MIGRACIÓN DEL HDL PARA LA SÍNTESIS LÓGICA SOBRE DISPOSITIVOS GAL

Juan C. Herrera Lozada, Agustín Cruz Contreras, Juan Carlos González Robles
Centro de Innovación y Desarrollo Tecnológico en Cómputo (CIDETEC – IPN)
Email: {jlozada; acruz; jgrobles}@ipn.mx

RESUMEN: El presente trabajo propone la migración del diseño de circuitos digitales modelados con OPAL hacia un Lenguaje Descriptor de Hardware contemporáneo, como lo es ABEL HDL. El enfoque está dirigido hacia la arquitectura GAL, de bajo impacto, pero profesionalmente socorrida en soluciones multicompuerta.
No se pretende abordar de manera formal la sintaxis de ABEL, ni tampoco las ventajas de los dispositivos GAL sobre otros PLDs de arquitectura simple, sólo se mencionan algunos aspectos que refuerzan el objetivo de la propuesta diferenciando los lenguajes implicados.

1. INTRODUCCIÓN

Las arquitecturas actuales de *Dispositivos Lógicos Programables* (PLDs) se clasifican en dos rubros generales: Simples y Avanzadas. Entre algunas de las características que las diferencian es posible mencionar el tipo de tecnología programable (*Fusible, Antifusible, SRAM*, etc.), la cantidad de *módulos lógicos* internos en términos de compuertas y la disposición de programabilidad física (en sistema o fuera de él). De forma tradicional los dispositivos PLA, PAL y GAL son representativos de las Arquitecturas Simples, mientras que los CPLDs y FPGAs lo son de las Arquitecturas Avanzadas. La *figura 1* muestra una aproximación a ambas arquitecturas.

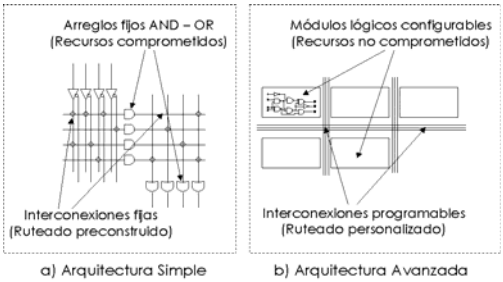

Figura1. Arquitecturas de los PLDs.

El dispositivo GAL (Arreglo Lógico Genérico) al igual que cualquier otro PLD y con la restricción en su capacidad, se utiliza para integrar una cantidad significativa de lógica combinatoria y/o secuencial en un solo circuito. Frente a los diseños basados en dispositivos monolíticos de aplicación específica los productos vertidos sobre PLDs ofrecen al diseñador una mayor flexibilidad, reducción del tiempo de desarrollo, menores posibilidades a fallos debidos a alambrado, ahorro en espacio físico y en una primera aproximación son apropiados para bajos volúmenes de producción y prototipos.

Comúnmente es posible encontrar en el mercado los dispositivos de catálogo GAL16V8, GAL20V8, GAL22V10 y GAL26V12 en diferentes versiones de empaque y velocidad. Estos en particular llegan a contener hasta unos cientos de compuertas dispuestas en arreglos AND - OR. Los costos actuales al segundo semestre del año 2003 fluctúan entre 1 y 6 dólares.

Las herramientas software para diseñar sobre PLDs varían en cuanto a sus alcances; las actuales resultan muy sofisticadas comparadas con sus predecesoras y se dirigen principalmente al modelado de soluciones sobre arquitecturas avanzadas, por lo que el diseño sobre GALs queda relegado a aplicaciones sencillas de decodificación, conteo, inicialización de interfaces y control básico.

La filosofía de diseño utilizando estas herramientas modernas es muy similar en todos los casos, unificando los métodos para la *captura del diseño*, *la simulación,* la *síntesis lógica* y la *configuración física* de los dispositivos. Todos los ambientes de desarrollo distribuidos por los fabricantes soportan, entre otros métodos de captura, el diseño a través de Lenguajes Descriptores de Hardware (*HDL – Hardware Description Language*).
Un HDL modela la operación funcional de una pieza de hardware (circuitos electrónicos y/o sistemas completos) en una forma textual. Al igual que en los lenguajes de programación comunes, se observan ciertas diferencias entre HDLs que van desde la sintaxis de codificación hasta los alcances y complejidad de los diseños descritos.

2. DISEÑO CON OPAL

El diseño con *Lógica Programable*, en específico sobre dispositivos GAL, se ha mantenido invariable desde hace 20 años en el rubro de la docencia y la investigación en varias Escuelas y Centros del Instituto. *OPAL* (Lenguaje Abierto de Arquitecturas Programables) es uno de los primeros Lenguajes Descriptores de Hardware, liberado exclusivamente para PLDs simples. Su primera versión se remonta a finales de la década de los 70's, distribuido originalmente por *National Semiconductor*. La última versión profesional del software data de 1991 funcionando sólo bajo modo MS-DOS; en Internet es posible encontrar la versión de estudiantes (*OPAL Jr.*) actualizada por una filial en 1997, que no permite la generación automática del archivo JEDEC (*translate OPAL -> JEDEC Map*, validada sólo en la versión profesional) y está limitada a unas cuantas líneas de código, pero que funciona bajo modo Windows.

Ambas versiones han sido utilizadas con resultados satisfactorios comprobados; los inconvenientes derivan en el editor de texto poco amable para el usuario, las pocas variantes en la sintaxis que dificultan la descripción de módulos complejos, además de que no forma parte de un ambiente de desarrollo completo basado en un esquema de *Proyectos*[1], como sucede con las herramientas modernas. La pantalla mostrada en la *figura 2* es el editor de OPAL en su versión profesional *1.01* de 1991.

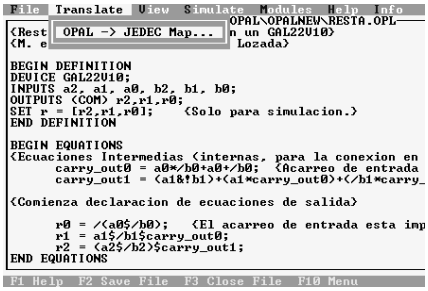

Figura 2. Editor de OPAL, versión profesional.

OPAL mantiene los tres tipos de descripción clásicos en un HDL: *Ecuaciones, Tablas de Verdad* y *Diagramas de Estado*. Los HDLs contemporáneos además de soportar los mismos tipos, incluyen palabras reservadas adicionales que facilitan la codificación aumentando los alcances en los diseños y permiten configurar a la vez arquitecturas de PLDs simples y avanzadas en entornos de desarrollo más sofisticados. Los HDLs más utilizados en la actualidad, tanto profesionalmente como en el ámbito estudiantil son tres: VHDL, Verilog y ABEL. Estos representan estándares respetados e incluidos en la mayoría de las herramientas de propietario.

3. DISEÑO CON ABEL - HDL

Durante varios años, los fabricantes de PLDs encontraron cierto beneficio en las arquitecturas simples, por lo que sus esfuerzos estaban dirigidos paralelamente a éstas y a las avanzadas. En la actualidad, varios de estos fabricantes abandonaron el mercado de los PLDs simples avocándose al diseño exclusivo sobre CPLDs y FPGAs, tal y como sucedió con *Xilinx* entre otros.

Lattice Semiconductor es hoy en día el mayor fabricante de dispositivos GAL, por lo que en sus herramientas incluye las necesarias para el diseño. *ispEXPERT*[2] es un ambiente de desarrollo completo para lógica sobre PLDs genéricos y dispositivos mayores de propietario, que acepta la captura a través de HDLs (ABEL, Verilog y VHDL) y diagramas esquemáticos. El software puede ser descargado en una versión de prueba, o adquirir una completa con licencia, directamente del sitio WEB de Lattice (*http://www.latticesemi.com*).

La *figura 3* exhibe la ventana principal del administrador del Proyecto, nombrado por Lattice como *Project Navigator*. Cuando se genera un nuevo *Proyecto* se indica hacia que tipo de PLD estará dirigido, lo que conlleva a que dentro del código HDL no es necesario declarar éste, tal y como se aprecia en la misma ventana.

[1] *Los ambientes de desarrollo completos trabajan basados en Proyectos. Es necesario generar un proyecto para hilvanar diversos módulos HDL y diagramas esquemáticos. Este tipo de soluciones está encaminado al diseño de sistemas complejos, sintetizables preferentemente sobre dispositivos de arquitectura avanzada.*

[2] *Lattice ha liberado una versión actualizada del software a la que denominó ispLEVER.*

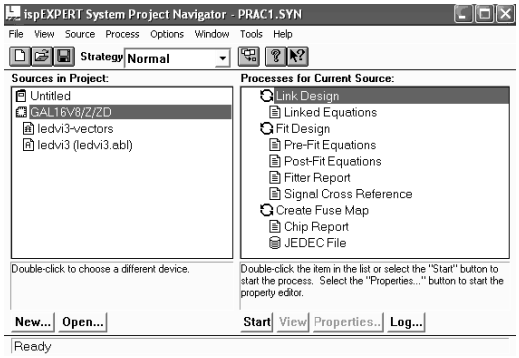

Figura 3. ispEXPERT de Lattice.

ABEL (*Lenguaje Avanzado de Ecuaciones Booleanas*) fue desarrollado por *Data I/O Corporation* en 1997, siendo capaz de modelar circuitos simples y complejos a través de una sintaxis más natural que la de OPAL, admitiendo *abstracciones comportamentales* (nivel registro) y *estructurales* (nivel compuerta) más intuitivas, propiciando una mayor flexibilidad para modularizar un diseño y posteriormente depurarlo sin afectar drásticamente el código original. En la *figura 4* se aprecia el editor para HDLs incluido en *ispEXPERT*.

Figura 4. Editor HDL de ispEXPERT.

4. MIGRACIÓN

Para fundamentar la propuesta de migración se realiza un análisis particular entre tres circuitos digitales clásicos modelados con los dos HDLs en cuestión: un *comparador*, una *unidad aritmética simple* y un *contador ascendente – descendente*.

a. Diseño de un comparador binario

ABEL incorpora la instrucción *When Then* para facilitar la lógica de multiplexaje en las descripciones comportamentales. Los códigos siguientes implementan un circuito comparador de dos números binarios; la versión OPAL procesa números de 3 bits y la correspondiente a ABEL, números de 4 bits.

Obsérvese la similitud, lo que de primera instancia implica que para este tipo de circuitos adaptarse al diseño con ABEL no requiere cambios drásticos si se conoce la sintaxis de OPAL.

Código OPAL	Código ABEL

BEGIN HEADER
Comparador de 2 números de 3 bits, en OPAL
END HEADER
 	
BEGIN DEFINITION
device GAL16V8;
input a2=2,a1=3,a0=4,b2=5,b1=6,b0=7;
output (com) may=15,men=16,eq=17;
set seta=[a2,a1,a0],setb=[b2,b1,b0];
END DEFINITION
 	
BEGIN EQUATIONS
eq = (seta == setb);
may = (seta > setb);
	MODULE compara;
	TITLE ' Comparador en ABEL';
	a2..a0 pin 2..4;
	b2..b0 pin 5..7;
	may, men, eq pin 15..17 istype 'com';
	a = [a2..a0];
	b = [b2..b0];
	
	EQUATIONS
	when (a==b) then {may=0; eq= 1; men=0}
	when (a<=b) then {may=0; eq= 0; men=1}
	when (a>b) then {may=1; eq= 0; men=0}
	
	TEST_VECTORS
	([a, b] -> [may, eq, men]);

```
 men = ( seta <  setb );
END EQUATIONS

BEGIN VECTOR
a2,a1,a0,b2,b1,b0 ;
000000
000100
101110
111100
END VECTOR
```

```
[0, 0] -> [.x., .x., .x.];
[0, 4] -> [.x., .x., .x.];
[5, 6] -> [.x., .x., .x.];
[7, 4] -> [.x., .x., .x.];
END
```

El esquema de la descripción textual se simplifica declarando agrupaciones comunes de bits (*sets*). Nótese que los dos listados anteriores incluyen sus respectivos vectores de simulación, que por cuestiones de espacio se omiten de los códigos subsecuentes.

b. Diseño de una unidad aritmética simple

Para modelar circuitos aritméticos en OPAL es necesario describir el módulo *estructuralmente*, es decir, utilizando sólo ecuaciones que infieren operadores lógicos. Para una adición de dos números de tres bits cada uno, resultaría complicado escribir una tabla de verdad de 64 combinaciones, por lo que la solución estiba en modelar sumadores en cascada a través de ecuaciones. ABEL puede modelar este mismo diseño con una descripción estructural, similar a la solución de OPAL, o bien, con una descripción comportamental que se limita a utilizar el operador suma (+) entre dos sets binarios.

En los códigos siguientes, la variable *op* permite adicionar o sustraer los números binarios (de 3 bits para OPAL y de 4 bits para ABEL). La solución estructural en OPAL requiere ecuaciones intermedias (nodos) para conectar los bloques sumadores; la versión comportamental de ABEL necesita obligatoriamente que los sets que intervienen en la operación sean del mismo tamaño. La directiva *@carry* limita el acarreo *Look ahead* (resultado premeditado, resolviendo de forma paralela) en las operaciones aritméticas de ABEL.

Código OPAL

```
BEGIN HEADER
Sumador- Restador 3 bits
END HEADER

BEGIN DEFINITION
device GAL22V10;
inputs a2, a1, a0, b2, b1, b0, op;
outputs (com) s0, s1, s2, s3;
END DEFINITION

BEGIN EQUATIONS
xor0 = b0$op;
xor1 = b1$op;
xor2 = b2$op;
cin1 = a0*(xor0)+a0*op+(xor0)*op;
cin2 = a1*(xor1)+a1*op+(xor1)*op;
cin3 = a2*(xor2)+a2*op+(xor2)*op;
s0=a0$(b0$op)$op;
s1=a1$(b1$op)$cin1;
s2=a2$(b2$op)$cin2;
s3=cin3*!op;
END EQUATIONS
```

Código ABEL

```
MODULE sum_res
TITLE 'Sumador- Restador 4 bits';

A3..A0, B3..B0, op pin;
R4..R0 pin istype 'com';

A4=0;
B4=0;

a=[A4, A3, A2, A1, A0];
b=[B4, B3, B2, B1, B0];
r=[R4, R3, R2, R1, R0];

EQUATIONS
@carry 1;
when op==0 then r=a+b;
else r=a-b;
END
```

ABEL también soporta *complemento a dos* para representar cantidades negativas, pero no tiene la capacidad de implementar multiplicaciones como sucede con HDLs más poderosos como Verilog y VHDL, que de alguna manera están sobrados para el diseño sobre dispositivos GAL.

Se puede apreciar con claridad que el código de OPAL no se puede extender fácilmente para números de más bits, mientras que la de ABEL sí. Para los 4 bits de cada número en la solución de ABEL, se requeriría una tabla de verdad de 2^9 combinaciones.

C. DISEÑO DE UN CONTADOR

El diseño secuencial, en específico para contadores binarios, toma dos rumbos posibles si se modelan con OPAL: una *tabla de transiciones* (soportada sólo por la versión profesional) o un *diagrama de estados*. ABEL, por su parte, soporta los anteriores más otros con descripción comportamental basada en estructuras *When Then* que simplifican considerablemente el código. Los listados siguientes modelan un contador binario ascendente – descendente, conmutable a través de la variable *up*.

El código escrito para OPAL implementa un contador de tres bits con un diagrama de estados, incorporando un *reset* y un *output enable*. El propio de ABEL implementa un contador de 4 bits, con carga paralela (*load*), detención de conteo (*hold*) y *reset*. Las entradas *din3, din2, din1* y *din0* son las líneas de datos que impondrán un valor para iniciar un conteo cuando se activa *load*. Nótese que ABEL, al igual que OPAL, acepta *directivas punto* para declarar una *sincronía con un reloj*, un *reset* y demás configuraciones que puede adaptar la salida secuencial.

Código OPAL	Código ABEL

BEGIN HEADER
Contador UP/DOWN de 3 bits
END HEADER
 	
BEGIN DEFINITION
device GAL16V8;
inputs reloj, up, reset, ce;
STATEBITS sb2,sb1,sb0;
STATE_NAMES cero=0,uno=1,dos=2,tres=3,cuatro=4,
cinco=5,seis=6,siete=7;
set cuenta=[sb2,sb1,sb0];
END DEFINITION
 	
BEGIN EQUATION
cuenta.C = reloj;
cuenta.OE = ce;
END EQUATION
 	
BEGIN STATE_DIAGRAM
STATE ALL:
IF reset THEN cero;
STATE cero:
IF up THEN uno
ELSE siete;
STATE uno:
IF up THEN dos
ELSE cero;
STATE dos:
IF up THEN tres
ELSE uno;
STATE tres:
IF up THEN cuatro
ELSE dos;
STATE cuatro:
IF up THEN cinco
ELSE tres;
STATE cinco:
IF up THEN seis
ELSE cuatro;
STATE seis:
IF up THEN siete
ELSE cinco;
STATE siete:
IF up THEN cero
ELSE seis;
END STATE_DIAGRAM
	MODULE contador
	TITLE 'Contador UP/DOWN de 4 bits'
	
	reloj, reset, hold, load, up pin;
	din3..din0 pin;
	cuenta3..cuenta0 pin istype 'reg,dc';
	din = [din3..din0];
	cuenta = [cuenta3..cuenta0];
	
	EQUATIONS
	@carry 2;
	cuenta.CLK = reloj;
	when load then cuenta := din;
	  else when !hold then cuenta:= cuenta;
	  else when up then cuenta:= cuenta + 1;
	  else cuenta:= cuenta - 1;
	END

5. CONCLUSIONES

La descripción con ABEL no sólo resulta más sencilla que con OPAL, sino que aporta mayor versatilidad en los diseños propiciando aumentar los alcances de los mismos. Con ABEL, la arquitectura GAL se mantiene como un fuerte contendiente en el diseño de circuitos digitales sencillos que podrían llegar a cuestionar el uso de algún microcontrolador para resolver situaciones de bajo impacto, con la gran ventaja de la reprogramabilidad y la reutilización de código para configurar dispositivos de mayor capacidad como CPLDs y FPGAs.

El software utilizado para complementar esta propuesta tiene una distribución aceptable por parte del fabricante, y por sí mismo representa una herramienta accesible para diseñadores principiantes y expertos. A través de Internet es posible obtener tutoriales del lenguaje y ejemplos detallados.

6. BIBLIOGRAFÍA

[1] Sebastian Michael J., Application - Specific Integrated Circuits, Addison Wesley, 2000.
[2] ispEXPERT, ABEL HDL Reference Manual, Lattice Semiconductor in line, 2002.
[3] John Wakerly, Digital Design, Prentice Hall, 2002.