FRONT-END C++ PARA PAQUETES DE BDDs

R. Capillas, J. Riera y J. Carrabina.
Departamento de Informática.
Facultad de Ciencias (Edificio Cn).
Campus Universitario.
08193 - Bellaterra (Barcelona).
Teléfono: 581-10-78. FAX: 581-30-33. e-mail: capillas@cnm.es

RESUMEN.- La representación de expresiones Booleanas mediante BDDs (Binary Decision Diagrams) es una de las más extendidas actualmente en aplicaciones de soporte al diseño de circuitos digitales debido a que proporcionan una representación canónica y unos algoritmos de manipulación eficientes. Habitualmente, los programadores de dichas aplicaciones se apoyan en paquetes de BDDs existentes, los cuales les proporcionan la posibilidad de trabajar con BDDs de una manera transparente y sencilla. La diversidad existente de paquetes de BDDs, con características y eficiencia distintos, hace pensar en algún tipo de unificación de criterios. En este artículo se presenta una herramienta que permite al programador de dichas aplicaciones abstraerse del paquete de BDDs con el cual se va a trabajar, y poder cambiar de paquete sin necesidad de tener que tocar ninguna línea de código de su aplicación.

1. INTRODUCCIÓN

El incremento de aplicaciones y herramientas que utilizan Diagramas de Decisión Binarios (BDDs) [1,2,3] en el campo de la síntesis, verificación y estimación de potencia consumida en circuitos digitales, entre otros, crean la necesidad de paquetes software que los manejen de manera eficiente. Dicho software se presenta habitualmente en forma de librerías de funciones listas para ser *linkadas* con la aplicación concreta. La aparición en los últimos años de diversos paquetes de BDDs, habitualmente en C y creados en el ambiente Universitario [4,6,7], crean la necesidad de algún tipo de unificación de criterios. Es decir, las diferencias entre las características que dichos paquetes ofrecen, en términos de utilización y tratamiento de la memoria, así como algorítmicamente, plantean las siguientes preguntas:

1. ¿Qué es mejor, utilizar alguno de los paquetes de BDDs existentes aunque no se adapten completamente a los requerimientos de la aplicación, o por el contrario crear rutinas propias de construcción y manipulación de BDDs que se adapten perfectamente a la aplicación concreta, pero con el esfuerzo en tiempo de diseño e implementación que esto conlleva?
2. ¿Con que criterios el programador de dichas aplicaciones escoge un paquete de BDDs para su aplicación concreta ? ¿Qué paquete es mejor dependiendo de las necesidades de la aplicación ?
3. ¿Cuanto esfuerzo costará la re-definición de dicha aplicación en el caso de que se quiera "*cambiar*" a otro paquete de BDDs con características "*más*" adecuadas a las necesidades de la aplicación concreta?

Si el programador pudiese, de alguna manera, *probar* de una forma rápida y sencilla distintos paquetes de BDDs sobre su aplicación sin necesidad de modificarla, entonces como respuesta a la primera pregunta diríamos que es mucho mejor hacer uso de un paquete de BDDs

existente, eliminando de esta manera una porción considerable del tiempo de diseño e implementación de nuestra aplicación.
El trabajo presentado en este artículo se centra en esta posibilidad y trata de responder al resto de preguntas mostrando una solución sencilla a la necesidad de poder probar distintos paquetes de BDDs sobre una aplicación concreta. Dicha solución esta basada en una nueva herramienta para la programación de aplicaciones que requieran la potencia de los BDDs, llamada *front-end C++ para paquetes de BDDs*. Es decir, consiste en un conjunto de funciones que encapsulan (esconden) las llamadas a las librerías de BDDs existentes, de manera que cambiar la librería/paquete a la cual la aplicación hace referencia (a la cual está *linkada*) es tan sencillo como recompilar dicha aplicación sin necesidad de retocar ninguna línea de código. De esta manera el programador puede diseñar su aplicación sin tener en cuenta el paquete de BDDs que más le conviene de cara a la eficiencia de su programa, y una vez terminado puede realizar las pruebas que considere oportunas cambiando de paquete sin demasiado esfuerzo. Así puede escoger que paquete se adecua más a las necesidades de su aplicación, y de esta manera obtener una mejor eficiencia por un lado en el tiempo requerido para el diseño del programa y por el otro en la eficiencia del programa en sí, en términos de utilización de memoria y tiempo de ejecución.

2. FRONT-END C++ PARA PAQUETES DE BDDS

El *Front-End* para paquetes de BDDs presentado en este artículo, ha sido desarrollado para programadores de aplicaciones en C++ basadas en BDDs. Se ha escogido el lenguaje C++ por ser uno de los lenguajes Orientados a Objeto más extendidos actualmente y con mayor proyección futura. La función del *front-end* es hacer de *interfaz* entre el programador y los paquetes de BDDs existentes tal y como muestra la Figura 1.

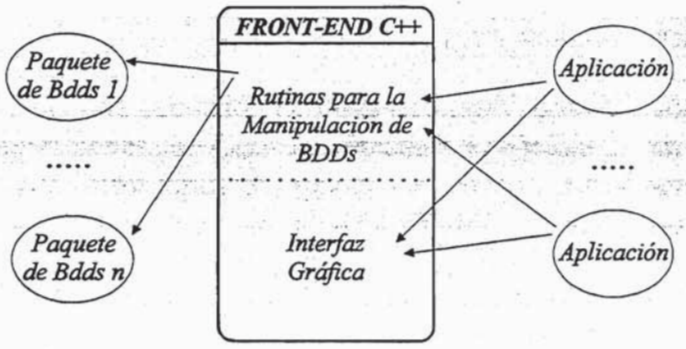

Figura 1 : Idea de la funcionalidad del Front-End C++.

El *front-end* ha sido diseñado para mantener la ortogonalidad entre los distintos paquetes. Es decir, si un tipo de operación no estaba definido en un paquete determinado, en la mayoría de los casos ésta ha sido programada en C++ para dicho paquete. De esta manera se ha conseguido un conjunto equilibrado de rutinas de manipulación de BDDs generalizado para todos los paquetes.
El *front-end* no solo proporciona las rutinas de manipulación de BDDs existentes en los paquetes a los cuales esconde, sino que también está provista de un conjunto de rutinas de *interfaz* gráfica que permiten dibujar un espacio de BDDs en una ventana del sistema, permitiendo así al usuario hacer un seguimiento de como es y como se comporta un determinado conjunto de BDDs durante la ejecución.
Adicionalmente, y tomando como punto de partida la división que realizo Bryant en [3] de los distintos tipos de operaciones sobre BDDs según su complejidad en términos de tiempo, el *front-end* es capaz de evaluar, automáticamente y de manera transparente al usuario, el

tiempo promedio que requieren las operaciones según su complejidad. De esta manera es posible comparar la eficiencia de nuestra aplicación según el paquete de BDDs que se este utilizando por debajo del *front-end*, así como establecer algún punto de comparación en términos de eficiencia entre los distintos paquetes. La división utilizada en el *front-end* está descrita en la Tabla I.

Complejidad/Tipo	Descripción de la operación
$O(n^3)$	Tipo COMPOSITION [3].
$O(n^2)$	Tipo APPLY [3].
O(nlog(n))	Tipo RESTRICT [3].
O(n)	Tipo SATISFY-ONE [3].
O(1)	Son aquellas operaciones que consisten en simples consultas, y por tanto requieren un tiempo constante.
Tabla	Son aquellas para la realización de las cuales es necesario acceder a alguna de las Tablas Hash internas que utiliza el *front-end*. Tienen complejidad O(1), aunque las diferenciamos de las anteriores debido a que estas son externas al paquete de BDDs.
Memoria	Son aquellas operaciones de gestión de la memoria .
Constructor / Destructor	Llamadas a constructores y destructores internos del *front-end*.

Tabla I : División de las operaciones que realiza el front-end según su complejidad.

Actualmente el *front-end* esta preparado para encapsular rutinas de los paquetes de Long [4], Brace [5] y Somenzi [6], aunque puede ser adaptado a cualquier nuevo paquete de BDDs de manera sencilla y en un tiempo razonable (en unas pocas horas). La Tabla II muestra el conjunto de funciones de manipulación de BDDs que ofrece el *front-end*, su complejidad y disponibilidad para los distintos paquetes de BDDs.

Funciones de la clase Manager (CBddManager)	Complejidad	Long [4]	Brace [5]	Somenzi [6]
bddAnd, bddNand, bddOr, bddNor, bddXor, bddXnor, bddNot, bddIte.	$O(n^2)$	S	S	S
bddCofactor (generalizado), bddCompose.	$O(n^3)$	S	S	S
bddDC.	O(1)	P	P	P
bddDelAssignedName, bddDelVar.	Tabla	P	P	P
bddDependsOn, bddSatisfy, bddSatisfyFraction.	O(n)	S	I	I
bddDynamicReorder.	O(1)	S	N	S
bddExists, bddForall, bddRelProd.	O(nlog(n))	S	I	I
bddGC.	Memoria	S	S	S
bddImplies.	$O(n^2)$	S	S	S
bddIntersects.	$O(n^2)$	S	I	S
bddNewVarAfter, bddNewVarBefore, bddNewVarLast, bddNewVarFirst.	Tabla	S	S	S
bddOne, bddZero, bddTotalSize, bddTotalVars..	O(1)	S	S	S
bddOverflow	O(1)	S	S	N
bddReduce	$O(n^2)$	S	I	S
bddReorder.	Memoria	S	N	S
bddSatisfySupport	O(n)	S	N	N
bddSubstitute	$O(n^2)$	S	I	I
bddSupport	O(n)	S	S	S
bddSwapVars	O(n)	S	N	S
bddTotalMem, bddTotalMemUsed.	O(1)	S	N	N
bddVarWithIndex.	O(1)	S	N	S
bddVarWithLevel.	O(1)	S	N	S
bddWithAssignedName, bddWithVarName.	Tabla	P	P	P
Constructores, Destructor.	Cons./Dest.	P	P	P
getAssignedNameTable, getBddManager, getVarNameTable.	O(1)	P	P	P

Funciones de la clase BDD (CBdd).	Complejidad	Long [4]	Brace [5]	Somenzi [6]
bddApplyFunction.	O(n)	P	P	P
bddAssignName.	Tabla	P	P	P
bddElse, bddThen.	O(1)	S	S	S
bddFree, bddUnfree.	Memoria	S	S	S
bddIf.	O(1)	S	S	S
bddIfLevel.	O(1)	S	S	S
bddLevels.	O(n)	I	I	I
bddNonComplemented.	O(1)	S	S	S
bddSize.	O(n)	S	S	S
bddType.	O(1)	S	S	I
Constructores, Destructor.	Cons./Dest.	P	P	P
getAssignedName, getBdd, getBddManager.	O(1)	P	P	P
getDCSet, getOffSet, getOnSet.	$O(n^2)$	P	P	P
getRefNumber.	O(1)	S	S	S
getVarName.	Tabla	P	P	P
isBddComplementary, isBddConstant, isBddNull, isBddOne, isBddZero.	O(1)	P	P	P
setBdd, setBddManager, setBddManager.	O(1)	P	P	P
showMe	no clasificada	P	P	P

Tabla II : Conjunto de funciones de manipulación de BDDs del *front-end*, con su correspondiente complejidad y su disponibilidad para los distintos paquetes. La nomenclatura utilizada es la siguiente : **P** significa función propia del *front-end*, **S** significa función disponible dentro del paquete de BDDs, **I** significa función implementada en el *front-end* y **N** significa que la función no esta disponible en el paquete de BDDs y tampoco ha sido implementada en el *front-end* (representadas también por recuadros sombreados).

2.1 bddDraw. Un ejemplo de aplicación

Uno de los puntos débiles de la representación de expresiones Booleanas mediante BDDs es lo poco intuitiva que se convierte su representación al ir aumentando éstos de tamaño. Esto complica mucho la tarea de depuración de aplicaciones que usen BDDs y crea la necesidad de tener alguna herramienta que muestre, dados un conjunto de expresiones Booleanas, su correspondiente espacio de BDDs. Con la finalidad de cubrir dicha necesidad y como ejemplo de una aplicación dependiente del *front-end* que utilice su potencia, se ha diseñado *bddDraw*. Esta aplicación ha sido creada como una herramienta tanto de soporte al programador de aplicaciones que requieran la potencia de BDDs, como a nivel educativo, ya que permite mostrar de una manera visual como a partir de una suma de productos se puede ir formando un BDD. Esto es importante porque permite seguir los cambios en la representación del BDD, número de nodos, su distribución, ordenación ... etc., dando al usuario una completa visión de como es el espacio de BDDs y como va cambiando. La Figura 2 muestra el aspecto del menú principal de *bddDraw* utilizando el paquete de Long [4], así como de la ventana que muestra un espacio de BDDs (en este caso el BDD correspondiente a la suma de productos $a + b\bar{c}$).

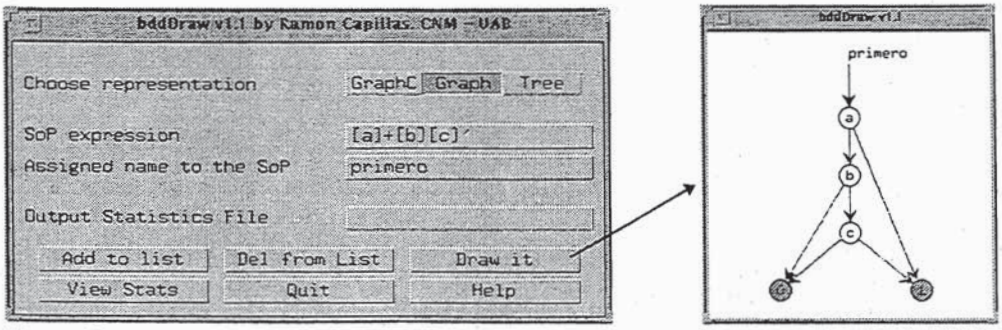

Figura 2 : Aspecto de algunas ventanas de la aplicación *bddDraw*.

3. RESULTADOS

Utilizando la división de los tipos de operaciones sobre BDDs explicada en la sección 2 se han realizado algunas comparaciones entre la eficiencia de los distintos paquetes de BDDs mediante la ejecución de algunos algoritmos de tratamiento de circuitos digitales sobre circuitos de ISCAS [7]. En particular la Tabla III muestra la suma total de los tiempos requeridos para leer y estimar la potencia consumida por los circuitos C17, C432, C499, C880, C1355, C1908, C2670 utilizando densidades de transición [8].
Debido a que los paquetes de Long [4] y Somenzi [6] ofrecen algoritmos de reordenación dinámica de las variables, estos se han probado sin reordenación dinámica y con un algoritmo de reordenación dinámica tipo sift (es decir, se mueve cada variable a través del orden para encontrar su posición optima, asumiendo que el resto de variables están fijas) .

PAQUETE		$o(n^3)$	$o(n^2)$	o(nlog(n))	o(n)	o(1)
LONG [4]	*Número Operaciones*	20299	46003	22162	26251	1643505
sin reordenación.	*Tiempo Total (seg.)*	*2,23346*	*32,61810*	*2,49976*	*10,3816*	*48,7148*
LONG [4]	*Número Operaciones*	20299	46003	22162	26251	1640018
reordenación sift.	*Tiempo Total (seg.)*	*2,43334*	*31,96720*	*2,70030*	*9,8167*	*48,5991*
BRACE [5]	*Número Operaciones*	26176	62204	22162	20840	1925719
	Tiempo Total (seg.)	*20,16620*	*30,21680*	*4,90055*	*46,8329*	*33,0832*
SOMENZI [6]	*Número Operaciones*	24826	74968	27291	30686	3999853
sin reordenación.	*Tiempo Total (seg.)*	*2,45027*	*7,04942*	*2,95025*	*169,7000*	*102,7500*
SOMENZI [6]	*Número Operaciones*	24826	74968	27291	30686	3990695
reordenación sift	*Tiempo Total (seg.)*	*2,68381*	*6,93368*	*2,81697*	*167,3360*	*99,5342*

Tabla III : Número de operaciones y tiempo total en segundos requeridos para la lectura y posterior estimación de la potencia consumida en los circuitos de ISCAS [7] : C17, C432, C499, C880, C1355, C1908, C2670.

Nótese que el número de veces que se ejecuta una operación de un determinado tipo difiere entre paquetes debido a que no todas las operaciones son llamadas directas a la librería, y por tanto su peso puede quedar distribuido entre otros tipos de operaciones, o con sucesivas llamadas a otra operación del mismo tipo dependiendo de su implementación.

4. CONCLUSIONES

En este artículo se ha presentado una herramienta de apoyo para programadores de aplicaciones que requieran la potencia de los BDDs. Dicha herramienta se presenta en forma de librería C++ proporcionando un conjunto equilibrado de rutinas para manipular BDDs que utilizan llamadas a las rutinas de los paquetes de a los cuales encapsula, y que además ofrece una serie de rutinas propias, como son una *interfaz* gráfica que permite visualizar un espacio de BDDs, así como estrategias para evaluar el coste en tiempo de ejecución de cada función del *front-end* de manera transparente al programador.

5. LINEAS FUTURAS

En el punto actual del proyecto estamos ultimando la incorporación al *front-end* del paquete de Shiple [9], y preparando una amplia gama de resultados para poder comparar los paquetes de BDDs con distintos algoritmos de tratamiento de circuitos digitales. Una vez obtenidos estos resultados se podrán optimizar al máximo las implementaciones

6. REFERENCIAS

[1] C. Y. Lee. "Representation of switching circuits by binary-decision programs". Bell System Technical Journal 38, pp. 985-999. 1959.

[2] S. B. Akers. "Binary Decision Diagrams". IEEE Transactions on Computers C-27, pp. 509-516. August 1978.

[3] Randal E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". IEEE transactions on computers, vol. C-35. N° 8, pp 677-691. August 1986.

[4] D. E. Long. "A Binary Decision Diagram (BDD) Package". Long@research.att.com.

[5] Karl S. Brace. bdd - a BDD package version BDD-VERSION". Carnegie Melloon University.

[6] F. Somenzi. "CUDD : CU Decision Diagram Package Release 1.1.0". Department of Electrical and Computer Engineering. University of Colorado at Boulder. <Fabio@Colorado.EDU>.

[7] Seayang Yang. "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0". Microelectronics Center of North Carolina, P.O. Box 12889, Research Triangle Park, NC 27709. January 15, 1991.

[8] F. N. Najim. "Feedback, Correlation, and Delay Concerns in the Power Estimation of VLSI Circuits". 32nd Design Automation Conference, 06/95, San Francisco, CA, USA.

[9] Tom Shiple. "Binary Decision Diagram (BDD) Package, Version 2.3". University of California, Berkeley. 1991.