Entorno de soporte para el autoaprendizaje en el diseño de circuitos digitales

D. Bañeres, I. Bermejo, R. Clarisó, J. Jorba, M. Serra, F. Santanach, A. Rodríguez
Universitat Oberta de Catalunya
Rambla del Poblenou, 156, 08018, Barcelona
{dbaneres,rclariso,jjorba,mserravi,fsantanach,jrodriguezgarr}@uoc.edu,ivan20@gmail.com

Resumen

El diseño de circuitos digitales forma parte de las competencias básicas de los nuevos Grados en Ingeniería Informática e Ingeniería de Telecomunicaciones. Un obstáculo importante para el aprendizaje de dichas competencias es que las herramientas académicas existentes para el diseño de circuitos no permiten validar si un diseño se ajusta a la especificación de partida. En este artículo, se describe un entorno de autoaprendizaje para que los estudiantes puedan realizar ejercicios de diseño de circuitos y recibir un feedback continuo.

Summary

The design of digital circuits is a basic competence of the new Degrees in Computer Science and Engineering of Telecommunications. An important hindrance in the learning process of these skills is that the existing academic tools for the design of circuits do not allow the student to validate if his design satisfies the specification. In this article, we describe an online environment where the students can verify their designs with an automatic feedback.

Palabras clave

Circuitos, verificación, autoevaluación, educación a distancia, Model Checking

1. Motivación

En los nuevos grados del espacio europeo de Ingeniería Informática e Ingeniería de Telecomunicaciones, los alumnos desarrollan las competencias relacionadas con los principios básicos del funcionamiento del mundo digital. Una competencia básica a adquirir es saber aplicar técnicas de análisis y síntesis de circuitos digitales sencillos. Es esencial que los alumnos sean capaces de diseñar circuitos digitales antes de ampliar sus conocimientos en las posteriores asignaturas de los respectivos grados.

El profesorado de esta área tiene muchas dificultades para diseñar el modelo de aprendizaje de las

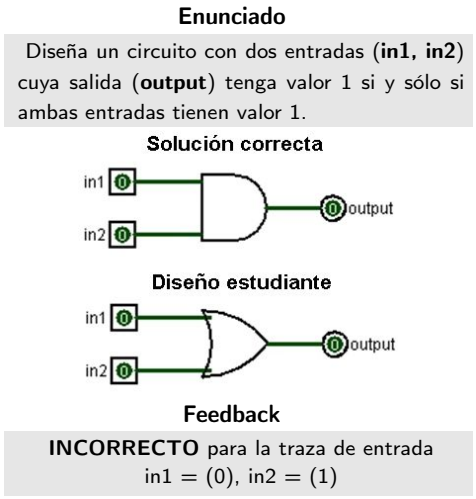

Figura 1: Ejemplo de uso de herramientas de verificación en un ejercicio de diseño de circuitos.

técnicas de diseño de circuitos. Normalmente, esta competencia está ligada a asignaturas obligatorias de primer curso (como por ejemplo *Fundamentos de Computadores*) con un alto número de estudiantes e históricos importantes sobre el nivel de abandono y de no superación de la asignatura. Un ejercicio "tipo" en este ámbito docente consiste en diseñar diversos circuitos a partir de una especificación de partida. Ésta se basa en un conjunto de entradas, de salidas y el comportamiento final deseado para el circuito en relación a estas señales. Al estudiante se le pide que describa los componentes del circuito que implementa este comportamiento. Las actividades relacionadas con el diseño de circuitos no son un proceso mecánico, sino más bien creativo y se puede obtener más de una solución válida. Por eso, es vital que los alumnos puedan desarrollar estas prácticas recibiendo un feedback constante que les permita evaluar el progreso de su aprendizaje. Habitualmente, para el profesorado es difícil dar un feedback inmediato, constante e individual a todos los estudiantes de una aula numerosa. Además

el problema es significativamente mayor cuando el sistema de aprendizaje es a distancia.

Esta inquietud surge también en la docencia de otros ámbitos temáticos dentro de la Informática. El trabajo en este campo ha dado lugar al desarrollo de diferentes herramientas para la autoevaluación de ejercicios, como por ejemplo LEARN-SQL [1] (bases de datos), ACME [11](matemáticas,estadística, bases de datos y otros),Jutge.org(programación)[10] o Pl-Man (lógica) [7], entre otros. La mayoría de estas herramientas se basan en la comprovación del resultado de *juegos de prueba*. Sin embargo, estas herramientas no resultan apropiadas para la autoevaluación de circuitos lógicos: el número de entradas posibles es muy elevado y no resulta factible definir el resultado esperado para cada una de ellas. Tampoco es posible identificar un subconjunto de casos "representativos".

Como ejemplo, consideremos un ejercicio de diseño de circuitos como el que se muestra en la Figura 1. El enunciado pide el diseño de una puerta AND. En la parte superior de la figura se muestra la solución correcta al ejercicio. Imaginemos que un estudiante propone la solución errónea que se muestra en la parte inferior de la figura, utilizando una puerta OR. Proponemos una herramienta que permita al estudiante comprobar por sí mismo si su solución es correcta y que, en caso de error, le proporcione como resultado un conjunto de valores para las entradas que ejemplifique el fallo y permita depurar el diseño. En este caso, la herramienta indicaría el error dando como ejemplo los valores de entrada $in1 = 1$ y $in2 = 0$, en los que el valor de la salida obtenida (1) difiere de la esperada en el enunciado (0). Este ejemplo es trivial y sólo se utiliza para ilustrar el proceso de resolución. Los ejercicios reales de diseño de circuitos tienen enunciados más complejos, que pueden llegar a tener decenas de puertas lógicas y otros elementos como registros y multiplexores. En este escenario, hay múltiples soluciones posibles y no es trivial identificar posibles errores, por lo que el feedback obtenido mediante una herramienta de autocorrección puede ser muy valioso.

Actualmente existen varias herramientas académicas mediante las cuales los alumnos pueden diseñar gráficamente circuitos lógicos [**?**, **?**, **?**]. Estas herramientas son muy intuitivas de utilizar pero tienen un inconveniente: "no se puede determinar si un diseño es válido a partir de una especificación dada". Para realizar esta verificación existen herramientas industriales de diseño y verificación [14].No obstante, su utilización presenta varios inconvenientes: su uso en asignaturas introductorias no es adecuado debido a su complejidad y alto coste económico. También existe otro tipo de herramientas que muestran de forma automática el diseño de circuitos a partir de un mapa de Karnaugh o grafo de estados [6] pero no permiten verificar si el diseño proporcionado por un alumno es correcto a partir de la especificación.

Precisamente para aportar solución a esta problemática, se propone un entorno virtual de autoaprendizaje que permita a los estudiantes una autoevaluación constante de sus diseños de circuitos.

El entorno tiene las siguientes características:

- El sistema de verificación es similar al utilizado en la industria de fabricación de microchips [14], aunque a menor escala. Se ha desarrollado una herramienta académica de verificación basada en analizar los circuitos mediante técnicas de Model Checking [8].
- La herramienta verifica si dos circuitos son *funcionalmente* iguales, es decir, se comprueba si tienen el mismo *comportamiento* (proporcionando un contraejemplo en caso contrario).
- El alumno utiliza un software de diseño y simulación de circuitos [2] que permite dibujar el circuito. Internamente, el proceso de verificación compara este diseño con la solución oficial mediante Model Checking.

Después de presentar los objetivos docentes (Sección 2) y la arquitectura del entorno (Sección 3), este artículo describe la experiencia de su aplicación en una asignatura obligatoria con el respectivo análisis de los resultados (Sección 4).

2. Objetivos docentes

El principal objetivo es proporcionar un entorno virtual para la autocorrección de ejercicios de diseño de circuitos con disponibilidad 24x7 (las 24 horas, los 7 días de la semana). Mediante este entorno, los estudiantes pueden comprobar de forma automática si su diseño es correcto. Además de facilitar el proceso de autoevaluación, la plataforma permite tener un control sobre la evolución de los alumnos en todo momento registrando su utilización y los patrones de uso. En universidades virtuales, como es el caso de la Universitat Oberta de Catalunya (UOC) [13], este

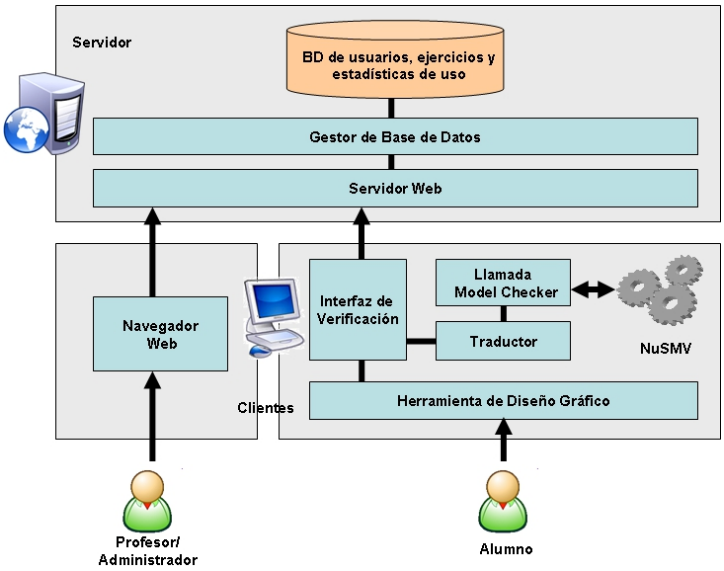

Figura 2: Arquitectura del entorno de verificación.

entorno permite incidir en el autoaprendizaje de los alumnos en competencias complejas.

En las asignaturas de primer curso donde se desarrolla la competencia de diseño de circuito digitales (en nuestro caso, *Fundamentos de Computadores*), los estudiantes utilizan una herramienta de diseño gráfico de circuitos [2]. Esta herramienta les permite diseñar gráficamente los circuitos sin necesidad de tener conocimientos de lenguajes de descripción de hardware [9,15]. Además, les permite simular de forma manual su diseño para valores concretos en las entradas. En este proceso de aprendizaje creemos fundamental que los estudiantes puedan recibir un feedback inmediato en el momento de resolver un ejercicio de diseño. El diseño de circuitos no es una actividad repetitiva, sino que requiere un alto grado de creatividad que debe adquirirse con la práctica. El feedback permite identificar los errores para corregirlos.

Este entorno permite el autoaprendizaje y autoevaluación de los estudiantes de una forma planificada. También se puede controlar en todo momento los ejercicios disponibles. De esta forma se puede planificar el conjunto de ejercicios que los alumnos tienen disponibles, dependiendo del material docente asignado para cada una de las semanas. Mediante este control de la información disponible y junto con los datos estadísticos, se puede conocer en todo momento la evolución individual de los alumnos y de la evolución colectiva de las aulas. Esta información permite al profesorado detectar las principales carencias de los estudiantes en los aspectos de diseño para poder incidir en los puntos débiles que se observen.

Creemos que este entorno puede potenciar la dedicación del estudiante al aprendizaje de las competencias del ámbito de tecnologías de computadores. Para potenciar y motivar aún más la introducción de esta plataforma en el aula, se propone asignar un peso en la evaluación de la asignatura a actividades prácticas ligadas a la utilización de esta plataforma.

En concreto, se pretende que este entorno facilite a los estudiantes el aprendizaje de la asignatura y que esto se vea reflejado en los resultados de la evaluación. En particular, se ha considerado tres métricas a evaluar: la *tasa de abandono* de la asignatura, la *tasa de aprobados* y la *nota media*.

3. Descripción de la plataforma

En esta sección, se describe el entorno de soporte al autoaprendizaje desarrollado. En primer lugar, se presenta la arquitectura de la plataforma. En segundo lugar, se explicita cómo se realiza el proceso de verificación.

3.1. Arquitectura

La plataforma utiliza un modelo cliente-servidor tal como se muestra en la Figura 2. En el servidor, el sistema está formado por una base de datos y un servidor Web y de aplicaciones (Apache con Tomcat) que hace de interfaz para comunicarse con los clientes. Internamente, la base de datos almacena la información sobre los usuarios y las colecciones de ejercicios existentes. Además, el sistema guarda los diseños enviados por los estudiantes como posible solución, para permitir al docente examinar el progreso de los estudiantes.

Los usuarios pueden acceder al entorno por dos vías diferentes: por navegador Web o mediante el programa de diseño gráfico Logisim [2], dependiendo de su perfil. Se han considerado dos perfiles de usuario diferentes en esta aplicación:

- Perfil de alumno: El alumno puede acceder a la plataforma mediante el programa de diseño de circuitos lógicos Logisim. El alumno puede ver en todo momento los ejercicios disponibles y su evolución en la resolución de los mismos.
- Perfil de profesor/administrador: El profesor accede a la plataforma mediante un interfaz Web. Mediante este interfaz se pueden añadir/modificar ejercicios, consultar los ejercicios resueltos por los estudiantes y ver estadísticas sobre la evolución de una aula o de un alumno en concreto. También se tienen permisos tanto para modificar los usuarios del sistema como para modificar las aulas disponibles.

Centrándonos en los estudiantes, el programa Logisim que utilizan ha sido adaptado con la introducción de un nuevo módulo (*VERILUOC*) que se puede ver en la Figura 5-(a). Este nuevo interfaz permite visualizar los ejercicios disponibles y permite verificar el circuito diseñado en ese momento mediante Logisim. En las siguientes secciones, describiremos el proceso de verificación junto con un ejemplo.

3.2. Proceso de verificación

Como se puede observar en la Figura 2, el proceso de verificación se produce en el ordenador del estudiante. Esta decisión se justifica para garantizar la escalabilidad de la plataforma: el proceso de verificación de un circuito requiere un cálculo intensivo, por lo que realizar este proceso en el servidor podría dificultar el acceso de los estudiantes al entorno durante los picos de utilización (como las fechas de entrega de las prácticas). Realizando la verificación en el cliente se evitan posibles sobrecargas del servidor y se garantiza la prestación de servicio. Es necesario observar que se han implementado medidas especiales de seguridad para que la solución propuesta por el equipo docente no sea visible para el estudiante ni sea almacenada en su ordenador.

Para el proceso de verificación, se estudiaron dos posibles implementaciones alternativas. La primera era utilizar técnicas de simulación y la segunda utilizar técnicas basadas en Model Checking. En la primera opción se debería utilizar algún intérprete de lenguaje de descripción de Hardware[1] para simular la ejecución de los circuitos. En este caso, se construye un test que simula para los dos circuitos todos los posibles valores en las entradas. Para cada valor se comprueba si el valor de cada una de las salidas coincide. Este tipo de simulación es bastante rápido para circuitos pequeños, pero aumenta exponencialmente el tiempo y la complejidad para circuitos con muchas entradas. Además, el problema se agudiza en el momento de verificar circuitos secuenciales.

La segunda técnica, que hemos seleccionado para esta plataforma, se fundamenta en Model Checking. La ventaja respecto a la anterior, es su eficiencia en tiempo en el proceso de verificación tanto en circuitos combinacionales como secuenciales. La técnica de Model Checking consiste en comprobar si un modelo cumple una condición dada. En nuestro caso (véase Figura 3), se traduce en construir a partir de los dos circuitos que se quiere comparar un nuevo circuito con las mismas entradas y con una única salida con la función $out_v = \bigwedge_{i=1}^{n} out1_i \bar{\oplus} out2_i$ donde $out1_i$ y $out2_i$ son las salidas i del circuito 1 y circuito 2 respectivamente, n el número de salidas del circuito y $\bar{\oplus}$ la operación XNOR. Este circuito deberá cumplir que la salida out_v siempre obtenga el valor 1. Model Checking comprobará de forma eficiente si esta condición se cumple para todos los posibles valores de entrada y, en el caso de circuitos secuenciales, para todos los posibles estados.

El proceso de verificación se puede observar en el diagrama de flujo de la Figura 4. Inicialmente, la solución proporcionada por el equipo docente está en formato Verilog. Actualmente, Logisim guarda los

[1] En el caso de utilizar como lenguaje de descripción Verilog, se puede utilizar algún software libre como Icarius [16].

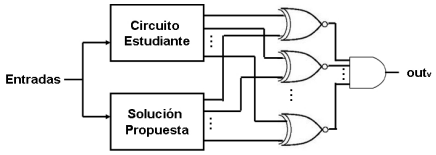

Figura 3: Modelo de verificación con Model Checking.

circuitos en un formato propio no compatible con ningún lenguaje de descripción de Hardware (formato CIRC). Con el propósito de desarrollar un verificador lo más independiente posible, existe el parser *CIRCtoVER* que traduce el formato de descripción de Logisim a Verilog (en la Figura 2 se define como *traductor*). En el momento en que los dos circuitos a verificar ya están en formato Verilog se puede iniciar el módulo de verificación *DiCiCheck* (**Di**gital **Ci**rcuit **Check**ing) que en la Figura 2 se define como *Llamada Model Checker*. Inicialmente, este módulo construye el modelo explicado anteriormente utilizando el módulo VERtoSMV. Un model checker[2] verificará si la condición propuesta se cumple sobre el modelo. Posteriormente, el sistema analiza la salida del model checker que en caso de no ser funcionalmente equivalentes ya devuelve una traza indicando para que valores de las entradas se ha detectado el error. Esta traza se indica al estudiante en la pestaña de *Verificación* en el interfaz *VERILUOC*. Nótese que en caso de ser un circuito secuencial devuelve el conjunto de estados que ha verificado correctamente hasta encontrar el error. Mediante esta información, el alumno recibe el feedback y puede comprobar manualmente en Logisim los valores de la traza problemática y corregir su diseño.

3.3. Ejemplo de verificación

En la Figura 5 se presenta, a modo de ejemplo, la verificación del circuito mostrado en la Sección de Motivación. El objetivo es diseñar un circuito que produzca el producto lógico de dos entradas. Las entradas se identifican por $in1$ y $in2$ y la salida por $output$. En la Figura 5-(a) podemos ver el interfaz VERILUOC con los posibles circuitos que se pueden verificar en la tabla superior. La tabla resume el número de intentos totales, fallidos y correctos realizados para cada ejercicio. La parte inferior se divide en dos pestañas: en la primera aparece una descripción textual del ejercicio, mientras que en la segunda

[2]Nuestro verificador utiliza el model checker NuSMV [4].

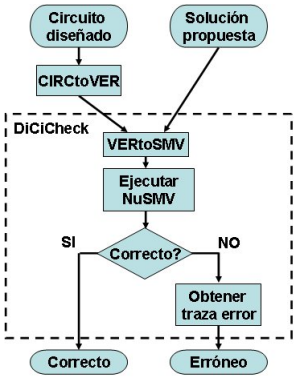

Figura 4: Proceso de verificación.

aparece el resultado de la verificación. Al seleccionar el circuito *EjemploAND2*, en la pestaña *Descripción* se muestra el enunciado del problema.

A partir de este enunciado, se diseña una posible solución (Véase Figura 5-(b)). En este caso, se ha implementado el circuito erróneamente a propósito, usando una puerta OR. Una vez terminado el diseño, se puede verificar mediante el botón *Verificar*. Antes de verificar el comportamiento del circuito, el verificador comprueba si las entradas y salidas se han definido acorde con el enunciado. En caso contrario, notifica este error al estudiante inmediatamente. En la Figura 5-(c) se puede observar el resultado en caso de error que aparece en la pestaña *Verificación*. El verificador devuelve un conjunto de valores para las entradas que producen un valor erróneo en las salidas. Con este feedback, el usuario puede simular con Logisim qué salida produce su diseño para esos valores de entrada. En este caso, el error es evidente. Si se modifica el diseño cambiando la puerta OR por una AND y se verifica de nuevo, el resultado correcto se puede observar en la Figura 5-(d).

4. Resultados

4.1. Prueba piloto

Para evaluar el impacto de la plataforma en el aprendizaje, se ha realizado una prueba piloto en una aula de la asignatura de *Fundamentos de Computadores* del Grado en Ingeniería Informática de la UOC [13].La asignatura consta de 245 alumnos, los cuales se encuentran divididos en 4 aulas. El aula escogida para realizar la prueba piloto consta de 61 estudiantes.

La evaluación de esta asignatura se realiza me-

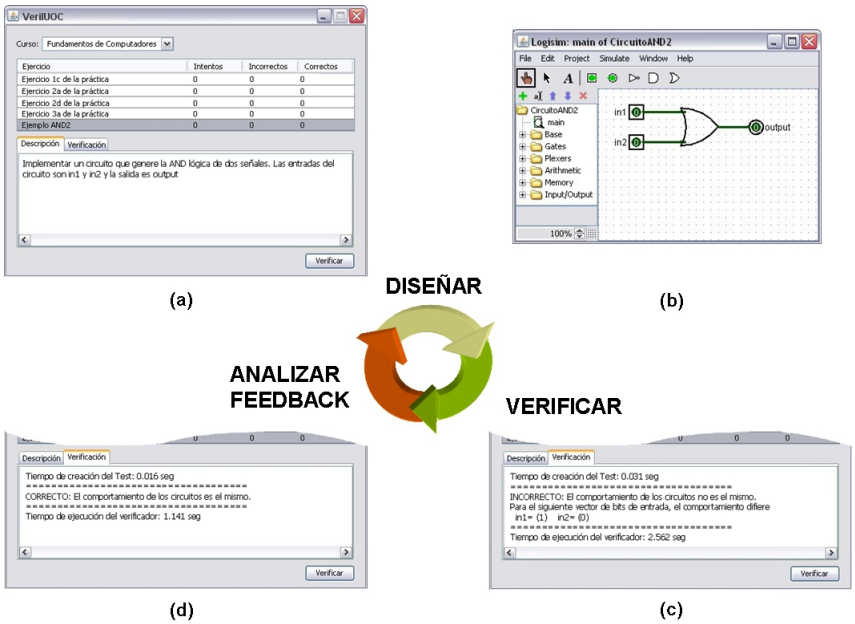

Figura 5: Ejemplo de verificación. (a) Interfaz VERILUOC de verificación. (b) Diseño erróneo de un producto lógico de dos entradas. (c) Resultado de la herramienta del diseño proporcionado. (d) Resultado de la herramienta al diseñar el circuito correcto (con una puerta lógica AND de 2 entradas).

diante una evaluación continua con 3 pruebas, una práctica y un examen final. La nota final se calcula de la siguiente forma $20\,\%EC + 40\,\%P + 40\,\%EX$, donde EC, P y EX son las notas de la evaluación continua, la práctica y el examen respectivamente. La práctica, con una duración aproximada de un mes, se realiza al final del semestre y permite relacionar todos los conceptos aprendidos en la asignatura. Concretamente, en ella se ejercitan los sistemas de numeración, circuitos combinacionales y secuenciales. Como parte de esta práctica, el estudiante debe diseñar un conjunto de circuitos que al final se combinan en un sistema complejo.

Para realizar la práctica, el estudiante recibe el enunciado, el programa Logisim modificado y un manual de instalación y uso de Logisim y VERILUOC. Mediante la plataforma VERILUOC, los estudiantes tienen la posibilidad de verificar 4 ejercicios de la práctica con un peso del 65 % (el 35 % restante está asignado a ejercicios no relacionados con diseño de circuitos). El uso del entorno en esta prueba piloto no ha sido obligatoria, pero se ha explicado al alumnado que su utilización ayuda significativamente a la superación de la práctica.

4.2. Evaluación de la prueba

Después de realizar la prueba, hemos analizado el rendimiento de los estudiantes en la práctica. Nótese que, finalmente, sólo un número reducido de estudiantes han utilizado la plataforma: 1) El total de alumnos que ha presentado la práctica han sido 31 ya que el resto no habían superado la evaluación continua (en esta asignatura hay un gran índice de abandono de estudiantes cuando no superan la evaluación continua). 2) El total que alumnos que han utilizado la herramienta han sido 14. Consideramos que una de las razones ha sido la no obligatoriedad de su uso. Por lo tanto, los resultados que presentamos no son muy significativos pero muestran unos primeros indicios del potencial de la herramienta.

En el Cuadro 1 podemos ver una comparación de las notas de prácticas de los estudiantes que han utilizado la plataforma con respecto al total de estudiantes de la asignatura. Aunque el número de alumnos que la han utilizado es reducido (50 % de los que han presentado la práctica), podemos ver que el porcentaje de estudiantes con notas de Excelente y Notable supera significativamente el global de la asignatura. La iteración en la resolución de los ejercicios a partir del feedback del entorno ayuda a obtener la

Calificación	% de estudiantes	
	VERILUOC	Global
Excelente	58 %	18 %
Notable	21 %	9 %
Aprobado	7 %	9 %
Suspenso	7 %	6 %
Muy Deficiente	0 %	5 %
No Presentado	7 %	53 %

Cuadro 1: Calificaciones en la práctica

solución correcta y obtener la máxima puntuación en los ejercicios donde podía usarse el verificador. Analizando las estadísticas de uso del entorno, hemos observado que los alumnos han utilizado un promedio de entre 3 a 6 intentos (según el ejercicio) hasta obtener la solución correcta. Este dato nos confirma que el feedback en la mayoría de los casos es positivo y ayuda a resolver algún caso particular que los estudiantes no habían tenido en cuenta.

Finalmente, también hemos analizado el resultado de superación de la asignatura (véase Cuadro 2). Teniendo en cuenta que el uso de la plataforma no era obligatorio, se ha realizado la comparación del global de la asignatura con respecto del aula donde se ha hecho la prueba piloto. Podemos observar que el porcentaje de aprobados sobre presentados se ve incrementado significativamente. También existe un descenso en el porcentaje de abandono de la asignatura. Si calculamos la nota media, podemos observar un ligero incremento en el aula de la prueba piloto. Si calculamos la nota media únicamente de los alumnos que ha utilizado la plataforma, podemos ver que la nota media es de 8,27. Este resultado final, nos confirma que la utilización de este entorno puede ayudar a adquirir las competencias de esta asignatura y así incrementar el rendimiento de la misma.

4.3. Valoración de los estudiantes

Hemos realizado una encuesta a los estudiantes para recoger su opinión. Todos concluyen que les ha ayudado en la realización de la práctica, pero han tenido diferentes dificultades en su utilización.

- Dificultades en la instalación: El programa Logisim está implementado en JAVA, por lo tanto, es multiplataforma. El inconveniente que conlleva la utilización del entorno es que se debe instalar un programa complementario para la verificación (el model checker NuSMV [4]). El manual proporcionado a los estudiantes describe paso a paso la instalación, pero los estudiantes se han encontrado con algunos problemas inesperados durante la instalación que dependen sobretodo del sistema operativo utilizado.
- Dificultades en la interpretación del feedback: En algunos casos los estudiantes no han sabido entender el error de su diseño a partir del feedback. Este problema aún es más evidente en el caso de los circuitos secuenciales, en los cuales se devuelve el conjunto de estados correctos hasta encontrar el erróneo.
- Dificultades en la utilización de Logisim: El programa es muy intuitivo de utilizar, pero hay algunos componentes que no están bien explicados en los manuales, por lo que los alumnos tienen dudas sobre su uso.

Métrica	Aula VERILUOC	Todas las Aulas
Present. / Total	49 %	47 %
Aprobados / Present.	93 %	77 %
Nota Media (Present.)	6,97	6,56

Cuadro 2: Resultados finales asignatura

Para resolver estos problemas de forma global, proponemos mejorar el manual de instalación y de uso de entorno. Otra posibilidad es crear un Wiki para almacenar toda esta información electrónicamente junto con una nueva sección de ”Preguntas más frecuentes” con los problemas más comunes.

4.4. Valoración del equipo docente

El equipo docente ha detectado algunos inconvenientes en la utilización del entorno:

- Casos de plagio: Este temor aparece en la mayoría de herramientas automáticas. El uso de las TIC en la enseñanza simplifica los métodos de copia de ejercicios.
- Mal uso del verificador: Ciertos estudiantes pueden hacer un mal uso del verificador usándolo como un mecanismo de “prueba y error” sin analizar realmente el feedback devuelto.

Para resolver estos problemas potenciales proponemos algunas soluciones. En el primer caso, se debería analizar, posteriormente a la entrega, el índice de similitud entre los diseños enviados por los estudiantes. Existen herramientas como JPlag [12]para realizar esta comprobación. Para controlar el mal

uso, proponemos limitar el número de verificaciones en un cierto periodo de tiempo.

5. Conclusiones y trabajo futuro

En este artículo hemos presentado un entorno para la verificación automática de circuitos lógicos y resultados preliminares con mejoras de los rendimientos en la asignatura relacionada.

Consideramos que hay dos aspectos destacables de esta aproximación. El primero es la integración de la plataforma con la herramienta de diseño de circuitos usada en la asignatura, de forma que el estudiante puede aprovechar la autocorrección sin tener que salir de la herramienta. El segundo aspecto es la utilización de Model Checking para la verificación de los circuitos. Esta técnica proporciona feedback al estudiante y, a diferencia de otros entornos para la corrección automática, no requiere hacer el esfuerzo de definir juegos de prueba ni se basa en la comparación directa de un circuito con otro (con lo que se permiten circuitos diferentes pero con el mismo comportamiento).

Este entorno aún tiene algunos inconvenientes en su utilización, los cuales han sido detectados y recogidos en la prueba piloto. Además, serán revisados en futuras versiones. Proponemos que en un futuro haya un repositorio de ejercicios de autoevaluación adicionales que ayuden a los alumnos a reforzar los conocimientos adquiridos.

Aunque nuestro primer objetivo ha sido utilizar el entorno en la asignatura de *Fundamentos de Computadores*, el verificador también es posible su utilización en otras asignaturas más avanzadas dentro del mismo ámbito, como *Electrónica Digital*. Una ventaja del analizador es que, al ser dependiente únicamente de Verilog, es posible utilizarlo en asignaturas donde se diseñen circuitos más complejos descritos con lenguajes de descripción de Hardware como Verilog [15], o bien VHDL [9].

Agradecimientos

Este trabajo ha sido financiado con la ayuda APLICA 2010 de la Univ. Oberta de Catalunya.

Referencias

[1] A. Abelló, M. E. Rodríguez, T. Urpí, X. Burgués, M. J. Casany, C. Martí, and C. Quer. LEARN-SQL: Automatic assessment of SQL based on IMS QTI specification. *ICALT*, pages 592–593, 2008.

[2] C. Burch. Logisim, a graphical tool for designing and simulating logic circuits. http://ozark.hendrix.edu/ burch/logisim/, 2010.

[3] CEDAR. CEDAR logic simulator. http://cedarlogic.scienceontheweb.net, 2010.

[4] Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Computer Aided Verification*, volume 2404, pages 241–268. Springer Berlin, 2002.

[5] Capilano Computing. *LogicWorks 5 Interactive Software*. Prentice Hall, 2004.

[6] J. García, J. Sanz, and B. Sotomayor. A new approach to educational software for logic analysis and design. In *International Conference on Education (IADAT e2004)*, 2004.

[7] M. J. Castel De Haro, F. Gallego, C. Pomares, P. Suau, C. J. Villagrá, and S. Cortés. e-VALUACIÓN en tiempo real. In *JENUI*, 2009.

[8] K. L. McMillan. *Symbolic Model Checking*. Kluwer, 1993.

[9] V. A. Pedroni. *Circuit Design with VHDL*. The MIT Press, 2004.

[10] J. Petit and S. Roura. Programación-1: Una asignatura orientada a la resolución de problemas. In *JENUI 2009*, 2009.

[11] F. Prados, I. Boada, J. Soler, and J. Poch. Automatic generation and correction of technical exercises. In *ICECE'05*, 2005.

[12] L. Prechelt, G. Malpohl, and M. Philippsen. Finding plagiarisms among a set of programs with JPlag. *Journal of Universal Computer Science*, 8(11):1016–1038, 2002.

[13] M. Serra, E. Santamaria, M.A. Rius, M. A. Huertas, and M. E. Rodríguez. Nuevos paradigmas de la educación: roles de acción docente en el entorno virtual de la UOC. In *CEDI'05*, 2005.

[14] Synopsys. Synopsys EDA tools. http://www.synopsys.com, 2010.

[15] F. Vahid. *Verilog for Digital Design*. Wiley, 2007.

[16] S. Williams. Icarius: a free compiler for Verilog. http://bleyer.org/icarus/, 2010.