Programación Automática de FPGAs en Aplicaciones SIMD: Estudio de la DWT 1-D

Germán León*, José M. Claver*, Germán Fabregat*

Resumen— **En este artículo presentamos el estado actual del desarrollo de una herramienta para la generación de circuitos en FPGAs a partir de especificaciones de alto nivel. La metodología utilizada está basada en la transformación inicial del algoritmo completo expresado en SmallTalk en una red de tablas (LUTs) que implementan las operaciones necesarias. La calidad de la circuitería resultante se garantiza utilizando técnicas basadas en la *inferencia de tipos* dependiendo del contexto de la aplicación.**

Inicialmente diseñada para aplicaciones simples que requieren únicamente el uso de circuitos combinacionales [9], [11], la herramienta se ha ampliado para aplicaciones SIMD mediante la inclusión del desenrollado de expresiones iterativas sobre vectores. Se ha realizado una prueba de su eficacia mediante el cálculo de la transformada wavelet de enteros (DWT).

Palabras clave— **Computación reconfigurable, arquitecturas reconfigurables, FPGA, transformada wavelet, compilación hardware.**

I. Introduccion

En la actualidad, se están realizando esfuerzos para el diseño de arquitecturas heterogéneas para aplicaciones de cálculo intensivo, que incrementan sus prestaciones mediante la utilización, parcial o total, de recursos hardware genéricos. Entre estas arquitecturas cabe destacar especialmente las dirigidas a la transformación de flujos contínuos de datos, propio de las aplicaciones multimedia, o a las aplicaciones empotradas que deben funcionar en entornos rápidamente cambiantes. Esta tendencia, que se concreta en las denominadas arquitecturas reconfigurables (RA), se ha visto beneficiada por la creciente integración y disponibilidad de recursos hardware [15]. La arquitectura de estas plataformas [8], y su programación [7], está siendo objeto en la actualidad de numerosos estudios en busca de configuraciones eficaces, que normalmente incluyen un conjunto de procesadores, circuitos reconfigurables para lógica aleatoria o aceleradores para computación intensiva, redes de elementos reconfigurables y varios tipos de memorias.

Los métodos utilizados para el desarrollo de aplicaciones para arquitecturas reconfigurables están dominados por las herramientas CAD/EDA industriales heredadas del diseño VLSI. Cada nuevo dispositivo que sale al mercado viene acompañada de un conjunto de librerías y herramientas para la planificación, simulación, emplazamiento y rutado. Estas herramientas trabajan a partir de lenguajes de descripción hardware (HDL) u otros métodos de especificación de circuitos lógicos de bajo nivel. Se han presentado algunas iniciativas en la dirección de arquitecturas públicas (Xilinx 6200) y entornos abiertos para el diseño de circuitos (Xilinx Jbits API). En el ámbito académico se están realizando esfuerzos en el diseño de herramientas de bajo nivel como VPR [1], librerías de componentes de circuitos como PAM-Xblox, compiladores construidos por encima de los HDL y generadores de lógica desde especificaciones de alto nivel [3], [6]. Pese a ello, aún no se han resuelto importantes problemas como son la portabilidad, reutilización y acceso abierto a los recursos de la arquitetura, que hacen prohibitivo un uso más general de las RA y la aplicación en ese campo de las técnicas comunmente utilizadas en ingeniería del software.

La aproximación tratada en este artículo está basada en los generadores de lógica desde especificaciones de alto nivel desarrolladas en [9], [11] y utiliza una modelización de lógica configurable desarrollada inicialmente para la arquitectura 6200 [10]. Este entorno permite describir tanto arquitecturas particulares como familias de arquitecturas, y proporciona meta-herramientas para el rutado de circuitos, conexión, dibujo, y edición. Éstas son utilizadas para soportar descripciones de alto nivel y el diseño de compiladores para generación lógica.

Las aplicaciones multimedia (imágenes, sonido, hipertexto) son un ejemplo claro de aplicaciones SIMD, caracterizadas por operaciones de grano fino y muy dependientes del flujo de datos. Para el aumento de las prestaciones en este tipo de aplicaciones la utilización de RA es una alternativa a las extensiones vectoriales SIMD de algunos procesadores de propósito general (caracterizadas por la teórica flexibilida de su uso) y a los ASICs que acompañan o integran los DSPs (menos flexibles y más específicas pero con un mayor rendimiento). El uso de RA potencialmente puede beneficiarse de las mejores características de ambos.

En este tipo de aplicaciones, el cálculo de la transformada wavelet (DWT) está siendo extensamente utilizada en los sistemas de almacenamiento y transmisión de imágenes y vídeo. En particular, existe un creciente interés por las DWT reversibles (sin pérdidas) basadas en aritmética entera. Dado que se trata de un caso paradigmático de las aplicaciones SIMD ha sido elegido como caso de estudio en este trabajo.

El resto de este artículo se organiza como sigue. A continuación (sección III) se analizan los problemas de la generación de código para arquitecturas reconfigurables, y las soluciones utilizadas en traba-

*Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12071–Castellón, Spain; {leon,claver,fabregat}@icc.uji.es. Este trabajo ha sido parcialmente subvencionado por el proyecto CICYT TIC2000-1151-C07-06.

jos previos. En la siguiente sección III se describe el proceso de compilación sobre lógica reconfigurable utilizado. El algoritmo elegido como caso ejemplo y sus aplicaciones se tratan en la sección IV y, finalmente, los primeros resultados obtenidos se comentan en la sección V.

II. Descripción del problema

En esta sección introducimos primero algunas consideraciones acerca de la implementación de algoritmos en RA, centrándonos en el caso de sistemas basados en FPGAs. En II-B discutimos diversos aspectos de la programación de aplicaciones SIMD en RA. Por último, en II-C presentamos a grandes rasgos el funcionamiento del compilador que genera la lógica a partir de una especificación del algoritmo en lenguaje SmallTalk.

A. Implementación en arquitecturas reconfigurables

Las arquitecturas reconfigurables pueden combinar recursos hardware de diferente granularidad: buses, líneas, conmutadores, memorias, procesadores, etc. Los componentes de grano fino en las RA están constituidos por biestables y pequeñas tablas de memoria (LUT) direccionadas por un conjunto de señales. Las LUTs suelen contener pocos bits (entre 1 y 4), y forman el recurso básico para la generación de lógica combinacional. Desde el punto de vista de la síntesis lógica, una LUT de n bits es la forma más general de producir cualquier función lógica de n variables booleanas. Existen algoritmos suficientemente conocidos que permiten adaptar cualquier tabla lógica para una arquitectura particular con un tamño de LUT dado.

De esta forma, una expresión sencilla cualquiera como por ejemplo $z = 3x+4(y+2t)$ se implementaría generando una LUT adecuada para cada operador que transformara los datos de entrada en la salida adecuada. Para diseñar este circuito sería necesario tener encuenta el tipo de tales datos. Los diferentes operadores generados se conectarían mediante los recursos de intercomunicación según las dependencias de datos marcadas por la expresión. Las figuras 1 y 2 ilustran este proceso. Los cuadrados pequeños de la parte superior representan los datos de entrada, y los rectángulos grises los operadores. Es interesante destacar cómo la implementación óptima en una RA no tiene por qué utilizar los operadores habituales, que aparecen en la figura 1, pues las LUTs otorgan la flexibilidad necesaria para generar los circuitos combinacionales más adecuados a cada caso. En la figura 2 se aprecia cómo las operaciones intermedias con constantes se pueden fusionar en una única LUT que trabaje con dos variables a su entrada.

B. Consideraciones de programación

Como se ha comentado en el ejemplo anterior, la generación de los operadores viene determinada por el tipo de los datos con que se está trabajando. Al mismo tiempo, la mayor parte de los sistemas actuales que permiten generar de forma automática circuitos lógicos sobre RA a partir de especificaciones de alto nivel, parten de funciones en C, C++ o Java. Estos lenguajes requieren de una declaración de tipos previa al uso de las variables a partir de unos tipos escalares adecuados a las unidades aritmético lógicas de los procesadores actuales. Estos tipos de datos por lo general no se adaptan adecuadamente a la implementeción en RA que se ha explicado, lo que da lugar a circuitos poco eficientes en cuanto a consumo de recursos y prestaciones, debido a la enorme distancia semántica entre los lenguajes utilizados y los recursos del hardware.

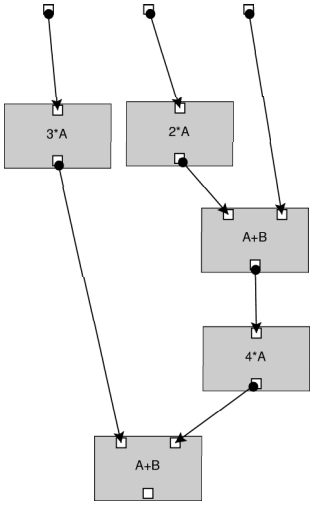

Fig. 1. Descomposición de la expresión $z = 3x + 4(y + 2t)$ en LUT

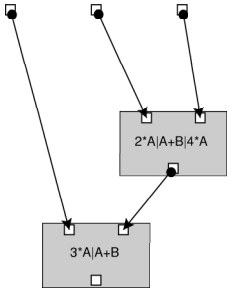

Fig. 2. Descomposición mejorada de la expresión $z = 3x + 4(y + 2t)$ en LUT

Otro problema viene provocado por el modelo de ejecución secuencial de los lenguajes considerados, completamente distinto al propio de las RA y del hardware en general. De hecho, la dificultad de la utilización eficaz de las extensiones SIMD de los procesadores actuales, que se adaptan exclusivamente a cierto tipo de problemas, es una consecuencia suficientemente conocida de este problema.

Frente a esta problemática, un lenguaje estrictamente orientado a objetos como es el SmallTalk ofrece ventajas evidentes para ser utilizado en el proceso de generación de lógica para RA. Por una parte, utiliza una asignación de tipos retardada hasta que los mensajes, que podríiamos considerar equivalentes a los operadores del ejemplo anterior, se envían a un objeto previamente instanciado. Por otra parte se trata de un lenguaje interpretado sobre una máquina

virtual de tal modo que la descripción de alto nivel de las operaciones se mantiene de forma puramente simbólica hasta el momento mismo de la ejecución. Estas características van a permitir que los tipos de los datos se puedan asignar de forma adecuada para la generación óptima de los operadores como se verá más adelante. El uso de SmallTalk permite también de forma similar la compilación desde el mismo tipo de especificación simbólica sobre un sistema empotrado heterogéneo, como se describe en [5]. De esta forma, en el mismo sistema se puede integrar la programación y simulación de aplicaciones y su compilación sobre arquitecturas heterogéneas sin ningún coste de desarrollo adicional.

C. Modelo de ejecución

El modelo de ejecución seguido por el compilador se adapta a la implementación en LUT de las operaciones según lo visto en el ejemplo anterior. Tras un análisis del flujo de datos de las expresiones se genera un grafo jerárquico cuyos nodos son las operaciones que serán implementadas en las LUT de la RA, y cuyos arcos son las dependencias de datos. En cada nivel de la jerarquía los nodos son evaluados sobre el conjunto de sus entradas para obtener el conjunto completo de salidas posibles. Esta evaluación se realiza para todos los nodos, teniendo en cuenta posibles realimentaciones de datos, y acaba dando para cada nodo una descripción tabular de su comportamiento que permite directamente su implementación sobre las LUTs de la RA.

Como se ve, este método presenta una independencia total hacia el tipo de los datos de entrada y salida de cada operador, y sólo tiene en cuenta los valores reales (conjuntos de entrada y salida de cada LUT) que van a intervenir en la realización de las operaciones. De esta forma, los valores en la entrada de cualquier LUT pueden codificarse de forma adecuada para tratar de forma óptima bien la lógica de la misma, bien el ancho de los buses de comunicación entre distintas LUT. La evaluación de los nodos y la recodificación de los valores de entrada y salida se realiza de forma muy sencilla en un lenguaje débilmente tipado como el SmallTalk.

III. Compilación y síntesis lógica

En este apartado se realiza una descripción más detallada del proceso de compilación, ilustrándolo con algunos ejemplos simples. Se pone de manifiesto especialmente las aportaciones realizadas en el tratamiento de estructuras de control iterativas.

Como se ha comentado antes, el compilador comienza generando un grafo de flujo de datos jerárquico (GFDJ) cuyos nodos indican las operaciones a realizar y cuyos arcos marcan las dependencias de datos entre ellos. Este grafo sufre unas primeras transformaciones para propagar constantes, eliminar expresiones comunes y quitar código no accesible. En este momento, con las entradas del GFDJ bien definidas, se realiza un proceso de asignación de tipos para los valores de entrada que continúa con la evaluación del conjunto de entradas y la inferencia de tipos para los arcos de salida e intermedios. Este proceso concluye con la generación de las tablas de verdad para todos los nodos del grafo, que es enviada a un conjunto de herramientas estándar para el particionado y adaptación a las características propias de la RA objeto de la compilación (ver [13]).

El método que se ha descrito anteriormente funciona de forma inmediata para expresiones de cualquier complejidad, pero requiere distintas consideraciones cuando se debe aplicar a estructuras algorítmicas iterativas, como las requeridas para tratar vectores. El problema básico es el desenrollado de un bucle, pero al tener como objeto una RA se debe hacer un uso adecuado de los recursos de tal forma que el número de iteraciones ejecutadas en paralelo pueda ser contenido en el dispositvo a programar. En el caso de que el número de iteraciones desdoblado sea inferior del requerido por el algoritmo, se deberá proveer un mecanismo que permita la ejecución iterativa. Con este fin se han desarrollado dos nuevos mensajes iterativos en SmallTalk que permiten considerar ambos casos:

- **collect:** es un mensaje que enviado sobre un rango y con un bloque como parámetro (ver figura 3) realiza el desenrollado total del bucle para todo el rango. Posteriormente se realiza el análisis descrito antes para el código desenrollado.
- **do:** es un mensaje que generará la ejecución iterativa, tantas veces cómo indique el rango, de la operación contenida dentro del bloque (ver figura 3), generando un circuito secuencial síncrono. Para la correcta implementación del código contenido en el bloque, que probablemente utilizará valores calculados en iteraciones previas, realiza un análisis de los datos consultados y modificados en cada iteración para, por una parte, completar el conjunto de valores de entrada requerido por el proceso de compilación descrito y por otra genera registros para el almacenamiento de los valores que se van a utilizar en iteraciones futuras y multiplexores para seleccionar los datos de entrada correctos en cada caso.

Las figuras 4 y 5 muestran la descomposición en LUT y registros (en su caso) resultantes tras aplicar los mensajes descritos al ejemplo sencillo del producto escalar de la figura 3.

IV. La transformada wavelet de enteros

Actualmente, la transformada wavelet está siendo ampliamente utilizada en el tratamiento y compresión de señales, imágenes y vídeo. Hasta hace poco, su uso se había limitado a aplicaciones de compresión con pérdidas, debido que esta transformación produce coeficientes en coma flotante que no son adecuados para las aplicaciones de codificacción con pérdidas. La introducción de transformadas wavelet que emplazan enteros en enteros, per-

```
nuevo:=0.
( 1 to: 20) do:[:i|
        nuevo:=nuevo+((A at:i) *(B at:i)).
].

nuevo:=0.
( 1 to: 2) collect:[:i|
        nuevo:=nuevo+((A at:i) *(B at:i)).
].
```

Fig. 3. Mensajes **collect:** y **do:** aplicados al producto escalar.

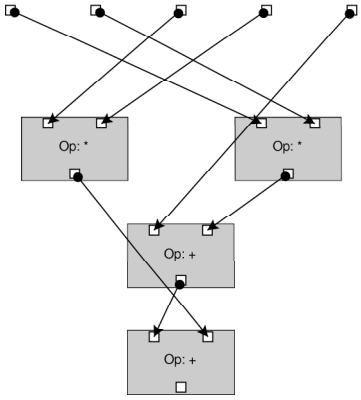

Fig. 4. Resultado tras aplicar el mensaje **collect:**.

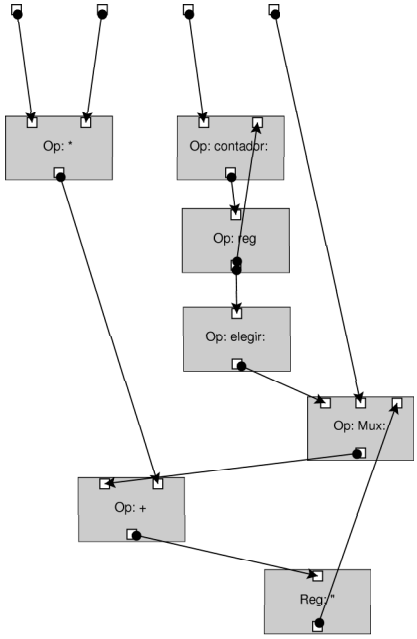

Fig. 5. Resultado tras aplicar el mensaje **do:**.

$$d^{(1)}[n] = x[2n+1] - x[2n]$$
$$s[n] = x[2n] + \lfloor \frac{d^{(1)}[n]}{2} \rfloor$$
$$d[n] = d^{(1)}[n] + \lfloor \tfrac{2}{8}(s[n-1] - s[n]) + \tfrac{3}{8}(s[n] - s[n+1]) + \tfrac{2}{8}d^{(1)}[n+1] + \tfrac{1}{2} \rfloor$$

Fig. 6. Transformada wavelet S+P

mitiendo la reconstruccióón perfecta de la imagen original, ha provocado el actual interes por la codificación de imágenes sin pérdidas [12], [2], [4]. Una de las ventajas más importantes de las tranformadas wavelet de enteros reversibles es la posibilidad de generar un flujo de bits completo de la imagen original que puede ser reconstruido con o sin pérdidas por el decodificador. En [2], Calder *et al.* introducen un método para construir transformadas wavelet de enteros reversibles mediante el esquema *lifting* de [14].

En este trabajo estudiamos la transformadas wavelet de Said y Pearlman [12], una de las más conocidas por sus buenas prestaciones medias y su uso en el algoritmo de compresión SPIHT. En la Figura 6 se muestra esta transformada en formato lifting.

V. Resultados Experimentales

La transformada wavelet para enteros descrita en la sección anterior ha sido implementada en SmallTalk y se ha tratado según lo descrito para su implementación sobre RA. La figura 7 muestra el código en SmallTalk utilizado. Las figuras 8 y 9 muestran la descomposición en LUT obtenida para los casos **collect:** y **do:** respectivamente.

VI. Conclusiones y trabajo futuro.

Se ha presentado un entorno de desarrollo capaz de generar hardware sobre RA a partir de especificaciones de alto nivel en lenguaje SmallTalk. El uso de este lenguaje es crucial por sus características que permiten retardar la asignación de tipos y evaluar los conjuntos de datos de entrada y salida durante la compilación. En particular se ha trabajado en un método eficaz para permitir gestionar de forma adecuada los bucles y demás estructuras iterativas del lenguaje. Se han presentado los primeros resultados de aplicar la gestión de bucles, y un caso de estudio, la transformada wavelet de enteros, sobre el que se está trabajando.

A la hora de finalizar este artículo ya se ha completado un nuevo mensaje SmallTalk que combina los **collect:** y **do:** permitiendo desenrollar un número dado de veces e iterar las necesarias para completar las iteraciones requeridas inicialmente. Un aspecto interesante en el que se está trabajando es la automatización de este proceso para una arquitectura dada, es decir, según los recursos de que se dispone y la complejidad de los operadores necesarios, determinar de forma automática el número de iteraciones a ejecutar en paralelo.

También se está completando la implementación del algoritmo wavelet presentado sobre diversas tarjetas con FPGA para realizar medidas sobre prestaciones. En esta línea es de especial importancia gestionar adecuadamente la comunicación entre el dispositivo reconfigurable y el host, algo que implica también el número de recursos disponibles y su utilización óptima.

Referencias

[1] V. Betz, J. Rose. VPR: A new packing, placement and routing tool for FPGA reasearch. *Field Programmable Logic and Applications*, LNCS, 1997.

[2] R. Calderbank, I. Daubechis, W. Sweldens, B.-L. Yeo. Wavelet transforms that map integers to integers. *Jour-*

```
( 1 to: 2 ) collect: [:i|
  |Old Now New Lold Lnow Lnew Hnow Hnew
   OldI NowI NewI|

  Old:=H at:(i-1).
  OldI:=L at:(i-1).
  Now:=H at:i.
  NowI:=L at:i.
  New:=H at:i+1.
  NewI:=L at:(i+1).

  Lnow:=Now+NowI.
  Lnew:=New+NewI.
  Lold:=Old+OldI.
  Hnow:=Now-NowI.
  Hnew:=New-NewI.

  Hnow:=Hnow-((((Lold-Lnow)+(Lnow-Lnew)-Hnew)/2)
        +(Lnow-Lnew)+3)*8.
  H at: i put: (Hnow).]
```

Fig. 7. Código de la transformada wavelet en SmallTalk.

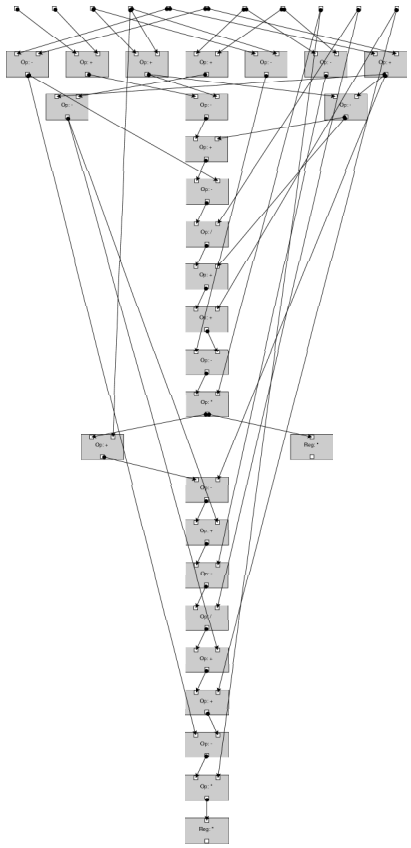

Fig. 8. Resultado tras aplicar el mensaje **collect:** a la transformada wavelet.

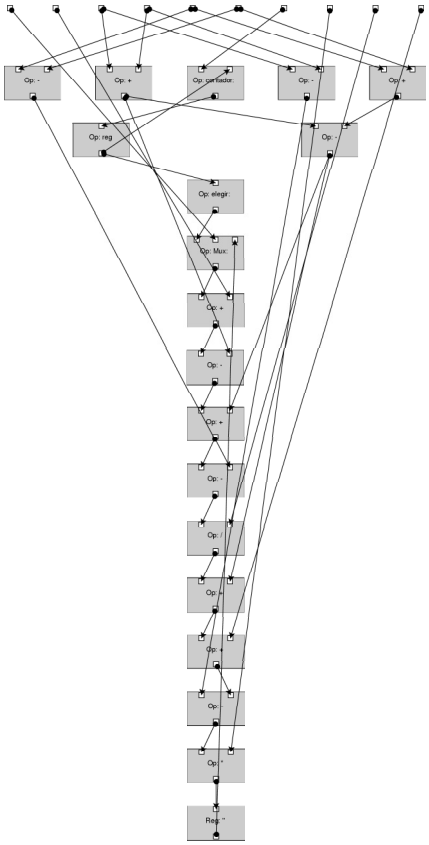

Fig. 9. Resultado tras aplicar el mensaje **do:** a la transformada wavelet.

nal of Applied and Computational Harmonics Analysis, 1997.

[3] N. Chu, K. Sulimma, A. Dehon. Object oriented circuit-generators in java. *FCCM'98*, Abril, 1998.

[4] S. Dewitte, J. Cornelis. Lossless integer wavelet transform. *IEEE Transactions on Image Processing*, vol. 4, pp. 158–160, Junio 1997.

[5] G. Fabregat, G. León, B. Pottier, O. Le Berre. Embedded system modelling and synthesis in OO environments: a smart-sensor case study *Compiler and Architecture Support for Embedded Systems*, Washington D.C., EE.UU, Septiembre 1999.

[6] J. Llopis, B. Pottier. Smalltalk blocks revisited, a logic generator for FPGAs. *FCCM'96*, Napa, CA, 1996.

[7] E. Lee. What's ahead for embedded software. *Computer IEEE*, Septiembre, 2000.

[8] C. Kozyrakis, D. Patterson. A new direction for computer architecture research. *Computer IEEE*, Noviembre, 1998.

[9] L. Lagadec, B. Pottier, O. Villellas. Compiling for reconfigurable architectures, a Reed-Solomon error correction case estudy. *A&S, Univ. de Bretagne Occidental*, Technical Report TR-2000/2, 2000.

[10] L. Lagadec. Abstraction, modélisation et outils de CAO pour les circuits intégrés reconfigurables. PH.D. *Univ. de Rennes 1*, Ph. D. thesis, 2000.

[11] L. Lagadec, B. Pottier, O. Villellas. From high level finctions to FPGA code generation. A LUT based approach. *A&S, Univ. de Bretagne Occidental*, Technical Report TR-2002/1, 2002.

[12] A. Said, W. Pearlman. An image multiresolution representation for lossless and lossy compression. *IEEE Transactions on Image Processing*, vol. 5, pp. 1303–1310, Sept. 1996.

[13] E. M. Sentovich. SIS: a system for sequential circuit synthesis. *EECS, Berkeley*, Technical Report UCB/ERL M92/41, Mayo 1992.

[14] W. Sweldens. The lifting scheme: A custom-design construction of biorthogonal wavelets. *Journal of Applied and Computational Harmonics Analysis*, vol.3(2), pp. 186–220, Abril 1997.

[15] J. Villasenor, W. Mangione-Smith. Configurable Computing. *Scientific American*, pp. 66–71, Junio 1997.