 lecturas consecutivas por distribución.

Las probabilidades de observación se modelan utilizando un modelo de Gaussianas mixtas. El computo que se realiza está descripto por la ecuación 2.13. Las distribuciones se computan en el dominio logarítmico por dos razones principales: el cómputo resulta más sencillo y mejora la estabilidad numérica. Para distribuciones Gaussianas mixtas, esta estrategia de computo conduce a una simplificación dado que el logaritmo de la suma de funciones Gaussianas resulta más complejo de computar. La simplificación consiste en computar el logaritmo de cada una de las distribuciones involucradas en la suma y luego establecer el máximo de esos valores como el valor final.

El módulo tiene la capacidad de computar una mezcla de distribuciones Gaussianas o sólo una de ellas. La estrategia adoptada para computar las probabilidades de observación es utilizar un formato de computo tipo árbol. Este forma de evaluar distribuciones consiste en computar un juego inicial de distribuciones llamado Gaussianas Nivel 1 (L1). Los resultados son ordenados y aquellos que tienen mayor valor son re-evaluados utilizando un juego de distribuciones más detallado llamado nivel 2 (L2). Esta estrategia ahorra tiempo de computo y energía ya que en vez de computar un juego de distribuciones detallado para cada probabilidad se utiliza inicialmente un juego más simple.

Los detalles de implementación de la unidad aritmética, la unidad de ordenamiento y el modo de cómputo tipo árbol se presentan en el Sección .1.

El funcionamiento del módulo GMM es el siguiente: cuando recibe un vector CCFM, la unidad de control aritmética realiza lecturas a memoria DRAM para obtener los parámetros (μ, Σ) para el cómputo de las gaussianas L1. El bloque aritmético evalua el valor del logaritmo de las gaussianas y

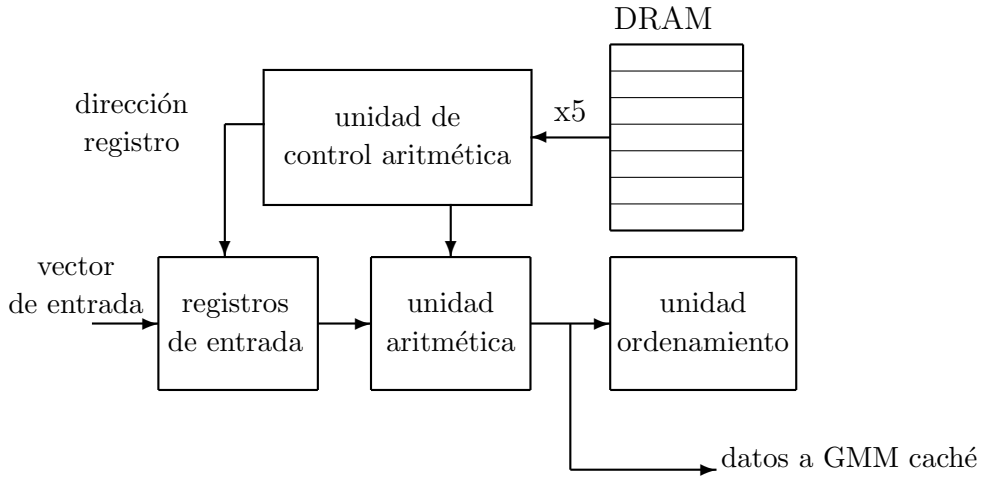


Figura 3.4: Diagrama en bloques módulo GMM

envia los resultados a la unidad de ordenamiento.

A medida que se computan las distribuciones L1, los valores son ordenados por la unidad de ordenamiento que crea una lista de hasta 64 valores. Una vez que todos los datos son ordenados, se leen en forma secuencial. El módulo tiene la capacidad de computar distribuciones L1 y L2 o sólo L1. Si sólo se evaluan las distribuciones L1, a medida que estas son evaluadas, los resultados son almacenados en la memoria cache de probabilidades. En el caso de que se evaluen ambos niveles, por cada uno de los valores leido de la unidad de ordenamiento, la unidad de control aritmética busca a memoria los parámetros necesarios para evaluar distribuciones L2 (una mixtura de distribuciones o sólo una distribución) y finalmente el resultado se almacena en la memoria caché de probabilidades. Esta última actua de buffer entre el módulo GMM y el módulo de cadenas de Markov oculta.

En el Sección .1 se describe con más detalle el diseño y funcionamiento de la memoria caché mencionada.

El módulo HMM está diseñado para computar el algoritmo Viterbi [34], por lo que devuelve como resultado el estado oculto más probable de una secuencia de eventos observables. El diseño está optimizado para reconocimiento de voz, sin embargo dado que el control de flujo de procesamiento utiliza datos de memoria es posible utilizar el mismo hardware como un decodificador de Viterbi más genérico dependiendo de la información almacenada en memoria.

La Fig. 3.5 muestra la arquitectura del módulo. La unidad de hipótesis inicia el lazo de procesamiento al escribir en la cola de ordenamiento un número configurable de estados de fonema. Los datos de la cola se leen secuencialmente y por cada uno de ellos el controlador busca la información de las dos posibles transiciones de estado utilizando los dos puertos dedicados de memoria. Esta información comprende las direcciones de la memoria caché de probabilidades y la probabiliad de transición. La cantidad de bits requeridos son 1024 bits por estado. Al considerar un bus de datos de 128 bits y dos puertos de memoria disponibles se requieren de dos lecturas de memoria consecutivas por puerto. Cuando se reciben los datos requeridos, el controlador realiza 4 lecturas consecutivas en los 4 puertos dedicados de la memoria caché de probabilidades y genera los dos posibles estados de fonema y los envia a la unidad aritmética.

La unidad aritmética utiliza los datos recibidos de la memoria caché de probabilidades junto con la probabilidad de transición para actualizar la probabilidad de los estados de acuerdo a la Ecuación 2.6. Luego la unidad de ordenamiento utiliza la probabilidad actualizada de los estados para crear una lista ordenada de los mismos. Cuando la cola de ordenamiento está va-

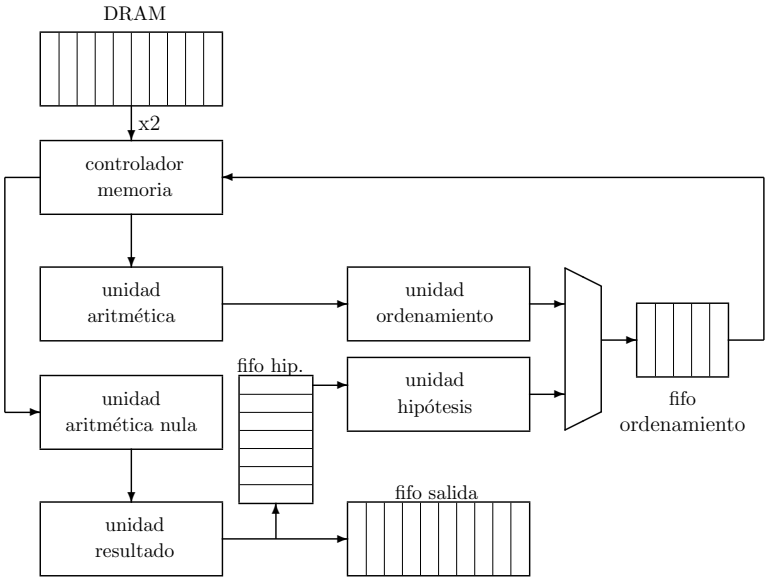


Figura 3.5: Arquitectura módulo HMM

cia y todos los estados fueron ordenados el procesamiento finalizó.

Una vez que la memoria caché de probabilidades tiene almacenados los valores correspondientes a un marco de datos nuevo, se inicia una nueva iteración en el lazo de procesamiento. Esto implica que la unidad de ordenamiento carga la lista ordenada en la cola y el procesamiento previamente descripto se repite.

La unidad aritmética nula actualiza la probabilidad de los estados nulos. La arquitectura cuenta con dos unidades artiméticas debido a que, a diferencia del resto de los estados, la actualización del estado nulo no requiere de la probabilidad de observación. El fonema cuyo estado actual es el nulo, es enviado a la unidad resultado que modifica el formato de los datos y envia el dato modificado a la cola de resultados y a la cola de hipótesis. La unidad de hipótesis recibe los datos de esta cola e inserta 64 nuevos estados iniciales en la cola de ordenamiento.

La unidad de ordenamiento consiste en un arreglo lineal de comparadores. El orden de complejidad es lineal en tiempo y espacio. Como resultado la unidad devuelve un arreglo ordenado de 64 fonemas comenzando por el de mayor probabilidad de estado. Esta unidad tiene un registro por cada etapa de ordenamiento por lo que permite una tasa de procesamiento de un dato por ciclo. Permite también el solapamiento de la operación de lectura y ordenamiento, de modo tal que el ordenamiento del próximo marco de datos puede comenzar mientras aún se este completando la lectura del marco anterior.

Los detalles de la arquitectura de esta unidad de procesamiento se pueden encontrar en el Sección .1.

3.2.2. Implementación

El diseño del procesador para reconocimiento de voz automático ocupa 15 columnas de una fila. Dado este requerimiento de área, el diseño cuenta con 15 nodos de comunicación ya que cada columna cuenta con uno. Esto permite que los pedidos de lectura y/o escritura se realicen en forma paralela optimizando así la tasa de datos. Como se detallará más adelante el diseño realiza varios pedidos de lectura en forma simultánea.

En el sistema en chip se diseñaron interfaces entre las redes N1 y N2 y los módulos de procesamiento. De esta manera, los modulos de procesamiento pueden recibir y entregar datos en ambas redes y también ser configurados a través de la red N2. Para realizar la integración fue necesario diseñar la lógica de control para que efectúe pedidos a la red N1 de una manera eficiente

teniendo en cuenta la estructura de la memoria DRAM. Ante la dificultad de trabajar en modo ráfaga y teniendo en cuenta que la interfaz de la memoria se comparte con el resto de las unidades de procesamiento presentes en el sistema en chip, el impacto de la latencia de la memoria principal es mayor. Para mitigar el efecto de la latencia se introdujo una jerarquía de memoria en los módulos de procesamiento, en particular para el almacenamiento de los parámetros de procesamiento.

A continuación se detalla el diseño de las interfaces, de la jerarquía de memoria y las arquitecturas de cómputo modificadas.

3.2.2.1. Módulo GMM

Los módulos cuentan con una jerarquía de memoria de dos niveles. Para procesadores de propósito general, la jerarquía de memoria tiene varios niveles comenzando por el nivel más bajo que consiste en un juego de registros ubicados junto al procesador. Los niveles subsiguientes son las memorias cache de nivel 1 y 2 (los procesadores actuales tienen hasta nivel 3). El último nivel en la jerarquía lo ocupan la memoria principal y el disco duro. A medida que se incrementa el nivel, se incrementa la capacidad de almacenamiento, la latencia y el tiempo de acceso. La ventaja principal de la jerarquía es que permite explotar la localidad temporal y espacial de los datos [15]. La localidad temporal da una noción de cuán probable es que los datos sean reutilizables por el procesador en procesamiento futuro. La localidad espacial da una noción de cuán probable es que datos adyacentes al dato requerido sean reutilizados por el procesador en el futuro. La jerarquía de memoria es la solución que los diseñadores digitales han encontrado para mitigar la

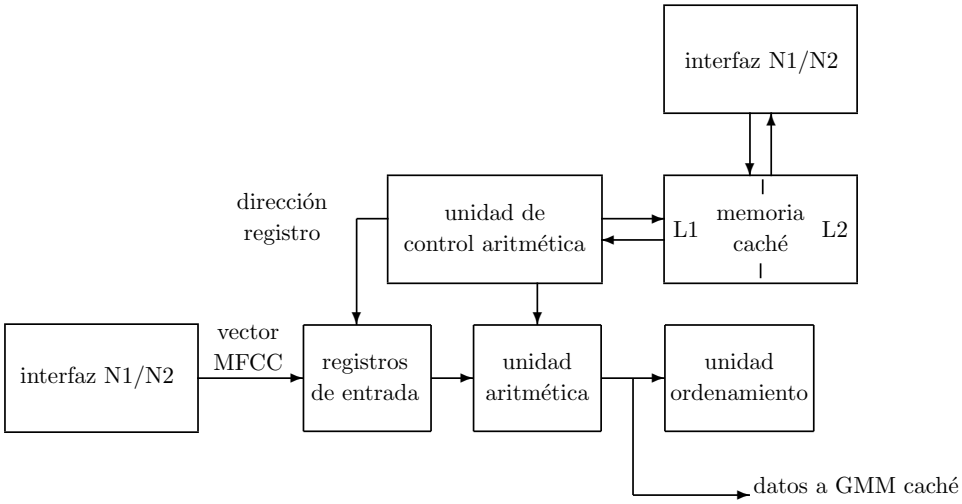


Figura 3.6: Diagrama en bloques módulo GMM integrado al sistema de multi-procesadores.

diferencia existente entre la velocidad de operación de las memorias DRAM y los procesadores [15].

La Fig. 3.6 muestra el diagrama en bloques del módulo GMM integrado a la arquitectura del sistema de multiprocesadores. Como se puede observar cuenta con las interfaces N1/N2 y se incluyeron memorias caché para el almacenamiento temporal de los parámetros de las distribuciones. El módulo cuenta con capacidad de almacenamiento para computar localmente 256 distribuciones L1 y 128 distribuciones L2 siguiendo la estrategia de cómputo descripta en la Sección .1.0.6. Por lo tanto el sistema es capaz de computar 384 distribuciones sin tener que buscar datos a memoria principal.

La memoria principal del CMP tiene memoria suficiente para el almacenamiento de parámetros de miles de distribuciones. Sin embargo como se explicó se requiere de una jerarquía de memoria dado que la latencia de la memoria principal puede impactar considerablemente en la tasa de procesa-

miento del módulo. Se diseño una memoria caché para el almacenamiento de parámetros de distribuciones L1 y otra para distribuciones L2.

Si el dato requerido por el procesamiento está presente en la memoria caché, dicho evento se llama *acierto*. Caso contrario es un *desacierto*. Cuando ocurre este último, se busca en memoria principal o en el nivel superior en la jerarquía un bloque de datos de un tamaño fijo y se ubica dicho bloque en la memoria de acuerdo a una determinada regla.

En términos de diseño, se deben tener en cuenta tres aspectos: el tamaño del bloque, la regla de ubicación y reemplazo de los bloques. La regla de ubicación trabaja con arreglos de bloques. La cantidad de bloques que pueden ser ubicados dentro de un arreglo indica el nivel de asociatividad. Por ejemplo si un arreglo consiste de cuatro bloques entonces el nivel de asociatividad es 4. Una vez que el bloque que se buscó en memoria principal se mapea a un arreglo particular, se ubica este último en cualquier posición dentro del arreglo. En el caso que todas las posiciones ya estén ocupadas la regla de reemplazo define cual de los bloques será reemplazado.

La elección de los tres aspectos mencionados en el párrafo anterior impacta en el rendimiento de la memoria caché. Sin embargo existe una relación de compromiso entre ellos. A medida que se incrementa el nivel de asociatividad, el rendimiento de la memoria también lo hace, pero por otro lado, su implementación resulta más compleja, aumenta la latencia y los beneficios en el rendimiento tienen menos impacto [15].

En el caso de las distribuciones L1, se evaluan los mismos modelos en cada marco de datos, por lo que la localidad temporal y espacial de la memoria caché L1 es alta. Esto genera un ahorro en tiempo de cómputo y energía. Para

minimizar la cantidad de accesos a memoria principal, el diseño maximiza el almacenamiento de los parámetros L1. Si es necesario el procesamiento de un número de distribuciones superior a la capacidad de almacenamiento de la memoria caché, esto puede ser llevado a cabo pero se deben buscar en memoria principal los datos que no se encuentran almacenados localmente lo que reduce la tasa de procesamiento.

En el caso de las distribuciones L2, los modelos a evaluar pueden diferir de marco a marco ya que depende del ordenamiento de los valores L1. Esto implica que su localidad temporal es baja y puede suceder que el módulo necesite de parámetros que no están almacenados localmente y por lo tanto se deberán efectuar lecturas a memoria principal.

Por lo expuesto anteriormente el diseño contempla dos memorias cachés, una para cada nivel (L1 y L2) dado que una sola memoria caché para ambos niveles de cómputo no es eficiente desde el punto de vista de la localidad. Sin embargo, el controlador es compartido por ambos niveles. Debido a restricciones en el área de silicio disponible, la capacidad de almacenamiento es 40 Kbytes para L1 y 20 KBytes para L2, lo que permite computar 256 y 128 distribuciones Gaussianas respectivamente sin tener que buscar datos a memoria principal. Por cada distribución Gaussiana, el bloque necesita un vector μ de 39 dimensiones, la matriz diagonal de covarianza Σ de 39x39, el peso de la mezcla y el valor etiqueta que identifica cada distribución. Cada uno de estos valores se representa por un número de 16 bits, dando un total de 1280 bits.

Las memorias caché L1 y L2 se implementaron con un nivel de asociatividad 2. No se implementó un nivel de asociatividad mayor por las razones

expuestas anteriormente[1]. El tamaño del bloque de datos elegido es de 1280 bits y la política de reemplazo implementada se conoce en la bibliografía como *el menos utilizado* [15]. La política de reemplazo del menos utilizado opera de la siguiente manera: cada bloque dentro del arreglo posee un contador para realizar el seguimiento de su frecuencia de utilización. Si se quiere escribir o leer un dato cuyo bloque no se encuentra en la memoria caché y a su vez el arreglo de bloques correspondiente se encuentra utilizado en su totalidad, se examinan los contadores, se reemplaza el bloque menos utilizado y se reinicia el contador. El ancho de palabra del bus de datos de la red N1 es 256 bits, entonces se requieren de 5 pedidos de lectura para obtener todos los datos necesarios para computar una sola distribución Gaussiana. A pesar de que el tamaño del bloque de datos es de 1280 bits, ambas memorias tienen un ancho de palabra de 640 bits para reducir el ancho del bus y la cantidad de cables en la implementación física. La memoria L1 se organizó en 4 bancos de 640 bits con una profundidad de 128 palabras y la memoria L2 tiene 2 bancos del mismo tamaño. Si un pedido a memoria caché resulta en un evento *acierto*, la latencia es de 8 ciclos de reloj: 4 ciclos para definir si el bloque ya se encuentra almacenado y 4 ciclos adicionales para leer dos palabras de 640 bits que conforman el pedido de lectura. Si por el contrario el pedido resulta en un evento *desacierto* la latencia es variable y depende de la memoria principal. En caso de que todos los parámetros requeridos para computar las distribuciones se encuentren almacenados localmente, la tasa de procesamiento es 12 ciclos por valor computado. Caso contrario, la tasa

[1]Tener en cuenta que para asociatividad 2 la tasa de eventos *miss* es equivalente a la de una memoria con nivel de asociatividad 1 pero del doble de tamaño [15]

de procesamiento queda supeditada a la latencia de la memoria principal. El diagrama en bloque de la memoria caché L1 y L2 se muestra en la Fig. 3.7. El paquete de datos de la red se divide en dos mitades de 128 bits ya que se utilizan memorias de 128x128 bits. Ante un evento de desacierto, se realizan 5 lecturas a memoria identificadas con el campo etiqueta. Cuando se reciben los paquetes junto con su etiqueta se utiliza esta última como selector del multiplexor de entrada de los bancos de memoria. Ante un evento de acierto, se realizan dos lecturas consecutivas de forma tal de obtener todos los datos que el cómputo de la Gaussiana requiere.

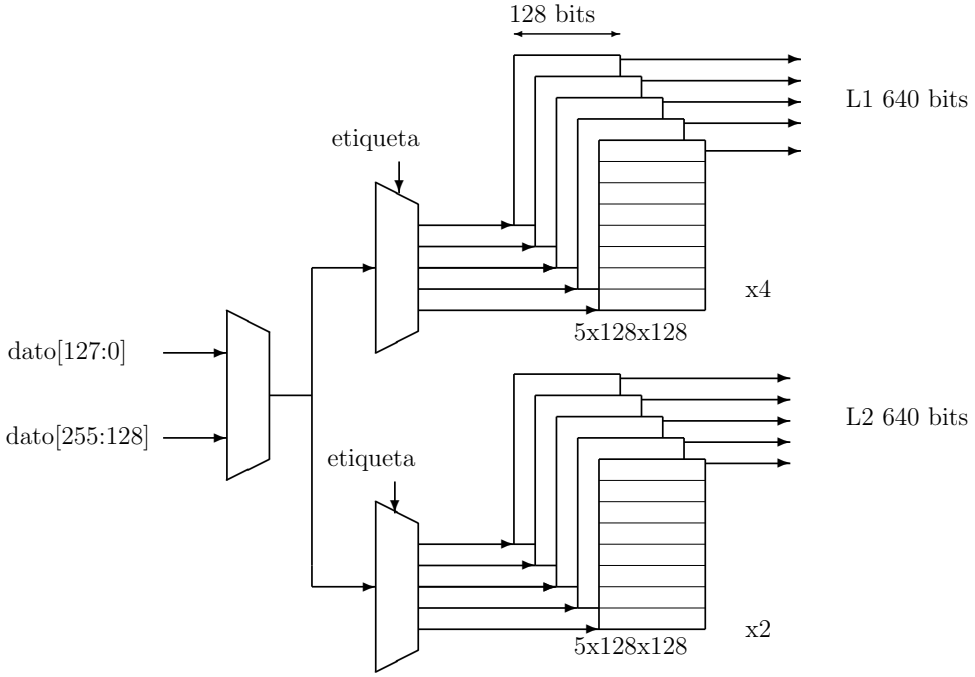


Figura 3.7: Diagrama en bloques de la memoria caché para el almacenamiento de parámetros L1 y L2.

El vector de entrada MFCC puede provenir tanto de la red N1 como de la red N2. Lo mismo sucede con los datos de la memoria caché. En principio

el diseño está pensado para que trabaje con la red N1 pero para búsqueda de fallas o ante un eventual mal funcionamiento de la memoria principal es posible realizar el procesamiento a través de la red N2. En este último caso los pedidos de lectura se envian al procesador CORTEXM0 que una vez que recibe los pedidos envía el dato correspondiente.

3.2.2.2. Módulo HMM

La Fig. 3.8 muestra la arquitectura del módulo HMM integrado a la arquitectura del sistema en chip. El principio de funcionamiento es similar al descripto en la Sección 3.2.1.

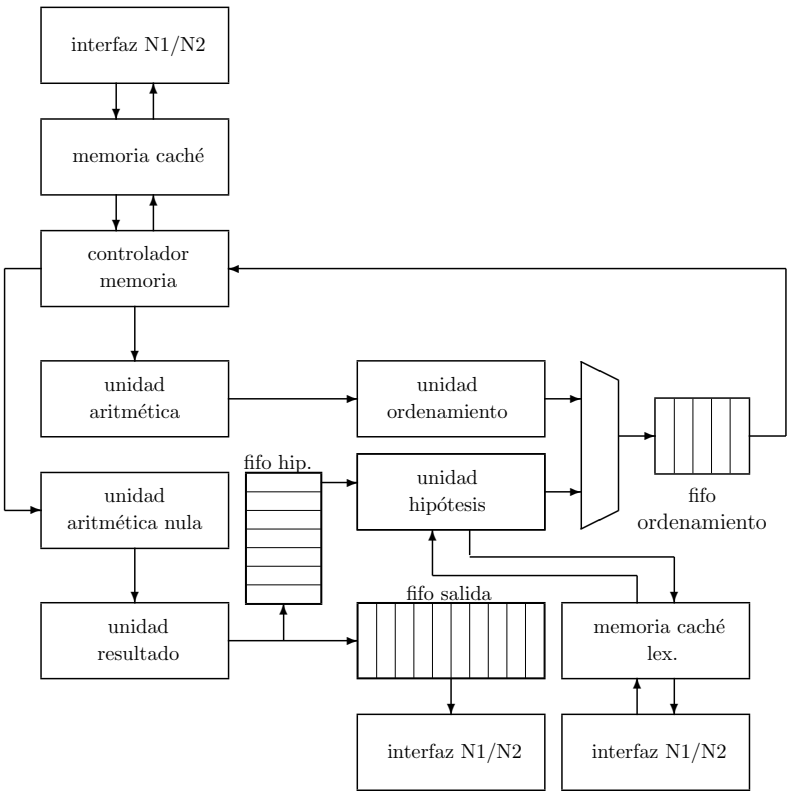


Figura 3.8: Arquitectura módulo HMM integrada al sistema de multiprocesadores

De forma similar que el caso anterior, el módulo cuenta con interfaces para la red N1 y N2 tanto para la entrada como salida de datos. Cuenta además con una memoria caché para el almacenamiento de los prámetros de las cadenas de Markov. La memoria caché se implementó con un nivel de asociatividad 1 para reducir el área de silicio, tiene una capacidad de almacenamiento de 4 KBytes y un ancho de palabra de 128 bits. Cada estado requiere de 1024 bits de información. La memoria se implementó con un ancho de palabra de 128 bits en vez de 1024 bits para reducir el bus interno. Por lo tanto es necesario leer 8 lugares de memoria para obtener toda la información necesaria. En el caso de que el dato se encuentre almacenado localmente el tiempo de respuesta de la memoria son 15 ciclos de reloj y se requieren 20 ciclos para computar la probabilidad de cada estado.

En caso que el dato no se encuentre localizado en memoria, el controlador de la memoria caché realiza 4 lecturas a memoria consecutivas a través de la red N1 para obtener toda la información necesaria ya que el bus de datos de la red es de 256 bits. En este caso el tiempo de respuesta de la memoria caché es variable ya que depende de la latencia de la memoria principal. Además se adicionó un modelo de árbol léxico para optimizar la tarea de inferencia del fonema más probable. Cuando el estado de los fonemas procesados es el tercero en la estructura (ver Fig. 2.2), la unidad de hipótesis permite la inserción de nuevos estados iniciales de fonemas de acuerdo a dos opciones. La primer consiste en incluir todos los posibles estados en el lazo de procesamiento. La segunda opción es buscar en un árbol léxico los fonemas más probables de acuerdo a un diccionario particular. La Fig. 3.9 muestra un ejemplo de árbol léxico en el idioma inglés. Cada circulo es un nodo en el árbol el cual puede

ser una raíz, una rama o una hoja del mismo. La Fig. 3.10 muestra el formato adoptado para describir los nodos del árbol en memoria. La Tabla 3.1 detalla la descripción de cada uno de los campos que componen este formato.

La unidad de hipótesis envía el fonema a la memorica caché que almacena los datos del árbol. El controlador de la memoria caché utiliza el campo "ptr"del fonema para generar direcciones para la memoria y realizar el pedido de lectura. Una vez que recibe el dato de memoria, identifica que tipo de nodo es el fonema: una rama, una hoja o la raíz. Esto se logra analizando el número de nodos que el fonema direcciona. Esta información está presente en el campo "#". Dependiendo de esta identificación, la unidad de control tiene dos modos de operación. Si es una rama, se busca la información de cada nodo que éste direcciona simplemente realizando tantas operaciones de lectura a memoria como nodos posea la rama. Estas operaciones se realizan a direcciones de memoria continuas a la del nodo. Si es una hoja, se busca la información de la raíz junto con todas las ramas por debajo de la misma. Si es la raíz se procesa como si fuera una rama.

La información que se obtiene de cada nodo es el número de identificación del fonema y el puntero a memoria "*prox. ptr.*"que se utiliza para localizar el nodo en el mapa de memoria del árbol.

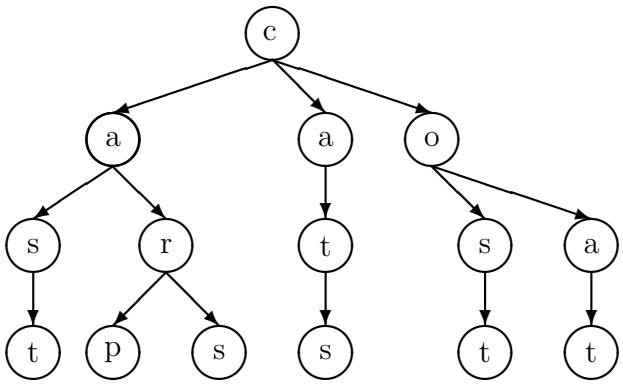


Figura 3.9: Ejemplo de árbol léxico.

campo	bits	descripción
palabra ID	15	número identificación de la palabra.
fonema ID	15	número identificación fonema
p(t)	18	probabilidad de transicionar al estado nulo
prox. ptr.	16	puntero al próximo nodo en el árbol léxico
#	10	cantidad de nodos por debajo de la rama

Tabla 3.1: Descripción de los campos utilizados para identificar los nodos del árbol léxico en memoria.

3.2.2.3. Interfaces dedicadas

Se diseñaron interfaces con la red N1 y N2 de modo tal que las unidades tengan acceso a estas redes. Estas interfaces poseen puertos de entrada y salida que permiten que la unidad de procesamiento se comunique con el nodo de comunicación. Las interfaces realizan un pedido de transacción y reciben paquetes de datos siguiendo el protocolo de 4 fases descripto en la Sección 3.1.1. Cada una ellas se conecta con un nodo de comunicación. Se diseñaron tres interfaces: la interfaz de comunicación N1, la interfaz de comunicación N2 y por último la interfaz de configuración que utiliza la red N2. Las Tablas 3.2, 3.3, 3.4 detallan los puertos con los que cuenta cada una de ellas. Los puertos relacionados con el sincronismo de las señales de datos y dirección

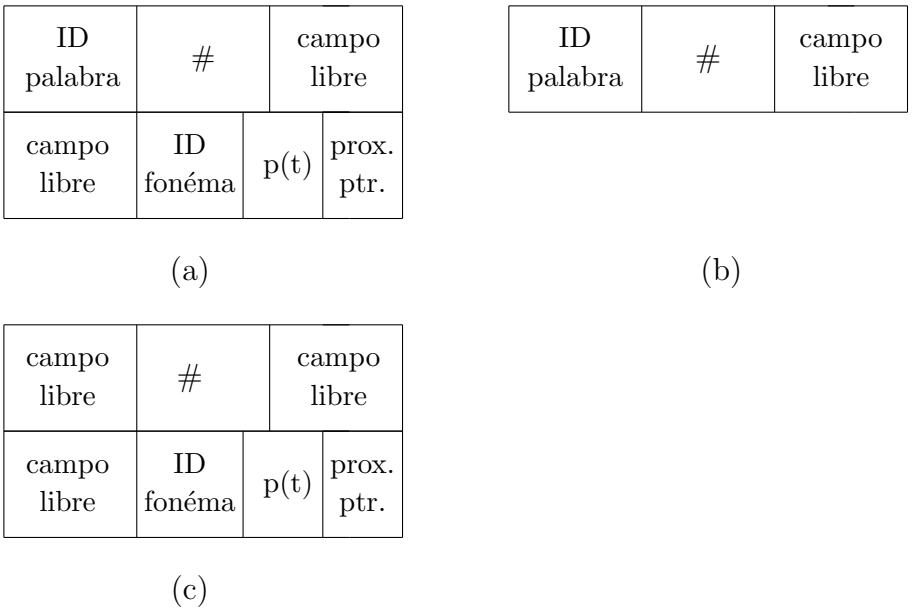


Figura 3.10: Formato adoptado para el árbol léxico. (a) Rama. (b) Hoja. (c) Raíz.

no se incluyeron en las tablas por una cuestión de simplicidad.

En el caso de la interfaz de comunicación N1, se configura la unidad de procesamiento para que realice lecturas o escrituras en un cierto rango de direcciones de la memoria principal. Luego, de acuerdo a los requerimientos del procesamiento la unidad realiza un pedido y presenta el dato, la dirección de memoria y el tipo de transacción a la interfaz, la cual gestiona las señales de sincronismo haciendo efectivo el pedido. Cuando este es reconocido por el nodo de comunicación, la interfaz envía la señal de reconocimiento a la unidad de procesamiento habilitando esta última a efectuar un nuevo pedido.

En el caso de la interfaz de comunicación N2, se configura la unidad de procesamiento para que envie paquetes a otra unidad dentro del arreglo. Luego la operación es similar a la de la interfaz N1.

Por último, la interfaz de configuración se comunica con la red N2 de

Nombre del puerto	**Nro. de bits**	**Descripción del puerto**
Puertos con conexión a la red		
PU_N1_data_o	256	salida de datos.
PU_N1_addr_o	40	salida de dirección.
PU_N1_tag_addr_o	16	salida de etiqueta del pedido.
PU_N1_op_o	2	salida de tipo de transacción: lectura o escritura.
N1_PU_data_i	256	entrada de datos.
N1_PU_tag_addr_i	16	entrada de etiqueta del pedido.
Puertos con conexión a la unidad de procesamiento		
PU_data_o	256	salida de datos.
PU_tag_o	16	salida de etiqueta del pedido.
PU_op_i	2	entrada de tipo de transacción.
PU_tag_i	16	entrada de etiqueta del pedido.
PU_addr_i	20	entrada de dirección.
PU_op_i	2	entrada de tipo de transacción.

Tabla 3.2: Señales de interfaz entre la red N1 y las unidades de procesamiento para el reconocimiento de voz automático.

forma tal que el procesador M0 configure las unidades. Los parámetros a configurar son selección del tipo de red a utilizar para recibir y enviar datos, las direcciones de la memoria principal para realizar pedidos de lectura y/o escritura, direcciones de la unidad de procesamiento destino para el envío de paquetes de datos y de sincronismo, parámetros propios de la unidad que definen el procesamiento de la misma y selección de señales internas para búsqueda de fallas. La Tabla 3.5 muestra cuántas interfaces dispone cada módulo del reconocimiento de voz automático.

Nombre del puerto	**Nro. de bits**	**Descripción del puerto**
Puertos con conexión a la red		
PU_N2_data_o	256	salida de datos.
PU_N2_tag_o	16	salida de etiqueta.
PU_N2_dest_horz_addr_o	5	salida de dirección horizontal.
PU_N2_dest_vert_addr_o	3	salida de dirección vertical.
N2_PU_data_i	256	entrada de datos.
N2_PU_tag_i	16	entrada de etiqueta.
Puertos con conexión a la unidad de procesamiento		
PU_data_o	256	salida de datos.
PU_tag_o	16	salida de etiqueta del pedido.
PU_dest_horz_addr_i	5	entrada de dirección horizontal.
PU_dest_vert_addr_i	3	entrada de dirección vertical.
PU_data_i	256	entrada de datos.
PU_tag_i	16	entrada de tipo de transacción.

Tabla 3.3: Señales de interfaz entre la red N2 y las unidades de procesamiento para el reconocimiento de voz automático.

Nombre del puerto	**Nro. de bits**	**Descripción del puerto**
Puertos con conexión a la red		
PU_N2_data_o	256	salida de datos.
PU_N2_tag_o	16	salida de etiqueta.
PU_N2_dest_horz_addr_o	5	salida de dirección horizontal.
PU_N2_dest_vert_addr_o	3	salida de dirección vertical.
N2_PU_data_i	256	entrada de datos.
N2_PU_tag_i	16	entrada de etiqueta.
Puertos con conexión a la unidad de procesamiento		
PU_data_o	256	salida de datos.
PU_tag_o	16	salida de etiqueta del pedido.
PU_dest_horz_addr_i	5	entrada de dirección horizontal.
PU_dest_vert_addr_i	3	entrada de dirección vertical.
PU_data_i	256	entrada de datos.
PU_tag_i	16	entrada de tipo de transacción.

Tabla 3.4: Señales de interfaz para la configuración de las unidades de procesamiento para el reconocimiento de voz automático.

Módulo	**Nro. de IF N1**	**Nro. de IF N2**	**Nro. de IF config. N2**	**Detalle**
Interfaz acústica	2	2	1	Interfaces N1 y N2 destinadas a la entrada y salida de datos.
GMM	2	2	1	Interfaces N1 y N2 destinadas a la memoria caché y la entrada de datos MFCC.
HMM	3	3	1	Interfaces N1 y N2 destinadas a las memorias caché y a la salida de datos.

Tabla 3.5: Cantidad de interfaces que dispone cada módulo.

3.2.3. Rendimiento

El rendimiento del módulo de reconocimiento de voz se mide en términos de la relación entre el tiempo de procesamiento y el tiempo real del audio. Dicha relación se nota con la letra griega η.

$$\eta = \frac{tiempo\ del\ audio}{tiempo\ de\ procesamiento} \tag{3.1}$$

La interfaz acústica genera un vector de características acústicas MFCC cada 10 mili segundos en tiempo real. Entonces para computar la mínima relación η para realizar la extracción de características en tiempo real se deben tener en cuenta el tiempo de procesamiento de cada etapa de la interfaz acústica. Comenzando por el solapamiento de 3 marcos de datos detallado en la Sección .1 se requieren de 480 datos (a una frecuencia de muestreo de

16 KHz se obtienen 160 muestras de audio digital cada 10 ms). Esto implica que la tasa de procesamiento debe ser 3 veces más rápido que el tiempo real. Luego la extensión de ceros que se realiza para lograr un marco de 512 muestras adiciona una aceleración de 1.067. Por último, la transformada discreta de Fourier requiere de 640 ciclos para procesar 512 muestras lo que implica una aceleración de 1.25. En total se requiere que el tiempo de procesamiento sea $3 \times 1{,}067 \times 1{,}25 \approx 4$ veces más rápido que el tiempo real del audio. Para la frecuencia de muestreo mencionada se requiere entonces que el reloj de procesamiento sea de al menos 64 KHz. En la implementación que aquí se presenta la frecuencia de operación máxima de 50 MHz permite una aceleración de 781.25.

En el caso del módulo GMM se realiza el cómputo de la aceleración para distintos números de distribuciones Gaussianas a computar. Como el número excede la cantidad de distribuciones que se pueden almacenar localmente se considera la latencia de la memoria principal. Si bien no se cuenta con la información de la latencia de la memoria se aproximó dicho número en 100 ns lo cual es un valor razonable teniendo en cuenta la tecnología DRAM [15]. La Tabla 3.6 muestra el valor de η para distinto número de Gaussianas computadas por marco. Las estimaciones considera la tasa de aciertos para la memoria caché implementada.

En el caso del módulo HMM se realiza también el cómputo de η para distintas cantidades de estados de la cadena de Markov computados por marco de datos. La Tabla 3.7 detalla los valores estimados. Para el valor de 16484 estados computados por marco, el valor de la relación es menor a uno por lo que se establece un máximo de 8192 estados por marco (para el valor

Nro. de Gaussianas por marco	η_{GMM}
512	71
1024	33.19
2048	16.07
4096	7.91
8192	3.92
16384	1.95
32678	0.91

Tabla 3.6: Rendimiento del módulo GMM para distinto número de distribuciones Gaussianas computadas.

Nro. de estados HMM por marco	η_{HMM}
256	124.6
512	62.3
1024	31.1
2048	15.6
4096	7.8
8192	3.89
16384	1.94

Tabla 3.7: Rendimiento del módulo HMM para distinto número de estados HMM computados.

considerado de latencia de la memoria principal). Aquí se consideró una tasa de aciertos cercana a cero de modo de considerar el caso más desfavorable.

Finalmente, dado que el módulo GMM se encuentra desacoplado del módulo HMM por la memoria caché de probabilidades, es válido decir que para 4096 distribuciones Gaussianas computadas y 4096 estados HMM la aceleración resulta 7.8. Otras configuraciones son posibles, es decir se podría implementar un sistema de mayor capacidad cómputo pero en detrimento de la relación η.

Tomando como casos de comparación los trabajos previamente analizados en la Sección 2.2 se construye la Tabla comparativa 3.8. Se describe el

Trabajo	**Etapas**	$\boldsymbol{\eta}$	**Tecn.**	$\mathbf{F_{max}}$ **[MHz]**	**Consumo [mW]**	**Tamaño**	**Mem. int. [KB]**
[18]	Eval-Inf	3.02	ASIC 40nm	200	144	3.86 mm^2	536
[19]	Inf	10	FPGA 90nm	100		58K celdas logicas	542
[17]	Eval-Inf	1.52	FPGA 90nm	100		21K celdas lógicas	393
[20]	Eval	1.4	FPGA 90nm	10	15.2	93K celdas lógicas	26
[10]	Ext-Eval-Inf	33.3	ASIC 0.18μm	80	421.5	350K celdas lógicas	
[21]	Ext-Eval-Inf	1	ASIC 65 nm	50	6	6.25 mm^2	138
Este trabajo	Ext-Eval-Inf	7.8	ASIC 55 nm	50		15 mm^2	84.8

Tabla 3.8: Comparativa de la arquitectura integrada con trabajos previos.

procesamiento que realiza cada implementación el cual se dividide en tres etapas: extracción del vector acústico (Ext), evaluación de las probabilidades de observación (Eval) e inferencia de los fonemas (Inf). El sistema integrado logra un buen rendimiento, siendo solo superado por [10]. Sin embargo, dicho sistema utiliza un módelo de Markov a nivel palabra lo cual es una limitante para vocabularios de gran tamaño y además calcula distribuciones Gaussianas simples y no sumatorias. En el caso de [21] el bajo consumo obtenido se logra reduciendo el voltaje de operación y utilizando la técnica de clock gating [35].

3.3. Implementación física

Las unidades de procesamiento descriptas en esta sección se implementaron en dos tecnologías diferentes: en tecnología CMOS de 55 nm de bajo voltaje y en un proceso CMOS de 16 nm. En ambos casos se utilizaron las herramientas DC Compiler de Synopsys y Innovus de Cadence. Previo a estas implementaciones en VLSI se realizó una implementación en FPGA para verificar el correcto funcionamiento del diseño. A continuación se describen todas las realizaciones realizadas.

3.3.1. Implementación en FPGA

Se realizó una implementación en una FPGA Xilinx SPARTAN6 XC6SLX150 la cual se encuentra embebida en una placa de prototipado OpalKelly XEM6310. El propósito de esta implementación es verificar el correcto funcionamiento del diseño.

La plataforma Opalkelly permite comunicarse con la PC a través de una puerto USB. El fabricante provee la interfaz de comunicación la cual es propiedad intelecutal del fabricante. La comunicación con la plataforma se puede realizar a través de scripts utilizando librerías implementadas en lenguaje C. Dichas librerías luego pueden ser utilizadas en Matlab, lenguaje C o Python. En este caso se implementaron varios scripts en lenguaje Python con la función de enviar datos al núcleo de procesamiento y luego recibir los resultados.

La interfaz de comunicación provista por el fabricante ya se encuentra implementada para un frecuencia de operación de 100MHz. Por lo tanto se

Bloque	**Tamaño [KB]**
Memoria caché GMM	20
Memoria caché de probabilidades	16
Memoria caché HMM	8
Memoria caché árbol léxico	0.0625

Tabla 3.9: Tamaño de los bancos de memoria implementados en la FPGA.

debe implementar el diseño a dicha frecuencia o utilizar un PLL de modo tal de poder operar el bloque de procesamiento a una frecuencia menor. La primer opción resultó en problemas de temporizado no por el tamaño de lógica sino debido al ruteo de las señales. Debido a esto, se implementó el diseño de acuerdo a la segunda alternativa planteada. Para ello fue necesario la sincronización de datos entre dos dominios de reloj utilizando colas de datos que operen en dos dominios de reloj y estructuras típicas para el sincronismo de señales tipo pulsos y nivel. Los núcleos de procesamiento operan a 50 MHz por lo que se instanció un PLL de Xilinx de modo tal de generar dicho reloj a partir del reloj de la interfaz de 100MHz.

La jerarquía de memoria junto con los controladores de memoria provistos por Xilinx de modo tal de poder comunicarse con la memoria DDR2 de alta densidad con la que cuenta la plataforma resulta la implementación más adecuada. Sin embargo, para el propósito de esta implementación fue suficiente colocar un banco de memoria local que cuente con la mayor capacidad posible de modo tal que permita el almacenamiento local de los parámetros.

Las memoria caché implementadas tienen los tamaños descriptos en la Tabla 3.9. Además, la Tabla 3.10 muestra los recursos de la FPGA que se utilizan en la implementación.

Elemento	Utilizado	Utilización
Registros	44354	24 %
Slices	14961	64 %
bloques DSP	60	33 %
bloques de memoria RAM	26	9 %
administrador de reloj	2	16 %

Tabla 3.10: Elementos utilizados en la FPGA.

3.3.2. Implementación en 55 nm

La Fig. 3.11 muestra la implementación de la interfaz acústica. Se agruparon distintas instancias de procesamiento en un mismo bloque para facilitar su implemenación. Estas instancias se detallan en el Apéndice .1. EL filtro pre-enfásis, el alineador de marcos y la ventana de Hamming se agruparon en un mismo bloque. Luego, la escala de Mel, el cómputo del logaritmo, la transformada del coseno discreta (DCT) y el cálculo de los diferenciales se agruparon también en un mismo bloque. Algunas de las instancias utilizan bancos de memoria y los mismos se implementaron utilizando bloques de memoria SRAM que permiten ser operados a bajo voltaje. El bloque de la FFT utiliza bloques de memoria SRAM de profundidad 128 y de ancho de palabra de 24 bits. El bloque DCT utiliza bloques de profundidad 64 y ancho de palabra de 24 bits. El resto de las memorias y colas se implementaron con registros. La tabla 3.11 detalla los bloques de memoria utilizados y el tamaño total de los bancos de memoria.

La interfaz ocupa 3 columnas de la fila dando un área total de 3.46 mm^2 cuyas dimensiones son 1.28 mm x 2.7 mm.

En la tablas 3.12 se detalla el área que ocupa cada sub-bloque de la interfaz acústica.

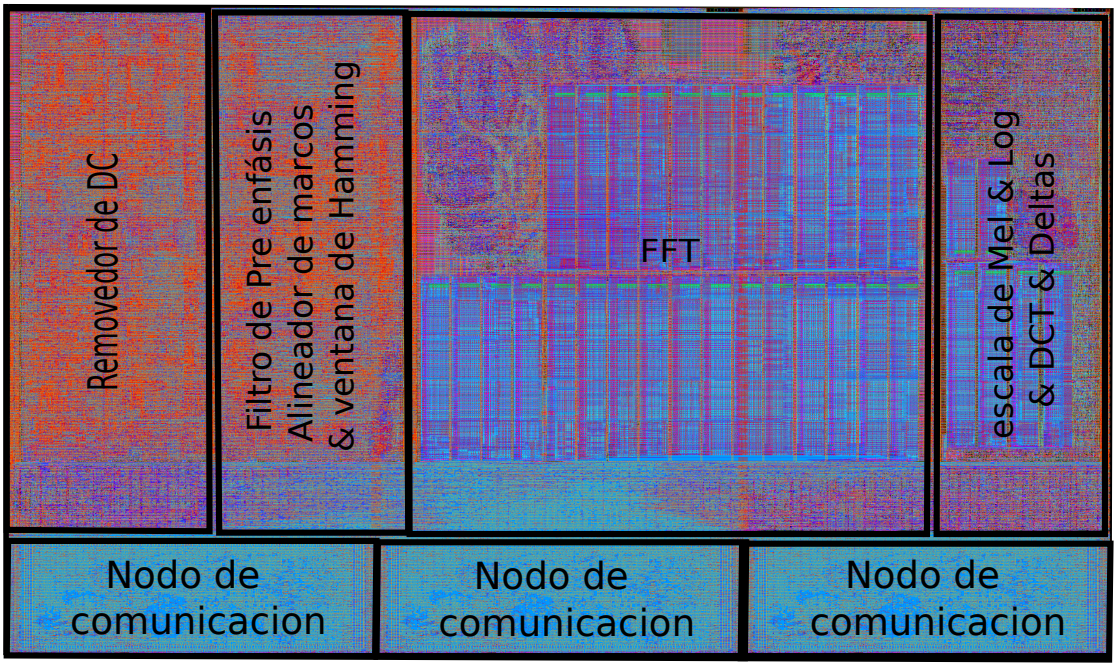


Figura 3.11: Implementación física de la interfaz acústica.

bloque	**tamaño del bloque de memoria**	**tamaño del banco de memoria**
FFT	128x24	84Kbits
DCT	64x24	1.536Kbits

Tabla 3.11: Bloques de memoria utilizados en la implementación de la interfaz acústica.

Al ser la interfaz acústica una unidad independiente resulta muy versatil dado que el vector de coeficientes acústicos puede ser la entrada de otras unidades de procesamiento. Por ejemplo, es posible crear una red neuronal de aprendizaje profundo utilizando arreglos de unidades que realizan la multiplicación vector-vector [29]. Esta red puede ser entrenada para realizar el reconocimiento de voz automático utilizando como datos de entrada el vector de coeficientes acústicos provisto por la interfaz.

La Fig. 3.12 muestra la simulación por eventos de la interfaz acústica.

bloque	**Área** $[mm^2]$
Removedor DC	0.5
Filtro pre-enfásis y alineador de marcos	0.5
FFT	1.37
scala de mel, log y deltas	0.42
interfaz acústica	3.46

Tabla 3.12: Área de silicio que ocupa la interfaz acústica.

En particular se observa actividad de las señales involucradas con la comunicación entre la interfaz y la memoria DDR (utilizando la red N1) y la primer etapa de procesamiento de la interfaz que está a cargo de remover la componente de continua de los datos que van ingresando. Se observa que ante un pedido de generación de un frame de vectores acústicos por parte del procesador CORTEXM0 se realizan múltiples pedidos a la memoria DDR (ver señal *PU_N1_req_o*). Estos datos son muestras de audio temporales y una vez obtenidos, tal como registra la actividad de la señal *N1_PU_req_i*, son procesados por la interfaz acústica. Los vectores resultantes se escriben en la memoria DDR o se envían directamente al módulo GMM a través de la red N2.

La implementación de los módulos GMM, HMM y la memoria caché de probabilidades ocupa 15 columnas de una fila. Se seccionó la imagen de la implementación de modo de brindar una mejor presentación de la misma. En la Fig. 3.13 se muestra la implementación del módulo GMM. La Fig. 3.14 muestra la implementación del módulo HMM. El área total que ocupan estos tres bloques es de 17.28 mm^2 cuyas dimensiones son 1.28 mm x 9 mm.

La Fig. 3.15 muestra la simulación por eventos del módulo GMM. Se puede observar que el módulo al no disponer de los parámetros que requiere

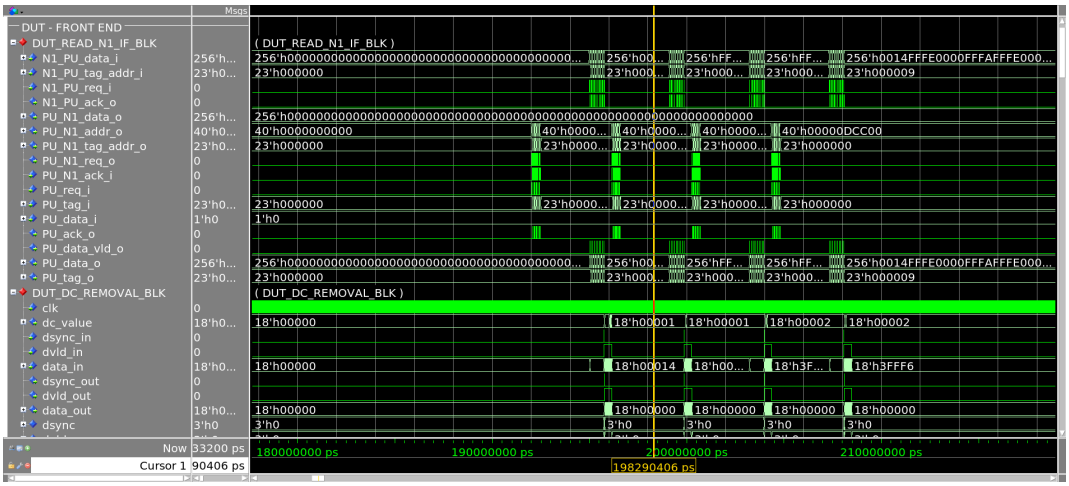


Figura 3.12: Simulación por eventos de la interfaz acústica.

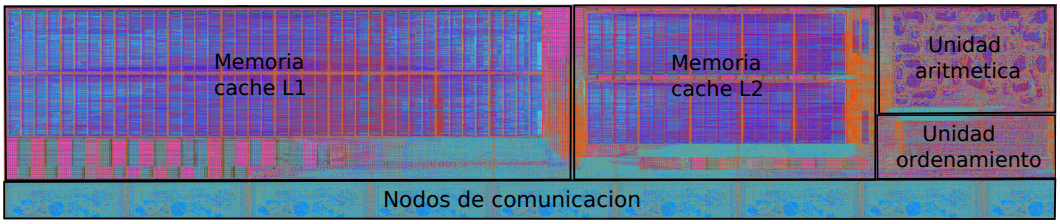


Figura 3.13: Implementación física del módulo GMM.

en la memoria caché, realiza pedidos de datos a la interfaz de la red N1 utilizando la señal *gmm_mem_rd_req_o*. A su vez la interfaz realiza los pedidos a memoria DDR y al recibir los datos los envia de vuelta al módulo a través de la señal *PU_data_o*. El módulo recibe los datos y los almacena en la memoria caché tal como se observa en la actividad de las señales *cache_wr_en_l1_o* y *cache_wdata_l1_o*. Estos datos son procesados y los valores resultantes se presentan a la salida de los puertos *result_vld_o*, *result_addr_o* *result_data_o* que se envían a la memoria caché de probabilidades a través de un bus de comunicación dedicado.

La Fig. 3.16 muestra la simulación por eventos del módulo HMM. El co-

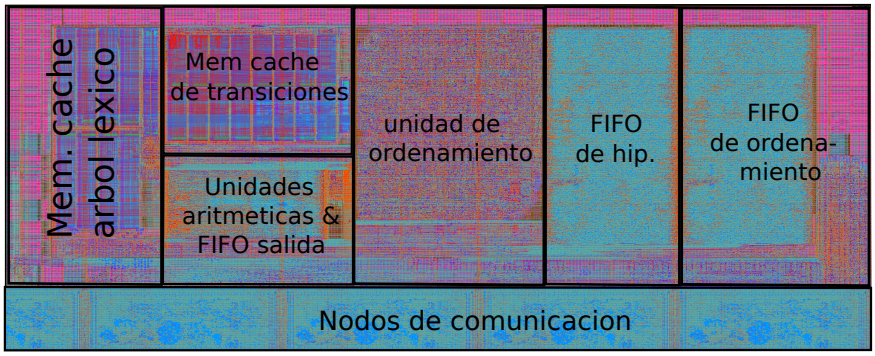


Figura 3.14: Implementación física del módulo HMM.

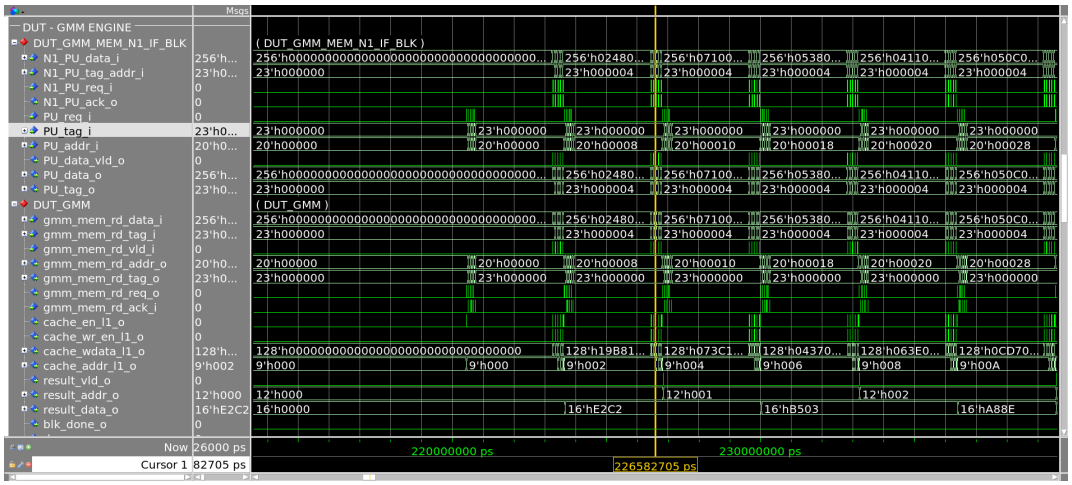


Figura 3.15: Simulación por eventos del módulo GMM.

mienzo del procesamiento ocurre con un pulso en la señal *frame_sync_i*. Este pulso lo genera la memoria caché cuando el módulo GMM completó la operación de escritura de las nuevas probabilidades. Se observa que el módulo al no disponer de los datos de las probabilidades de transición en la memoria caché, realiza pedidos de datos a la interfaz de la red N1 utilizando la señal *tmem_rd_req_o*. Al igual que en el caso anterior, la interfaz realiza los pedidos a memoria DDR y al recibir los datos los envia de vuelta al módulo a través de la señal *PU_data_o* que son luego almacenados en la memoria caché de

transiciones. A su vez, el módulo busca en la memoria local los valores de probabilidad de observación. Esto se puede observar en el bus de direcciones *gmm_rd_addrx_o* y la señal *gmm_rd_req_o*. Una vez que el módulo disponde de todos estos datos, los fonemas son procesados y los más probables se presentan a la salida de los puertos *result_dvld_o* y *result_data_o* que se envian a la memoria DDR u otra unidad de procesamiento tal como el procesador CORTEXM0 a través de la red N2 .

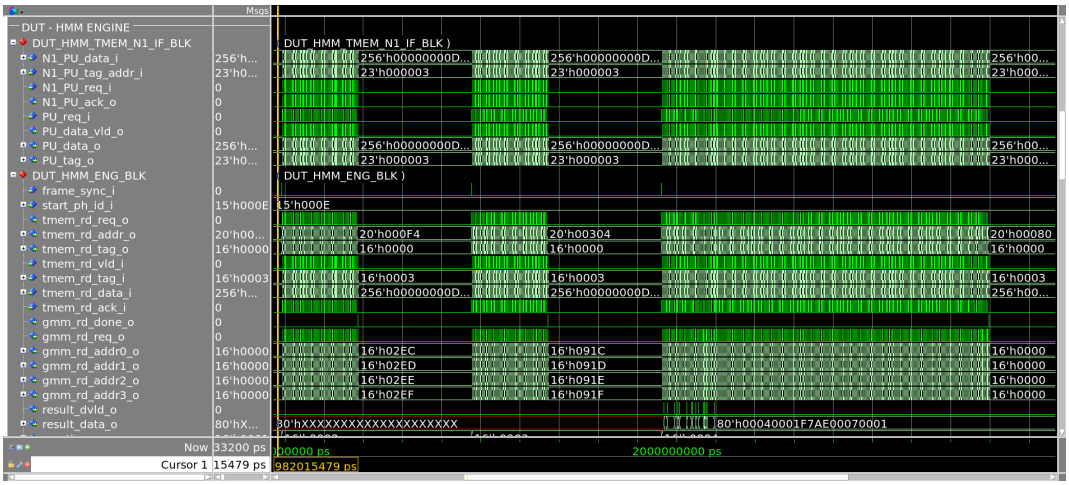


Figura 3.16: Simulación por eventos del módulo HMM.

La Fig. 3.17 muestra la memoria caché de probabilidades. Se puede observar que está compuesta de dos grupos de bancos de memoria tal como se describió en la Sección .1.0.8.

La Fig. 3.18 muestra la simulación por eventos de la memoria. Esta recibe los datos del módulo GMM tal como se evidencia en la actividad de las señales *wr_addr_i* y *wr_data_i*. Se observa claramente como se realiza en primer lugar la operación de escritura de las probabilidades y luego la lectu-

ra, por parte del módulo HMM, a través de las señales *rdx_req_i*, *rdx_addr_i*, *rdx_vld_o* y *rdx_data_o*.

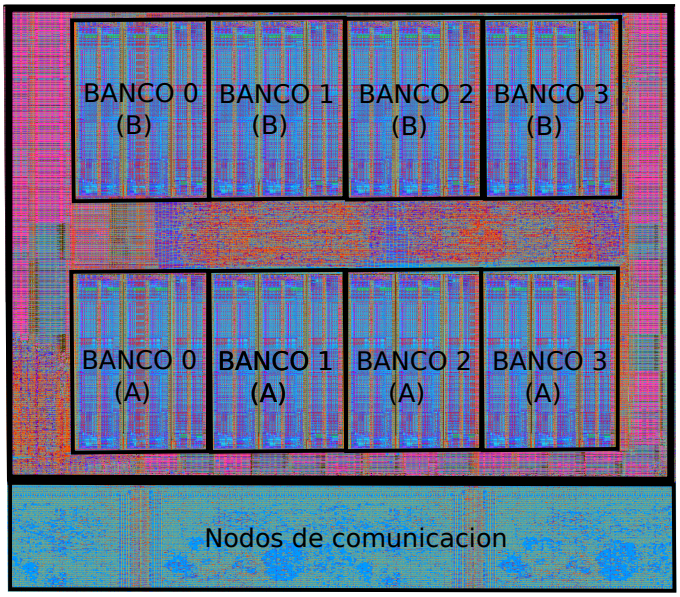


Figura 3.17: Implementación física de la memoria cache de probabilidades.

La Fig. 3.19 muestra la simulación por eventos de los módulos de procesamiento operando en conjunto. Se puede observar que los módulos trabajan en cascada, teniendo la memoria caché como almacenamiento local intermedio.

La Tabla 3.13 muestra los bloques de memorias de bajo consumo utilizadas en la implementación. El área total ocupada teniendo en cuenta la interfaz acústica, los módulos GMM, HMM y la memoria caché de probabilidades es de 20.74 mm^2. Recordar que una columna requiere un área de 1.281 mm x 0.9 mm sin incluir el nodo de comunicación.

En las Tablas 3.14, 3.15 y 3.16 se detalla el área que ocupa cada sub-

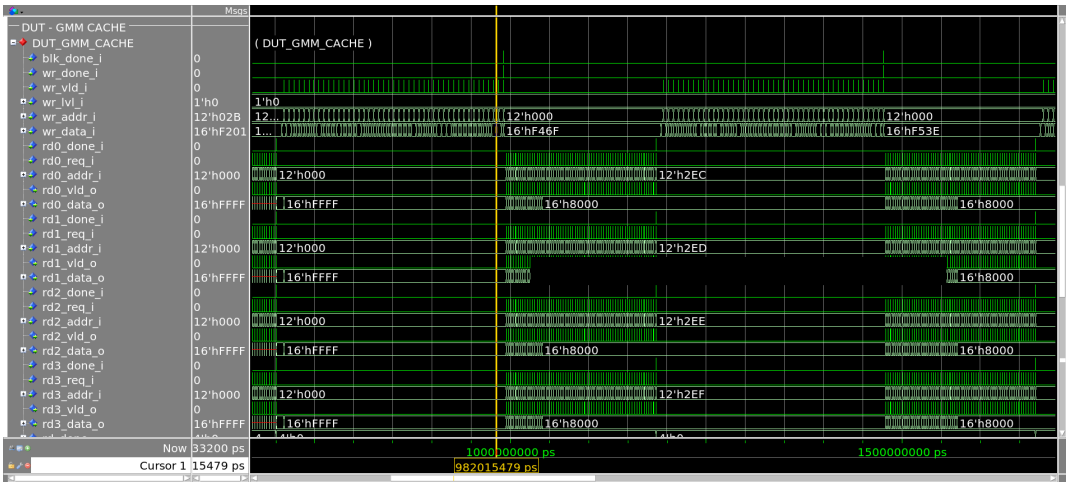


Figura 3.18: Simulación por eventos de la memoria caché de probabilidades.

bloque	**tamaño del bloque de memoria**	**tamaño del banco de memoria**
memoria caché L1	128x32	40 KBytes
memoria caché L2	128x32	20 KBytes
memoria caché árbol léxico	256x32	4 KBytes
memoria caché HMM	64x32	4 KBytes

Tabla 3.13: Bloques de memoria utilizados en la implementación de los módulos GMM y HMM.

bloque de los núcleos de procesamiento.

Ser realizaron estimaciones del consumo de potencia de los bloques de procesamiento utilizando la herramienta Voltus IC Power Integrity. Esta herramienta permite realizar estimaciones de potencia en base a vectores de testeo aplicados a los puertos de entrada del diseño. En este caso, estos vectores fueron creados a partir de simulaciones por eventos realizadas para el caso típico de reconocimiento de dígitos. Se llevaron a cabo análisis de consumo de potencia de todos los bloques implementados.

El consumo de potencia se clasifica en tres tipos, dependiendo de como

bloque	**Area** $[mm^2]$
unidad ordenamiento	0.28
unidad artimética	0.84
banco de memoria caché L2	0.9
banco de memoria caché L1	3.74
memoria caché	2.32
módulo GMM	4.23

Tabla 3.14: Área de silicio que ocupa el módulo GMM y los sub-bloques que lo componen.

bloque	**Área** $[mm^2]$
memoria caché de probabilidades	1.78

Tabla 3.15: Área de silicio que ocupa la memoria caché de probabilidades.

bloque	**Área** $[mm^2]$
unidad ordenamiento	0.75
FIFO ordenamiento	0.6
FIFO hipótesis	0.6
FIFO salida	0.22
memoria caché	0.46.
memoria caché árbol lexico	0.47
módulo HMM	3.7

Tabla 3.16: Área de silicio que ocupa el módulo HMM y los sub-bloques que lo componen.

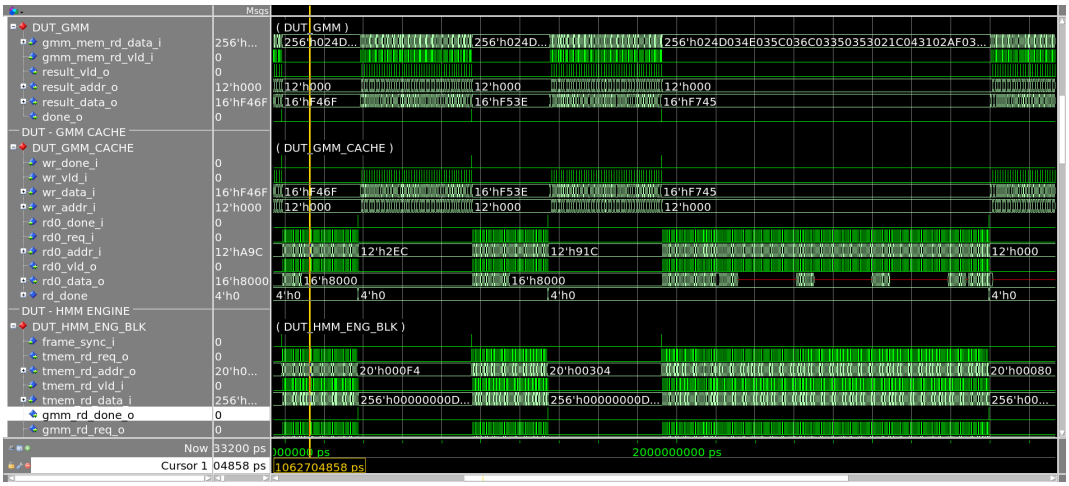


Figura 3.19: Simulación por eventos de los módulos operando en conjunto.

ésta se consume en los circuitos integrados:

- potencia por transiciones: representa el consumo debido a la carga y descarga de las capacidades de interconexión entre celdas.

- potencia interna: representa el consumo debido a la carga y descarga de capacidades de interconexión y de los dispositivos dentro de una celda. Este consumo es dependiente del estado lógico de los puertos de entrada y de la carga del puerto de salida.

- potencia de pérdida: representa el consumo de los dispositivos cuando no están realizando una transición. Este consumo también es dependiente del estado de los puertos de entrada.

Las Tablas 3.17, 3.18, 3.19 y 3.20 muestra el consumo de la interfaz acústica, del módulo GMM, la memoria caché de probabilidades y HMM respectivamente, junto con el de cada uno de sus sub-bloques implementados. Se

bloque	**Potencia por Transiciones [mW]**	**Potencia Interna [mW]**	**Potencia por pérdida [mW]**	**Potencia Total [mW]**
Removedor DC	16.85	6.56	1.44	24.82
Filtro pre-enfásis y alineador de marcos	16.82	7.29	1.40	25.52
FFT	29.26	5.22	0.34	34.82
scala de mel, log y deltas	10.44	3.02	0.50	13.97
interfaz acústica	97.95	31.35	4.79	123.25

Tabla 3.17: Consumo de potencia de la interfaz acústica.

consideró una frecuencia y voltaje de operación de 50 MHz y 1.08 V respectivamente.

bloque	**Potencia por Transiciones [mW]**	**Potencia Interna [mW]**	**Potencia por pérdida [mW]**	**Potencia Total [mW]**
unidad ordenamiento	8.26	2.89	0.52	11.68
unidad artimética	20.02	12.48	1.05	33.57
banco de memoria caché L2	13.71	32.92	0.10	46.74
banco de memoria caché L1	81.25	9.70	0.19	91.15
memoria caché	28.28	66.56	0.36	95.24
módulo GMM	140.45	95.74	2.34	238.55

Tabla 3.18: Consumo de potencia del bloque GMM.

bloque	**Potencia por Transiciones [mW]**	**Potencia Interna [mW]**	**Potencia por pérdida [mW]**	**Potencia Total [mW]**
memoria caché de probabilidades	10.35	5.38	0.82	16.51

Tabla 3.19: Consumo de potencia de la memoria caché de probabilidades.

bloque	**Potencia por Transiciones [mW]**	**Potencia Interna [mW]**	**Potencia por pérdida [mW]**	**Potencia Total [mW]**
unidad ordenamiento	22.77	7.93	1.71	32.42
FIFO ordenamiento	21.27	10.64	1.13	33.04
FIFO hipótesis	17.79	9.72	1.14	28.66
FIFO salida	5.47	3.19	0.42	9.08
memoria caché	0.68	0.32	0.1	1.09
memoria caché árbol lexico	0.34	0.25	0.08	0.67
módulo HMM	72.77	35.41	4.94	113.14

Tabla 3.20: Consumo de potencia del bloque HMM.

3.3.3. Implementación en 16 nm

Se implementó la arquitectura del módulo GMM en un proceso VLSI FinFet de 16 nm. Debido a inconvenientes en los archivos del kit de diseño hubo varias reglas de diseño que eran violadas por el ruteador. Este problema se acentuaba a medida que se incrementaba la densidad de compuertas instanciadas. Por lo tanto, se implementó una versión reducida del módulo GMM con una memoria caché cuya capacidad es de 1.2KBytes. Esta memoria se implementó con registros ya que no se contaba con memorias SRAM para esta tecnología. El diseño opera a una frecuencia máxima de operación de 200 MHz según los reportes de temporizado de la herramienta Innovus.

El diseño cuenta con una interfaz serial tanto para configurar el módulo, enviar datos a la memoria caché y al registro de entrada correspondiente al vector acústico CCFM. Para leer los datos de salida del núcleo de procesamiento se utiliza nuevamente una interfaz serial.

Se tenía un área total disponible de 0.5 mm x 1.7 mm siendo el área de utilización de 11 %.

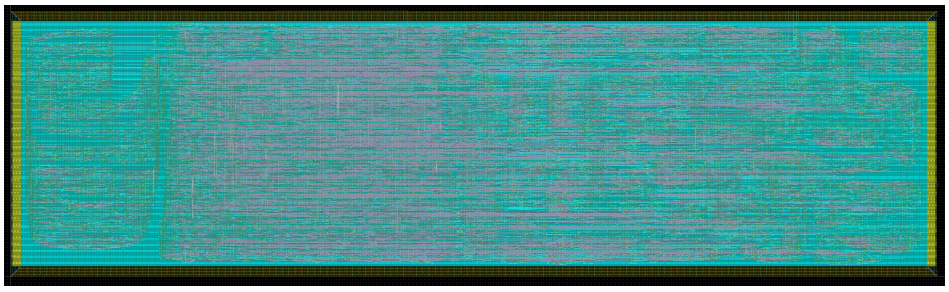


Figura 3.20: Implementación física del bloque GMM en proceso FinFet 16nm.

Al igual que en la implementación en la tecnología de 55nm, se realizó una estimación del consumo de potencia del módulo GMM. La Tabla 3.21 detalla

bloque	Potencia por Transiciones [mW]	Potencia Interna [mW]	Potencia por pérdida [mW]	Potencia Total [mW]	Frec. [MHz]
GMM	37.37	50.55	0.15	88.07	100
GMM	25.12	31.47	0.15	56.74	50

Tabla 3.21: Consumo de potencia de la memoria caché de probabilidades.

el consumo para dicho módulo. Notar que a diferencia del caso anterior, no se discrimina consumo por cada bloque. Esto es así porque se adoptó otra metodología de implementación, en la cual no se utilizó jerarquía en el diseño mientras que en el caso anterior si. Para las estimaciones, se consideró un voltaje de operación de 0.72 V y una frecuencia de 50MHz y 100 MHz.

En comparación con el diseño anterior se obtiene un consumo de potencia reducido para la misma frecuencia de operación. El hecho de que el valor en tensión de operación es menor respecto al caso anterior es uno de los factores que impactan en el consumo. Sin embargo, recordar que la memoria caché es una versión significativamente reducida lo cual también impacta drásticamente en el consumo. Debido a esto último, no resulta sencillo realizar una comparación entre los dos cuadros de estimaciones.

3.4. Conclusiones

En esta sección se describió el diseño completo para el reconocimiento de voz automático. Este diseño contempla la interfaz acústica, la unidad de Gaussianas mixtas y la unidad de cadenas de Markov ocultas. Estos módulos se integraron en la arquitectura del chip de multiprocesadores descripta en

la Sección 3.1.1.

De modo tal de mitigar el impacto negativo de la latencia de le memoria principal en las transferencias de datos se introdujo una jerarquía de memoria. Dicha jerarquía está compuesta por memorias caché de primer nivel que se comunican con la memoria principal a través de las interfaces de red diseñadas para tal propósito. El diseñõ de las mismas comprende la elección del nivel de asociatividad adecuado, el tamaño del bloque y la capacidad requerida. Aspectos que han sido abordados a lo largo del capítulo.

Se diseñaron interfaces con las redes N1 y N2, de modo que la interfaz acústica como la unidad de Gaussianas mixtas pueda tomar datos de cualquiera de las dos redes lo cual permite una mayor flexibilidad. Por otro lado, el módulo HMM puede escribir el dato de salida en la memoria principal o puede enviarlo a otra unidad de procesamiento en el arreglo de multiprocesadores. Otro ejemplo de la flexibilidad mencionada es la unidad de interfaz acústica, la cual puede ser utilizada por múltiples unidades de procesamiento. Por ejemplo, el vector de coeficientes de frecuencias cepstrales puede ser la entrada a una red neuronal profunda implementada con las unidades que realizan la multiplicación vector-vector [29]. La configuración de las tres unidades está a cargo del procesador CORTEXM0

Se implementaron los diseños descriptos en un proceso VLSI 55nm de Globalfoundries. Se implementó también una versión escalada del módulo GMM en un proceso VLSI de 16nm con transistores del tipo FinFet de TSMC. Se describen las implementaciones en término de frecuencia y voltaje de operación, área de silicio y estimaciones de consumo de potencia utilizando la herramienta de layout físico con vectores de testeo generados a partir de

simulaciones por eventos.

Capítulo 4

Análisis de Paralelismo

4.1. Introducción

El paralelismo es una estrategia para reducir el consumo de energía de los bloques de procesamiento digitales. Al incluir paralelismo en el procesamiento es posible reducir la frecuencia de operación manteniendo la tasa de procesamiento. Esta reducción impacta considerablemente en el consumo. En este capítulo se presenta el análisis de paralelismo y de eficiencia energética realizado sobre la unidad de procesamiento GMM. Primero, se analiza el paralelismo a nivel micro-arquitectura. Esto es, como afecta la asociatividad y la tasa de aciertos de la memoria caché en el paralelismo y la aceleración obtenida al paralelizar los datos de la memoria local de dos maneras diferentes. Segundo, se analiza la eficiencia energética a nivel de CMP, es decir cuanto más eficiente en consumo de energía puede ser una arquitectura de múltiples procesadores al incluir unidades de procesamiento en paralelo teniendo en cuenta la comunicación con la memoria principal. Como se verá

a lo largo del capítulo la comunicación con la memoria principal juega un rol importante en el tiempo de procesamiento y por ende en el consumo de energía.

4.2. Paralelismo a nivel micro-arquitectura

Dado que la latencia de la memoria caché son 8 ciclos de reloj es posible paralelizar el cómputo de las distribuciones Gaussianas instanciando múltiples unidades aritméticas. De esta forma se puede mitigar el efecto de la latencia de la memoria caché y maximizar la tasa de procesamiento. Esto es verdadero siempre y cuando los parámetros necesarios para el cómputo se encuentren en memoria local. Caso contrario, como se explica más adelante, la paralelización no presenta ventajas.

A continuación se analizan las ventajas y deventajas de utilizar memorias caché de uno o más niveles de asociatividad. Luego, se analizan dos modificaciones a nivel micro-arquitectura que permiten paralelizar el cómputo de las distribuciones. La eficiencia en el cómputo de dichas modificaciones se evalúa para dos casos: (1) cuando todos los parámetros requeridos se encuentran almacenados localmente; y (2) cuando algunos de ellos deben ser buscados en memoria principal.

La Fig. 4.1 muestra la tasa de desaciertos de una memoria caché de 80KB en función del porcentaje de datos sobre la cantidad total de datos que requieren todas las Gaussianas a computar, dos niveles de asociatividad y una sola unidad aritmética operando. El tamaño del bloque de la memoria es de 1280 bits, que es la cantidad de bits necesarios para una función, por lo que

la memoria permite almacenar 512 Gaussianas. Como es de esperar, cuando es posible almacenar todos los parámetros la tasa de desaciertos es nula. A medida que se incrementa la cantidad de funciones a evaluar se obtiene una tasa de desaciertos mayor. En el caso ideal todos los parámetros de las funciones a evaluar se almacenan en la memoria local y no resulta necesario ir a buscar datos a memoria principal. Sin embargo esta estrategia no resulta escalable dado que al incrementar el número de funciones a evaluar también lo hace el tamaño de la memoria que se requiere. Es posible observar que si sólo se puede almacenar el 50 % de los parámetros, la tasa de desaciertos es del 100 % por lo que el rendimiento del procesador se ve afectado seriamente por la latencia de la memoria principal. Además si la memoria cache tiene nivel de asociatividad 2 al comparar la dos figuras se observa que para la misma relación de capacidad de almacenamiento, la tasa de desaciertos que se obtiene resulta mayor o igual.

Por otro lado las Figs. 4.2 y 4.3 muestran gráficamente la razón por la cual resulta más beneficioso, en términos de tasa de desaciertos, utilizar el mapeo directo en vez de un nivel de asociatividad mayor. En dichas figuras se consideró una politica de reemplazo llamada *least used* en la bibliografía. Los rectangulos representan los bloques memoria que permiten almacenar dos tercios de los datos requeridos y las letras A, B, C representan los tercios del total de datos. En el eje horizontal se representa el tiempo de procesamiento y en el eje vertical los marcos procesados. Las búsqueda de datos en memoria caché para el procesamiento del primer marco, resulta en desaciertos ya que la memoria está inicialmente vacía; estos son los desaciertos obligatorios [15]. Al

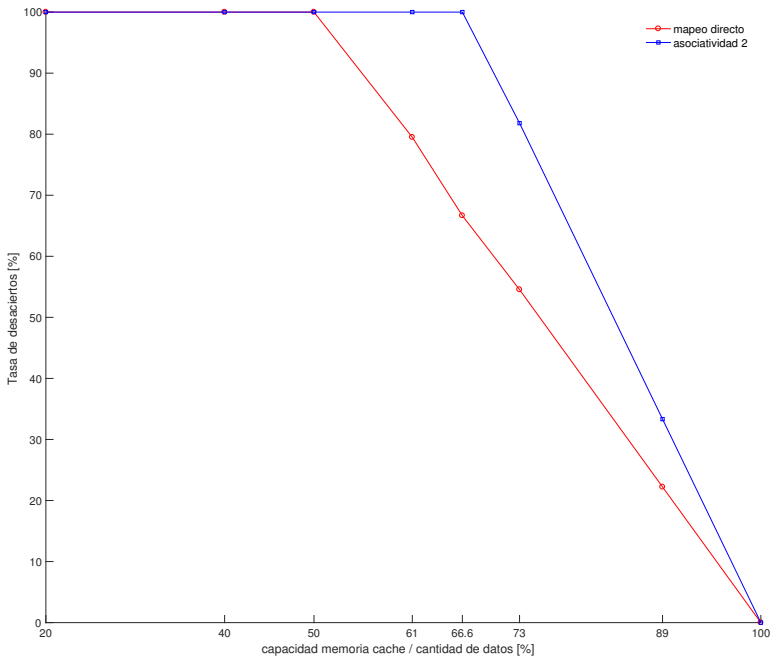
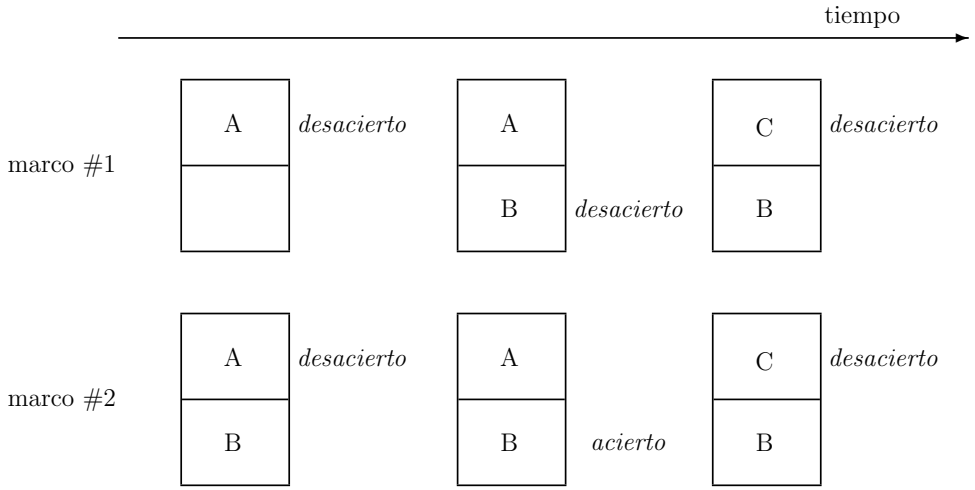


Figura 4.1: Tasa de desaciertos de la memoria caché vs. porcentaje de datos que se pueden almacenar localmente.

pedir el primer tercio A, estos se buscan en memoria principal y se almacenan en la memoria local de acuerdo a un mapeo directo de las direcciones de memoria. Se realiza lo mismo para el tercio B y C. Luego, al comenzar el procesamiento del segundo marco, el tercio B y C se encuentra en memorial local y sólo el tercio B puede ser reutilizado. Por otro lado, en el caso de la Fig. 4.3 la memoria está compuesta de dos bloques que componen los dos niveles de asociatividad. En el procesamiento del segundo marco no es posible reutilizar ningún dato debido a la política de reemplazo utilizada. El mismo análisis se puede realizar para niveles de asociatividad mayor.

La principal causa de desaciertos en el procesamiento es el patrón de acceso a memoria que resulta fijo. Una mejora substancial en la tasa de desaciertos resulta de invertir el orden del acceso a memoria, es decir realizar



tasa de desaciertos = 66.66 %

(a)

Figura 4.2: Almacenamiento de las funciones para una memoria caché de mapeo directo con capacidad igual a dos tercios del total.

el acceso a memoria comenzando por las direcciones de mayor valor hacia las de menor. De esta manera, se reducen los pedidos a memoria principal lo cual impacta de manera positiva en el consumo de energía y la tasa de procesamiento. Realizando dicho cambio en la lógica de control se obtienen las tasas de desaciertos que se muestran en la Fig. 4.4.

4.2.1. Micro-arquitectura de caché compartido

Teniendo en cuenta el análisis previo se propone una arquitectura que cuenta con capacidad de cómputo en paralelo y una memoria caché de mapeo directo. La arquitectura propuesta se muestra en la Fig. 4.5. Como se puede observar se instancian varias unidades aritméticas. De modo tal de considerar múltiples pedidos de lectura a la memoria caché y disponer de un control

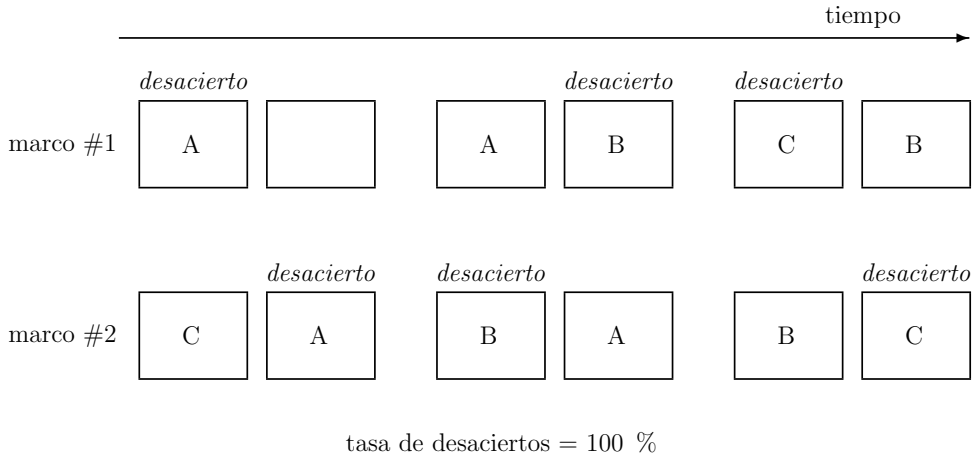


Figura 4.3: Almacenamiento de las funciones para una memoria caché de asociatividad 2 con capacidad igual a dos tercios del total.

de la cantidad de Gaussianas computadas es necesario también replicar las unidades de control aritméticas.

Si bien la memoria caché dispone de múltiples puertos el acceso a los bloques de memoria se realiza a través de un único puerto. Además se considera sólo un puerto de comunicación con la memoria principal. Si múltiples pedidos a memoria resultan en desaciertos, entonces se deben realizar los correspondientes pedidos de datos a memoria principal en forma secuencial. La lógica de control de la memoria caché tiene en cuenta esta nueva característica. Los pedidos de lectura de cada puerto se atienden en forma secuencial utilizando el campo etiqueta de la dirección de memoria presentada en cada puerto de acceso. La memoria cuenta con un bus de salida al cual se conectan las unidades aritméticas.

La arquitectura de la memoria cuenta con un pipeline de modo de acelerar la tasa de entrega de datos. La lógica de control utiliza punteros a los puertos tanto para la operación de lectura como de escritura, en el caso que haya

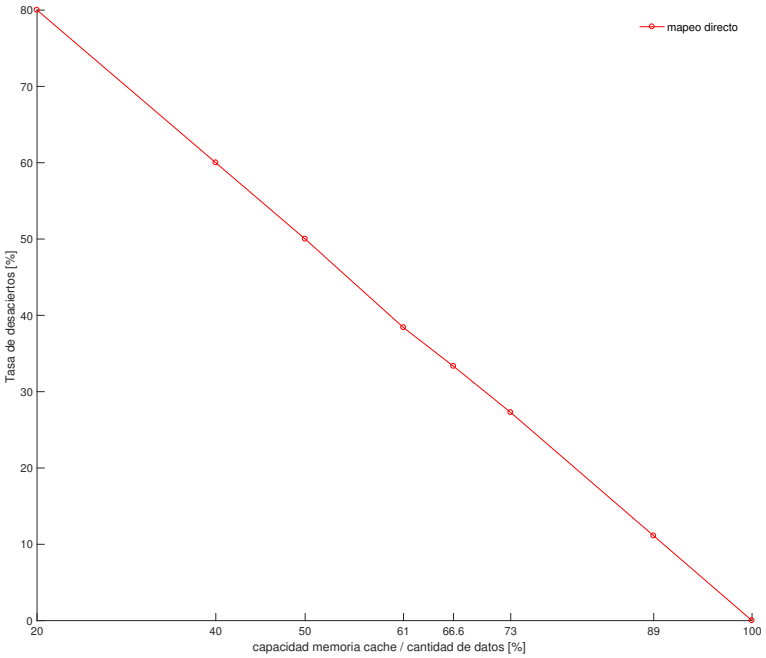


Figura 4.4: Tasa de desaciertos de la memoria caché vs. porcentaje de datos que se pueden almacenar localmente con inversión del orden de accesos a memoria.

habido un desacierto. La Fig. 4.6 muestra la arquitectura de la memoria caché y la operación del pipeline.

La etapa de búsqueda se refiere a determinar si el dato se encuentra almacenado en forma local. La etapa de búsqueda a memoria principal comprende además el almacenamiento del dato en la memoria local. Por lo tanto estas dos etapas compiten por el mismo recurso. Como se puede observar se coloca una etapa de espera para evitar el solapamiento de estas dos etapas del pipeline.

De forma de evaluar los beneficios de la arquitectura propuesta se recurre a la ecuación que modela la aceleración [36]:

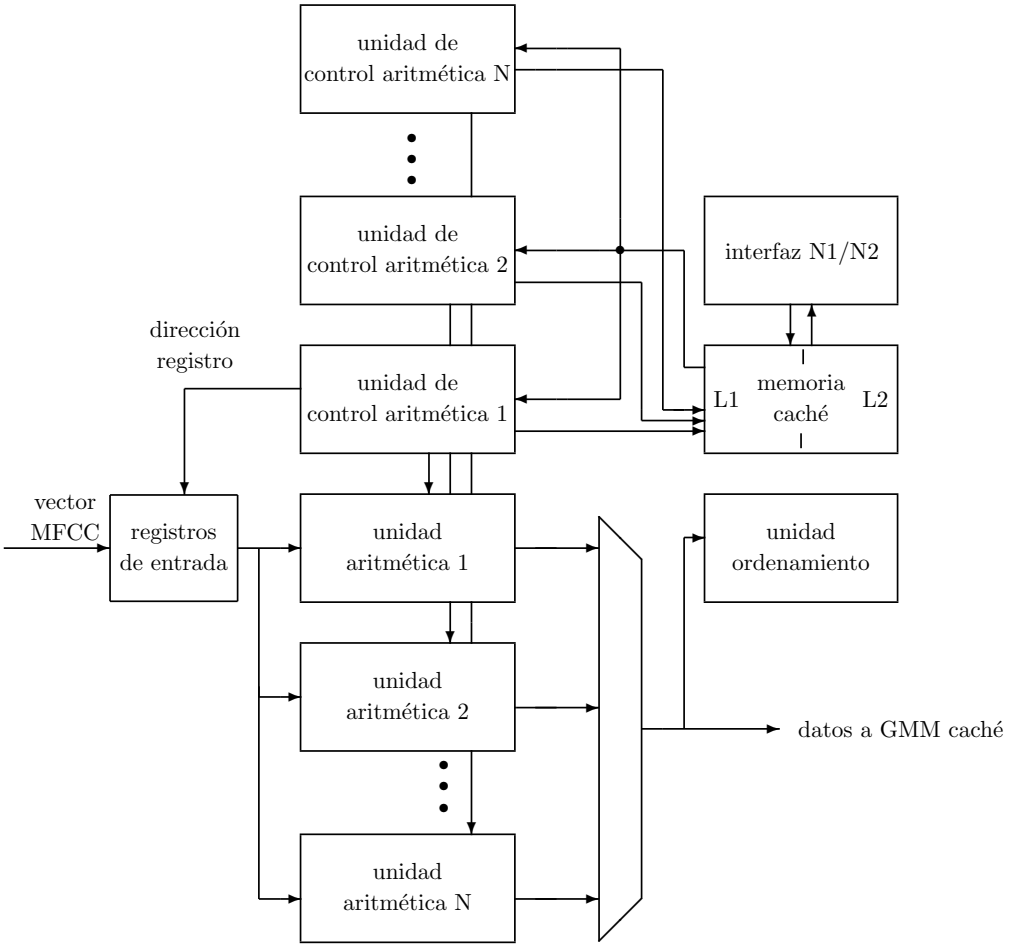


Figura 4.5: Diagrama en bloque del módulo GMM optimizado con una memoria caché con múltiples puertos.

$$Aceleracion = \frac{T_{viejo}}{T_{nuevo}} \tag{4.1}$$

donde T_{viejo} es el tiempo de procesamiento con la arquitectura serie y T_{nuevo} es el tiempo de procesamiento de la arquitectura optimizada que cuenta con cierto nivel de paralelismo en aquella porción del algoritmo cuyo cómputo permite ser paralelizado.

Si se denota F_{serie} la fracción del algoritmo que se computa de manera

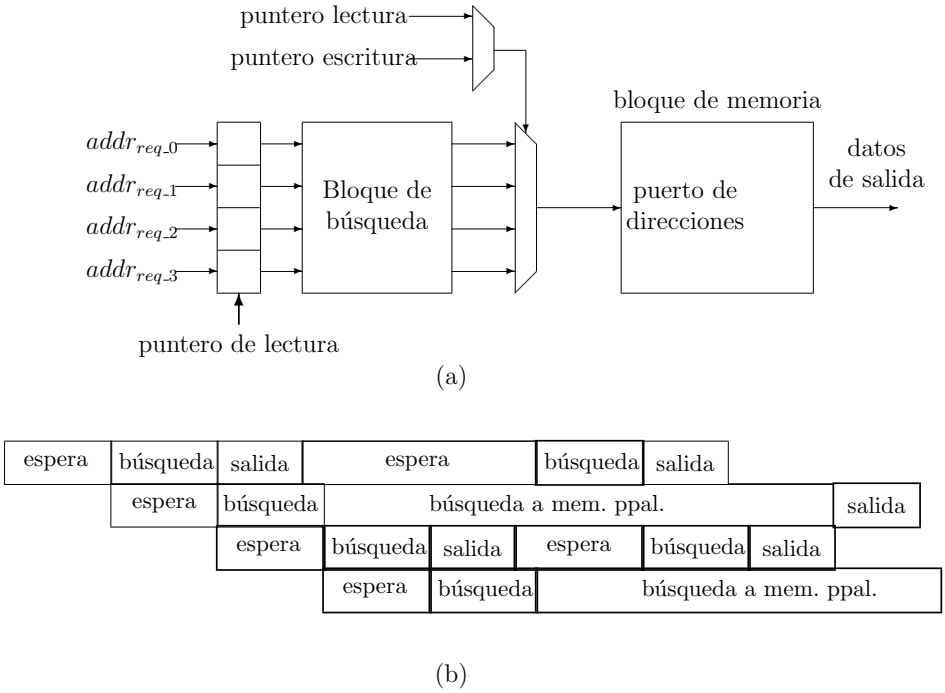


Figura 4.6: (a) Diagrama en bloque de la memoria caché con múltiples puertos de lectura. (b) Operación del pipeline.

serial y $F_{paralelo}$ la porción del algoritmo cuyo cómputo puede ser paralelizado y N_{proc} la cantidad de procesadores en paralelo que se instancian, la ecuación de la aceleración también se puede expresar como

$$Aceleración = \frac{1}{F_{serie} + \frac{F_{paralelo}}{N_{proc}}} \tag{4.2}$$

La fracción F_{serie} y $F_{paralelo}$ se definen como:

$$F_{serie} = \frac{Num.\ de\ instrucciones\ serie}{Num.\ de\ instrucciones\ total} \tag{4.3}$$

$$F_{paralelo} = \frac{Num.\ de\ instrucciones\ paralelo}{Num.\ de\ instrucciones\ total} \tag{4.4}$$

Tabla 4.1: Tabla de instrucciones que computa la arquitectura.

Instrucción	**cantidad de ciclos**	**Q**
$mfcc\_load$	40	1
$cache\_fecth$	$\propto N_{proc}$	$\propto$ tasa de aciertos
mem_fetch	500	$\propto$ tasa de desaciertos
gmm_proc	8	N_{gmm}

Es posible pensar las operaciones que realiza la arquitectura en términos de instrucciones. La Tabla 4.1 detalla las instrucciones que la arquitectura computa, cuantos ciclos requiere cada una de ellas y cuantas veces se computan en el algoritmo. La cantidad de veces que se computan las instrucciones referidas a búsqueda a memoria es proporcional a la tasa de aciertos en la memoria caché. Por otra parte, la cantidad de veces que se computa la instrucción gmm_proc es equivalente a la cantidad de Gaussianas que se pretende evaluar N_{gmm}.

Entonces Q es la cantidad de veces que se computa dichas instrucciones y Q_{total} es la cantidad de instrucciones, por lo tanto:

$$F_{serie} = \frac{Q_{mfcc\_load} + Q_{mem\_fetch}}{Q_{total}} \tag{4.5}$$

$$F_{paralelo} = \frac{Q_{cache\_fetch} + Q_{gmm\_proc}}{Q_{total}} \tag{4.6}$$

donde Q_{total} es igual a:

$$Q_{total} = Q_{mfcc\_load} + Q_{cache\_fetch} + Q_{mem\_fetch} + Q_{gmm\_proc} \tag{4.7}$$

Entonces F_{serie} y $F_{paralelo}$ depende de la cantidad de datos que se encuentren almacenados en memoria local ya que la búsqueda a memoria principal

se realiza de manera secuencial. Por lo tanto estas fracciones varían con el tamaño de la memoria caché. Es de esperar que así lo haga la aceleración.

Notar que la cantidad de ciclos que toma la instrucción $cache\_fectch$ depende de la cantidad de unidades de procesamiento N_{proc}. Un modelo del tiempo de procesamiento permite observar con mayor detalle el impacto en la aceleración que posee la tasa de desaciertos y la cantidad de procesadores en paralelo. El tiempo de procesamiento de la arquitectura sin paralelizar se modela como:

$$T_{viejo} = T_{mfcc} + T_{pipeline} + N_{gmm} T_{proc\_pipeline} \tag{4.8}$$

donde T_{mfcc} es el tiempo requerido para cargar el vector de características acústicas, $T_{pipeline}$ es la latencia del pipeline y $T_{proc\_pipeline}$ es el tiempo que requiere el procesamiento de una Gaussiana en el pipeline siendo N_{gmm} la cantidad de gaussianas a computar. Este último temporizado es igual a la cantidad de ciclos que requiere obtener un dato de la memoria caché más los ciclos que inserta la lógica de control del pipeline. Entonces, (4.8) se puede reescribir como

$$T_{viejo} = T_{mfcc} + T_{pipeline} + N_{gmm} \times (T_{cache} + T_{control}) \tag{4.9}$$

donde T_{cache} es la latencia de la memoria caché. El costo en tiempo de las unidades de control aritméticas se modela con el término $T_{control}$. Si todos los parámetros se encuentren almacenados en memoria local la ecuación que modela el tiempo de procesamiento de la arquitectura con paralelismo es:

$$T_{nuevo} = T_{mfcc} + T_{pipeline} + \frac{N_{gmm}}{N_{proc}} \times (T_{cache} + T_{control}) \tag{4.10}$$

Esta ecuación se cumple cuando la lógica de control de la memoria caché es menor o igual a $T_{control}$. A medida que se incrementa el número de procesadores, la latencia de la memoria también lo hace debido al costo en tiempo de su lógica de control. Recordar que dicha lógica permite la utilización de la memoria con múltiples puertos. Bajo dichas consideraciones el tiempo de procesamiento se modela como:

$$T_{nuevo} = T_{mfcc} + T_{pipeline} + \frac{N_{gmm}}{N_{proc}} \times T_{cache} \tag{4.11}$$

Es decir que el costo en tiempo de las unidades de control aritméticas pasa a ser transparente para el tiempo de procesamiento de una función Gaussiana.

La Tabla 4.2 muestra el valor de T_{cache} y $T_{control}$ para distintos valores de N_{proc}. Como se puede observar, para la arquitectura sin optimización la latencia de la memoria caché es $T_{cache} = 6$ ciclos y $T_{control} = 2$ ciclos. Cuando el número de unidades es mayor o igual a 8, el término T_{cache} se modela como $2 \times N_{proc}$. Es decir que la latencia de la memoria caché queda gobernada por la lógica de control que permite la operación con múltiples puertos.

Reescribiendo (4.11), la expresión del tiempo de procesamiento para $N_{proc} > 8$ es

$$T_{nuevo} = T_{mfcc} + T_{pipeline} + 2 \times N_{gmm} \tag{4.12}$$

N_{proc}	T_{cache}	$T_{control}$
1	6	2
2	7	2
4	8	2
8	16	2
16	32	2
32	64	2

Tabla 4.2: Tabla de los parámetros del modelo para el tiempo de procesamiento.

Esta ecuación muestra que la aceleración llega a su máximo con 8 procesadores en paralelo.

Hasta ahora se analizó el tiempo de procesamiento considerando que todos los parámetros se encuentran en la memoria local. Si esto no es así, se deben buscar los datos de la memoria principal. El modelo del tiempo de procesamiento para este caso se describe como:

$$T_{viejo} = T_{mfcc} + T_{pipeline} + N^1_{gmm}(T_{cache} + T_{control}) + N^2_{gmm}(T_{mem.ppal.}) \quad (4.13)$$

$$T_{nuevo} = T_{mfcc} + T_{pipeline} + \frac{N^1_{gmm}}{N_{proc}}(T_{cache} + T_{control}) + N^2_{gmm}(T_{mem.ppal.}) \quad (4.14)$$

donde N^1_{gmm} es el número de Gaussianas cuyos parámetros se encuentran en la memoria local y N^2_{gmm} de aquellas Gaussianas cuyos parámetros deben ir a buscarse a memoria principal. Debido a que esta búsqueda no puede ser realizada en paralelo y que la latencia $T_{mem.ppal.}$ es de al menos 500 ciclos de reloj (recordar que son 5 pedidos a memoria principal por cada Gaussiana,

mucho mayor que T_{cache}) no se logra una aceleración considerable.

En la Fig. 4.7 se puede observar la tasa de aceleración que se logra incorporando unidades de procesamiento en paralelo. Para esta gráfica se considera una latencia de memoria principal de 100 ciclos de reloj, una frecuencia de operación de la unidad de procesamiento de 100 Mhz y que los demás anillos de la red N1 no están realizando pedidos a memoria. La figura contempla la cantidad de procesadores en paralelo y la capacidad de la memoria caché. Los valores se obtuvieron de la simulación lógica de la implementación a nivel RTL de la arquitectura propuesta.

Si la capacidad de la memoria caché es menor al 100 % entonces la aceleración que se obtiene es igual o menor a 1,11 veces. Este valor resulta bajo porque el costo en tiempo de procesamiento al buscar los datos a memoria tiene mayor impacto que la ganancia obtenida de paralelizar el cómputo de aquellas gaussianas cuyos parámetros se encuentran en memoria local. Si la capacidad de la memoria es igual o mayor a la cantidad de datos requerida para computar todas las funciones, entonces se maximiza el paralelismo y se logra una aceleración máxima de 3,7 veces con 8 unidades de cómputo.

4.2.2. Micro-arquitectura de caché individual

La arquitectura propuesta anteriormente resulta adecuada cuando las unidades de procesamiento comparten datos, sin embargo para la aplicación aquí descripta esta situación no ocurre por lo que un mejor enfoque es una arquitectura de múltiples unidades de procesamiento donde cada una de ellas

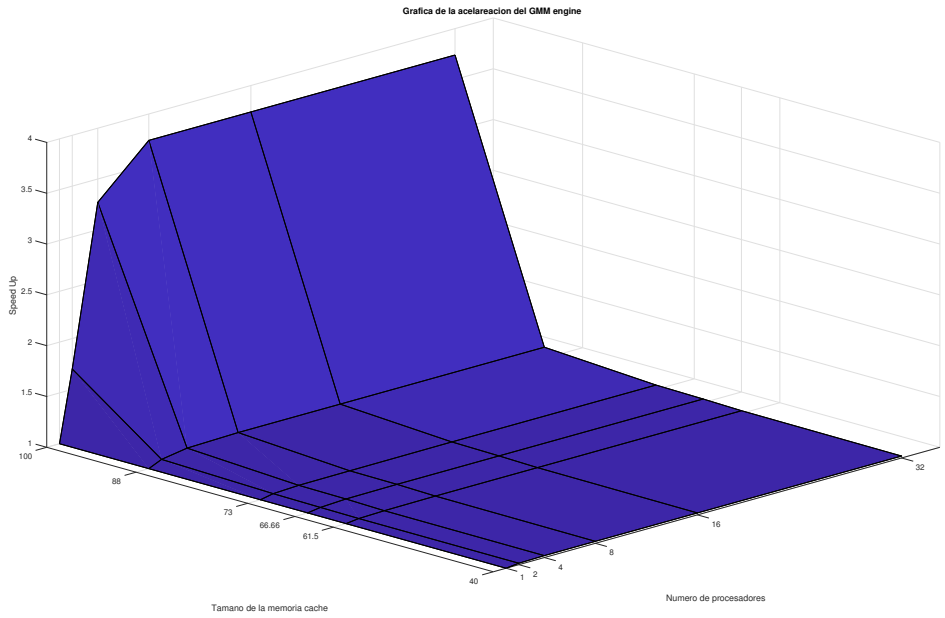


Figura 4.7: Arquitectura optimizada con caché compartido: aceleración del módulo GMM para distintos tamaños de memoria caché y número de unidades de procesamiento.

cuenta con una memoria caché. Dado que no hay datos compartidos entre las unidades no es necesario realizar ningún protocolo de coherencia entre las memorias caché [15] [37]. La Fig. 4.8 muestra la nueva arquitectura propuesta.

La Fig. 4.9 muestra la aceleración que se logra con esta arquitectura. La misma resulta ser mayor que la lograda con la primer arquitectura propuesta dado que se elimina el costo adicional de la lógica de control para tener en cuenta los múltiples puertos. Es posible observar que el máximo de aceleración es de 15,4 veces y se logra para 32 procesadores. Sin embargo, la aceleración que se logra cuando la capacidad de la memoria es menor a la cantidad de datos requerida es de 1,20 veces para 32 procesadores y una tasa de desaciertos del 11 %, siendo este el mejor de los casos analizados.

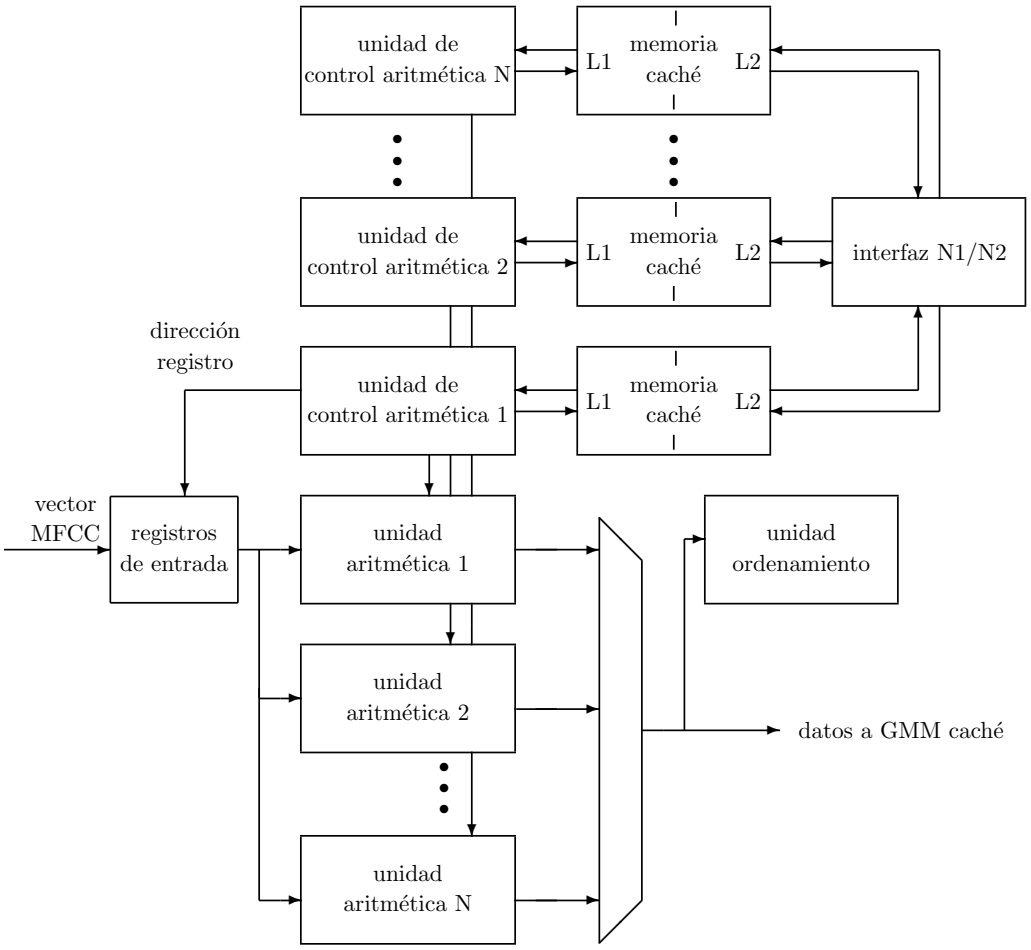


Figura 4.8: Diagrama en bloque del módulo GMM optimizado con múltiples memoria cachés.

La ganancia en aceleración aún continúa siendo baja. Cabe destacar que la aceleración satura en 32 procesadores.

La saturación en la aceleración, mencionada en el párrafo anterior, se debe a que no es posible paralelizar completamente el algoritmo dado que los datos que computa cada unidad aritmética deben ingresar a la unidad de ordenamiento. Disponer de una unidad de ordenamiento de múltiples puertos es muy costosa en término de área por lo que esa operación se realiza en forma

serial. De no ser así los valores de aceleración que se lograrían serían mayores.

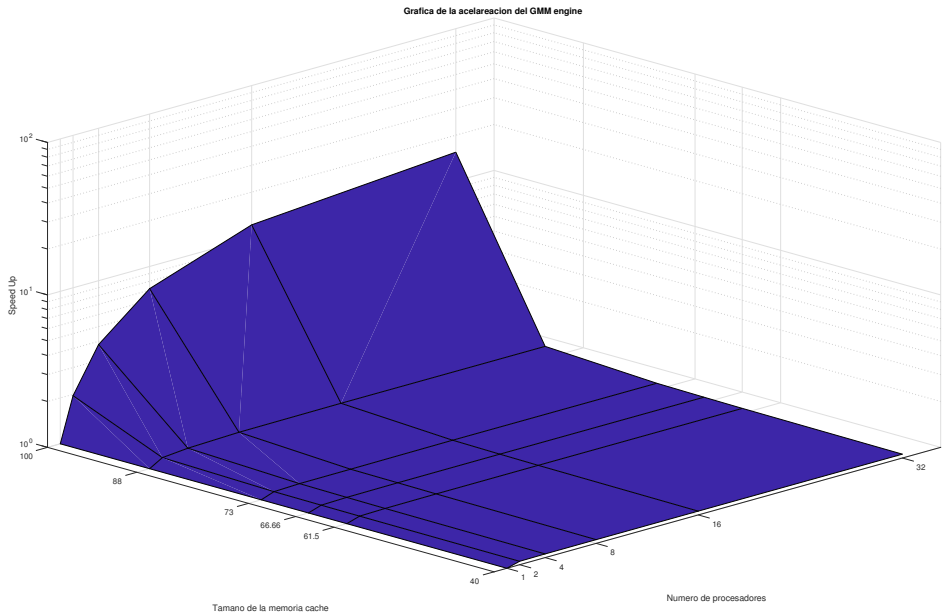


Figura 4.9: Arquitectura de caché individual: aceleración del módulo GMM para distintos tamaños de memoria caché y número de unidades de procesamiento.

4.2.3. Eficiencia energética

No sólo la aceleración en el tiempo de procesamiento se debe tener en cuenta al momento de seleccionar un nivel de paralelismo sino también el consumo de energía ya que el objetivo es utilizar el paralelismo como una estrategia para disminuir el mismo. En esta Sección se analiza la micro-arquitectura de caché individual teniendo en cuenta ambas variables.

Una métrica muy utilizada en la bibliografía es el producto entre la energía consumida y el tiempo de procesamiento [38]. Resulta muy útil porque en una misma figura es posible sopesar de manera equivalente ambas variables. Existen otras figuras de mérito como por ejemplo $T_{proc} \times E^2$ que le da más

preponderancia a la energía consumida. Es posible utilizar otras figuras simplemente incrementado la potencia de las variables.

En la Fig. 4.10 se puede observar que, debido a que el tiempo de procesamiento aumenta a medida que la capacidad de la memoria caché disminuye, el producto aumenta considerablemente y por tanto la arquitectura se vuelve menos eficiente. Además, como resulta natural, a medida que se instancian más unidades en paralelo el producto aumenta. Notar que en la energía no se consideró el consumo de la memoria local como tampoco la energía consumida por cada transacción de la memoria principal, lo cual hubiera arrojado un producto aún mayor.

Las estimaciones de potencia se obtuvieron de la síntesis lógica de la arquitectura para distintas frecuencias y voltajes de operación.

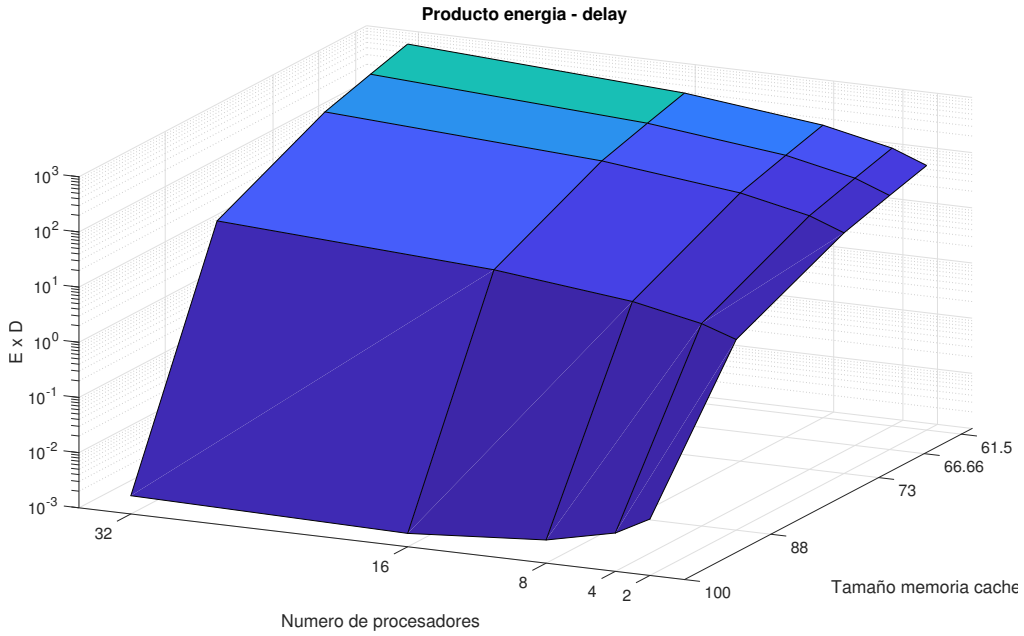


Figura 4.10: Producto $E \times T_{proc}$ para distintos tamaños de memoria caché y número de unidades de procesamiento para la arquitectura de caché individual.

Es interesante analizar por separado el caso en el que el tamaño de la memoria caché es suficiente para almacenar todos los parámetros requeridos. La Fig. 4.11 muestra el producto $T_{proc} \times E$ para distintos valores de tensión y frecuencia. De la misma se concluye que 16 procesadores es la configuración más eficiente. La frecuencia de operación es la máxima para cada voltaje de trabajo. Se observa que la reducción del voltaje no presenta ventaja en cuanto a eficiencia siendo el punto de trabajo óptimo 1.08V y 617Mhz.

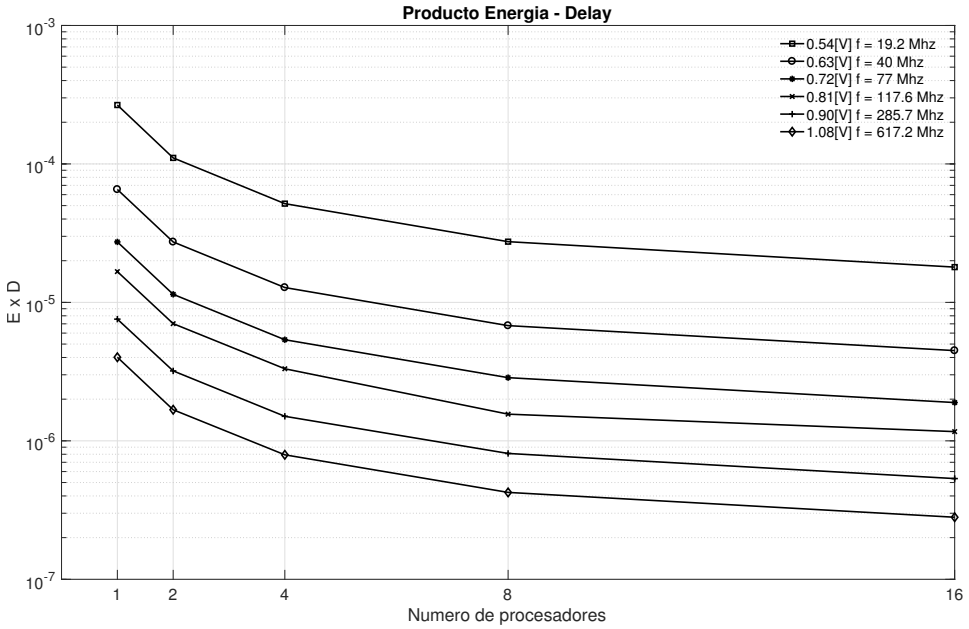


Figura 4.11: Producto $E \times T_{proc}$ para distintos número de unidades de procesamiento para la arquitectura de caché individual con tasa de desaciertos nula.

Resulta de interés analizar como estas figuras de mérito se comportan cuando la tasa de desaciertos no es nula. La Fig. 4.12 muestra el producto $E \times T_{proc}$ cuando la tasa de desaciertos es del 33.33 %. Para este análisis se consideró que la memoria principal opera a 100 MHz con una latencia de 100 ciclos. Se observa que agregar paralelismo a nivel micro-arquitectura no

resulta ventajoso desde el punto de vista energético. Por otra parte, al operar a frecuencias cada vez más bajas se obtienen arquitecturas más eficientes ya que el consumo de potencia disminuye. Sin embargo se observa que el punto óptimo de trabajo se encuentra para 0.63V y 40MHz. Por debajo de dicha tensión y frecuencia la arquitectura no resulta más eficiente.

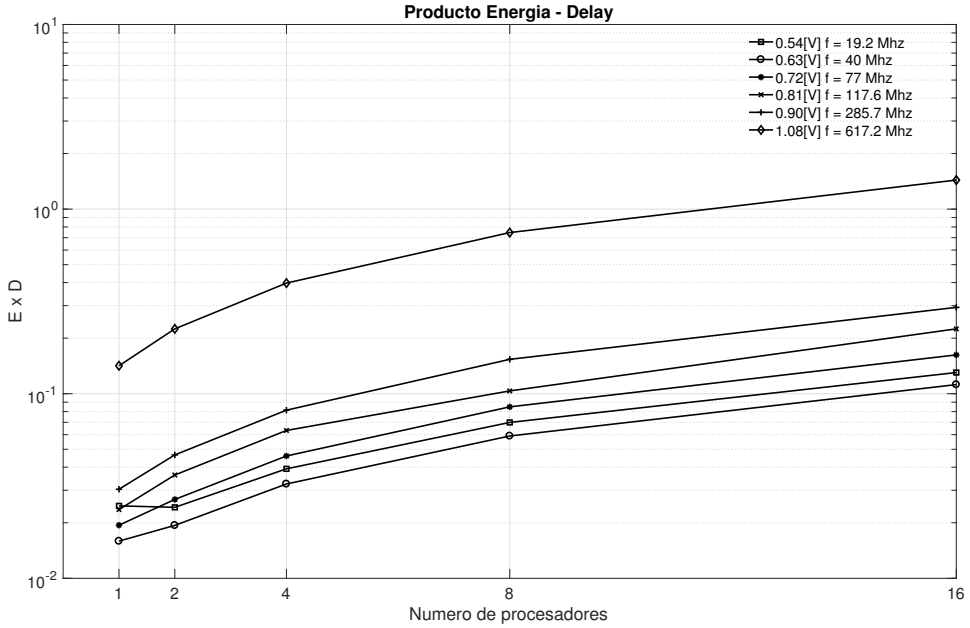


Figura 4.12: Producto $E^2 \times T_{proc}$ para distintos número de unidades de procesamiento para la arquitectura de caché individual con tasa de desaciertos del 33.33 %.

La Fig. 4.13 muestra el producto $E \times T_{proc}$ en función del voltaje operando los procesadores a máxima frecuencia para el número de procesadores que minimiza el producto. Podemos observar que la tecnología presenta un punto de operación óptimo dependiendo de si debe o no buscar datos a memoria principal.

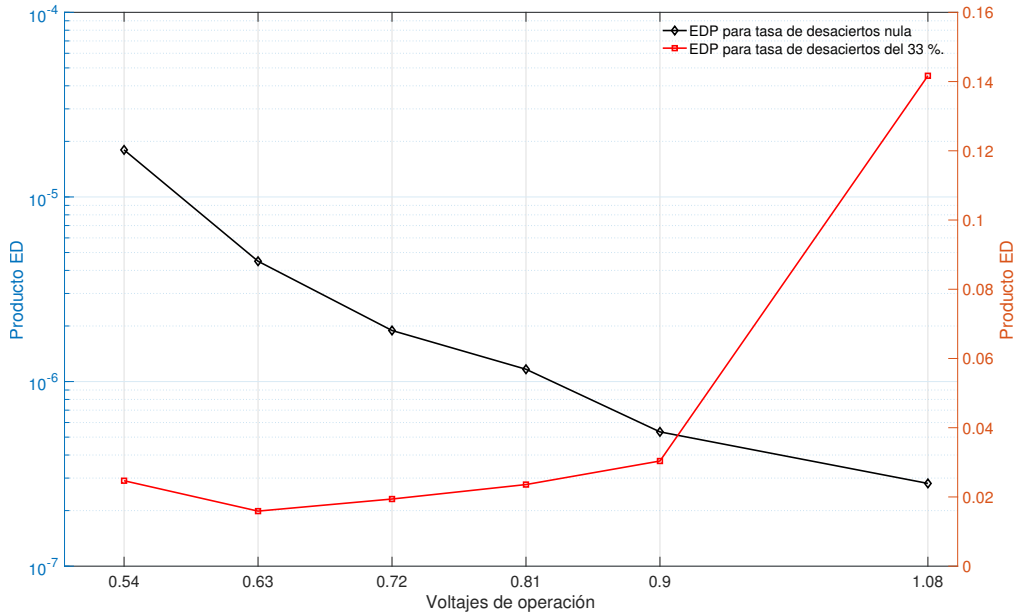


Figura 4.13: Producto $E^2 \times T_{proc}$ para distintos voltajes de operación para tasas de desaciertos nula y del 33 %.

4.2.4. Discusión

De este análisis se concluye que el paralelismo a nivel micro-arquitectura depende fuertemente de la velocidad de la memoria y de cuán eficiente es la jerarquía de memoria en su tarea de mitigar el impacto de la latencia de la memoria principal. En el caso aquí analizado se observa en primer lugar que al incorporar más unidades aritméticas la aceleración que se logra no es substancial si se deben buscar datos a memoria principal, incluso para tasas de desaciertos muy bajas. Por el contrario, de ser posible almacenar todos los parámetros localmente entonces la paralelización del cómputo adquiere sentido. En segundo lugar, se observa que debido a que el patrón de acceso a memoria es fijo, no resulta beneficioso incorporar un nivel de asociatividad mayor a 1 para la memoria caché. Esto resulta beneficioso desde el punto de

vista de consumo de energía y área de silicio ya que el diseño resulta menos complejo.

Una forma de poder explotar aún más el paralelismo es adicionar un tercer nivel más a la jerarquía de memoria en el CMP. Llamamos J0 el nivel más bajo en la jerarquía (dicho nivel es la actual memoria implementada), J1 el nivel intermedio de memoria y J2 la memoria principal. Entonces agregando el nivel J1 se reduce la latencia entre la memoria caché J0 y la memoria principal. Es decir que se reduce el costo en tiempo de un desacierto T_{cache}. El nuevo diseño debería responder a un evento de desacierto de una manera más eficiente. Por ejemplo, proveer de canales de comunicación entre los niveles J0 y J1 con suficiente ancho de banda para poder alimentar las unidades aritméticas en una menor cantidad de ciclos.

Otra alternativa es utilizar unidades de acceso a directo a memoria con estructuras que permitan realizar predicción en la carga de datos desde memoria principal. Mientras los procesadores están realizando el procesamiento de un conjunto de parámetros, las unidades de acceso directo se encargan de ir a buscar a memoria un nuevo conjunto de parámetros a procesar. En este diseño, la aceleración para tasas de desaciertos no nulas será mayor que la obtenida con sólo dos niveles de jerarquía. Utilizando el modelo descripto previamente para la arquitectura de caché compartido es posible modelar la aceleración para distintos valores de latencia de la memoria caché ante un desacierto. A modo de ejemplo, en la Fig. 4.14 se muestran los valores de aceleración para un valor de latencia de 20 ciclos de reloj. Como se puede observar, es posible obtener una aceleración de 1,6 veces para 4 unidades de procesamiento y una memoria con el 88 % de la capacidad que se requiere.

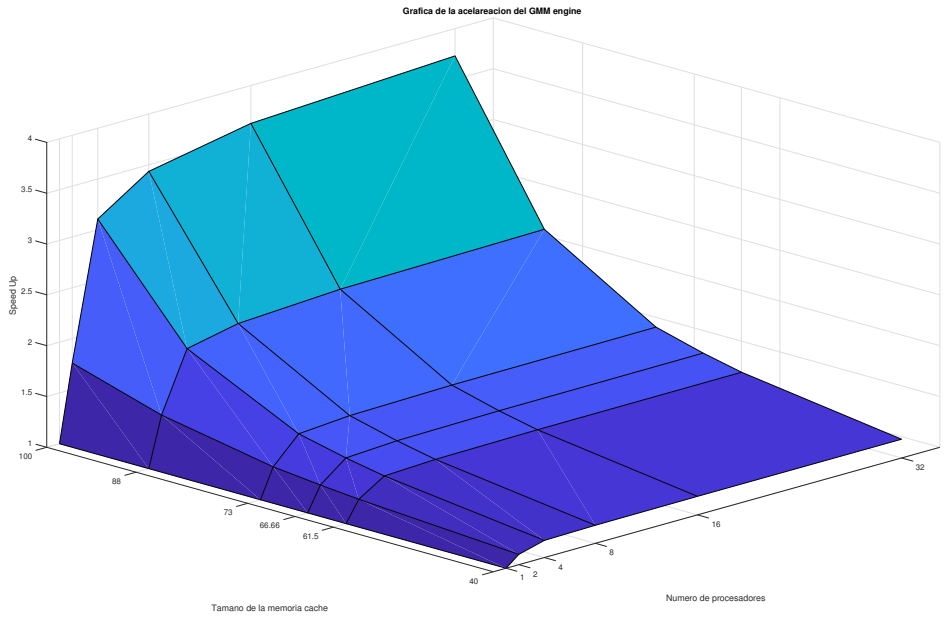


Figura 4.14: Aceleración del módulo GMM para distintos tamaños de memoria caché y número de unidades de procesamiento con un nivel adicional en la jerarquía de memoria.

Desde el punto de vista de la energía, el análisis realizado resalta la importancia de disponer de una tecnología que permita operar con voltajes bajos ya que abre un abanico de posibles puntos de trabajo. El número óptimo de unidades de procesamiento en paralelo es 16 para una tasa de desaciertos nula. Si se considera que no es posible almacenar todos los parámetros en la memoria local, el paralelismo a nivel micro-arquitectura no presenta ventaja, mientras que al operar las unidades de procesamiento en puntos de trabajo con voltajes y frecuencias reducidos se obtienen configuraciones más eficientes. A esto último se debe agregar que, del análisis realizado se observa que reducir el voltaje de operación no siempre resulta beneficioso en términos de la métrica utilizada. En la Fig. 4.13, la arquitectura analizada presenta un único punto óptimo de trabajo en 0.63V, 40 MHz que no coincide con la

menor tensión de trabajo disponible en la tecnología que es 0.54V.

4.3. Paralelismo a nivel de CMP

El diseño de múltiples procesadores en chip se inicia a raíz de la limitación impuesta por la disipación de potencia en procesadores de propósito general. A medida que la tecnología de integración reducía el tamaño de los transistores, la densidad de potencia de consumo se mantenía constante para un chip de tamaño fijo operando a máxima frecuencia [39]. Esto era así porque el voltaje escalaba en la misma proporción que la frecuencia de operación y la capacidad de los transistores. Sin embargo, a partir del advenimiento de tecnologías de 90nm e inferiores no fue posible mantener el escalamiento en el voltaje debido a las pérdidas de energía por fugas en el dieléctrico del transistor. Con el uso de arreglos de múltiples procesadores fue posible operar a una frecuencia menor manteniendo un crecimiento en el rendimiento computacional a expensas de mayor área de silicio. Sin embargo esta estrategia tiene sus propias problemáticas. Por ejemplo, las variaciones en la manufactura hace más díficil la sincronización de líneas de procesamiento en paralelo, la dificultad de implementar memorias SRAM que operen a bajo voltaje y resulten estables [40] y el incremento en el consumo por comunicación debido a mayores distancias de interconexión entre procesadores y la memoria principal, entre otras.

En este trabajo el foco del diseño no está puesto en conseguir máximo rendimiento computacional sino un arreglo de procesadores que minimice la energía y el tiempo de cómputo a la vez y obtener la aceleración que requiera

la aplicación. Es por esto que se trabajó con las métricas energía × delay, brindándole diferente peso a la energía en el producto. Esta sección analiza cuán eficientes en energía son las arquitectecturas de múltiples procesadores variando el número de procesadores instanciados en el arreglo. Este número impacta de manera drástica sobre el tiempo de procesamiento y el consumo de potencia. La cantidad de procesadores permite paralelizar en mayor o menor medida los datos a procesar pero a la vez modifica el ancho de banda disponible de la memoria principal y el tiempo de acceso a la memoria principal. Como resultado, el tiempo de procesamiento se ve afectado. En el caso del consumo de potencia, la relación es directa. Aquí se analiza entonces cuanto se reduce el producto energía por delay a medida que se varía el tamaño del arreglo de procesadores teniendo en cuenta la comunicación con la memoria principal.

Una posible configuración de paralelismo a nivel CMP se muestra en la Fig. 4.15. Se observa que el cómputo de las funciones Gaussianas se puede distribuir en diferentes unidades de cómputo. Cada unidad puede realizar pedidos a la memoria principal utilizando los anillos de la red N1. Cada unidad envía el resultado de su cómputo a un banco de memoria, al cual se accede a través de la red de comunicación N2. De esta manera se mitiga el impacto de la latencia de la memoria principal. La unidad HMM puede leer de dicha memoria los valores de las funciones Gaussianas que requiera.

Utilizando la metodología descripta en [1] y [41] se puede estimar el número óptimo de unidades de procesamiento teniendo en cuenta la comunicación de dichas unidades con la memoria principal. El método de optimización se basa en construir una función de costo (que puede incluir tiempo, área y

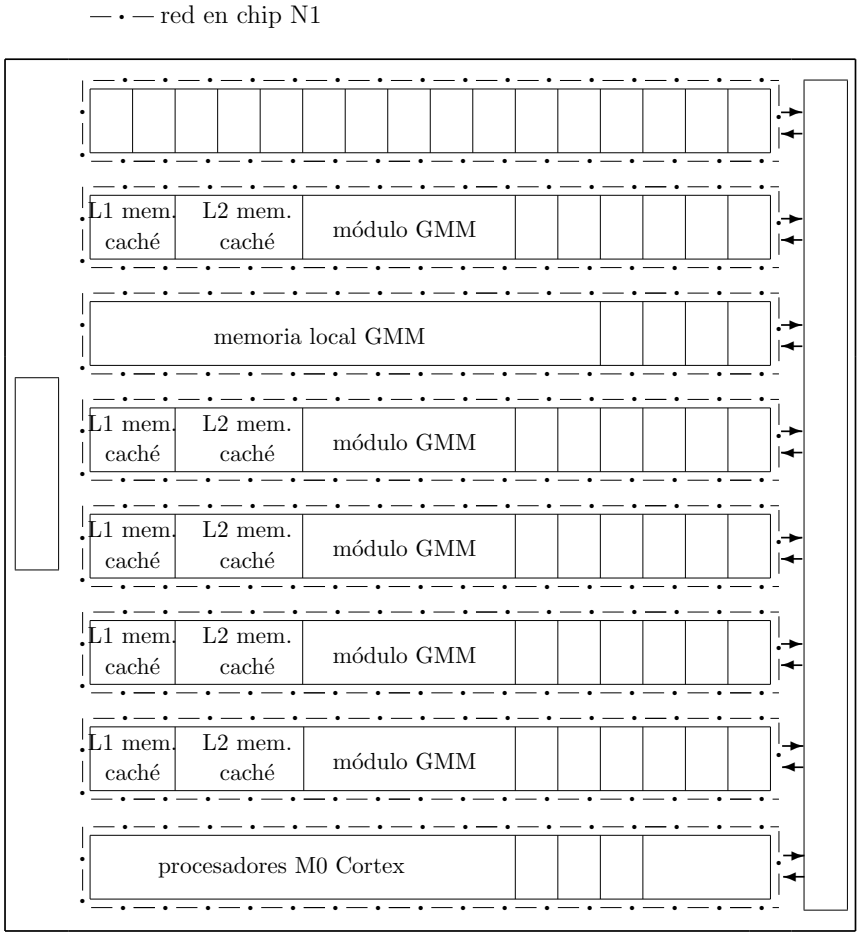


Figura 4.15: Paralelismo del módulo GMM en la arquitectura del CMP.

energía) y luego obtener el mínimo de dicha función. Los valores obtenidos de N, el área de los procesadores y de la memoria son entonces los óptimos [41].

Para el caso que aquí se analiza el área de los procesadores y de la memoria caché ya están definidos por lo que se busca el número óptimo de procesadores en paralelo teniendo en cuenta la red N1 y la memoria principal. Para ello se debe generar un módelo de comunicación de la red N1 teniendo en cuenta las características de la memoria principal.

El modelo de comunicación que se ajusta a la red N1 es el que se muestra en la Fig. 4.16 tal como está expresado en [1]. Cada uno de los anillos de la red N1 se puede modelar como una FIFO y la interfaz de la memoria principal es el servidor que se encarga de atender los pedidos a memoria en cada uno de las colas. Este modelo tiene el mismo comportamiento que una sola FIFO con una sola entrada y un sólo servidor. La tasa de arribo de los pedidos por cada anillo se denomina λ_i y la tasa de despacho de los pedidos del servidor se denomina μ. Para este modelo el tiempo promedio que los pedidos permanecen en las colas está definido por:

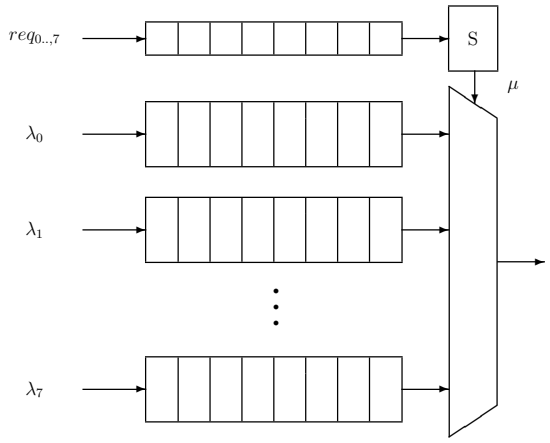


Figura 4.16: Modelo de la red N1 con múltiples entradas, una por cada anillo, compitiendo por el mismo recurso y el servidor atendiendo cada uno de los pedidos.

$$t = \frac{1}{\mu - \lambda} \tag{4.15}$$

donde λ es

$$\lambda = \sum_{i=0}^{N-1} \lambda_i \tag{4.16}$$

El costo en tiempo de la arquitectura CMP se expresa como [1]:

$$J_D = (F_{serie} + \frac{F_{paralelo}}{N}) \times (G_0 D_0 + G_1 D_1 + G_2 D_2) \tag{4.17}$$

donde F_{serie} y $F_{paralelo}$ son la fracción del algoritmo que se computa de forma serie y paralelo respectivamente, G_i es el porcentaje de las instrucciones que se computan en un tiempo D_i y N es la cantidad de unidades de procesamiento en paralelo.

Para el diseño bajo análisis, se considera $F_{serie} \approx 0$ y por lo tanto $F_{paralelo} \approx 1$. El porcentaje de instrucciones con referencia a datos de memoria que se encuentran en la memoria local se denota con G_0. Dado que no hay un nivel intermedio en la jerarquía de memoria, G_1 es cero. Por último, G_2 es el porcentaje de instrucciones con referencia a datos de memoria que no se encuentran en la memoria caché. En la Tabla 4.3 se muestra el valor de los parámetros G_i y D_i. Los parámetros HR y MR hacen referencia a la tasa de aciertos y desaciertos de la memoria caché, respectivamente. D_2 se modela de la siguiente manera:

$$D2 = \alpha_1 + \frac{1}{\mu - \lambda} \tag{4.18}$$

donde α_1 es que una constante que modela la latencia mínima de la memoria.

Por otra parte, λ se puede expresar como:

$$\lambda = \alpha_0 \times N = \frac{G_2 \times N}{CPI} \times N = \frac{MR}{CPI} \times N^2 \tag{4.19}$$

Tabla 4.3: Tabla de los parámetros correspondientes a (4.17).

Parámetro	**Valor**	**Tiempo**	**Valor**
G_0	HR	D_0	8
G_1	0	D_1	0
G_2	MR	D_2	$\alpha_1 + \frac{1}{\mu-\lambda}$

donde CPI es la cantidad de ciclos por instrucción y α_0 es el factor de escalamiento de la tasa de arribo de los pedidos a memoria. El valor de CPI se puede expresar como

$$CPI = G_0D_0 + G_1D_1 + G_2D_2 \tag{4.20}$$

Por lo tanto α_0 está definido de la siguiente manera:

$$\alpha_0 = \frac{MR \times N}{(G_0D_0 + G_1D_1 + G_2D_2)} \tag{4.21}$$

Utilizando los valores de los parámetros de la Tabla 4.3, (4.21) se puede reescribir como:

$$\alpha_0 = \frac{MR \times N}{(HR \times 8 + MR \times (\alpha_1 + \frac{1}{\mu-\alpha_0 \times N})} \tag{4.22}$$

A continuación es necesario estimar los parámetros: α_0, α_1, μ, MR, HR y CPI. La memoria caché tiene una capacidad de almacenamiento para 384 Gaussianas. La tasa de desaciertos y aciertos dependerá de la cantidad de Gaussianas que cada unidad de procesamiento tiene asignada. Esta cantidad está a su vez definida por la cantidad de unidades de procesamiento que se instancian por lo que depende de N, de la siguiente manera:

$$HR = \frac{N_{cache} \times N}{N_{total}} \tag{4.23}$$

$$MR = 1 - HR \tag{4.24}$$

donde N_{cache} es la cantidad de Gaussianas que se pueden almacenar en la memoria caché y N_{total} es la cantidad de Gaussianas total a computar. En este caso N_{cache} es igual a 384 y se consideró que el objetivo es computar 100000 Gaussianas.

El valor de μ es inversamente proporcional a la latencia del anillo y proporcional a la cantidad de anillos:

$$\mu = \frac{\#anillos N1}{latencia\ anillo\ N1} \tag{4.25}$$

El parámetro α_1 se define como la suma de la latencia de la memoria principal, de la interfaz con el puerto de la memoria principal y dos veces la latencia del anillo de la red N1, ya que hay que tener en cuenta el tiempo que le toma al pedido llegar hasta la interfaz de la memoria y el tiempo que le toma al dato llegar a la unidad de procesamiento destino:

$$\alpha_1 = 2 \times\ latencia\ anillo\ N1\ +\ latencia\ mem.\ ppal\ +\ latencia\ I/F\ mem. \tag{4.26}$$

La Tabla 4.4 muestra la latencia que introduce cada componente de la red de comunicación, la memoria principal y la tasa de despacho de los pedidos

Tabla 4.4: Latencia de cada componente de comunicación y la memoria principal en [*ciclos*] y la tasa de despacho de los pedidos a memoria en [$ciclos^{-1}$]
.

Parámetro	**ciclos @1GHz**	**ciclos @300 MHz**	**ciclos @40MHz**
I/F memoria ppal.		146	20
anillo N1		32	5
memoria ppal.	10	3	1
α_1		213	31
μ		0.25	1.875

a memoria. La memoria principal opera a 1 GHz mientras que el anillo de la red N1 y la interfaz con la memoria operan a 300 MHz. Para estimar el valor de μ y α_1, se convierten los valores de latencia de cada componente a la frecuencia de operación de la unidad de procesamiento que es 40 MHz. En caso de que dicha conversión arroje un número con decimal se considera el valor entero superior.

El controlador de memoria posee puertos de comunicación con los bancos de memoria que operan a 1GHz. Dado que la latencia de la memoria es variable y no se dispone información del fabricante acerca del valor mínimo, se consideró un valor de 10 ciclos (equivalente a la latencia mínima de una memoria DDR2).

Para obtener el valor de N que minimiza la función costo se debe calcular la derivada respecto a N de (4.27) e igualar a cero:

$$J_D = \frac{1}{N} \times (HR \times 8 + MR \times (\alpha_1 + \frac{1}{\mu - \alpha_0 \times N})) \qquad (4.27)$$

La Ec. (4.22) y (4.27) forman un sistema de ecuaciones. Para obtener el

gráfico de la funcion costo vs N, se realiza la solución cerrada de este sistema. Para obtener el valor de N óptimo se obtiene la derivada primera y se iguala a cero. La Fig. 4.17 muestra la función costo J_D y el valor de D_2 para varios valores de N. En este caso, el valor óptimo es 17.38, por lo que se asume $N = 17$. Para dicho valor cada unidad de procesamiento debe calcular 5882 Gaussianas lo cual implica una aceleración respecto al tiempo real de 2.04 veces.

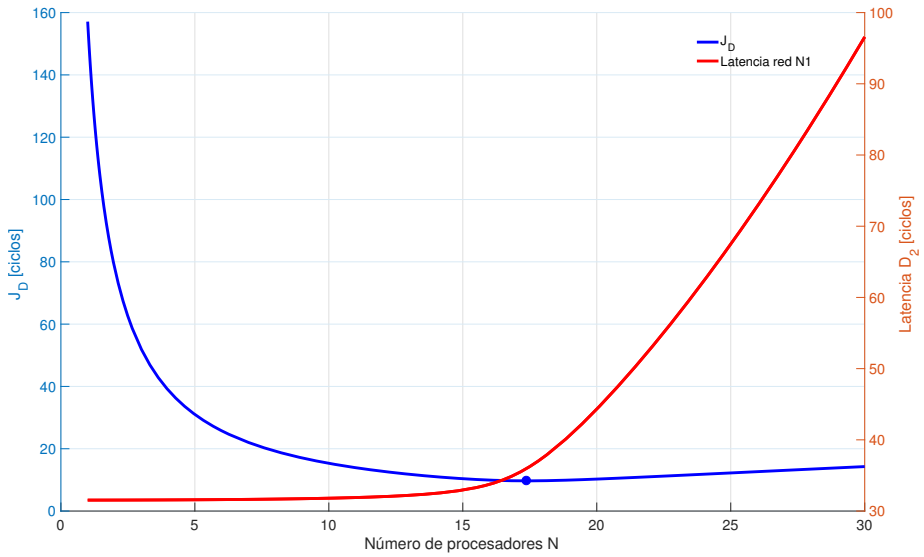


Figura 4.17: Función costo J_D y D_2 vs cantidad de procesadores N.

Como se mencionó anteriormente la latencia de la memoria principal es variable por lo que puede ser mayor que el valor considerado hasta aquí. La Tabla 4.5 detalla el valor de N óptimo para distintos valores de latencia mínima de la memoria principal. Como se observa, la optimización brinda un valor de N mayor a medida que la latencia aumenta. A medida que la comunicación con la memoria principal resulta más costosa en tiempo, la

Tabla 4.5: Valores de N óptimo para diferentes valores mínimos de la latencia de la memoria principal.

Latencia memoria ppal. @ f_{op} = 1 Ghz. [ciclos]	**N óptimo**
100	18.19
200	19.21
400	21.11
500	22.00

Tabla 4.6: Valores óptimos de N diferentes puntos de trabajo definidos por la frecuencia y tensión y tasas de despacho de pedidos.

f_{op}, V_{op}	N	μ
40 MHz, 0.63 V	17.38	1.87
77 MHz, 0.72 V	16.98	0.97
117 MHz, 0.81 V	16.87	0.63
285.7 MHz, 0.99 V	16.71	0.26
617.2 MHz, 1.08 V	16.65	0.12

cantidad de ciclos por instrucción (CPI) se incrementa, reduciendo así la tasa de arribo de pedidos λ. Esto hace posible que un número mayor de unidades realice pedidos a memoria en paralelo disminuyendo así el tiempo de procesamiento total. A su vez, un número mayor de unidades disminuye la tasa de desaciertos y por lo tanto disminuye la tasa de arribo de pedidos.

Es válido analizar que sucede cuando aplicamos la misma metodología para distintos puntos de trabajo de tensión y frecuencia. La Tabla 4.6 muestra el número óptimo de procesadores para los distintos casos analizados. Se puede observar que a medida que aumenta la frecuencia de operación el valor óptimo de N es un número cada vez menor. Esto se explica por la reducción en la tasa de despacho μ, respecto de la tasa de procesamiento de las unidades, a medida que se aumenta la frecuencia de operación.

4.3.1. Consideración de la energía

Es posible, incluir no sólo el tiempo de procesamiento sino también la energía de consumo al momento de analizar el paralelismo. Para ello, se utiliza nuevamente la métrica dada por el producto entre la energía consumida y el tiempo de procesamiento. Continuando con la metodología de [41] se construye la función costo considerando el tiempo de procesamiento y el consumo de energía de la arquitectura del CMP.

De la síntesis lógica se obtiene la estimación de potencia que consume la arquitectura de un sólo procesador para una frecuencia y voltaje de operación de 40 MHz y 0.63V. En la sección anterior se observa que resulta el punto óptimo de trabajo. La función costo, sin tener en cuenta el consumo de energía de la comunicación con la memoria principal, está dado por:

$$J_{E\_D} = \frac{1}{N} \times (HR \times 8 + MR \times (\alpha_1 + \frac{1}{\mu - \alpha_0 \times N})) \times (E \times N)^{\gamma} \quad (4.28)$$

Si se utiliza la estimación de potencia P, la función costo queda:

$$J_{E\_D} = \frac{1}{N} \times (HR \times 8 + MR \times (\alpha_1 + \frac{1}{\mu - \alpha_0 \times N}))^{\gamma+1} \times P^{\gamma} \quad (4.29)$$

donde el parámetro γ es el exponente de la energía. Realizando la optimización de (4.29) para γ igual a 1 resulta $N = 16, 17$. La Fig. 4.18 muestra la función costo en función de la cantidad de procesadores.

La Tabla 4.7 muestra el número óptimo de unidades a instanciar para

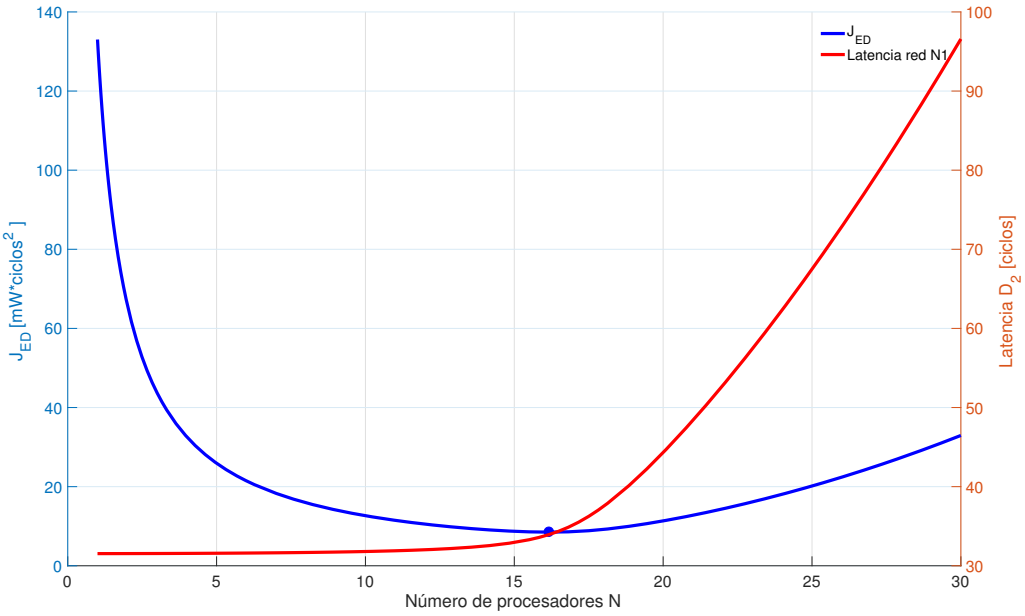


Figura 4.18: Función costo J_{ED} y D_2 vs. cantidad de procesadores N.

distintos puntos de operación. Se observa que a medida que se incrementa la frecuencia de operación la tasa de despacho de pedidos a memoria, μ, disminuye y por otro lado la potencia de consumo por procesador aumenta. Debido a estos dos factores, la optimización entrega como resultado un número menor de unidades de procesamiento en paralelo. Por otro lado, al utilizar una métrica en la cual se pondera más el consumo de energía, $E^2 \times D$, la optimización arroja como resultado un número aún menor de unidades a instanciar.

La Fig. 4.19 muestra la función costo vs. el número de procesadores N para distintos puntos de operación. Nuevamente se observa que el punto óptimo de trabajo es 0.63V y 40MHz.

Tabla 4.7: Valores óptimos de N para dos valores de γ y diferentes puntos de trabajo definidos por frecuencia y tensión.

pto. op.	f_{op}, V_{op}	$\gamma = 1$	$\gamma = 2$	$\gamma = 3$
1	40 MHz, 0.63 V	16.17	15.60	15.21
2	77 MHz, 0.72 V	15.77	15.20	14.82
3	117 MHz, 0.81 V	15.66	15.10	14.71
4	285.7 MHz, 0.99 V	15.50	14.94	14.55
5	617.2 MHz, 1.08 V	15.45	14.88	14.50

4.3.2. Discusión

En el marco del diseño de arreglos de múltiples procesadores resulta lógico que a medida que se trabaja con frecuencias de operación menores, el costo en tiempo de ir a buscar datos a memoria disminuye y aumenta la tasa de despacho de pedidos respecto a la tasa de procesamiento de las unidades. Si la memoria que se utiliza resulta más lenta que las unidades de procesamiento y se dispone de suficiente área de silicio o de tecnologías con gran densidad de integración, una solución es operar a una frecuencia de operación menor. Esta solución brindará configuraciones de arreglos más eficientes en términos de rendimiento y energía. En esta sección se analizó el número óptimo de unidades a instanciar en paralelo utilizando una función de costo [41].

En el primer análisis realizado se observa que cuando el costo en tiempo de ir a buscar datos a memoria aumenta, ya sea por diseño propio de la memoria o porque la unidad de procesamiento opera a frecuencias superiores, se debe disminuir el número de unidades en paralelo ya que la tasa de despacho impone un límite en la cantidad de procesadores a instanciar. Es válido notar que el número óptimo de unidades se obtuvo teniendo en cuenta la tasa de desaciertos de la memoria caché. Esta última varía con el número de proce-

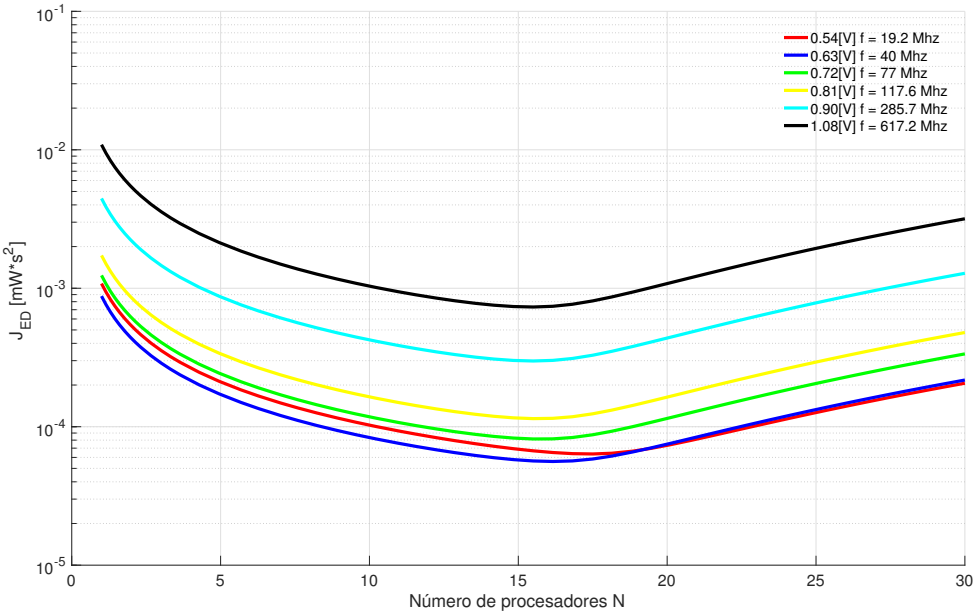


Figura 4.19: Función costo J_{ED} vs. cantidad de procesadores N para varios puntos de trabajo.

sadores ya que la cantidad de datos total a procesar se divide entre todas las unidades. Por el contrario, cuando los procesadores operan a frecuencias bajas, la tasa de despacho de la interfaz de memoria aumenta respecto a la velocidad de procesamiento de las unidades permitiendo así instanciar un número mayor de unidades en paralelo. Además, al instanciar un número mayor de unidades la cantidad de datos a procesar por cada unidad disminuye. En consecuencia, la tasa de desaciertos se reduce y la tasa de arribo de pedidos al controlador a memoria también.

El segundo análisis realizado considera la energía y provee de una exploración más completa del espacio de diseño. La función costo es el producto energía y tiempo de procesamiento y el número óptimo de unidades que minimiza dicha función es menor que el obtenido en el primer análisis. Este comportamiento es de esperar ya que ahora se tiene en cuenta el consumo de

energía de los procesadores y se acentúa a medida que se le da más peso a la energía en el producto.

Es conveniente señalar que este resultado depende del modelo seleccionado para la memoria y la red N1 y son fuertemente dependientes de los parámetros utilizados. Por lo que este análisis debe ser adaptado si se trabaja con otra topología de red y memoria.

4.4. Conclusiones

El paralelismo es una estrategia para reducir el consumo de energía en esta era de la tecnología en la cual el volumen de datos a procesar crece a tasas elevadas. Si se requiere del procesamiento de un gran volumen de datos en tiempo real, se necesita de poder de cómputo, núcleos de procesamiento eficientes que cuenten con jerarquía de memoria que permita acceso a memorias de almacenamiento de datos masivas con tasas de transferencia elevadas.

En esta Sección se describió el impacto de la paralelización del algoritmo teniendo en cuenta la latencia de la memoria principal y el tamaño de la memoria caché así como también el nivel de asociatividad de las mismas. Tal como se mencionó previamente se concluye que el paralelismo a nivel microarquitectura depende fuertemente de la velocidad de la memoria y de cuan eficiente es la jerarquía de memoria en su tarea de mitigar el impacto de la latencia de la memoria principal. El costo de ir a buscar datos a memoria principal impacta drásticamente el tiempo de procesamiento por lo que contar con estructuras de transferencia de datos eficientes resulta preponderante en

el paralelismo.

Se analizó la memoria caché y el patrón de acceso a memoria de la unidad GMM y se realizaron modificaciones de modo tal de incrementar la tasa de aciertos y lograr una jerarquía más eficiente. Se analizó el hecho de incrementar la asocitiavidad en la memoria caché y como conclusión se observa que no resulta beneficioso.

Se presentaron dos alternativas de paralelismo a nivel micro-arquitectura y se analizó cada una por separado desde el punto de vista de la aceleración en el tiempo de procesamiento y el consumo de energía teniendo en cuenta la latencia de la memoria principal. Para ello se utilizó la métrica energía por delay que permite analizar ambas variables en conjunto.

Se plantean además posibles soluciones para mitigar aún más el costo en tiempo de ir a buscar datos a memoria. Una de ellas es adicionar un tercer nivel más a la jerarquía de memoria en el CMP. Otra alternativa es utilizar unidades de acceso a directo a memoria con estructuras que permitan realizar predicción en la carga de datos desde memoria principal. Mientras los procesadores están realizando el procesamiento de un conjunto de parámetros, las unidades de acceso directo se encargan de ir a buscar a memoria un nuevo conjunto de parámetros a procesar.

Se analizó también el paralelismo a nivel de CMP. Esto es, como impacta el número de procesadores que componen el CMP en el tiempo de procesamiento y el consumo de energía. Se incorpora en el análisis la comunicación con la memoria principal. Dado que el acceso memoria es compartido por todas las unidades de procesamiento, al incorporar más unidades se incrementa el tiempo de comunicación con la memoria. Del análisis se obtiene un núme-

ro óptimo de unidades de procesamiento teniendo en cuenta la comunicación con la memoria y el consumo de potencia de las unidades de procesamiento.

Capítulo 5

Unidades de propósito general

En este capítulo se describen unidades de propósito general que fueron diseñadas e implementadas para integrarse en el Sistema en chip. Una de ellas es un sistema de cómputo de propósito general basado en el procesador CORTEX M0 [42] que cuenta con jerarquía de memoria, unidad de acceso directo a memoria y periféricos de comunicación. El diseño se focalizó en lograr mayor eficiencia en la transferencia de datos con la memoria principal de manera tal de maximizar el rendimiento computacional del sistema y reducir el consumo de energía. La otra unidad es un multiplicador matriz-vector de propósito general cuyo diseño se focaliza en el paralelismo de manera de poder reducir la frecuencia y el voltaje de operación manteniendo una tasa de procesamiento elevada elevada. Ambos diseños se integraron como parte del sistema de multiprocesadores y se comunican con las redes en chip descriptas en la Sección 3.1.1.

5.1. Procesador

El procesador y sus periféricos es uno de los sistemas integrados en el sistema de multi procesadores. Tal como se mencionó en la Sección 3.1.1 el CMP cuenta con múltiples unidades de procesamiento. El procesador M0 está a cargo de la configuración de las mismas y también arbitra el procesamiento y la comunicacón entre ellas. Alguno de los parámetros que el procesador configura para cada unidad de procesamiento son: inicio de direcciones de memoria DDR para lectura y/o escritura, direcciones destino a otras unidades de procesamiento, cantidad de iteraciones para la lectura de datos de la memoria DDR.

Las unidades de procesamiento trabajan en dos fases: de procesamiento y de comunicación. El procesador M0 inicia el procesamiento de las unidades. Luego recibe de cada una de las unidades involucradas en el procesamiento un paquete de sincronización que indica el fin del computo. Una vez que todos estos paquetes fueron recibidos el procesador inicia la fase de comunicación. Esto lo lleva a cabo enviando paquetes de reconocimiento de fin de cómputo a las mismas unidades. La Figura 5.1 muestra el diagrama temporal de las fases de configuración, comunicación y procesamiento.

El otro propósito del sistema es realizar operaciones diversas de propósito general. De forma tal de mitigar el costo en tiempo de ir a buscar datos a memoria principal se adicionó una jerarquía de memoria que incluye una memoria caché. El sistema cuenta, además, con una unidad de acceso directo que busca datos a memoria principal en paralelo con la ejecución de instrucciones por parte del procesador. Una vez finalizada la transferencia,

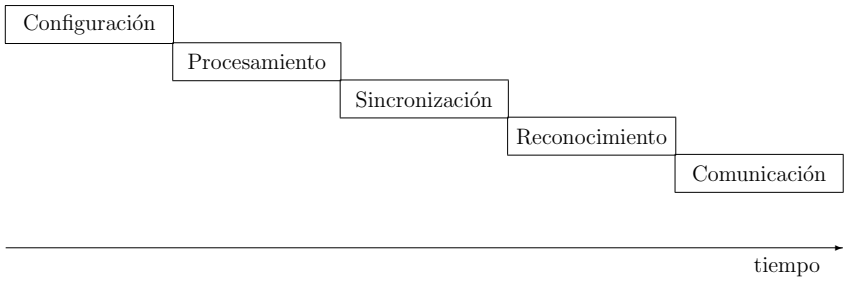


Figura 5.1: Evolución de las fases de configuración, procesamiento y comunicación entre unidades del CMP en el tiempo.

los datos son procesados . De esta manera, se logra una mayor aceleración en el procesamiento.

Para el diseño del sistema, se tomó el procesador de propósito general CORTEX M0 diseñado por ARM. El procesador CORTEX M0 es el más pequeño en términos de área de silicio provisto por la compañía ARM. El diseño está optimizado para aplicaciones de microcontroladores y sistemas embebidos. Las propiedades intelectuales que la compañía provee en forma gratuita son: el procesador, el decodificador y el multiplexor de señales para el AMBA bus. En cuanto a periféricos, la compañía provee un transmisor-receptor asincrónico universal (UART, del inglés universal asynchronous receiver-transmitter).

Se elegió este core en vez de implementar uno propio para proveer de mayor confiabilidad al sistema. Además se instanciaron dos sistemas idénticos en la última fila del CMP, uno principal y uno de resguardo para proveer de redundancia al sistema. En caso de que por fallas de fabricación no funcione el principal se procede a trabajar con el sistema de resguardo.

Al sistema se le adicionaron unidades de comunicación inter-chip y también de comunicación hacia el exterior, esto es, una unidad QSPI.

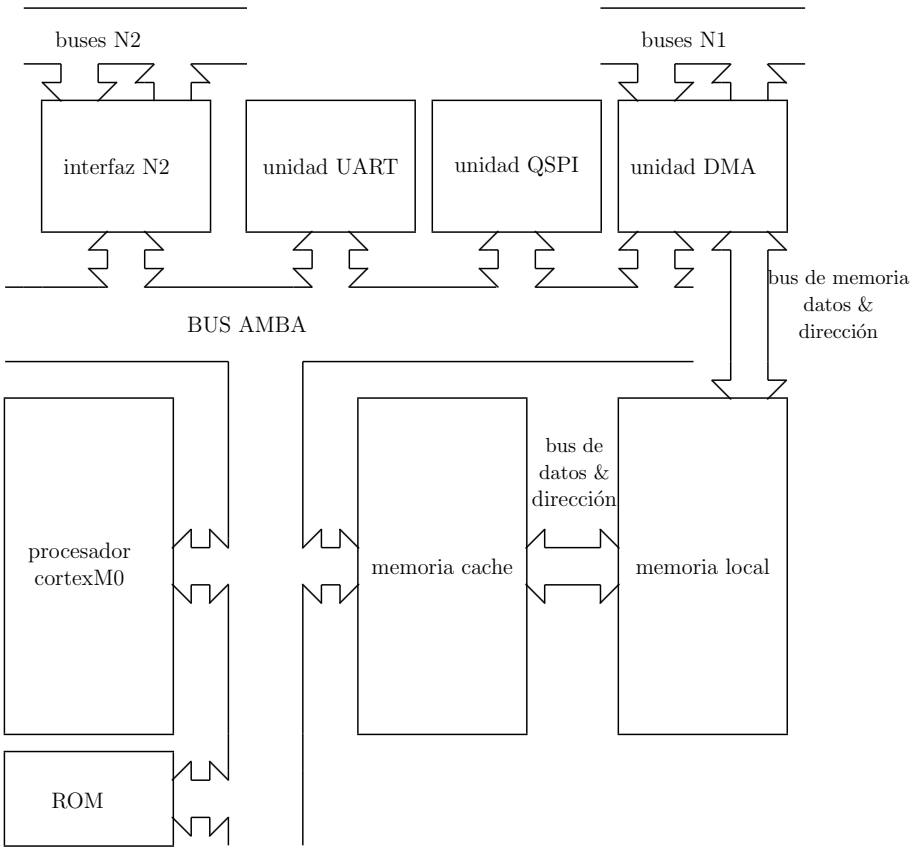


Figura 5.2: Diagrama en bloques del sistema.

A continuación se describe el diseño e implementación de la arquitectura del sistema de propósito general, la jerarquía de memoria junto con la memoria caché y sus periféricos de comunicación. Luego se realiza lo mismo para la unidad de multiplicación matriz-vector.

5.1.1. El procesador CORTEX M0

El procesador se comunica con sus periféricos a través de un bus AMBA. La Fig. 5.2 muestra el diagrama en bloques del sistema completo.

El procesador se inicializa con el programa almacenado en la memoria

ROM. Dicho programa espera que se cargue la memoria local con instrucciones que son enviadas por un dispositivo externo. El sistema cuenta con con una memoria local del tipo SRAM de 32KB y una memoria caché de 2KB tanto para datos como para instrucciones dado que su arquitectura es Von Neumann.

El procesador CORTEX M0 posee 3 etapas de pipeline, un juego de instrucciones de 32 bits del tipo RISC [43] y un bus de datos también de 32 bits. Tal como se mencionó cuenta con una arquitectura Von Neumman la cual incluye además un controlador de interrupciones. Este último permite el anidamiento de interrupciones y gestionar un máximo de 32. Las interrupciones permiten atender eventos mientras se ejecuta el software cargado en el M0. La arquitectura cuenta además con un multiplicador serial, el cual requiere 32 ciclos de reloj para completar la operación.

Como se detalló previamente el chip cuenta con dos redes para comunicarse con la memoria principal y el arreglo de unidades de procesamiento. Se diseñó una interfaz para la red N2 de modo tal que el procesador pueda comunicarse con las unidades. Además se incluyó una unidad de acceso directo a memoria (DMA, del inglés Direct Memory Access) para realizar transferencias de datos entre el sistema y la memoria principal con intervención mínima del procesador. Esta última también cuenta con una interfaz para la red N1.

El procesador se comunica con el mundo exterior a través de la unidades UART y QSPI. El propósito principal de la unidad UART es para transferencia de instrucciones hacia el procesador, envío de datos hacia el exterior y testeo del sistema. La unidad SPI se incluyó para tener la posibilidad de

conectar una memoria externa de manera de incrementar la capacidad de memoria del sistema.

El sistema completo está mapeado a memoria. Esto implica que para acceder a cualquier periférico desde el procesador, simplemente se debe acceder a regiones de memoria específicos. La Tabla 5.1 muestra el inicio de las direcciones que se deben utilizar para cada periférico. En el caso de la memoria caché se requiere que esta sea ejecutable por lo que se le asignó el intervalo de direcciones que el procesador tiene destinado para la ejecución de instrucciones.

El bus AMBA posee un decodificador que toma la dirección presentada por el procesador y en base a ésta habilita el periférico correspondiente. Además cuenta con un multiplexor para seleccionar el dato de salida del periférico seleccionado. Dicho multiplexor también está controlado por el decodificador.

Por último, al iniciar el sistema el reloj proviene de una señal externa. La tasa de baudios de las transmisiones a través de la UART se establecen con la frecuencia de dicho reloj. Luego, una vez que todas las instrucciones fueron transmitidas, se selecciona el reloj del sistema a través de un circuito asincrónico diseñado por Tomas Figliola de la Universidad de Johns Hopkins. El circuito es un multiplexor de dos entradas que selecciona la fuente del reloj realizando las transiciones de una a otra sin fallas. Se accede a dicho circuito como un periférico del bus AMBA. Más adelante se realiza una descripción del mismo.

Cuando inicia, el procesador comienza a leer instrucciones de la dirección

0x00000000 que es la región asignada a la memoria ROM[1]. Este programa le indica al procesador que debe leer instrucciones recibidas a través de la UART, almacenarlas en la memoria RAM y por último ejecutarlas. Además el programa también tiene en cuenta la programación de las rutinas que sirven las interrupciones.

La velocidad de transmisión de datos de la UART depende del reloj del sistema. Esto implica que la velocidad se puede configurar simplemente cambiando la frecuencia del reloj. La transmisión es completamente serial.

La primer palabra de 32 bits enviada a través de la UART indica cuantas instrucciones van a ser transmitidas. Entonces, el procesador ya tiene conocimiento de cuantas instrucciones debe aguardar. El programa almacenado en la memoria ROM opera con la UART sin interrupciones esto implica que el procesador verifica constantemente si hay un dato disponible. Una vez que todas las instrucciones fueron transmitidas, es posible activar las interrupciones. Esto se hace así para tener más flexibilidad en el manejo de la memoria RAM. De lo contrario, sería necesario destinar una porción de la memoria como pila y establecer su puntero.

El programa de inicio almacenado en la ROM opera la memoria de la siguiente manera: las instrucciones que son recibidas son almacenadas en los últimos lugares de la memoria. Una vez que todas las instrucciones fueron recibidas, el programa establece el puntero de la pila en el primer lugar de la memoria. Luego, el procesador salta a la dirección de memoria establecido por la segunda palabra. Esta dirección es 0x60000000 tal como lo expresa la

[1]El programa almacenado en esta memoria fue diseñado por Dan Mendat de la Universidad de Johns Hopkins.

Tabla 5.1: Direcciones de memoria iniciales para cada periférico del sistema.

Periférico	**Dirección de inicio**
ROM	0x00000000
Cache	0x60000000
Interfaz N2 escritura	0x53000000
Interfaz N2 lectura	0x54000000
DMA	0x57000000
UART	0x52000000
SPI	0x58000000
Selector de reloj	0x5600 0000

Tabla 5.1, dando inicio a la ejecución del programa cargado.

5.1.2. Diseño de la memoria

El diseño de la memoria consiste en una jerarquia de memoria de tres niveles. Para este diseño, en primer lugar, se incorporó la memoria local con el objetivo de mitigar el impacto de la latencia de la memoria DDR, la cual puede tomar un número considerable de ciclos de reloj en el bus de la red N1. La memoria local cuenta con una capacidad de almacenamiento de 32 KB, que es el máximo permisible debido al área de silicio disponible. La memoria es un banco de memorias de 64x32 de bajo consumo. La Fig. 5.3 muestra la organización del banco. El ancho de la palabra es 256 bits al igual que el bus de la red N1 destinada a la memoria DDR. Esto se definió así para simplificar el movimiento de datos entre la memoria local y la memoria principal DDR. Posee puertos dedicados para la unidad DMA y la memoria caché. Estas dos unidades se describen más adelante.

La implementación de la memoria local requiere el uso de registros de entrada y salida dado que de otro modo no es posible sintetizar el diseño

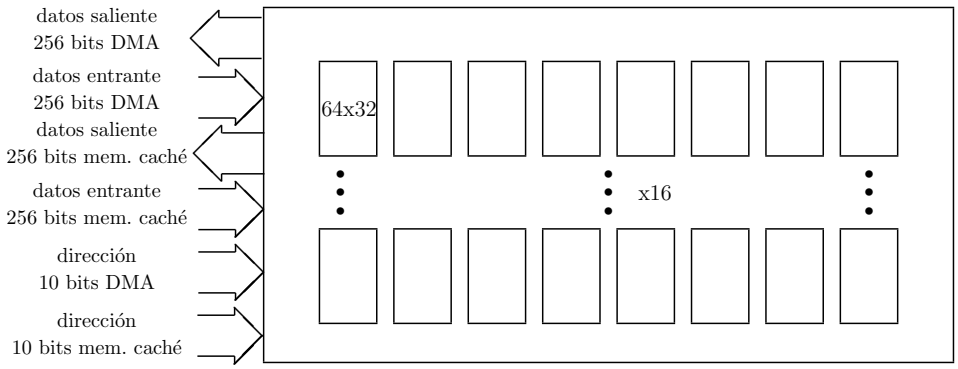


Figura 5.3: Arquitectura de la memoria local.

a la frecuencia de operación, definida en 166 MHz. La inclusión de dichos registros incrementa la latencia de la memoria local: se requiere de dos ciclos de reloj para realizar una lectura y un ciclo para realizar una escritura, una vez que la dirección y el dato se presentan en los puertos de la memoria. A su vez, la memoria local debe ir conectada al bus AMBA. Sin embargo, el protocolo AMBA establece que para transferencias de lectura y escritura entre periféricos y el procesador, el dato debe estar presente en el bus un ciclo de reloj después de haber presentado la dirección [44]. De no ser así, el dispositivo de almacenamiento debe forzar ciclos de espera en el bus, afectando la velocidad de procesamiento del procesador. Esto no resulta aceptable dado que el procesador debe acceder a memoria frecuentemente ya sea para la búsqueda de instrucciones o datos.

La solución adoptada para conectar el banco de memorias al bus AMBA sin por ello reducir la velocidad de procesamiento del procesador, consiste en incluir una memoria caché en un nivel adicional en la jerarquía de memoria.

El procesador accede a la memoria caché a través del bus AMBA y ésta a su vez se conecta a través de puertos dedicados a la memoria local. La

memoria caché se diseñó con un ancho de palabra de 256 bits siendo completamente compatible con el bus de datos de la memoria local. El procesador y la memoria a su vez operan con palabras de 32 bits.

En este caso, se implementó una memoria de asociatividad de dos vías con un tamaño del bloque de 256 bits y se adoptó la politica de reemplazo denominada el último utilizado descripta en la Sección 3.2.2.1. El bloque a su vez está dividido en 8 palabras de 32 bits. El procesador puede acceder a cualquiera de estas palabras en forma individual. La capacidad total de la memoria es de 2KB. Cuando el procesador realiza una lectura o escritura y el bloque no está almacenado (un evento *miss*), el controlador de la memoria caché busca a memoria local el dato requerido y selecciona un bloque para ser reemplazado.

En cuanto a la operación de escritura en las memorias caché, existen diferentes métodos. Un ejemplo es la escritura del dato en la memoria caché e inmediatamente se procede a la actualización del bloque correspondiente en la memoria local. Otra posibilidad es escribir en la memoria caché sin actualizar el contenido en memoria local hasta que la regla de reemplazo selecciona el bloque para ser reemplazado. Si el bloque que se pretende modificar no se encuentra en la memoria caché, existen dos opciones: 1) escribir directamente en la memoria local e ignorar la memoria caché o 2) traer el bloque de la memoria local y luego escribir el dato siguiendo alguna de las reglas previamente mencionadas. Todas estas estrategias poseen ventajas y desventajas [15]. En este diseño se implementó la estrategia (1) debido a su simpleza en la implementación. En el caso que el bloque a modificar no se encuentre en memoria se trae el bloque de memoria local y luego se escribe

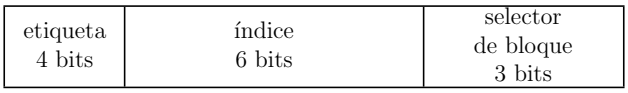


Figura 5.4: Utilización de la dirección de memoria por parte de la memoria caché.

siguiendo la estrategia de escritura adoptada.

La Fig. 5.4 muestra como se divide la dirección del procesador en los campos: selector de bloque, índice y etiqueta. El índice se utiliza para direccionar el arreglo de bloques correspondiente. La etiqueta se utiliza para identificar los bloques dentro de un arreglo. Por último, el selector de bloque se utiliza para seleccionar una de las 8 palabras de 32 bits que conforman el bloque. Notar que sólo se utilizan 14 de los 32 bits que tiene disponible el procesador para direccionar memoria.

Cuando se escribe un bloque a memoria también se almacena en memoria la etiqueta. Luego si se quiere acceder a un bloque, lo primero que el controlador de memoria hace es verificar que la etiqueta de la dirección de memoria a la que se quiere acceder, coincida con la etiqueta almacenada. De ser así el bloque se encuentra en memoria, caso contrario el controlador debe buscar el bloque en la memoria local. La dirección de memoria local se construye concatenando el campo etiqueta e índice.

En la Fig. 5.5 se muestra la arquitectura de la memoria caché. La memoria posee asociatividad de dos vías por lo tanto cuenta con dos memorias de bloque y de etiquetas. Un bloque se escribe en una de las dos memorias de acuerdo a la politica de reemplazo. Al momento de buscar un bloque particular, el controlador de memoria verifica en las dos memorias de etiqueta por lo que la búsqueda se paraleliza.

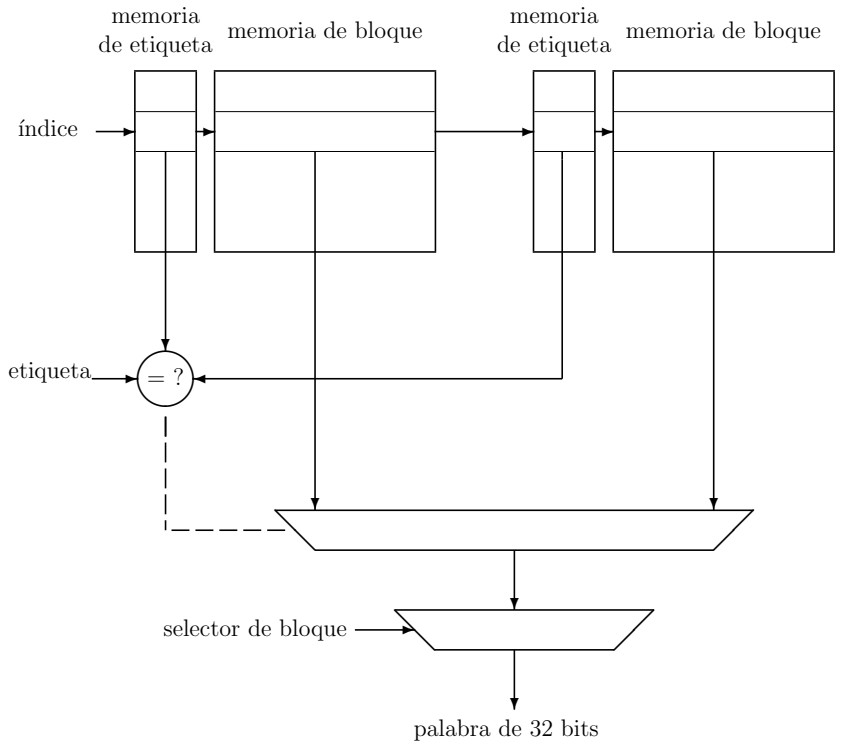


Figura 5.5: Diagrama en bloques de la memoria caché.

La unidad de acceso directo a memoria realiza transferencias de datos entre la memoria local y la memoria principal. Si la transferencia es desde la memoria principal a la memoria local, puede suceder que existan problemas de coherencia de datos entre la memoria local y la memoria caché. Esto sucede porque hay datos que fueron modificados en la memoria local y no así en la memoria caché. La solución a este problema es la inclusión de un protocolo de coherencia ante eventos similares al descripto. Existen varios protocolos en la bibliografía [37] [45]. En este diseño se implementó el protocolo llamado snooping [15] debido a su sencilla implementación.

5.1.2.1. Unidad de acceso directo a memoria (DMA)

La unidad de acceso de directo a memoria tiene como propósito realizar transferencias de datos entre la memoria principal DDR y la memoria local del procesador con mínima intervención por parte de este último. Esta unidad resulta útil en la configuración de las unidades de procesamiento. Para algunas de ellas su procesamiento se basa en datos almacenados en la memoria principal DDR. El procesador puede cargar datos a memoria utilizando la unidad de DMA, configurar las unidades de procesamiento a través de la red e inicializar el procesamiento.

La unidad de DMA se comunica directamente con la memoria principal y la memoria local sin intervención del procesador, excepto cuando este accede a la unidad para su configuración. La memoria local está organizada en bancos brindando acceso simultáneo entre la unidad DMA y el procesador. Mientras la unidad DMA realiza transferencias de datos en un banco de memoria, el procesador continua ejecutando instrucciones almacenadas en otro banco.

La unidad DMA cuenta con un juego de registros cuyo propósito es configurar y controlar las transferencias y además proveer información al procesador del estado de las mismas. Estos registros están mapeados al bloque de memoria cuyo inicio es la dirección 0x57000000. La Tabla 5.2 detalla todos los registros que la unidad DMA posee.

El procesador configura la dirección inicial y final de la memoria local y principal definiendo así la cantidad de palabras de 256 bits a ser transferidas. La dirección de la memoria principal es de 40 bits por lo que el procesador

debe escribir en dos registros de 32 bits para conformar la dirección.

El procesador puede acceder a los bytes que conforman una palabra de 32 bits. Por ejemplo la dirección 0x60000003 accede al cuarto byte de la primer palabra de 32 bits. Para leer datos de la memoria alineados con palabras de 32 bits, el procesador debe acceder a lugares de memoria múltiplos de 4. Por otro lado, el ancho de palabra de la memoria local es de 256 bits, lo que equivale a 8 palabras de 32 bits. Dicho esto, el mapeo de la dirección del procesador al de la memoria local consiste en dividir la primera por la constante 32.

Una vez configuradas las direccciones, el procesador define el tipo de transferencia: lectura de la memoria principal o escritura a la misma accediendo al registro dma_trsf_type. Esta última configuración inicia la transferencia. Luego el procesador puede acceder a los registros $dma\_status\_reg\_0$ y $dma\_status\_reg\_1$ para conocer el estado en el que la transferencia se encuentra. Además cuenta con la posibilidad de abortar o poner en pausa una transferencia accediendo al registro de control dma_ctrl_reg.

Una vez que la transferencia finalizó, la unidad dispara una interrupción por nivel. Existen dos interrupciones: una asociada a la lectura y la otra a la escritura. Cuando el procesador atiende la interrupción la rutina de servicio debe limpiar el pedido de interrupción. Esto lo realiza simplemente escribiendo en el registro $dma\_wr\_irq\_clr$ o $dma\_rd\_irq\_clr$, dependiendo del tipo de transferencia que haya sido completada.

La Fig. 5.6 muestra el diagrama en bloques de la unidad DMA. La unidad de control genera los pedidos de lectura y escritura tanto para la red N1 (o memoria principal) como para la memoria local. La unidad también

Tabla 5.2: Tabla de registros internos de la unidad de DMA.

Registro	**Descripción**	**dirección memoria**	**Detalle**
dma_ctrl_reg	registro de control	0x5700_0000	Bit 0 : transferencia cancelada. Bit 1 : transferencia en pausa.
sys_addr_0_l	32 bits bajos del registro 0 de dirección de memoria ppal.	0x5700_0004	
sys_addr_0_h	8 bits altos del registro 0 de dirección de memoria ppal.	0x5700_0008	
sys_addr_1_l	32 bits bajos del registro 1 de dirección de memoria ppal.	0x5700_000C	
sys_addr_1_h	8 bits altos del registro 1 de dirección de memoria ppal.	0x5700_0010	
tcm_addr_0	registro 0 de dirección de mem. local	0x5700_0014	Bits 9-0.
tcm_addr_1	registro 1 de dirección de mem. local	0x5700_0018	Bits 9-0
dma_trsf_type	tipo de transferencia.	0x5700_001C	0x0000_0000 : lectura. 0x0000_0002 : escritura.
dma_status_reg_0	registro de estado 0.	0x5700_0024	Bit 0 : transferencia escritura finalizada. Bit 1 : transferencia lectura finalizada. Bit 2 : transferencia lectura en progreso. Bit 3 : transferencia escritura en progreso.
dma_status_reg_1	registro de estado 1.	0x5700_0028	Bits 9-0 : dirección de mem. local escritura. Bits 25-16 : dirección de mem. local lectura.
dma_wr_irq_clr	registro de interrupción de escritura	0x5700_002C	limpia pedido de interrupción por transf. escritura
dma_rd_irq_clr	registro de interrupción de lectura	0x5700_0030	limpia pedido de interrupción por transf. lectura

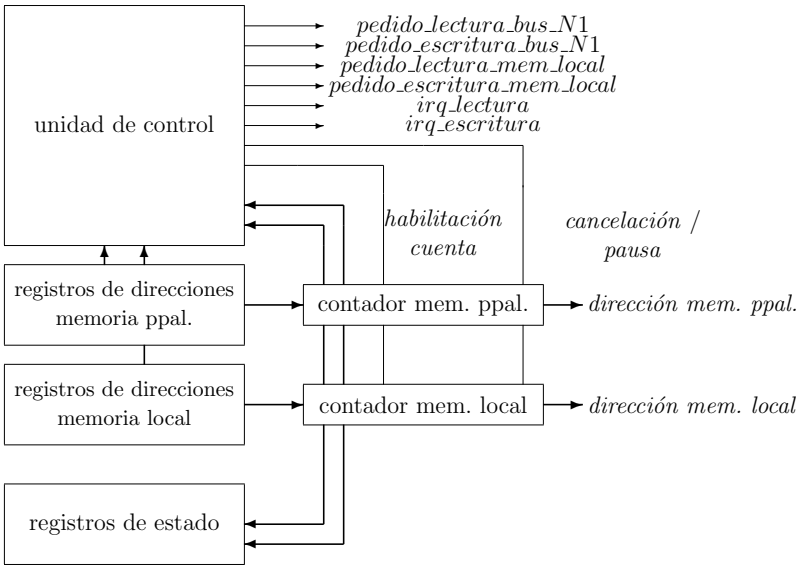


Figura 5.6: Diagrama en bloques de la unidad DMA.

está a cargo de pausar o establecer nuevamente a cero los contadores de direcciones lo que implica la cancelación de la transferencia. Además dispara las interrupciones de nivel del procesador cuando las transferencias fueron completadas.

Los registros de direcciones son los que indican las direcciones de inicio y fin de la transferencia. Los contadores generan las direcciones para ambas memorias utilizando el dato almacenado en estos registros. Cuando el pedido de lectura o escritura fue realizado, la unidad de control habilita el contador a incrementar su cuenta.

Los datos provenientes de la red son enviados a la memoria local y los que provienen de la memoria local son enviados al bus de la red.

5.1.3. Interfaces de datos

5.1.3.1. Interfaz con la NOC

La interfaz para la red N1 maneja las señales de sincronismo del bus de acuerdo al protocolo de 4 fases descripto en la Sección 3.1.1. Recibe los pedidos de lectura o escritura de la unidad DMA y presenta en el bus de la red N1 la dirección generada por la unidad DMA como así también el dato proveniente de la memoria local. En el caso de recibir un paquete de datos de la red, éste es enviado a la unidad DMA que a su vez escribe en la memoria local.

Esta interfaz no cuenta con registros de configuración y de estado ya que estos se implementaron en la unidad DMA.

En el caso de la red N2, el procesador se conecta a través de una interfaz que le permite recibir y enviar paquetes de 256 bits a la red. Esta interfaz expande las capacidades del procesador permitiendole configurar las unidades de procesamiento, recibir paquetes de sincronismo y de datos y generar los de reconocimiento.

En el caso de la recepción, la interfaz genera dos interrupciones de nivel dependiendo de la clase de paquete que se recibe, habiendo dos clases: de datos y de sincronismo. La interfaz recibe el paquete y lo almacena en 8 registros de 32 bits. Cuando el procesador recibe la interrupción ejecuta la rutina de servicio. La interfaz permite leer los 8 registros en forma independiente por lo que la rutina puede leer un número variable de estos dependiendo del caso. Una vez que se leyeron todos los datos necesarios el procesador limpia el pedido de interrupción.

Para la escritura, la interfaz permite que el procesador escriba 8 registros de 32 bits que conforman un paquete de 256 bits o sólo alguno de ellos. Además el procesador debe establecer la dirección de la unidad de procesamiento destino, y la dirección de un registro interno de la unidad. La interfaz también provee la posibilidad de enviar un paquete de datos, de control o de reconocimiento y genera un pedido de interrupción para indicarle al procesador que la escritura se realizó.

Tal como se expresa en los párrafos anteriores la interfaz cuenta con un juego de registros. Estos se encuentran mapeados a memoria partiendo de la dirección 0x53000000 para los registros asociados a la escritura y 0x54000000 a la recepción. La Tabla 5.3 presenta en detalle los mismos. Sumado a los registros de datos y de direcciones, la interfaz cuenta con un registro de habilitación de las interrupciones y tres registros adicionales para limpiar los pedidos de interrupción. La Fig. 5.7 muestra el diagrama en bloques de la interfaz N2.

Tabla 5.3: Tabla de registros de la interfaz N2.

Nombre de registros	**Descripción**	**Dirección bus AMBA**	**Valores**
n2_sd_pkt_word_0	Palabra 0 del paquete de envío.	0x5300_0000	
n2_sd_pkt_word_1	Palabra 1 del paquete de envío	0x5300_0004	

n2_sd_pkt_word_2	Palabra 2 del paquete de envío	0x5300_0008	
n2_sd_pkt_word_3	Palabra 3 del paquete de envío	0x5300_000C	
n2_sd_pkt_word_4	Palabra 4 del paquete de envío	0x5300_0010	
n2_sd_pkt_word_5	Palabra 5 del paquete de envío	0x5300_0014	
n2_sd_pkt_word_6	Palabra 6 del paquete de envío	0x5300_0018	
n2_sd_pkt_word_7	Palabra 7 del paquete de envío	0x5300_001C	
n2_data_pkt_send	Escritura en este registro escribe un paquete de datos en el bus.	0x5300_0020	
n2_ack_pkt_send	Escritura en este registro escribe un paquete de reconocimiento en el bus.	0x5300_0024	

n2_reg_addr	Dirección horizontal y vertical de la unidad de procesamiento.	0x5300_0030	Bits 4-0 : Dirección horizontal. Bits 7-5 : Dirección vertical.
n2_rd_pkt_word_0	Palabra 0 del paquete recibido	0x5400_0000	
n2_rd_pkt_word_1	Palabra 1 del paquete recibido.	0x5400_0004	
n2_rd_pkt_word_2	Palabra 2 del paquete recibido	0x5400_0008	
n2_rd_pkt_word_3	Palabra 3 del paquete recibido	0x5400_000C	
n2_rd_pkt_word_4	Palabra 4 del paquete recibido	0x5400_0010	
n2_rd_pkt_word_5	Palabra 5 del paquete recibido	0x5400_0014	
n2_rd_pkt_word_6	Palabra 6 del paquete recibido	0x5400_0018	
n2_rd_pkt_word_7	Palabra 7 del paquete recibido	0x5400_001C	

n2_rd_irq_en	Registro de habilitación de interrupciones.	0x5400_0024	Bit 0 : habilitación. si el valor es 0 las interrupciones están deshabilitadas.
n2_drd_irq_clr	registro de limpieza interrupción de recepción de datos.	0x5400_0028	
n2_ard_irq_clr	registro de limpieza interrupción de recepción de reconocimientos.	0x5400_002C	

5.1.4. Unidad QSPI

La unidad QSPI se incluyó para conectar el procesador M0 con una memoria no embebida del sistema. Esto permite enviar y recibir datos desde el M0 como también expandir la memoria disponible del sistema.

Esta unidad fue codificada por Daniel Mendat de la Universidad de Johns Hopkins. Su implementación física estuvo a cargo de Alejandro Pasciaro-

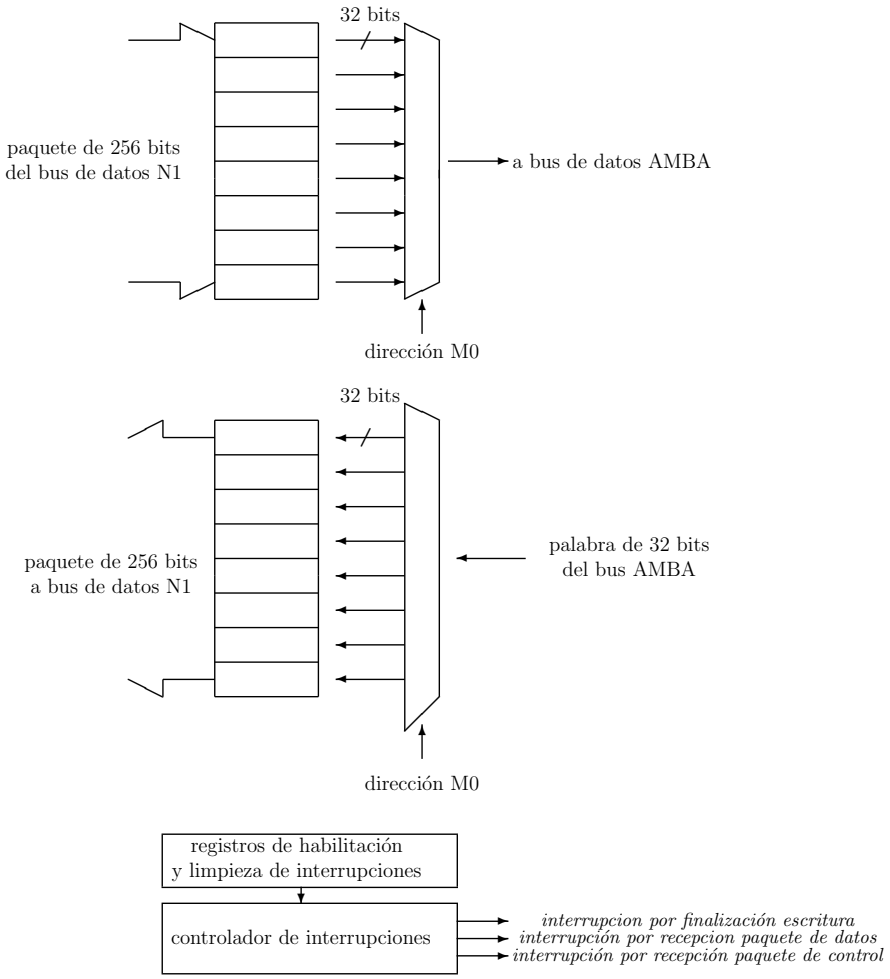


Figura 5.7: Diagrama en bloques de la interfaz N2.

ni. La unidad permite la comunicación a traves de 4 canales SPI en forma simultánea. El protócolo SPI [46] simple permite la comunicación serial y sincrónica entre dos dispositivos, en la cual uno de ellos es el maestro y el otro el esclavo. En este caso, el diseño implementado permite la comunicación simultánea entre cuatro dispositivos esclavos y un maestro.

Las señales que el protocolo utiliza son: MISO, MOSI, CS y SCK. La señal MISO y MOSI son de datos y el sentido de los datos es del esclavo al maestro y viceversa. La señal CS es la selección de los esclavos y SCK es el

reloj de la transmisón que se envía a los esclavos junto con los datos. Estas dos últimas señales son comunes a los 4 esclavos.

Para iniciar una transferencia el dispositivo maestro baja la señal CS y comienza a manejar la señal del reloj. Los datos son siempre leidos en el flanco descendente del reloj. Los datos se transmiten del bit más significativo al menos. Cuando la transferencia finalizó la señales de CS y SCK vuelven a su estado inicial.

Las memorias disponibles en el mercado que cumplen con los requisitos del sistema son los modelos Microchip 23A1024 y 23LC1024 SRAMs [47]. El procesador cuando inicia una transferencia envía en primer lugar la instrucción de lectura o escritura de 8 bits y luego la dirección de memoria de 24 bits a través de la señal MOSI. Si la operación es de lectura el esclavo responde con un byte de dato utilizando la señal MISO. Si la operación es de escritura el procesador envía el byte de dato.

En este caso el diseño contempla la comunicación con cuatro esclavos en forma simultánea. Las cuatro memorias comparten la misma señal de CS y SCK, la misma instrucción y dirección de configuración. De esta manera es posible realizar cuatro operaciones de lectura o escritura a la misma vez incrementando así la tasa de transferencia.

El procesador trabaja con palabras de 32 bits, esto permite escribir o leer 4 bytes a la misma vez. Si se quiere realizar una escritura el procesador debe escribir en registros mapeados a memoria para configurar la instrucción, la dirección y los 4 bytes. Si se quiere realizar una lectura, el procesador escribe en el registro de instrucción y de dirección y espera el dato de las memorias.

Cuando se completa una transferencia la unidad QSPI genera una in-

Tabla 5.4: Tabla de registros internos de la unidad de QSPI.

Registro	**Descripción**	**dirección memoria**	**Detalle**
instruction_spi	registro de instrucción	0x5800_0000	Bits 31-24.
address_spi	registro de dirección	0x5800_0000	Bits 23-0.
write_value_spi	registro de valor de escritura	0x5800_0100	
clear_read_irq_reg	registro de limpieza interrupción lectura	0x5800_1100	Bit 0.
clear_write_irq_reg	registro de limpieza interrupción escritura	0x5801_0000	Bit 0.

terrupción. Luego el procesador ejecuta la rutina de servicio que tomará distintas acciones dependiendo de si fue una operación de lectura o escritura lo que disparó la interrupción.

La Tabla 5.4 detalla los registros mapeados a memoria que la unidad posee.

5.1.5. Selección de reloj e interrupciones

El selector del reloj permite seleccionar una de las dos entradas de reloj disponibles en el sistema. Se diseñó de tal manera que la transición de una entrada a la otra suceda libre de fallas. Cuenta con tres entradas de control para seleccionar una de dos entradas posibles. Estas entradas son el reset general del sistema y dos señales de control provenientes del procesador. Las dos entradas de reloj son el reloj externo y el reloj proveniente de la red N2, siendo este último configurable. La salida del selector es la señal de reloj para el procesador y sus periféricos.

El modo de funcionamiento es el siguiente: cuando la señal de reset está en estado activo, el selector selecciona la fuente de reloj externo. Una vez

que se carga el programa a memoria, el procesador puede cambiar la fuente de reloj manejando las dos señales de control que tiene disponibles. Para ello el procesador realiza una escritura al registro de control del selector. Recordar que todos los periféricos conectados al AMBA bus están mapeados a memoria. La Fig. 5.8 muestra el diagrama temporal del selector.

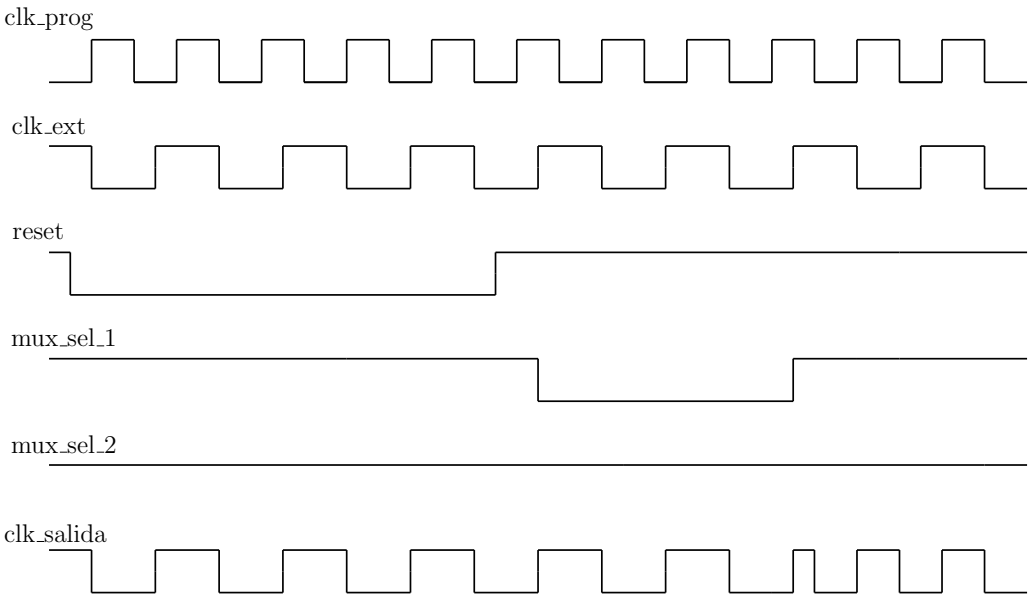


Figura 5.8: Diagrama temporal del selector de reloj.

El sistema cuenta con 9 interrupciones, todas los rutinas de servicios son cargadas a memoria a través de la UART por lo que son programables por el usuario. Las interrupciones se implementaron por nivel para evitar cualquier tipo de falla en el anidamiento.

La lista de interrupciones se detalla a continuación:

- interrupción por transmición de la UART finalizada.
- interrupción por lectura DMA finalizada.

- inerrupción por escritura DMA finalizada.
- interrupción por recepción de paquete de datos de la red N2.
- interrupción por recepción de paquete de control de la red N2.
- interrupción por recepción de paquete de reconocimiento de la red N2.
- interrupción por envío de paquete de la red N2.
- interrupció por lectuta SPI finalizada.
- interrupció por escritura SPI finalizada.

5.2. Unidad de multiplicación y suma

La unidad de multiplicación y suma realiza multiplicaciones vector-matriz. El tamaño del vector es de 16 elementos y la matriz es de 16×16. Cada componente tanto del vector como de la matriz es un número fraccional representado por 16 bits en punto fijo.

La Fig. 5.9 muestra la operación y el diagrama en bloques de la unidad. La multiplicación del vector D_i y la matriz de pesos W_i da como resultado un vector de salida O_i de tamaño 16. La unidad realiza 4 multiplicaciones vector-matriz en paralelo como se muestra en la Fig. 5.9(b).

La unidad puede cargar tanto la matriz de pesos W_i como el vector de entrada D_i utilizando la red N1 o N2. Una vez que se cargan todos los datos requeridos, la unidad realiza la multiplicación y los vectores resultantes O_i se envian a la interfaz de la red uno a la vez.

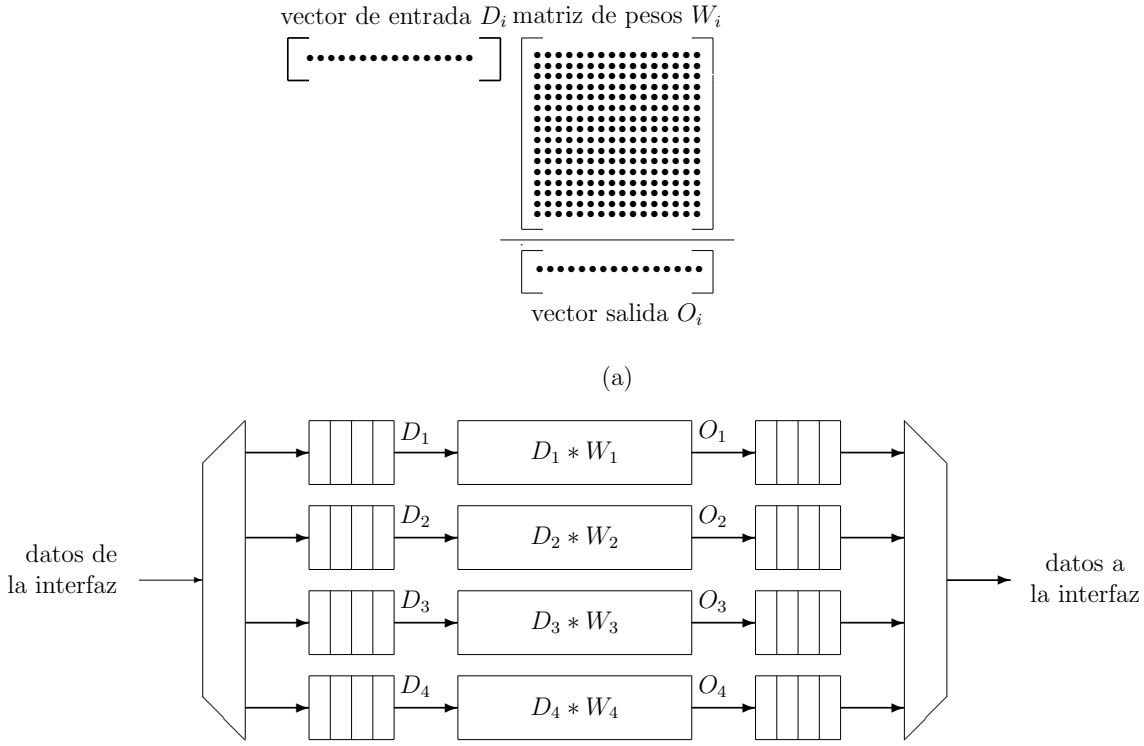


Figura 5.9: (a) Operación realizada por la unidad. (b) Diagrama en bloques.

La Fig. 5.10 muestra la micro-arquitectura de la unidad. Cada multiplicación del vector de entrada D_i y el vector columna W_{ji} se realiza en paralelo, donde los índices i, j se utilizan para notar la matriz y columna . El valor acumulado O_{ji} tiene una resolución de 36 bits de manera que se escala para resultar en un elemento de 16 bits. El escalado es completamente programable, esto implica que es posible seleccionar cualquier bit como inicial en el rango de bits [0:19].

La unidad realiza las multiplicaciones vector-matriz en 16 ciclos de reloj y el número de ciclos requerido para cargar un nuevo vector de entrada es variable dependiendo de la latencia de la red. La unidad opera a 100 MHz y su tasa de procesamiento estimada es de 25.6 Gbps considerando que la

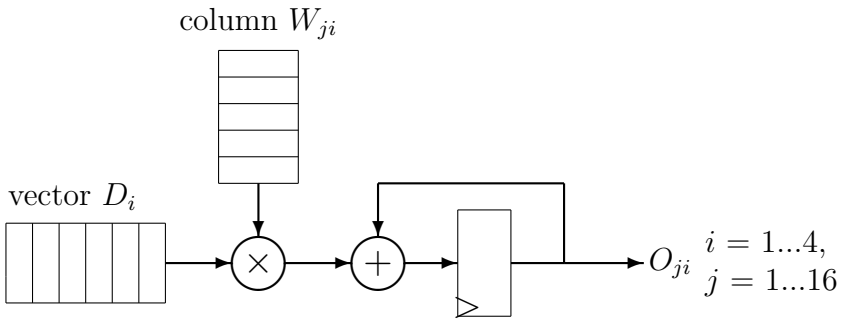


Figura 5.10: Arquitectura de la multiplicación vector -vector.

latencia de la red es nula.

Como se mencionó previamente los datos de entrada pueden provenir de la red N1 o N2. El ancho de palabra de ambas redes es de 256 bits por lo que un paquete tiene toda la información requerida para cargar un vector de entrada o un vector columna de la matriz. Por lo tanto se requieren 17 paquetes para cargar un vector de entrada y su correspondiente matriz de pesos.

Existen dos modos de carga de datos para el vector de entradas: (1) cargar el paquete de la red en forma completa o (2) cargar sólo los 16 bits menos significativos a una de las componentes del vector. El campo de dirección de registro de la red especifica el vector y la componente que va a cargarse. Los vectores columnas de la matriz de pesos se pueden cargar de acuerdo al primer modo. Se adicionó este segundo modo de carga para obtener una mayor flexibilidad en la utilización de la unidad.

Es interesante poder contar con un arreglo de estas unidades que permita la implementación de operaciones más complejas como es la transformada discreta de Fourier. El CMP cuenta con varias de estas unidades por lo que utilizando la red N2 es posible realizar este tipo de procesamiento. Recordar

que en el algoritmo de butterfly de la transformada discreta de Fourier [48] los datos de salida de una operación de butterfly se entrecruzan para luego ser procesados por la siguiente operación. Este cruzamiento se puede llevar a cabo utilizando este segundo modo de carga de datos.

En cuanto al formato de los vectores de salida se diseñó la unidad para tener un control completo del mismo. Esta permite enviar ya sea a un vector resultado completo o solo una componente del mismo. En este último caso, la componente a enviar ocupa los 16 bits menos significativos del paquete de la red. El campo de dirección de la red selecciona la componente a enviar. La unidad destino recibirá el paquete junto con la dirección.

El valor del campo dirección de la red establece el formato. Si se envia un vector completo a la red entonces los valores de este campo deben ser:

- $addr[13:12]$ selecciona el vector de salida O_j.
- $addr[4:0] = 0$.

Por otro lado, si se envia sólo una componente entonces:

- $addr[13:12]$ selecciona el vector de salida O_j.
- $addr[4:0]$ con valores en el rango [1:16] selecciona la componente a enviar.

La Tabla 5.5 muestra los registros internos de la unidad que tienen diversos propósitos: carga de datos, configuración del escalado y habilitación del cómputo.

Registros	Descripción	valor del campo dirección
W_{ij}	vector columna de la matriz de pesos	$waddr[14] = 0$. $waddr[13:12]$ selecciona la matriz W_i. $waddr[3:0]$ selecciona el vector columna W_{ij}.
D_i	vector de entrada carga completa	$waddr[14] = 0$. $waddr[13:12]$ selecciona el vector de entrada D_i. $waddr[11:0]$ =16.
D_i	vector de entrada carga parcial	$waddr[14] = 0$. $waddr[13:12]$ selecciona el vector de entrada D_i. $waddr[3:0]$ selecciona la componente a cargar.
Escalado	registro de escala	$waddr[14] = 0$. $waddr[13:12]$ selecciona la escala . $waddr[11:0] = 17$
habilitación	habilita el cómputo	$waddr[14] = 0$

Tabla 5.5: Registros de la unidad multiplicación y suma.

5.3. Implementación física

La implementación de este diseño se llevó a cabo en la tecnología CMOS 55nm Ultra low power. Se utilizó la herramienta Design Compiler de Synopsys para la síntesis lógica del bloque e Innovus de Cadence para la sintésis física. La frecuencia de operación máxima es 166 MHz. Las memorias se implementaron con bloques SRAM cuyo tamaño es de 64x32. Estos bloques se implementaron en la misma tecnología lo cual permite que el diseño completo pueda operar a un voltaje mínimo de 0.4 Volts. La Fig. 5.11 muestra la implementación física de los procesadores. Como se puede observar el sistema ocupa 5 columnas en la fila y utiliza 2 nodos de comunicación de los 5 que dispone. El área total que ocupa es de 5.76 mm^2, siendo las dimensiones de una columna de 1.28 mm x 0.9 mm.

La Tabla 5.6 muestra el área que ocupa la memoria caché, la memoria

bloque	**Area** $[mm^2]$
memoria caché	0.92
memoria local	4.22
M0CORTEX y periféricos	1.73

Tabla 5.6: Área que ocupa la jerarquía de memoria y el procesador CORTEXM0 y sus periféricos.

local y el procesador con todas sus unidades y periféricos incluidos (excepto la memoria local).

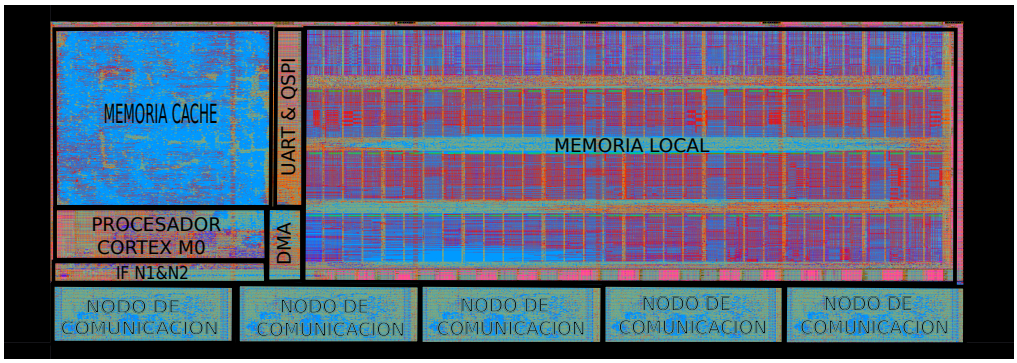


Figura 5.11: Implementación física del procesador CORTEX M0.

La Fig. 5.12 muestra la simulación por eventos del sistema. En esta simulación, el procesador realiza la configuración del resto de las unidades procesamiento. Se puede observar de la actividad en la señal *NETW_req_o* que el procesador envía paquetes de configuración a través de la red N2 utilizando su interfaz correspondiente. A su vez, una vez finalizado el procesamiento de las unidades, el procesamiento recibe paquetes de sincronización como se puede observar de las señales agrupadas en *DUT-N2_NETW_RD_IF*.

La Tabla 5.7 muestra la estimación de potencia del sistema utilizando la herramienta Voltus IC Power Integrity para un voltaje de operación de 1.08 V y una frecuencia de 166 MHz. Se utilizó una simulación por eventos para la

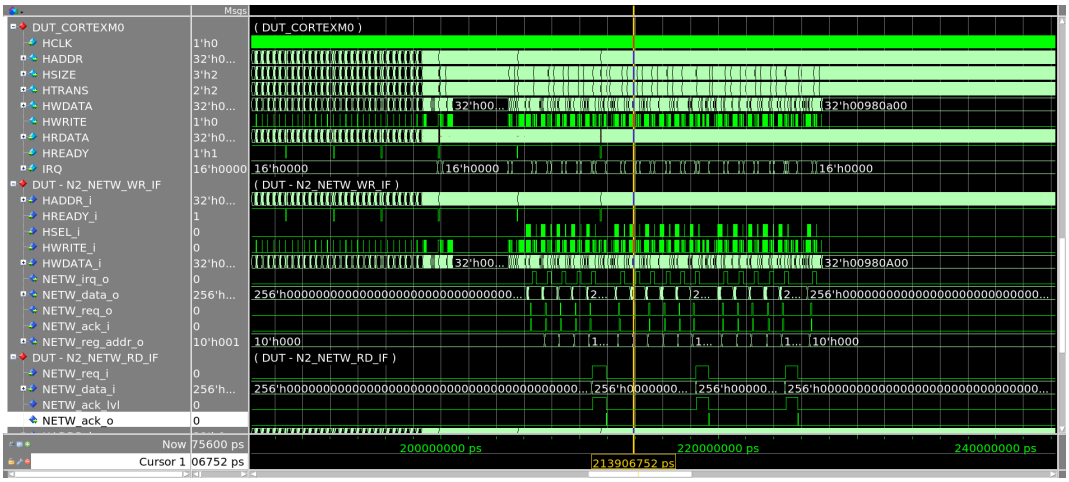


Figura 5.12: Simulación por eventos del sistema de propósito general.

bloque	**Potencia por Transiciones [mW]**	**Potencia Interna [mW]**	**Potencia por pérdida [mW]**	**Potencia Total [mW]**
memoria caché	60.77	35.25	1.54	97.58
memoria local	41.0	30.64	1.11	72.75
M0CORTEX y periféricos	126.11	78.54	2.95	207.61

Tabla 5.7: Consumo de potencia del sistema de propósito general basado en el CORTEXM0.

creación de los vectores de testeo. Estos vectores son utilizados para estimar la frecuencia de transición de las señales del sistema. En dicha simulación el sistema realiza la configuración de las unidades de procesamiento del sistema en chip involucradas en el reconocimiento de voz y la sincronización de las fases de procesamiento y comunicación de estas unidades.

La implementación de la unidad de multiplicación y suma se observa en la Fig. 5.13. La unidad ocupa una columna entera lo que equivale a un área de 1.15 mm^2. La frecuencia de operación máxima es de 100 MHz.

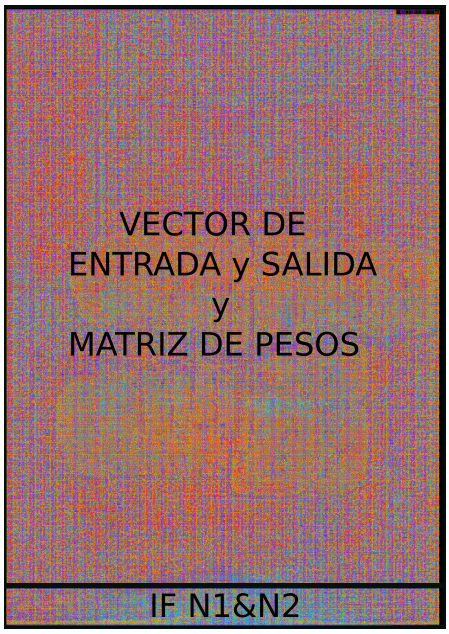


Figura 5.13: Implementación física de la unidad de multiplicación y suma.

La Fig. 5.14 muestra la simulación de la unidad. Se puede observar que los datos del vector y la matriz se reciben a través de la red N1 (ver señal *N1_PU_req_VDD_PU_1_i*). Una vez que los datos son procesados, tal como lo indica el fin del pulso de la señal por nivel *acc_en*, el resultado es enviado a través de la red N2 (ver señal *PU_N2_req_1_o*).

La Tabla 5.8 muestra la estimación de consumo para la unidad. Al igual que en los casos anteriores se realizó la estimación utilizando vectores de testeo generados a partir de la simulación mostrada en la Fig. 5.14.

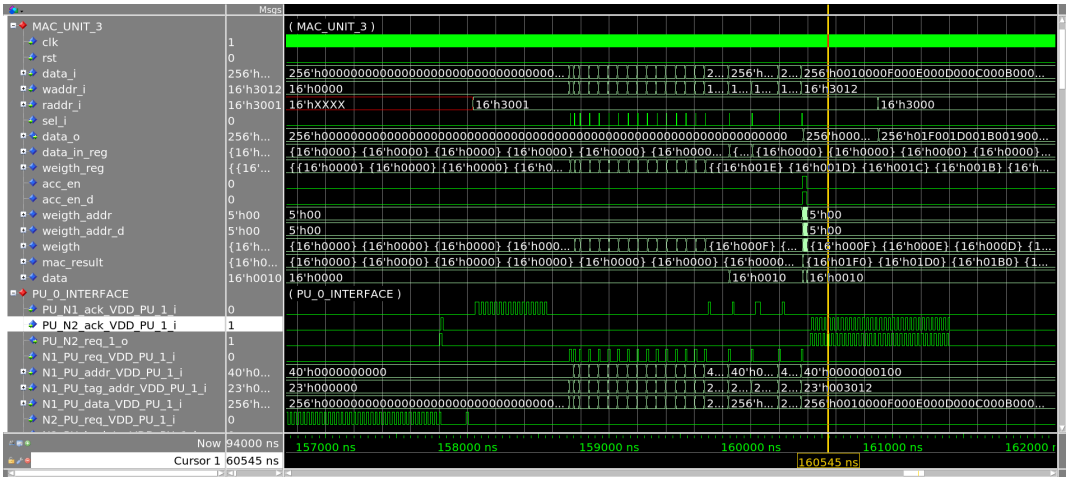


Figura 5.14: Simulación por eventos de la unidad de multiplicación y suma.

bloque	**Potencia por Transiciones [mW]**	**Potencia Interna [mW]**	**Potencia por pérdida [mW]**	**Potencia Total [mW]**
Unidad de multiplicación y suma	312.70	122.48	2.40	437.60

Tabla 5.8: Consumo de potencia de la unidad de multiplicación y suma.

5.4. Conclusiones

En esta sección se describió el diseño e implementación del sistema de proósito general basado en el procesador CORTEX M0. Como parte del arreglo de multiprocesadores, este sistema permite el cómputo de operaciones genéricas en 32 bits y además cumple la función de árbitro para el resto de las unidades de procesamiento. El procesador se encarga de configurar las unidades y de establecer las fases de procesamiento y comunicación entre ellas.

Se describió la arquitectura de los periféricos diseñados conectados a través del bus AMBA que integran el procesador a las redes en chip N1 y N2 y permiten la comunicación del procesador con el mundo exterior.

Se describió la jerarquía de memoria como estrategia para mitigar la latencia de la memoria principal. Esta jerarquía comprende una memoria local de 32 KBytes y una memoria caché de 2 KBytes. La memoria local se implementó con un arreglo de bloques de memoria SRAM de bajo voltaje. Se conecta a una unidad de acceso directo a memoria que permite realizar movimientos de datos entre la memoria principal y la memoria local con intervención mínima del procesador. La memoria caché se conecta a la memoria local a través de puertos dedicados y con el procesador a través del bus AMBA. Se diseño con un nivel de asociatividad 2 y un tamaño de bloque de 256 bits compatible con la memoria local y el bus de datos de la red N1.

Se diseño una unidad de multiplicación vector-matriz optimizada, que cuenta con un nivel de paralelismo que permite aprovechar al máximo el ancho del bus de datos de las redes en chip. Al disponer de este nivel de

paralelismo puede ser operada a voltajes reducidos y así obtener una arquitectura más eficiente desde el punto de vista del consumo de energía.

Los diseños descriptos se implementaron en un proceso VLSI de 55nm de Globalfoundries al igual que los núcleos de procesamiento de reconocimiento de voz. Se describe la implementación física en términos de su frecuencia y voltaje de operación, área de silicio que ocupa cada una de las unidades y estimaciones de consumo de potencia de las mismas.

Capítulo 6

Conclusiones

En esta tesis, se analiza el paralelismo como técnica de optimización de unidades de procesamiento para lograr puntos de operación eficientes en cuanto a tiempo de procesamiento y consumo de energía. El contexto de este análisis se da en un sistema en chip, en el cual las unidades de procesamiento implementadas integran un arreglo de procesadores. El sistema en chip, llamado CMP a lo largo de la tesis, cuenta con redes de comunicación integradas que permiten la interconexión de múltiples procesadores y una red en chip para la comunicación de las unidades de procesamiento con una memoria de datos masiva. La eficiencia en el consumo de energía se logra a través del paralelismo en sus diferentes niveles: a nivel micro-arquitectura y a nivel de CMP. Al disponer de varias unidades de procesamiento en paralelo, es posible operar a bajas frecuencias sin detrimento del rendimiento computacional. Se presenta la integración de una unidad de procesamiento con propósito específico, como es el reconocimiento de voz automático, que es computacionalmente intensiva y requiere de una gran cantidad de datos

para realizar su procesamiento. Es por esto que resulta interesante analizar la técnica de paralelismo como estrategia para lograr un procesamiento de datos eficiente ya que el algoritmo permite paralelizar una gran parte de las operaciones. De manera tal de poder explotar este paralelismo, se requiere de una gran cantidad de datos de memoria principal por lo que se requiere de estructuras que permitan un manejo eficiente de los datos. Esto mismo concepto se aplicó a unidades de de propósito general, donde nuevamente el diseño se focaliza en maximizar el paralelismo y en optimizar la comunicación con la memoria principal.

Se presenta un análisis de paralelismo a nivel micro-arquitectura y a nivel de sistema de uno de los bloques de procesamiento que componen el procesador para reconocimiento de voz. Se proponen dos maneras de paralelizar el camino de datos interno de la unidad de procesamiento y se analizan ambas revelando sus ventajas y desventajas. Este análisis revela la importancia de tener una jerarquía de memoria eficiente y cuan importante es tener los datos almacenados localmente. Por otra parte, para realizar el análisis a nivel sistema, se utilizan funciones de costo en tiempo de procesamiento y energía para obtener un número de procesadores óptimo a instanciar que minimiza la energía y el tiempo de procesamiento teniendo en cuenta la comunicación con la memoria principal.

Se presentan dos implementaciones en dos tecnologías de integración: 55 nm y 16 nm. En el proceso de 55 nm se implementaron todas las unidades de procesamiento junto con sus nodos de comunicación para las redes integradas en el sistema. En el caso del proceso de 16 nm, se implementó la unidad de procesamiento de GMM para funcionamiento aislado. Todas las imple-

mentaciones se encuentran descriptas en términos de frecuencia y voltaje de operación, simulaciones por evento, área de silicio y estimaciones de consumo de potencia de cada bloque de procesamiento.

El futuro, a largo plazo, nos indica que debemos construir unidades de procesamiento simples y optimizadas, que trabajen en paralelo y que se encuentren fuertemente interconectadas y que además sean capaces de almacenar datos localmente. De esta manera se reduce el tiempo de procesamiento y el consumo de energía. Esto es lo que grupos de investigación de universidades y companías tecnológicas tales como IBM y Google están persiguiendo. Un ejemplo de una arquitectura digital que va en esa dirección es el procesador Tensorflow [49] y TrueNorth [50]. A mediano y corto plazo es posible utilizar sistemas de múltiples procesadores conectados a memorias de almacenamiento masivo compuestas por múltiples capas de almacenamiento con múltiples canales de comunicación. Dichas memorias están siendo actualmente construidas [51]. Si las unidades de procesamiento que componen el CMP cuentan con la capacidad de predecir los parámetros que va a utilizar en futuras operaciones de procesamiento con una buena precisión, se podrán entonces construir sistemas de procesamiento más eficientes en tiempo de procesamiento y energía. Sin embargo, el rendimiento final de este tipo de arquitectura depende fuertemente de la aplicación para la cual se diseñe el sistema.

Apéndice

.1. Arquitectura del procesador para el reconocimiento de voz automática

La arquitectura utilizada en esta tesis se basa en la propuesta de [1], que se planteó para una tecnología 3D y resuelve la implementación del algoritmo de reconocimiento de voz automática implementado las tres etapas de procesamiento previamente mencionadas. Utiliza un modelo de cadenas de Markov de tres estados a nivel fonema y sumatorias de distribuciones Gaussianas para las probabilidades de observación. La Figura 1 muestra las capas de procesamiento que componen la estructura. El diseño cuenta con dos niveles de silicio particionados de la siguiente manera: el nivel superior tiene una memoria DRAM y su controlador que permite transferencias a tasa doble y la interfaz acústica que realiza la extracción de las características acústicas. El nivel inferior tiene dos procesadores dedicados al cómputo de las distribuciones Gaussianas mixtas y a la búsqueda de los fonemas más probables utilizando el algoritmo de Viterbi. Estos procesadores se denominan GMM y HMM (del inglés Gaussian Mixture Model y Hidden Markov Model). Además

este nivel cuenta con una memoria caché en la cual se almacenan las evaluaciones de las mixturas de distribuciones Gaussianas. La integración se realizó a nivel bloque, es decir que los bloques de procesamiento completos fueron ubicados en diferentes niveles de obleas. La conexión de estos niveles entre si se realiza a través de vías 3D, que permiten lograr anchos de buses superiores a 128 bits [23].

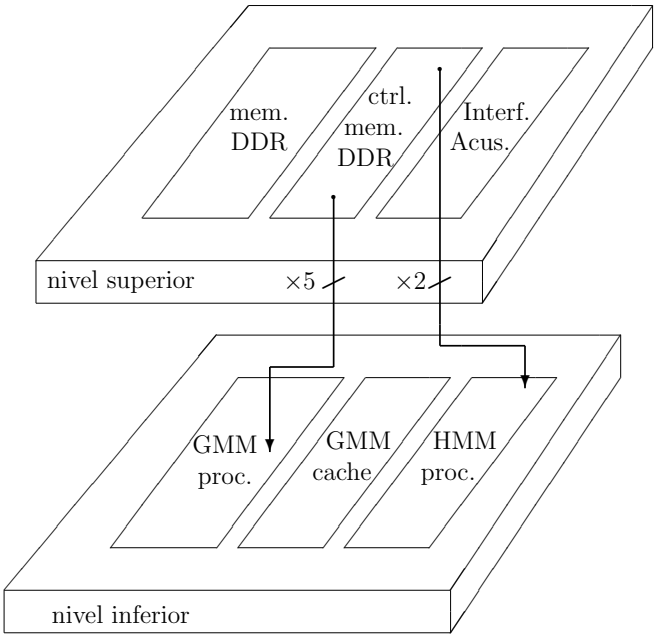


Figura 1: Estructura 3D de la arquitectura propuesta en [1].

.1.0.1. Interfaz acústica

La interfaz acústica convierte una señal variante en el tiempo a una representación en el dominio frecuencia. La características que la interfaz extrae se denominan CCFM como se detalló anteriormente. El procesamiento de extracción de estas características comprende las siguientes operaciones [14]:

- Removedor de la componente de DC de las muestras entrantes.

- filtrado pre-enfasis enfatiza o amplifica componentes de alta frecuencia de la señal.

- procesamiento FFT genera las componentes espectrales de la señal. Cada componente es una medida de la energía de las frecuencias que conforman dicha componente.

- Mapeo a la escala de frecuencias de Mel integra las componentes espectrales utilizando ventanas triangulares. Ests ventanas están espaciadas linealmente hasta 1 KHz y luego en forma logaritmica. Se realiza este tipo de espaciamiento para aproximar la resolución frecuencial del oido humano.

- transformada de coseno discreta. Este paso se realiza para decorrelacionar las componentes espectrales, lo cual resulta importante para el modelado de las distribuciones Gaussianas en particular por la matriz de covarianza. Además se truncan las componentes de alta frecuencia quedando así un vector de coeficientes espectrales de longitud típica 12.

- Finalmente se computa la velocidad (Δ) y aceleración ($\Delta\Delta$) de la serie temporal de coeficientes. Esto se realiza para adicionar una dinámica temporal a las características extraídas.

La Fig. 2 muestra el diagrama en bloque de la interfaz acústica.

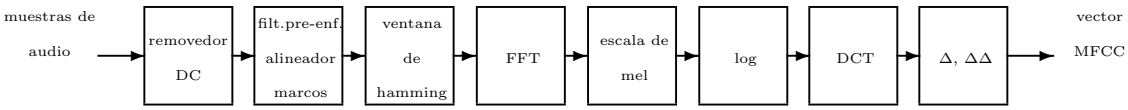


Figura 2: Diagrama en bloques de la interfaz acústica

.1.0.2. Protocolo de marcos de datos

Es comun en el procesamiento de audio que el procesamiento frecuencial se realice sobre marcos de datos, los cuales se generan agrupando muestras de audio en un intervalo temporal de 10 milisegundos. Los marcos de datos se transmiten a través del flujo de procesamiento en forma secuencial con una señal de sincronismo que indica el inicio del marco.

A lo largo del flujo de procesamiento la longitud del marco es variable, la cual guarda una relación con el tipo de operación que se efectua sobre los datos. La Tabla .1.0.2 muestra la longitud del marco de datos a lo largo del flujo.

Tabla 1: Longitud de marco de datos en el flujo de procesamiento.

Etapa	**Longitud del marco de datos saliente**
alineador de marcos	480
FFT	512
escala de mel	40
log	40
DCT	13
Δ, $\Delta\Delta$	39

La interfaz acústica provee un vector CCFM cada diez milisegundos. Sin embargo, de modo de contar con más información en el tiempo, se trabaja con un marco de datos de 30 milisegundos. Para lograr esto, se desplaza el marcos de datos cada diez milisegundos. Esta operación se muestra en la Fig.

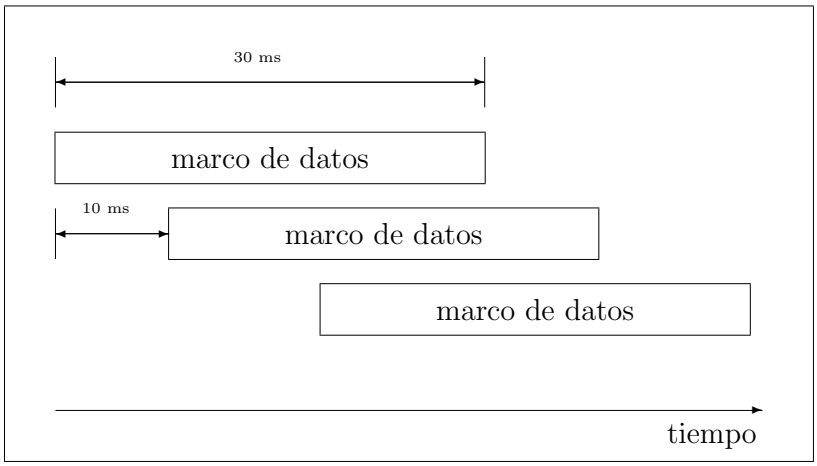


Figura 3: Solapamiento temporal de los marcos de datos.

3.

Considerando una frecuencia de muestreo de audio de 16 KHz, la cantidad de muestras en un intervalo de 30 ms viene dado por la siguiente ecuación:

$$\frac{30ms}{1/16KHz} = \frac{30ms}{62{,}5\mu s} = 480 \tag{1}$$

Este marco se extiende con ceros de manera de poder aplicar una transformada rápida de Fourier de 512 puntos. El bloque que realiza el filtrado en la escala de Mel reduce el marco de datos a 40 y luego de aplicar la transformada de coseno discreta el largo del vector se reduce a 13. Por último, el bloque que computa la diferencia temporal incrementa el largo en 13. Dado que hay dos de ellos en tandem, la longitud final del vector resulta de 39.

De manera tal de realizar el procesamiento de los 3 marcos de datos se utiliza una FIFO para realizar el alineamiento de los mismos. Cuando tres marcos de datos se encuentran almacenados se procede a la lectura de la FIFO. Esta se implementa como un buffer circular con capacidad de alma-

cenamiento para 4 marcos. Al inicio del procesamiento se debe esperar el almacenamiento de 3 marcos y el puntero de lectura se encuentra en la posición cero. Luego el puntero de lectura se establece en el inicio del próximo marco. De esta manera despueés de cada escritura de un marco, es posible leer 3 marcos de datos solapados en el tiempo. En la Fig. 2 este buffer se denomina alineador de marcos.

La Fig. 4 muestra la posición de los punteros de escritura y lectura en dos casos: luego de la escritura de los tres marcos iniciales (m1, m2 y m3) y luego de la escritura del marco siguiente (m4).

.1.0.3. Módulo GMM

.1.0.4. Árbol aritmético

El árbol aritmético realiza el cómputo del logaritmo de la distribución Gaussiana. El exponente $\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{(}-1)(\mathbf{x}-\mu)$ requiere de 39 sumas, 78 multiplicaciones y un desplazamiento (asumiendo que la matriz de covarianza Σ es diagonal). La arquitectura que se implementó es la que se muestra en la Fig. 5.

La entrada al módulo GMM es un vector de dimensión 39. De la memoria

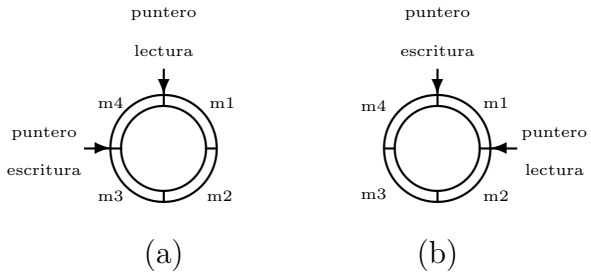


Figura 4: Posición de los punteros en el buffer circular: (a) Luego de la escritura de los 3 marcos inciales. (b) Luego de la escritura del próximo marco.

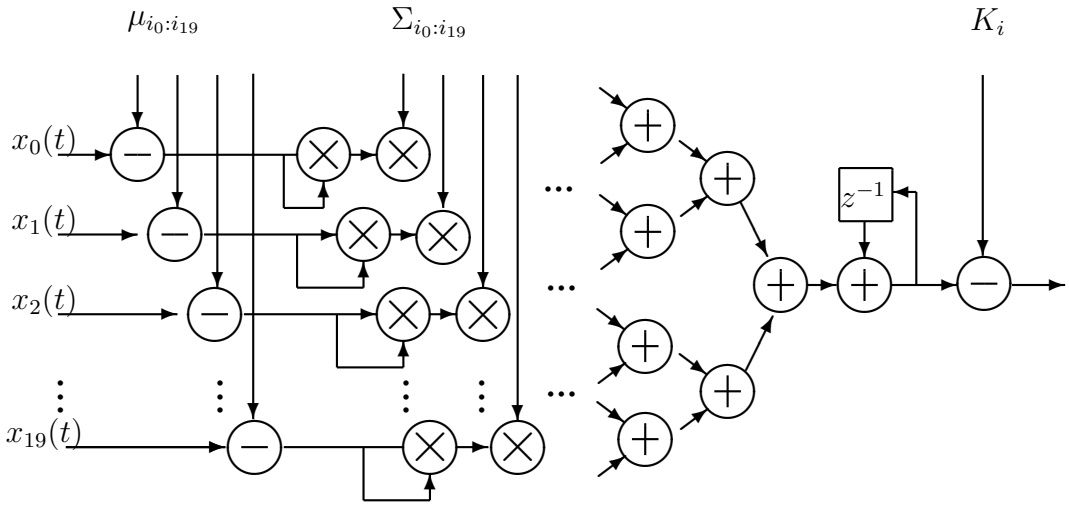


Figura 5: unidad aritmética para el cálculo en paralelo de la parte exponencial de la función gaussiana.

es posible leer en forma simulatánea 5 puertos lo que da una transferencia de 5x128 bits que es igual a 40 * 16 bits. Cada parámetro se representa con 16 bits por lo que es posible leer 40 parámetros (μ, Σ) de la memoria por ciclo. Para realizar el cómputo de la Gaussiana en forma totalmente paralela son necesarios 79 parámetros por ciclo. Dado que el ancho de banda de la memoria permite sólo 20 parámetros por ciclo se computa una distribución Guassiana cada dos ciclos de reloj. El árbol de sumadores está implementado en 5 niveles separados por registros.

El resultado del cómputo de cada Gaussiana es la entrada de datos de la unidad de ordenamiento y a la memoria caché GMM.

.1.0.5. Unidad de ordenamiento

La unidad de ordenamiento crea una lista de ordenamiento de longitud N. Los datos de entrada son el valor de la guassiana y una etiqueta que diferen-

cia cada gaussiana. La lista se genera ordenando los valores de las gaussianas. Los datos que entran a la unidad se comparan con los valores previamente ordenados y se insertan en la lista por debajo del valor inmediatamente superior y por arriba del valor inmediatamente inferior. Estas comparaciones se realizan en paralelo por lo que el tiempo de ordenamiento es lineal. El sorter que se implementó tiene la capacidad de ordenar hasta 64 valores. La cantidad de valores a ordenar es configurable.

.1.0.6. Cómputo de distribuciones Gaussianas en formato árbol

El módulo HMM asume que el módulo GMM provee todas las probabilidades de observación requeridas para cada marco de datos. Esto requiere el cómputo de la probabilidad de observación del vector **x** para cada distribución gaussiana en el sistema. Esto puede ser llevado a cabo pero una mejor estrategia es utilizar una organización de cómputo tipo árbol. En este método se computa un primer grupo de Gaussianas con un nivel de detalle reducido llamado nivel 1. Luego la unidad de ordenamiento provee las 64 Gaussianas de mayor valor en este nivel. Las gaussianas resultantes se computan con más detalle en un segundo nivel llamado nivel 2.

La Fig. 6 muestra la jerarquía de procesamiento para el caso de que la unidad de ordenamiento provea las 3 gaussianas de mayor valor: g_1, g_4 y g_N. Cada una de las gaussianas seleccionadas apuntan a un grupo de gaussianas en el nivel 2: $[g_1^1 : g_1^{16}]$, $[g_4^1 : g_4^{16}]$ y $[g_N^1 : g_N^{16}]$ y se re-evaluan utilizando el módelo provisto por las 16 gaussianas. Utilizando este método sólo las gaussianas más relevantes son evaluadas utilizando un módelo de gaussianas mixtas con más detalle. Al resto de las gaussianas se le asigna el valor compu-

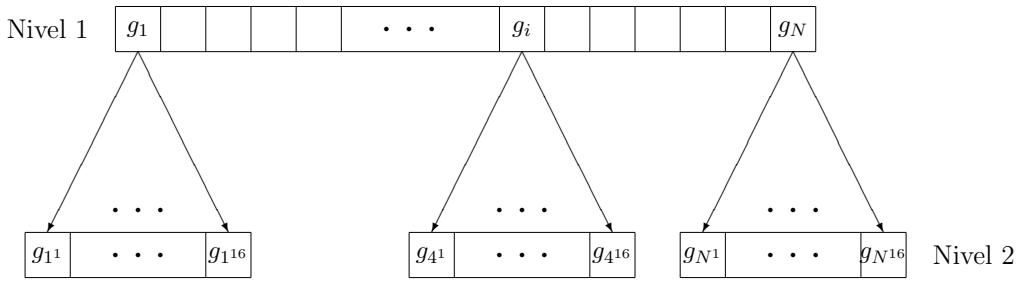


Figura 6: Jerarquía de procesamiento de las gaussianas

tado en el nivel 1 o un valor constante de respaldo. Esto última asignación depende del modo de operación que se elija.

.1.0.7. Diagrama temporal del módulo GMM

La Fig. 7 muestra el diagrama temporal del módulo GMM. La interfaz acústica trabaja por marcos de muestras de audio. La interfaz devuelve un vector MFCC por cada marco de datos recibido. Con dicho vector el módulo GMM computa el valor de las gaussianas en el formato árbol previamente descripto. Las operaciones de lectura de memoria, evaluación aritmética y ordenamiento se solapan en el tiempo dado que se implementaron en modo pipeline.

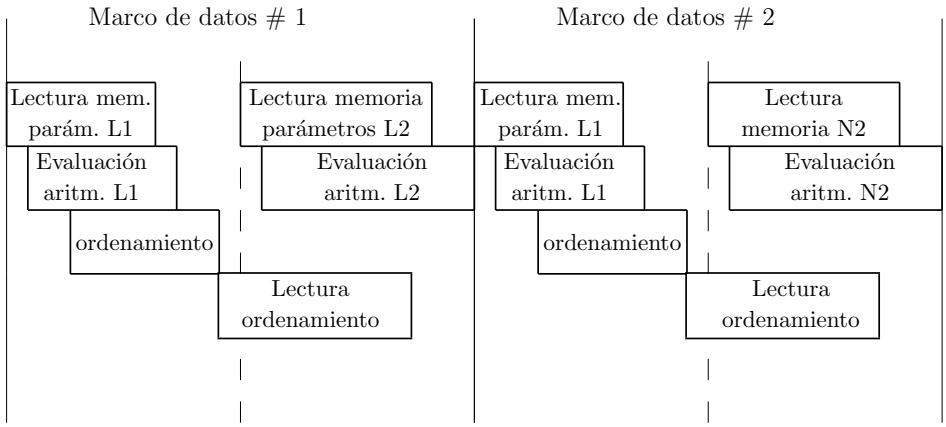


Figura 7: Diagrama temporal del módulo GMM.

.1.0.8. Memoria caché de probabilidades

La memoria caché se incluyó en el diseño para incrementar la tasa de procesamiento. En vez de computar distribuciones Gaussianas en función de la demanda del módulo HMM, estas se computan una única vez y luego son almacenadas en memoria. El módulo HMM tiene entonces disponibles en memoria los valores computados para un marco de datos. De esta manera si se requiere dos veces del mismo valor de gaussiana no es necesario que el módulo GMM vuelva a recomputar dicho valor.

La memoria caché consiste de dos grupos de bancos de memoria trabajando en formato ping-pong como se muestra en la Fig. 8 (a). Esto se hizo así con el objetivo de incrementar el paralelismo y tener acceso a cuatro valores de probabilidad en forma simultánea. El mismo dato del módulo GMM se almacena en paralelo en los cuatro bancos y cada uno de ellos puede leerse en forma paralela e independiente. De esta manera el módulo HMM tiene la capacidad de evaluar múltiples valores de gaussianas en paralelo. La memoria posee entonces un puerto de escritura y cuatro de lectura.

La operación de los bancos en formato ping-pong, como se mencionó anteriormente, permite que en un mismo marco de datos, los valores almacenados en memoria puedan leerse de un grupo de bancos mientras el módulo GMM escribe los nuevos valor del próximo marco de datos en el otro grupo. Cuando el módulo HMM evalua el próximo marco de datos el rol de las grupos se intercambia.

Cada banco está a su vez subdivido en dos: una mitad está destinada al almacenamiento de valores L1 y la otra al almacenamiento de valores L2 como se muestra en la Fig. 8 (b). Cada banco de memoria tiena capacidad de almacenar 256 valores L1 y otros 256 valores L2. La capacidad es de 8 Kbits por banco, por lo que la capacidad total de la memoria es de 64 Kbits.

Cuando el módulo GMM escribe el dato en memoria debe indicar a los bancos de memoria a que nivel de cómputo corresponde. Los bancos poseen un vector de bit válido de manera de poder hacer un seguimiento de aquellas gaussianas que fueron computadas en el nivel 2 como así también un puntero local a los valores del nivel 2 almacenados. Cuando el módulo HMM realiza una lectura y coloca un índice de gaussiana en el bus, primero se verifica que haya un valor de nivel 2 disponible. De ser así utiliza los bits superiores del índice para obtener el puntero local del nivel 2 con el cual direcciona el banco del nivel 2. De no existir un valor disponible del nivel 2, entonces se direcciona el valor almacenado en el banco del nivel 1. Si el valor buscado no se encuentra almacenado en la memoria se presenta un valor de respaldo configurable en el bus de datos.

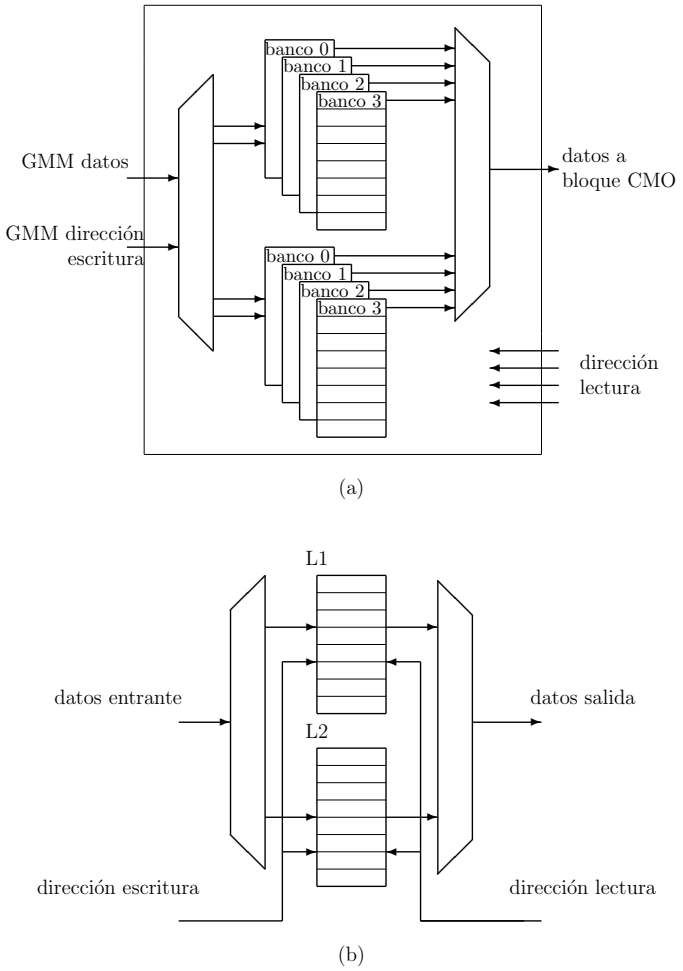


Figura 8: Arquitectura de memoria caché para almacenamiento de los valores de probabilidad computados por el módulo GMM.

.1.0.9. Módulo HMM

La unidad aritmética utiliza los datos recibidos de la memoria caché de probabilidades junto con la probabilidad de transición para actualizar la probabilidad de los estados de acuerdo a la Ecuación 2.6. Luego la unidad de ordenamiento utiliza la probabilidad actualizada de los estados para crear una lista ordenada de los mismos. Cuando la cola de ordenamiento está vacia y todos los estados fueron ordenados el procesamiento finalizó.

Una vez que la memoria caché de probabilidades tiene almacenados los valores correspondientes a un marco de datos nuevo, se inicia una nueva iteración en el lazo de procesamiento. Esto implica que la unidad de ordenamiento carga la lista ordenada en la cola y el procesamiento previamente descripto se repite.

La unidad aritmética nula actualiza la probabilidad de los estados nulos. La arquitectura cuenta con dos unidades artiméticas debido a que, a diferencia del resto de los estados, la actualización del estado nulo no requiere de la probabilidad de observación. El fonema cuyo estado actual es el nulo, es enviado a la unidad resultado que modifica el formato de los datos y envia el dato modificado a la cola de resultados y a la cola de hipótesis. La unidad de hipótesis recibe los datos de esta cola e inserta 64 nuevos estados iniciales en la cola de ordenamiento.

La unidad de ordenamiento consiste en un arreglo lineal de comparadores. El orden de complejidad es lineal en tiempo y espacio. Como resultado la unidad devuelve un arreglo ordenado de 64 fonemas comenzando por el de mayor probabilidad de estado. Esta unidad tiene un registro por cada etapa de ordenamiento por lo que permite una tasa de procesamiento de un dato por ciclo. Permite también el solapamiento de la operación de lectura y ordenamiento, de modo tal que el ordenamiento del próximo marco de datos puede comenzar mientras aún se este completando la lectura del marco anterior.

La profundidad de la cola de ordenamiento define cuantos estados pueden ser procesados en una iteración. La Tabla 2 muestra las distintas colas y su tamaño.

Cola	**Tamaño [Kbits]**	**Ancho de palabra [bits]**	**Profundidad**	**# estados**
Ordenamiento	25.25	101	256	256
Salida	5	80	64	64
Hipótesis	5	80	64	64

Tabla 2: Tamaño de las colas en el módulo HMM.

La Fig. 9 muestra la evolución en el tiempo de las operaciones que realiza el módulo. Las operaciones se solapan dado que las unidades que componen al módulo están implementadas en modo pipeline. A medida que se lee la unidad de ordenamiento, se busca a memoria los datos necesarios para la actualización de la probabilidad de los estados. Se realiza la evaluación aritmética y en caso de que alguno de los estados procesados sea nulo se procede a la actualización de las hipótesis. Por último, se realiza el ordenamiento de los estados.

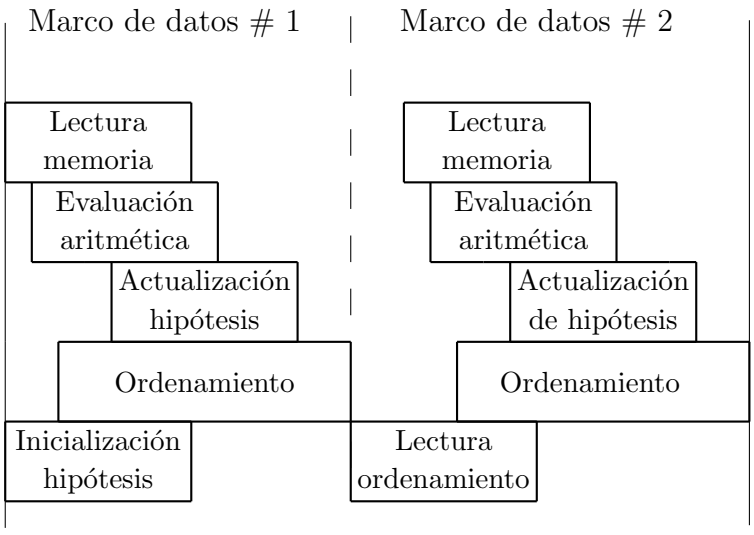


Figura 9: Diagrama temporal del módulo HMM.

.2. Descripción de las redes en chip

La red N1 esta destinada para la comunicación entre los procesadores y la memoria principal DDR, con una frecuencia de operación de 300 MHz. La red es del tipo *token ring* y consiste en ocho anillos que operan en forma independiente en cada fila (ver Fig. 3.2). Posee canales de comunicación separados para los comandos y para los datos. Cada columna puede realizar un pedido de lectura o escritura a memoria cada 32 ciclos de reloj. Esto es asi debido a que entre columnas hay dos registros de manera de que la red pueda operar a la frecuencia especificada. Existe una interfaz de memoria que toma el pedido de estos anillos y responde a los mismos en forma secuencial. Si por alguna razón los pedidos no son respondidos por la interfaz, el pedido en la unidad de procesamiento se mantiene vigente hasta que recibe una respuesta satisfactoria. De esta manera la interfaz de las unidades de procesamiento con el anillo resulta menos compleja. La memoria permite un ancho de banda máximo de 320 Gbps a una frecuencia de operación de 1.6 GHz y un paquete de datos de 256 bits. La interfaz con la memoria posee un ancho de banda máximo de 9.6 Gbps. Por último cada anillo posee un ancho de banda máximo teórico de 1.2 Gbps.

La red de comunicación N2 esta destinada para la comunicación entre unidades de procesamiento. Es una malla en donde cada slot de las filas tiene acceso a los cuatro vecinos más cercanos (dirección este, oeste, norte y sur). La malla es una red del tipo destino. Esto implica que las unidades de procesamiento reciben paquetes sin conocimiento de la fuente de envío. Para el envío de paquetes la unidad de procesamiento tiene una dirección

de destino configurable. La Fig. 10 muestra la red junto con el arreglo de procesadores y sus nodos de comunicación que le brindan a las unidades de procesamiento acceso a ambas redes. El ruteo de los paquetes a través de la red se logra sin buffers intermedios lo que asegura que ningún paquete es descartado y todos llegan a destino. Al no tener buffers o colas intermedias, los paquetes puede que eventualmente sean desviados de su ruta óptima. El tamaño del paquete es de 256 bits.

Si bien hay unidades que ocupan más de una columna, éstas aún tienen disponible el nodo de comunicación de cada columna que ocupan. Tal es el caso de los módulos de reconocimiento de voz, las memorias caché y los procesadores CORTEXM0.

La comunicación de una unidad de procesamiento con la memoria principal se realiza utilizando las señales de datos, dirección y sincronismo de la red N1 detallada en la Tabla 3. La red cuenta con un bus de 16 bits para etiquetar el pedido. Luego cuando el dato pedido llega a la unidad de procesamiento tiene asociada dicha etiqueta. Esto es útil para almacenar los datos en determinados registros de la unidad de procesamiento. El bus de datos posee 256 bits y los datos se leen o escriben utilizando el protocolo de comunicación de 4 fases dado que la velocidad de procesamiento de las unidades puede ser diferente a la de la red. Este protocolo es útil cuando se comunican dos unidades que operan en dominios de reloj distintos ya que asegura que los datos en el bus permanecen estables hasta que el receptor y emisor adquieren dichos datos correctamente. La Fig. 11 muestra el comportamiento de la señales de acuerdo al protocolo mencionado. El emisor coloca el dato en el bus y establece una senãl de pedido de transacción. El receptor sensa el cam-

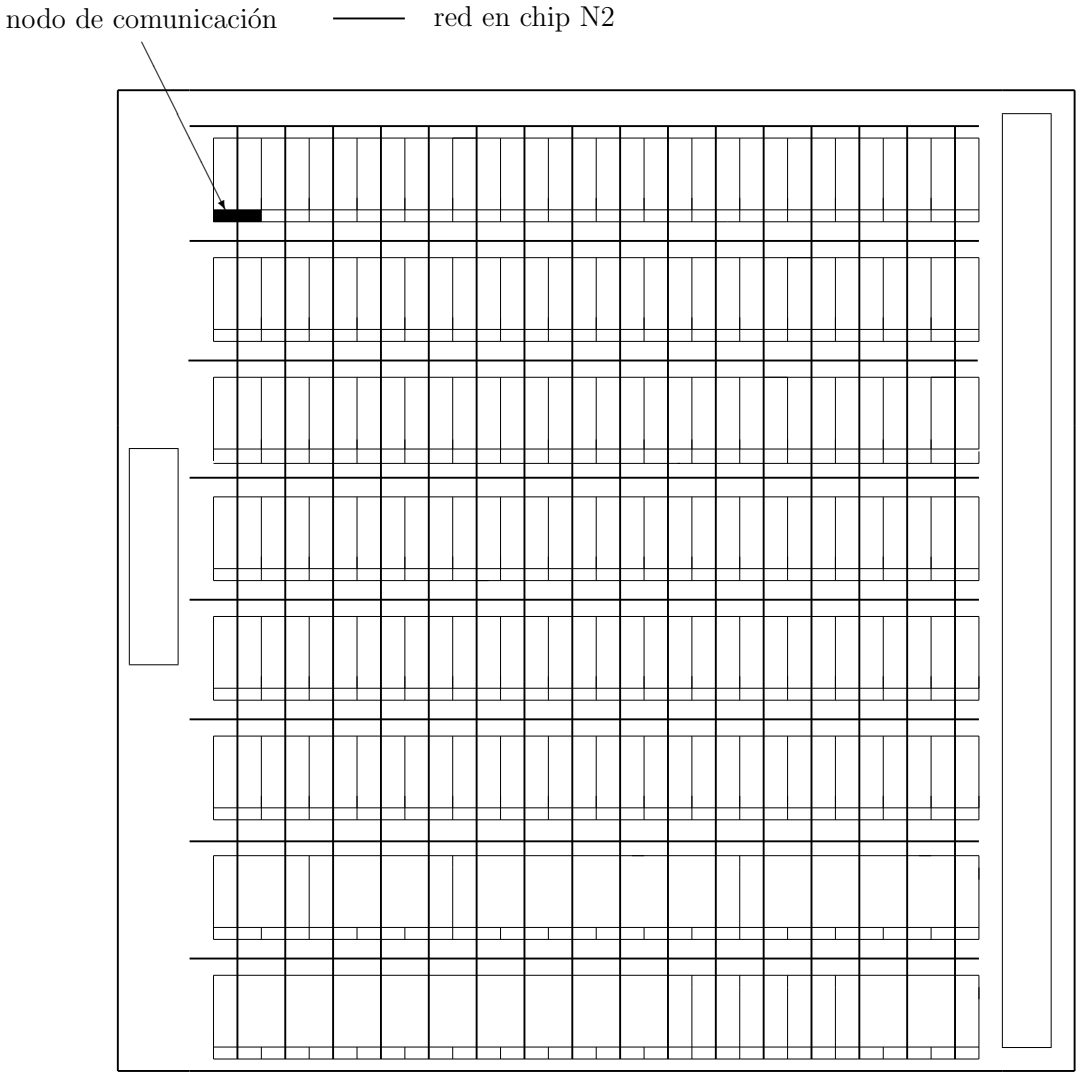


Figura 10: Red en chip N2 junto con arreglo de unidades de procesamiento. Se muestra el nodo de comunicación adjunto a cada unidad.

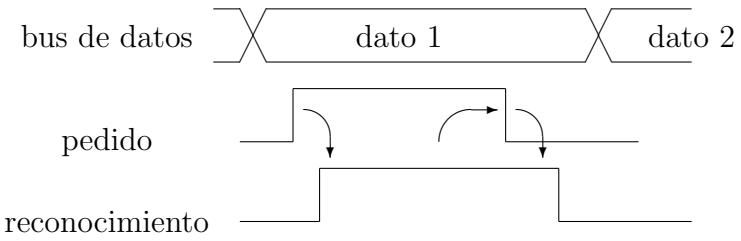


Figura 11: Protocolo de comunicación de 4 fases.

bio de estado en dicha línea, adquiere el dato del bus y establece la señal de reconocimiento. Cuando el emisor recibe la senãl de reconocimiento establece nuevamente la señal de pedido a su estado inicial. Finalmente cuando este último evento sucede el receptor también establece a su estado inicial la señal de reconocimiento y el emisor puede iniciar una nueva transacción. En este caso, el receptor y emisor pueden ser tanto el nodo de comunicación como la unidad de procesamiento dependiendo de quien inicie la transferencia.

La comunicación entre las unidades de procesamiento se realiza a través de las señales disponibles en la red N2 que se detallan en la Tabla 4. Esta red utiliza el mismo protocolo de comunicación de la red N1. Se utiliza un bus de dirección de registro de 16 bits para seleccionar los registros internos de la unidad de procesamiento destino. La unidad de procesamiento destino es direccionada en la malla utilizando un bus de dirección horizontal y vertical. Además la red N2 también distribuye el reset y el reloj programable para las unidades de procesamiento.

Red	**Bus**	**Nro. de bits**	**Descripción**
		Datos entrantes	
N1	*n1_data_i*	256	bus de datos.
N1	*n1_tag_i*	16	bus de etiquetas para identificar los pedidos a memoria.
N1	*n1_req_i*	1	señal de pedido entrante.
N1	*n1_ack_o*	1	señal de reconocimiento del pedido entrante.
		Datos salientes	
N1	*n1_data_o*	256	bus de datos.
N1	*n1_tag_o*	16	bus de etiquetas para identificar los pedidos a memoria.
N1	*n1_req_o*	1	señal de pedido saliente.
N1	*n1_ack_i*	1	señal de reconocimiento del pedido saliente.
N1	*n1_addr_o*	40	bus de direcciones de memoria.
N1	*n1_cmd_o*	2	tipo de transacción (lectura o escritura).

Tabla 3: Señales de la red N1.

Red	**Bus**	**Nro. de bits**	**Descripción**
		Datos entrantes	
N2	*n2_data_i*	256	bus de datos entrante.
N2	*n2_tag_i*	16	bus de direcciones de registros de la unidad de procesamiento.
N2	*n2_req_i*	1	señal de pedido entrante.
N2	*n2_ack_o*	1	señal de reconocimiento del pedido entrante.
N2	*n2_is_data_i*	1	señal que indica si el paquete entrante es de datos o de control.
		Datos salientes	
N2	*n2_data_o*	256	bus de datos saliente.
N2	*n2_tag_o*	16	bus de direcciones de registros de la unidad de procesamiento.
N2	*n2_req_o*	1	señal de pedido saliente.
N2	*n2_ack_i*	1	señal de reconocimiento del pedido saliente.
N2	*n2_is_data_o*	1	señal que indica si el paquete saliente es de datos o de control.
N2	*n2_addrx_o*	1	bus de dirección horizontal.
N2	*n2_addry_o*	1	bus de dirección vertical.

Tabla 4: Señales de la red N2.

Bibliografía

[1] Cassidy, Andrew S.: *Three topics in single-chip parallel computing: Theoretical foundations, speech recognition, and the silicon cortex*. Tesis de Doctorado, 2010, ISBN 9781124421896. https://search.proquest.com/docview/847570986?accountid=11752.

[2] *Asistente Inteligente Siri*. https://www.apple.com/es/siri.

[3] Klatt, Edward C.: *Voice-activated dictation for autopsy pathology*. Computers in Biology and Medicine, 21(6):429 – 433, 1991, ISSN 0010-4825. http://www.sciencedirect.com/science/article/pii/001048259190044A.

[4] *Asistente Inteligente Amazon Alexia*. https://www.amazon.com/b?node=17934677011.

[5] Leavitt, N.: *Let's Hear It for Audio Mining*. Computer, 35:23–25, Octubre 2002, ISSN 0018-9162. doi.ieeecomputersociety.org/10.1109/MC.2002.1039511.

[6] Chandrakasan, Anantha P., Samuel Sheng y Robert W. Brodersen: *Low Power CMOS Digital Design.* IEEE JOURNAL OF SOLID STATE CIRCUITS, 27:473–484, 1995.

[7] 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`.

[8] Huang, Xuedong, Alex Acero y Hsiao Wuen Hon: *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edición, 2001, ISBN 0130226165.

[9] Rabiner, Lawrence y Biing Hwang Juang: *Fundamentals of Speech Recognition.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993, ISBN 0-13-015157-2.

[10] Yoshizawa, Shingo, Naoya Wada, Noboru Hayasaka y Yoshikazu Miyanaga: *Scalable architecture for word HMM-based speech recognition and VLSI implementation in complete system.* IEEE Transactions on Circuits and Systems I: Regular Papers, 53:70–77, 2006.

[11] Lin, Edward C., Kai Yu, Rob A. Rutenbar y Tsuhan Chen: *Moving Speech Recognition from Software to Silicon: the In Silico Vox Project.*

[12] Huggins-Daines, D., M. Kumar, A. Chan, A. W. Black, M. Ravishankar y A. I. Rudnicky: *Pocketsphinx: A Free, Real-Time Continuous Speech Recognition System for Hand-Held Devices.* En *2006 IEEE Internatio-*

nal Conference on Acoustics Speech and Signal Processing Proceedings, volumen 1, páginas I–I, May 2006.

[13] Jelinek, Frederick: *Statistical Methods for Speech Recognition*. MIT Press, Cambridge, MA, USA, 1997, ISBN 0-262-10066-5.

[14] Muda, Lindasalwa, Mumtaj Begam y I. Elamvazuthi: *Voice Recognition Algorithms using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques*. CoRR, abs/1003.4083, 2010. `http://dblp.uni-trier.de/db/journals/corr/corr1003.html#abs-1003-4083`.

[15] Patterson, David A. y John L. Hennessy: *Computer Organization and Design, Fifth Edition: The Hardware/Software Interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edición, 2013, ISBN 0124077269, 9780124077263.

[16] Krishna, Rajeev, Scott Mahlke y Todd Austin: *Architectural Optimizations for Low-power, Real-time Speech Recognition*. En *Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '03, páginas 220–231, New York, NY, USA, 2003. ACM, ISBN 1-58113-676-5. `http://doi.acm.org/10.1145/951710.951740`.

[17] Choi, Young kyu, Kisun You y Wonyong Sung: *FPGA-based implementation of a real-time 5000-word continuous speech recognizer*. En *2008 16th European Signal Processing Conference*, páginas 1–5, Aug 2008.

[18] He, Guangji, Yuki Miyamoto, Kumpei Matsuda, Shintaro Izumi, Hiroshi Kawaguchi y Masahiko Yoshimoto: *A 54-mw 3x-real-time 60-kword continuous speech recognition processor VLSI.* IEICE Electronics Express, advpub, 2013.

[19] Lin, Edward C. y Rob A. Rutenbar: *A Multi-fpga 10X-real-time High-speed Search Engine for a 5000-word Vocabulary Speech Recognizer.* En *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '09, páginas 83–92, New York, NY, USA, 2009. ACM, ISBN 978-1-60558-410-2. http://doi.acm.org/10.1145/1508128.1508141.

[20] Li, Peng y Hua Tang: *Design of a Low-Power Coprocessor for Mid-Size Vocabulary Speech Recognition Systems.* IEEE Transactions on Circuits and Systems I: Regular Papers, 58:961–970, 2011.

[21] Price, Michael, James R. Glass y Anantha Chandrakasan: *A 6 mW, 5,000-Word Real-Time Speech Recognizer Using WFST Models.* IEEE Journal of Solid-State Circuits, 50:102–112, 2015.

[22] Deng, Y., S. Chakrabartty y G. Cauwenberghs: *Analog auditory perception model for robust speech recognition.* En *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, volumen 3, páginas 1705–1709 vol.3, July 2004.

[23] Jacob, P., A. Zia, O. Erdogan, P. M. Belemjian, J. Kim, M. Chu, R. P. Kraft, J. F. McDonald y K. Bernstein: *Mitigating Memory Wall Effects*

in High-Clock-Rate and Multicore CMOS 3-D Processor Memory Stacks. Proceedings of the IEEE, 97(1):108–122, Jan 2009, ISSN 0018-9219.

[24] Andreou, A. G., T. Figliolia, K. Sanni, T. S. Murray, G. Tognetti, D. R. Mendat, J. L. Molin, M. Villemur, P. O. Pouliquen, P. Julian, R. Etienne-Cummings y I. Doxas: *Bio-inspired system architecture for energy efficient, BIGDATA computing with application to wide area motion imagery.* En *2016 IEEE 7th Latin American Symposium on Circuits Systems (LASCAS)*, páginas 1–6, Feb 2016.

[25] Maschal, R. A., S. S. Young, J. P. Reynolds, K. Krapels, J. Fanning y T. Corbin: *New Image Quality Assessment Algorithms for CFA Demosaicing.* IEEE Sensors Journal, 13(1):371–378, Jan 2013, ISSN 1530-437X.

[26] Molin, J. L., T. Figliolia, K. Sanni, I. Doxas, A. Andreou y R. Etienne-Cummings: *FPGA emulation of a spike-based, stochastic system for real-time image dewarping.* En *2015 IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS)*, páginas 1–4, Aug 2015.

[27] Federico, M. Di, P. Julián y P. S. Mandolesi: *SCDVP: A Simplicial CNN Digital Visual Processor.* IEEE Transactions on Circuits and Systems I: Regular Papers, 61(7):1962–1969, July 2014, ISSN 1549-8328.

[28] Arulampalam, M. S., S. Maskell, N. Gordon y T. Clapp: *A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking.* IEEE Transactions on Signal Processing, 50(2):174–188, Feb 2002, ISSN 1053-587X.

[29] Sanni, K., T. Figliolia, G. Tognetti, P. Pouliquen y A. Andreou: *A Charge-Based Architecture for Energy-Efficient Vector-Vector Multiplication in 65nm CMOS.* En *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, páginas 1–5, May 2018.

[30] Villemur, M., P. Julian y A. G. Andreou: *Energy aware simplicial processor for embedded morphological visual processing in intelligent internet of things.* Electronics Letters, 54(7):420–422, 2018, ISSN 0013-5194.

[31] Matsuo, M., N. Hayasaka, K. Okumura, E. Hosomi y C. Takubo: *Silicon interposer technology for high-density package.* En *2000 Proceedings. 50th Electronic Components and Technology Conference (Cat. No.00CH37070)*, páginas 1455–1459, 2000.

[32] Sunohara, Masahiro, Takayuki Tokunaga, Takashi Kurihara y Mitsutoshi Higashi: *Silicon interposer with TSVs (Through Silicon Vias) and fine multilayer wiring.* En *2008 58th Electronic Components and Technology Conference*, páginas 847–852, May 2008.

[33] Karakiewicz, R., R. Genov y G. Cauwenberghs: *1.1 TMACS/mW Fine-Grained Stochastic Resonant Charge-Recycling Array Processor.* IEEE Sensors Journal, 12(4):785–792, April 2012, ISSN 1530-437X.

[34] Viterbi, A.: *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm.* IEEE Transactions on Information Theory, 13(2):260–269, April 1967, ISSN 0018-9448.

[35] Durgam, A. R. y K. Choi: *Optimized clock gating cell for low power design in nanoscale CMOS technology.* En *Fifth Asia Symposium on Quality Electronic Design (ASQED 2013)*, páginas 85–88, Aug 2013.

[36] Amdahl, Gene M.: *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities.* En *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), páginas 483–485, New York, NY, USA, 1967. ACM. http://doi.acm.org/10.1145/1465482.1465560.

[37] Papamarcos, Mark S. y Janak H. Patel: *A Low Overhead Coherence Solution for Multiprocessors with Private Cache Memories.* En *In Proc. 11th ISCA*, páginas 348–354, 1984.

[38] Gonzalez, R. y M. Horowitz: *Energy dissipation in general purpose processors.* En *1995 IEEE Symposium on Low Power Electronics. Digest of Technical Papers*, páginas 12–13, Oct 1995.

[39] Venkatesh, Ganesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson y Michael Bedford Taylor: *Conservation Cores: Reducing the Energy of Mature Computations.* En *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, páginas 205–218, New York, NY, USA, 2010. ACM, ISBN 978-1-60558-839-1. http://doi.acm.org/10.1145/1736020.1736044.

[40] Qazi, M., M. Sinangil y A. Chandrakasan: *Challenges and Directions for Low-Voltage SRAM.* IEEE Design Test of Computers, 28(1):32–43, Jan 2011, ISSN 0740-7475.

[41] Cassidy, A. S. y A. G. Andreou: *Beyond Amdahl's Law: An Objective Function That Links Multiprocessor Performance Gains to Delay and Energy.* IEEE Transactions on Computers, 61(8):1110–1126, Aug 2012, ISSN 0018-9340.

[42] *CORTEX - M0 Devices, Generic User Guide.* `http://infocenter.arm.com/help/topic/com.arm.doc.dui0497a/DUI0497A_cortex_m0_r0p0_generic_ug.pdf`.

[43] Hennessy, John L. y David A. Patterson: *Computer Architecture, Sixth Edition: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edición, 2011, ISBN 012383872X, 9780123838728.

[44] *AMBA bus specification.* `https://developer.arm.com/products/architecture/amba-protocol`.

[45] Monchiero, Matteo, Gianluca Palermo, Cristina Silvano y Oreste Villa: *Efficient Synchronization for Embedded On-chip Multiprocessors.* IEEE Trans. Very Large Scale Integr. Syst., 14(10):1049–1062, Octubre 2006, ISSN 1063-8210. `http://dx.doi.org/10.1109/TVLSI.2006.884147`.

[46] `https://www.st.com/content/ccc/resource/technical/document/technical_note/58/17/ad/50/fa/c9/48/07/DM00054618`.

pdf/files/DM00054618.pdf/jcr:content/translations/en.DM00054618.pdf.

[47] https://www.mouser.com/datasheet/2/268/20005142B-514625.pdf.

[48] Cooley, J.W. y J.W. Tukey: *An Algorithm for the Machine Calculation of Complex Fourier Series.* Mathematics of Computation, 19:297–301, Enero 1965.

[49] Jouppi, Norman P., Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh K Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, Richard C. Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox y Doe Hyun Yoon: *In-datacenter performance analysis of a tensor processing*

unit. 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), páginas 1–12, 2017.

[50] Merolla, Paul A., John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar y Dharmendra S. Modha: *A million spiking-neuron integrated circuit with a scalable communication network and interface*. Science, 345(6197):668–673, 2014, ISSN 0036-8075. `http://science.sciencemag.org/content/345/6197/668`.

[51] `https://tezzaron.com/applications/diram4-3d-memory/`.