

Implementación SystemC sintetizable de un procesador asociativo para el algoritmo de Dijkstra

Màrius Montón, David Castells, Antoni Portero, Jordi Carrabina

Dept. Microelectrònica i Sistemes Electrònics

Universitat Autònoma de Barcelona

{mariaus.monton, david.castells, antoni.portero, jordi.carrabina}@uab.es

Resumen

En este artículo se describe la arquitectura e implementación de un sistema para resolver el algoritmo de Dijkstra [5]. El objetivo de este algoritmo es encontrar el camino mínimo entre un nodo y todos los nodos alcanzables en un grafo dirigido. Nuestra arquitectura se basa en el uso de módulos asociativos para resolver los pasos más costosos del algoritmo.

En todo el flujo de diseño se ha usado SystemC, partiendo de una descripción funcional se ha refinado hasta la síntesis con Cynthesizer de Forte Design Systems [2].

1. Introducción

El algoritmo de Dijkstra para encontrar el camino mínimo entre dos nodos de un grafo es usado en múltiples aplicaciones: routing en redes, routing en NoCs (Network on Chip) [2], búsqueda en ontologías [1, 7], etc.

El algoritmo en pseudo-código es el siguiente:

Dado un grafo $G = (V, A)$, con costes positivos d_{ij} para todas las aristas, nodo de inicio s , y un conjunto de nodos P , se encuentra el camino mínimo de s a todos los demás nodos :

Inicialmente $P = \{s\}; D_s = 0; D_j = d_{sj}$ para $j \in N$.

Paso 1: (encontrar el nodo más próximo) Encontrar $i \notin P$ tal que

$$D_i = \min_{j \notin P} D_j$$

$P = P \cup \{i\}$. Si P contiene todos los nodos paramos. FIN

Paso 2: Para todo $j \notin P$

$$D_j = \min[D_j, D_i + d_{ij}]$$

Repite Paso 1.

Algoritmo 1. Algoritmo de Dijkstra para caminos óptimos

Como cada paso del algoritmo requiere un número de operaciones proporcional a $|N|$, y los pasos se iteran $|N|-1$ veces, se obtiene $O(|N|^2)$.

2. Diseño hardware

En este algoritmo, las operaciones más costosas son dos: buscar un valor mínimo de entre de un conjunto de valores dado, y saber si un nodo ya lo hemos visitado.

Estas dos operaciones, por ser las más costosas en tiempo, serán las que requerirán un diseño hardware *ad-hoc*.

2.1. Shifter-Sorter

Para obtener la arista de menor coste en cada iteración, usamos un shifter-sorter[3, 4]. En este módulo se irán almacenando ordenadamente las aristas D_{ij} de manera que obtener el mínimo de este conjunto tarda un solo ciclo.

Este módulo funciona como un registro de desplazamiento, de manera que al insertar un nuevo elemento éste se almacena en la posición adecuada para que la estructura siga estando ordenada, de ahí el nombre que le damos al módulo. Se puede insertar un elemento a cada ciclo de reloj, y a cada ciclo se puede leer el mínimo del conjunto, siendo por tanto la complejidad en tiempo de obtener el mínimo de un conjunto $O(1)$ y la complejidad en área de este módulo de orden $O(N)$. A este módulo para esta

obtiene un código SystemC con la descripción RTL del módulo con las modificaciones deseadas (*loop_unrolling*, *arrays* en memorias o en registros, etc.) que puede ser simulado junto con los demás módulos no sintetizables del sistema (generación de estímulos, etc.). En la Figura 3 se puede ver la implementación en SystemC con las macros para Synthetizer de nuestra memoria asociativa. Se especifica con la macro CYN_UNROLL que deseamos hacer *loop unrolling* del bucle *for* que busca el tag de entrada por todas las posiciones de memoria. Con la directiva CYN_PROTOCOL especificamos que ese bloque debe cumplir las especificaciones temporales indicadas mediante la sentencia *wait()*.

```

{
  CYN_PROTOCOL("assoc read input");
  tag_aux = tag.read();
}
if (fill_search == true) {
  CYN_PROTOCOL("fill_search");
  array[index++] = tag_aux;
  wait();
} else {
  for(int i=0; i < N_ASSOC_POS; i++) {
    CYN_UNROLL( COMPLETE, N, "match");
    if (array[i] == tag_aux) {
      match_aux = true;
    }
  } // for
  {
    CYN_PROTOCOL("assoc write result");
    if (match_aux == true) {
      match.write(true);
    } else {
      match.write(false);
    }
    wait();
  }
}
}

```

Figura 3. Código en SystemC para Synthetizer

Una vez validado el sistema a nivel SystemC-RTL, se procede a la obtención de los ficheros Verilog. La generación de los ficheros HDL se basa en el uso de unas bibliotecas proporcionadas por los fabricantes para mapear el RTL generado en ellas. Esto ha sido un problema debido a que ForteDS no soporta ninguno de los fabricantes mayoritarios de FPGAs, y por tanto hemos usado una biblioteca por defecto (y por tanto no optimizada ni en área ni en velocidad) para nuestra generación HDL.

Los ficheros resultantes pueden sintetizarse con cualquier herramienta de síntesis, en nuestro caso hemos usado QuartusII 4.2 de la casa ALTERA.

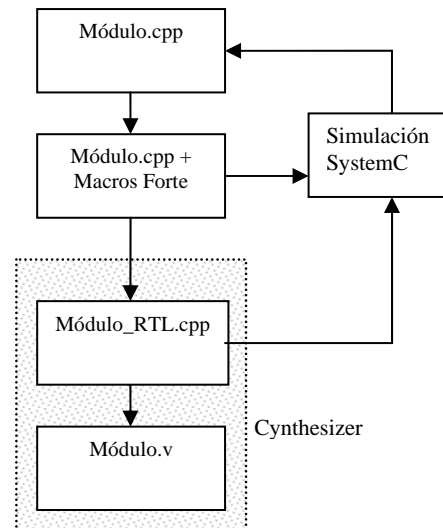


Figura 4. Flujo de diseño ForteDS

4. Resultados

Hemos sintetizado los ficheros resultantes de Cynthesizer con QuartusII 4.1, y eligiendo una FPGA STRATIS EP1S80F1020C5. En la tabla 1 podemos ver los resultados para grafos totalmente conexos con diferente número de vértices.

4.1. Ocupación

Los resultados muestran el crecimiento cuadrático en la ocupación debido al módulo ShifterSorter $O(|N|^2)$.

Esta ocupación puede mejorarse si se añade una funcionalidad extra al módulo ShifterSorter, que permita eliminar registros correspondientes a nodos ya resueltos. De esta manera, el tamaño del ShifterSorter será sólo de $O(|N|)$.

$ N $	LEs	RAM Bits	Clock (MHz)
3	1515	4992	97
5	2505	27136	82
8	2383	63552	83
10	6334	137216	53

Tabla 1. Resultados de síntesis para distinto número de vértices

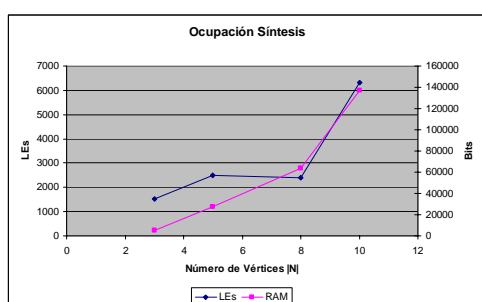


Figura 5. Ocupación en LEs y Bits de memoria

4.2. Velocidad

Como ya se ha comentado, el número de ciclos para encontrar los caminos mínimos será del orden $O(|N|^2)$, ya que aunque las operaciones más costosas (búsqueda de mínimo y presencia del vértice en conjunto de ya visitados) se realizan en tiempo 1, en el peor caso para cada vértice hay que iterar $N-1$ veces (Una iteración por cada vértice que sale de él).

$ N $	# Ciclos	Clock Speed (MHz)	Tiempo (ns)
3	26	97	260
5	64	82	768
8	183	83	2196
10	297	53	5643
16	783	45	17226
32	3029	30	99957

Tabla 2. Tiempos y ciclos de ejecución para distinto número de vértices

Estos resultados son preliminares, y se observa alguna anomalía debido a una implementación de los módulos no suficientemente detallada. Como se estudió en [4], la frecuencia de reloj del ShifterSorter no

depende del número de elementos a ordenar, por lo que esta descripción en SystemC del ShifterSorter no es totalmente equivalente a la descripción VHDL citada.

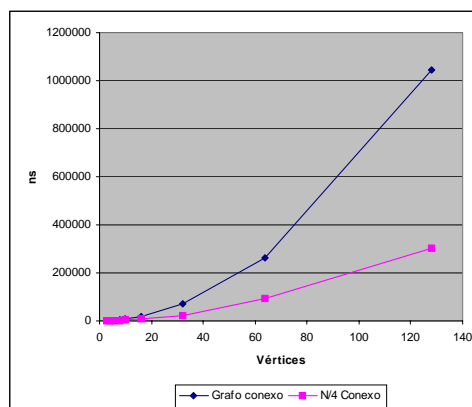


Figura 6. Tiempo y ciclos de ejecución

Estos resultados se obtienen aplicando el algoritmo en grafos totalmente conexos; si tenemos en cuenta las aplicaciones posibles, los grafos que habrá que tratar tendrán diferente conectividad. Así por ejemplo, en un router NoC de una topología Mesh, o torus-2D el grafo de routing que se obtiene tiene mucha menos conectividad [2, 6]. Se puede observar en la Figura 5 los dos tiempos de ejecución para solucionar grafos conexos y conexos $|N/4|$.

Conclusiones

Hemos presentado una implementación del algoritmo de Dijkstra en FPGA usando módulos asociativos para reducir la complejidad temporal del problema. Hemos visto también que esta solución puede ajustarse a un problema concreto de manera que se logra reducir la complejidad en ocupación de nuestro Hardware.

Hemos descrito también nuestra experiencia en el uso de SystemC para la descripción y síntesis de un sistema complejo.

Por último, se han comentado algunas modificaciones para mejorar la complejidad del módulo ShifterSorter hacia un orden lineal a costa de aumentar la complejidad del diseño.

También queremos resaltar la dificultad que representa lograr una descripción en SystemC exacta de unos módulos ya implementados en VHDL. En el caso del módulo ShifterSorter, no nos ha sido posible realizar una descripción en SystemC que resulte en el mismo RTL que la descripción previa hecha en VHDL.

Referencias

- [1] Ana G. Maguitman et al. Algorithmic Detection of Semantic Similarity. WWW 2005, May 10-14, 2005, Chiba, Japan.
- [2] D. Bertozzi, et al. NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip. IEEE transactions on parallel and distributed systems, vol. 16, no. 2, february 2005.
- [3] D. Castells-Rufas, M. Montón, J. Carrabina. Módulo de clasificación de datos con lógica de control mínima (Shifter Sorter). IV Jornadas de Computación Reconfigurable y Aplicaciones. (JCRA). Septiembre 2004.
- [4] D. Castells-Rufas, M. Montón, Ll. Ribas, J. Carrabina. High performance Parallel Linear Sorter Core Design. The international embedded Solutions Event. (GSPX). September 27-30, 2004.
- [5] E.W. Dijkstra. A Note on Two Problems in Connexion with Graphs. Numerische Mathematik, 1:269–271, 1959
- [6] S Murali, G De Micheli. Bandwidth-Constrained Mapping of Cores onto NoC Architectures. Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'04).
- [7] Iryna Gurevych et al. Semantic Coherence Scoring Using an Ontology. Proceedings of HLT-NAACL 2003.
- [8] ForteDS. <http://www.fortedds.com/products/cynthesizer.asp>. 2005.