Una Experiencia con Lava en el Taller de Organización del Computador

Lic. Nicolás Wolovick
nicolasw@hal.famaf.unc.edu.ar
Fa.M.A.F. - U.N.C.
Argentina

Resumen

El estudio del hardware de una computadora y cómo este trabaja debe ser parte de la formación básica en Ciencias de la Computación. En nuestra carrera la primera materia en esta temática es Organización del Computador, la cual carecía de prácticos de taller que permitieran integrar y profundizar conceptos. Siguiendo un trabajo de O'Donnell se implementó un práctico de taller para describir, simular y verificar circuitos lógicos utilizando un *HDL* funcional. El enfoque resultaba compatible con los objetivos pedagógicos y las restricciones presupuestarias existentes. Los resultados fueron positivos y los objetivos por demás cumplidos.

Palabras Clave HDL, organización del computador, educación, lenguajes funcionales, Lava, hardware, descripción, verificación.

1 Introducción

En el segundo año de la currícula de nuestra Licenciatura en Ciencias de la Computación está incluída la materia "Organización del Computador". Los objetivos de la asignatura fueron, desde que se creó la carrera en 1993, que el alumno comprenda el funcionamiento de una computadora sencilla, desde el nivel de compuertas al de lenguaje ensamblador. En la primera parte de la materia se dan temas generales como representaciones numéricas, circuitos lógicos combinacionales y secuenciales, bloques constructores, aritmética binaria y memoria; para que en la segunda parte, se pueda presentar una arquitectura de conjunto de instrucciones (*ISA*) y la microarquitectura que la implementa.

El esquema de la asignatura consistía en el dictado de clases teóricas donde se introducían los temas, y la resolución de guías de ejercicios prácticos relacionadas a los teóricos. En general los alumnos que pasaban por la materia no recibían ningún tipo de estímulo respecto a la temática, que indudablemente resulta importante en la formación de cualquier universitario dedicado a las ciencias de la computación.

Estas falencias fueron causadas, por un lado, por problemas presupuestarios, que provocaban la asignación de docentes de áreas afines a la materia (física, electrónica), pero sin conocimientos formales de ciencias de la computación. Por otro lado, también faltaba una política académica que apuntalara por igual todos los tópicos de la currícula.

A partir de 1998, a cinco años de haberse creado, la carrera se encontraba más sólida y tenía una masa crítica de docentes como para empezar a mejorar su plan de estudio y las materias. Un cambio en el plan de estudios y la incorporación de talleres de programación, primero en Algoritmos, y luego en Paradigmas y Compiladores, ayudaron a fortalecerla.

Cuando nos hicimos cargo de los prácticos de la materia en el 2000 y detectamos esta situación, entendimos que en el 2001 debíamos generar un taller que les permitiera a los alumnos dejar un poco el papel y poner sus diseños en funcionamiento.

2 Elección de la Herramienta

Existían muchas posibilidades para plantear estos prácticos, pero la construcción y simulación de circuitos digitales chicos y medianos, se presentaba como una oportunidad concreta. Restricciones presupuestarias (sólo software libre), de equipamiento (laboratorio con pocas y viejas máquinas) y de recursos humanos (un profesor de teórico, otro de práctico y tres ayudantes alumnos para aproximadamente cien alumnos), llevaron a la decisión de utilizar alguna herramienta de software libre, preferentemente de código abierto, corriendo sobre sistemas operativos libres, que requiriera de poco hardware y que no insumiera muchas horas docentes para su enseñanza.

A finales del año pasado se les acercó a los integrantes de la cátedra, un artículo de O'Donnell [4], que trataba el tema de la enseñanza de arquitectura de computadoras en el pregrado utilizando Hydra, un lenguaje funcional de descripción de harware (*HDL*). Este artículo resultó fundamental, pues planteaba muy claramente los problemas de la enseñanza en las asignaturas relacionadas con el hardware, y mostraba como, a través de este *HDL* inmerso en el lenguaje funcional Haskell [8], se lograba solucionarlos, además de aumentar la cantidad y profundidad de los temas tratados y brindar una herramienta de simulación para los diseños.

Las condiciones estaban dadas, no se requería de inversión alguna pues todos los componentes necesarios eran libres y de código abierto (Hydra, Hugs [9] y Linux), los requerimientos de hardware mínimos, y el tiempo de enseñanza del *HDL* acotado dado que en la currícula de nuestra carrera se utiliza Haskell desde los primeros años. Además de todo esto, la ideas planteadas por Hydra eran interesantes: circuitos como funciones, funciones como mecanismo de abstracción, jerarquía de señales, y circuitos de alto orden[1].

Lamentablemente el paquete completo de Hydra no estaba disponible al momento de iniciar el cuatrimestre correspondiente de la materia, y sólo se obtuvo una pequeña parte denominado Hydra Lite [5], que resultaba más un ejemplo de un curso de programación funcional que un lenguaje de descripción de hardware.

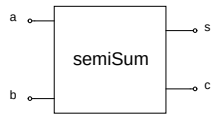
2.1 Lava

Buscando referencias a Hydra que nos permitieran llegar al paquete completo, dimos con Lava [1], un heredero del mismo Hydra y el lenguaje relacional Ruby. El proyecto estaba activo pues se había desarrollado entre 1999 y el 2000, resultaba un superconjunto de Hydra y además poseía una documentación excelente.

[1] Circuitos que a partir de circuitos generan circuitos, idea paralela a las funciones de alto orden de los lenguajes funcionales.

Lava al igual que Hydra se encuentra inmerso en Haskell, pero su diseño giró en torno al objetivo de que una única *descripción* de circuito alcanzara para obtener múltiples *interpretaciones*. Veremos esto más en detalle por medio de ejemplos que también nos mostrarán el sabor de Lava.

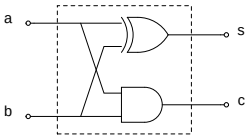
Las descripciones de circuitos combinacionales sencillos se obtienen en base a una biblioteca estándar que brinda casi todas las funciones booleanas de cero a dos entradas. Veamos cómo describir un circuito semisumador a partir de la siguiente especificación matemática y tabular.



$$a + b = c\ 2^1 + s\ 2^0$$

a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Una posible implementación descripta con un diagrama lógico y con Lava se muestra a continuación.




```
semiSum (a,b) = (s,c)
  where
        s = xor2 (a,b)
        c = and2 (a,b)
```

Lo primero que vemos es la directa relación existente entre el esquemático y la descripción Lava, también notamos que aunque el circuito tiene dos entradas y dos salidas, en Lava se estructura con una entrada y una salida de pares, denominadas señales compuestas. En general toda descripción de circuito tiene una única entrada y una única salida, donde cualquiera de las dos podrá incluir *señales simples*, *señales compuestas* y *señales compuestas genéricas*[2].

Al circuito anteriormente descripto le podemos aplicar una *interpretación de simulación*.

```
Main> simulate semiSum (high,high)
(low,high)
Main> simulateSeq semiSum domain
[(low,low),(high,low),(high,low),(low,high)]
```

Para utilizar la *interpretación de verificación*, necesitamos describir un circuito que capture la idea de una proposición, y queremos comprobar que ésta sea una tautología.

```
prop_semiSum_conmutativo (a,b) = ok
  where
        out1 = semiSum (a,b)
        out2 = semiSum (b,a)
        ok   = out1 <==> out2
```

Si `prop_semiSum_conmutativo` devuelve `high` en todo su dominio, entonces verificaremos que este circuito particular cumple con la propiedad conmutativa.

```
Main> verify prop_semiSum_conmutativo
Proving: ... Valid.
```

[2]Nombres para las señales, las uplas de señales y las listas de señales.

También podríamos haber simulado en todo el dominio con resultados equivalentes.

```
Main> simulateSeq prop_semiSum_conmutativo domain
[high,high,high,high]
```

El último de los productos que se pueden obtener de una descripción es la *interpretación de construcción*, que genera código *VHDL* para, por ejemplo, construir físicamente los diseños en una *FPGA*.

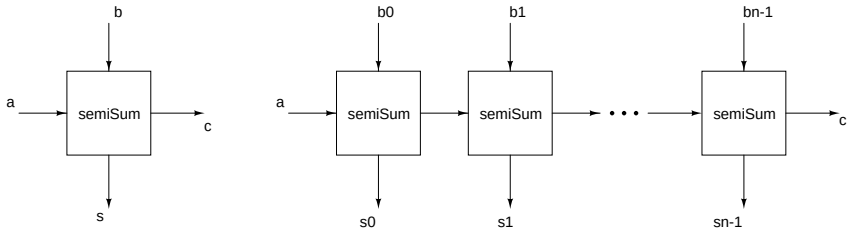
```
Main> writeVhdl "semiSum" semiSum
Writing to file "semiSum.vhd" ... Done.
```

La herencia de Ruby se hace palpable en sus circuitos de cableo y patrones de conexión[3]. Con ellos podemos obtener definiciones claras, económicas y genéricas. El semisumador "a là Ruby" sería

```
semiSum' = copy ->- (xor2 -|- and2)
```

con lo cual tenemos una definición *pointless*, con información explícita de geometría, *sharing* y paralelismo[4].

Si al semisumador lo pensamos con sus entradas al oeste y norte, y sus salidas al sur y este de la caja que lo representa, mediante el *patrón de fila* podemos obtener un sumador de bit.




```
bitSum :: (Signal Bool,[Signal Bool]) -> ([Signal Bool],Signal Bool)
bitSum = row semiSum
```

Este diseño resulta *genérico*, es decir sirve para cualquier cantidad de bits, y como los anteriores admite interpretaciones de simulación y verificación, pero ésta última restringida a comprobar propiedades para cualquier n fijo. La interpretación de construcción no está disponible en este tipo de circuitos.

Aunque Lava posee bastantes patrones de conexión en su biblioteca, pueden surgir necesidades no contempladas, en cuyo caso es posible recurrir a definiciones Haskell y describir circuitos utilizando funciones recursivas.

Finalmente mencionamos que la herramienta también permite describir e interpretar circuitos secuenciales, y es capaz de sintetizarlos a partir de lenguajes específicos, como, por ejemplo, alguno que permita describir circuitos de control.

[3] Circuitos de alto orden.

[4] En Ruby hubieramos escrito $semiSum = fork; [XOR2, OR2]$.

3 Como se Estructuró el Práctico

El principal problema a atacar era la falta de docentes, por lo que el objetivo fue que el estudio, la resolución y su entrega fueran lo más autónomo posible.

Se escribió un tutorial de Lava siguiendo los lineamientos del que se obtiene con la distribución [2], pero salvando la barrera del idioma y mostrando gran cantidad de descripciones y sus interpretaciones para que el alumno en una primera etapa sólo necesite copiar y comprobar resultados. En el tutorial se remarcaron los conceptos de la materia como especificación versus implementación, circuitos lógicos y diseños genéricos, además de presentar Lava desde el diseño de circuitos combinacionales hasta la forma de describir circuitos o patrones de circuitos mediante funciones recursivas en Haskell, pasando por verificación de circuitos combinacionales, descripción de circuitos secuenciales y patrones de conexión, entre otros temas. Las ideas de circuitos genéricos, diseños "a là Ruby" y verificación de propiedades relevantes fueron las más extensamente tratadas, pues resultaban nuevas en el marco de la materia.

A partir de este tutorial de aproximadamente veinte páginas, se dieron dos clases teóricas de una hora cada una donde se presentaba el *HDL* y se mostraba como describir, simular, verificar y construir los circuitos que se había visto en el téorico o en el práctico.

Paralelamente al dictado de éstas clases se instalaba el paquete Lava junto al probador de teoremas Vis [10] en las treinta máquinas que componen nuestro laboratorio y se creaba la página del práctico [11] con todo el material necesario: tutorial, software e instrucciones de instalación, referencias bibliográficas y la guía de trabajos prácticos.

En el práctico se plantearon nueve problemas a resolver, de los cuales cada grupo de hasta cuatro personas, debían elegir dos para ser entregados en la fecha del primer parcial a cuenta de un 30% de la nota total de éste. Los problemas consistían en describir, simular y verificar circuitos combinacionales y secuenciales, alguno de los cuales eran conocidos y otros se presentaban por primera vez. Cada sub-ítem presentaba una dificultad mayor y en casi todos los casos el último punto era opcional y consistía en obtener la generalización del circuito a n bits.

Luego del lanzamiento del práctico, se programaron dos clases de práctico en el laboratorio de manera tal que los grupos pudieran trabajar y plantear sus dudas.

Se pactó que la entrega de los resultados sería por correo electrónico y que incluiría un archivo por problema, con la documentación inmersa en el código a fin de tener documentación y código mezcladas y en un único formato. También se pidió que en la documentación se incluyeran copias de los resultados de las interpretaciones.

La corrección estuvo a cargo del jefe de trabajos prácticos y de los ayudantes alumnos. Como éstos hacían su primera experiencia en un cargo docente, sólo se les pidió informar los prácticos para luego, junto a ellos, valorarlos y calificarlos.

4 Resultados

En general, la respuesta fue buena y esto se notó desde que se comenzó a tratar el tema en las clases teóricas donde los alumnos ya mostraban interés. Se sabía que ciertos problemas iban a ocurrir, como, por ejemplo, lo que marca O'Donnell respecto a la confusión entre el *HDL* inmerso en Haskell y Haskell mismo, o la dificultad en asimilar que cualquier circuito puede ser descripto utilizando circuitos de cableo y patrones de conexión. El primer problema se atacó

directamente, controlando que cada grupo no cometa este error, mientras que en el segundo sólo se brindó apoyo a aquellos grupos que ya habían asimilado los conceptos básicos y querían hacer el punto opcional.

De los prácticos entregados, la mayoría evitaba las definiciones "a là Ruby", utilizando definiciones locales con la cláusula `where`. Los pocos grupos que utilizaron este tipo de definición plantearon dudas acerca de su conveniencia, pues no entendían por qué se preferían esas definiciones rebuscadas. Hubo que volver a explicar los conceptos, que a la luz del práctico ya resuelto, fueron entendidos.

La gran mayoría resolvió los dos primeros problemas pensando que eran los más sencillos, pero afortunadamente eran todo lo contrario y a la vez, interesantes. En el ejercicio número uno se planteó la creación de un ordenador de dos bits y a partir de éste, uno de cuatro y, opcionalmente, su generalización. Sin ninguna planificación previa, en la materia Algoritmos II se estaba tratando el tema de los algoritmos de ordenación, con lo cual se generó mayor interés y los alumnos recrearon de manera independiente alguno de los diseños clásicos estudiados en la literatura.

En general, la mayoría de los grupos trabajaron bastante y calificaron bien, sólo unos pocos no llegaron a ningún resultado o confundieron los conceptos. Varios grupos obtuvieron mejores resultados en el taller que en el exámen escrito.

5 Conclusiones

A través de Lava se lograron los objetivos planteados inicialmente, de que los alumnos se sintieran motivados probando sus diseños, y transmitir que el tópico de arquitectura y diseño de hardware es interesante en sí mismo.

Pero no sólo eso se logró, también se presentaron y reforzaron muchas de las ideas recurrentes en las Ciencias de la Computación, como abstracción, generalización, verificación automática de propiedades, simulación y especificación versus implementación. Además, los estudiantes conocieron un *HDL* para "programar" hardware y comprendieron que hoy en día prácticamente no existen diferencias en los ciclos de desarrollo del software y del hardware. Para lograr todo esto, resultó fundamental que Lava fuera un producto sólido, que propone ideas claras.

Seguramente la relación costo/beneficio lograda no se podría haber obtenido si se hubiera utilizado un *HDL* comercial –por su sintaxis complicada o la cantidad de detalles de bajo nivel que involucra–, o bien simuladores gráficos de circuitos esquemáticos –por la poca capacidad de generalizar y representar patrones de circuitos–, y esto sin contar con las restricciones no funcionales que existían como la falta de recursos de software y hardware.

Actualmente Lava no está disponible pues se está renegociando su forma de distribución con Xilnix Inc., la componente comercial de este proyecto, pero por la actitud abierta que mostraron sus desarrolladores ante nuestros requerimientos, pensamos que esto no es una dificultad importante.

Finalmente, podemos decir que los alumnos respondieron a la propuesta trabajando y generando ideas interesantes, y ésta fue la recompensa para los docentes involucrados en este proyecto.

6 Trabajos Futuros

La experiencia fue lo suficientemente buena como para plantearse la posibilidad de expandir el práctico a fin de incluir proyectos más largos que involucren más temas de la materia, como por ejemplo la construcción de un *datapath* completo o de una pequeña arquitectura. También resulta claro que se podría empezar a utilizar `Lava` en el dictado de las clases teóricas como forma de complementar y en algunos casos suplantar los diagramas esquemáticos.

Creemos que la materia debería girar en torno al diseño y construcción de una pequeña arquitectura en todos sus niveles de abstracción, poniendo a `Lava` como herramienta capaz de describirla, simularla, verificarla y finalmente construirla.

Referencias

[1] Per Bjesse and Koen Claessen and Mary Sheeran and Satnam Singh, *Lava: Hardware Design in Haskell*, International Conference on Functional Programming, 174–184, 1998

[2] Koen Claessen and Mary Sheeran, *A Tutorial on Lava: A Hardware Description and Verification System*, `http://www.cs.chalmers.se/~ koen/Lava/tutorial.ps`, August 2000

[3] John O'Donnell, *Hardware description with recursion equations*, Proceedings of the IFIP 8th International Symposium on Computer Hardware Description Languages and their Applications, North-Holland, 363–382, April 1987

[4] John O'Donnell, *From Transistors to Computer Architecture: Teaching Functional Circuit Specification in Hydra*, Functional Programming Languages in Education, 195–214, 1995

[5] John O'Donnell, *Introduction to Digital Circuit Specification with Hydra Lite*, Unversity of Glasgow Technical Report, May 1999

[6] Steven Johnson, *Synthesis of Digital Designs from Recursion Equations*, The ACM Distinguished Dissertation Series, The MIT Press, 1984

[7] Geraint Jones and Mary Sheeran, *Circuit Design in Ruby*, In Staunstrup, editor, Formal Methods for VLSI Design, Amsterdam, Elsevier Science Publications, 1990

[8] Simon Peyton Jones, John Hughes (editors) *Report on the programming Language Haskell 98*, `http://haskell.org`, February 1999

[9] M. P. Jones *The Hugs distribution.*, `http://haskell.org/hugs`, February 1999

[10] *VIS: A system for Verification and Synthesis*, The VIS Group, `http://www-cad.eecs.berkeley.edu/~ vis/`

[11] Nicolás Wolovick *Taller de Descripción de Hardware con Lava*, `http://hal.famaf.unc.edu.ar/~ odc/taller1.html`, Abril 2001