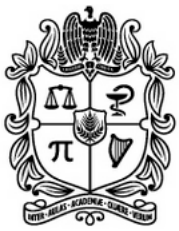


UNIVERSIDAD
NACIONAL
DE COLOMBIA

Prototipo de una Herramienta de Software de Calificación Automática con proceso de Realimentación para el Apoyo al Aprendizaje de Lenguajes de Descripción de Hardware

Andrés Francisco José Corso Pinzón

Universidad Nacional de Colombia
Facultad de Ingeniería
Departamento de Ingeniería de Sistemas e Industrial
Bogotá D.C., Colombia
2023

Prototipo de una Herramienta de Software de Calificación Automática con proceso de Realimentación para el Apoyo al Aprendizaje de Lenguajes de Descripción de Hardware

Andrés Francisco José Corso Pinzón

Informe final de Trabajo final de maestría presentado como requisito parcial para optar al título de:
Magíster en Ingeniería - Ingeniería de Sistemas y Computación

Director:
Felipe Restrepo Calle, Ph.D.
Co-Director:
Jhon Jairo Ramírez Echeverry, Ph.D.

Línea de Investigación:
Computación Aplicada - Educación en Ingeniería
Grupo de Investigación:
Programming Languages and Systems - PLaS

Universidad Nacional de Colombia
Facultad de Ingeniería
Departamento de Ingeniería de Sistemas e Industrial
Bogotá D.C., Colombia
2023

La educación ayuda a la persona a aprender a ser lo que es capaz de ser
-Hesíodo

El hardware es lo que hace a una máquina rápida; el software es lo que hace que una máquina rápida se vuelva lenta
-Craig Bruce

Agradecimientos

A mi familia por todo su apoyo, al grupo de investigación PLaS y a todas las personas que dieron su granito de arena para que esto fuera posible. A Laura por su apoyo incondicional y darme ánimo para ser mejor cada día.

Resumen

Los lenguajes de descripción de hardware se han popularizado en el diseño de electrónica digital debido al aumento de la complejidad de los circuitos actuales. Debido a lo anterior, se hace necesario apoyar el proceso de aprendizaje de electrónica digital utilizando estos lenguajes. El objetivo principal de este trabajo es desarrollar un prototipo de una herramienta de software de calificación automática con proceso de realimentación para el apoyo al aprendizaje de Lenguajes de Descripción de Hardware. Para lograrlo, este trabajo presenta la selección de características que debería incluir el prototipo, basado en otras herramientas similares. Posteriormente se realizó el diseño e implementación de un prototipo que fue usado por estudiantes del curso de Electrónica Digital de la Universidad Nacional de Colombia por cinco (5) semanas. Con este estudio se obtuvieron las percepciones de su uso. Los estudiantes encontraron útil para el proceso de aprendizaje tener una herramienta de calificación automática con realimentación, pues al estar disponible en línea da una ventaja al no depender de un profesor o tutor.

Palabras clave: HDL, calificación automática, realimentación, aprendizaje, percepciones, diseño digital, lenguajes de descripción de hardware, electrónica digital.

Abstract

Título en inglés: Prototype of an automatic qualification software tool with feedback process for hardware description language learning support

Hardware description languages have become popular in digital electronics design due to the increasing complexity of today's circuits. Therefore, it is necessary to support the learning process of digital electronics using these languages. The main objective of this work is to develop a prototype of an automatic assessment software tool with feedback process to support the learning of Hardware Description Languages. To achieve this, this work presents the selection of features that the prototype should include, based on other similar tools. Additionally, a prototype was designed and implemented for use with students of the Digital Electronics course at the National University of Colombia for 5 weeks. With this study, the perceptions of its use were obtained. It was found that students find it useful in the learning process to have an automatic grading tool with feedback, since being available online gives an advantage by not depending on a teacher or tutor.

Keywords: HDL, automatic grading, feedback, learning, perceptions, digital design, hardware description languages, digital electronics

Este Trabajo Final de maestría fue calificado en marzo de 2023 por el siguiente evaluador:

Diego Alexander Tibaduiza Burgos PhD.
Profesor Facultad de Ingeniería
Universidad Nacional de Colombia

Contenido

Lista de Figuras

Lista de Tablas

Lista de Códigos

1. Introducción

1.1. Motivación

En el mundo actual, la tecnología juega un papel fundamental. Todo el avance tecnológico que se ha tenido en las últimas décadas se debe al desarrollo y uso de semiconductores (Teepe, 2014). Los semiconductores son el componente principal de los transistores. La unión de estos componentes electrónicos, en diferentes configuraciones, es lo que ha permitido crear los circuitos digitales que están presentes en los dispositivos electrónicos que conocemos hoy en día. Impulsados por conceptos emergentes, como el Internet de las cosas (IoT) (Georgakopoulos and Jayaraman, 2016), donde todo está conectado a Internet y se realizan diferentes funciones por medio de un circuito electrónico que está en su interior, diferentes dispositivos electrónicos están más presentes en la cotidianidad. Además, grandes avances se han presentado en distintos ámbitos como en las comunicaciones, los automóviles, la medicina, el entretenimiento etc., donde a través del desarrollo de circuitos se han resuelto un sin fin de problemas (Teepe, 2014).

Los circuitos lógicos están compuestos principalmente por compuertas lógicas y biestables en diferentes configuraciones (Corsini and Rizzo, 1991). Existen dos formas para definir la configuración de los componentes mediante la descripción de hardware. La primera consiste en conectar los componentes gráficamente para obtener el comportamiento deseado y la segunda es mediante un Lenguaje de Descripción de Hardware, HDL por sus siglas en inglés (*Hardware Description Language*). En otras palabras, la estructura interna de un circuito lógico puede ser definida usando un esquemático (conexión gráfica de los componentes) o mediante la descripción del comportamiento usando un HDL. Los HDL son lenguajes especializados que sirven para describir y programar circuitos digitales. La estructura del circuito, una vez finalizada, puede ser sintetizada y proporciona un mapa de elementos lógicos que puede ser implementado en un circuito integrado o descargado en un dispositivo lógico programable como una FPGA o un CPLD para su respectivo uso. Actualmente, la industria ha acogido los HDL como estándar para el desarrollo de circuitos, debido a que cada vez los diseños son más complejos y realizar la configuración de forma gráfica se convierte en una tarea casi imposible (Mepits, 2014). Actualmente los dos HDL más usados tanto en la industria como en la enseñanza son: Verilog y VHDL (La Meres, 2019).

Los avances tecnológicos son posibles gracias a la creación de circuitos electrónicos y día a día más desarrolladores son necesarios para el diseño y verificación de estos circuitos (Georgakopoulos and Jayaraman, 2016). Para poder realizar de manera efectiva y correcta la tarea de describir hardware o

diseñar circuitos, es esencial tener claros los conceptos base. Es en las universidades donde los fundamentos del diseño digital son usualmente adquiridos. Es por ello que la IEEE Computer Society y la ACM (por sus siglas en inglés Association for Computing Machinery), proveen una guía para la creación de currículos académicos en diferentes áreas de la tecnología (ACM and IEEE-CS, 2016), en la cual se incluye el módulo de diseño digital. En este se sugiere el contenido que debe tener el curso para cubrir correctamente las bases. Adicionalmente este curso también debe incluir la enseñanza de los HDL. En particular en Colombia, uno de los programas de pregrado que está más relacionado con el desarrollo de hardware es la carrera de Ingeniería Electrónica. Una de las principales ramas de este programa es la Electrónica Digital, la cual incluye un curso que sirve de base de cursos más avanzados y donde se presentan los HDL junto con los conceptos básicos del diseño digital. En diferentes universidades en el mundo (Corsini and Rizzo, 1991), (Pereira et al., 2012), (Jutman et al., 2002), (Madanayake et al., 2012) esta asignatura se divide en una primera parte enfocada a los conceptos básicos de los circuitos lógicos, para luego sumergirse en poner en práctica dichos conceptos por medio de HDL, principalmente Verilog o VHDL. Específicamente en la Universidad Nacional de Colombia, esta asignatura tiene el nombre de Electrónica Digital I (UNAL, 2019).

De acuerdo con lo anterior, es indispensable aprender a crear circuitos electrónicos por medio de HDL debido a que se evidencia que las empresas que desarrollan hardware usualmente trabajan con estos en el ámbito profesional (Greenwood, 2013). Esto hace necesario enfocarse en la enseñanza de las bases de estos lenguajes para que los conceptos queden claros desde el inicio (Nutter et al., 2014). Sin embargo, se han identificado algunas dificultades en el aprendizaje de las bases del diseño digital. El software utilizado por la industria usualmente no está enfocado a la enseñanza (Cirka and Kaluz, 2017), por lo que se hace difícil al profesor y al estudiante utilizar estas herramientas en el proceso educativo. Por otro lado, el acceso al laboratorio, donde los estudiantes tienen tanto las herramientas como un profesor o monitor que les puedan ayudar en el desarrollo de las actividades, en muchas ocasiones es limitado y depende de la disponibilidad del mismo (Muchlas and Novianta, 2016). Adicionalmente, para obtener mejores resultados los estudiantes requieren tener una constante realimentación de los problemas resueltos. No obstante, usualmente los profesores emplean mucho tiempo en dar las observaciones y calificaciones lo que ralentiza el proceso de aprendizaje (Carroll, 2011). Por lo que la principal razón por la que la realimentación no llega a tiempo son los altos tiempos en la calificación (Baneres et al., 2014). Si los estudiantes de Electrónica Digital no cuentan con una realimentación constante, el aprendizaje de conceptos se hace más lento ya que se debe esperar a que el profesor califique y proporcione la realimentación correspondiente de cada problema.

1.2. Trabajos Relacionados

Identificados algunos de los inconvenientes para la enseñanza y el aprendizaje de las bases de la electrónica digital, varios investigadores han optado por intentar realizar esfuerzos que ayuden a la enseñanza de las bases del diseño digital. Tradicionalmente, las bases de la electrónica digital se han

enseñado por medio de la representación gráfica de las compuertas lógicas y las conexiones entre ellas. Es por ello que algunos investigadores han optado por implementar herramientas que utilizan la representación gráfica de los circuitos para enseñar a diseñar hardware. Por ejemplo, Mateev et al. (2004) crean una herramienta que permite la comparación de pequeños circuitos lógicos contra una expresión booleana, lo que permite al estudiante entender cómo transformar una función lógica en su representación gráfica. Por otro lado, además de permitir la interacción con las compuertas lógicas, Robal and Kalja (2007) incluyen en su herramienta información teórica de los diferentes componentes lógicos, así como la posibilidad de cambiar el valor de las entradas y así poder observar el comportamiento de las diferentes compuertas lógicas.

Sin embargo, como Stanisavljevic et al. (2013) mencionan, estas herramientas no permiten la calificación directa de los circuitos diseñados por los estudiantes. Por lo que la herramienta, desarrollada por estos autores en la Universidad de Belgrado, permite realizar la comparación entre la salida de la simulación del diseño del estudiante y la salida esperada del ejercicio asignado. Este mismo enfoque fue utilizado por Baneres et al. (2014). En ese caso, se extendieron las funcionalidades de la herramienta LogiSim (Burch, 2011). Lo que permitió a los investigadores asignar ejercicios a los estudiantes e informar, en caso de algún problema, los valores de las entradas con las que el circuito diseñado no funcionaría correctamente.

Por otra parte, uno de los principales problemas es que usualmente los tiempos de calificación son largos, por lo que los estudiantes pueden no recibir a tiempo la realimentación necesaria. Por lo cual, una parte de estos trabajos ha estado dirigida hacia la calificación de forma automática, donde el estudiante pueda probar sus diseños en cualquier momento y adicionalmente obtener realimentación de los errores, sin depender de los laboratorios ni del profesor directamente. Justamente, se han llevado a cabo esfuerzos adicionales para ayudar a facilitar el trabajo de los profesores. Es el caso de la herramienta desarrollada por Roy et al. (2016), en la cual adicionalmente a la calificación y retroalimentación ofrecida al estudiante, se presenta la creación de problemas dinámicamente. Esto permite al profesor tener diferentes problemas sobre un mismo tema simplemente cambiando parámetros con el fin de obtener nuevos ejercicios.

Al mismo tiempo, teniendo en cuenta la complejidad de los circuitos actuales, algunos educadores han incluido en sus herramientas el soporte para los Lenguajes de Descripción de Hardware (HDL). Por ejemplo, en el enfoque propuesto por Nutter et al. (2014), no se crea una herramienta que el estudiante pueda utilizar, sino un proceso automático de calificación tanto de circuitos diseñados gráficamente como los diseños realizados utilizando HDL. Este proceso se realiza utilizando software propietario (Cadence®) que permite la comparación lógica entre circuitos empleando una herramienta LEC (Logic Equivalence Checker). Se realizan diferentes comparaciones del circuito del estudiante contra diferentes circuitos almacenados en una base de datos. Una vez llega la fecha de entrega, se evalúa cada una de las entregas de los estudiantes. La calificación consiste en realizar la comparación con cada diseño de la base de datos. Si el diseño del estudiante coincide exactamente con alguno,

se envía por correo electrónico la realimentación correspondiente a dicho circuito. En caso que no coincida con ningún circuito de la base de datos, se realiza la calificación manual y se incluye, como un registro nuevo en la base de datos, el circuito con su respectiva realimentación para ser utilizado en el futuro.

Además de la comparación de la equivalencia lógica, la verificación formal también se hace presente en la calificación de diseños realizados utilizando HDL, la herramienta más usada para esto es llamada NuSVM (Cimatti et al., 2002). Esta herramienta usa diagramas de decisión binaria y un SAT (solucionador al problema de satisfacibilidad booleana) para crear una verificación simbólica de modelos. Los SAT son utilizados en varias herramientas para comparar el circuito del estudiante contra el circuito provisto por el profesor. Un desarrollo que incorpora esta herramienta es el presentado por Petit et al. (2018), quienes exponen su solución en un sitio web llamado `jutge.org`. A pesar que la herramienta fue implementada inicialmente como un juez virtual que apoya la enseñanza de lenguajes de programación, también fue incluido el soporte a Verilog. Sobre esta plataforma se han creado diferentes cursos que contienen varios ejercicios base para el aprendizaje de diseño digital. La realimentación de esta herramienta se enfoca en presentar el caso, llamado contraejemplo, en que el circuito del estudiante no obtuvo la misma salida que el circuito del profesor. Sin embargo, otro tipo de errores, como errores de sintaxis, no son mostrados claramente.

1.3. Identificación del Problema

La forma de evaluar el diseño de circuitos lógicos usualmente recae sobre el profesor o el monitor de la asignatura y debido a la complejidad de los ejercicios y a que cada estudiante tiene una solución diferente, el proceso de calificación se vuelve engorroso. En el método de enseñanza de circuitos lógicos de Jansen and Dusch (2014) un profesor asistente o monitor que se encarga de ayudar al momento de escribir código y es quien realiza la calificación de los trabajos. Sin embargo, esto no garantiza tener realimentación constante y a tiempo.

La situación ideal de la educación en Electrónica Digital usando Lenguajes de Descripción de Hardware sería que existiera una herramienta en línea que permitiera a los estudiantes de la asignatura diseñar, simular y probar sus diseños en cualquier momento del día. Esta herramienta debería incluir ejercicios identificados por el profesor que desarrollan las habilidades de los estudiantes. Además de esto los ejercicios realizados serían calificados automáticamente, e igualmente se daría una realimentación sin la intervención del profesor. Este instrumento permitiría al profesor enfocarse en los temas que los estudiantes evidencian más dificultad y no gastar tiempo en la corrección de ejercicios y tareas. Esta herramienta estaría acorde a las necesidades de los estudiantes, y dado que la realimentación es inmediata los estudiantes podrían aprender de sus errores inmediatamente. El auto-estudio aumenta el desempeño de los estudiantes (Stanisavljevic et al., 2013), por lo que la herramienta que se use para apoyar el aprendizaje de la electrónica digital debe tener una realimentación efectiva. El proceso de

realimentación es aquel que otorga al estudiante más herramientas para obtener mejores resultados en su proceso formativo.

De acuerdo con la revisión bibliográfica realizada, en la literatura se encuentran pocas evidencias de trabajos en los que se hayan desarrollado herramientas que tengan las características mencionadas. El mayor vacío en la literatura se evidencia en el método ideal para evaluar a los estudiantes. A continuación se enuncian algunos de los problemas encontrados:

- La mayoría de las herramientas que realizan algún tipo de calificación se enfoca en determinar si la solución a los ejercicios planteados a los estudiantes está bien o está mal, y sólo informa al estudiante esta decisión sin dar mayor información. Esto conlleva a una retroalimentación limitada.
- Debido a los tiempos de calificación, la realimentación no es inmediata.
- El software que habitualmente se usa para el diseño de circuitos lógicos en la industria y en la academia, usualmente no está enfocado a la enseñanza (Cirka and Kaluz, 2017), lo que dificulta el aprendizaje por parte de los estudiantes.

Por lo tanto, se hace necesario conocer las impresiones de los estudiantes al utilizar una herramienta, durante el proceso de aprendizaje, que sea alterna al software propietario, que permita la calificación automática de diseños digitales descritos utilizando HDL, proporcione una realimentación inmediata de los errores en el código y se pueda acceder de manera remota. Por lo que este trabajo intenta dar respuesta a la siguiente pregunta de investigación: *¿Cuáles son las percepciones de los estudiantes en cuanto al proceso de aprendizaje de HDLs al usar una herramienta de software de calificación automática de apoyo?*

1.4. Objetivos

1.4.1. Objetivo General

Desarrollar un prototipo de una herramienta de software de calificación automática con proceso de realimentación para el apoyo al aprendizaje de Lenguajes de Descripción de Hardware.

1.4.2. Objetivos Específicos

I. Identificar características que ayudan al aprendizaje de Lenguajes de Descripción de Hardware mediante una revisión de literatura de herramientas de apoyo.

II. Diseñar una herramienta de software de calificación automática con proceso de realimentación para el apoyo al aprendizaje de Lenguajes de Descripción de Hardware.

III. Implementar un prototipo de una herramienta de software de calificación automática con proceso de realimentación para el apoyo al aprendizaje de Lenguajes de Descripción de Hardware.

IV. Evaluar las percepciones de los estudiantes de un curso de electrónica digital al usar el prototipo implementado.

1.5. Alcance

Este Trabajo Final de Maestría pretende diseñar e implementar un prototipo de una herramienta de software para apoyar el proceso de aprendizaje de Lenguajes de Descripción de Hardware, teniendo en cuenta experiencias relacionadas de otros estudios. Una vez implementado se procederá a incluirlo en el proceso educativo de dos cursos de Laboratorio de Electrónica Digital I de la Universidad Nacional de Colombia, durante el primer período académico de 2020. Esta asignatura está incluida en la malla curricular de los programas de pregrado de Ingeniería Electrónica, Ingeniería Eléctrica e Ingeniería Mecatrónica. Los estudiantes lo utilizarán durante 4 semanas y finalizado este tiempo responderán una encuesta acerca del uso del prototipo. Esto con el fin de conocer sus percepciones.

1.6. Estructura del documento

El resto de este documento está organizado de la siguiente manera. En el Capítulo 2 se presenta una breve descripción de los HDL, algunas herramientas que apoyan el aprendizaje de Electrónica Digital y la identificación de las características que ayudan al aprendizaje de HDL basadas en dichas herramientas; en el Capítulo 3 se muestra el diseño del prototipo utilizando las características seleccionadas en el capítulo anterior; en el Capítulo 4 se describe la implementación de los prototipos de la herramienta de software que fueron realizados; en el Capítulo 5 se encuentra el diseño del estudio realizado; en el Capítulo 6 se presenta la experiencia de la evaluación del prototipo realizado con los estudiantes de Electrónica Digital así como los resultados y el análisis de los datos obtenidos; finalmente, en el Capítulo 7 se exponen las conclusiones de este Trabajo Final, las contribuciones y el trabajo futuro.

2. Aprendizaje de Electrónica Digital

Este capítulo presenta una introducción a los Lenguajes de Descripción de Hardware (HDL), algunos trabajos relacionados que han propuesto herramientas de apoyo al aprendizaje del curso básico de Electrónica Digital y la selección de características deseadas para incluir en la herramienta que se diseñará en este trabajo final.

La IEEE y la ACM (2016), en su guía para la creación de currículos de carreras relacionadas con la computación, manifiestan que se debe realizar la inclusión de los lenguajes de descripción de hardware cuando se está enseñando las bases del diseño digital. Adicionalmente, dado que el desarrollo de la herramienta alrededor de la cual gira este trabajo final estará enfocado en los HDL en la Sección 2.1 se realiza una corta introducción a estos lenguajes. Por otro lado, una vez expuestos de manera breve los HDL, en la Sección 2.2 se presentarán algunas herramientas que han intentado apoyar el aprendizaje del diseño digital. La Electrónica Digital va de la mano a los HDL y no es posible entender correctamente estos lenguajes sin entender las bases de los circuitos lógicos. Teniendo en cuenta estas herramientas y la experiencia del autor como Ingeniero Electrónico, se procederá en la Sección 2.3 a seleccionar las características deseadas que tendrá la herramienta de software a desarrollar.

2.1. Lenguajes de Descripción de Hardware

Actualmente el Electrónica Digital se desarrolla utilizando el diseño digital asistido por computador. Esto debido al tamaño y complejidad de los circuitos actuales. Estas herramientas para el diseño de los circuitos utilizan lenguajes de descripción de hardware. De acuerdo con el libro 'Introduction to Logic Circuits & Logic Design with VHDL' (La Meres, 2019), los dos lenguajes más populares son VHDL y Verilog HDL (más conocido simplemente como Verilog). Estos dos HDL surgieron en la década de los 80. Verilog como lenguaje de Cadence, una reconocida empresa del sector, y VHDL impulsado por el Departamento de Defensa de los Estados Unidos de América. Posteriormente, fueron liberados para uso público dada la aceptación en el gremio. Actualmente, después de varias actualizaciones, son estándar IEEE. Específicamente, el estándar para Verilog es el IEEE Std 1364 y para VHDL el IEEE Std 1076.

El objetivo de estos lenguajes es permitir sintetizar diseños escritos en texto a través de todo el flujo del diseño digital. Los HDL se utilizan para modelar diferentes niveles de abstracción como de sistema, algorítmico, RTL(Resistor-transistor logic), de compuertas y de circuito. El diseño a alto nivel

permite realizar implementaciones complejas sin tener que preocuparse por los detalles a bajo nivel. Adicionalmente, también son utilizados para construir la simulación de los diseños, con la diferencia que estas construcciones no son sintetizables, es decir, no se pueden llevar a un circuito físico. Estas simulaciones se realizan por medio de archivos comúnmente llamados *testbench* los cuales contienen los estímulos que se aplicarán a los diseños que se quieren probar (La Meres, 2019).

VHDL y Verilog están enfocados en dividir el problema en pequeñas partes e ir uniéndolas para obtener el diseño final. Cada lenguaje tiene un nombre para estas pequeñas partes. En VHDL son llamadas entidades (entity) y en Verilog se conocen como módulos (module). Usualmente, cada parte está en un archivo diferente. La extensión de estos archivos es `.vhd` para VHDL y `.v` para Verilog. Por buenas prácticas se recomienda que el nombre del archivo corresponda con el nombre de la entidad o módulo, según el lenguaje. Cada entidad o módulo contiene una declaración de los puertos de entrada/salida que posee y el comportamiento de dicha parte. En los archivos de VHDL también se deben declarar las librerías con las que se va a trabajar. En el Código **2-1** hay un ejemplo del diseño de una compuerta XOR en Verilog. Mientras, el diseño de la misma compuerta se presenta como ejemplo en el Código **2-2** esta vez en VHDL.

```
1 // Declaración del módulo
2 module compXOR(a, b, y);
3     //Declaración de puertos entradas/salidas
4     input a,b;
5     output y;
6     //Declaración del comportamiento del módulo
7     xor(y,a,b);
8 endmodule
```

Código 2-1: Ejemplo de una compuerta XOR en Verilog

Como se puede observar en los dos ejemplos presentados, los comentarios de una línea se expresan de manera diferente. Además, en Verilog toda la información está dentro del mismo bloque (module), mientras que en VHDL se tiene el bloque entidad (entity) y bloque arquitectura (architecture). En VHDL cada entidad puede tener más de una arquitectura. Como se puede observar en los ejemplos, Verilog posee una sintaxis más simple.

La sintaxis de VHDL es similar al lenguaje ADA, mientras que la sintaxis de Verilog es similar a la del lenguaje de programación C. Esta situación ha generado una aceptación mayor dentro de los programadores porque se hace más natural y más fácil de aprender. Estos HDL, al ser estándar IEEE, están respaldados por toda la comunidad, a tal punto que es posible usarlos en conjunto, desarrollando algunas partes en uno y otras en el otro, con la tranquilidad de poder integrarlos fácilmente. En los

```vhdl
-- Declaración de librerías
library IEEE;
use IEEE.std_logic_1164.all;
--Declaración de la entidad
entity compXOR is
    -- Declaración puertos entrada/salida
    port(
        a: in std_logic;
        b: in std_logic;
        y: out std_logic);
end compXOR;
--Declaración del comportamiento de la entidad
architecture rtl of compXOR is
begin
  y <= a xor b;
end rtl;
```

Código 2-2: Ejemplo de una compuerta XOR en VHDL

últimos años el escoger entre VHDL o Verilog para un diseño se da por una razón meramente de preferencia personal.

El desarrollo del prototipo de una herramienta de software que se presenta en este informe de trabajo final estará enfocado en los dos lenguajes de descripción de hardware más populares: VHDL y Verilog HDL.

2.2. Herramientas de apoyo al aprendizaje de Electrónica Digital

A continuación se presentan algunas herramientas y procesos que se utilizan en diferentes universidades en el mundo para apoyar los cursos de circuitos digitales. Esta revisión se hizo con el fin de identificar características deseadas que tendrá la herramienta de software a desarrollar en este trabajo.

Mateev et al. (2004) evidencian que el aprendizaje de los circuitos lógicos depende en gran medida de las prácticas que se realicen en el laboratorio. Dado que algunos estudiantes presentan dificultades con los conceptos básicos del álgebra de Boole, los autores desarrollaron una aplicación para complementar las prácticas de laboratorio. Esta herramienta en línea, permite a los estudiantes clari-

ficar conceptos básicos del diseño digital. Mediante ejercicios, la herramienta comprueba si el circuito realizado por el estudiante de forma gráfica corresponde con una función booleana previamente ingresada.

Robal and Kalja (2007) identifican la importancia de las herramientas en línea por lo que implementan e-EDU, con la cual no hay limitaciones de tiempo ni distancia para la enseñanza de circuitos lógicos. Esta herramienta incluye los conceptos básicos de los circuitos digitales y la posibilidad de cambiar las entradas en los componentes para ver cómo se comportan las salidas. Adicionalmente, se puede guardar información de los estudiantes así como las tareas de los mismos. Dentro de las ventajas se encuentra que la realimentación de las actividades queda guardada en el sistema y el estudiante la puede consultar en cualquier momento.

Stanisavljevic et al. (2013) identifican algunas dificultades con el software que usualmente se utiliza para la enseñanza de diseño digital. El principal problema es que no se puede realizar la calificación automática con las herramientas más usadas (HASE, JHDL, ISE WebPACK, PSIM, Logisim, Virtual Vulcan y e-EDU). Adicionalmente, no existe una que cumpla con todas las necesidades exigidas por los autores, por lo que desarrollaron su propia herramienta llamada SDLDS. Está enfocada al diseño digital conectando los componentes mediante un esquemático para obtener el comportamiento deseado. Fue desarrollada en Java y es una aplicación de escritorio, la cual cada estudiante debe tener en su equipo. Permite al estudiante realizar autoaprendizaje, actividades de laboratorio y exámenes. Realiza la simulación del circuito del estudiante, y su salida es comparada con una salida esperada, para así poder dar un veredicto acerca de la corrección de la solución.

Baneres et al. (2014) argumentan que el diseño de circuitos más que un proceso mecánico es un proceso creativo, por lo que es necesario que los estudiantes tengan práctica constante y obtengan realimentación de cada una de estas prácticas para mejorar su progreso de aprendizaje. Dado que no existe una solución única a un problema de diseño digital, es difícil para el profesor dar realimentación individual y más en cursos con un gran número de estudiantes. Es por esto que presentan la herramienta VerilUOC. Este software permite al estudiante crear sus diseños de forma gráfica y poder comparar la equivalencia con la solución del profesor. La realimentación consiste en un contra ejemplo donde el diseño del estudiante no haya cumplido con lo deseado. Adicionalmente, estos autores resaltan la importancia de la usabilidad de la herramienta, ya que los estudiantes están más acostumbrados a interfaces amigables con el usuario. La herramienta está dividida en dos aplicaciones: la primera parte es una aplicación de escritorio desarrollada en Java integrada con LogiSim (Burch, 2011), donde se pueden diseñar los circuitos que dan solución a los ejercicios planteados y obtener la realimentación correspondiente; la segunda aplicación consiste en un desarrollo web donde el profesor puede hacer seguimiento al avance de los estudiantes y añadir o actualizar nuevos ejercicios.

Nutter et al. (2014) presentan un proceso automatizado de calificación de diseños digitales en la Universidad de Manchester. El cual fue utilizado en un curso de primer año de Ciencias de la Compu-

tación. Tradicionalmente en esta universidad, cuando el estudiante resuelve un ejercicio tiene que entregar en papel, una hoja de respuestas, además de los esquemáticos, archivos de prueba y diagrama de tiempos. La hoja de respuestas incluye un campo donde un profesor asistente debe firmar para comprobar que el diseño está funcionando en el laboratorio. Posteriormente, esta información es analizada y se da una realimentación escrita a mano junto con la calificación final. Este enfoque presenta algunos inconvenientes. Por ejemplo, en la Universidad de Manchester este curso lo toman cerca de 150 estudiantes por año, por lo que los tiempos de calificación son extensos y la realimentación no llega en el momento oportuno. Por otro lado, la calificación es realizada por diferentes asistentes docentes lo cual puede generar inconsistencias en las calificaciones y realimentaciones diferentes según el evaluador. La entrega en físico hace que el estudiante tenga que imprimir todos los documentos necesarios. Por lo que para contrarrestar estos desafíos, los autores proponen un proceso automatizado para la calificación de estos ejercicios. Se inicia entregando a los estudiantes los archivos con el cascarón inicial del proyecto, donde se especifican los puertos a usar y las cabeceras de los módulos que se usan en los testbench. Una vez el estudiante finaliza la solución, envía electrónicamente los archivos. Una vez se cumple la fecha de entrega, se ejecuta el proceso automático de calificación utilizando software de Cadence (propietario) y se da una realimentación personalizada a los estudiantes vía correo electrónico. Finalmente, se realiza una sesión presencial con el asistente docente. El proceso automático de calificación se realiza de dos maneras diferentes. La primera utiliza la simulación de los diseños. La salida de esta simulación es comparada con una tabla de verdad que contiene los valores esperados. Si alguna salida no corresponde a la deseada, la realimentación incluye el caso con falla, el resultado observado y el resultado esperado. Al final de la simulación se envía un correo al estudiante con los errores resaltados en un formato preestablecido. Por otro lado, para circuitos más complejos no se utiliza la simulación, en cambio, se hace uso de una herramienta de comprobación de equivalencia lógica para evaluar el diseño del estudiante contra varios diseños predefinidos. Los diseños para comparar se encuentran en una base de datos y cada uno posee una realimentación específica, con los errores más comunes. De no encontrar equivalencia con ningún diseño, se debe realizar la calificación de forma manual y el diseño se incluye en la base de datos para futuras comparaciones.

Roy et al. (2016) presentan una herramienta web para que los estudiantes puedan realizar la construcción de circuitos, simularlos y analizar los resultados. Para ello desarrollaron su propio simulador que está incluido en su herramienta llamada COLDVL. La herramienta contiene los enunciados de los ejercicios que el estudiante puede resolver, además provee un espacio para que el estudiante diseñe su circuito de forma gráfica. Al simular el circuito, el estudiante puede ver el diagrama de tiempo generado y analizar su contenido. Adicional a los ejercicios por defecto, es posible generar actividades de forma dinámica, para que cada estudiante tenga ejercicios diferentes, de tres temas diferentes. Estos temas son: Funciones Lógicas, Máquinas de Estado Finito y Sumadores de *carry look-ahead*. Finalmente, para la calificación automática de los ejercicios se puede realizar mediante la simulación o la verificación formal.

En la Universidad Politécnica de Cataluña, se desarrolló una herramienta que utiliza los jueces virtua-

les para ayudar a la enseñanza de lenguajes de programación llamada Jutge.org (Petit et al., 2018). La inclusión de los jueces en la enseñanza brinda algunas ventajas como lo son la calificación continua e inmediata, así como permitir al estudiante progresar a su propio ritmo. Inicialmente, Jutge.org estaba enfocada a los lenguajes de programación, sin embargo, se agregó soporte para Lenguajes de Descripción de Hardware. En particular, la herramienta tiene ejercicios que se deben resolver usando Verilog. Mediante la verificación formal se realiza la comparación del diseño del estudiante con el modelo de referencia. Si se presenta algún error en la lógica del circuito descrito por el estudiante, la herramienta muestra un contraejemplo de las condiciones en las que el circuito no funciona correctamente. Es una herramienta en línea, disponible en `https://www.jutge.org`. Permite ver los problemas en un navegador web y realizar un envío con la solución planteada mediante la carga de archivos desde el computador del estudiante. Los autores utilizaron la herramienta en un curso intensivo de 7 días de diseño de circuitos lógicos para estudiantes de secundaria. Se observó que los estudiantes utilizaban la herramienta en horario adicional. Los autores enuncian que la productividad observada no hubiera sido posible ni teniendo 2 o 3 instructores. En una encuesta realizada finalizado el curso, mostró un alto grado de satisfacción por parte de los estudiantes. Sin embargo, los autores resaltan que la realimentación al dar los errores puntuales genera que se pierda en los estudiantes la habilidad de simular y encontrar errores por sí mismos.

En síntesis, utilizando la información recogida de las herramientas antes descritas se puede observar que la mayoría está enfocada al diseño utilizando componentes gráficos pero se han visto avances con el uso de HDL. Por otra parte, las herramientas se han inclinado por estar disponibles en la web en lugar de ser aplicaciones de escritorio. Adicionalmente, se identifican ciertas características que son compartidas entre los trabajos expuestos: algunas herramientas permiten que tanto el profesor como el usuario tenga conocimiento del progreso del estudiante durante el curso. Por otra parte, la calificación automática de los ejercicios ha sido deseada por algunos autores para facilitar el trabajo del profesor. Así mismo, algunas herramientas proveen una realimentación adicional al estudiante con información de los fallos que puede estar cometiendo. Tanto la calificación automática como la realimentación están presentes cada vez que el estudiante quiere evaluar su diseño, menos en el proceso presentado por Nutter et al. (2014) en el cual solo se ejecutan una vez cumplida la fecha de entrega. Finalmente, una característica compartida por algunos trabajos que funcionan sobre Internet, es la posibilidad de editar el diseño en línea sin la necesidad de tener una herramienta adicional instalada en el computador del estudiante. El resumen de que herramientas tienen las características antes mencionadas se muestra en la Tabla **2-1**.

2.3. Selección de las características que ayudan al aprendizaje de HDL

Actualmente los Lenguajes de Descripción de Hardware son utilizados ampliamente en la industria (Greenwood, 2013), por lo que se sugiere que el aprendizaje de las bases de la electrónica digital esté

Tabla 2-1.: Resumen de características de trabajos relacionados

Herramienta		Diseño		Disponibilidad		Funcionalidades			
Nombre	**Autores**	**Gráfico**	**HDL**	**Escritorio**	**Web**	**Seguimiento**	**Calificación**	**Realimentación**	**Editor en línea**
	Mateev et al. (2004)	✓			✓			✓	✓
e-EDU	Robal and Kalja (2007)	✓			✓	✓			✓
SDLDS	Stanisavljevic et al. (2013)	✓		✓		✓			
VerilUOC	Baneres et al. (2014)	✓		✓	✓	✓	✓	✓	
	Nutter et al. (2014)	✓	✓				✓*	✓*	
COLDVL	Roy et al. (2016)	✓			✓		✓	✓	✓
Jutge.org	Petit et al. (2018)		✓		✓	✓	✓	✓	

* La calificación y la realimentación se ejecuta una única vez en la fecha de entrega

acompañado del uso de estos lenguajes (ACM and IEEE-CS, 2016). Para utilizar correctamente los Lenguajes de Descripción de Hardware es necesario tener conocimiento de las bases de la electrónica digital, por lo que a pesar que la mayoría de las herramientas presentadas previamente giran en torno al diseño utilizando componentes gráficos, estos trabajos aportan elementos para el aprendizaje de circuitos lógicos. Por lo que teniendo en cuenta las herramientas antes presentadas y la experiencia del autor de este trabajo final como estudiante de Ingeniería Electrónica, se seleccionaron algunas características que podrían aportar al aprendizaje de HDL.

Como se puede ver en los trabajos presentados, es ideal tener una herramienta adicional para enseñar electrónica digital. Esto se debe a que los estudiantes necesitan practicar constantemente (Mateev et al., 2004). Además que esta práctica debe estar acompañada de realimentación que permita fortalecer el proceso de aprendizaje (Baneres et al., 2014). Esta herramienta complementaria también debe ofrecer a los estudiantes la realimentación en el momento adecuado y no sobrecargar a los profesores con el trabajo de revisar todos los ejercicios de los estudiantes (Nutter et al., 2014). Por lo que la calificación automática y la realimentación constante son pilares claves para el aprendizaje de los Lenguajes de Descripción de Hardware.

Por otro lado, existen otras ideas presentadas en los trabajos relacionados que ayudan en forma indirecta a tener mejores condiciones para el aprendizaje del diseño de circuitos digitales. Entre ellas se encuentra el valor de las aplicaciones a las que se puede acceder vía Internet pues elimina restricciones de tiempo y distancia (Robal and Kalja, 2007). Otro aspecto a considerar en el software es la interfaz y experiencia de usuario (UX/UI), ya que la herramienta debe ser amigable con el usuario (Baneres et al., 2014). Por otra parte, con el ánimo de garantizar que el estudiante no tenga que instalar ninguna aplicación adicional en su computador y los ejercicios asignados puedan ser resueltos desde el navegador web, la opción más apropiada es la integración de un editor en línea, con lo cual el estudiante podrá ver su código con resaltado de sintaxis.

Acorde con lo anterior, sintetizando las ideas presentadas, las características elegidas que serán integradas a la herramienta presentada en este trabajo final son:

- Uso de Lenguajes de Descripción de Hardware para solucionar problemas de electrónica digital.
- Calificación automática de los ejercicios solucionados por los estudiantes.
- Realimentación constante de errores tanto del código como del diseño planteado por los estudiantes.
- Acceso en línea con disponibilidad 24/7.
- Editor en línea con resaltado de sintaxis.

3. Diseño de la herramienta para el apoyo al aprendizaje de HDL

En este capítulo se presenta el diseño de una herramienta de software de calificación automática con proceso de realimentación, teniendo en cuenta las características elegidas que ayudan al aprendizaje de HDL. Se utilizará la estrategia de desarrollo orientada al prototipado (Pomberger et al., 1998). A continuación se presentará la definición de los requerimientos, la arquitectura que tendrá la herramienta y el diseño conceptual de la misma.

3.1. Metodología

Para el diseño e implementación de la herramienta se utilizó la estrategia de desarrollo orientada a la creación de prototipos. Pomberger et al. (1998) definen esta estrategia como un proceso no lineal sino iterativo que muestra resultados lo más pronto posible. La experiencia de estos autores les ha enseñado que al no tener descripciones estáticas y permitiendo que la especificación de requisitos y la arquitectura se desarrollen paso a paso es más factible lograr el objetivo deseado, ya que un prototipo no requiere contener la solución a todos los requerimientos.

La Figura **3-1**, tomada de Pomberger and Bischofberger (1992), muestra los pasos y ciclos que tiene el desarrollo de software utilizando el paradigma de creación de prototipos. Inicialmente, se tiene un problema a resolver del cual se realiza un análisis de los requerimientos. La definición inicial de los requerimientos se logra con una descripción informal de las necesidades del usuario. Con los requerimientos generados se procede a la construcción o modificación de un prototipo. Una vez finalizado el prototipo se propone realizar una evaluación del mismo con el usuario, para así encontrar problemas y nuevamente definir los requerimientos. Los autores presentan 3 tipos de prototipos: (1) el prototipado exploratorio que permite refinar los requerimientos al poder revisarlos rápidamente con el usuario con ejemplos realistas. En este caso, los factores más importantes son la funcionalidad, la facilidad en la modificación y la velocidad de desarrollo sobre la calidad del prototipo. (2) el prototipado experimental está enfocado en probar la validez de las especificaciones o de la arquitectura. Finalmente, (3) la creación de prototipos evolutivos está enfocada en el desarrollo incremental del sistema, donde prototipos anteriores sirven de base para continuar creando nuevas versiones y llegando a prototipos que servirán de base al sistema final.

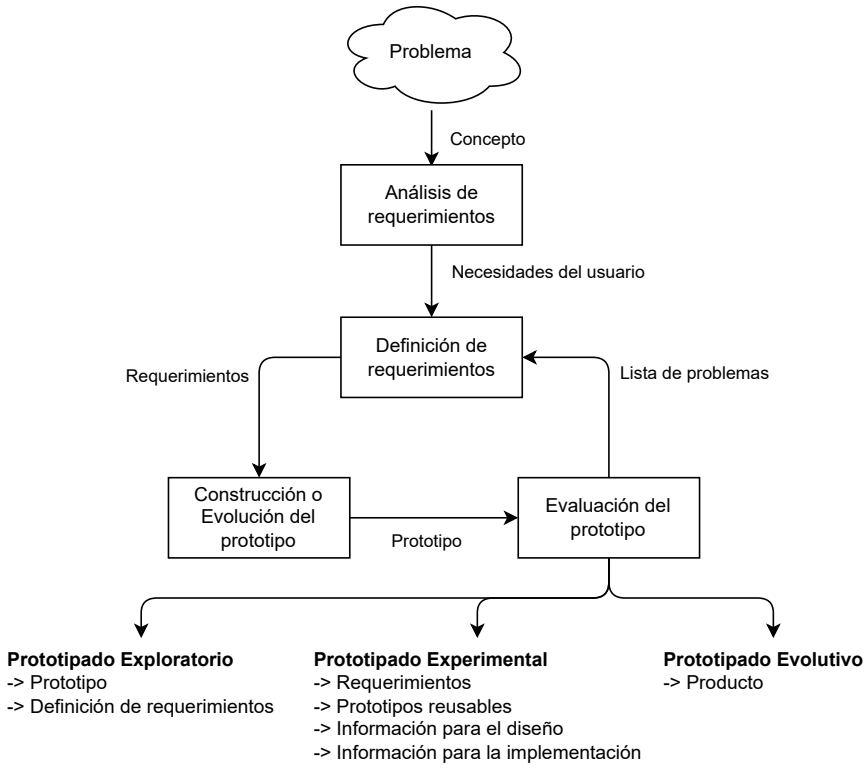


Figura 3-1.: Desarrollo de software usando el paradigma de creación de prototipos. Adaptado de Pomberger and Bischofberger (1992)

El primer paso en el desarrollo de software usando el paradigma de creación de prototipos, corresponde al análisis de los requerimientos. Este paso corresponde a identificar la situación actual y cuáles son las necesidades que tiene el usuario. En el caso de este trabajo, las necesidades son las características elegidas que ayudan al aprendizaje de HDL que fueron identificadas en el capítulo anterior.

3.2. Definición de requerimientos

El segundo paso dentro del desarrollo orientado a la creación de prototipos es transformar la descripción informal de las necesidades del usuario a los requerimientos que serán la base del sistema deseado. Partiendo de las características elegidas que ayudan al aprendizaje de HDL, se definen los requerimientos planteados en la Tabla **3-1**.

El sistema deseado debe ser una aplicación web sobre Internet que facilite el acceso desde cualquier lugar y en cualquier momento. Adicionalmente, que permita mostrar al usuario los problemas (tareas) relacionados con la electrónica digital que debe resolver. Estos problemas deben poderse solucionar utilizando HDLs, mostrando la sintaxis de los mismos. Una vez completados, el sistema debe proveer una calificación y/o la realimentación de los errores cometidos de manera automática.

Tabla 3-1.: Relación características elegidas y requerimientos iniciales

Característica elegida	**Requerimiento**
Uso de Lenguajes de Descripción de Hardware para solucionar problemas de electrónica digital.	El sistema debe mostrar problemas de electrónica digital.
	El sistema permite resolver los ejercicios usando HDL.
Calificación automática de los ejercicios solucionados por los estudiantes.	El sistema debe dar una calificación teniendo en cuenta el código del estudiante.
Realimentación constante de errores tanto del código como del diseño planteado por los estudiantes.	Se debe mostrar una realimentación de los errores en el diseño.
Acceso en línea con disponibilidad 24/7.	El sistema debe ser una aplicación web.
Interfaz amigable con el usuario.	El sistema debe ser fácil de usar.
Editor en línea con resaltado de sintaxis.	El usuario debe ver la sintaxis del HDL que esté utilizando.

3.3. Arquitectura de la herramienta

Dados los requerimientos, se define la arquitectura deseada de la herramienta. La Figura **3-2** muestra la arquitectura inicial de la herramienta. Al ser un desarrollo web, se divide en dos grandes partes: front-end y back-end. El front-end es la parte del sistema con la que el usuario interactúa directamente. En este caso, será el encargado de mostrar el problema de electrónica digital, recibir la solución realizada por el estudiante y proveer de forma gráfica la realimentación y la calificación. Por otro lado, el back-end se encarga de la manipulación de los datos. Particularmente, realizará la calificación del código del estudiante y retornará la información necesaria para mostrar la realimentación, teniendo en cuenta la información del problema.

Para evaluar soluciones utilizando Lenguajes de Descripción de Hardware a problemas de electrónica digital, primero se debe determinar la forma de comprobar que un diseño cumpla con lo esperado. Existen varias formas para saber si un diseño digital cumple con las especificaciones solicitadas. En la mayoría de los casos es necesario contar con un modelo de referencia a partir del cual se realizará la comparación. Las tres formas más comunes encontradas en las herramientas similares para hacer esta comparación son la verificación formal, la equivalencia lógica y la verificación mediante la simulación. La verificación formal consiste en transformar el diseño a modelos matemáticos con el fin de probar el correcto funcionamiento del código mediante razonamiento lógico. Por otra parte, la equivalencia lógica realiza la comparación mediante la simulación simbólica de los diseños. Finalmente, la verificación por simulación utiliza una serie de estímulos que son aplicados tanto a la implementación que quiere ser verificada como al modelo de referencia, y se comparan las salidas de las dos implemen-

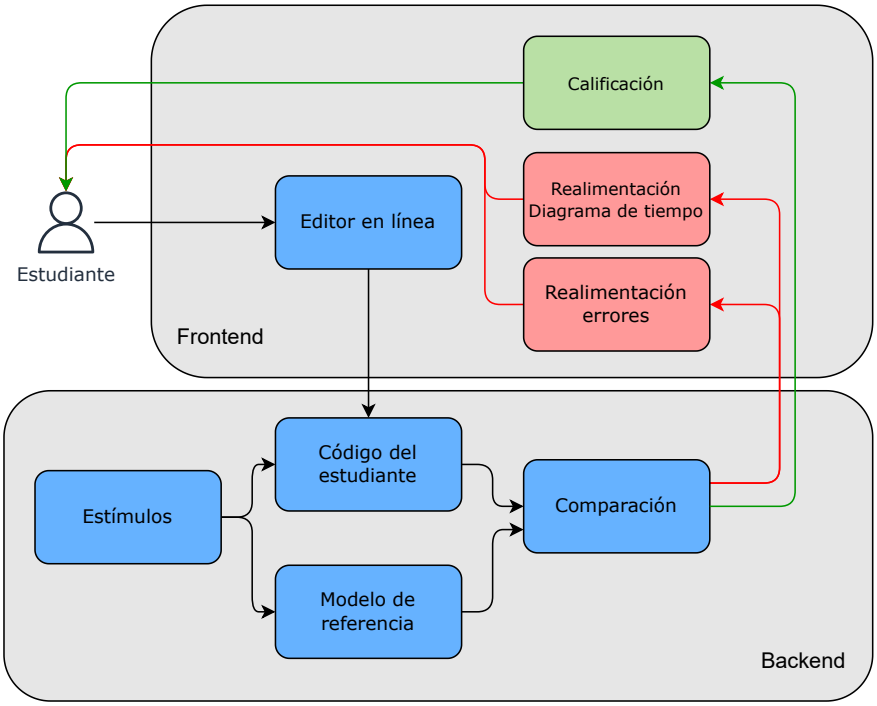


Figura 3-2.: Arquitectura de la herramienta

taciones para encontrar posibles diferencias (Fujita et al., 2008). Para esta herramienta se seleccionó la verificación utilizando simulación. Esto debido a que la verificación formal en muchas ocasiones puede ser computacionalmente costosa; y la equivalencia lógica generalmente utiliza herramientas privativas, y dado el alcance de este Trabajo Final, es posible realizar las acciones necesarias con la comparación de salidas en la simulación del código del estudiante y el modelo de referencia.

Las simulaciones utilizando HDLs necesitan dos componentes. El primero es el diseño. Este contiene las sentencias para definir el comportamiento del circuito y puede estar compuesto por uno o más archivos. El segundo es un archivo comúnmente llamado *testbench* que incluye el llamado a los archivos de diseño para su simulación. Este archivo contiene las instrucciones para aplicar estímulos en las entradas del diseño en diferentes instantes de tiempo y poder registrar los valores en las salidas (La Meres, 2019). Estos estímulos van a ser aplicados tanto al código del estudiante como a un modelo de referencia previamente creado.

Por su parte, el modelo de referencia contiene la lógica que da solución al problema de electrónica digital que fue presentado al estudiante, mientras que el código del estudiante se obtiene de lo que el estudiante envíe desde el navegador web. A estos dos diseños se le aplican los mismos estímulos y se procede a realizar la comparación de las salidas. En el caso que la simulación del código del estudiante falle, se reporta el error. Por otro lado, si la simulación finaliza sin ningún error, hay dos posibilidades. Si la comparación de las salidas de la simulación, tanto del diseño del estudiante como del modelo

de referencia, son iguales, el bloque comparación reporta que el código del estudiante dio solución al problema satisfactoriamente. Si la comparación de las salidas determina que son diferentes, se retornan los valores para las entradas (estímulos) en los cuales el diseño propuesto por el estudiante no tuvo el mismo comportamiento que el modelo de referencia.

Todo este proceso será realizado en el back-end, que tendrá como entradas el código del estudiante y podrá tener 3 salidas: los errores durante la simulación, los casos de error cuando exista una diferencia con la simulación del modelo de referencia, y la validación que el diseño se comportó de manera esperada con los estímulos aplicados.

Adicionalmente, el estudiante debe poder interactuar con una interfaz gráfica de usuario (front-end), donde puede introducir sus diseños y recibir la realimentación de su solución. El editor en línea permite al estudiante introducir el código sin tener que depender de herramientas adicionales. Este editor debe incluir el resaltado de la sintaxis.

Una vez que el estudiante somete a evaluación su solución, y teniendo en cuenta la respuesta del back-end, el sistema procede a entregar la realimentación correspondiente al estudiante. La herramienta contendrá realimentación sumativa y realimentación formativa. La realimentación sumativa tiene como objetivo hacer que el estudiante comprenda hasta donde ha llegado en cuanto a los objetivos de aprendizaje del curso y usualmente va conectada a una calificación cuantitativa. Por otro lado, la realimentación formativa está enfocada en proveer información directa al estudiante que permita conocer cómo está siendo su desempeño en la resolución del problema y cómo podría hacerlo mejor (Kusairi, 2019). En el caso de la herramienta se planea dar tanto una calificación como mostrar los errores de compilación y las diferencias de las salidas con respecto al modelo de referencia.

Una manera de mostrar las diferencias entre el modelo de referencia y la solución del estudiante es por medio de un diagrama de tiempo. Los diagramas de tiempo son una representación gráfica de la relación entre la salida y la entrada con respecto al tiempo. Es una forma útil de ver el comportamiento de los valores lógicos de la señal (La Meres, 2019). La Figura **3-3** muestra un ejemplo de un diagrama de tiempo. El eje *x* corresponde al tiempo y en el eje *y* podemos observar diferentes señales. En este ejemplo, las señales A, B y C corresponden a entradas que van tomando diferentes valores a través del tiempo. En la señal D se muestra la salida que tendría el circuito cuando las señales de entrada tienen ciertos valores. Para el caso de la herramienta de este trabajo final, en caso que alguna de las salidas no corresponda con el comportamiento durante la simulación, se mostrará tanto la salida del circuito del estudiante como la del modelo de referencia.

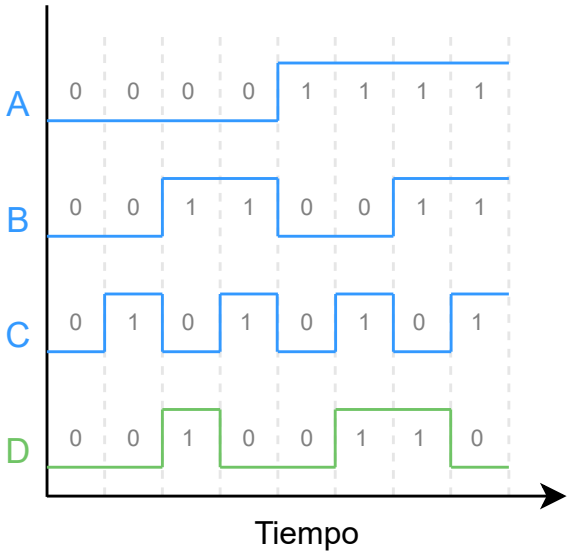


Figura 3-3.: Ejemplo de un diagrama de tiempo

3.4. Diseño conceptual

Siguiendo la estrategia de desarrollo orientada a creación de prototipos, en la Figura **3-4** se observa un bosquejo inicial de las partes necesarias que debe tener el prototipo deseado.

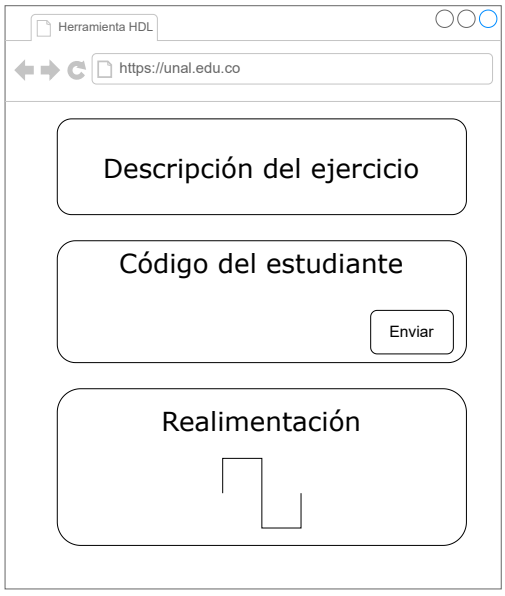


Figura 3-4.: Bosquejo de la herramienta

La aplicación web debe tener una parte para transmitir el enunciado del problema de electrónica digital al estudiante. Adicionalmente, debe poseer un editor en línea donde el usuario escriba el código

HDL de la solución y pueda enviar la propuesta de solución. Finalmente, la aplicación web debe mostrar la realimentación de la calificación y de los errores en la simulación o un diagrama de tiempo con las diferencias encontradas. Teniendo en cuenta el diseño presentado en este capítulo se realizarán dos prototipos. El primero será un prototipo experimental que tendrá como fin experimentar con la posibilidad de realizar la comparación entre un código objetivo y el código a evaluar, mostrando las diferencias en un diagrama de tiempo. Por otra parte, el segundo prototipo será un prototipo evolutivo que tomará como base el prototipo experimental y será el prototipo usado por los estudiantes.

4. Implementación del Prototipo de la herramienta UNCode-Digital

Este capítulo muestra la implementación de dos prototipos de una herramienta de software de calificación automática para el apoyo a la enseñanza de HDL. Inicialmente, se presenta un prototipo experimental que fue implementado con el objetivo de probar el proceso de simulación y el proceso de realimentación utilizando diagramas de tiempo. Este prototipo sirve como base para la implementación del prototipo que finalmente será usado con los estudiantes. El prototipo evolutivo será una integración del prototipo inicial, conteniendo los requerimientos de diseño.

4.1. Prototipo experimental

El primer prototipo realizado, al ser un prototipo experimental, incluyó algunos de los requerimientos, los cuales son listados a continuación:

- Compilar y simular código escrito en HDL, específicamente en los lenguajes Verilog y VHDL.
- Realizar la comparación entre los resultados del código del estudiante y los del modelo de referencia a partir de unos estímulos conocidos.
- Traducir los resultados de la comparación a un diagrama de tiempos.

4.1.1. Recursos necesarios para el prototipo experimental

Para realizar el primer prototipo fue necesario utilizar e integrar algunas herramientas de software ya existentes. A continuación se presenta una descripción de cada una de ellas.

Simuladores HDL

El software que realiza la tarea de compilar, sintetizar y/o simular un diseño digital tradicionalmente ha estado ligado a las empresas que desarrollan el hardware donde finalmente se va a sintetizar el código o a grandes compañías dedicadas a este fin. En otras palabras, generalmente se trata de aplicaciones comerciales. Sin embargo, al igual que en el software, las versiones Open Source de hardware han tenido gran aceptación en los últimos años. Esta iniciativa no sólo está enfocada en el hardware,

sino también en el software necesario para el diseño de hardware. Por lo que la comunidad está haciendo un esfuerzo en la creación de herramientas de síntesis de lenguajes de descripción de hardware no propietarias. Para el prototipo desarrollado durante este trabajo final se desea que la herramienta sea independiente de los fabricantes, por lo que se optó por herramientas open source. La mayoría de estas herramientas son suficientemente robustas para utilizarlas en la docencia. En nuestro caso serán usadas para simular diseños digitales sencillos.

Los dos HDL más usados tanto en la docencia como en la industria son Verilog y VHDL. Para este prototipo se necesita un simulador para cada uno de ellos.

En primer lugar, existen dos principales simuladores open source de Verilog, los cuales son: Icarus Verilog[1] y Verilator[2]. La principal ventaja de la herramienta Verilator es la posibilidad de escribir pruebas en lenguaje C++, lo que permite que la ejecución de las mismas sea eficiente. Sin embargo, dado que los docentes que utilizarán este prototipo pueden estar más familiarizados en realizar las pruebas directamente con el lenguaje Verilog, se decidió utilizar Icarus Verilog. Adicionalmente, Icarus Verilog tiene una gran comunidad por lo que de presentarse alguna situación posiblemente la solución podría encontrarse más rápido.

En el caso de VHDL, la principal alternativa open source es GHDL[3]. Esta herramienta permite analizar, compilar, simular y sintetizar código VHDL. Soporta completamente las versiones de VHDL hasta 2002 y parcialmente las revisiones de 2008 y 2019.

Por lo anterior, las dos herramientas para simulación seleccionadas fueron Icarus Verilog y GHDL para Verilog y VHDL, respectivamente.

Diagrama de tiempo

Las herramientas para mostrar las formas de onda resultantes de las simulaciones, también llamados diagramas de tiempos, usualmente están acopladas en el software de desarrollo. Sin embargo, existen alternativas para la visualización, tales como: GTKWave[4], Gigawave viewer[5] y Wavedrom[6]. Sin embargo, las dos primeras opciones son aplicaciones de escritorio, y dado que una de las características deseadas es no depender de instalar software adicional en el computador del estudiante, éstas fueron descartadas. Por lo que la herramienta seleccionada fue WaveDrom, pues permite crear diagramas de tiempo a partir de texto plano, y adicionalmente permite una fácil integración con una aplicación web.

[1] `http://iverilog.icarus.com/`
[2] `https://www.veripool.org/verilator/`
[3] `https://github.com/ghdl/ghdl`
[4] `https://github.com/gtkwave/gtkwave`
[5] `https://www.syncad.com/vcd_waveform_viewer.htm`
[6] `https://wavedrom.com/`

WaveDrom fue propuesta por Chapyzhenka and Probell (2016) y permite realizar la representación de diagramas de tiempo partiendo de un archivo basado en la sintaxis de JSON llamado WaveJSON. Es una herramienta Open Source con licencia MIT, está escrita en JavaScript y es integrable a los navegadores web. La idea inicial de la herramienta nació de querer crear una forma de comunicar, de manera visual, el comportamiento de un circuito. De esta manera, los autores plantean una alternativa de visualización que no dependa de hacer la representación en una herramienta de gráficos, una hoja de cálculo o incluso crear una simulación para obtener el diagrama de tiempo; teniendo en cuenta que estos métodos de creación de formas de onda tienen varios inconvenientes como: el tiempo que se necesita, diferentes estilos según el autor, dificultad en el control de versiones, entre otros. Todo ello motivó la creación de WaveDrom.

El Código **4-1** muestra un ejemplo de cómo podría ser un archivo de texto que WaveDrom analizaría y transformaría en un diagrama de tiempo. En la Figura **4-1** podemos ver la representación gráfica del archivo de texto presentado anteriormente. En éste se observa el nombre de las señales y su comportamiento a través del tiempo. Adicionalmente, se pueden tener señales de varios tipos, como por ejemplo, señal de reloj, bus de datos o señal binaria. La señal *clk* que corresponde al reloj del sistema (comúnmente usada en diseños digitales) tiene una configuración especial para no tener que hacer la conmutación en cada intervalo de tiempo. Además, se pueden declarar valores binarios o condensar varios valores en un bus de datos, los cuales pueden tener diferentes colores. También permite mostrar el valor de una señal como desconocido o agregar saltos en el tiempo. Por otra parte, es posible agregar anotaciones entre dos puntos del diagrama como la que se observa en el ejemplo con la anotación "*mensaje*".

```
{
  "signal" : [
      { "name": "clk", "wave": "p.....|..." },
      { "name": "Bus", "wave": "x.345x|=.x",
        "data": ["0x2", "0xD", "0xF", "0x4"] },
      { "name": "Entrada", "wave": "0.1..0|1.0",
        "node": "..A..B..."},
      {},
      { "name": "Salida", "wave": "1.....|01." }
  ],
  "edge": ["A+B mensaje"]
}
```

Código 4-1: Ejemplo de la especificación de un diagrama de tiempo en WaveJSON

La mayoría de las características mencionadas serán utilizadas en el desarrollo de este trabajo final. La

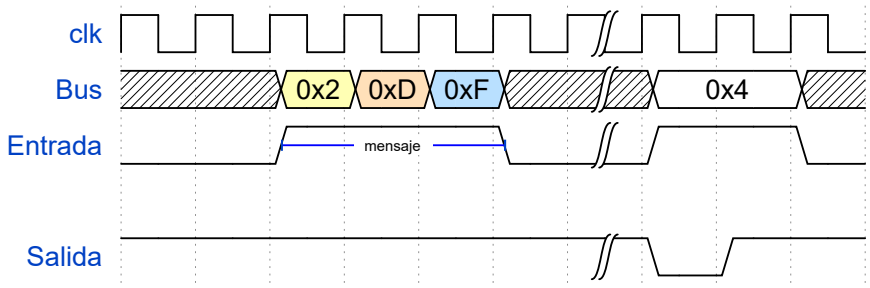


Figura 4-1.: Ejemplo de la representación gráfica de un diagrama de tiempo en WaveDrom

documentación completa del proyecto WaveDrom[7]. Adicionalmente, provee un editor en línea para probar los WaveJSON y ver su representación gráfica[8]

4.1.2. Desarrollo del prototipo experimental

El primero de los 3 requerimientos elegidos para este prototipo es *Compilar y simular código Verilog y VHDL*. Esto con el fin de realizar la comparación entre los resultados del código del estudiante y los del modelo de referencia a partir de los estímulos. Para ello se hace necesaria la creación de un archivo de pruebas, más conocido por su nombre en inglés *testbench*. Este archivo de pruebas contiene el código para generar estímulos que van a ser la entrada del código del estudiante y del modelo de referencia, además del código para observar las salidas. La comparación se realiza con las salidas de los dos casos, el código a probar, que es escrito por el estudiante y el modelo de referencia que es escrito por el profesor.

Para realizar la compilación y simulación fueron utilizadas las herramientas seleccionadas en la sección anterior. Por un lado, para realizar una simulación de Verilog se tienen dos etapas, compilación y simulación. La primera etapa consiste en compilar el código, esto incluye el diseño a analizar y el archivo testbench realizado para probar el diseño. En el caso de Icarus Verilog la compilación se realiza con el siguiente comando:

```
iverilog -o compilacion.out design.v testbench.v
```

Con la bandera `-o` se especifica el archivo de salida que será utilizado en la siguiente etapa. Adicionalmente, se colocan todos los archivos Verilog que serán compilados. Con esta ejecución en caso de existir algún error de sintaxis, éste sería reportado.

Por otro lado, en el caso de la simulación se realiza con el comando:

```
vvp compilacion.out
```

[7] `https://wavedrom.com/`
[8] `https://wavedrom.com/editor.html`

El comando `vvp` está incluido en el software de Icarus Verilog y permite ejecutar la simulación del archivo generado en la etapa anterior. Con esto se realiza la simulación de la cual se obtiene en la salida estándar los mensajes que se hayan colocado en el testbench y un archivo VCD con la información del comportamiento de las señales. El formato VCD es un estándar para *dump files* generados por herramientas de simulación de Verilog y cuyo fin es servir de entrada de las herramientas que muestran los diagramas de tiempo. En este archivo se guarda cualquier cambio que se observe en las señales declaradas en el testbench o los módulos instanciados en éste.

Por otra parte, para realizar la simulación de los diseños en VHDL utilizando GHDL también se tienen 2 etapas. La primera consiste en analizar el código, lo cual se realiza con el siguiente comando:

```
ghdl -a design.vhd testbench.vhd
```

La bandera `-a` especifica que se va a realizar el análisis de los archivos que se envían como parámetro. De este análisis se obtiene cualquier error de sintaxis del código en caso de existir.

La segunda etapa es llamada *construir y ejecutar*. A diferencia de Icarus donde se ejecutan todas las pruebas sin importar el módulo donde se encuentren, en el caso de GHDL se debe especificar el nombre de la entidad que contiene la configuración y las pruebas que se quieren realizar. Para ello se ejecuta un comando como el siguiente:

```
ghdl -r nombre_entidad --vcd=archivo.vcd
```

Por lo anterior, se debe tener claro el nombre de la entidad para poder ejecutar la simulación deseada. Los archivos VCD hacen parte de la definición de Verilog, sin embargo, GHDL incluye la posibilidad de generar archivos con el mismo formato para simulaciones de VHDL, lo cual se consigue con la bandera `--vcd=`, especificando el nombre del archivo de salida.

Con los 4 comandos presentados anteriormente se ejecuta la compilación y ejecución de los archivos en lenguaje de descripción de hardware utilizando los dos simuladores elegidos, con lo que se completó el primer requerimiento.

Para desarrollar los siguientes dos requerimientos, se desarrolló una aplicación web básica, que funciona con PHP y Javascript, y contiene tres campos de texto para ingresar el que sería el diseño del estudiante, el testbench y el modelo de referencia. Para el requerimiento *Realizar la comparación entre los resultados del código del estudiante y los del modelo de referencia* se utilizaron los archivos VCD generados por los simuladores. Para interpretar dichos archivos se utilizó un paquete del lenguaje de programación Python llamado Verilog_VCD[9]. Este paquete permite analizar los archivos de salida de la simulación y transformarlos en un diccionario de Python con toda la información de la simulación.

[9] `https://pypi.org/project/Verilog_VCD/`

Con los datos cargados en Python se procedió a traducir automáticamente hacía la especificación de WaveJSON la simulación con el modelo de referencia. Con esto se obtuvo la descripción del comportamiento del modelo de referencia en formato usado por WaveDrom. La siguiente acción consistió en realizar este mismo proceso pero ahora con el código a evaluar (el que sería ingresado por el estudiante) y realizar la comparación con la salida del modelo de referencia obteniendo un nuevo WaveJSON, cuya representación usando WaveDrom se mostraría en la aplicación web. Esto con el fin de ejecutar el requerimiento *Traducir los resultados de la comparación a un diagrama de tiempo*. En caso de no existir ningún error se mostraría en comportamiento de todas las señales. Por otro lado, si existe alguna diferencia se muestra la señal obtenida y la señal esperada, mostrando los momentos en que la señal fue diferente. Para presentar el comportamiento se crearon dos archivos GIF para Verilog y VHDL. Todo el código anteriormente mencionado puede ser revisado en el siguiente repositorio de GitHub: https://github.com/andrescorso/UNCode-Digital_V0.

En la Figura **4-2** se muestra el diagrama de tiempo obtenido al comparar dos diseños escritos en Verilog. El modelo de referencia contiene estas dos líneas:

```
...
assign F = (!(B) && C) || (B && (!C) && (!D));
assign G[1] = C;
...
```

Sin embargo, en el código a evaluar las asignaciones se hicieron de manera incorrecta, no se realizó una negación y la asignación a un bit de la señal G no es el adecuado, como se muestra a continuación:

```
...
assign F = ((B) && C) || (B && (!C) && (!D));
assign G[1] = B;
...
```

Por lo que en el diagrama de tiempo de la Figura **4-2** se muestran dos señales por cada señal que no corresponda con el diseño esperado. El comportamiento esperado se enuncia con el * al lado del nombre de la señal. Adicionalmente, si la señal es de un bit, se muestra una línea en rojo los momentos en que las señales no concuerdan. En el caso de un bus de datos, todo el valor cambia de color a rojo, mostrando que no corresponde al valor esperado.

4.2. Prototipo evolutivo

El prototipo experimental se toma como base para el desarrollo de un nuevo prototipo. Esta vez, incluyendo todos los requerimientos. Para cumplir con ellos, a continuación se presentan los nuevos recursos utilizados durante la implementación.

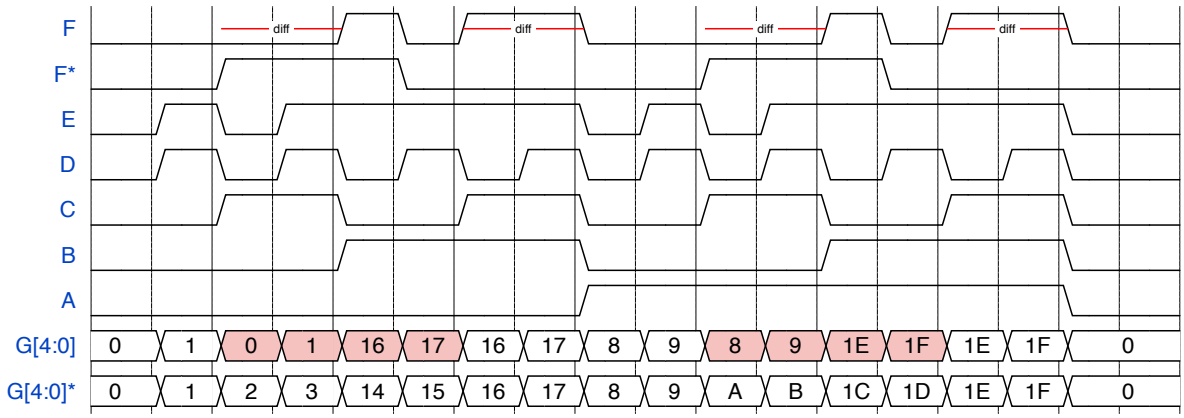


Figura 4-2.: Ejemplo del primer acercamiento a la retroalimentación mediante diagramas de tiempo

4.2.1. Recursos utilizados en el prototipo evolutivo

UNCode

Los jueces virtuales son usados para apoyar el proceso de aprendizaje de los estudiantes en la enseñanza de la programación de computadores. Es por esto que fue desarrollada en la Universidad Nacional de Colombia la herramienta UNCode (Restrepo-Calle et al., 2018). UNCode es un ambiente educativo para la enseñanza de lenguajes de programación. Las características relacionadas con la enseñanza que incluye esta herramienta son:

- Evaluación automática: Sin la intervención del profesor, el software realiza la evaluación de la codificación realizada por un estudiante de la solución a un problema de programación. Esta evaluación se basa en la ejecución de casos de prueba con entradas conocidas, a partir de las cuales se hace la comparación de las salidas esperadas del programa con respecto a las salidas obtenidas por la solución propuesta por los estudiantes.

- Realimentación sumativa: Con el fin de dar una medición del progreso del estudiante, la herramienta UNCode genera una calificación numérica entre 0 y 100 %. Esta calificación se calcula de acuerdo al número y peso de los casos de prueba que la solución del estudiante haya cumplido satisfactoriamente. De esta forma, el sistema permite obtener una calificación parcial.

- Realimentación formativa: Se informa de la existencia o no de errores de sintaxis, semántica y/o eficiencia de la solución. Además, provee mecanismos para ayudar a los estudiantes a mejorar sus propuestas de solución por medio de herramientas que les permiten hallar información acerca de cómo están haciendo la tarea y cómo podrían hacerla de mejor manera.

Considerando que los lenguajes de descripción de hardware tienen algunas características similares a los lenguajes de programación, y por otra parte, que es posible aprovechar el desarrollo realizado de la herramienta UNCode para utilizar jueces virtuales para enriquecer la enseñanza, UNCode será

utilizado como base para el desarrollo del prototipo.

La herramienta UNCode fue desarrollada sobre el calificador automático INGInious (Université Catholique de Louvain, 2014). La documentación de INGInious está disponible en el enlace: `http://inginious.readthedocs.org/`. La documentación del desarrollador, la cual incluye las bases para entender la herramienta y el cómo crear nuevos plugins, sirvió como base para realizar la integración. Los plugins son funcionalidades adicionales que se pueden desarrollar e incluir en el proyecto general sin tener que modificar el código base.

El código fuente de UNCode está disponible en la organización de Github JuezUN[10]. Esta organización contiene un total de 11 repositorios. Los principales son INGInious-containers e INGInious, los cuales contienen el código de los contenedores de Docker[11] y de la herramienta en general. Existe un repositorio llamado Deployment que posee los archivos para el despliegue de la herramienta en un servidor. Al ser una extensión de INGInious, ya contenía ejemplos prácticos por lo que se facilitó la modificación necesaria en este prototipo. Se realizó la instalación de un entorno local de desarrollo de la herramienta para empezar a trabajar en el nuevo prototipo.

La herramienta UNCode está desplegada sobre un servidor con sistema operativo CentOS 7. Posee un servidor web accesible por Internet, en la dirección `https://uncode.unal.edu.co/`, el cual es el portal donde los profesores crean los ejercicios y los estudiantes los resuelven.

La Figura **4-3** muestra el flujo de la forma en que UNCode evalúa el código del estudiante. El flujo empieza con el estudiante presentando su solución a un problema de programación determinado, entonces el código es analizado buscando algún error de sintaxis, el cual es informado al estudiante en caso de existir. De lo contrario, se realiza la ejecución del código y se compara con la salida esperada de cada uno de los casos de prueba. Si existe alguna diferencia entre la salida esperada y la salida de la ejecución, estas son reportadas por medio de texto. En caso que no exista ningún error de sintaxis y la salida de cada ejecución sea la esperada, se informa que todo está correcto al estudiante. Cada envío es almacenado en la plataforma, así como el resultado del mismo.

Contenedores de Docker

Para el manejo de los contenedores, INGInious y por ende UNCode utilizan Docker. Según su sitio web de documentación, Docker es un software para la creación y administración de contenedores. Un contenedor empaqueta un software con todas sus dependencias de forma que la aplicación pueda correr en cualquier plataforma de forma rápida y confiable. Una de las ventajas de Docker es ejecutar varias aplicaciones en la misma infraestructura dando un mejor uso a esta, además de tener procesos

[10] `https://github.com/JuezUN`
[11] `https://www.docker.com/`

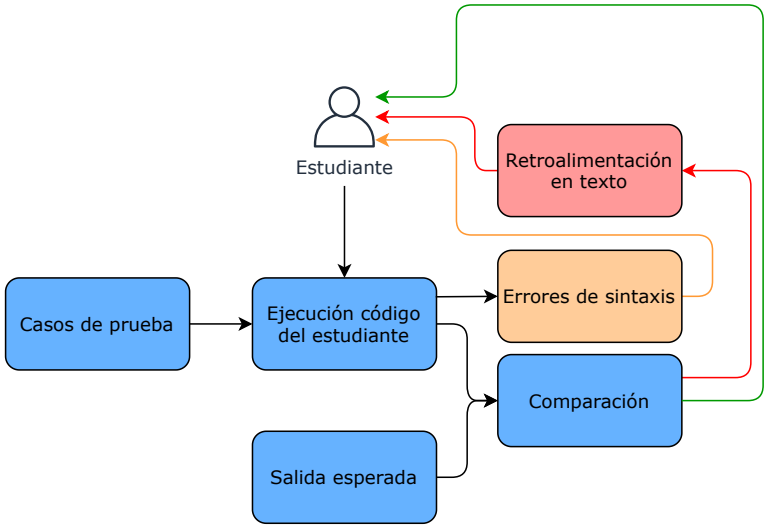


Figura 4-3.: Evaluación de código fuente usando UNCode

totalmente aislados lo que mejora la seguridad. Su funcionamiento gira en torno a imágenes, las cuales puedes ser construidas mediante un archivo que se conoce como Dockerfile. Una vez ejecutada la imagen por medio de Docker, ésta se convierte en un contenedor, que va a ejecutar una tarea en particular.

Los archivos Dockerfile son archivos de texto que contienen las instrucciones para construir imágenes de contenedores automáticamente. Existen diferentes tipos de instrucciones que sirven para obtener el comportamiento deseado. El Código **4-2** muestra un ejemplo de un archivo Dockerfile que usa UNCode para crear uno de sus contenedores. En este archivo podemos observar algunas palabras reservadas que se usan con un objetivo específico. Si la imagen va a estar basada en otra se incluye la instrucción *FROM*. En la línea 5 vemos cómo se agrega metadatos a la imagen que luego va a ser usada por UNCode. Con el comando *RUN* se ejecutan acciones dentro de la imagen; en este caso, se presenta la instalación de algunas dependencias. Con la instrucción *ADD* se agrega el contenido a la imagen. Para el ejemplo del Código **4-2**, se agregó la información que está en la carpeta desde donde se ejecutó la creación de la imagen a una carpeta llamada INGInious. Con este archivo y la ejecución de la creación se obtiene como resultado una imagen que contiene la información que se especificó en el archivo y donde se ejecutaron los comandos necesarios para obtener la aplicación con sus dependencias.

4.2.2. Modificaciones a UNCode

Para el desarrollo del prototipo, se planea agregar un comportamiento adicional a UNCode, como se muestra en la Figura **4-4**. En esta podemos ver la modificación al comportamiento habitual de UNCode, ya que ahora se tiene un modelo de referencia el cual también va a recibir los estímulos, al igual que el código del estudiante. Adicionalmente, se propone mostrar al estudiante la realimentación en

```
# DOCKER-VERSION 1.1.0

# Inherit from the base container, which has all the needed scripts to launch tasks
FROM    unjudge/uncode-c-base
LABEL   org.inginious.grading.name="multiple_languages"

# Add java 1.8
RUN     yum -y install java-1.8.0-openjdk-devel

# Add java 1.7
RUN     yum -y install java-1.7.0-openjdk-devel

# Add gcc
RUN     yum install -y gcc gcc-c++ gdb cpp make valgrind binutils libstdc++ clang clang-devel llvm automake \
        check check-devel CUnit CUnit-devel zlib-devel openssl-devel time jansson-devel

RUN     yum clean all

ADD     . /INGInious
RUN     cp -R /INGInious/grading/. /usr/lib/python3.5/site-packages/grading/
RUN     rm -R /INGInious
```

Código 4-2: Ejemplo de un archivo Dockerfile

un diagrama de tiempo el cual es comúnmente utilizado en el análisis de diseños digitales. Al ser el prototipo una adición al soporte de HDL a UNCode, se refiere a éste como UNCode-Digital.

4.2.3. Arquitectura UNCode-Digital

Como se ha mencionado, UNCode está basado en INGInious, el cual está pensado como un asistente en línea para la enseñanza de lenguajes de programación. Es decir, permite al estudiante probar las soluciones a ejercicios y obtener realimentación sin la intervención del profesor. La Figura **4-5** muestra la arquitectura general de UNCode. INGinious está dividido en back-end y front-end. En el back-end la información de los usuarios y de los cursos se guarda en una base de datos NoSQL (MongoDB[12]) y se utilizan contenedores de Docker para ejecutar cada uno de los códigos de los estudiantes. El

[12] https://www.mongodb.com/

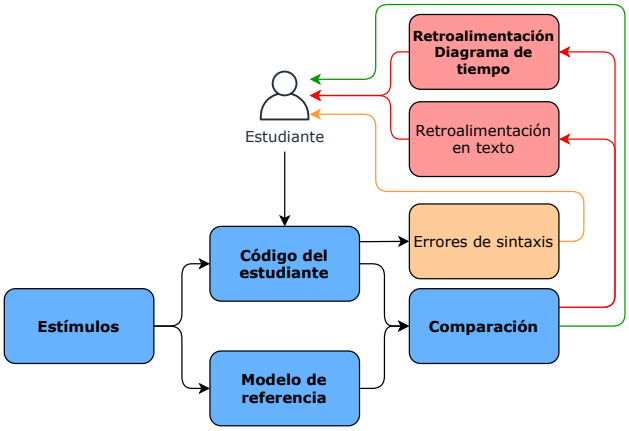


Figura 4-4.: Modificación de calificación de código usando UNCode-Digital

agente INGinious se encarga de utilizar el contenedor según la tarea que se esté resolviendo. Existen diferentes contenedores por grupos de lenguajes de programación y contienen la información del compilador o ejecutable que se utiliza para compilar/ejecutar los códigos. En la parte del front-end, según el usuario que esté conectado, se tiene acceso a la zona de administración, donde se tiene control de la herramienta, o simplemente, el acceso a los diferentes cursos que posee la herramienta. Los cursos están compuestos por estudiantes, tareas y envíos. Cada curso puede tener diferentes estudiantes registrados. Las tareas son los ejercicios o problemas que plantea el profesor. Los envíos son cada uno de los intentos que realizan los estudiantes con una solución de una de las tareas que existen en la herramienta.

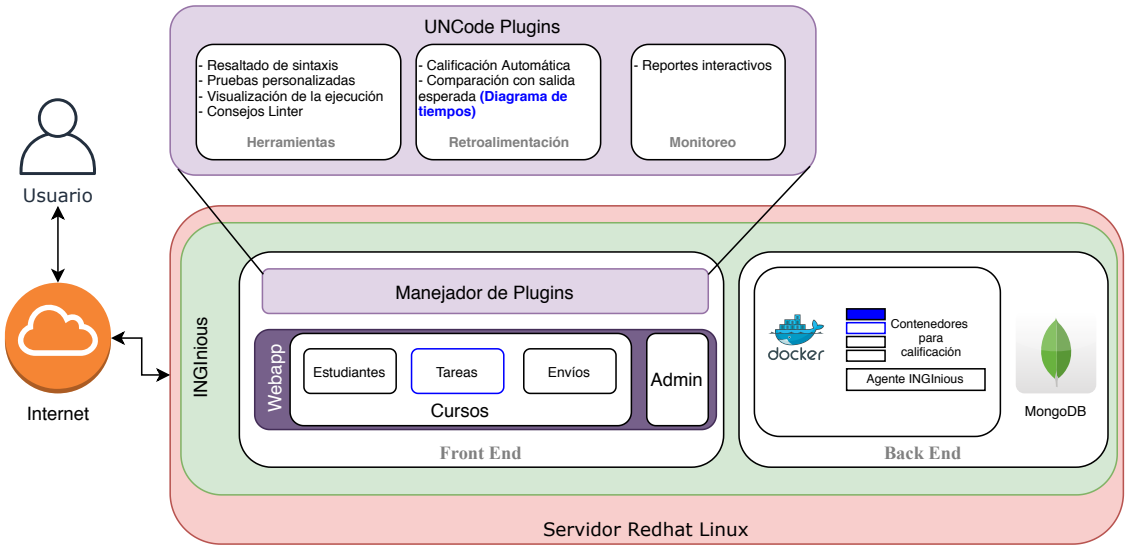


Figura 4-5.: Arquitectura modificada con la inclusión de UNCode-Digital

Por otra parte, INGInious es extensible mediante plugins, los cuales son elementos adicionales que se

pueden llegar a activar para complementar la herramienta base. Adicionando y creando nuevos plugins el proyecto UNCode (Restrepo-Calle et al., 2018) logró aumentar la capacidad de la herramienta INGinious, agregando características para proveer más usos a los estudiantes y a los profesores. Se incluyeron nuevas herramientas, nuevas formas de realimentación y formas de monitorear el avance del curso. Dentro de las herramientas adicionales se encuentra el resaltado de sintaxis, el poder realizar pruebas personalizadas por parte de los estudiantes, el poder visualizar la ejecución del código paso a paso y ofrecer recomendaciones de buenas prácticas de programación (linter). Por el lado de la realimentación, permite la calificación automática de los ejercicios según el número de casos de prueba exitosos, así como la comparación con la salida esperada. También, incluye reportes para los estudiantes y profesores del avance de los diferentes cursos mediante reportes interactivos.

4.2.4. Desarrollo del prototipo integrando UNCode

En la Figura **4-5** se encuentra en color **azul** las partes que van a ser agregadas y modificadas para obtener el prototipo UNCode-Digital. Primero se debe crear un contenedor que incluya los compiladores para realizar la simulación del código escrito en Lenguajes de Descripción de Hardware (Verilog y VHDL). Además, modificar un contenedor existente que servirá como base del nuevo contenedor con las instrucciones para realizar la simulación y comparación entre los diseños. Por otro lado, agregar al plugin de realimentación el mostrar el diagrama de tiempo si existe algún problema con la solución al problema. Finalmente, modificar la forma en que se crean las tareas dentro de la herramienta para incluir las necesidades de las nuevas funcionalidades.

Una de las formas de retroalimentación de UNCode es la que se genera a partir de la diferencia en texto entre la salida obtenida y la salida esperada. Con el fin de dar información adicional a los estudiantes del comportamiento del código, se realizó un cambio en la forma en que se ejecuta la comparación de los diseños, con el fin de analizar texto y dar esta retroalimentación al estudiante. El nuevo enfoque no toma la información de los archivos VCD; en cambio, permite al profesor decidir en qué momento y cuáles señales se van a evaluar. Esto se consiguió mediante la inclusión de sentencias en los testbenchs. Los comandos `$monitor()` y `$display()` en Verilog y `report` en VHDL permiten imprimir texto en la salida estándar durante la simulación. El testbench debe utilizar estos comandos, cada vez que se quiera evaluar las señales, con el siguiente formato:

`T` , *instante* , `INPUTS` , *entrada1*, *valor1*, ..., *entradaN*, *valorN*, , `OUTPUTS` , *salida1*, *valor1*, ..., *salidaN*, *valorN*

Con este formato se logra solo ver las señales deseadas y usar la retroalimentación en texto que provee UNCode.

Para continuar el desarrollo del prototipo es necesario crear un nuevo contenedor de Docker que tiene como función realizar la compilación, simulación y comparación de códigos escritos en HDLs. Antes

de crear la imagen Docker que contiene los simuladores y el código de calificación y retroalimentación fue necesario modificar una imagen que sirve de base a todos los contenedores usados en UNCode. La modificación fue la inclusión de 2 nuevas clases (VerilogProjectFactory y VHDLProjectFactory) en el archivo projects.py. Estas dos clases incluyen las instrucciones para realizar la compilación y simulación según el lenguaje. Una vez actualizada la imagen unjudge/uncode-c-base se procedió a realizar la creación de la nueva imagen. El código fuente está disponible en este enlace. Este corresponde a un contenedor basado en la imagen que fue modificada (unjudge/uncode-c-base) y contiene el software base de INGInious y las librerías de calificación. Adicionalmente, se instalaron los simuladores dentro de la imagen y se incluyó el código de calificación y generación de la retroalimentación. La comparación que se realiza consiste en contrastar las salidas en texto de la simulación del código del estudiante y la salida de la simulación del modelo de referencia en la búsqueda de diferencias.

Por otra parte, la integración de los diagramas de tiempo se realizó a partir de la retroalimentación por texto de UNCode. Una vez la herramienta termina la simulación y la comparación, se retorna al navegador la diferencia que existe entre las dos salidas. Con estos datos y mediante el código escrito en JavaScript se traduce la comparación a diagramas de tiempo utilizando el software WaveDrom. La Figura **4-6** muestra cómo se ve la retroalimentación en texto junto con la retroalimentación con el diagrama de tiempo, si el estudiante se equivoca en su respuesta. A manera de ejemplo, en esta imagen animada se muestra el proceso de evaluación del código al intentar solucionar una tarea por parte de un estudiante usando VHDL.

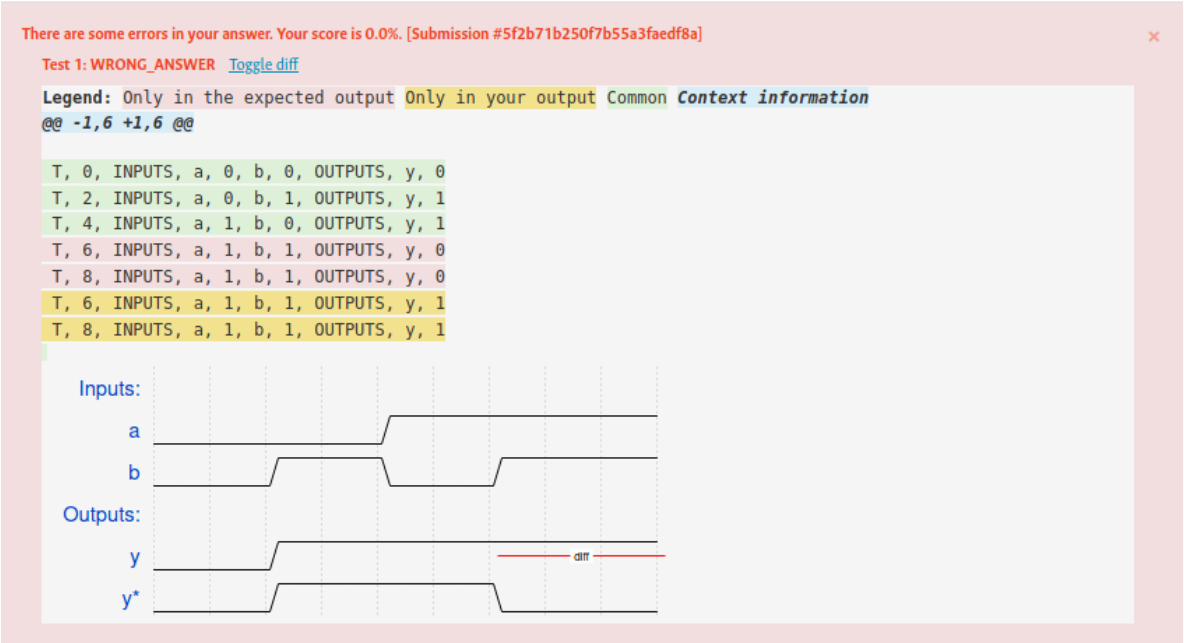


Figura 4-6.: Integración de WaveDrom con UNCode-Digital

En cuanto a los ejercicios con lenguajes de programación, UNCode utiliza un sistema de archivos con entradas y archivos con salidas. Los archivos con entradas, como su nombre lo dice, contienen la información de las entradas que se aplicarán al código del estudiante. Mientras que los archivos con salidas contienen la salida esperada de la ejecución con cada archivo con entradas. Sin embargo, el enfoque que se usa para nuestro caso incluye un modelo de referencia y un archivo testbench. Adicionalmente, para el caso de VHDL se puede especificar la entidad con la que se va a trabajar. Se realizó el cambio para que se pudiera ver como se ve en la Figura **4-7**. En esta imagen animada se puede observar la diferencia y el cambio realizado.

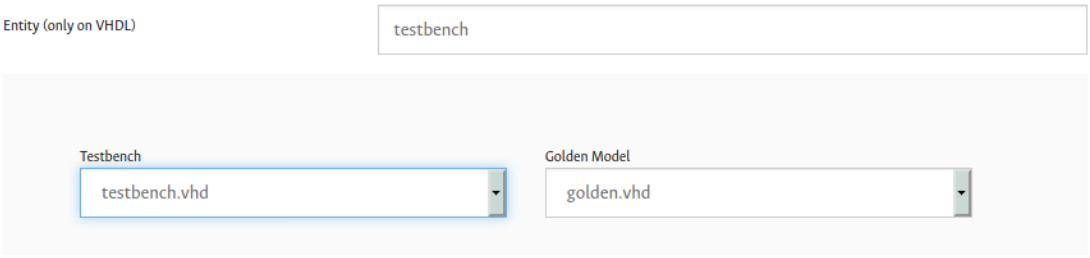


Figura 4-7.: Interfaz para asignar el modelo de referencia y el testbench

Por último, se procede a configurar un plugin adicional que permite la inclusión de WaveDrom en la herramienta. La activación se realiza en el archivo `configuration.yaml` que contiene toda la configuración de la herramienta. El comportamiento fue agregado al archivo de inicialización del plugin multilang, como se muestra en el Código **4-3**. Con el cual se agregan los scripts necesarios a la página web. En este caso se agrega la ubicación de los scripts de WaveDrom y el JavaScript creado para la calificación y creación de la retroalimentación.

En esta sección se presentó el desarrollo de la implementación del prototipo de una herramienta de software de calificación automática con proceso de realimentación para el apoyo al aprendizaje de lenguajes de descripción de hardware, particularmente Verilog y VHDL. Se realizó la integración con UNCode, por lo que el prototipo con la parte implementada en este trabajo final se encuentra activo en `https://uncode.unal.edu.co`. Al igual que el nuevo código hace parte del proyecto UNCode disponible en los repositorios de `https://github.com/JuezUN`.

4.3. Uso del prototipo UNCode-Digital

A continuación se presentan el funcionamiento de dos partes fundamentales del prototipo de la herramienta alrededor de los lenguajes de descripción de hardware. Por un lado, la creación de las tareas por parte del profesor o administrador. Por otro lado, la forma de uso por parte del estudiante al resolver uno de los problemas propuestos por el profesor.

```
64      use_wavedrom = plugin_config.get("use_wavedrom", False)
65      if use_wavedrom:
66          plugin_manager.add_hook("javascript_footer", lambda:
            ↪  "https://wavedrom.com/skins/default.js")
67          plugin_manager.add_hook("javascript_footer", lambda:
            ↪  "https://wavedrom.com/wavedrom.min.js")
68          if use_minified:
69              plugin_manager.add_hook("javascript_footer", lambda:
                ↪  "/multilang/static/hdlgrader.min.js")
70          else:
71              plugin_manager.add_hook("javascript_footer", lambda:
                ↪  "/multilang/static/hdlgrader.js")
```

Código 4-3: Creación plugin use_wavedrom

4.3.1. Creación de tareas

La creación de una tarea se realiza dentro de un curso. Para ver el proceso de forma interactiva está disponible esta imagen animada. La Figura **4-8** ilustra los pasos e información necesaria para crear una tarea.

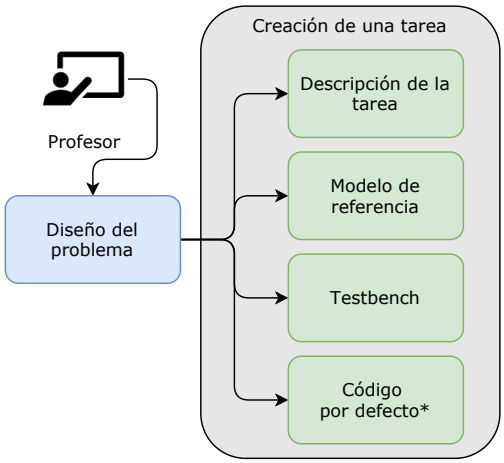


Figura 4-8.: Creación de Tareas

Inicialmente, el profesor debe planear cuál es el problema que desea que contenga la tarea. Para la creación de la tarea es necesario la descripción de la misma, esto se realiza dentro de la herramienta por medio de lenguaje ReStructured Text (RST [13]). El lenguaje RST es un lenguaje de marcas para

[13] https://docutils.sourceforge.io/rst.html

mostrar textos de forma estructurada. Adicionalmente, debe configurar el uso de HDL y el lenguaje en particular que será usado, en algunos formularios dentro del sitio web. Para agregar los archivos necesarios (modelo de referencia y testbench) se tienen dos opciones. La primera consiste en crear y editar los archivos directamente en la herramienta. La segunda, cargando los archivos desde el computador del profesor. Finalmente, se da la opción de agregar un archivo que contiene el código por defecto que verá el estudiante al abrir la tarea. Este archivo no es necesario para el funcionamiento pero si bastante útil para guiar a los estudiantes.

La Figura **4-9** muestra cómo es la página donde el estudiante solucionará la tarea una vez el profesor realice su creación. En esta se incluye la descripción de la tarea y la caja de texto donde el estudiante coloca el código de la solución. Este texto tiene resaltado de sintaxis según el lenguaje en el que se esté escribiendo. Adicionalmente, presenta el historial de los envíos para calificación que ha realizado el estudiante y un botón para que se ejecute el proceso de evaluación. También presenta información de la tarea, la calificación obtenida, el nombre del estudiante y cuál es el envío que se está utilizando para la calificación final.

Figura 4-9.: Elementos dentro de una tarea

4.3.2. Uso por parte del estudiante

Una vez registrado, el estudiante tendrá acceso a los cursos donde haya sido agregado por el profesor, a los cursos sin contraseña o a los cursos de los que conozca la contraseña. En la Figura **4-10** se observa un ejemplo de los cursos en los que está registrado un estudiante y la posibilidad de registrarse en otros.

Figura 4-10.: Cursos de un estudiante de prueba

Dentro del curso, el estudiante puede ver las tareas y su calificación de cada ejercicio. Por ejemplo, la Figura **4-11** muestra dos actividades dentro del curso "Laboratorio de Electrónica Digital I - Grupo 7 - 2020 -1". En una tiene calificación de 100 %, mientras que la otra su calificación es 0 % ya sea porque no ha dado solución al problema o porque no lo ha intentado.

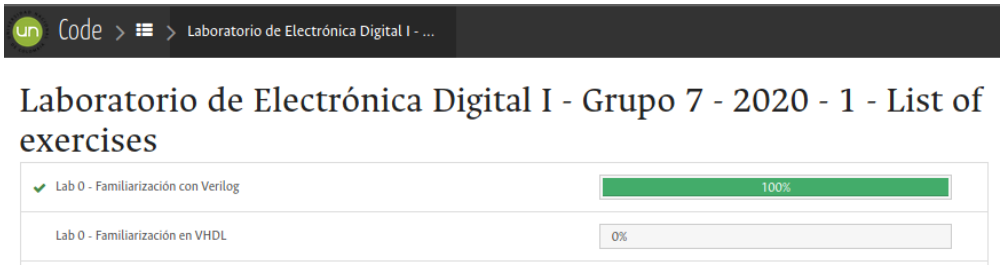


Figura 4-11.: Ejercicios disponibles dentro de un curso

Cuando el estudiante esté dentro de la tarea observará una página similar a lo mostrado en la Figura **4-9**. Una vez enviado el código existen 3 posibilidades. Estas se pueden observar en esta imagen animada. La primera es que el código cumpla con el modelo referencia, por lo que reportaría que se han pasado todas las pruebas como se observa en la Figura **4-12**. La segunda posibilidad es que exista un error de sintaxis o algún error en la estructura del código por lo que le aparecería al estudiante como retroalimentación los errores generados por el compilador, los cuales indican el tipo de error y

la línea en el código donde se produjo. Por ejemplo, si el estudiante envía el código que se observa en la Figura **4-13** recibiría un error de sintaxis debido a que falta un ';' al final de la línea 4. Finalmente, la última opción de respuesta es cuando hay un error en la lógica y el comportamiento de la simulación del código del estudiante no corresponde con el modelo de referencia. Este caso está expuesto en la Figura **4-14**. El objetivo del ejercicio de ejemplo era diseñar un módulo que hiciera un XOR entre dos señales de entrada, sin embargo, el estudiante utilizó el operador '|' que corresponde a un OR, por lo que obtiene la retroalimentación de diferencia en texto y diagrama de tiempo. En esta se observa que en el caso que las dos entradas sean '1' lógico debería obtener un '0' lógico, pero está obteniendo un '1' lógico.

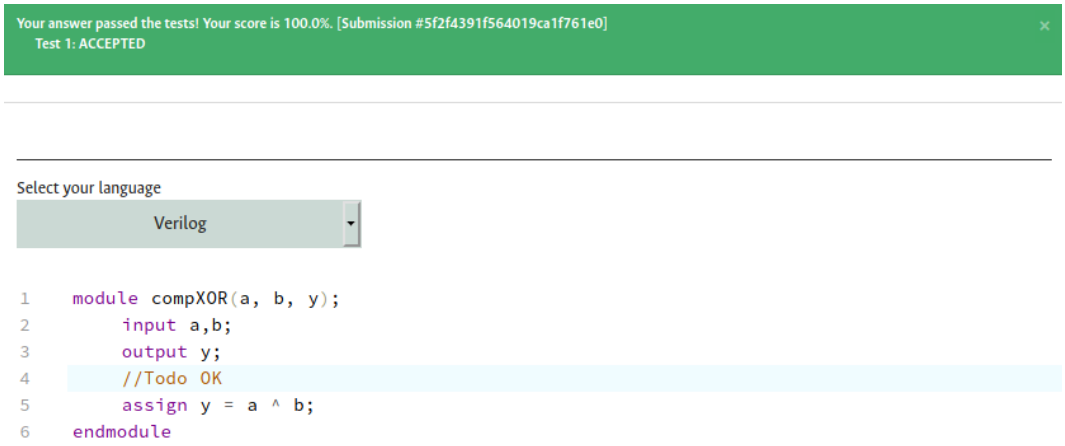


Figura 4-12.: Ejemplo retroalimentación, código sin errores

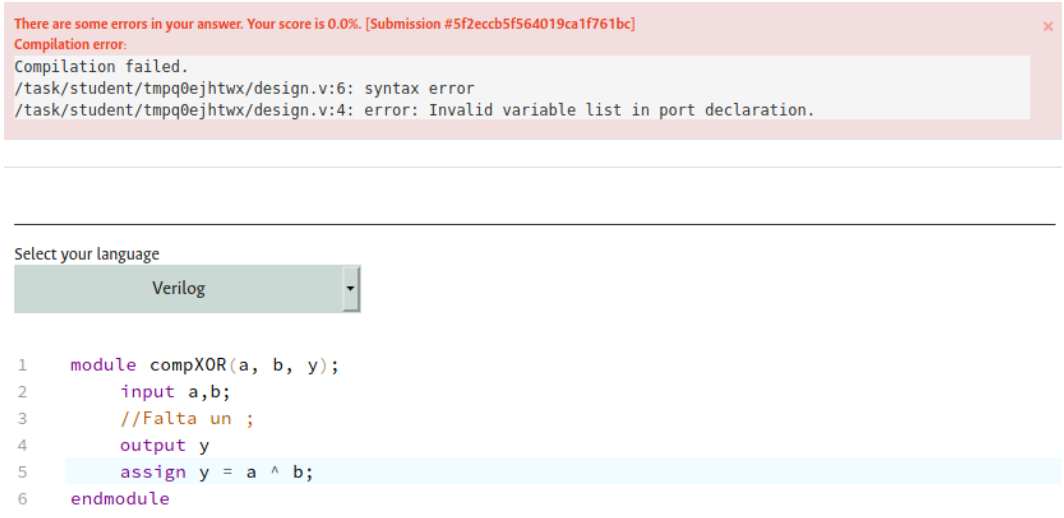
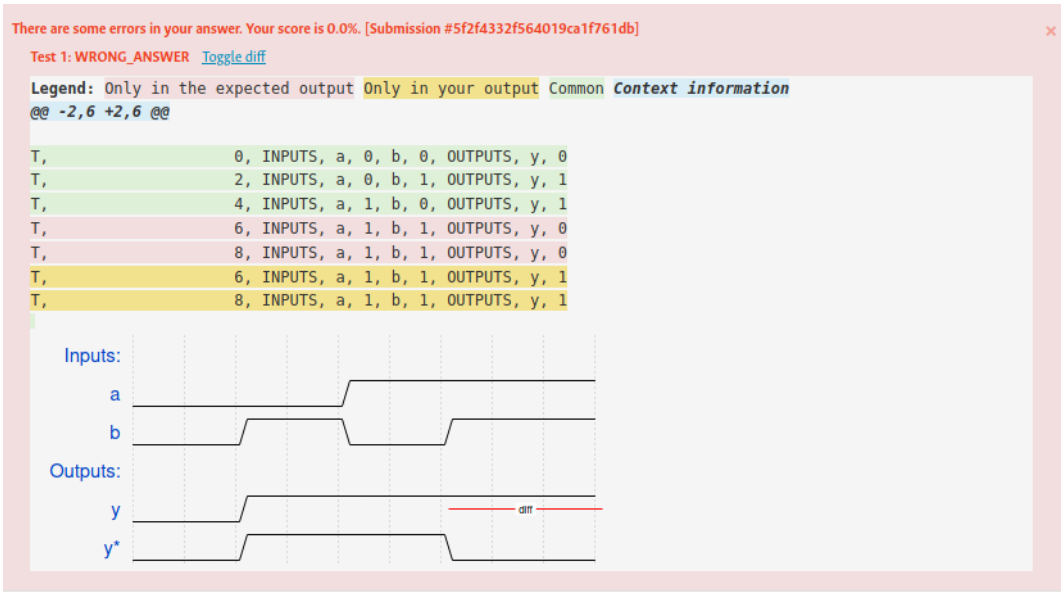


Figura 4-13.: Ejemplo retroalimentación, código con error se sintaxis



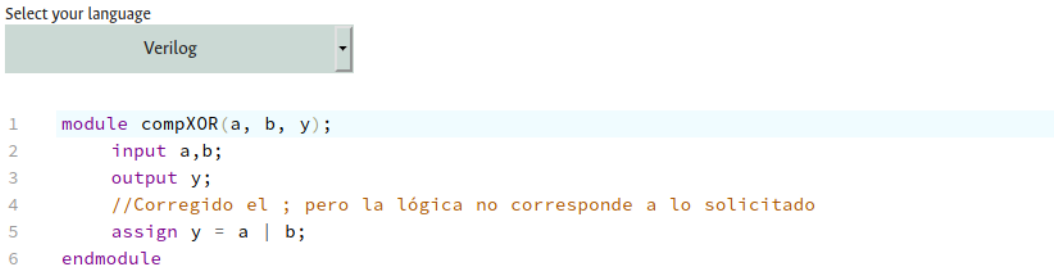


Figura 4-14.: Ejemplo retroalimentación, código con en la lógica

En resumen, se presentó la forma de crear tareas y las posibilidades que tiene el estudiante cuando está solucionando algún ejercicio disponible dentro del curso. Teniendo el prototipo funcional se procede al siguiente paso de este trabajo, el cual consiste en diseñar el estudio que se va a realizar para obtener las percepciones de los estudiantes al usar el prototipo UNCode-Digital.

5. Diseño del estudio

Una vez finalizada las fases de diseño e implementación, el siguiente paso fue el diseño del estudio a realizar para obtener las percepciones de los estudiantes sobre el uso de la herramienta de apoyo en la enseñanza de Lenguajes de Descripción de Hardware. Este capítulo describe el diseño de la experiencia que se desarrolló durante el primer semestre de 2020. Inicialmente se presenta un contexto sobre los participantes en la experiencia de aprendizaje y la explicación de las actividades realizadas durante dicha experiencia. Los estudiantes utilizaron la herramienta UNCode-Digital durante 5 semanas. Una vez finalizado el uso planificado de la herramienta por parte de los estudiantes se procedió a aplicar una encuesta para conocer sus opiniones.

5.1. Participantes

Los participantes en este estudio fueron estudiantes de la asignatura Electrónica Digital I de la Universidad Nacional de Colombia. Esta asignatura está enfocada a que los estudiantes conozcan los fundamentos de la electrónica digital: representaciones numéricas en diferentes bases, la lógica combinacional, lógica secuencial y finaliza con máquinas de estado finito. Se tuvo acceso a estos participantes ya que el autor de este trabajo ofició durante el primer semestre de 2020 como asistente docente, impartiendo a cuatro grupos la asignatura Laboratorio de Electrónica Digital I. En este sentido, el muestreo fue por conveniencia.

En el marco de esta investigación, se considerará como participante al estudiante que realizó al menos un envío en la plataforma UNCode-Digital. Un total de 57 estudiantes interactuó con la herramienta. Los participantes pertenecían a diferentes carreras universitarias, específicamente: 23 de Ingeniería Eléctrica (40.35 %), 21 de Ingeniería Electrónica (36.84 %), 11 de Ingeniería Mecatrónica (19.30 %) y 2 de Ingeniería Mecánica (3.51 %).

5.2. Actividades del laboratorio usando UNCode-Digital

El laboratorio de la asignatura Electrónica Digital I tiene una intensidad horaria de 2 horas presenciales semanales. El lenguaje que se usa para describir el hardware de los circuitos que se implementan en el laboratorio en esta asignatura es Verilog, por lo que todos los ejercicios fueron enfocados a este lenguaje, a pesar de tener también VHDL disponible en la plataforma UNCode-Digital. Las actividades de la experiencia estuvieron repartidas en cinco (5) semanas, en las cuales los estudiantes utilizaron

la herramienta. Debido a las condiciones mundiales durante el primer semestre de 2020 (pandemia Covid-19), los laboratorios se realizaron de manera remota sincrónica.

Se diseñaron prácticas pensadas para guiar al estudiante desde lo más básico y cada vez incrementar un poco el nivel de dificultad. En la Figura **5-1** se muestra la metodología usada durante los laboratorios, en los cuales el profesor realizaba la explicación de un tema y se asignaba a los estudiantes uno o varios ejercicios para que los resolviera usando el prototipo UNCode-Digital.

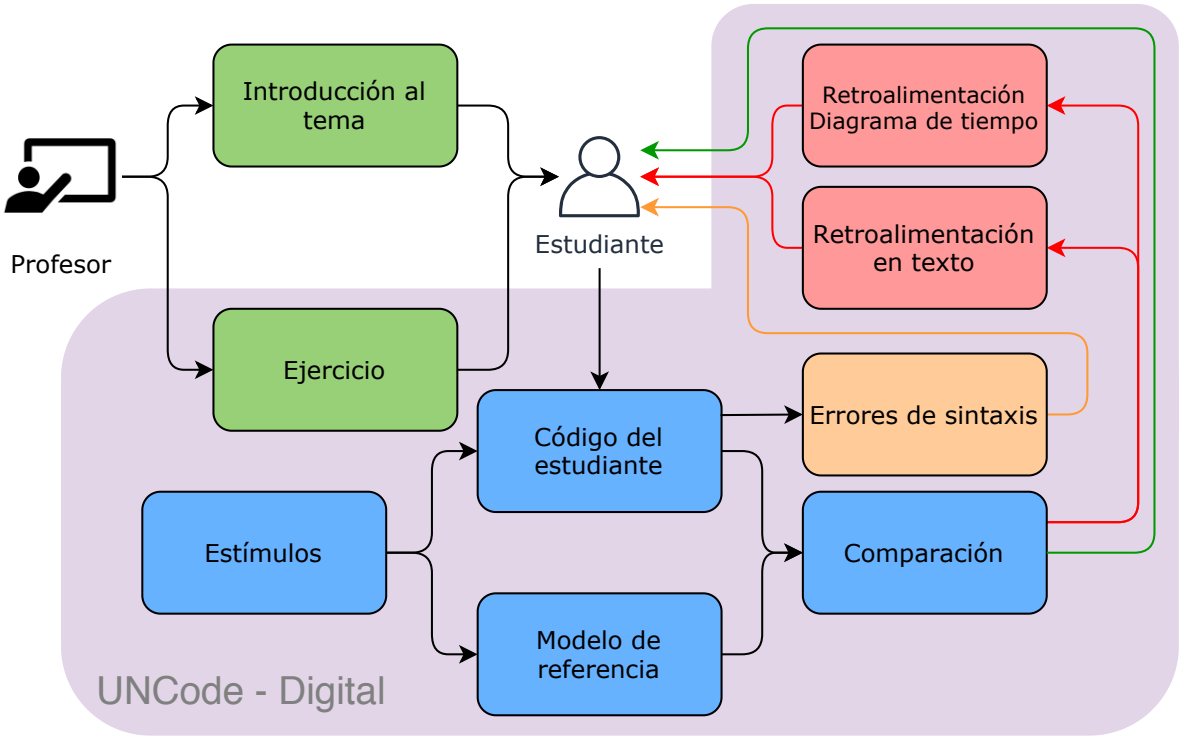


Figura 5-1.: Metodología de los laboratorios

La introducción al tema consistió en realizar mediante vídeos, grabados por el profesor, una explicación al tema del laboratorio, así como la instrucción de los componentes del lenguaje que se iban a utilizar. Posteriormente, se les asignaba un ejercicio para desarrollar de forma individual en la herramienta. Teniendo en cuenta esto, el estudiante implementaba su código como solución al ejercicio planteado. Una vez este código del estudiante era sometido a evaluación, la herramienta le mostraba al estudiante la salida del compilador en caso de que existiera algún error de sintaxis. De lo contrario, la herramienta comparaba la salida de la ejecución del código del estudiante con la salida de la aplicación de los estímulos sobre el modelo de referencia para así obtener el resultado. De estar correcto, se le informaba al estudiante que todo estaba correcto; de otra forma, se le mostraba la realimentación con el Diagrama de Tiempo y la retroalimentación utilizando la diferencia en texto resaltado con colores.

Durante las 5 semanas que duró el experimento se realizaron diferentes actividades con ayuda de la herramienta implementada como parte de este trabajo final. La Tabla **5-1** muestra en resumen las actividades realizadas en cada semana y la codificación en el nombre. Las cuales se explican en detalle a continuación.

Tabla 5-1.: Actividades por semana

Semana	Actividad	Código
1	Familiarización con la herramienta	T0
	Lógica Combinacional	T1
2	Lógica Combinacional	T2
3	Lógica Combinacional	T3
4	Módulos	T4
5	Condicionales	T5

5.2.1. Familiarización con la herramienta

En la primera semana se creó un ejercicio en la herramienta que permitía realizar la familiarización con la misma y además entender la estructura básica del lenguaje de descripción de hardware Verilog. Este laboratorio consistió en diseñar una compuerta XOR de dos entradas y una salida. Esta actividad corresponde a la codificación T0.

5.2.2. Lógica Combinacional

Una vez el estudiante está familiarizado con la herramienta, y durante las primeras 3 semanas, se trabajó con el concepto de lógica combinacional, realizando 2 laboratorios y algunos ejercicios adicionales utilizando la herramienta.

En el primer laboratorio (T1) se utilizó la herramienta para apoyar el aprendizaje de las diferentes formas que tiene Verilog para describir un circuito, por lo que se generaron tres ejercicios (T1_1, T1_2 y T1_3). Las 3 formas para describir el comportamiento de un circuito son: la descripción estructural, donde se instancian compuertas y módulos ya existentes; la descripción funcional, la cual usa los operadores y las asignaciones para expresar la lógica; y finalmente, la descripción procedimental, donde las instrucciones están contenidas en bloques *always*. Así mismo, como trabajo adicional se dejaron 3 ejercicios (T2_1, T2_2 y T2_3) de reducción de la lógica utilizando álgebra de Boole y codificando el resultado usando las descripciones antes mencionadas. Finalmente, se exploraron los mapas de Karnaugh, para obtener el circuito por max-términos y min-términos. La herramienta además revisaba que los estudiantes estuvieran usando el número mínimo de agrupaciones para cada caso. Se realizaron: 1 ejercicio con 3 variables (T3_1), 2 ejercicios con 4 variables (T3_2 y T3_3), y 1 con 5 variables

(T3_4). Estos temas pueden ser profundizados en el libro de circuitos lógicos y diseño digital de La Meres (2019).

5.2.3. Módulos

Los lenguajes de descripción de hardware tienden a ser modulares, es decir, se divide el problema en pequeñas partes y luego se acoplan en un módulo general. Por esto, durante la semana 4 se realizó un laboratorio (T4) en el cual los estudiantes debían realizar un medio sumador, un sumador completo y un sumador de 4 bits. Esto con el fin de ir instanciando cada módulo en el módulo superior. El código de esta tarea es T4_1.

5.2.4. Condicionales

En la quinta semana, la última actividad de la experiencia, el laboratorio (T5) se enfocó en dar herramientas adicionales a los estudiantes para el diseño de circuitos lógicos utilizando lenguajes de descripción de hardware. Se pasó a condicionales, los cuales permiten diseñar un circuito con una abstracción de mayor nivel. En este caso, el objetivo fue mostrar en un display de 7-segmentos una palabra utilizando la sentencia `if / else` (T5_1) y realizar la codificación del abecedario utilizando la sentencia `case` (T5_2).

5.3. Cuestionario para obtener percepciones

Para recolectar información sobre las opiniones de los estudiantes y de allí extraer las percepciones, se realizó un cuestionario en línea mediante la plataforma Google Forms. En este se presentaba el consentimiento informado, disponible como Anexo A, en el que se informaba al estudiante de la investigación y se le solicitaba la autorización para usar sus respuestas en este trabajo final. También, se aclaraba que las respuestas eran confidenciales y no influían en ningún proceso académico.

Las preguntas estaban orientadas a conocer la experiencia que cada estudiante tuvo durante el uso de la herramienta. La utilidad de algunas características de la herramienta fue indagada mediante preguntas cerradas usando una escala Likert de 6 puntos y su correspondiente pregunta abierta *¿Por qué?*. Las características fueron la calificación automática, la retroalimentación, la disponibilidad en línea, la utilidad general para aprender lenguajes de descripción de hardware, además de elementos por mejorar y factores positivos. Finalmente, para tener información no cubierta con las preguntas se dejó una pregunta de comentarios adicionales.

5.3.1. Preguntas

Todas las preguntas, excepto la última, eran obligatorias de responder. Con esto se aseguró no sólo conocer el nivel de conformidad, sino el detalle de las percepciones. A continuación se exponen las preguntas tal como fueron presentadas en el cuestionario a los estudiantes una vez finalizaron las 5 semanas de uso de la herramienta:

Las siguientes preguntas se refieren a su experiencia utilizando la herramienta UNCode para el desarrollo de los laboratorios de Electrónica Digital I.

Para cada una de las siguientes afirmaciones escoja su nivel de acuerdo o desacuerdo según la escala. Los valores de la escala tienen la siguiente interpretación:

1. Totalmente en desacuerdo
2. En desacuerdo
3. Algo en desacuerdo
4. Algo de acuerdo
5. De acuerdo
6. Totalmente de acuerdo

Preguntas con escala Likert:

- La calificación automática que me ofrecía la herramienta UNCode durante los laboratorios me fue útil en el aprendizaje del Lenguaje de Descripción de Hardware Verilog. ¿Por qué?
- La retroalimentación automática usando DIAGRAMA DE TIEMPOS que me ofrecía la herramienta UNCode durante los laboratorios me ayudó a identificar los errores que tenían mis diseños y a analizar cómo corregirlos. ¿Por qué?
- La retroalimentación automática usando TEXTO RESALTADO CON COLORES que me ofrecía la herramienta UNCode durante los laboratorios me ayudó a identificar los errores que tenían mis diseños y a analizar cómo corregirlos. ¿Por qué?
- Considero que la disponibilidad online de la herramienta UNCode es una ventaja importante. ¿Por qué?
- UNCode fue útil en mi proceso de aprendizaje del Lenguaje de Descripción de Hardware Verilog. ¿Por qué?

Preguntas abiertas:

- ¿Qué se podría mejorar de la herramienta UNCode utilizada durante los laboratorios?
- ¿Qué aspectos destaca de la herramienta UNCode utilizada durante los laboratorios?
- ¿Tiene algún comentario adicional?

Las primeras cinco preguntas tienen respuesta en escala Likert y un cuadro de texto para la justificación. Las tres preguntas restantes tenían la posibilidad de respuesta abierta. Una copia del cuestionario se encuentra disponible en `https://forms.gle/BKNoJDwhLwAyrdrE7`.

5.3.2. Procesamiento de respuestas

Para obtener los resultados de este trabajo final, se analizó, además de la interacción de los estudiantes con UNCode-Digital, las respuestas tipo Likert del cuestionario mediante estadística descriptiva, con la ayuda de diagrama de barras apiladas poder observar el nivel de acuerdo o desacuerdo con cada una de las afirmaciones.

Por otra parte, para el análisis cualitativo de las respuestas a las preguntas abiertas del cuestionario se tuvo como referencia el libro Research Social Methods de Bryman (2012). El objetivo de las preguntas era conocer específicamente las opiniones de los estudiantes sobre ciertas características de la herramienta, estas fueron la calificación automática, la retroalimentación, el estar en línea y la utilidad en el proceso de aprendizaje. Por lo que la codificación giró en torno a identificar el juicio que los estudiantes les daban a estas características. Siguiendo el enfoque presentado por el autor, se tomaron las respuestas abiertas y se realizó la codificación de las respuestas dadas por los estudiantes. La codificación es el proceso de identificar por medio de la detección de palabras claves o frases el significado a lo que el estudiante quiere decir. Los códigos obtenidos son llamados indicadores. La agrupación de estos indicadores ayuda a detectar categorías, agrupándolos mediante aspectos en común. Una respuesta puede tener diferentes indicadores y hacer parte de diferentes categorías. Al analizar y clasificar las categorías en una abstracción superior, se obtienen las temáticas. Estas engloban varias categorías con características similares y representan de forma general las percepciones de los alumnos.

El proceso anteriormente descrito es un proceso iterativo, el cual fue realizado por el autor de este trabajo final y los directores del mismo. En cada iteración se realizó el proceso de codificación y categorización para dar una mejor interpretación a las respuestas de los estudiantes. Para lograr esto, se utilizaron varias hojas de cálculo que contenían las respuestas de los estudiantes y la codificación para ir organizando y obteniendo las diferentes categorías y temáticas. En el proceso se identificaron opiniones sobre otras características adicionales de la herramienta, así como ideas de mejora para la misma. El análisis cualitativo se realizó sobre las respuestas de 48 estudiantes.

6. Análisis y discusión de resultados

6.1. Resultados

En esta sección se presentan los resultados obtenidos del uso de la herramienta y de las respuestas de los estudiantes al cuestionario. Inicialmente, se presentan los resultados cuantitativos de la interacción que tuvieron los estudiantes con UNCode-Digital. Posteriormente, se muestra el análisis de las percepciones de los estudiantes respecto al uso de una herramienta de apoyo al aprendizaje de HDL para dar respuesta a la pregunta de investigación: *¿Cuáles son las percepciones de los estudiantes en cuanto al proceso de aprendizaje de HDLs al usar una herramienta de software de calificación automática de apoyo?*.

6.1.1. Análisis de la interacción con UNCode-Digital

A continuación se presentan los resultados de la interacción que tuvieron los estudiantes con la herramienta. De la herramienta UNCode-Digital se recolectó información de su uso. Particularmente, el número de envíos realizados por los estudiantes, es decir, el número de intentos de solución a las tareas propuestas. Los envíos fueron catalogados en respuesta correcta, respuesta incorrecta o error de compilación. La participación en las diferentes tareas se describe en la Figura **6-1**. El promedio de participación por tarea fue de 52 estudiantes de los 57 inscritos en la asignatura, es decir, el 91.22 %. Cabe anotar que solo 53 estudiantes participaron resolviendo o intentando resolver más de 7 ejercicios, en otras palabras, más de la mitad de los ejercicios propuestos. En 6 ejercicios algunos estudiantes no llegaron a la solución; mientras que en los otros 8, la totalidad de los estudiantes que intentaron resolver el ejercicio finalmente lo consiguió.

La Figura **6-2** muestra la información por tarea, con los códigos de la Tabla **5-1**, de los envíos realizados por los estudiantes. Los envíos con respuesta correcta tuvieron un promedio de 68 envíos con una desviación estándar de 4. Se realizó un total de 4488 envíos en las 14 tareas. Las dos tareas con más envíos fueron la tarea T2_1 (Álgebra de Boole con descripción estructural) y T4_1 (Sumador 4 bits). La tarea T2_1 daba al estudiante una ecuación booleana y se solicitaba que antes de realizar la codificación de la misma utilizando la descripción estructural, se simplificara la ecuación utilizando álgebra de Boole y adicionalmente, se pedía incluir una salida adicional, con el número de términos de la ecuación resultante. Se identificaron dos causas al aumento de envíos. La primera, algunos estudiantes tuvieron problemas con la simplificación y no llegaron a la mínima expresión posible. La otra causa identificada fue que algunos estudiantes se les dificultó asignar un número sin ninguna lógica a

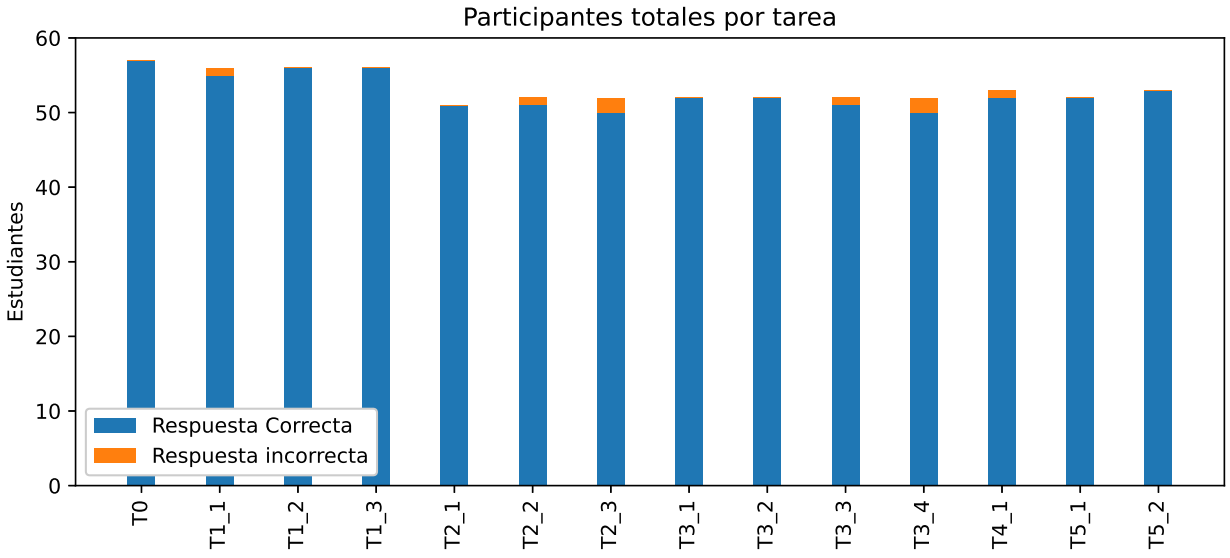


Figura 6-1.: Total de participantes por tarea y resultado final

una salida utilizando la descripción estructural de Verilog. Por otra parte, la causa identificada para el aumento de envíos en la tarea T4_1 fue que hasta este punto, los ejercicios siempre habían sido utilizando solo un módulo de Verilog y al abordar la descripción e interconexión de más de un módulo, esto fue un desafío para algunos estudiantes.

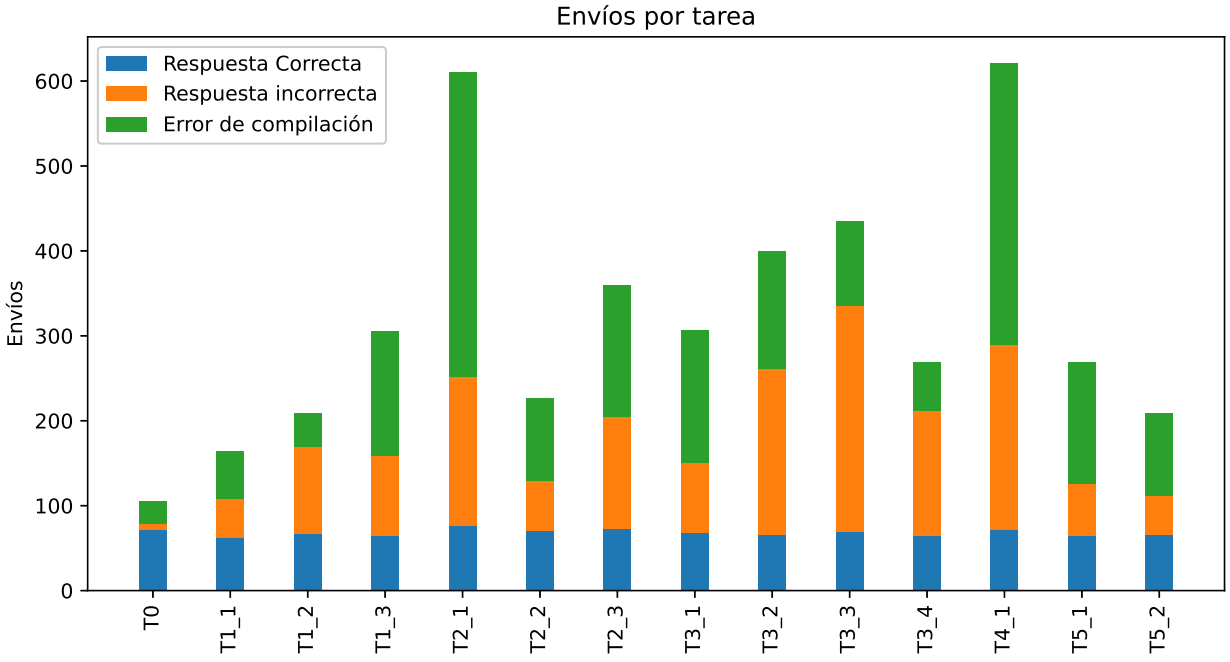


Figura 6-2.: Número de envíos por tarea

La Figura **6-3** muestra el diagrama de cajas del número de envíos por estudiante en cada una de las tareas. Esta figura nos permite observar que en todas las tareas la mediana estuvo por debajo de 10 envíos. Existen algunos datos atípicos sobre todo en la tarea T2_1, donde 6 estudiantes realizaron más de 30 envíos con un máximo de 80.

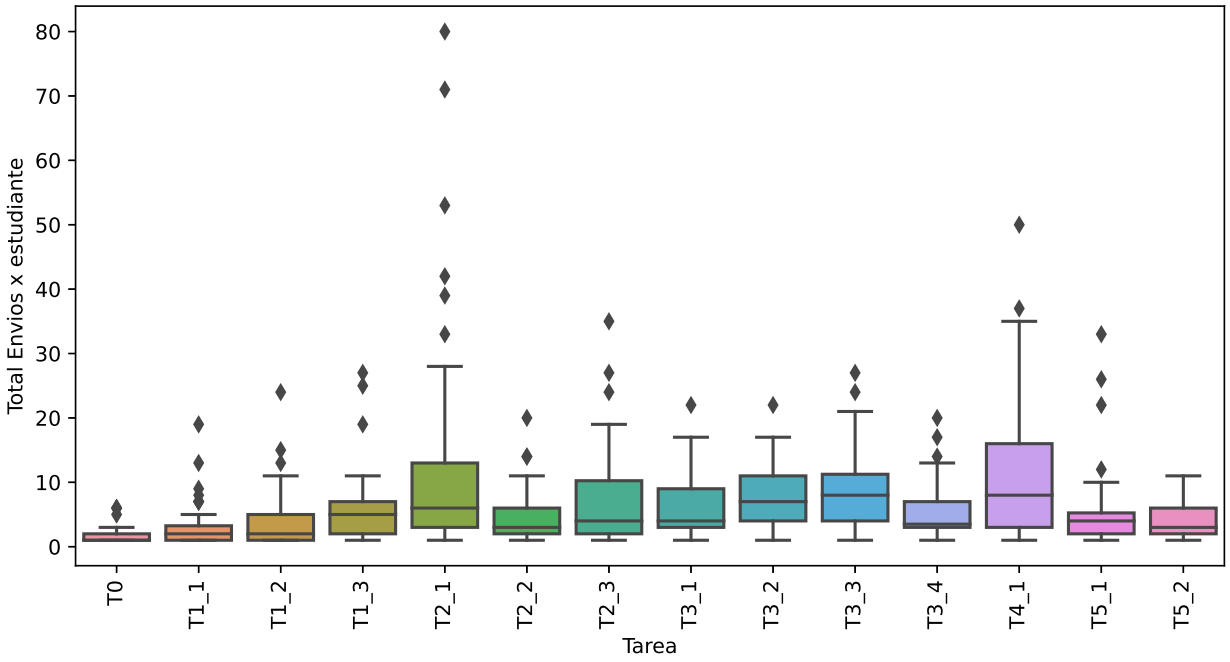


Figura 6-3.: Diagrama de cajas del número de envíos por estudiante en cada tarea

6.1.2. Análisis de las percepciones de los estudiantes

De los 57 estudiantes que utilizaron la herramienta por lo menos una vez, 48 respondieron al cuestionario (84.21 %). Como se muestra en la Figura **6-4**, la distribución de los programas de pregrado, de los que hacen parte los participantes que dieron sus opiniones, corresponde a 18 estudiantes pertenecientes a Ingeniería Electrónica (37.50 %), 17 a Ingeniería Eléctrica (35.42 %), 11 a Ingeniería Mecatrónica (22.92 %) y 2 a Ingeniería Mecánica (4.17 %).

La Figura **6-5** muestra en forma de barras apiladas, las respuestas de los estudiantes sobre los niveles de acuerdo/desacuerdo en escala de Likert para las 5 afirmaciones, que se enuncian en la Tabla **6-1**, de la encuesta anteriormente descrita.

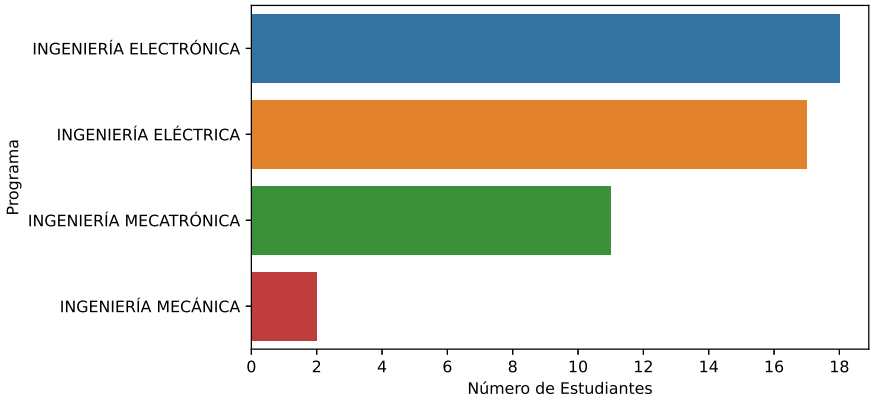


Figura 6-4.: Número de estudiantes participantes por programa

Tabla 6-1.: Afirmaciones en el cuestionario

#	**Afirmación**
1	La calificación automática que me ofrecía la herramienta UNCode durante los laboratorios me fue útil en el aprendizaje del Lenguaje de Descripción de Hardware Verilog
2	La retroalimentación automática usando DIAGRAMA DE TIEMPOS que me ofrecía la herramienta UNCode durante los laboratorios me ayudó a identificar los errores que tenían mis diseños y a analizar cómo corregirlos
3	La retroalimentación automática usando TEXTO RESALTADO CON COLORES que me ofrecía la herramienta UNCode durante los laboratorios me ayudó a identificar los errores que tenían mis diseños y a analizar cómo corregirlos
4	Considero que la disponibilidad online de la herramienta UNCode es una ventaja importante
5	UNCode fue útil en mi proceso de aprendizaje del Lenguaje de Descripción de Hardware Verilog

De los 48 estudiantes que respondieron el cuestionario la mayoría muestran un grado de nivel de acuerdo en todas las afirmaciones. Para la cuarta afirmación, respecto a la disponibilidad en línea como una ventaja, el 100 % de los encuestados están de acuerdo (70.8 % Totalmente de acuerdo y 29.2 % De acuerdo). La primera y la quinta afirmación, relacionada con la calificación automática y la utilidad en el proceso de aprendizaje, también presentan una aprobación generalizada del 95.8 %. El 91.7 % aprueba la realimentación usando diagramas de tiempo (afirmación 2), mientras que el 89.6 % está de acuerdo con la realimentación usando texto resaltado por colores (afirmación 3). Por otro lado, solo algunos estudiantes expresaron algún nivel de desacuerdo, correspondiente al 4.2 %, 8.3 %, 10.4 % y 4.2 %, respectivamente para las afirmaciones 1, 2, 3 y 5.

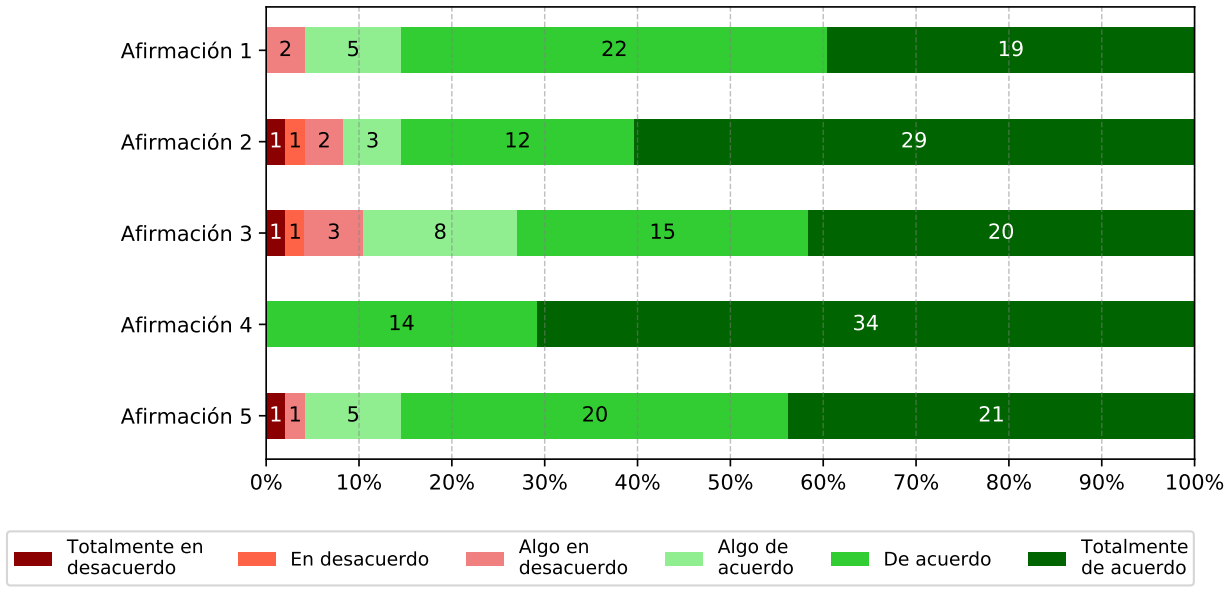


Figura 6-5.: Nivel de acuerdo o desacuerdo expresado por los estudiantes

En cuanto al análisis cualitativo a las preguntas de respuesta abierta, se utilizó el procedimiento descrito en la Sección 5.3.2. Se obtuvieron 5 temáticas, organizadas respecto al mayor número de categorías corresponden a: la percepción de la utilidad de la herramienta en el proceso de aprendizaje, la percepción de la realimentación, la percepción de las características de la herramienta, la percepción de la calificación automática y la percepción de las mejoras que se podrían realizar. Para cada una de las temáticas se presentará a continuación una tabla que incluye las categorías agrupadas en dicha temáticas, su descripción, los indicadores que la componen, una cita de ejemplo de las opiniones de los estudiantes y el número de opiniones que se registraron en cada indicador.

Percepción de la utilidad de la herramienta en el proceso de aprendizaje

La temática donde se obtuvieron más categorías fue las percepciones de la utilidad de la herramienta dentro del proceso de aprendizaje. La Tabla **6-2** se presentan las categorías encontradas. Se destaca la metodología usada durante los laboratorios, el apoyo al aprendizaje, que la herramienta está enfocada al aprendizaje, las condiciones en las cuales fue utilizada (durante los primeros meses de la pandemia ocasionada por el virus SARS-CoV-2) y opiniones adicionales que señalan la utilidad de la herramienta. Por otro lado, también se encontraron comentarios (3) que enuncian que la herramienta no contribuyó en el proceso de aprendizaje.

Tabla 6-2: Resultado análisis de la utilidad de la herramienta en el proceso de aprendizaje

Categoría	Descripción	Indicador	Cita de ejemplo	# de opiniones
Metodología del laboratorio	Los estudiantes expresaron que la metodología de los laboratorios fue favorable al proceso de aprendizaje dado el enfoque y las herramientas utilizadas (Vídeos + UNCode-Digital)	Metodología usada	*"me gustó el seguimiento de las los laboratorio virtuales, se explicaba el tema y teniendo claro los conceptos era posible iniciar a programar los ejercicios prouestos"*[sic]	11
		Enfoque a principiantes	*"Si fue útil, ya que permite obtener una orientacion adecuada del trabajo realizado, mejor aun para los que somos principiantes en el manejo de verilog."*[sic]	4
		Herramienta complementaria	*"Ayuda a ver los errores, es una herramienta complementaria practica para probar si uno entiende los vídeos guía del docente"*	4
		Videos de la teoría	*"Más que UnCode los videos del profesor, sin los videos UnCode no sería útil e inútil me sentiría yo también. Pero los videos más UnCode son una combinación excelente."*[sic]	4
Apoyo al aprendizaje	Los estudiantes sienten que la herramienta es un apoyo y que mejoran con el uso	Mejorar con el uso	*"Porque la autocalificacion lograba revisar el programa y encontrar mis errores, y así poder corregirlos e ir aprendiendo"*[sic]	20
		Considera fue útil	*"Es una herramienta que sirve de gran apoyo en el aprendizaje."*	5
Enfoque al aprendizaje	La herramienta tiene un enfoque hacía el aprendizaje de forma didáctica, constante y autónoma	Herramienta Didáctica	*"Porque es muy didáctica de manejar, además da flexibilidad para uno corregir los errores, hasta obtener respuesta de lo que se requiere."*	15
		Práctica constante con realimentación	*"fue importante porque solo practicando lograba ver que los conceptos me quedaran claros, y en UNCode identificaba mejor mis falencias y me ayudaba a corregirlas."*	6
		Autonomía	*"Es una herramienta bastante útil, porque nos ayuda a practicar y entender la codificación en verilog, además de poder encontrar errores y mejorar de manera autónoma."*	5

Continúa en la siguiente página

Tabla 6-2: Resultado análisis de la utilidad de la herramienta en el proceso de aprendizaje (Continuación)

Categoría	Descripción	Indicador	Cita de ejemplo	# de opiniones
Condiciones externas	Los laboratorios fueron desarrollados durante un evento mundial inesperado	Pandemia (COVID-19)	*"Puede ser usada en cualquier parte, un claro ejemplo es la coyuntura por la que estamos pasando."*	10
Opiniones adicionales	Esta categoría agrupa opiniones adicionales de los estudiantes sobre la herramienta	Herramienta completa	*"En comparación con otros software que he usado para diferentes lenguajes de programación, me parece una herramienta muy completa. Que permita varios intentos y que dé la retroalimentación es muy útil."*	3
		Superior a otras herramientas	*"...poder identificar los errores para hacer las respectivas correcciones. Con otros simuladores sería mucho más tedioso y quizá no se obtendría el mismo aprendizaje."*	3
		Familiarización con el lenguaje	*"Los primeros laboratorios que se desarrollaron con UNCode permitieron una correcta familiarización con verilog"*	2
Poco útil	Algunos estudiantes creen que la herramienta no fue útil durante el proceso de aprendizaje	Necesidad de recursos adicionales	*"Porque creo que el proceso de aprendizaje es ajeno a esta plataforma, en mi caso, he aprendido Verilog gracias a vídeos, lecturas y programación de distintos ejercicios en otras plataformas"*	2
		Proceso previo	*"Ya tenia un proceso de aprendizaje desarrollado"*	1

Percepción de la realimentación

En la Tabla **6-3** se presenta la temática de las percepciones de la realimentación. Encontramos 3 categorías positivas, las cuales se refieren a la utilidad de la realimentación para encontrar errores, la utilidad para saber qué se debe modificar para corregir errores y otras ventajas asociadas a la realimentación como lo son la facilidad, rapidez, claridad y la posibilidad de verificar diseños mediante la realimentación inmediata. Por otra parte, los estudiantes señalaron algunos desafíos con las realimentaciones que la herramienta proveía, como la dificultad para identificar los errores y el uso de solo uno de los tipos de realimentación.

Tabla 6-3: Resultado análisis de Percepción de la realimentación

Categoría	Descripción	Indicador	Cita de ejemplo	# de opiniones
Utilidad para encontrar errores	La herramienta ayudó a identificar tanto errores generales como específicos con la ayuda de la comparación	Identificar errores	*"Permite identificar en qué parte el código genera un error en cuanto a lo que se espera"*	64
		Contraste con salida esperada	*"Facilitaba la visualización de las entradas y salidas esperadas. Lo cual ahorra bastante tiempo en la verificación del funcionamiento."*	30
Utilidad para modificar el código	La herramienta contribuyó a modificar el código para conseguir el resultado deseado	Ayuda para corregir errores	*"Ayuda para corregir errores & Obtener una visualización de los resultados de forma didáctica y visual nos permite a los estudiantes poder relacionar el código que estamos ejecutando con el error cometido y se puede corregir e identificar de forma sencilla."*	33
Ventajas que da la realimentación	Adicional a la identificación y corrección de errores, los estudiantes creen que la realimentación da algunas ventajas durante el proceso de aprendizaje	Facilidad en la identificación de errores	*"Permitía detectar fácilmente la salida errónea, y con ello se permite corregir el problema que provoca dicho error."*[sic]	22
		Rapidez en la identificación de errores	*"Es util para contrastar con el resultado esperado y ver rapidamente en que parte del código está el error."*	15
		Claridad en la identificación de errores	*"Porque permite observar claramente que variable o variables, no se están comportando como deberían, según el diseño planteado."*	5
		Verificación con realimentación	*"Permite práctica la programación de una forma mucho más dinámica y eficaz. Ayuda a reafirmar el conocimiento adquirido en clase y la retroalimentación es mucho más rápido lo que permite tener una continuidad con el ejercicio que se está realizando."*	10

Continúa en la siguiente página

Tabla 6-3: Resultado análisis de Percepción de la realimentación (Continuación)

Categoría	Descripción	Indicador	Cita de ejemplo	# de opiniones
Inconvenientes con la realimentación	Los estudiantes enuncian algunos desafíos que tuvieron con la herramienta, como la falta de claridad en la realimentación y el uso de solo una realimentación	Dificultad en identificar errores	*"Si bien ayuda en algo en la identificación de los errores, creo que no es lo suficientemente claro."*	22
		Uso de un solo tipo de realimentación	*"No uso el diagrama de tiempos sino la tabla generada"*, *"La verdad no me fijé tanto en la retroalimentación con colores."*	3

Percepción de las características de la herramienta

La tercera temática corresponde a las percepciones de las características de la herramienta. Las categorías incluidas en esta temática son presentadas en la Tabla **6-4**. Se observa que los estudiantes tienen una buena percepción de la experiencia del usuario, destacando la facilidad y la velocidad de la compilación y verificación de la solución. Por otro lado, en las categorías obtenidas en esta temática se resalta el aspecto visual, el editor de texto y la realimentación por texto y por diagrama de tiempo. Opciones dentro de UNCode para ver el historial y poder realizar varios intentos también fueron resaltadas. Por último, se subrayan las propiedades al ser una herramienta online.

Tabla 6-4: Resultado análisis de Percepción de las características de la herramienta

Categoría	Descripción	Indicador	Cita de ejemplo	# de opiniones
Experiencia del usuario	Dentro de las características de la herramienta, los estudiantes destacan la interacción que tuvieron con la herramienta	Facilidad de uso	*"Su fácil manejo, y el ambiente amigable (aspecto visual) que tiene"*	15
		Velocidad de la herramienta	*"Es intuituva, fácil de usar, es bastante rápida en la compilación y la verificación de las respuestas y el diagrama de tiempos es algo nuevo y muy útil."*[sic]	6

Continúa en la siguiente página

Tabla 6-4: Resultado análisis de Percepción de las características de la herramienta (Continuación)

Categoría	Descripción	Indicador	Cita de ejemplo	# de opiniones
Características de interfaz de usuario	Los estudiantes destacaron características de los elementos visuales de la herramienta	Aspecto Visual	*"Tiene una buena interfaz, es una herramienta fácil e intuitiva de usar."*	6
		Editor de texto	*"la apariencia que se le da al texto (colores e identación) son apropiados para comenzar a familiarizarse con el lenguaje y la sintaxis."*	6
		Realimentación con gráfica de diagrama de tiempo	*"Tener una guía gráfica de lo que tiene que dar contra lo que a uno le está dando facilita la corrección de errores."*	11
		Diferencia de Salida en Texto	*"Es incluso mejor para encontrar los errores que ver el diagrama, pues ademas de ver en que variable está el error, permite saber en que parte de la tabla de verdad se encuentra."*[sic]	5
Opciones dentro de la herramienta	Esta categoría agrupa las opciones que provee UNCode y que los estudiantes encontraron destacables	Varios intentos	*"La facilidad de lectura en la funcionalidad de las respuestas pedidas y el hecho de permitir ilimitados intentos, resultan muy útiles en la verificación."*	6
		Revisión del Avance	*"Permite revisar codigos anteriores en cualquier momento apoyándose en ellos para realizar alguno nuevo."*[sic]	6
Propiedades de herramienta online	Al ser una herramienta en línea los estudiantes enfatizaron en las propiedades que encontraron más útiles	Acceso	*"No todos tenemos la posibilidad descargar los software de ISE, me parece muy buena esta posibilidad de aprender por internet"*	15
		Sin descargas	*"Me parece buena herramienta, debido a que es en línea y no se necesita descargar ningún tipo de archivos y así poder trabajar desde cualquier equipo"*	11
		Disponibilidad	*"porque aveces salen cosas que debemos realizar a lo largo del día, y que se pueda utilizar en la noche o madrugada ayuda mucho para las personas que tenemos que trabajar y estudiar, y demás tareas en la vida cotidiana."*[sic]	7

Percepción de la calificación automática

La Tabla **6-5** muestra el resumen de las categorías e indicadores encontrados al analizar las respuestas de los estudiantes que componen la temática de percepciones de la calificación automática dentro de la herramienta UNCode-Digital. En este caso tenemos dos categorías. Por un lado, se puede observar que los estudiantes encuentran utilidad en la calificación automática, principalmente en aspectos relacionados con la inmediatez de la calificación y la posibilidad de calificar correctamente múltiples diseños que cumplieran con los requerimientos de la tarea. Por otra parte, también se mencionan algunas falencias de la calificación automática, como que no se consideraron calificaciones intermedias de los ejercicios a soluciones parcialmente correctas.

Tabla 6-5: Resultado análisis de Percepción de la calificación automática

Categoría	Descripción	Indicador	Cita de ejemplo	# de opiniones
Utilidad de la calificación automática	Los estudiantes encontraron útil en el proceso de aprendizaje el obtener una calificación sin depender de los tiempos del profesor ni de una solución específica	Calificación Inmediata	*"La herramienta de corrección automática de UNCode me fue útil porque así podía verificar que el código respondía a la solución del problema que se me presentara."*	24
		Interpretar múltiples soluciones	*"Me sorprendió que la plataforma interpretaba las múltiples soluciones que se le pueden dar a un mismo ejercicio"*	6
Falencias de la calificación Automática	Los estudiantes creen que hay aspectos por mejorar en la calificación que ofrece la herramienta	No hay calificación intermedia	*"Solo calificaba 0 % y 100 % pero nunca observe que me marcara 25 %, 50 % o 80 %"*	7

Percepción de las mejoras que se podrían realizar

Finalmente, la Tabla **6-6** presenta las sugerencias respecto a las percepciones de los estudiantes de las mejoras que pueden existir. Existen algunos estudiantes que enuncian que no cambiarían nada, pero hay otros que sugieren mejoras. Las más destacadas son: la modificación en la manera que se presentan los errores, incluir documentación dentro de la herramienta, integrar los errores al código para señalar el punto exacto donde está la falla, incluir el manejo de archivos en lugar de un solo editor de texto, la integración con hardware físico, mayor número de ejemplos, potenciar el uso en

otras asignaturas y tener un mayor número de ejercicios dentro de la herramienta.

Tabla 6-6: Resultado análisis de Percepción de las mejoras que se podrían realizar

Categoría	Descripción	Indicador	Cita de ejemplo	# de opiniones
Ideas de mejora	Algunos estudiantes consideran que se puede mejorar: la forma en que se da realimentación, integrar documentación, la inclusión de hardware, extender los ejercicios y los ejemplos, entre otros.	Modificar la presentación de errores	*"Una descripción mas detallada de los errores para aquellas personas que hasta ahora se encuentran en contacto con las misma"*	14
		Documentación dentro de UNCode	*"Añadir información sobre conceptos básicos para iniciar a programar."*	9
		Integrar error con el código	*"Sí porque permitía ver una comparación entre la tabla de verdad esperada y la entregada por el código propuesto, aunque no permitía ver en que línea de código podría estar directamente el error."*	4
		Manejo de múltiples archivos	*"manejar más de un archivo para poder trabajar más de un modulo sin saturar uno solo"*	2
		Integración con FPGA	*"Se podría proyectar a futuro montar una plataforma de simulación en tiempo real como la que utilizamos en el laboratorio (LabsLand) Donde se pudiera interactuar con la FPGA en tiempo real. Sin embargo, esto seria una proyección; porque como tal la herramienta para aprender a programar es excelente"*	2
		Ejemplos adicionales	*"de pronto mostrar un ejemplo parecido al ejercicio en cada laboratorio dentro de la herramienta."*	2
		Extensión	*"si, por las ventajas ya nombradas, seria bueno que se amplie a otros leguajes y a mas materias, es muy didactica."*[sic]	2
		Mayor número de ejercicios	*"Añadir más ejercicios que sean opcionales, a modo de estudio"*	2

Continúa en la siguiente página

Tabla 6-6: Resultado análisis de Percepción de las mejoras que se podrían realizar (Continuación)

Categoría	Descripción	Indicador	Cita de ejemplo	# de opiniones
Sin cambios	Esta categoría agrupa las opiniones de los estudiantes que creen que no hay necesidad de cambios en la herramienta	Nada que agregar	*"Todo está abierto a mejoras, pero no se me ocurre alguna."*	9

6.2. Discusión

Con el fin de conocer las percepciones de los estudiantes de un curso de electrónica digital al usar un prototipo de una herramienta de software de calificación automática con proceso de realimentación, se diseñó e implementó un prototipo llamado UNCode-Digital, que fue utilizado en el desarrollo de los laboratorios de la asignatura Electrónica Digital durante el período 2020-I en la Universidad Nacional de Colombia. Posteriormente, los estudiantes respondieron un cuestionario sobre las características y la utilidad de la herramienta. Esta experiencia nos permitió responder a la pregunta de investigación: *¿Cuáles son las percepciones de los estudiantes en cuanto al proceso de aprendizaje de HDLs al usar una herramienta de software de calificación automática de apoyo?*.

Teniendo en cuenta los resultados de las interacciones de los estudiantes con las tareas, se observa un uso constante desde la segunda tarea. Adicionalmente, se observó que algunas tareas tenían más envíos que otras, así como que algunos estudiantes realizaron un número considerable de envíos. Esta información está al alcance del profesor que está usando la herramienta y podría ayudar en el proceso de formación al poder identificar los temas donde los estudiantes necesitan más ayuda o refuerzo. Al tener el historial de todos los envíos de los estudiantes, el análisis podría ser más profundo. Por ejemplo, en la Figura **6-2** se puede observar que en la tarea 4, dos estudiantes realizaron más envíos que los demás, con lo que el profesor podría proveer una realimentación adicional en estos casos.

En el análisis cualitativo de las respuestas del cuestionario se obtuvieron 5 temáticas las cuales ayudan a comprender mejor las percepciones de los estudiantes después de usar la herramienta. Adicionalmente, el análisis cuantitativo ofrece un soporte de los hallazgos.

La temática con más categorías fue la percepción de la utilidad de la herramienta en el proceso de aprendizaje. Como indicador global de toda la experiencia, en el análisis cuantitativo se detectó que más del 95 % de los estudiantes estuvo de acuerdo en señalar que la herramienta fue útil dentro

del proceso de aprendizaje de Verilog. Es importante mencionar, que los estudiantes destacaron la metodología utilizada durante los laboratorios y que estos fueron realizados durante la pandemia del COVID-19 de forma remota sincrónica. Según las percepciones de los estudiantes, fue una herramienta que sirvió de apoyo al aprendizaje y que estaba enfocado a este, pues permite mejorar con el uso de una forma, didáctica, constante y autónoma.

Por otra parte, también se encontró una temática relacionada con la percepción de la realimentación. En esta se pudo encontrar que la herramienta, no solo permitió encontrar y corregir el código para obtener la solución, sino que un número considerable de estudiantes cree que este proceso fue fácil, rápido y claro. Las respuestas de los estudiantes muestran una fuerte tendencia hacia estar de acuerdo con la utilidad de la herramienta para encontrar los errores cometidos, en contraste a lo expuesto por Nutter et al. (2014) durante su experiencia en la Universidad de Manchester, pues para ellos los resultados fueron parcialmente positivos. Cabe aclarar que la herramienta desarrollada en dicha universidad, solo envía la realimentación una vez por laboratorio, los ejercicios incluyen mayor complejidad y también ponen a prueba los testbench de los estudiantes. Con los comentarios de los estudiantes de la herramienta presentada en este trabajo, se podría sugerir que proveer realimentación varias veces por ejercicio supone una ventaja en el proceso de aprendizaje. Estos comentarios están acorde con que más del 90 % encontró alguno de los dos tipos de realimentación útil para identificar los errores y analizar cómo corregirlos. Sin embargo, para algunos estudiantes la realimentación no fue tan clara como les gustaría.

Así mismo, el análisis arrojó la temática de la percepción de las características de la herramienta. Algunos estudiantes destacaron que la experiencia del usuario, dada la facilidad y la velocidad al usar la herramienta. Adicionalmente, también resaltan el aspecto visual, los dos tipos de realimentación, la posibilidad de realizar varios intentos y poder realizar la revisión del avance en el curso. Cabe destacar que todos los estudiantes estuvieron de acuerdo en señalar que la disponibilidad en línea es una ventaja importante de la herramienta, ya que permitía acceso desde cualquier lugar, la ventaja de no tener que descargar software adicional y la disponibilidad en cualquier momento del día. Estos comentarios son congruentes con lo encontrado por Baneres and Saiz (2016), quienes observaron que una debilidad de su herramienta fue que algunos estudiantes tuvieron dificultad con la instalación de la misma y por lo tanto no pudieron utilizarla.

La cuarta temática identificada fue la percepción de la calificación automática. Se puede observar que al no depender de una solución específica ni esperar a la calificación de un profesor, esta calificación es útil dentro del proceso de aprendizaje. Los estudiantes destacaron que la herramienta no esperaba una solución en específico, sino que las posibles soluciones eran evaluadas correctamente. Adicionalmente, se encuentra utilidad en tener la calificación de forma automática. Lo que conlleva a que más del 95 % de los estudiantes encuestados creen que la calificación automática de la herramienta fue útil de alguna forma en el aprendizaje de los HDL. A este mismo resultado se llegó en el estudio de Baneres and Saiz (2016) que a pesar de no estar enfocado a los lenguajes de descripción de hardware,

si está enfocado a la enseñanza de circuitos lógicos. Por lo que se observa que la evaluación automática ayuda en el proceso de aprendizaje de diseño digital.

Finalmente, la última temática fue la percepción de las mejoras que se podrían realizar. Algunos estudiantes no encontraron nada que agregar, mientras que otros hicieron recomendaciones. Se puede observar que algunas de las sugerencias tienen que ver con modificar elementos dentro de la herramienta, mientras que otras consisten en adicionar características. Al igual que en la herramienta de Baneres and Saiz (2016) y de Nutter et al. (2014), los estudiantes sugieren dar mayor claridad en la realimentación, en el caso de estas herramientas estas presentan un contraejemplo en el caso que la solución no cumpla con todos los requisitos. En ese orden de ideas, los autores sugieren que se debe continuar mejorando en la correcta forma de realimentar a los estudiantes, lo cual también aplica en menor medida para UNCode-Digital según las percepciones de los estudiantes. Por otro lado, los estudiantes solicitan tener un mayor número de ejemplos y de ejercicios, lo cual también fue identificado por Baneres and Saiz (2016). Por último, al igual que se sugiere en Nutter et al. (2014) se deben seguir recolectando datos y mejorando la herramienta para aumentar la satisfacción de los estudiantes.

Por todo lo anterior, podemos decir que UNCode-Digital adicionalmente a proveer una realimentación sumativa, ayuda al estudiante mediante una realimentación formativa a identificar y corregir errores durante el proceso de aprendizaje de lenguajes de descripción de hardware.

7. Conclusiones y Trabajo Futuro

7.1. Conclusiones Generales

Este trabajo final presentó el desarrollo de la herramienta de software UNCode-Digital, la cual permite realizar calificación automática con proceso de realimentación durante el proceso de aprendizaje de Lenguajes de Descripción de Hardware. Como se enunció en el Capítulo 3, para el diseño de la herramienta se utilizó la estrategia de desarrollo orientada a la creación de prototipos. A partir ahí se implementó un prototipo exploratorio que sirvió de base del prototipo evolutivo, los cuales fueron presentados en el Capítulo 4. El primer prototipo sirvió de base para implementar el segundo, llamado UNCode-Digital. Como fue presentado en el Capítulo 5, teniendo esta implementación se procedió a diseñar un estudio para obtener las percepciones de los estudiantes al utilizar la herramienta. El estudio fue realizado durante el primer semestre del año 2020 con estudiantes de la asignatura Electrónica Digital de la Universidad Nacional de Colombia. Los estudiantes utilizaron la herramienta por 5 semanas durante el desarrollo de las prácticas de laboratorio. Debido a la pandemia del COVID-19, estas prácticas fueron realizadas de manera remota sincrónica. Finalizadas las actividades con la herramienta, los estudiantes respondieron un cuestionario que buscaba obtener las percepciones que tuvieron con respecto a la utilidad de la herramienta en el proceso de aprendizaje. De esta forma, fue posible tener datos cuantitativos y cualitativos relacionados con las percepciones de los estudiantes, cuyo análisis fue presentado en el Capítulo 6.

En general, los resultados del estudio evidenciaron percepciones favorables por parte de los estudiantes acerca de la utilidad de la herramienta para aprender el lenguaje de descripción de hardware Verilog. Acorde con los resultados del estudio, los participantes señalan que las características elegidas para ser incluidas en la herramienta, presentadas en el Capítulo 2, fueron integradas de manera satisfactoria. Los datos cuantitativos muestran una tendencia positiva hacia la utilidad de la herramienta en el proceso de aprendizaje de los lenguajes de descripción de hardware. Adicionalmente, el análisis cualitativo permitió conocer las percepciones de los estudiantes al usar la herramienta. Entre estas percepciones los estudiantes destacaron la utilidad de la calificación automática, la realimentación, las características de la herramienta y el soporte al proceso de aprendizaje. Se encontraron similitudes y diferencias con otros estudios. Al tener una herramienta en línea de la cual se obtiene tanto realimentación sumativa como formativa durante el proceso de aprendizaje de los HDL, los estudiantes obtienen una ventaja importante en su proceso formativo. Con el apoyo de la herramienta, los estudiantes lograron no sólo identificar los errores sino corregirlos. Igualmente, esta herramienta ayudó a dar una realimentación más rápida por parte del docente que de otra forma no hubiera sido

posible; permitiendo a cada estudiante aprender de los errores cometidos de manera inmediata. Así mismo, los estudiantes también destacaron que poder observar el diagrama de tiempos fue útil en el proceso de aprendizaje. Además, la disponibilidad en línea fue la característica más destacada. De igual forma, otras características como el editor en línea con resaltado de sintaxis y la interfaz gráfica también tuvieron aceptación por parte de los estudiantes. Por otra parte, fue resaltada la metodología utilizada durante las cinco semanas del estudio. Dado que este estudio fue desarrollado durante la pandemia del COVID-19, los estudiantes remarcaron que su uso ayudó a mitigar los desafíos que se presentaron en la enseñanza.

7.2. Contribución

Este trabajo hace dos contribuciones principales al área de la enseñanza de los lenguajes de descripción de hardware y la ingeniería: en primer lugar, introducimos una herramienta de calificación de HDLs (Verilog y VHDL) la cual se encuentra disponible públicamente por medio de UNCode, que proporciona a los docentes un calificador automático que ofrece realimentación sumativa y formativa a los estudiantes de forma inmediata; y en segundo lugar, proporcionamos evidencia empírica sobre los beneficios del uso de la herramienta propuesta en la academia a través del estudio reportado sobre el uso de la herramienta en cuatro de los laboratorios de la asignatura electrónica digital, donde los estudiantes expresaron lo útil que fue la herramienta para su proceso de aprendizaje.
Como resultado de este trabajo se presenta el siguiente producto académico:

- Extensión Software UNCode[1] - Como prototipo de la herramienta de software se extendió la capacidad que tenía UNCode dando soporte adicional a los lenguajes de descripción de hardware Verilog y VHDL. Con esta adición, los profesores de electrónica digital tienen una herramienta adicional que los estudiantes encontraron útil durante el proceso de aprendizaje en 2020.

7.3. Trabajo Futuro

En cuanto a cambios en la herramienta, el análisis de los datos recogidos sugiere que el prototipo podría tener algunas mejoras. Por ejemplo, la modificación de la realimentación para agregar maneras de encontrar errores más fácil y clara. Por otro lado, el objetivo de un diseño digital es no solo simularlo sino sintetizarlo en algún dispositivo físico, por lo que la inclusión de formas de probar el código en hardware, podría aumentar la utilidad de la herramienta en el proceso de aprendizaje. Adicionalmente, la herramienta podría tener más ejercicios con los que los estudiantes puedan probar. En cuanto al prototipo realizado, si bien la comparación utilizando testbench funciona para circuitos pequeños, si se quiere realizar en proyectos más grandes, se vuelve complicado probar todas las posibles combinaciones en las entradas, por lo que se podrían explorar otras formas de realizar la comparación.

[1] https://github.com/JuezUN/INGInious/wiki/How-to-create-a-task#hdl-tasks-verilogvhdl

Debido al alcance de este trabajo, los resultados encontrados no pueden ser generalizados y se hace necesario realizar estudios experimentales futuros sobre no solo de las percepciones sino del rendimiento académico, la participación o la motivación. Al igual, es necesario socializar el uso de esta herramienta con otros docentes, para aumentar los estudios de caso, las investigaciones y los datos almacenados.

A. Consentimiento informado del cuestionario de percepciones

El grupo de investigación PLaS del departamento de Ingeniería de Sistemas e Industrial de la Universidad Nacional de Colombia está llevando a cabo una investigación sobre el uso de herramientas de software en la educación. El propósito particular de esta investigación es conocer las percepciones que tienen los estudiantes sobre el uso de una herramienta de software de calificación automática con proceso de realimentación para el apoyo al aprendizaje de Lenguajes de Descripción de Hardware. Este estudio está enfocado en los estudiantes de Electrónica Digital I de la Universidad Nacional de Colombia.

Si tiene alguna inquietud sobre este estudio, contacte al Investigador Principal: Andrés Francisco José Corso Pinzón, acorso@unal.edu.co

La información que usted proporcionará será tratada de manera confidencial y anónima y en ningún momento las respuestas serán presentadas de forma que usted pueda ser identificable.

Completar la encuesta le llevará cerca de 30 minutos. La encuesta consta de preguntas sobre su experiencia utilizando la plataforma UNCode durante los laboratorios de la asignatura Electrónica Digital I en el período 2020-1.

Debido a que no hay respuestas correctas ni incorrectas, le pedimos responder esta encuesta sin ningún tipo de prevención y ser tan preciso como le sea posible. La información que nos brinde será tratada con total confidencialidad, no es calificable, ni influirá en ningún proceso académico que adelante en la Universidad. La información personal será borrada al finalizar esta investigación.

Agradecemos su colaboración y su tiempo ya que los datos aquí recogidos serán de vital importancia en el estudio en curso.

Bibliografía

ACM and IEEE-CS (2016). *Computer Engineering Curricula 2016*. IEEE.

Baneres, D., Clariso, R., Jorba, J., and Serra, M. (2014). Experiences in digital circuit design courses: A self-study platform for learning support. *IEEE Transactions on Learning Technologies*, 7(4):360–374.

Baneres, D. and Saiz, J. (2016). Intelligent Tutoring System for Learning Digital Systems on MOOC Environments. *Proceedings - 2016 10th International Conference on Complex, Intelligent, and Software Intensive Systems, CISIS 2016*, pages 95–102.

Bryman, A. (2012). *Social Research Methods*. Oxford University Press, 4th edition edition.

Burch, C. (2011). Logisim.

Carroll, C. R. (2011). Teaching Ground-Floor Digital Circuits to Pre-Engineering Students. *ASEE Annual Conference and Exposition, Conference Proceedings*.

Chapyzhenka, A. and Probell, J. (2016). WaveDrom Rendering Beautiful Waveforms from Plain Text. *SNUG 2016*, pages 1–42.

Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. (2002). NuSMV 2: An opensource tool for symbolic model checking. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2404:359–364.

Cirka, L. and Kaluz, M. (2017). A web-based tool for design of Simulink models. *Proceedings of the 2017 21st International Conference on Process Control, PC 2017*, pages 92–97.

Corsini, P. and Rizzo, L. (1991). SSCSSC: A tool for the Teaching of Digital Circuits. *IEEE Transactions on Education*.

Fujita, M., Ghosh, I., and Prasad, M. (2008). *VERIFICATION TECHNIQUES FOR SYSTEM-LEVEL DESIGN*. Elsevier.

Georgakopoulos, D. and Jayaraman, P. P. (2016). Internet of things: from internet scale sensing to smart services. *Computing*, 98(10):1041–1058.

Greenwood, G. W. (2013). Teaching Hardware Description Languages to Satisfy Industry Expectations. *International Journal of Electrical Engineering & Education*, 46(3):239–247.

Jansen, D. and Dusch, B. (2014). Every student makes his own microprocessor. *10th European Workshop on Microelectronics Education, EWME 2014*, pages 97–101.

Jutman, A., Ubar, R., Hahanov, V., and Skvortsova, O. (2002). Practical works for on-line teaching design and test of digital circuits. *Proceedings of the IEEE International Conference on Electronics, Circuits, and Systems*, 3:1223–1226.

Kusairi, S. (2019). A web-based formative feedback system development by utilizing isomorphic multiple choice items to support physics teaching and learning. *Journal of Technology and Science Education*, 10:117–126.

La Meres, B. J. (2019). *Introduction to logic circuits & logic design with VHDL*. Springer Nature Switzerland, 2nd editio edition.

Madanayake, A., Wijenayake, C., Joshi, R. M., Grover, J., Carletta, J., Adams, J., Hartley, T., and Ogunfunmi, T. (2012). Teaching freshmen VHDL-based digital design. *ISCAS 2012 - 2012 IEEE International Symposium on Circuits and Systems*, pages 2701–2704.

Mateev, V., Manoilov, P., and Iliev, M. (2004). Tutoring tool for logical schemes design. *Proc. International Conference on Computer Systems and Technologies*, page 1.

Mepits (2014). Hardware description language. https://www.mepits.com/tutorial/143/vlsi/hardware-description-language. [Online; accessed 30-June-2019].

Muchlas and Novianta, M. A. (2016). An online lab for digital electronics course using information technology supports. *Proceedings - 2015 International Conference on Science in Information Technology: Big Data Spectrum for Future Information Economy, ICSITech 2015*, pages 299–302.

Nutter, P. W., Pavlidis, V. F., and Pepper, J. (2014). Efficient teaching of digital design with automated assessment and feedback. *10th European Workshop on Microelectronics Education, EWME 2014*, pages 203–207.

Pereira, M. C., Viera, P. V., Raabe, A. L. A., and Zeferino, C. A. (2012). A basic processor for teaching digital circuits and systems design with FPGA. *SPL 2012 - 8th Southern Programmable Logic Conference*.

Petit, J., Roura, S., Carmona, J., Cortadella, J., Duch, J., Giménez, O., Mani, A., Mas, J., Rodríguez-Carbonell, E., Rubio, E., De San Pedro, E., and Venkataramani, D. (2018). Jutge.org: Characteristics and Experiences. *IEEE Transactions on Learning Technologies*, 11(3):321–333.

Pomberger, G. and Bischofberger, W. (1992). *Prototyping-Oriented Software Development*. Springer-Verlag.

Pomberger, G., Bischofberger, W., Kolb, D., Pree, W., and Schlemm, H. (1998). Prototyping-oriented software development - concepts and tools. *Structured Programming*, 12.

Restrepo-Calle, F., Ramírez-Echeverry, J. J., and Gonzalez, F. A. (2018). Uncode: Interactive System for Learning and Automatic Evaluation of Computer Programming Skills. *EDULEARN18 Proceedings*, 1(July):6888–6898.

Robal, T. and Kalja, A. (2007). Applying e-environments in teaching the basics of digital logic. *Proceedings - MSE 2007: 2007 IEEE International Conference on Microelectronic Systems Education: Educating Systems Designers for the Global Economy and a Secure World*, pages 41–42.

Roy, G., Ghosh, D., Mandal, C., and Mitra, I. (2016). Aiding Teaching of Logic Design and Computer Organization through Dynamic Problem Generation and Automatic Checker Using COLDVL Tool. *Proceedings - IEEE 7th International Conference on Technology for Education, T4E 2015*, pages 15–22.

Stanisavljevic, Z., Pavlovic, V., Nikolic, B., and Djordjevic, J. (2013). SDLDS-system for digital logic design and simulation. *IEEE Transactions on Education*, 56(2):235–245.

Teepe, G. (2014). The growing importance of microelectronics from a foundry perspective. *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–1.

UNAL (2019). Ingeniería electrónica. https://ingenieria.bogota.unal.edu.co/es/formacion/pregrado/ingenieria-electronica.html. [Online; accessed 10-Dec-2019].

Université Catholique de Louvain (2014). Inginious. https://inginious.org/. [Online; accessed 16-June-2020].