Diseño de redes en chip de propósito específico con información de rutado físico

David Atienza [1,2], Srinivasan Murali[3,2], Federico Angiolini[4], Luca Benini[4], Giovanni De Micheli[2], José M. Mendías[1] y Román Hermida[1]

[1]DACYA/UCM, Madrid, Spain. E-mail:{datienza, mendias, rhermida}@dacya.ucm.es
[2]LSI/EPFL, Lausanne, Switzerland. E-mail:{david.atienza,giovanni.demicheli}@epfl.ch
[3]CSL/Stanford University, Stanford, USA, E-mail: murali@stanford.edu
[4]DEIS/UNIBO, Bologna, Italy. E-mail: fangiolini,lbenini@deis.unibo.it

Resumen—**Debido a la creciente demanda de comunicación entre procesadores y dispositivos de memoria en *Sistemas en Chip (SoCs)*, recientemente se ha propuesto el nuevo paradigma de *Redes en Chip (NoCs)* para SoCs. Con objeto de lograr que las NoCs sean una opción factible para la industria, es necesario diseñar NoCs a medida para cada tipo de aplicación que se desea ejecutar en los SoCs. En este artículo presentamos una nueva metodología de diseño para NoCs a medida, que tiene en consideración la información de rutado físico de los enlaces de la NoC y el emplazamiento en el sistema final de los componentes del SoC a la hora de definir la topología de NoC adecuada. Esto permite detectar y eliminar a las violaciones de tiempo y realizar correctas estimaciones de disipación de potencia en el sistema de interconexionado. Este nuevo flujo de diseño define e instancia de manera complétamente automática el código HDL sintetizable para NoCs a medida, incorporando así mismo mecanismos para evitar los problemas de interbloqueos (deadlocks) en el enrutado. Nuestros resultados con varios sistemas SoC complejos demuestran que el diseño físico final de la topología NoC que se debe utilizar puede realizarse en menos de 4 horas, en vez de semanas. Además, con esta metodología se obtienen mejoras medias en el consumo cercanas a 3× y del rendimiento de 2× (alcanzando frecuencias cercanas a 900 MHz) con respecto a las posibles alternativas de diseño manual basadas en NoCs más regulares tipo malla, Clos, etc.**

Palabras clave— **Redes en chip, sistemas en Chip, automatización, topología, diseño a medida.**

I. Introducción

Para vencer los problemas de escalabilidad y complejidad de los nuevos *Sistemas en Chip (SoCs)* formados por gran cantidad de componentes prediseñados, en los últimos años se está proponiendo una prometedora tecnología de interconexión denominada *Network-On-Chip (NoC)* que parece capaz de reemplazar con garantías tanto a los grandes buses como a las largas interconexiones dedicadas presentes en los futuros SoC [3]. Esta tecnología extrapola algunos de los conceptos presentes en una red de computadores a la interconexión de múltiples IP-cores difundidos sobre un substrato común.

Algunas de las fases más importantes del diseño de NoCs es la elección de la topología o estructura de la red y varios parámetros de diseño, tales como la frecuencia o el ancho de banda entre cada interconexión. En relación a las topologías que se pueden utilizar, se ha comprobado que las topologías regulares de multiprocesadores (malla, Clos, torus, etc.) aplicadas a SoCs producen NoCs con mayor consumo de energía y peor rendimiento que las topologías NoC a medida [4][8]. Esto se debe a que dichas topologías regulares sólo son interesantes cuando las características del tráfico que va a existir en el sistema final no se pueden determinar en tiempo de diseño. Sin embargo, en SoCs las aplicaciones que podrán ser ejecutadas son conocidas en tiempo de diseño para la mayor parte de las arquitecturas SoC de última generación, tales como, Philips Nexperia [1], ST Nomadik [2], etc. Además, los diseños NoCs basados en topologías regulares (como las mallas) no tienes porque producir diseños físicos finales más regulares y predecibles en sus interconexiones, ya que en SoCs los componentes (procesadores, memorias) no tienen porqué tener tamaños uniformes [4].

En este artículo proponemos una metodología completa y automática de diseño de NoCs a medida, libres de interbloqueos en el enrutado de paquetes, para cada tipo de aplicación (o conjunto de aplicaciones) que se van a ejecutar en un cierto diseño de SoC. Nuestra metodología permite optimizar la topología NoC diseñada finalmente de acuerdo a dos métricas objetivo (o una combinación lineal de ambas): minimizar la disipación total de potencia del sistema y/o la latencia global de los paquetes de información dentro de la NoC. Además, el flujo de diseño propuesto permite indicar a los diseñadores de sistemas las ligaduras de diseño más habituales en el diseño de NoCs, tales como el área total o la longitud máxima de las interconexiones. Nuestros resultados con varios sistemas SoC reales (entre 8 y 42 componentes) demuestran que los diseños de NoCs obtenidos automáticamente mejoran significativamente los diseños de NoCs obtenidos manualmente para estas aplicaciones, por ejemplo, en media $3.8\times$ menos disipación de potencia, $1.55\times$ menos latencia y $1.28\times$ menos longitud de las interconexiones, comparados con diseños basados en mallas (Secciones 5 y 6). Más aún, los diseños SoC finales con las topologías NoC generadas automáticamente pueden ser diseñadas en unas pocas horas, mientras que los diseños manuales requiren varias semanas.

II. Trabajo previo

Extenso trabajo de investigación ha sido realizado en el estudio de mecanismos eficientes de diseño y síntesis de SoCs basados en buses [7]. Aunque algunos problemas de diseño existentes en NoCs son similares a los de los buses (ancho de banda,et.), otros nuevos han surgido con las NoCs, como definición del tamaño de los conmutadores, definición de tablas de rutado, etc. Métodos para recopilar estadísticas y analizar patrones de tráfico en las NoC y en los buses han sido desarrolados para poder estudiar estos nuevos problemas [6].

Este trabajo ha sido subvencionado mediante el Proyecto CICYT TIN2005-5619, el Proyecto FNS de Suiza 20021-109450/1, y una Beca de Profesores UCM en el Extranjero para David Atienza.

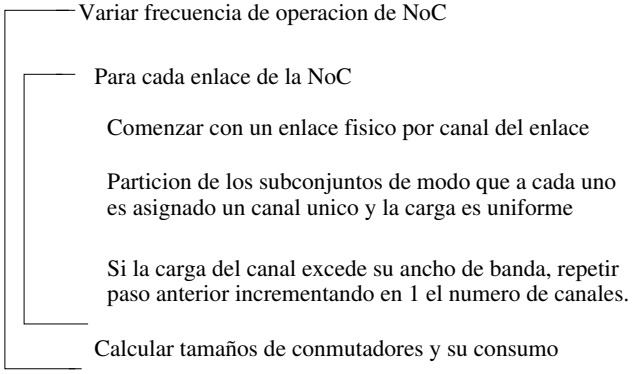


Fig. 1. Flujo de diseño de NoC propuesto

El problema de la selección de topologías y ubicación de componentes en sistemas basados en NoCs utilizando topologías regulares (mallas, torus, etc.) han sido ya explorados [8]. Además, [9] propone una herramienta para realizar el diseño de dichas topologías y obtener estimaciones del área y la longitud de cada interconexión interna de la NoC. Contrariamente a nuestra propuesta en este artículo, estos trabajos sólo permiten la selección de una topología entre un conjunto de topologías estándar definidas en una cierta librería.

Recientemente se ha estudiado el mismo problema de la generación de topologías en redes de sistemas multiprocesador [10]. En este caso, como los patrones de tráfico no son fáciles de predecir, la mayoría de los trabajos en dicho área utilizan un sistema basado en árboles (similar a los árboles de Steiner) y sólo aseguran la conectividad con ligaduras de diseño (energía, rendimiento, etc.) a nivel de nodos o interconexiones individuales [10]. Por ello, este tipo de técnicas no es directamente aplicable al contexto del diseño y síntesis de NoCs a medida. Dicho concepto de NoCs a medida ha sido estudiado anteriormente en [11], pero en este caso no se utiliza la información del enrutado físico del sistema final. Además, el número y tamaño de las particiones en la NoC para realizar las optimizaciones debe ser introducido manualmente por el diseñador.

Finalmente, trabajos previos han afrontado el complejo problema de generar automáticamente código simulable y sintetizable de las NoCs a nivel de transferencia de registros (*Register Transfer Level o (RTL)*) [12]. También, la definición de modelos de estimación de energía y área para NoCs han sido estudiados [13]. Estas propuestas son complementaria al trabajo presentado en este artículo, ya que necesitan una primera topología NoC que simular y que puede ser suministrada mediante la metodología automática que se indica en este artículo.

III. Flujo de Diseño

El flujo de diseño de NoCs propuesto se muestra en la Figura 1. Como entrada al sistema, el usuario debe especificar los objetivos y ligaduras de diseño que la NoC final debe satisfacer. Asimismo, las características del tráfico, tamaño de los componentes NoC y sus modelos de area y consumo de energía son aceptados como parámetros (Sección 4).

Durante este flujo, la arquitectura NoC que optimiza

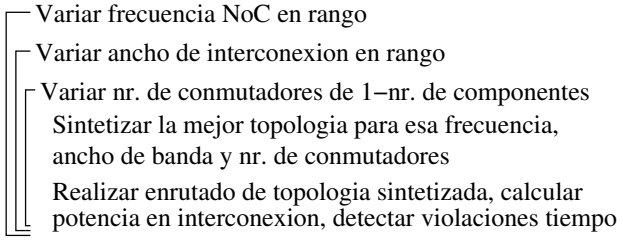


Fig. 2. Síntesis de la arquitectura NoC (fase 2 del flujo de diseño)

los objetivos indicados por el diseñador y que satisface las ligaduras de diseño es sintetizada automáticamente. Las diferentes fases de este proceso se muestran en la Figura 2 y se explican en detalle en la Sección 5. Como parámetros arquitectónicos más importantes, este flujo de diseño explora, respetando las restricciones de diseño (rendimiento, potencia, etc.), principalmente la frecuencia de operación deseada para la NoC final, el tamaño de las interconexiones entre conumtadores. El ancho de banda de las interconexiones de la NoC es el producto de la frecuencia de la NoC y el ancho de las interconexiones, y la síntesis de la topología final asegura que el tráfico de cada enlace es menor o igual que el ancho de banda disponible.

La síntesis se realiza para cada posible conjunto de valores de los parámetro arquitectónicos, variando el número de conmutadores usado en la NoC. El algoritmo comienza con todos los componentes (procesadores, memorias, etc.) conectados al mismo conmutador y varía iterativamente la topología NoC hasta llegar a explorar la ubicación de un componente por conmutador. Durante el proceso de síntesis de cada topología NoC se define el tamaño de los conmutadores y la conectividad de los componentes de tal manera que no haya interbloqueos en el enrutado para los distintos flujos de datos entre los componentes existentes en la NoC. Durante este proceso, para obtener estimaciones exactas del área y tamaño de las interconexiones en el diseño final, el proceso completo de diseño físico se realiza automáticamente, encontrando la posición en el plano de cada componente del diseño SoC y los elementos internos de la NoC. Para ello utilizamos la herramienta de enrutado PARQUET [16]. De este modo se obtienen las violaciones de tiempos en las interconexiones y disipación de energía para cada diseño NoC concreto, lo que permite seleccionar la más idónea según las ligaduras de diseño impuestas por el diseñador.

En la última fase del flujo de diseño (fase 3 en la Figura 1), el código RTL (en SystemC) de los conmutadores, interfaces de red e interconexiones para la topología elegida son generados automáticamente. Para ello utilizamos Xpipes[5], una librería de macros de componentes NoC *soft* y la herramienta asociada XpipesCompiler [12] para interconectar todo el SoC. También se obtiene código RTL sintetizable que se puede usar para verificar la NoC con ayuda de una FPGA. Por último, el emplazamiento físico de los componentes y el rutado de las interconexiones NoC se realiza automáticamente con Cadence SoC Encounter [17].

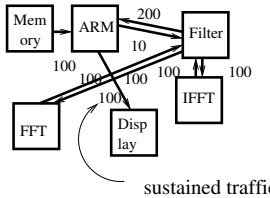


Fig. 3. Ejemplo: Filtro

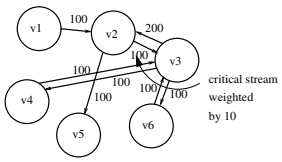


Fig. 4. Gráfico con flujos de tráfico críticos y mantenidos

IV. Modelos de entrada

Los anchos de banda requeridos entre los distintos componentes y la NoC se obtiene a partir de los valores máximos, mínimos y ligaduras en las latencias de cada uno de los elementos internos del SoC considerado, de manera análoga a [8] y [9]. Además, las prioridades de los distintos flujos de información entre cada par de componentes tiene un valor de criticalidad que el usuario puede definir. A partir de estos valores, se construye un grafo de componentes para cada aplicación ejecutada en el SoC conectado por la NoC del siguiente modo:

Definición 1: El grafo de componentes SoC es un grafo dirigido, $G(V, E)$ donde cada vértice $v_i \in V$ representa un componente y la arista dirigida (v_i, v_j), que denota como $e_{i,j} \in E$, representa la comunicación entre los componentes v_i y v_j. El peso de cada arista $e_{i,j}$, denotado por $comm_{i,j}$, representa el radio de ancho de banda de v_i a v_j, normalizado según la criticalidad de la comunicación. El conjunto F representa el conjunto de todos los flujos de tráficos, con valor para cada uno de ellos f_k, $\forall k \in 1 \cdots |F|$, que representa el ancho de banda sostenido entre los vértices fuente (s_k) y destino (d_k) de dicho flujo.

En la Figura 4 se muestra un ejemplo de grafo de componentes para una aplicación de filtrado simple (ver Figura 3). Los vértices del grafo de componentes se anotan con el ancho de banda requerido (según [17]).

Para poder optimizar y evaluar nuestros diseños NoC a medida, hemos construido modelos analíticos de disipación de potencia y área de los componentes NoC para la arquitectura propuesta en Xpipes [5]. De este modo, las estimaciones de potencia para los componentes, y las capacitancias y resistencias de las interconexiones internas de la NoC se han validado utilizando Cadence SoC Encounter usando una librería tecnológica de 0.13μm. La actividad de los componentes y líneas de comunicación se han obtenido mediante módulos de tráfico funcional de las aplicaciones utilizadas. Utilizando la información obtenida anteriormente de capacitancia, resistencia y actividad de conmutación se procede a la estimación de la disipación de consumo y energía utilizando Synopsys PrimePower [18]. Dada la modularidad y simetría intrínseca de los componentes NoC, hemos podido verificar que nuestros modelos analíticos son muy precisos (con errores máximo y medio de menos del 7% y 5 %, respectivamente).

V. Algoritmos de diseño de topologías NoC

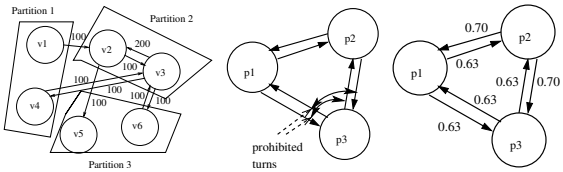
El algoritmo básico se muestra en el Algoritmo 1. En la versión actual, el mecanismo de síntesis puede minimizar la frecuencia de operación ($freq_\theta$) o el ancho de banda (lw_θ). El diseñador determina por tanto los rangos interesantes para dichos parámetros y se explora un conjunto discreto de puntos de diseño (ϕ) de manera análoga a [7]. En nuestros experimentos soportamos 8 valores de frecuencia y 4 anchos de banda, suministrando 32 puntos del conjunto ϕ. El resto de pasos (3-15 en el algoritmo 1) se repiten para cada punto de ϕ.

Como la síntesis de una topología y la asignación de recursos es un problema NP-completo [11], hemos desarrollado heurísticas para su implementación, cubriendo para cada punto del espacio de diseño θ topologías con un número de conmutadores que oscilan entre un único conmutador para todos los componentes a uno por cada uno. Además, en cada uno de estos pasos (i), el grafo de entrada de componente se particiona en i particiones disjuntas (paso 3) de tal manera que las aristas entre los elementos de una misma partición son mayores que las de distintas particiones. (Figure 5(a)), y el número de vértices en cada partición es aproximadamente el mismo. De este modo, los flujos con mayores requerimientos de ancho de banda se ubicaran en el mismo conmutador, minimizando de este modo la disipación de energía en los flujos de datos y sus interconexiones.

Tras resolver el tráfico dentro de una partición, en los pasos 5-9 se establecen las conexiones entre particiones de manera que soportan el ancho de banda requerido. Con este fin, en el paso 5 se genera el Grafo de Coste de Conmutación (*Switch Cost Graph o SCG*). SCG es un grafo conectado con i vertices, donde i es el número de particiones (o conmutadores) en la topología actual. SCG representa la información de conectividad lógica entre conmutadores. No obstante, SCG no implica que dos conmutadores tengan que estar unidos físicamente, la conectividad final entre conmutadores se determina según SCG en el procedimiento *PATH_COMPUTE*, que se explica a continuación.

En las NoCs habitualmente se utiliza control de flujo tipo agujero de gusano (*wormhole*) para reducir la latencia de comunicación y los requerimientos de almacenamiento intermedio en los conmutadores [4]. Este tipo de enrutado puede provocar interbloqueos en el enrutado de paquetes debido a dependencias cíclicas de recursos [19]. En nuestra metodología preprocesamos el grafo SCG y eliminamos ciertos giros usando el algoritmo propuesto en [15] para eliminar dichas dependencias cíclicas, construyendo el conjunto de giros prohibidos (*Prohibited Turn Set (PTS)*). Dicho conjunto PTS se utiliza junto con SCG y como resultado, logramos evitar dichas dependencias con una complejidad polinómica y eliminando como máximo 1/3 del número total de giros. Un ejemplo de su aplicación se muestra en la Figura 5. Se definen 3 particiones (Figura 5(b)) para el grafo de componentes del ejemplo del filtro (Figura 3). Tras aplicar el algoritmo de prohibición de giros se construye SCG (Figura 5(b)). En este caso, los giros prohibidos se indican mediante arcos circulares, es decir, los giros alrededor del vértice P3 están prohibidos y no es posible incluir ningún camino que utiliza el conmutador P3 como conmutador intermedio.

Las conexiones físicas entre los conmutadores se fi-



(a) Min-cut partitions (b) SCG graph (c) Path selection

Fig. 5. Ejemplos del algoritmo usado

Algorithm 1 Algoritmo de diseño de topologías

1: Elegir punto de diseño θ de ϕ: $freq_\theta, lw_\theta$
2: **for** i = 1 to $|V|$ **do**
3: Encontrar i particiones disjuntas del grafo de componentes
4: Establecer un conmutador con N_j entradas y salidas para cada partición, $\forall j \in 1 \cdots i$. N_j es el número de vértices (cores) en partición i. Comprobar violaciones de ligaduras de ancho de banda.
5: Construir SCG con pesos en las aristas inicializados a 0
6: Construir PTS a partir del grafo SCG para eliminar interbloques de enrutado
7: Inicializar ρ a 0
8: Encontrar los caminos para los flujos de datos entre los conmutadores usando la función *PATH_COMPUTE(i, SCG, ρ, PTS, θ)*
9: Evaluar la disipación de potencia de cada conmutador y latencia según los caminos elegidos
10: Repetir los pasos 8 y 9 incrementando ρ en pasos sucesivos hasta que las restricciones de latencia sean satisfechas o hasta que ρ alcanza ρ_{thresh}
11: Si ρ_{thresh} es alcanzado y las restricciones de latencia no han sido satisfechas, ir al paso 2 y evaluar el siguiente valor de i
12: Calcular el diseño físico actual. Obtener área, longitud de interconexiones. Comprobar violaciones de tiempo en las interconexiones y evaluar la disipación de energía en cada interconexión.
13: Si la frecuencia alcanza o supera $freq_{theta}$, ya la solución actual mejora el diseño anterior, cumpliendo todas las ligaduras, anotar el punto de diseño y las topología
14: **end for**
15: Repetir los pasos 2-14 para cada punto del espacio de diseño definido en θ
16: Para la mejor topología y punto del espacio de diseño según los objetivos del usuario, generar la información para instanciar el código HDL de Xpipes y sintetizar con Cadence SoC Encounter

jan en el paso 8 del Algoritmo 1 mediante el procedimiento *PATH_COMPUTE*, que estima el coste del establecimiento de enlaces físicos y la búsqueda de caminos para los flujos de tráficos. En este caso presentamos su funciomiento si se desea minimizar la métrica de disipación de energía, en el caso de la latencia es similar.

Un ejemplo ilustrando el funcionamiento de *PATH_COMPUTE* se muestra en Ejemplo V. En dicho procedimiento, los flujos se ordenan en orden decreciente de sus requerimientos de comunicación, de tal modo que los flujos con mayores radios de transferencia se deciden primero. Esta heurística es la que ha producido los mejores resultados (de minimización de ancho de banda y disipación de energía) en nuestros experimentos y en otros trabajos previos [9]. A continuación, para cada flujo se evalúa la cantidad de potencia consumida en cada conmutador, si el tráfico que genera pasa por el mismo. El valor de disipación de potencia en cada conmutador depende de su tamaño y de la cantidad de tráfico que ya ha sido enrutado a través de él. También depende de la cantidad de canales físicos existentes ya que cada nuevo canal abierto afecta a todos los flujos de información que circulan a través de dicho conmutador, y deben ser recalculados cada vez que se abre uno nuevo. En este caso, se comprueba igualmente que el tamaño actual de dicho conmutador sigue respetando la frecuencia de operación impuesta.

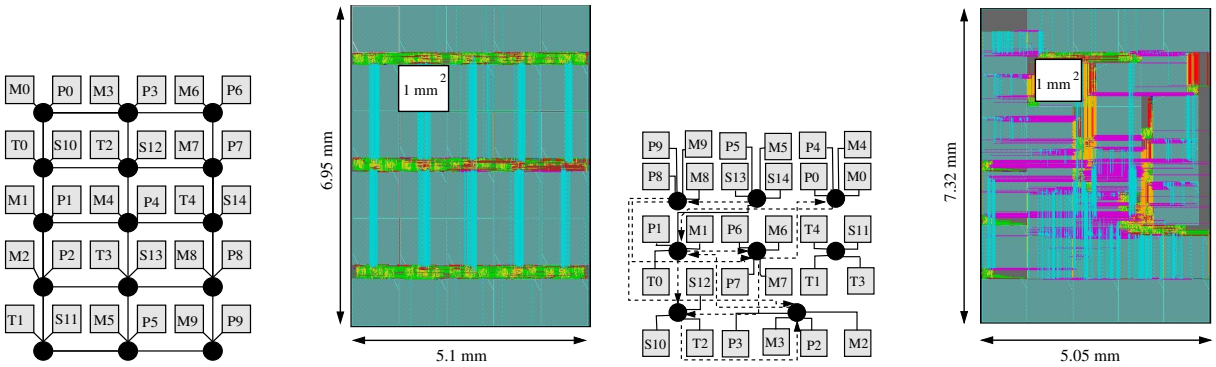
Una vez el coste de atravesar un flujo cada posible conmutador ha sido calculado, se procede a seleccionar un camino para dicho flujo. Para realizar este proceso se aplica un algoritmo similar al de la búsqueda del camino más corto según un origen y un destino, o Algoritmo de Dijkstra [20], en el paso 15 del procedimiento *PATH_COMPUTE*. Cuando se elige dicho camino, sólo los caminos que nos incluyen ningún giro dentro del conjunto PTS construido anteriormente son considerados. Este proceso se itera para el resto de flujos de datos existentes en el SoC bajo consideración.

En el caso del grafo SCG de la Figura V, vamos a considerar a modo de ejemplo el caso del flujo existente entre los vértices $v1$ y $v2$ (de valor 100), entre la Partición 1 ($p1$) y la Partición 2 ($p2$). Inicialmente suponemos que no existe ningún camino físico entre ningún conmutador. De este modo, si enrutamos el flujo a traves de una interconexión entre dos conmutadores, primero hay que crear el canal. El coste de dicho enrutado entre dos conmutadores cualesquiera, se obtiene en el paso 11 del procedimiento *PATH_COMPUTE*. El grafo SCG con las aristas anotadas se muestra en la Figura 5(c). Se comprueba que el coste de las aristas que comienzan en $p2$ son diferentes del resto debido a la diferencia en el valor inicial de actividad de conmutación en $p2$, en relación a los demás conmutadores. Esto se debe a que el conmutador $p2$ tiene que soportar los flujos entre los vértices $v2$ y $v3$ dentro de la partición considerada actualmente. En este caso, el camino menos costoso para este flujo, el cual es entre los conmutadores $p1$ y $p2$, es seleccionado. De este modo, hemos definido el camino físico entre los dos conmutadores, y será considerado para el resto de flujos existentes. Además, el tamaño y la actividad de conmutación han variado para ambos conmutadores, lo cual se anota para futura iteraciones del algoritmo.

Algorithm 2 PATH_COMPUTE(i, SCG, ρ, PTS, θ)

1: Inicializar el conjunto $PHY(i1, j1)$ a falso y $Bw\_avail(i1, j1)$ a $freq_\theta \times lw\theta, \forall\, i1,\ j1 \in 1 \cdots i$
2: Inicializar $switch\_size\_in(j)$ y $switch\_size\_out(j)$ a N_j, $\forall\ j \in 1 \cdots i$. Encontrar $switching\_activity(j)$ para cada conmutador, según el flujo de tráfico existente dentro de la partición.
3: **for** cada flujo f_k, $k \in 1 \cdots |F|$ en orden decreciente de coste de cada flujo f_c **do**
4: **for** i1 de 1 a i y j1 de 1 a i **do**
5: {Encontrar el coste marginal de usar la interconexión i1, j1}
6: {Si el enlace físico existe & tiene un ancho de banda que soporta el flujo}
7: **if** $PHY(i1, j1)$ y $Bw\_avail(i1, j1) \geq f_c$ **then**
8: Encontrar $cost(i1, j1)$, el valor marginal de disipación de energía que permite reusar la interconexión ya existente
9: **else**
10: {Tenemos que abrir un nuevo enlace físico entre i1 y j1}
11: Asignar $cost(i1, j1)$, la disipación de potencia marginal para abrir y utilizar el canal. Evaluar si las ligaduras de frecuencia del conmutadore se satisfacen.
12: **end if**
13: **end for**
14: Asignar $cost(i1, j1)$ a la arista $W(i1, j1)$ en SCG
15: Encontrar el camino de menor coste entre las particiones que tienen los flujos con fuente (s_k) y como destino (d_k), y estando presente en el SCG. Elegir solamente entre aquellos caminos que no tienen ningún giro prohibido almacenado en PTS
16: Actualizar PHY, Bw_avail, $switch\_size\_in$, $switch\_size\_out$ y $switching\_activity$ para las interconexiones y conmutadores incluidos en el camino seleccionado
17: **end for**
18: Devolver los caminos elegidos, los nuevos tamaños de los conmutadores y la nueva conectividad entre los conmutadores.

El procedimiento *PATH_COMPUTE* devuelve los tamaños de los conmutadores, la conectividad entre conmutadores y componentes y los caminos utilizados para cada flujo de tráfico. El objetivo para definir dichos caminos es la minimización de la disipación de poten-



(a) Topología a mano (b) Diseño físico (c) Síntesis automática (d) Diseño físico

Fig. 6. (a), (b) Topologia diseñada a mano y su diseño final. M: procesadores ARM7, T: generadores de trafico, P, S: memorias privadas y compartidas (c), (d) Topología sintetizada automáticamente y su diseño final. En Figura (c), las interconexiones bidireccionales se indican con líneas negras y las unidireccionales con líneas de puntos.

cia en los conmutadores. De este modo, una vez que los caminos han sido establecidos, si las restriciones indicadas para la latencia no se cumplen, el algoritmo modifica la latencia de manera iterativa, utilizando el parámetro ρ (en los pasos 7, 10 y 11 del Algoritmo 1). El límite inferior para ρ, indicado como ρ_{thresh}, se fija al valor de disipación de energía del flujo de mayor radio de transferencia cuando cruza el conmutador mayour en el SCG. Para este valor de ρ, para todos los tráficos, es mejor el seleccionar el camino con el menor número de conmutadores que el camino con menor disipación de energía. El valor de ρ se varía en varios pasos incrementales hasta que la latencia de todos los caminos han sido satisfechas o alcanzan el valor ρ_{thresh}.

En el paso final se optimiza de maneral conjunta el área y la longitud de las interconexiones, dando igual prioridad a ambas, y se elige así la mejor topología de todas la exploradas, según los criterios del diseñador. A continuación, se realiza la instanciación de la topología usando los componentes de la NoC de Xpipes y se emplea Cadence SoC Encounter para diseñar el circuito físico final.

Finalmente, en resumen, el proceso completo de síntesis de las NoC a medida propuesto escala de manera polinómica con respecto al número de cores utilizado en el diseño y el número de topologías exploradas también. Por ello, esta metodología permite un tiempo de diseño de NoCs muy rápido, incluso para SoCs con 30 o 40 componentes (Sección 6).

VI. Experimentos y casos de estudio

En esta sección presentamos los resultados obtenidos con la metodología propuesta en comparación con los mecanismos de estado del arte en el diseño de NoCs con topologías regulares. Para el primer grupo de experimentos realizado hemos utilizado una arquitectura NoC para un cierto diseño SoC que ejecuta varias aplicaciones multimedia [14]. El diseño consta de 30 elementos: 10 procesadores ARM7 con sus respectivas caches, 10 memorias privadas (una por procesador), 5 generadores de tráfico a medida, 5 memorias compartidas y dispositivos para dar soporta a la comunicación entre los procesadores. La NoC diseñada a mano consta de 15 conmutadores conectados en una red tipo malla 5x3 (Figura 6(a)). El diseño físico (Figura 6(b)) se realizó utilizando la herramienta

TABLA I
Comparaciones con topologías estándar

Appl	Topol.	Potencia (mW)	Avg. Saltos	Area mm^2	Tiempo (mins)
VPROC	custom	79.64	1.67	47.68	68.45
	malla	301.8	2.58	51.0	
	opt-malla	136.1	2.58	50.51	
MPEG4	custom	27.24	1.5	13.49	4.04
	malla	96.82	2.17	15	
	opt-malla	60.97	2.17	15.01	
VOPD	custom	30.0	1.33	23.56	4.47
	malla	95.94	2.0	23.85	
	opt-malla	46.48	2.0	23.79	
MWD	custom	20.53	1.15	15	3.21
	malla	90.17	2.0	13.6	
	opt-malla	38.60	2.0	13.8	
PIP	custom	11.71	1	8.95	2.07
	malla	59.87	2.0	9.6	
	opt-malla	24.53	2.0	9.3	
IMP	custom	52.13	1.44	29.66	31.52
	malla	198.9	2.11	29.4	
	opt-malla	80.15	2.11	29.4	

Cadence SoC Encounter y se mantuvo la estructura de malla en el enrutado. Cada elemento tiene un area de 1 mm^2 [14]. El proceso completo, desde la especificación de la topología hasta la generación del diseño físico final, tardó varias semanas. La frecuencia final obtenida fue de 885 MHz, determinada por el camino crítico en las etapas múltiples de la lógica de los conmutadores, y su consumo fue de 368.08 mW.

A continuación se utilizó la metodología propuesta para diseñar automáticamente la NoC. En este caso se fijó la frecuencia de operación al mismo valor que la alcanzada manualmente (885 MHz) y se intentó minimizar el consumo de energía. Los resultados obtenidos con SoC Encounter se muestran en las Figuras 6(c) y 6(d). La NoC final preserva la frecuencia de trabajo de 885 MHz, pero sólo incluye 8 switches. Asimismo, dicho diseño automático logra reducir significativamente la potencia disipada (277.08 mW), es decir, una mejora de $1.33\times$ con respecto al diseño manual. Esta redución se logra principalmente mediante una distribución eficiente de las memorias compartidas entre los procesadores ARM, lo que se realizó de manera manual en el otro diseño. El área del diseño físico final, aunque realizado automáticamente, es muy cercano al obtenido manualmente (apenas un incremento del 4.3% en relación al diseño manual). Finalmente, el tiempo de ejecución obtenido mejora al del diseño manual en un 10% y la latencia media de paquetes se reduce igualmente en un 11.3%.

En el segundo grupo de experimentos se utilizaron

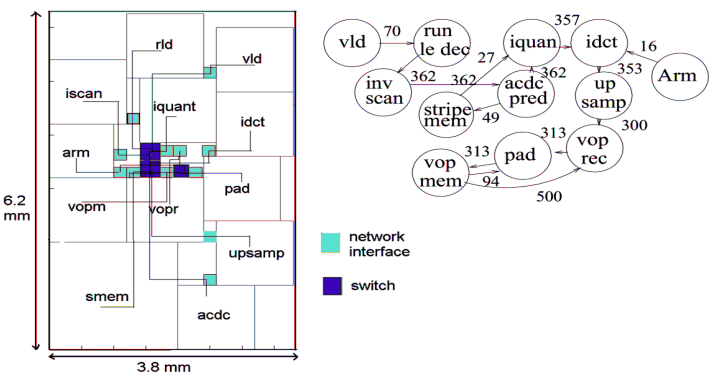


Fig. 7. Diseño final de VOPD con NoC a medida, y su grafo de componentes

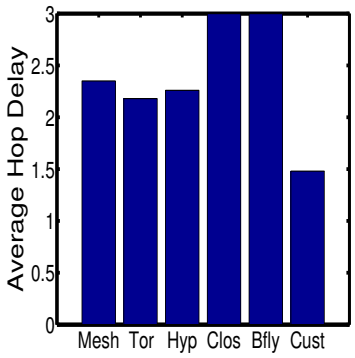


Fig. 8. Comparación de rendimiento entre topologías NoC

6 casos de estudio multimedia [4]: un procesador de video (VPROC) de 42 componentes, un decodificador MPEG4 (MPEG4) de 12 componentes, un decodificador de planos de objetos de video (Video Object Plane Decoder o VOPD) con 12 componentes, una aplicación de video multivista (Multi-Window Display o MWD) de 12 cores, una aplicación de Foto-en-Foto (Picture-in-Picture o PIP) con 8 componentes y una aplicación de procesado de imagen escalable (IMage Processing o IMP) con 23 componentes.

Hemos generado en todos los casos topologías NoC basadas en mallas, considerando un solo componente por conmutador, y todos los conmutadores con 5 puertos de entrada y salida. Además, hemos generado también una variante optimizada de las mallas para cada uno de estos diseños (OPT-MESH), donde los puertos y enlaces no utilizados en los flujos de tráfico han sido eliminados. El diseño físico final (incluyendo rutado de la NoC) para uno de los casos de estudio (VOPD) se muestra en la Figura 7, los resultados para el resto de casos fueron muy similares. Asimismo, los resultados que indican la disipación de potencia de la NoC (en conmutadores e interonexiones), latencia media de los paquetes y área se muestran en la Tabla I, y verifican que la latencia media de las topologías de malla y opt-mesh es la misma. Además, la topología diseñada con la metodología automática propuesta reduce significativamente los resultados manuales en cuanto a disipación de potencia ($2.78\times$ en media) y en latencia media ($1.59\times$ en media).

Finalemente, en el último grupo de experimentos hemos comparado topologías automáticamente generadas con topologías estándar (hipercubo, Clos, etc.) generadas usando la herramienta Sunmap[9], y tratando de maximizar el rendimiento. La aplicación utilizada en este caso es un sistema de procesado de señal con 25 componentes. Los resultados obtenidos se muestran en la Figura 8, y verifican que la metodología automática propuesta mejora significativamente el rendimiento final del sistema ($1.73\times$ de media) en relación a las topologías estándar.

VII. Conclusiones

En este artículo hemos presentado una metodología automática de generación de topologías NoC a medida que es capaz de satisfacer las restricciones de diseño (rendimiento, disipación de potencia, energía) de un variado conjunto de aplicaciones de SoC. Con objeto de obtener resultados más fiables y un menor número de iteraciones en el diseño final, la metodología propuesta hace uso de la información del diseño físico y enrutado de las NoCs en la fase de diseño de la topología NoC y ubicación de componentes de procesamiento y de memoria. Nuestros resultados experimentales en varios sistemas SoC demuestran que las topologías NoC sintetizadas automáticamente mejoran la disipación de energía ($2.78\times$ en media) y reducen la latencia ($1.59\times$ en media) de otras posibles soluciones basadas en topologías estándar (malla, torus, hipercubo, Clos y mariposa) optimizadas manualmente.

Referencias

[1] S. Dutta et al., “Viper: An MP-SOC for Advanced Set-Top Box and Digital TV Systems”, IEEE D&T of Computers, 2001.
[2] A. Artieri et al., “Nomadik Multimedia Platform for Mobile Devices”, STM Tech. Report TA305, 2003.
[3] L.Benini and G.De Micheli, “Networks on Chips: A New SoC Paradigm”, IEEE Computers, 2002.
[4] D. Bertozzi et al., ”NoC Synthesis Flow for Customized Domain Specific MPSoC”, IEEE Trans. PDS, 2005.
[5] S. Stergiou et al., “×pipesLite: a Synthesis Oriented Design Library for NoCs”, Proc. DATE 2005.
[6] K.Lahiri et al., “Design Space Exploration for Optimizing On-Chip Communication Architectures”, IEEE TCAD, 2004.
[7] S. Pasricha et al., “Floorplan-aware automated synthesis of bus-based communication architectures”, Proc. DAC, 2005.
[8] J. Hu et al., ”Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures’, Proc. DATE 2003.
[9] S. Murali et al., “SUNMAP: A Tool for Automatic Topology Selection and Generation for NoCs”, Proc. DAC, 2004.
[10] R. Ravi et al., “Approximation algorithms for degree-constrained minimum-cost network design problems”, Algorithmica, 2001.
[11] A.Pinto et al., “Efficient Synthesis of NoC”, ICCD 2003.
[12] A. Jalabert et al., “×pipesCompiler: A tool for instantiating application specific NoC”, Proc. DATE 2005.
[13] T. T. Ye et al., “Analysis of power consumption on switch fabrics in network routers”, Proc. DAC 2003.
[14] F. Angiolini et al., “Contrasting a NoC and a Traditional Interconnect Fabric with Layout Awareness”, Proc. DATE, 2006.
[15] D. Starobinksi et al., “Application of network calculus to topologies using turn-prohibition”, IEEE Trans. Networking, 2003.
[16] S. N. Adya, I. L. Markov, ”Fixed-outline Floorplanning : Enabling Hierarchical Design”, IEEE Trans. VLSI, 2003.
[17] Cadence Enterprise, 2005, http://www.cadence.com
[18] Synopsys, 2005, http://www.synopsys.com
[19] W. J. Dally, B. Towles, ”Principles and Practices of Interconnection Networks”, Morgan Kaufmann , 2003.
[20] T. H. Cormen et al., ”Introduction to Algorithms”, The MIT Press, 1990.