

# UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

## Máster Universitario en Ingeniería Industrial

### PROYECTO Fin de Máster

|              |  |
|--------------|--|
| TÍTULO       | DESARROLLO DE LABORATORIO REMOTO DE LA<br>FPGA LIBRE ICEZUM ALHAMBRA   |
| AUTOR        | MARIA DESAMPARADOS HERNAIZ PEREZ   |
| DIRECTOR     | SERGIO MARTÍN GUTIÉRREZ  |
| DEPARTAMENTO | DEPARTAMENTO DE INGENIERÍA<br>ELÉCTRICA, ELECTRÓNICA, CONTROL,<br>TELEMÁTICA Y QUÍMICA APLICADA A LA<br>INGENIERÍA |

ESCUELA TÉCNICA SUPERIOR  
DE INGENIEROS INDUSTRIALES

# UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

DEPARTAMENTO    *DEPARTAMENTO DE INGENIERÍA ELÉCTRICA,  
ELECTRÓNICA, CONTROL, TELEMÁTICA Y  
QUÍMICA APLICADA A LA INGENIERÍA*

TÍTULO            DESARROLLO DE LABORATORIO REMOTO DE LA  
                      FPGA LIBRE ICEZUM ALHAMBRA  
AUTOR            MARIA DESAMPARADOS HERNAIZ PEREZ  
DIRECTOR        SERGIO MARTÍN GUTIÉRREZ

## ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES

*“El verdadero progreso es el que pone la tecnología al alcance de todos”*

*Henry Ford (1863-1947)*

## **RESUMEN**

---

La aplicación de las tecnologías de la información y la comunicación en la educación, desde hace varios años está promoviendo nuevas formas de aprendizaje. Cada vez son más los centros educativos que cuentan con este tipo de dispositivos.

A partir de mayo de 2015, el ingeniero austriaco Clifford Wolf hizo ingeniería inversa de la familia ICE40 de Lattice, creando el primer sintetizador de FPGAs libre de la historia. Este hito ha constituido una revolución digital en el campo de las FPGA. Desde 2015 se han desarrollado diferentes tarjetas entrenadoras que utilizan la FPGA Lattice iCE40, una de ellas es la IceZUM Alhambra, que cuenta con su IDE, Icestudio.

Con las FPGAs se tiene la posibilidad de programar y manipular diferentes tipos de elementos en la industria, por ello es interesante que los estudiantes de ingeniería industrial conozcan la capacidad que tienen estos dispositivos, programándolos y utilizándolos, y de paso conocer los lenguajes HDL, y la lógica programable.

En este proyecto se desarrolla un laboratorio remoto para poder manipular desde casa la FPGA IceZUM Alhambra proporcionando al estudiante de ingeniería industrial la posibilidad de aprender a programarla sin desplazarse, hecho que cobra importancia en la Educación a Distancia ya que estudiante y centro educativo no suelen coincidir en el espacio.

Durante el desarrollo de este trabajo se ha implementado una página web para controlar remotamente la FPGA libre IceZUM Alhambra, se ha diseñado un montaje con diferentes periféricos a controlar con lo FPGA, se han elaborado los materiales didácticos necesarios para la utilización del laboratorio remoto y por último se ha elaborado una memoria de prácticas en orden creciente de dificultad para realizar en laboratorio remoto

**ÍNDICE DE CONTENIDOS**

---

|  |           |
|--|-----------|
| <b>1. INTRODUCCIÓN.....</b>                                    | <b>9</b>  |
| 1.1. MOTIVACIÓN.....   | 9         |
| 1.2. OBJETIVOS .....   | 10        |
| 1.3. ESTRUCTURA DE LA MEMORIA .....                            | 10        |
| <b>2. ESTADO DEL ARTE.....</b>                                 | <b>11</b> |
| 2.1. LABORATORIOS REMOTOS.....                                 | 11        |
| 2.1.1. TIPOS DE LABORATORIOS .....                             | 11        |
| 2.1.2. EVOLUCIÓN DE LOS LABORATORIOS VIRTUALES Y REMOTOS ..... | 16        |
| 2.1.3. ARQUITECTURA DE UN LABORATORIO REMOTO .....             | 17        |
| 2.1.4. DISEÑO DE UN LABORATORIO REMOTO.....                    | 18        |
| 2.2. DESARROLLO WEB.....                                       | 22        |
| 2.2.1. PÁGINA WEB .....  | 22        |
| 2.2.2. HTML .....  | 22        |
| 2.2.3. LENGUAJE CSS .....                                      | 22        |
| 2.2.4. PHP .....   | 23        |
| 2.2.5. SQL Y MYSQL.....  | 24        |
| 2.2.6. SERVIDOR WEB.....                                       | 24        |
| 2.2.7. APACHE .....  | 24        |
| 2.2.8. ARQUITECTURA CLIENTE – SERVIDOR .....                   | 24        |
| 2.3. FPGAS .....   | 25        |
| 2.3.1. INTRODUCCIÓN AL HARDWARE RECONFIGURABLE.....            | 26        |
| 2.3.2. ESTADO ACTUAL DE LA TECNOLOGÍA FPGA.....                | 27        |
| 2.3.3. METODOLOGÍA DE TRABAJO CON HARDWARE RECONFIGURABLE..... | 28        |
| 2.3.4. ESTRUCTURA GENERAL DE LA FPGA.....                      | 29        |
| 2.3.5. DISEÑO BASADO EN COMPONENTES SIMPLES.....               | 30        |
| 2.3.6. DISEÑO BASADO EN MÓDULOS REUTILIZABLES .....            | 31        |
| 2.4. LENGUAJES HDL.....  | 31        |
| 2.4.1. HISTORIA DE LOS LENGUAJES HDL.....                      | 31        |
| 2.4.2. FUNDAMENTOS .....                                       | 32        |
| 2.4.3. ESTRUCTURADOS Y NO ESTRUCTURADOS .....                  | 33        |
| 2.4.4. TIPOS DE HDL.....                                       | 33        |
| 2.4.5. LENGUAJE VHDL .....                                     | 34        |
| 2.4.6. LENGUAJE VERILOG .....                                  | 34        |
| 2.4.7. COMPARACIÓN VHDL & VERILOG .....                        | 35        |
| 2.5. FPGAS LIBRES.....   | 37        |
| 2.5.1. INTRODUCCIÓN A LAS FPGAS LIBRES .....                   | 37        |
| 2.5.2. PROYECTO ICESTORM.....                                  | 38        |
| 2.5.3. FLUJO DE TRABAJO .....                                  | 40        |
| 2.5.4. PLACAS CON FPGAS LIBRES .....                           | 41        |
| 2.5.5. PLACA ICEZUM ALHAMBRA .....                             | 45        |
| 2.5.6. ENTORNO DE DESARROLLO ICESTUDIO .....                   | 48        |
| <b>3. ARQUITECTURA Y DESARROLLO DEL SISTEMA.....</b>           | <b>50</b> |
| 3.1. DEFINICIÓN DE LA METODOLOGÍA DE TRABAJO.....              | 50        |
| 3.2. ANÁLISIS DE ALTERNATIVAS .....                            | 51        |

|  |           |
|--|-----------|
| 3.3. DISEÑO ARQUITECTÓNICO .....   | 52        |
| 3.4. DISEÑO DE LABORATORIO REMOTO DE LA FPGA LIBRE ICEZUM ALHAMBRA ..... | 53        |
| 3.4.1. ANÁLISIS DE ALTERNATIVAS .....                                    | 53        |
| 3.4.2. IMPLEMENTACIÓN DEL LABORATORIO REMOTO .....                       | 55        |
| 3.4.3. CÁMARA IP .....   | 60        |
| 3.5. DESARROLLO DE UN ENTORNO WEB PARA CONTROL REMOTO .....              | 61        |
| 3.5.1. ANÁLISIS DE ALTERNATIVAS .....                                    | 61        |
| 3.5.2. PERFILES DE ACCESO .....  | 62        |
| 3.5.3. PLANIFICACIÓN .....   | 63        |
| 3.5.4. DIAGRAMA DEL PROCESO .....  | 63        |
| 3.5.5. ESTRUCTURA DE LA PROGRAMACIÓN .....                               | 65        |
| 3.5.6. ÁRBOL DE DIRECTORIOS .....  | 69        |
| 3.5.7. BASE DE DATOS .....   | 69        |
| 3.6. DESARROLLO DE MATERIALES DIDÁCTICOS .....                           | 72        |
| 3.6.1. ALTERNATIVAS DE APRENDIZAJE .....                                 | 72        |
| 3.6.2. PROPUESTA DE PRÁCTICAS A REALIZAR .....                           | 74        |
| 3.6.3. RECURSOS DE APOYO PEDAGÓGICOS .....                               | 76        |
| 3.6.4. EJEMPLO DE APLICACIÓN DEL LABORATORIO REMOTO .....                | 78        |
| 3.7. INSTALACIÓN Y MONTAJE DEL LABORATORIO REMOTO .....                  | 79        |
| <b>4. PRESUPUESTO DEL LABORATORIO REMOTO .....</b>                       | <b>80</b> |
| <b>5. CONCLUSIONES .....</b>   | <b>81</b> |
| <b>6. BIBLIOGRAFÍA .....</b>   | <b>84</b> |
| <b>APÉNDICES .....</b>   | <b>86</b> |

- A01-MANUAL DE INSTALACIÓN Y MONTAJE
- A02-MANUAL DE USUARIO
- A03-MANUAL DE ADMINISTRADOR
- A04-GUÍA DE PRÁCTICAS
- A05-MANUAL DE USO ICESTUDIO
- A06-MANUAL VERILOG
- A07-MANUAL DE USO ICARUS VERILOG + GTKWAVE

#### CURRÍCULUM VITAE DEL AUTOR

## ÍNDICE DE ILUSTRACIONES

|  |    |
|--|----|
| <i>Figura 2.1: Diagrama laboratorio tradicional .....</i>  | 15 |
| <i>Figura 2.2: Diagrama laboratorio remoto.....</i>  | 16 |
| <i>Figura 2.3: Arquitectura general de un Laboratorio Remoto .....</i>                                 | 18 |
| <i>Figura 2.4: Esquema del procedimiento de un servidor web .....</i>                                  | 25 |
| <i>Figura 2.5: Pasos habituales en el flujo de trabajo con FPGA .....</i>                              | 28 |
| <i>Figura 2.6: Arquitectura interna de la FPGA (Xilinx) .....</i>                                      | 29 |
| <i>Figura 2.7: Arquitectura interna de los tipos de FPGA .....</i>                                     | 30 |
| <i>Figura 2.8: Flujo de trabajo en el diseño y configuración de una FPGA .....</i>                     | 37 |
| <i>Figura 2.9: Herramientas usadas para crear el bitstream .....</i>                                   | 39 |
| <i>Figura 2.10: Placa IceZUM Alhambra .....</i>  | 46 |
| <i>Figura 2.11: Pinout de la placa ICEZUM Alhambra.....</i>  | 47 |
| <i>Figura 3.1: Metodología de trabajo.....</i>   | 50 |
| <i>Figura 3.2: Arquitectura del Laboratorio Remoto FPGA IceZUM Alhambra.....</i>                       | 53 |
| <i>Figura 3.3: Pines de la IceZUM Alhambra.....</i>  | 55 |
| <i>Figura 3.4: Esquema de conexión LED externo.....</i>  | 56 |
| <i>Figura 3.5: Posición servos. ....</i>   | 57 |
| <i>Figura 3.6: Futuras ampliaciones Laboratorio Remoto IceZUM Alhambra.....</i>                        | 58 |
| <i>Figura 3.7: Esquema de conexión del Laboratorio Remoto IceZUM Alhambra. ....</i>                    | 59 |
| <i>Figura 3.8: Diagrama de proceso de la página web - Laboratorio Remoto IceZUM Alhambra .....</i>     | 64 |
| <i>Figura 3.9: Estructura programación de la página web - Laboratorio Remoto IceZUM Alhambra .....</i> | 66 |
| <i>Figura 3.10. Descripción y parámetros de la función exec (PHP). ....</i>                            | 68 |
| <i>Figura 3.11: Estructura de la base de datos del Laboratorio Remoto FPGA IceZUM Alhambra .....</i>   | 71 |
| <i>Figura 3.12: Acceso a video de demostración desde la página de compilación local.....</i>           | 78 |
| <i>Figura 3.13: Acceso a video de demostración desde la página de compilación remota.....</i>          | 79 |
| <i>Figura 3.14. Montaje Laboratorio Remoto FPGA IceZUM Alhambra .....</i>                              | 79 |

**ÍNDICE DE TABLAS**

---

|  |    |
|--|----|
| TABLA 2.1: COMPARACIÓN TIPOS DE LABORATORIOS.....                                    | 13 |
| TABLA 2.2: ETAPAS PARA EL DISEÑO Y DESARROLLO DE UN Laboratorio Remoto .....         | 21 |
| TABLA 2.3: COMPARACIÓN VHDL vs Verilog .....   | 35 |
| TABLA 3.1: ESTUDIO DE ALTERNATIVAS: LABORATORIO REMOTO FPGA IceZUM Alhambra .....    | 51 |
| TABLA 3.2: DISEÑO SISTEMA REAL: MONTAJE LABORATORIO .....                            | 54 |
| TABLA 3.3: ALTERNATIVAS DE ACCESO A MONITORIZACIÓN DE RESULTADOS CÁMARA .....        | 60 |
| TABLA 3.4: DISEÑO DE FUNCIONAMIENTO DE LA WEB – MODO DE OPERACIÓN.....               | 61 |
| TABLA 3.5: ANÁLISIS DE LOS FACTORES QUE INFLUYEN EN EL DISEÑO DE LAS PRÁCTICAS ..... | 73 |

**LISTADO DE SÍMBOLOS**

|                 |   |
|-----------------|---|
| <b>ADA</b>      | Lenguaje de programación orientado a objetos y fuertemente tipado                             |
| <b>AIM-Lab2</b> | Automated Internet Measurement Laboratory   |
| <b>ASCII</b>    | Código estándar americano para intercambio de información.                                    |
| <b>ASIC</b>     | Circuito integrado de aplicación específica en inglés Application Specific Integrated Circuit |
| <b>AWS</b>      | Amazon Web Services   |
| <b>CI</b>       | Circuitos integrados  |
| <b>CPLD</b>     | Complex Programmable Logic Device   |
| <b>CSS</b>      | Cascading Style Sheets, en español "Hojas de estilo en cascada"                               |
| <b>DIESEL3</b>  | Distance Internet-Based Embedded System Experimental Laboratory                               |
| <b>DIP</b>      | Dual In-line Package  |
| <b>DSP</b>      | Digital Signal Processing   |
| <b>EDA</b>      | Electronic Design Automation  |
| <b>EET</b>      | Tecnología de Ingeniería Electrónica  |
| <b>FPGA</b>     | Programmable Gate Array   |
| <b>FTDI</b>     | Future Technology Devices International   |
| <b>GAL</b>      | Generic Array Logic   |
| <b>HDL</b>      | Hardware Description Language   |
| <b>HTML</b>     | Hyper Text Markup Language  |
| <b>IBT</b>      | Intelligent Blended Training  |
| <b>IDE</b>      | Entorno integrado de desarrollo.  |
| <b>IOB</b>      | Bloque de entrada-salida.   |
| <b>IoT</b>      | Internet of Things  |
| <b>IP</b>       | Protocolo de internet.  |
| <b>JS</b>       | JavaScript, lenguaje de programación  |
| <b>LabVIEW</b>  | Laboratory Instrumentation Engineering Workbench  |
| <b>LR</b>       | Laboratorio remoto  |
| <b>LT</b>       | Laboratorio tradicional   |
| <b>LV</b>       | Laboratorio Virtual   |
| <b>MIT</b>      | Instituto Tecnológico de Massachusetts  |
| <b>MinGW</b>    | Minimal GNU for Windows   |
| <b>MySQL</b>    | Gestor de base de datos relacionales multiplataforma basado en SQL                            |
| <b>NTNU1</b>    | Norwegian University of Science and Technology  |
| <b>OIT</b>      | Instituto de Tecnología de Oregon   |
| <b>PAL</b>      | Programmable Array Logic  |
| <b>PCF</b>      | Fichero donde aparecen los pines restringidos de la placa y su uso                            |

|                |  |
|----------------|--|
| <b>PHP</b>     | Hypertext Preprocessor   |
| <b>PLA</b>     | Programmable Logic Array   |
| <b>PnR</b>     | Emplazado o ruteado (Place and Route)  |
| <b>RAM</b>     | Memoria de acceso aleatorio.   |
| <b>RLMS</b>    | Remote Laboratory Management System  |
| <b>ROM</b>     | Memoria de solo lectura  |
| <b>SQL</b>     | Lenguaje de Consulta Estructurado, Structured Query Language en inglés                           |
| <b>SRAM.</b>   | Static RAM   |
| <b>TCP/IP</b>  | Transmission Control Protocol/Internet Protocol  |
| <b>TIC</b>     | Tecnologías de la información y comunicación   |
| <b>UART</b>    | Universal Asynchronous Receiver-Transmitter, en español: Transmisor-Receptor Asíncrono Universal |
| <b>UMH</b>     | Universidad Miguel Hernández   |
| <b>UNED</b>    | Universidad Nacional de Educación a Distancia  |
| <b>UniSA</b>   | Universidad de Australia del Sur   |
| <b>USB</b>     | Bus serie universal.   |
| <b>Verilog</b> | Lenguaje estándar de descripción de hardware   |
| <b>VHDL</b>    | Lenguaje estándar de descripción de hardware   |

## 1. INTRODUCCIÓN

La aplicación de las tecnologías de la información y la comunicación en la educación, desde hace varios años está promoviendo nuevas formas de aprendizaje. En la actualidad ya existen muchos ejemplos de laboratorios remotos en uso, y con experiencias muy positivas. Por ello su uso ha crecido exponencialmente desde principios del siglo XXI. Cada vez son más los centros educativos que cuentan con este tipo de dispositivos, pero aún queda mucho camino por hacer, puesto que se conocen pocas experiencias en centros que no sean de educación superior.

Las FPGAs se conocen desde los años 80, pero era una tecnología privativa, lo que implicaba que sólo unos pocos tenían acceso a ella, y menos aún conocían sus detalles. A partir de mayo de 2015, el ingeniero austriaco Clifford Wolf hizo ingeniería inversa de la familia ICE40 de Lattice, publicó toda su información (proyecto Icestorm) y creó el primer sintetizador de FPGAs libre de la historia. Este hito ha constituido una revolución digital en el campo de las FPGA, puesto que hace que estos dispositivos sean accesibles al público en general. Desde 2015 se han desarrollado diferentes tarjetas entrenadoras que utilizan la FPGA Lattice iCE40, una de ellas es la IceZUM Alhambra, que cuenta con su IDE, Icestudio que incluye las herramientas del proyecto Icestorm.

Con las FPGAs se tiene la posibilidad de programar y manipular diferentes tipos de elementos en la industria (el control y arranque de motores, encendido y apagado de alumbrado por sectores, control PID, aplicaciones domóticas), por ello es interesante que los estudiantes de ingeniería industrial conozcan la capacidad que tienen estos dispositivos, programándolos y utilizándolos, y de paso conocer los lenguajes HDL, y la lógica programable.

### 1.1. MOTIVACIÓN

Un laboratorio remoto para poder manipular desde casa la FPGA IceZUM Alhambra proporcionaría al estudiante de ingeniería industrial la posibilidad de aprender a programarla sin desplazarse, hecho que cobra importancia en la Educación a Distancia ya que estudiante y centro educativo no suelen coincidir en el espacio.

Por otro lado, el desarrollo de un laboratorio remoto, proporciona a la autora de este proyecto la posibilidad de aplicar esta tecnología en el ejercicio profesional como profesor de secundaria de Formación Profesional, y poder implementar dispositivos similares que permitan a los alumnos conocer los conceptos la aplicación de la lógica programable.

## 1.2. OBJETIVOS

El presente Proyecto Final de Master consiste en el desarrollo del laboratorio remoto de la FPGAs libre IceZum Alhambra. Los objetivos que se pretenden alcanzar son:

- ❖ Implementar una página web para controlar remotamente la FPGA libre IceZUM Alhambra
- ❖ Diseñar un montaje con diferentes periféricos a controlar con lo FPGA
- ❖ Elaborar los materiales didácticos necesarios para la utilización del laboratorio remoto.
- ❖ Programar varias prácticas en orden creciente de dificultad para realizar en laboratorio remoto

## 1.3. ESTRUCTURA DE LA MEMORIA

En primer lugar, en el apartado 2 de la memoria, se realiza un estudio del estado del arte de los laboratorios remotos, profundizando en el conocimiento de los tipos de arquitectura que hay para elaborar esta clase de herramientas de aprendizaje. Por otro lado, se expone el trabajo de investigación realizado, con respecto a las FPGAs libres, lenguajes HDL y desarrollo web.

Una vez realizado el estudio de la documentación, en el apartado 3 se explica y desarrolla la arquitectura del sistema. En este apartado se expone un análisis de alternativas con la finalidad de mostrar las diferentes decisiones que se han tomado a medida que se avanzaba en el trabajo.

Para comenzar la etapa de diseño del Laboratorio Remoto, se han valorado las posibilidades reales desde el punto de vista técnico, tecnológico y educativo del desarrollo de la experiencia a ser realizada por los estudiantes con acceso remoto. Esta etapa vertebraliza el desarrollo del Laboratorio Remoto.

Una vez definido el equipo de hardware necesario, la FPGA IceZUM Alhambra y los periféricos instalados, se procede al desarrollo de la aplicación web, haciéndose uso de las tecnologías de desarrollo web con sus lenguajes respectivos tanto para el backend (lado del servidor) y frontend (lado del cliente). Del lado del servidor se utiliza: PHP, MySQL y Apache. Mientras tanto para el desarrollo del lado del cliente se utiliza: HTML, JavaScript y CSS. De esta manera se logra que la aplicación web se convierta en un entorno atractivo y sobre todo de fácil navegación para el usuario final.

Por último se han diseñado los recursos educativos necesarios para el uso del Laboratorio Remoto. Por un lado se han elaborado manuales de uso de la aplicación web, y por otro se han elaborado manuales de uso de la IDE Icestudio, y del lenguaje HDL verilog, así como de Icarus Verilog (compilador de lenguaje Verilog) y del simulador GTKWave para verificar el funcionamiento de un diseño.

Finalmente, en el apartado 4 se realiza el presupuesto de Laboratorio Remoto y en el apartado 5 se exponen las conclusiones y líneas de trabajo futuro.

## 2. ESTADO DEL ARTE

### 2.1. LABORATORIOS REMOTOS

El concepto de “Internet of Things” (IoT-Internet de las cosas) en la última década se ha extendido exponencialmente y se ha convertido en una parte esencial de la vida cotidiana. IoT se refiere a un sistema de redes de dispositivos digitales y electrónicos interconectados, capaces de comunicarse entre sí y recoger y transferir datos sin interacción directa con el hombre.

Hoy, hay más de 16 miles de millones de dispositivos conectados y operando a nivel mundial, el equivalente de 2 dispositivos por persona, y es un número que se espera que crezca en los próximos años, hasta que toque 41 mil millones por 2020.

Según (Luis Rosado, 2005) la estructuración de información mediante sistemas hipermedia y multimedia, e Internet, son herramientas valiosas en la creación de sistemas de apoyo al aprendizaje. Una de las soluciones de e-Learning más interesantes son los e-laboratorios. Trasladando este entorno a la enseñanza actual, los elementos necesarios para abordar la realización de prácticas de forma virtual son los laboratorios virtuales y remotos, accesibles a través de Intranet, Internet o ambientes computacionales, donde el alumno realiza las prácticas de una forma lo más similar posible a como si estuviese en las dependencias del laboratorio tradicional, simulando e interactuando con instrumentos virtuales.

#### 2.1.1. TIPOS DE LABORATORIOS

En el laboratorio tradicional, los recursos en personas y espacios son restringidos, debido a su masificación y a problemas presupuestarios; se requiere la presencia física del estudiante y la supervisión del profesor. Una solución a estos problemas se puede encontrar en la aplicación de los avances tecnológicos a la docencia e investigación universitaria y, en concreto, el uso de laboratorios virtuales y remotos. El Laboratorio Virtual acerca y facilita la realización de experiencias a un mayor número de alumnos, aunque alumno y laboratorio no coincidan en el espacio. Permite simular fenómenos y modelos físicos, conceptos abstractos, mundos hipotéticos, controlar la escala de tiempo, etc, ocultando el modelo matemático y mostrando el fenómeno simulado de forma interactiva. La creciente complejidad de las actividades en el Laboratorio Tradicional y el desarrollo de las TIC y la Computación, han hecho que los Laboratorios Virtuales evolucionen, transformándose en laboratorios remotos, donde el alumno utiliza y controla los recursos del laboratorio, a través de una red local (Intranet) o bien a través de Internet.

#### *El laboratorio tradicional*

El laboratorio tradicional está consensuado en el ámbito académico y forma parte de la experiencia cotidiana de los docentes y los alumnos. Mientras que en el aula, el profesor transmite al alumno gran cantidad de información en poco tiempo, el Laboratorio Tradicional se presta para la demostración cuantitativa de datos experimentales, aclarar conceptos, verificar leyes o inducirlas: es el lugar ideal para que el alumno aprenda a utilizar sus conocimientos en situaciones reales. Aunque el Laboratorio Tradicional es lento en la transmisión de información, facilita el planteamiento de problemas que

permitan al estudiante aplicar sus conocimientos sobre la naturaleza, entrenándose en la aplicación del método científico.

### ***Laboratorio virtual***

---

Un laboratorio virtual es un sistema computacional que pretende aproximar el ambiente de un laboratorio tradicional. [1]

Los experimentos se realizan paso a paso, siguiendo un procedimiento similar al de un Laboratorio Tradicional: se visualizan instrumentos y fenómenos mediante objetos dinámicos, imágenes o animaciones. Se obtienen resultados numéricos y gráficos, tratándose éstos matemáticamente para la obtención de los objetivos perseguidos en la planificación docente de las asignaturas.

Cuando el profesor trabaja con sus alumnos en un Laboratorio Virtual, ha de escoger el nivel de interactividad del Laboratorio. En este sentido, distinguimos los siguientes niveles de interactividad de los Laboratorios Virtuales:

- ❖ Nivel 1: Laboratorio Virtual basado en una aplicación informática que se ejecuta en un ordenador aislado o en red local (sin conexión a Internet). Se trata de un PC aislado o perteneciente a una red local, en el que existe una aplicación que proporciona al alumno un ambiente de simulación de determinados fenómenos de interés. El Laboratorio Virtual se concibe como un programa ejecutable cerrado, donde el alumno arranca la aplicación desde el ordenador donde se ha grabado o a través de una red local. Los contenidos teóricos y las actividades se plantean como una serie de hipertextos.
- ❖ Nivel 2: Laboratorio Virtual a través de Internet, basado en páginas web estáticas sin realizar ningún tipo de simulación. Las páginas web contienen información sobre las actividades prácticas de laboratorio, con enlaces a otros lugares donde se halla más información.
- ❖ Nivel 3: Laboratorio Virtual a través de Internet, basado en páginas web con objetos dinámicos de simulación (applets de Java, Flash, etc). El alumno está conectado a Internet y un conjunto de utilidades asociadas al navegador, permiten la tutorización on-line mediante texto, voz o correo electrónico; posibilidad de videoconferencia; control y visualización de la pantalla del usuario y la posibilidad de control remoto de su ordenador. Las posibilidades multimedia y de simulación, son especialmente útiles en entornos de enseñanza basados en Internet utilizando applets.

### ***Laboratorios remotos***

---

La creciente complejidad de las actividades prácticas de laboratorio y el desarrollo de las TIC y la Computación, han hecho que los Laboratorios Virtuales evolucionen, transformándose en laboratorios remotos. El Laboratorio Remoto es un sistema basado en instrumentación real de laboratorio, no prácticas simuladas, que permite al estudiante realizar actividades prácticas de forma local o remota, transfiriendo la información entre el proceso y el estudiante de manera uni o bidireccional. El alumno

utiliza y controla los recursos disponibles en el laboratorio, a través de estaciones de trabajo de una red local (Intranet) o bien a través de Internet.

La diferencia entre el Laboratorio Remoto y el Laboratorio Virtual, reside en el tipo de computación subyacente y tratamiento del material: el Laboratorio Remoto se basa en instrumentos reales (tarjetas de adquisición de datos, instrumentos de medida, conexiones en interfaces diversas, comunicación de datos,...), mientras que en el Laboratorio Virtual sólo existen procesos de computación basados en simulaciones, ya sean applets de Java, Flash,...o bien programas o ambientes computacionales ejecutados en ordenadores aislados o en una Intranet. Los Laboratorio Remoto presentan mayores ventajas que los Laboratorios Virtuales, debido a que éstos proporcionan un mayor nivel de interactividad, y el alumno entra en contacto con equipamiento real, en lugar de entrar en contacto con programas simulados.

La transformación de un Laboratorio Tradicional en un Laboratorio Remoto requiere implementar el hardware y el software necesarios con el fin de automatizar y supervisar las experiencias que en él se lleven a cabo, así como la gestión de la comunicación, el control y la transferencia de información con los usuarios de dichas instalaciones de forma remota.

En la tabla siguiente se enumeran las ventajas e inconvenientes de cada tipo de laboratorio definido según (Luis Rosado, 2005) [1].

TABLA 2.1: COMPARACIÓN TIPOS DE LABORATORIOS

|                         | VENTAJAS  | INCONVENIENTES   |
|-------------------------|---|--|
| Laboratorio Tradicional | <ul style="list-style-type: none"><li>▪ Alta interactividad con el experimento</li><li>▪ Motivación que supone observar el experimento</li><li>▪ Desarrollo de habilidades cognitivas</li></ul> | <ul style="list-style-type: none"><li>▪ El material es muy caro</li><li>▪ En la enseñanza a distancia no es posible realizar actividades prácticas.</li><li>▪ Recursos humanos y materiales restringidos</li><li>▪ Las prácticas requieren una supervisión directa por parte del profesor</li><li>▪ Poco tiempo para las sesiones de laboratorio que dificulta la asimilación de los conceptos</li></ul> |

TABLA 2.1: COMPARACIÓN TIPOS DE LABORATORIOS

|                     | VENTAJAS   | INCONVENIENTES   |
|---------------------|--|--|
| Laboratorio Virtual | <ul style="list-style-type: none"> <li>▪ Acerca y facilita a un mayor número de alumnos la realización de experiencias, aunque alumno y laboratorio no coincidan en el espacio</li> <li>▪ Flexibiliza el horario de prácticas y evita la saturación por el solapamiento con otras asignaturas</li> <li>▪ Fomenta el trabajo personal del alumno</li> <li>▪ Reducen el coste del montaje y mantenimiento.</li> <li>▪ Son una alternativa barata y eficiente</li> <li>▪ El alumno es un participante activo, más que un observador</li> <li>▪ Permite obtener una visión más intuitiva de aquellos fenómenos que en su realización manual no aportan suficiente claridad gráfica</li> <li>▪ Los estudiantes aprenden mediante prueba y error, sin miedo a sufrir o provocar un accidente, sin avergonzarse de realizar varias veces la misma práctica, ya que pueden repetirlas sin límite; sin temor a dañar alguna herramienta o equipo</li> </ul>                         | <ul style="list-style-type: none"> <li>▪ No puede sustituir la experiencia práctica</li> <li>▪ Es importante que cada simulación en el Laboratorio Virtual venga acompañada de un guion que explique el concepto a estudiar, así como las ecuaciones del modelo utilizado. El objeto es que el estudiante realice una actividad ordenada y progresiva, que conduzca a alcanzar objetivos básicos concretos</li> <li>▪ El alumno no utiliza elementos reales en el Laboratorio Virtual, lo que provoca una pérdida parcial de la visión de la realidad</li> <li>▪ No siempre se dispone de la simulación adecuada para el tema que el profesor desea trabajar</li> </ul>  |
| Laboratorio Remoto  | <ul style="list-style-type: none"> <li>▪ Optimización del material de laboratorio y recursos humanos</li> <li>▪ El ahorro en material de laboratorio es considerable</li> <li>▪ La respuesta de los sistemas es la de un sistema real y no utiliza la simulación más que para la comparación de los resultados</li> <li>▪ Supone no tener que duplicar los materiales y poder acceder a ellos a través de la red como si se estuviese en el mismo puesto</li> <li>▪ El Laboratorio Remoto amplía la oferta horaria del alumno en su formación</li> <li>▪ Los límites espaciales y temporales no son restrictivos para el trabajo de laboratorio</li> <li>▪ El alumno no necesita disponer del software de simulación.</li> <li>▪ Los Laboratorio Remoto ofrecen la posibilidad de controlar de forma remota las aplicaciones basadas en instrumentos virtuales, donde destacan la modularidad y el carácter abierto de los objetos dinámicos de instrumentación</li> </ul> | <ul style="list-style-type: none"> <li>▪ La experimentación en tiempo real que exige períodos de muestreo relativamente pequeños, requiere el uso de recursos que por lo general, resultan costosos, además de la necesidad de disponer de sistemas operativos de tiempo real</li> <li>▪ Al conectar sistemas reales de laboratorio a Internet, en tareas de monitorización, es necesario implementar los protocolos de comunicaciones correspondientes</li> <li>▪ Tanto el hardware como el software han de ser suficientemente robustos para que no fallen en ningún momento, mientras el alumno los está utilizando</li> <li>▪ En los centros docentes no universitarios, no hay muchas experiencias, debido a la escasa formación del profesorado, la falta de medios informáticos y el coste que supone implementar estos sistemas</li> </ul> |

Actualmente, las universidades llevan unos años trabajando con un modelo de enseñanza e-Learning (presencia a distancia), debido a que en la enseñanza tradicional o en el Laboratorio Tradicional, es necesaria la presencia física del profesor y de los estudiantes, lo que implica recursos restringidos de infraestructura física, espacio y tiempo, lo que genera adquisición de conocimientos insuficientes por parte de los estudiantes, además la implementación y utilización de experimentos en los Laboratorio Tradicional tiene un alto costo, teniendo en cuenta que se debe mantener una infraestructura masificada por la cantidad

de estudiantes que requieren acceder a ella; además dicha infraestructura física suele ser sensible a desgaste acelerado por uso indebido y algunas de estas infraestructuras frecuentemente son subutilizadas, creando dificultades en el desarrollo de los laboratorios por parte de los estudiantes; la solución entonces es la mencionada anteriormente, la introducción de un modelo de enseñanza e-Learning, con el desarrollo de laboratorios remotos y virtuales, proporcionando a los estudiantes la realización de las experiencias, aunque no se coincida en espacio y tiempo con el profesor. [2]

La implementación de un Laboratorio Remoto no implica reemplazo de los laboratorios tradicionales o de la infraestructura actual, los Laboratorio Remoto llegan a complementar la educación de los estudiantes incluyendo tecnología a su formación, además un diseño adecuado puede proporcionar: realización de experiencias remotas con equipos reales, es decir tele-presencia en el laboratorio y análisis de datos experimentales reales con flexibilidad en el desarrollo del laboratorio debido a la posibilidad de elegir tiempo y lugar para su realización.

Un Laboratorio Remoto permite a los estudiantes desarrollar tareas o actividades de laboratorio a través de la red de redes utilizada para mayor intercambio de información a nivel mundial, es decir a través de Internet sin estar cerca físicamente de los equipos de laboratorio.

En un Laboratorio Tradicional, los estudiantes ejercen una interacción con los equipos de laboratorio mediante acciones físicas incluyendo manipulación directa con las manos y como resultado se obtiene retroalimentación percibida fisiológicamente por los órganos de los sentidos como sentido del tacto, de la visión y del audio. En la Figura 2.1, se ilustra este concepto.

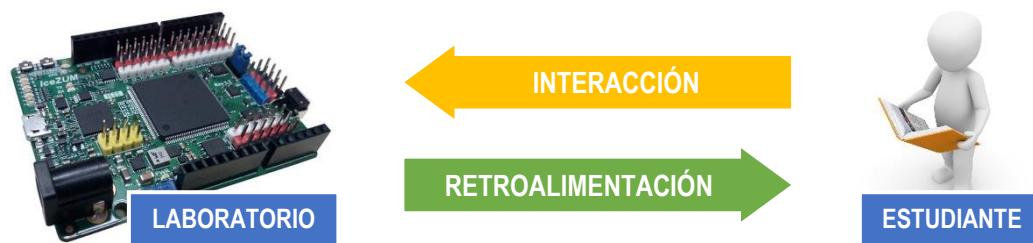


Figura 2.1: Diagrama laboratorio tradicional

En un Laboratorio Remoto dicha interacción se produce entre el estudiante en un lugar remoto y los equipos de laboratorio, a estos dos elementos (estudiantes y equipos de laboratorios) se le suma una nueva etapa de infraestructura remota, la cual es la encargada de hacer llegar las acciones del estudiante al laboratorio y enviar la información procedente de los equipos al estudiante. En la Figura 2.2, se ilustra este concepto.



Figura 2.2: Diagrama laboratorio remoto

### 2.1.2. EVOLUCIÓN DE LOS LABORATORIOS VIRTUALES Y REMOTOS

En el año 1994 se dieron los primeros indicios de aprendizaje a distancia, utilizando redes de comunicaciones, donde los resultados no fueron adecuados obteniendo videos o clases a ser vistas por los estudiantes de aproximadamente 6 imágenes por segundo.

En el año 1996 el Instituto Tecnológico de Massachusetts (MIT) implementó el primer Laboratorio Remoto, igualmente en el mismo año en Europa se desarrolla el proyecto DYNACORE el cual permite el control a distancia de un telescopio en España.

En 1999 con el proyecto BiC se obtuvieron mejores resultados utilizando redes de comunicación de banda ancha, con respecto a los primeros indicios en el año 1994.

Igualmente en el año 1999 la NTNU1 (Norwegian University of Science and Technology) desarrolló el AIM-Lab2 (Automated Internet Measurement Laboratory), el cual es un laboratorio en línea desarrollado con lenguaje de programación Java; de la misma manera la Universidad de Hagen desarrolló un laboratorio teleoperado en el área de control y robótica.

En Enero del año 2000, la empresa Motorola logra tener configuración, acceso y gestión remota para instalación y mantenimiento de redes locales, de la misma manera la empresa IBM a finales del año 2001 utiliza la tecnología Contivity Virtual Private Network para proporcionar acceso remoto a sus laboratorios a cualquier hora del día o de la noche y con toda seguridad.

La Universidad de Australia del Sur (UniSA) en el año 2001 desarrolla un laboratorio remoto en línea, en el cual los datos adquiridos en las prácticas pueden ser trasladados a un estudiante en ubicación remota para su posterior análisis y visualización.

En el año 2001 el investigador Bernardo Wagner con un grupo de colaboradores de la Universidad de Hannover, Alemania, desarrollaron un laboratorio para aprendizaje en línea, basado en tecnologías bases como HTML y Java, el cual incluye un conjunto de prácticas en tiempo real, con conceptos y modelos teóricos que son aplicados en la experimentación.

La Universidad Nacional de Taiwán con el departamento de educación industrial en el año 2001, implementó un sistema experto de aprendizaje a distancia con un agente pedagógico el cual ofrece una guía para el estudiante en el desarrollo de la experimentación.

En el año 2002 se inició un proyecto en la Universidad de Dakota del Norte para evaluar la posibilidad de ofrecer a los estudiantes trabajos de prácticas operadas a distancia para reducir el número de estudiantes en los laboratorios del campus a la mitad.

La Universidad de Ulster en el norte de Irlanda, en el año 2002 desarrollo un laboratorio a distancia para sistemas embebidos a nivel de maestría con una arquitectura cliente servidor, este desarrollo se basa en un programa de 3 años, llamado: proyecto DIESEL3 (Distance Internet-Based Embedded System Experimental Laboratory)

La empresa Sun Microsystems en el año 2004 pone en marcha el proyecto SunCampus en el cual desarrolla un entorno formativo avanzado llamado Sjuncampus por medio del cual el estudiante puede acceder a varios recursos como: Portal Web y laboratorios de prácticas accedidas de forma remota, llamados "elabs"; basándose en un concepto propio llamado Intelligent Blended Training (IBT), metodología de educación compuesta por formación presencial y formación vía web.

A mediados de la pasada década (aproximadamente en 2005), se realizan conferencias, congresos y desarrollo de laboratorios remotos buscando estrategias docentes para la educación basada en el uso de las TIC, muestra de esto es el premio a la innovación en la docencia, el cual fue ganado por la Universidad Miguel Hernández (UMH) en España , por el desarrollo de Recolab: laboratorio remoto de control utilizando Matlab/Simulink, propuesta pionera que permitió trabajar remotamente con equipos físicos utilizados en el laboratorio con lo cual se rompieron barreras físicas, temporales y de espacio.

La Universidad Nacional de Educación a Distancia (UNED) y su Departamento de Informática y Automática ha implementado laboratorios remotos y virtuales de automática utilizando la herramienta LabVIEW (Laboratory Instrumentation Engineering Workbench) de la empresa National Instruments, en los cuales desarrolla una arquitectura cliente/servidor en donde los clientes (estudiantes) interactúan con sistemas e-Learning como páginas web y applets Java y la infraestructura del servidor ha sido desarrollada con aplicaciones LabVIEW y tarjetas de adquisición de datos National Instruments; de la misma manera el Instituto de Tecnología de Oregon (OIT) y su departamento de Tecnología de Ingeniería Electrónica (EET) desarrollaron un laboratorio de electrónica real basado en la Web en el cual los estudiantes deben hacer cálculos y medidas experimentales de una variedad de circuitos electrónicos y comparar sus resultados.

### 2.1.3. ARQUITECTURA DE UN LABORATORIO REMOTO

Un Laboratorio Remoto permite trabajar sobre equipos reales por lo cual se requieren componentes de hardware y software que hagan posible la manipulación de dichos equipos. Para ello se diseña una arquitectura que haga posible este proceso. A continuación se describe brevemente la arquitectura general de un Laboratorio Remoto [3] [4]:

- ❖ Estudiante o usuario: accede por medio de una computadora o dispositivo móvil con conexión a Internet, si así lo permite el Laboratorio Remoto.
- ❖ Servidor web: encargado de mostrar el audio/video del laboratorio, las acciones que puede realizar sobre el laboratorio y los resultados de esas acciones.

- ❖ Servidor de base de datos: contiene la información del laboratorio (experimentos, datos de los alumnos, etc).
- ❖ Laboratorio: equipo que se controla o manipula a distancia.
- ❖ Cámara web: muestra lo que está ocurriendo en el laboratorio.

En la Figura 2.3 se muestra el esquema general de la arquitectura de un Laboratorio Remoto.

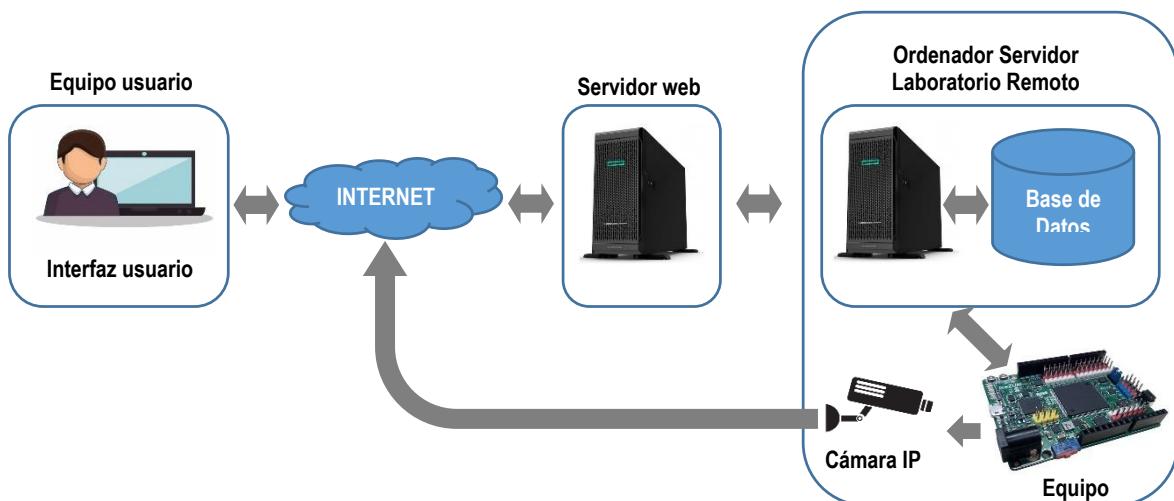


Figura 2.3: Arquitectura general de un Laboratorio Remoto

Hay proyectos de Laboratorio Remoto que poseen arquitecturas más sencillas y en algunos casos no se cuenta con cámara web, sobre todo en las experiencias de circuitos eléctricos y electrónicos. Existen otras arquitecturas más complejas que brindan recursos valiosos a usuarios, docentes e investigadores. Por ejemplo aquellas que permiten gestionar sus experiencias propias, además de otros Laboratorio Remoto. A los proyectos que presentan estas arquitecturas se les denomina RLMS (Remote Laboratory Management System)

#### 2.1.4. DISEÑO DE UN LABORATORIO REMOTO

Para llevar a cabo la experimentación remota se requiere de los siguientes requisitos [5] [6]:

- ❖ Telecontrol: Es necesario para poder manipular remotamente el experimento.
- ❖ Tele-presencia: Es muy importante que el estudiante que está realizando el experimento remotamente tenga una sensación real del mismo. Mediante la transmisión de audio y/o vídeo el estudiante puede tener la sensación de que está físicamente presente en el experimento.
- ❖ Recogida de datos: Los datos del experimento se han de poder recoger de alguna forma y guardarlos en el servidor para poderlos consultar y/o tratar por los estudiantes a posteriori.

- ❖ Planificación: Si se comparte un recurso entre diferentes estudiantes, deberá haber un sistema de planificación de acceso al recurso compartido.
- ❖ Seguridad: Un requisito importante es que el experimento debe ser controlado. No se debe permitir que los estudiantes remotamente puedan dañar las instalaciones o recursos que se están utilizando en la experimentación.
- ❖ Comunicación síncrona: La comunicación síncrona es necesaria para el telecontrol del experimento. Entre las técnicas existentes de comunicación síncrona cabe destacar: el chat o la videoconferencia. Habrá técnicas de comunicación síncrona que no serán factibles en entornos del hogar debido al gran ancho de banda que consumen.
- ❖ Entorno colaborativo: El entorno colaborativo virtual facilita la comunicación entre los estudiantes que estén separados geográficamente. La tecnología que hay detrás de cualquier herramienta de colaboración síncrona es un mecanismo que permite a un usuario mandar actualizaciones a otros usuarios acerca de las interacciones que se han realizado en el entorno colaborativo. Los componentes del grupo necesitan tener la misma vista del experimento en tiempo real.
- ❖ Entorno de realidad virtual multiusuario para un laboratorio remoto: La comunicación entre el estudiante y el experimento se realiza en su totalidad a través de la WWW, donde la interfaz para el estudiante será un navegador web. Éste provee una plataforma para la transmisión de información así como un entorno para la ejecución de software cliente. El servidor web es la interfaz entre el usuario y el experimento.
- ❖ Gestión de acceso: Mediante el sistema de gestión de acceso se lleva un control de las reservas por parte de los estudiantes de cita para utilizar los recursos del laboratorio. Este sistema tendrá dos interfaces: interfaz de administrador e interfaz de estudiante. La interfaz de administrador sirve para crear y borrar cuentas, controlar los accesos, los tiempos de acceso, etc.
- ❖ Interfaz con el experimento: Cuando llega el día del experimento, el estudiante se conecta al servidor con el navegador web y éste carga la página web del experimento. De manera opcional se puede ver y escuchar en tiempo real y con imágenes y sonidos reales lo que está pasando en el laboratorio donde se realiza el experimento de manera que de sensación de presencia. Para empezar el experimento es necesario que el estudiante se autentique en el sistema. Una vez el estudiante se ha autenticado y ha entrado en el sistema empieza el experimento. Toda la información respecto a la práctica se transfiere mediante una conexión TCP/IP<sup>1</sup>.

---

<sup>1</sup> Protocolo de Control de Transmisión/Protocolo de Internet (en inglés *Transmission Control Protocol/Internet Protocol*)

- ❖ Visualización de los resultados: Mientras se está realizando el experimento toda la información que se manda desde/hacia el servidor es almacenada en un fichero ASCII en el propio servidor. Despues los estudiantes pueden recuperar esta información para su posterior tratamiento o análisis de los resultados.

Se recomienda iniciar la etapa de diseño con la identificación de una práctica que se podría desarrollar de forma remota. Se debe valorar las posibilidades reales desde el punto de vista técnico, tecnológico y educativo del desarrollo de la experiencia a ser realizada por los estudiantes con acceso remoto. Si se concluye que ese Laboratorio Remoto es factible se deben analizar aspectos tales como [7]:

- ❖ Definir el equipo de Hardware necesario que se debe adquirir para el desarrollo del Laboratorio Remoto.
- ❖ Realizar una proyección en relación a los lenguajes de programación idóneos para permitir el acceso remoto de los equipos reales y todos los aspectos de Software requeridos.
- ❖ Proyectar el diseño de los recursos educativos necesarios para el uso del Laboratorio Remoto
  - Se deben realizar manuales técnicos preferiblemente en formato multimedia
  - Manuales didácticos con el fin de que las experiencias cumplan con los objetivos de aprendizaje que se busca alcanzar.
  - Se deben elaborar video-tutoriales cortos donde se expliquen tanto detalles técnicos como educativos de estos recursos
  - Video-tutorías
- ❖ Gestionar la compra del equipo que se definió como necesario
- ❖ Acordar los tiempos académicos necesarios.

Una vez que se han superado estas etapas, se inicia el desarrollo propiamente dicho del Laboratorio Remoto en el espacio asignado para el proyecto.

Para el desarrollo del Laboratorio Remoto lo primero que se debe realizar es definir la metodología a seguir; se pueden establecer las siguientes etapas [7]:

- ❖ Montaje del Laboratorio Remoto: es la etapa medular del desarrollo debido a que se debe automatizar la experiencia de laboratorio para que sea operada de forma remota.
- ❖ Desarrollo de la aplicación web: cuando la experiencia se encuentra automatizada se procede a la realización de la aplicación web; la misma debe ser desarrollada en un lenguaje de programación que permita el acceso desde diferentes dispositivos (computadoras, Smartphone y Tabletas).

- ❖ Revisión del producto desarrollado: se realiza una revisión para valorar la calidad del producto, identificando fallos y mejoras.
- ❖ Validación del Laboratorio Remoto: una vez que se han incorporado los aspectos señalados en la revisión, se debe proceder a validar el Laboratorio Remoto con una muestra de estudiantes y especialistas en el diseño, desarrollo y uso de estos recursos educativos, con el fin de que el mismo reúna las características tecnológicas y educativas para su utilización en la enseñanza.
- ❖ Elaboración de recursos de apoyo pedagógicos: una vez superada la etapa de validación se debe desarrollar los materiales educativos que se utilizarán.
- ❖ Habilitación del Laboratorio Remoto: esta etapa implica que el Laboratorio Remoto está disponible para el empleo del mismo para la experimentación por parte de los estudiantes.

A continuación se muestran las etapas descritas anteriormente:

**TABLA 2.2: ETAPAS PARA EL DISEÑO Y DESARROLLO DE UN Laboratorio Remoto**

| Etapas                                     | Procedimientos  |
|--|---|
| <b>Selección</b>                           | Identificación de una experiencia   |
| <b>Diseño</b>                              | Valoración de la viabilidad de la práctica                                |
|  | Determinación de los equipos Hardware requeridos                          |
|  | Determinación de los desarrollos web necesarios                           |
|  | Compra de equipos   |
| <b>Desarrollo del Laboratorio Remoto</b>   | Definición de la metodología de trabajo                                   |
|  | Montaje del Laboratorio Remoto  |
|  | Desarrollo de la aplicación web   |
|  | Revisión final por parte de los encargados                                |
|  | Mejora de aspectos detectados en la revisión                              |
|  | Prueba piloto con especialistas en Laboratorio Remoto y estudiantes       |
|  | Mejora final del Laboratorio Remoto                                       |
| <b>Desarrollo de materiales didácticos</b> | Diseño de los apoyos pedagógicos  |
|  | Edición de materiales   |
| <b>Habilitación</b>                        | Habilitación web del Laboratorio Remoto                                   |
|  | Definición de protocolos de uso   |
| <b>Empleo didáctico</b>                    | Uso del Laboratorio Remoto en los cursos de la UNED y otras instituciones |

## 2.2. DESARROLLO WEB

Para que el laboratorio remoto sea funcional se hace uso de las tecnologías de desarrollo web con sus lenguajes respectivos tanto para el backend (lado del servidor) y frontend (lado del cliente). Del lado del servidor se utiliza: PHP, MySQL y Apache. Mientras tanto para el desarrollo del lado del cliente se utiliza: HTML, JavaScript y CSS. De esta manera se logra que la aplicación web se convierta en un entorno atractivo y sobre todo de fácil navegación para el usuario final.

### 2.2.1. PÁGINA WEB

Una página web dinámica interactiva es la que permite variar el aspecto y comportamiento en función de decisiones que toma el usuario a través de la interfaz gráfica de la página. Para ello se varía la estructura de la página. [8]

Para lograr interactividad en una página web, esta envía una petición al servidor de tal manera que el código se ejecuta en el servidor y este responde a su vez con una nueva página web en el navegador. La página web dinámica trabaja conforme a la información ingresada por el cliente, el sitio web cambiara de acuerdo a la forma en que haya sido definido en la programación.

El código que se ejecuta en el navegador cuando se ha realizado una petición, responde con una página web con extensión html, en caso que se esté utilizando un lenguaje de servidor respondería con la extensión asp, jsp o php.

### 2.2.2. HTML

El HTML (Hyper Text Markup Language), lenguaje de marcado de hipertexto, es un lenguaje de marcas (utiliza etiquetas, como marcas para delimitar elementos del lenguaje), que sirve para describir el contenido y la estructura de las páginas web, que pueden ser interpretadas y visualizadas a través de los navegadores de Internet (clientes web: Firefox, Internet Explorer, Chrome, etc.). [9]

Permite publicar información que contengan tablas, imágenes, textos, etc., obtener información por medio de enlaces que conllevan a otras páginas web. Además HTML permite diseñar formularios para inserción de datos, búsqueda de información, etc. Pero el código HTML por sí solo no resuelve esas peticiones y para ello es necesario programación del lado del servidor, en este caso aparecen los lenguajes de programación PHP, JSP, etc.

Se trata de un lenguaje sencillo, muy fácil de comprender, del cual existen aplicaciones capaces de generar el código HTML automáticamente y permite realizar páginas ligeras que se carguen rápidamente en el navegador. Por el contrario, los sitios web realizados en HTML son reconocidos de diferentes maneras dependiendo del navegador y su código es limitado en cuanto a su sintaxis.

### 2.2.3. LENGUAJE CSS

Este lenguaje es el encargado de definir la presentación de los datos que se indican en el código HTML. Para incluir las hojas de estilos CSS en la página web, es recomendable incluir un enlace a la

dirección donde se ubica el fichero CSS. El navegador interpreta las características CSS. Este lenguaje es enviado directamente desde el servidor. El navegador interpreta el código CSS y lo aplica a los elementos HTML. En el caso de que exista código CSS que no esté relacionado con ningún HTML, este no afectara a la página web. [8]

La hoja de estilos no es más que un grupo de especificaciones que indican el formato mediante el cual se van a presentar los elementos HTML de un determinado sitio web en nuestro navegador.

Permite tener un solo archivo CSS y se puede utilizar en diversas páginas web, haciendo referencia mediante las etiquetas, ayuda a mejorar el tiempo de respuesta del sitio web y detallar el aspecto de una página web. Aunque puede existir código CSS que no se llegue a utilizar en nuestro sitio web.

#### 2.2.4. PHP

Es un lenguaje de programación interpretado, que se ejecuta en el lado del servidor. Este lenguaje tiene una licencia de software libre y de código abierto. Es un lenguaje rápido, con una gran librería de funciones y abundante documentación. El tipo de implementación con respecto a cómo se combina con el código, es similar a ASP. El código PHP se interpreta en el servidor y se envía al navegador en forma de HTML. [8]

- ❖ Este lenguaje de programación tiene las siguientes características:
- ❖ Lenguaje sencillo de comprender.
- ❖ Es rápido a la hora de procesar cada línea de código.
- ❖ Es soportado por varias plataformas de entorno web.
- ❖ Posee características de lenguaje orientado a objetos.
- ❖ De fácil conexión con base de datos.
- ❖ Se puede alojar en hosting.
- ❖ Se puede encontrar fácilmente información referente a este lenguaje lo cual facilita el desarrollo de cualquier prototipo.
- ❖ Para verificar lo que se está creando con el lenguaje es necesario contar en nuestra pc con un servidor web.
- ❖ Es interpretado en el servidor lo cual hace que durante varias peticiones colapse el sitio web.
- ❖ El contenido en muchas ocasiones no es compatible con alguno de las versiones de los navegadores

## 2.2.5. SQL Y MYSQL

SQL (Lenguaje de Consulta Estructurado, Structured Query Language en inglés) es el lenguaje de acceso a base de datos relacionales más extendido. Con este sistema, el cliente especifica las instrucciones para crear, borrar o dotar de contenido las tablas de la base de datos, además de permitir su interrogación.

MySQL, a su vez, es un gestor de base de datos relacionales multiplataforma basado en SQL. Existen interfaces entre MySQL y numerosos lenguajes de programación habitualmente utilizados para diseñar programas servidores de internet. [10]

Para que una aplicación web funcione necesita de una base de datos. Se puede trabajar con MySQL y para trabajar conjuntamente con Apache existe una herramienta llamada PhpMyAdmin. Esta herramienta permite crear una base de datos de manera gráfica, y para acceder a la misma se lo hace mediante un navegador.

## 2.2.6. SERVIDOR WEB

Un servidor Web es quien se encarga de responder a las peticiones que realizan los usuarios, esta respuesta está conformada por todos los archivos necesarios para que se pueda presentar la página web que anteriormente haya solicitado el usuario. Para que el servidor web cumpla con dicha función es necesario que trabaje con el Protocolo de Transferencia de Hipertexto HTTP. Los ordenadores y los aparatos que se han configurado para esta función también se los conoce como servidores Web.

## 2.2.7. APACHE

Apache HTTP Server se encarga de procesar peticiones y posterior a eso entregar documentos de tipo web. Las repuestas que apache envía en base a las peticiones está en formato HTML.

Una de sus grandes características que llama la atención al ser de código abierto y sobre todo gratuito, es soportada por varias plataformas que usan sistemas operativos robustos.

Apache tiene como oficio confirmar las peticiones de los diferentes sitios web, este software es el encargado de negar o permitir el acceso a la información de acuerdo a las normativas establecidas, ya que son los usuarios quienes intentan acceder al sitio web en un determinado momento.

## 2.2.8. ARQUITECTURA CLIENTE – SERVIDOR

El modelo cliente – servidor describe el proceso que engloba la iteración entre un equipo local denominado CLIENTE y un equipo remoto llamado SERVIDOR. Muchos clientes pueden realizar distintas peticiones a lo cual el servidor está obligado a responder a cada petición con la debida información que ha sido solicitada. Para que esta comunicación tenga éxito tanto el cliente como el servidor debe estar conectado a una red local o una red mundial como lo es el Internet.

Dentro del modelo cliente – servidor existen dos procesos: los clientes (front - end), que se encargan de solicitar peticiones de conexión para obtener información, y los servidores (back – end), que tratan esas peticiones, obtienen la información y la envían a los procesos clientes. Las características básicas de una arquitectura cliente – servidor se pueden resumir en la siguiente lista [11]:

- ❖ El proceso cliente proporciona la interacción con el usuario y el resto del sistema, mientras que el servidor gestiona los recursos compartidos.
- ❖ Los procesos cliente y servidor pueden ejecutarse en el mismo nodo.
- ❖ Un servidor puede dar soporte a múltiples clientes.
- ❖ Los clientes realizan peticiones y, por lo tanto, son agentes activos, mientras que los servidores actúan como pasivos en la comunicación.
- ❖ La relación entre los clientes y los servidores se limita a la comunicación del mensaje.
- ❖ Es un sistema escalable tanto horizontal (a nivel de clientes) como vertical (a nivel de servidores).

En la siguiente imagen se puede observar el procedimiento de un servidor Web:

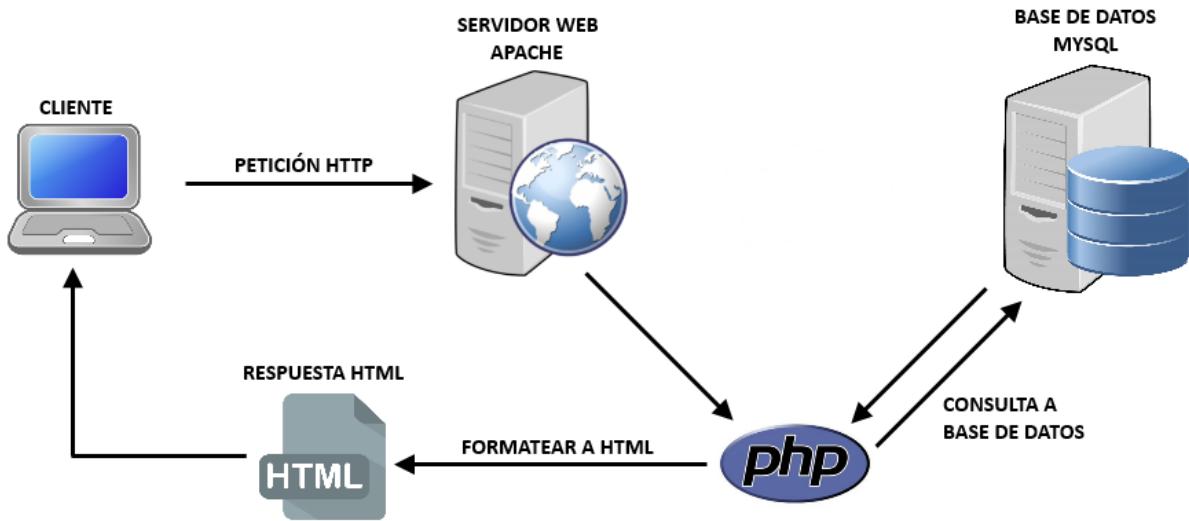


Figura 2.4: Esquema del procedimiento de un servidor web

## 2.3. FPGAs

Los Dispositivos FPGA (Fiel Programmable Gate Array, en español Arreglo de Compuertas Programable en el Campo), como su nombre lo dice son un arreglo matricial de bloques lógicos programables en cualquier espacio físico, útil para la implementación de los circuitos digitales.

Es un dispositivo semiconductor que posee varios bloques lógicos cuya interconexión y funcionalidad se puede programar por el usuario. La lógica programable puede emitir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica, un sistema combinacional y hasta sistemas complejos. La tecnología FPGA permite realizar diseños a medida, de bajo coste de desarrollo, incluso para la producción de pocas unidades. Estas características la hacen muy interesante para realizar prototipado rápido. Especialmente tiene un gran interés dentro del campo docente.

De manera esquemática, el diseño es muy predecible debido a la interconexión de compuertas y elementos lógicos, pero presenta limitaciones en el momento de crear librerías o interconectar subprocessos. Debido a esto, comúnmente se usan lenguajes de programación especializados del FPGA, conocidos como HDL (Hardware Description Language). Los dos estándares en cuanto a lenguajes de descripción de hardware son VHDL y Verilog, estando el uso del primero más extendido en Europa y el segundo en Estados Unidos.

### 2.3.1. INTRODUCCIÓN AL HARDWARE RECONFIGURABLE

Los circuitos integrados tradicionales, con el microprocesador al frente de una familia de la que también forman parte microcontroladores, ASIC (Application Specific Integrated Circuit), DSP (Digital Signal Processing) y otros, se caracterizan por contar con una microarquitectura hardware fija. La función de este tipo de circuitos se establece durante su diseño y queda fijada de forma permanente durante el proceso de fabricación. Algunos circuitos integrados (CI), como es el caso del microprocesador, son de propósito general. Por ello incorporan la capacidad de ser programados, ejecutando funciones definidas por software. Otros, como los ASIC y DSP, se diseñan desde un principio para satisfacer una función específica, siendo menos flexibles que un microprocesador pero ofreciendo a cambio un mejor rendimiento, mayor eficiencia energética, entre otras ventajas.

Paralelamente a los anteriores, durante las últimas tres décadas también se han desarrollado CI configurables por el usuario. Inicialmente estas soluciones, plasmadas en circuitos como las PAL (Programmable Array Logic) y PLA (Programmable Logic Array), se basaban en tecnologías de fusibles, por lo que únicamente podían configurarse una vez. La disponibilidad de nuevos tipos de memorias, incluyendo las estáticas y las no volátiles, han hecho posible fabricar CI reconfigurables.

Los orígenes del hardware reconfigurable se remontan más de medio siglo atrás, concretamente a 1959, y tienen lugar en la UCLA a iniciativa del matemático John Pasta. Este plantea un desafío: crear un ordenador en el que se combine una parte de hardware fijo con otra variable configurable por el operador, naciendo así el concepto de arquitectura F+V. La configurabilidad se conseguía merced a unos módulos que actuaban como componentes básicos y que el operario podía instalar e interconectar en una placa base que, posteriormente, se instalaba en el ordenador en cuestión.

El diseño de circuitos digitales empleando integrados discretos, con unas pocas puertas lógicas, dispuestos sobre una placa de conexión es algo del pasado, aunque sigue siendo una herramienta didáctica muy útil para la comprensión de los principios más básicos del funcionamiento de un computador. Actualmente es habitual el uso de estos recursos sustituyendo los CI discretos por microcontroladores como la Raspberry y Arduino, combinándolos con diversos sensores, leds, etc., o bien recurriendo a entrenadores digitales: aplicaciones software que simulan el funcionamiento del hardware.

Durante la década de los 70 aparecen los primeros circuitos integrados con arquitectura configurable, las anteriormente citadas PAL y PLA. Estos ofrecen matrices de puertas lógicas con conexiones sin establecer, de forma que el CI puede realizar distintas funciones según el proyecto en el que vayan a utilizarse. Las conexiones no se cablean, como ocurría con los integrados discretos, sino que se fijan mediante un proceso que consiste en quemar unos fusibles internos usando para ello un hardware a medida, como podía ser una placa conectable a un PC.

En la década de los 80 el fabricante Lattice introduce los GAL (Generic Array Logic), análogos a los PAL pero con una tecnología de establecimiento de las conexiones basada en memoria EEPROM. A diferencia de PAL y PLA, que eran CI que únicamente podían configurarse una vez, los GAL podían ser reconfigurados tantas veces como se necesitase, aportando mucha más flexibilidad.

Actualmente tanto GAL como PLA/PAL están en desuso, merced a los avances experimentados en tecnologías de hardware reconfigurable. En su lugar se recurre a integrados de tipo CPLD (Complex Programmable Logic Device) y FPGA. El encapsulado e estos deja atrás el clásico DIP (Dual In-line Package) con unas pocas decenas de pines, mientras que la tecnología de programación pasa a estar basada en memoria tipo Flash o bien SRAM. (Static RAM). La mayor ventaja respecto a los CI ya mencionados estriba en la mayor densidad de elementos básicos reconfigurables, pasando de pocas decenas de puertas lógicas a miles o millones de ellas. Además, en general no es necesario disponer de hardware especializado para establecer la configuración de estos CI, esta se obtiene directamente de la memoria del ordenador o de la placa de entrenamiento correspondiente. [12]

### 2.3.2. ESTADO ACTUAL DE LA TECNOLOGÍA FPGA

A pesar de ser una tecnología disponible desde hace más de 30 años, el primer circuito FPGA fue comercializado por Xilinx en 1985, la difusión de este tipo de CI, haciendo dicho producto accesible al usuario en general, es un hecho relativamente reciente. A ello han contribuido múltiples factores, entre los cuales habría que destacar la adquisición de Altera, el segundo mayor fabricante de FPGA mundial, por parte de Intel en 2015. La entrada en escena del gigante de los microprocesadores ha incrementado el nivel de competencia, provocando el lanzamiento de productos basados en FPGA con mayores capacidades y a menor precio.

En 2016 Intel presentó sus procesadores para servidores Xeon con circuitería FPG A integrada, destinados al diseño de centros de procesamiento de datos que, como los creados por Microsoft para sus servicios Azure y Bing a partir de su proyecto Catapult, combinan un microprocesador clásico con la posibilidad de implementar en hardware las partes más críticas para el rendimiento y consumo del sistema. En el extremo opuesto, en cuanto a nivel de prestaciones se refiere, también se tiene la familia de procesadores Atom E600C con FPGA integrada, producto dirigido a sistemas de tipo IoT (Internet of Things) y empotrados.

Por otra parte Xilinx, el primer fabricante mundial de dispositivos FPGA, también cuenta con productos que aún en un mismo encapsulado hardware reconfigurable con núcleos de procesamiento de tipo ARM. Ambas gamas de producto, la de Intel y la Xilinx, hacen posible la implementación de soluciones híbridas hardware-software. Esta es una tendencia que está llegando incluso a los servicios de computación en la

nube, como los ofrecidos por AWS (Amazon Web Services), en los que el desarrollador cuenta con instancias de ejecución que combinan los servidores tradicionales con hardware reconfigurable de tipo FPGA. [12]

Por último no hay que olvidar que aunque las FPGAs se conocen desde los años 80, era una tecnología privativa. Esto implicaba que sólo unos pocos tenían acceso a ella, y menos aún conocían sus detalles. A partir de mayo de 2015 esta situación cambia gracias al trabajo de ingeniería inversa realizado por el ingeniero austriaco Clifford Wolf sobre la familia ICE40 de Lattice, creando el primer sintetizador libre de la historia y publicó toda su información en el proyecto Icestorm<sup>2</sup>.

A partir de la información del proyecto Icestorm se crea Icestudio, que es una herramienta para diseño y síntesis de circuitos digitales en FPGAs libres, creada por Jesús Arroyo. Se trata de un software libre y multiplataforma. Este entorno de desarrollo es simple y ofrece una interfaz que permite instalar todas las herramientas necesarias para trabajar con las FPGAs libres. [13]

### 2.3.3. METODOLOGÍA DE TRABAJO CON HARDWARE RECONFIGURABLE

El procedimiento de trabajo con hardware reconfigurable implica múltiples pasos específicos, estando más cerca del proceso de diseño de circuitos tipo ASIC que del desarrollo de software. Los pasos fundamentales son los representados esquemáticamente en la Figura 2.5



Figura 2.5: Pasos habituales en el flujo de trabajo con FPGA

Se parte con el diseño de la solución que quiere implementarse en el hardware reconfigurable, tarea para la cual se puede, dependiendo de la complejidad del caso, elaborar un esquemático o bien describir el hardware mediante un lenguaje apropiado (VHDL o Verilog).

Tras compilar el diseño se procede normalmente a probarlo, usando para ello una herramienta de simulación. Esta puede ser de bajo nivel, construyendo waveforms que representan los cambios en las señales a lo largo del tiempo, o bien estar descrita en forma de test bench usando el propio lenguaje de descripción de hardware.

Una vez ya se han confirmado el correcto funcionamiento del circuito en el entorno de simulación llega el momento de sintetizar dicho circuito, determinando el hardware de la FPGA que se empleará para implantarlo. Asimismo es necesario emplazar las señales de entrada/ salida, estableciendo los pines de integrado al que se conectarán. Los elementos conectados a dichos pines pueden estar predeterminados o

<sup>2</sup> <http://www.clifford.at/icestorm/>

no, dependiendo del hardware concreto que se emplee. El paso final consiste en transferir la configuración generada por la herramienta de diseño/desarrollo al hardware, configurando la FPGA. [12]

En una FPGA el proceso de carga o programación es llamado configuración (no programación), usando un archivo de configuración llamado “bits de configuración” (“configuration bitstream”) que define la funcionalidad del FPGA.

La FPGA puede opcionalmente auto-cargar el archivo de configuración, por ejemplo desde una memoria externa no-volátil. Una FPGA puede también ser configurado por un dispositivo tipo microprocesador, DSP procesador, PC o algún otro dispositivo que pueda comunicarse inteligentemente con la FPGA.

#### 2.3.4. ESTRUCTURA GENERAL DE LA FPGA

Los elementos básicos que constituyen una FPGA como las de Xilinx se pueden ver en la Figura 2.6 y son las siguientes:

1. **Bloques Lógicos:** La estructura y contenido se denomina arquitectura en la cual está la descripción del sistema. Hay muchos tipos de arquitecturas, que varían principalmente en complejidad (desde una simple puerta hasta módulos más complejos). Suelen incluir biestables para facilitar la implementación de circuitos secuenciales. Otros módulos de importancia son los bloques de entrada/salida.
2. **Recursos de interconexión:** Cuya estructura y contenido se denomina arquitectura derutado.
3. **Memoria RAM,** que se carga durante el RESET para configurar bloques y conectarlos.

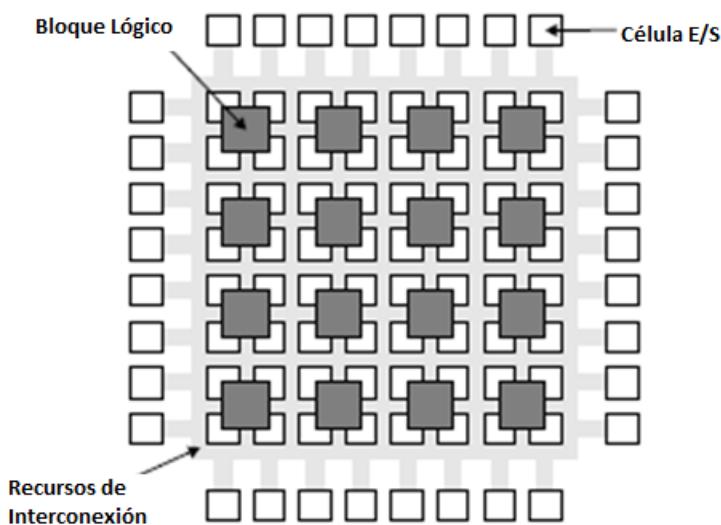


Figura 2.6: Arquitectura interna de la FPGA (Xilinx)

Por supuesto, no todas las FPGA son iguales. Dependen de los fabricantes para poder encontrar con diferentes soluciones. Las FPGA que existen en la actualidad en el mercado se pueden clasificar como pertenecientes a cuatro grandes familias, dependiendo de la estructura que adoptan los bloques lógicos que tengan definidas. Las cuatro estructuras se pueden ver en la Figura 2.7, sin que aparezcan en la misma los bloques de entrada/salida. [14]

- ❖ Matriz simétrica, como son las de XILINX
- ❖ Basada en canales, como ACTEL
- ❖ Mar de puertas, como ORCA
- ❖ PLD jerárquica, como ALTERA o CPLD de XILINX.

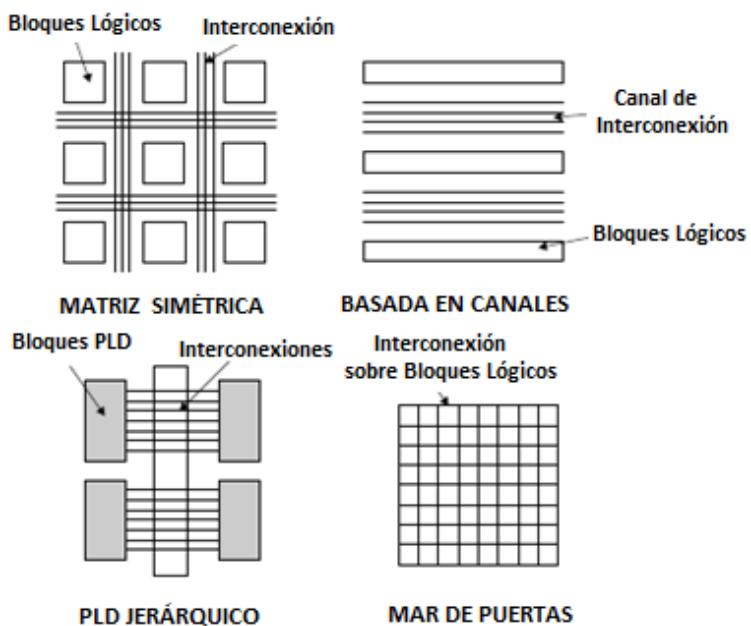


Figura 2.7: Arquitectura interna de los tipos de FPGA

### 2.3.5. DISEÑO BASADO EN COMPONENTES SIMPLES

Las herramientas de diseño facilitadas por los diferentes fabricantes de productos FPGA, así como las disponibles a través de licencias de código libre, ofrecen bibliotecas de puertas lógicas y algunos componentes simples, como los codificadores, descodificadores y multiplexores, que pueden ser empleados durante las prácticas de asignaturas como Electrónica Digital

Partiendo de esos elementos simples es posible crear circuitos biestables y, a partir de estos, unidades básicas de memoria de un bit que, a su vez, serían el pilar de la creación de registros con el ancho de palabra que interesa. Análogamente, con los demás elementos básicos es posible diseñar máquinas de estados que actúen como unidad de control. Las placas de entrenamiento cuentan con una señal de reloj

integrada que es posible conectar a los circuitos diseñados permitiendo así la creación de circuitos tanto combinacionales como secuenciales.

Para la realización de este tipo de prácticas los estudiantes pueden efectuar sus diseños con esquemáticos, colocando componentes y conectándolos, o bien aprender los fundamentos más básicos de un lenguaje de descripción de hardware HDL. Con independencia de ello los diseños una vez validados se sintetizan y transfieren a la placa FPGA, ejecutándose en hardware real en lugar de ser una simulación. [12]

### 2.3.6. DISEÑO BASADO EN MÓDULOS REUTILIZABLES

Tanto VHDL como Verilog, los dos lenguajes de descripción de hardware de uso más habitual, contemplan la posibilidad de crear bibliotecas de módulos con componentes prediseñados. Se pueden tener módulos con elementos básicos, como registros, sumadores, unidad de control, etc. [12]

## 2.4. LENGUAJES HDL

---

Un Lenguaje de Descripción Hardware (HDL) es un lenguaje de programación software utilizado para modelar el funcionamiento previsto de un circuito hardware. Existen dos aspectos de los HDL que facilitan las descripciones hardware:

- ❖ **El modelado de comportamiento abstracto:** un lenguaje es declarativo con el fin de facilitar la descripción del comportamiento hardware del circuito implementado. El diseño o los aspectos estructurales no perjudican al comportamiento hardware.
- ❖ **Modelado de la estructura hardware:** La estructura hardware puede ser modelada en un HDL independiente al HDL usado para modelar el comportamiento del diseño.

Este comportamiento puede ser modelado y representado en varios niveles de abstracción durante el proceso de diseño. Los modelos con altos niveles de abstracción describen el comportamiento del diseño de forma abstracta, mientras que los modelos con niveles bajos de abstracción incluyen más detalle. Los lenguajes de descripción hardware son lenguajes especializados en la descripción de estructuras, el diseño de las mismas y el comportamiento del hardware. Con estos lenguajes se pueden representar diagramas lógicos, circuitos con diferentes grados de complejidad, desde expresiones booleanas hasta circuitos más complejos. Estos lenguajes sirven para representar sistemas digitales de manera legible para las máquinas y para las personas

### 2.4.1. HISTORIA DE LOS LENGUAJES HDL

El aumento de la dificultad del diseño electrónico, así como los costos de desarrollo tecnológico y tiempo de lanzamiento de productos impulsó al desarrollo de herramientas de diseño electrónico automático (EDA, Electronic Design Automation). Estas herramientas son de gran utilidad a los ingenieros en las diferentes etapas del diseño digital. Los lenguajes de descripción de hardware (HDL) son el principal componente que ha contribuido a complementar estas herramientas para el diseño digital. Hacia fines de los 80 mientras los diseños de circuitos integrados de aplicación específica (ASICs) crecían en complejidad,

los sistemas gráficos para describir estos empezaban a mostrar limitaciones. El desarrollo, visualización, depuración y mantenimiento de estos sistemas era cada vez más complicada ya que mientras los diseños alcanzaban las 5000 compuertas lógicas, la cantidad de hojas de esquemas gráficos crecían de igual forma. El seguir planos esquemáticos con decenas de hojas daba paso a que surjan muchos errores y que el proceso se vuelva muy lento para depurar estos errores.

Es así que en 1985 una compañía llamada Automated Integrated Design Systems (renombrada después como Gateway Design Automation en 1986), introduce un nuevo producto llamado Verilog, era el primer simulador lógico que integraba un lenguaje de alto nivel y simulación a nivel de compuertas lógicas. En 1984 se crea VHDL por Texas Instruments.

#### 2.4.2. FUNDAMENTOS

Estos lenguajes tienen una serie de fundamentos necesarios para entender la tecnología que se va a explicar a continuación [15]:

##### ***Niveles de abstracción***

---

Los niveles de abstracción se refieren al grado de detalle de una descripción HDL respecto a su implementación física. Un diseño se puede representar bajo distintos punto de vista.

- ❖ Funcional: Relación entre entradas y salidas sin referencia a la implementación. Dada la posibilidad de que varios procesos puedan necesitar un mismo recurso, o la necesidad de sincronizar procesos independientes, se utilizan instrucciones similares a las de los lenguajes de programación concurrente. La precisión temporal son relaciones causales sin planificación temporal.
- ❖ Arquitectural: División en bloques funcionales, con planificación en el tiempo de las acciones a desarrollar. Se agrupa diferentes acciones en diferentes estados y se sincronizan por un reloj. En él se describe el sistema mediante diagramas de transferencia, tablas de verdad o ecuaciones lógicas
- ❖ Lógico: Poca precisión debido a retrasos de componentes y conexiones. Está formado por componentes de circuito expresados en ecuaciones o puertas lógicas y elementos de una biblioteca.

##### ***Estilos de descriptores***

---

En los HDL se pueden diferenciar tres estilos de descripciones:

- ❖ Descripción estructural: Especifica los elementos que se pueden usar y sus interconexiones. Estos elementos pueden ser: procesadores, memorias, puertas lógicas, bloques funcionales o transistores.

- ❖ Descripción algorítmica: Describe el funcionamiento del circuito, componente o modulo. Es similar a la descripción de los programas software. Estos descriptores pueden ser un sistema, algoritmos de comportamiento, transferencias de registros o ecuaciones lógicas.
- ❖ Descripción flujo de datos: Genera los datos necesarios para la realización física del módulo, estos datos reflejan el flujo de información y dependencia entre operaciones y datos. Este tipo de descriptor puede ser un grupo de bloques, particiones físicas, un trazado o un plano base.

#### 2.4.3. ESTRUCTURADOS Y NO ESTRUCTURADOS

Dentro de los lenguajes de descripción hardware se puede diferenciar dos clases, los HDL estructurados y los no estructurados.

- ❖ Los no estructurados son lenguajes sencillos orientados a la realización de un único circuito, entendiendo como circuito un único modulo o componente con una única función.
- ❖ Los lenguajes estructurados permiten la realización de circuitos complejos o de varios circuitos. Este tipo de lenguaje se caracteriza por:
  - Su capacidad multinivel, permitiendo así diferentes niveles de jerarquía
  - Su capacidad de combinar sus diferentes estilos de descripciones
  - Sus instrucciones, que definen una sintaxis independiente del nivel de descripción
  - Su independencia tecnológica, que le hace no depender de los fabricantes de los componentes físicos
  - Su universalidad, que garantiza la posibilidad de utilizar un gran número de herramientas de diseño
  - Su facilidad de comprensión y lectura que simplificar la documentación del circuito.

#### 2.4.4. TIPOS DE HDL

Existen tres tipos de lenguajes HDL:

1. **De bajo nivel:** Permiten definir un circuito a nivel de arquitectura (FlipFlops, compuertas básicas, ecuaciones lógicas) como son los lenguajes: PALASM, CUPL, ABEL
2. **De nivel medio:** permiten definir un circuito en modo jerárquico, así como la generación condicional/iterativa de hardware; en ciertos permiten el uso de descripciones de comportamiento (funciones aritméticas, máquinas de estado), como es el lenguaje: AHDL
3. **De alto nivel:** no sólo posibilitan mayor nivel de abstracción, sino que también son usados para la simulación, para la síntesis del generador de estímulos y el monitor de salidas, como son los lenguajes: VHDL, Verilog [16]

#### 2.4.5. LENGUAJE VHDL

El lenguaje de descripción de hardware VHDL (Very High Speed Integrated Circuits) es un conjunto de recursos que permite describir el modelado de circuitos digitales con estructuras jerárquicas, desde puertas lógicas hasta algoritmos de programación de sistemas de diseño digital avanzados.

En la década de 1980, los rápidos avances en la tecnología de los circuitos integrados impulsaron el desarrollo de prácticas estándar de diseño para los circuitos digitales. VHDL se creó como parte de tal esfuerzo y se convirtió en el lenguaje estándar industrial para describir circuitos digitales, principalmente porque es un estándar oficial de la IEEE. En 1987 se adoptó la norma original para VHDL, llamada IEEE 1076. En 1993 se adoptó una norma revisada, la IEEE 1164.

En sus orígenes, VHDL tenía dos propósitos centrales. Primero, servía como lenguaje de documentación para describir la estructura de circuitos digitales complejos. Como estándar oficial del IEEE, ofreció una forma común de documentar los circuitos diseñados por varias personas. Segundo, VHDL aportó funciones para modelar el comportamiento de un circuito digital, lo que permitió emplearlo como entrada para programas que entonces se usaban para simular la operación del circuito.

En particular VHDL permite tanto una descripción de la estructura del circuito (descripción a partir de subcircuitos más sencillos), como la especificación de la funcionalidad de un circuito utilizando formas familiares a los lenguajes de programación. [17]

#### 2.4.6. LENGUAJE VERILOG

Verilog HDL se originó en la empresa Automated Integrated Design Systems (más tarde rebautizada como Gateway Design Automation) en 1985. La empresa fue una empresa privada fundada en ese momento por el Dr. Prabhu Goel, el inventor del algoritmo de generación de pruebas Podem. Verilog HDL fue diseñado por Phil Moorby, quien luego se convertiría en el jefe de diseño de Verilog-XL y el primer Fellow Corporativa de Cadence Design Systems. Pasarela Design Automation creció rápidamente con el éxito de Verilog-XL y finalmente fue adquirida por Cadence Design Systems, San Jose, CA en 1989.

Es utilizado para diseñar sistemas electrónicos, Verilog soporta el diseño, prueba e implementación de circuitos analógicos, digitales y de señales mixta a diferentes niveles de complejidad.

Posee una sintaxis similar a la del lenguaje de programación C. El lenguaje tiene un preprocesador como C, y la mayoría de palabras reservadas de control como while, if entre otras son similares.

Verilog es uno de los HDL más utilizados y permite descripciones abstractas y representaciones en bajo nivel es decir puede describir sistemas digitales en base a compuertas, e incluso en base a transistores.

Permite que en un diseño se puedan usar diferentes niveles de descripción de sistemas digitales en un mismo ambiente, las diferentes descripciones pueden ser simuladas para verificar el funcionamiento y además pueden ser sintetizadas es decir traducidas a la interconexión de componentes básicas de un dispositivo programable.

Además permite la descripción estructural del diseño en base a componentes básicas, y descripciones más abstractas que se enfocan en la conducta del sistema. La conducta puede describirse mediante expresiones lógicas y también empleando procedimientos. Un diseño basado en descripciones funcionales o de comportamiento puede resultar lento y de gran tamaño. Las descripciones en niveles estructurales permiten un ahorro de los circuitos lógicos para maximizar la velocidad, minimizar el tamaño y más bajo costo. [18]

#### 2.4.7. COMPARACIÓN VHDL & VERILOG

Desde la creación de Verilog, se debate cuál de los dos lenguajes es el más apropiado para diseñar. La comunidad científica ha llegado a la conclusión de que no existe una razón importante para decantarse por un lenguaje u otro. Por un lado, el lenguaje VHDL es creado como un lenguaje para resolver las necesidades del diseño en su época, y otras futuras, por lo tanto, cuando este lenguaje fue pensado para ser duradero y dar la posibilidad de crear diseños con mayor dificultad. El resultado es un lenguaje más rígido y con una sintetización en el código que lo hace más legible, aunque sus reglas de diseño obligan a diseñar de forma, a veces, poco ágil. Por otro lado, Verilog nace para cubrir la necesidad de mejorar un flujo de diseño y por lo tanto, es un lenguaje que cubre todas las necesidades de diseño y simulación. El resultado es un lenguaje simple y flexible, fácil de aprender aunque tiene ciertas limitaciones. [16]

A continuación se van a enumerar una serie de diferencias entre ambos lenguajes:

**TABLA 2.3: COMPARACIÓN VHDL vs Verilog**

**COMPARACIÓN CARACTERÍSTICAS**

| VHDL   | Verilog   |
|--|---|
| <ul style="list-style-type: none"> <li>▪ Se presta en más niveles abstractos.</li> <li>▪ Fuertemente tipado, por lo que los errores de sintaxis se encuentran con mayor facilidad.</li> <li>▪ Este lenguaje tiene la capacidad para definir tipos personalizados.</li> <li>▪ Extremadamente detallado en codificación.</li> <li>▪ Cuenta con lista de sensibilidad. Si falta alguna señal en esta lista puede causar grandes diferencias entre la simulación y la síntesis.</li> <li>▪ Para realizar operaciones con señales de diferentes tipos (aunque estén fuertemente relacionados) es necesario convertir las señales al mismo tipo.</li> <li>▪ Este lenguaje no distingue entre mayúsculas y minúsculas. Los usuarios pueden cambiar el caso, siempre y cuando los caracteres en el nombre estén en el mismo orden</li> </ul> | <ul style="list-style-type: none"> <li>▪ Se presta hasta el nivel de puertas.</li> <li>▪ Débilmente tipado, por lo que el código es más propenso a errores debido a la combinación accidental de diferentes tipos de señal.</li> <li>▪ No tiene soporte para tipos personalizados.</li> <li>▪ Descripciones de bajo nivel más cerca del hardware real.</li> <li>▪ Carece de lista de sensibilidad.</li> <li>▪ Es un lenguaje compacto. La conversiones del lenguaje son similares al lenguaje C. Existe la posibilidad de mezclar y combinar las señales de tipos parecidos sin convertir los datos.</li> <li>▪ Sensibilidad a las mayúsculas. Diferencia entre mayúsculas y minúsculas.</li> </ul> |

| <b>COMPARACIÓN A NIVEL DE CÓDIGO</b>   |  |
|--|--|
| <b>VHDL</b>  | <b>Verilog</b>   |
| <ul style="list-style-type: none"> <li>▪ Este lenguaje es un poco complicado de aprender debido a que está basado en derivaciones de los lenguajes ADA y Pascal.</li> <li>▪ Se pueden desarrollar diseños digitales de distintos estilos como el funcional, flujo de datos y de estilo estructural. Este lenguaje le permite al programador crear un diseño de acuerdo a sus necesidades o a la pericia que tenga para llegar a la resolución de cualquier diseño digital. El único inconveniente es que se necesitan más líneas de código para resolver los diferentes diseños digitales.</li> <li>▪ Para obtener una resolución de cualquier diseño de manera más fácil y rápida se recomienda desarrollar la programación del diseño digital estudiando su estructura, comportamiento y dividiéndolos en pequeños procesos, para tener un código más claro y mejor estructurado</li> <li>▪ Se necesita declarar la entidad y la arquitectura del diseño digital a desarrollar de manera obligatoria para que se pueda ejecutar el programa.</li> <li>▪ Es obligatorio la declaración de la entidad (ENTITY) y de la arquitectura (ARQUITECTURE) del diseño a desarrollar para que se puede ejecutar el programa, de otra manera, no se podrá.</li> <li>▪ Presenta una sola forma de asignar valores.</li> <li>▪ La declaración de las puertas básicas conocidas (AND; OR; NOT; etc) se realizan llamándolas por su debido nombre.</li> <li>▪ Las estructuras de selección (IF, CASE, WITH, etc) en su sintaxis de declaración tienen forma similar a las de un lenguaje de programación como en pascal.</li> <li>▪ A través de la palabra reservada del programa “Event” se pueden generar diferentes eventos o pulsos de reloj.</li> <li>▪ Se pueden declarar procesos y funciones.</li> </ul> | <ul style="list-style-type: none"> <li>▪ Es de fácil aprendizaje debido a que es una derivación del lenguaje C y todo ingeniero lo conoce.</li> <li>▪ Se pueden desarrollar los ejercicios de la misma manera que presenta VHDL, pero con la ventaja que se necesita menos líneas de códigos para la resolución del mismo diseño digital debido a que el lenguaje es más abstracto.</li> <li>▪ Al desarrollar la programación de los diseños estudiando su estructura, su comportamiento y dividiéndolos en pequeños procesos se obtendrá la resolución del diseño de manera fácil y rápida.</li> <li>▪ No se necesita una entidad externa para declarar los datos ya que vienen incluido dentro de la misma arquitectura en este caso llamado MODULE (modulo) en donde se realiza el desarrollo del diseño.</li> <li>▪ Presenta dos formas de asignar valores.</li> <li>▪ La declaración de las puertas lógicas básicas conocidas se las realiza llamándolas por su nombre o a través de símbolos.</li> <li>▪ Las estructuras de selección (IF; CASE; WITH; etc) en su declaración es muy semejante a la declaración del lenguaje C.</li> <li>▪ A través de la palabra reservada “Posedge y Negedge” se pueden generar diferentes eventos o pulsos de reloj.</li> <li>▪ Se pueden declarar módulos y sub módulos</li> </ul> |
| <b>COMPARACIÓN A NIVEL SIMULACIÓN</b>  |  |
| <b>VHDL</b>  | <b>Verilog</b>   |
| <ul style="list-style-type: none"> <li>▪ Se puede observar el comportamiento del diseño en distintos instantes de tiempo.</li> <li>▪ Fácil de manipular los datos de entrada y salida del diseño.</li> <li>▪ Se utiliza el mismo sistema para la simulación en ambos lenguajes</li> </ul>  | <ul style="list-style-type: none"> <li>▪ Debido a que lleva más código para la solución de un diseño digital así mismo su tiempo de simulación va a aumentar pero en pocos ms.</li> <li>▪ No define ningún tipo de control de simulación o capacidades de monitoreo dentro del lenguaje. Estas capacidades son herramientas dependientes.</li> </ul>   |
|  | <ul style="list-style-type: none"> <li>▪ Debido a que lleva menos código para la solución de un mismo diseño digital así mismo su tiempo de simulación va a disminuir pero en pocos ms.</li> <li>▪ Define un conjunto de control básico de simulación en el lenguaje.</li> </ul>   |

| <b>COMPARACIÓN A NIVEL DE LIBRERÍAS</b>   |  |
|---|--|
| VHDL  | Verilog  |
| <ul style="list-style-type: none"> <li>Se necesita declarar las librerías de manera obligatoria para que pueda ser ejecutadas las líneas de código</li> </ul>                 | <ul style="list-style-type: none"> <li>No necesitan declararse que ya se ejecutan automáticamente. Carece de librerías.</li> </ul> |
| <b>COMPARACIÓN A NIVEL DE SOFTWARE</b>  |  |
| VHDL  | Verilog  |
| Al ser dos lenguajes estandarizados, estas herramientas coinciden en algunas características:   |  |
| <ul style="list-style-type: none"> <li>Son de fácil instalación, manipulación y aprendizaje.</li> <li>La mayoría de herramientas pueden manipular ambos lenguajes.</li> </ul> |  |

## 2.5. FPGAS LIBRES

Se denominan FPGAs libres a aquellas FPGAs cuya información técnica está publicada con licencia libre, con el suficiente detalle como para poder crear a partir de ella toolchains que permitan cerrar el ciclo completo de trabajo en FPGA.

### 2.5.1. INTRODUCCIÓN A LAS FPGAS LIBRES

Las FPGAs se conocen desde los años 80, pero era una tecnología privativa, lo que implicaba que sólo unos pocos tenían acceso a ella, y menos aún conocían sus detalles. A partir de mayo de 2015, el ingeniero austriaco Clifford Wolf hizo ingeniería inversa de la familia ICE40 de Lattice, publicó toda su información (proyecto Icestorm) y creó el primer sintetizador de FPGAs libre de la historia.

En la industria, las FPGA se utilizan como el paso previo a la manufactura de chips a la medida. Las grandes compañías diseñan sus chips en FPGAs, los prueban y una vez tienen un diseño listo lo envían a fabricar. Este proceso ahorra muchísimo tiempo, dinero y recursos.

Tal y como se ha mencionado en el apartado 2.3.3 de esta memoria configurar una FPGAs normalmente sigue un flujo de trabajo que requiere de varias etapas que se explican en el siguiente diagrama:

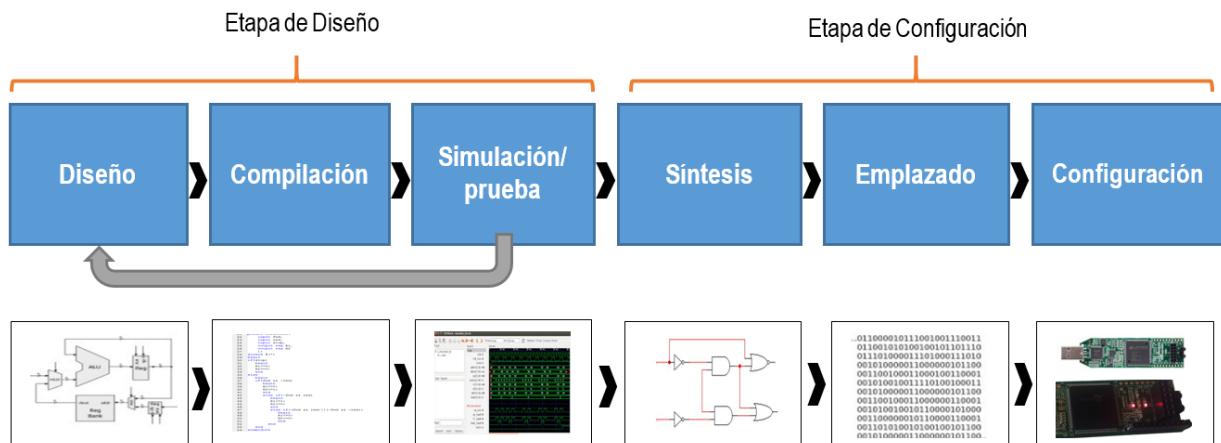


Figura 2.8: Flujo de trabajo en el diseño y configuración de una FPGA

- ❖ Diseño del circuito: Donde se diseña la disposición y conexiones de nuestro circuito.
- ❖ Compilación: Escritura de código HDL: Verilog o VHDL para describir las conexiones de nuestro circuito.
- ❖ Simulación: Antes de configurar el hardware hay que simularlo para detectar errores que pudieran dañar la placa.
- ❖ Síntesis: Aquí un software toma el código HDL de alto nivel y lo convierte a otro código HDL pero que ahora describe las conexiones en función de los elementos lógicos o analógicos disponibles para un modelo de FPGA en particular.
- ❖ Emplazado o ruteado (Place and Route o PnR): que esencialmente toma el código de HDL, busca las funciones disponibles en la FPGA y considerando su disposición física, trata de realizar las conexiones en función de lograr ciertos objetivos que pueden incluir: Utilizar el mínimo de componentes lógicos, utilizar rutas de conexiones más cortas, reducir tiempos de retardo en la propagación de señales, re-utilizar bloques existentes, etc.
- ❖ Generación del archivo de configuración (bitstream): Que esencialmente es un software que genera a partir del resultado del PnR el archivo de configuración específico o "bitstream" que se utilizará para indicar a la FPGA como se conectarán sus componentes internamente.

Hasta hace algunos años, ya existían varios proyectos Open-Source que tenían capacidad de lograr las etapas de diseño, compilación y simulación, mientras que las etapas de síntesis y emplazado eran logrados parcialmente por algunas herramientas Open-Source en base a las hojas técnicas del fabricante. Sin embargo la generación del BitStream era la pieza faltante para tener un flujo de trabajo completamente Open-Source. Esto era porque tanto el formato como las características de este archivo de configuración se han mantenido en secreto por prácticamente todos los fabricantes de FPGA, algunos de ellos incluso aplican estrategias para evitar que el mismo diseño genere el mismo archivo de configuración cada vez que se ejecuta la herramienta. Así que hasta hace muy poco era prácticamente imposible tener un flujo de trabajo completamente Open-Source para trabajar con FPGAs.

Esto fue así hasta que apareció el proyecto IceStorm [19] que sirve para generar archivos de configuración y cargar configuraciones a las FPGA ICE40 del fabricante Lattice y está basado en el trabajo de ingeniería inversa de dichos chips. La primera toolchain libre de la historia, para cerrar el ciclo completo de diseño en FPGA, desde el código HDL hasta la generación del bitstream, y su carga para la configuración de la FPGA.

### 2.5.2. PROYECTO ICESTORM

En mayo de 2015 ocurrió un hito histórico: se tuvieron por primera vez todas las herramientas necesarias para generar el bitstream a partir de código en Verilog usando sólo software libre, gracias al proyecto icestorm, liderado por Clifford Wolf.

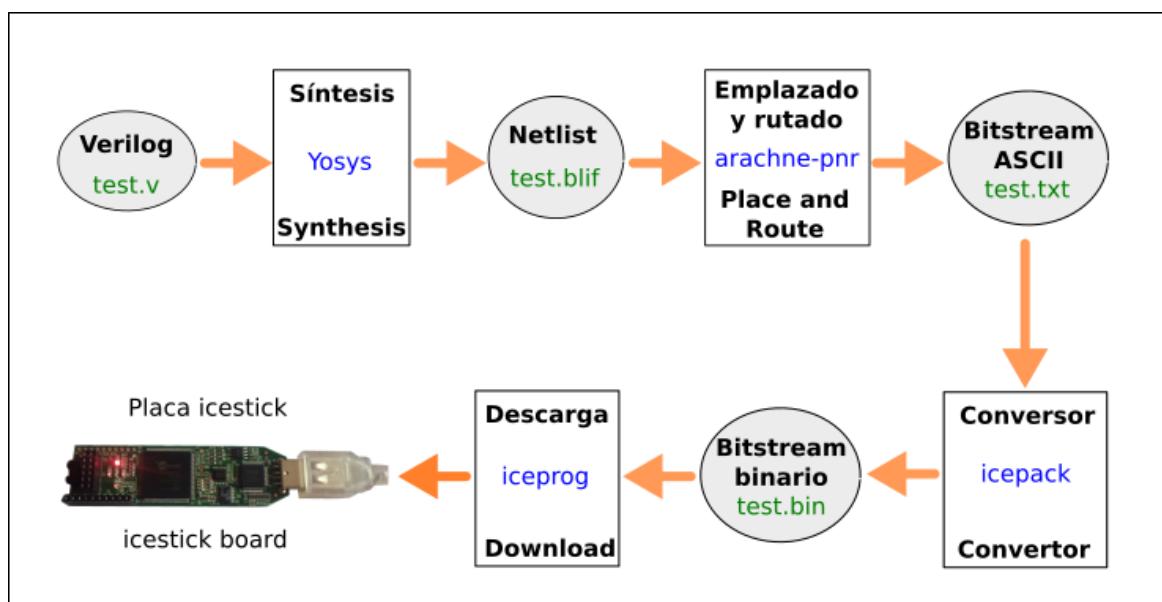
El proyecto IceStorm tiene como objetivo la ingeniería inversa y la documentación del formato bitstream de FPGA Lattice iCE40 y proporciona herramientas simples para analizar y crear archivos de flujo de bits. El flujo IceStorm (Yosys, Arachne-pnr e IceStorm) es un flujo Verilog-to-Bitstream de fuente totalmente abierta para la FPGA iCE40.

El enfoque del proyecto está en los chips iCE40 LP / HX 1K / 4K / 8K. (La mayor parte del trabajo se realizó en los modelos HX1K-TQ144 y HX8K-CT256). Los modelos iCE40 UltraPlus también son compatibles, incluidos DSP, osciladores, RGB y SPRAM. Los modelos iCE40 LM, Ultra y UltraLite aún no son compatibles.

El trabajo de ingeniería inversa no es nada trivial, es esencialmente tratar de conocer los detalles internos del chip FPGA sin tener acceso a ninguna documentación del fabricante y esto es lo que ha logrado el equipo de Mathias Lasser y Clifford Wolf.

El proyecto se ha basado en la Lattice iCE40 porque tiene una arquitectura muy minimalista con una estructura muy regular. No hay muchos tipos diferentes de mosaicos o unidades de funciones especiales. Esto lo hace ideal para la ingeniería inversa. Además, con Lattice iCEstick hay disponible una plataforma de desarrollo económica y fácil de usar, que hace que sea interesante para todo tipo de proyectos. (El iCEstick cuenta con un dispositivo HX1K. Lattice también vende una placa de arranque iCE40-HX8K con un chip HX8K).

En la siguiente figura se muestran las diferentes herramientas usadas en las etapas, y las extensiones de los archivos que se van generando:



*Figura 2.9: Herramientas usadas para crear el bitstream*  
*Fuente: <https://github.com/Obijuan/open-fpga-verilog-tutorial>*

Se parte de los ficheros fuente en **verilog (.v)**. Usando el sintetizador **Yosys**, se generan los ficheros **netlist (.blif)**. El emplazado y rutado se realiza con **arachne-pnr**, generándose el bitstream en formato **ASCII (.txt)**. Con **icepack** se crea el bitstream **binario (.bin)** que finalmente se envía a la FPGA con **iceprog**.

En la línea de comandos, los pasos a seguir para llevar el fichero test.v hasta la FPGA serían:

```
yosys -p "synth_ice40 -blif test.blif" test.v  
arachne-pnr -d 1k -p test.pcf test.blif -o test.txt  
icepack test.txt test.bin  
iceprog test.bin
```

### 2.5.3. FLUJO DE TRABAJO

- ❖ **Diseño:** Lápiz y papel o la pizarra
- ❖ **Escritura de HDL:** Por el momento las herramientas soportan únicamente el lenguaje Verilog. Para generar los archivos se pueden utilizar cualquier editor que soporte Verilog
- ❖ **Simulación:** Puede realizarse utilizando Icarus Verilog + GTKWave. Icarus verilog crea un fichero ejecutable a partir del código Verilog. Al ejecutarlo se realiza la simulación. Los resultados se vuelcan a un fichero .vcd que se visualiza con la herramienta Gt\_kwave. Esto nos permite inspeccionar las señales para comprobar su correcto funcionamiento
- ❖ **Síntesis:** Se realiza utilizando Yosys.
- ❖ **Ruteado:** Se realiza utilizando Arachne-pnr. Para hacer el mapeo es necesario un fichero con extensión .pcf, en el cual se asociarán las etiquetas utilizadas por el usuario con los pines de la FPGA.
- ❖ **Generación de Bitstream:** IceStorm, que también provee de algunas utilidades para guardar el bitstream directamente en la FPGA a través de un convertidor USB-Serie.

## 2.5.4. PLACAS CON FPGAS LIBRES

### *Lattice Icestick*

Esta es la placa que utilizó Clifford Wolf para hacer ingeniería inversa, y por eso es una pieza de coleccionista: es la primera placa en la que se sintetizó hardware usando sólo herramientas libres

|                      |  |
|----------------------|--|
| <b>Imagen</b>        |  |
| <b>Nombre</b>        | Icestick   |
| <b>Fabricante</b>    | Lattice  |
| <b>FPGA</b>          | ICE40HX-1K   |
| <b>Frecuencia</b>    | 12Mhz  |
| <b>Leds</b>          | 4 rojos, 1 verde   |
| <b>Observaciones</b> | Es la placa usada originariamente por Clifford Wolf para hacer ingeniería inversa  |

### *Placa Icezum Alhambra*

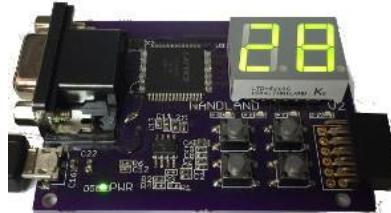
Es la placa que se usará en este proyecto

|                          |  |
|--------------------------|--|
| <b>Imagen</b>            |  |
| <b>Nombre</b>            | Icezum Alhambra  |
| <b>Autor</b>             | Eladio Delgado   |
| <b>Fabricante</b>        | Mareldem Technologies, Pinos del valle, Granada, España                              |
| <b>Donde conseguirla</b> | Alhambrabits   |
| <b>FPGA</b>              | ICE40Hx1K  |
| <b>Frecuencia</b>        | 12Mhz  |
| <b>Periféricos</b>       | 8 LEDs, 2 pulsadores, 4 canales A/D  |
| <b>Observaciones</b>     | Hardware libre. Compatible Arduino   |

***Lattice Breakout Board***

|                      |  |
|----------------------|--|
| <b>Imagen</b>        |  |
| <b>Nombre</b>        | Breakout Board   |
| <b>Fabricante</b>    | Lattice  |
| <b>FPGA</b>          | ICE40HX-8K CT256   |
| <b>Frecuencia</b>    | 12Mhz  |
| <b>Leds</b>          | 8 verdes   |
| <b>Observaciones</b> | Acceso a 40 pines de la FPGA   |

***Nandland Go Board***

|                      |  |
|----------------------|--|
| <b>Imagen</b>        |  |
| <b>Nombre</b>        | Nandland Go Board  |
| <b>Fabricante</b>    | Crowdfunding   |
| <b>FPGA</b>          | ICE40HX-1K   |
| <b>Frecuencia</b>    | 12Mhz  |
| <b>Leds</b>          | 8 verdes   |
| <b>Periféricos</b>   | 4 leds verdes, 2 Displays de 7 segmentos, 4 pulsadores, 1 conector VGA               |
| <b>Observaciones</b> | El conector de VGA integrado da mucho juego  |

**Kéfir I**

|                      |  |
|----------------------|--|
| <b>Imagen</b>        |  |
| <b>Nombre</b>        | Kéfir I  |
| <b>Autor</b>         | Salvador E. Tropea   |
| <b>Institución</b>   | Instituto Nacional de Tecnología Industrial (INTI), Argentina                      |
| <b>FPGA</b>          | iCE40HX4K-TQ144  |
| <b>Frecuencia</b>    | 24Mhz  |
| <b>Periféricos</b>   | 4 Capsenses  |
| <b>Observaciones</b> | Hardware libre. Kit educativo, inclusivo y recicitable. Compatible Arduino         |

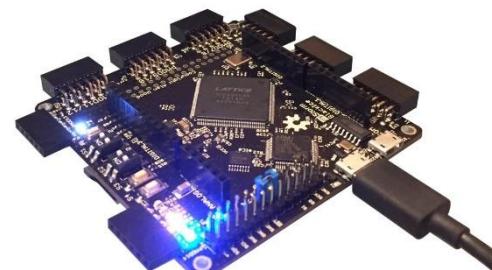
**Icoboard**

|                      |  |
|----------------------|--|
| <b>Imagen</b>        |  |
| <b>Nombre</b>        | Icoboard 1.0   |
| <b>Fabricante</b>    | Trenz Electrónic   |
| <b>FPGA</b>          | ICE40Hx8K  |
| <b>Frecuencia</b>    | 100Mhz   |
| <b>Periféricos</b>   | SRAM 8Mx16, 3 LEDs, 2 Pulsadores   |
| <b>Observaciones</b> | Sombrero para Raspberry PI   |

**Olimex iCE40HX1K-EVB**

|                      |  |
|----------------------|--|
| <b>Imagen</b>        |  |
| <b>Nombre</b>        | Olimex iCE40HX1K-EVB   |
| <b>Fabricante</b>    | Olimex   |
| <b>FPGA</b>          | ICE40Hx1K  |
| <b>Frecuencia</b>    | 100Mhz   |
| <b>Periféricos</b>   | SRAM 256Kx16, 2 LEDs, 2 Pulsadores   |
| <b>Observaciones</b> | Hardware libre   |

**Mystorm Blackice**

|                                   |  |
|-----------------------------------|--|
| <b>Imagen</b>                     |  |
| <b>Nombre</b>                     | Mystorm Blackice   |
| <b>Fabr(Gonzalez, s.f.)icante</b> | Mystorm, UK  |
| <b>FPGA</b>                       | ICE40Hx4K  |
| <b>Frecuencia</b>                 | 100Mhz   |
| <b>Periféricos</b>                | SRAM 256Kx16, 6 LEDs, 3 Pulsadores   |
| <b>Observaciones</b>              | Hardware libre   |

## 2.5.5. PLACA ICEZUM ALHAMBRA

IceZUM es una FPGA diseñada con los mismos patrones que Arduino, por lo que se puede utilizar para su ampliación todos los “escudos” (shields) que existen en el mercado para Arduino y es un dispositivo de los denominados de Open Hardware. [20]

El hardware de la placa IceZUM Alhambra cuenta con los siguientes dispositivos:

- ❖ **FPGA:** Lattice iCE40HX1K-TQ144 [21]. En esta FPGA hay 1280 Celdas lógicas, formadas cada una de ellas por una Look-up-table (LUT) y un flip-flop, 64 Kbits de memoria RAM y un PLL. Esta es una FPGA realmente pequeña y de bajo coste, con recursos lógicos reducidos pero suficientes para la realización de sistemas de nivel de complejidad reducido.
- ❖ **RELOJ:** Reloj MEMS a 12 MHz.
- ❖ **INTERRUPTOR:** Cuenta con un pequeño interruptor para desactivar los pines.
- ❖ **REGULADOR DE VOLTAJE:** Permite alimentar la placa con un rango de 6 a 17 voltios.
- ❖ **CORRIENTE DE ENTRADA:** Permite hasta una corriente de 3 amperios, útil para alimentar actuadores como, por ejemplo, motores de robots.
- ❖ **PINES:** Cuenta con 20 pines de Entrada/Salida a 5V y 10 a 3.3V.
- ❖ **INTERFAZ:** Programación mediante un USB micro-B, FTDI 2232H para programación e interfaz UART con un PC.
- ❖ **LEDS:** La placa cuenta con 8 leds que el usuario puede utilizar en sus programas.
- ❖ **PULSADORES:** Dispone de dos pulsadores utilizables por el usuario.
- ❖ **ANALÓGICO:** Cuenta con cuatro entradas analógicas conectadas a un convertidor analógico/digital ADC QFN16, con una precisión de 12 bits, conectado a la FPGA con un bus I2C propio.
- ❖ **PROTECCIÓN:** La placa cuenta con protección contra cortocircuitos y polaridad inversa.

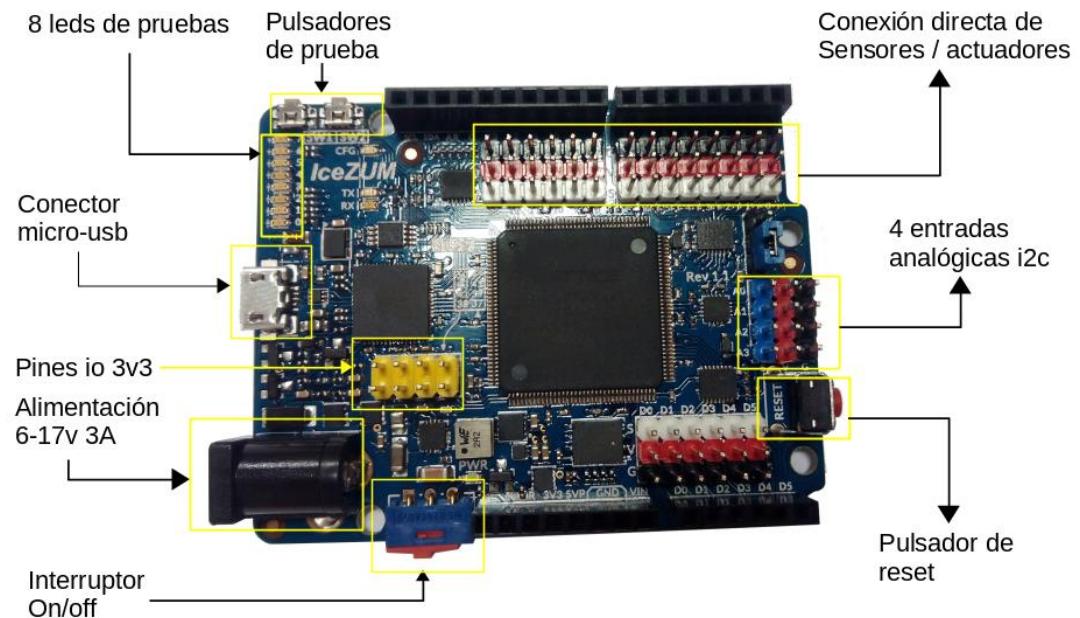


Figura 2.10: Placa IceZUM Alhambra

Fuente: FPGAwars

Además de todo esto, el fabricante proporciona datos sobre la placa, tales como el pinout, el PCF (fichero donde aparecen los pines restringidos de la placa y su uso), la hoja de características técnicas de la placa (datasheet), o reglas de diseño de la placa.

# iCEZUM

## Alhambra

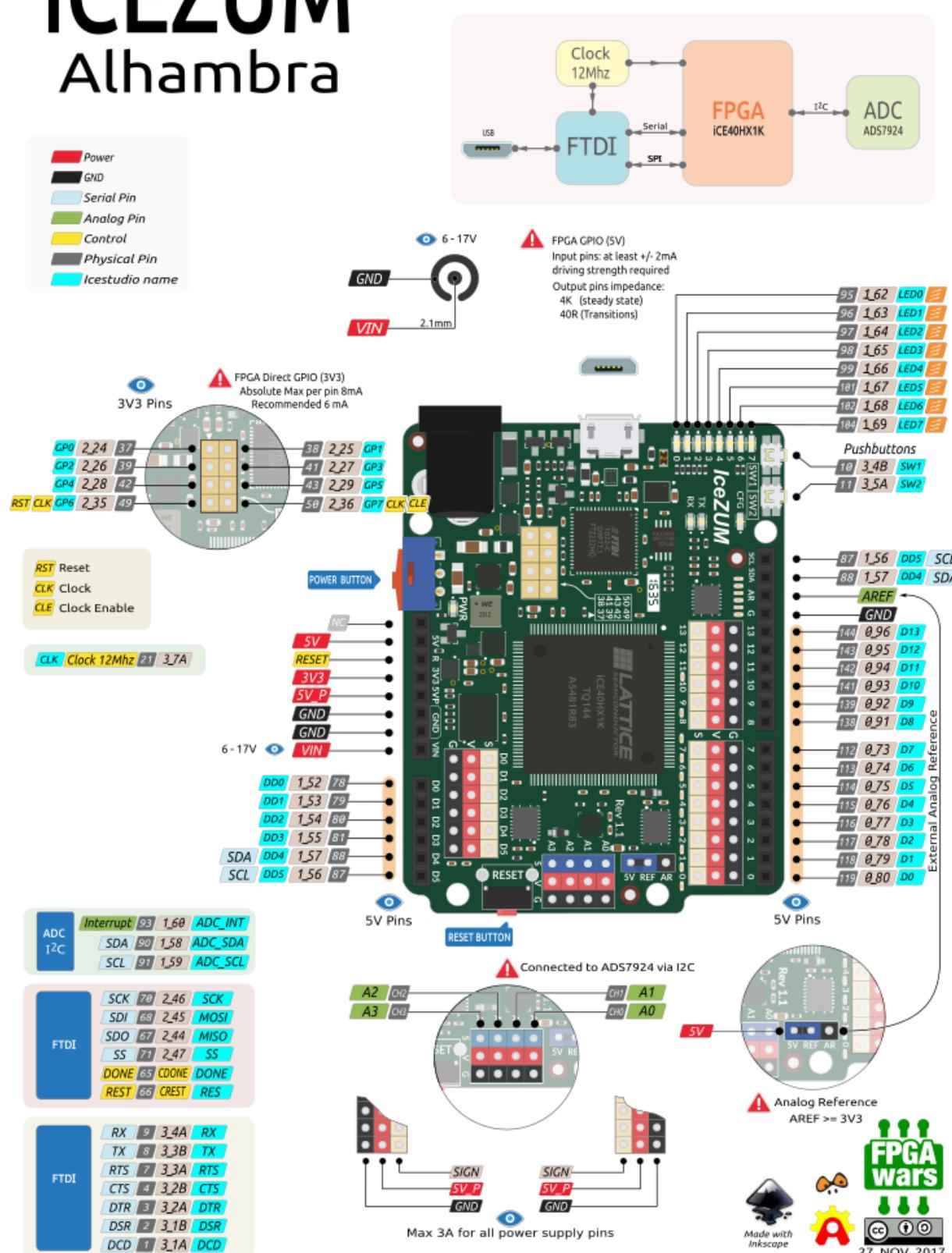


Figura 2.11: Pinout de la placa ICEZUM Alhambra.

Fuente: FPGAwars

Por otra parte, el entorno de desarrollo que suministra el fabricante de la placa es el denominado IceStudio [22]. Cabe destacar de él su máxima simplicidad, ofreciendo una interfaz que permite instalar todas las herramientas necesarias, tanto para la síntesis de HDL, como los drivers FTDI (Future Technology Devices International, circuito USB 2.0 a UART) necesarios para programar la IceZUM.

### 2.5.6. ENTORNO DE DESARROLLO ICESTUDIO

Icestudio es una herramienta para diseño y síntesis de circuitos digitales en FPGAs libres, creada por Jesús Arroyo. Está programada en nodejs. Es software libre y multiplataforma. Corre en los sistemas GNU/Linux, Mac OS y Windows. La interfaz ofrece una biblioteca de objetos ya creados, tales como:

- ❖ ELEMENTOS BÁSICOS: Pines de entrada, de salida, comentarios, constantes y módulos con las entradas y salidas programables en Verilog.
- ❖ BIT: Se puede añadir un bit tanto a 1 como a 0.
- ❖ CONFIGURACIÓN: En este menú desplegable se puede encontrar configuraciones pull-up para los pines y buffers tri-estado.
- ❖ LÓGICA: En este menú desplegable se pueden encontrar dispositivos tales como multiplexores, demultiplexores, módulos para controlar visualizadores de 7 segmentos, todo tipo de puertas lógicas y algunos tipos de flip-flops.

Este entorno de desarrollo también permite explotar el potencial de los lenguajes de desarrollo hardware exportando a Verilog la traducción del diseño que se realice, que puede ser modificado manualmente. Otra gran ventaja del entorno es que permite añadir otros diseños realizados anteriormente como módulos a nuevos diseños. Además, el entorno de desarrollo permite añadir nuevas colecciones (o bibliotecas), es decir, empaquetar todos los diseños en una colección para añadirlos a los menús desplegables, con lo que se pueden añadir colecciones de otros diseñadores con todos los módulos ya implementados que se necesiten.

Este IDE está construido en nodeJS sobre Icestorm, el cual es una secuencia de herramientas creadas mediante el uso de ingeniería inversa para poder generar bitstreams compatibles con las FPGAs Lattice iCE40.

Una vez generados los diseños con IceStudio, se puede proceder a su simulación. Para ello hay que hacer uso de la herramienta Icarus Verilog [23], que sintetiza y simula el diseño exportado anteriormente. Antes de esto también hay que exportar el testbench (fichero de test) en IceStudio, que es otro archivo en Verilog necesario para declarar los estímulos de entrada para el diseño.

Una vez completado el proceso, se puede analizar el comportamiento del diseño mediante un analizador temporal, como, por ejemplo GTKWave. Con este software y el sistema IceZUM, se pueden generar los diseños, simularlos, y cargarlos y ejecutarlos en la placa.

IceStudio permite encapsular diseños para ser usados como módulos en otros diseños más complejos, este problema se puede solventar diseñando previamente los módulos que los estudiantes puedan necesitar para llevar a cabo las prácticas propuestas. De esta forma, los estudiantes sólo tendrán que seleccionar y conectar los módulos necesarios para implementar la solución de cada una de las prácticas.

### 3. ARQUITECTURA Y DESARROLLO DEL SISTEMA

#### 3.1. DEFINICIÓN DE LA METODOLOGÍA DE TRABAJO

Para desarrollar el laboratorio remoto basado en la IceZum Alhambra lo primero que se va a definir son las etapas que se van a seguir para la consecución de los objetivos de este proyecto.

- 1- Análisis de alternativas, se analiza las distintas alternativas presentes para implementar un laboratorio remoto y se justifica la solución adoptada
- 2- Diseño de laboratorio remoto de la FPGA libre IceZum Alhambra, esta etapa consiste en el montaje de la maqueta. En esta fase se valorará si se conectan o no periféricos a la placa, y en caso de conectarse, que tipo de periféricos se conectan y cuantos, para ello se realizará un estudio de alternativas de las diferentes posibilidades.
- 3- Por otro lado se procede a la realización de la aplicación web, que estará basada en el software de otros laboratorios ya existentes en el departamento de Ingeniería Eléctrica, Electrónica y Control de la UNED
- 4- Una vez implementado el entorno web, y elaborada la maqueta se procederá a elaborar los materiales didácticos necesarios para poder utilizar el laboratorio remoto por parte de los usuarios. En este apartado se propondrán una serie de prácticas a realizar.
- 5- Como fase final se procederá a instalar y montaje del laboratorio en la ubicación que el departamento de Ingeniería Eléctrica, Electrónica y Control de la UNED tiene destinada para estos equipos.
- 6- Una vez finalizado el proceso se realizará un presupuesto del equipo.

Todas estas etapas se encuentran interconectadas entre si y a medida que se ha avanzado en la implementación de cada una de ellas han surgido cuestiones que han modificado las etapas anteriores.

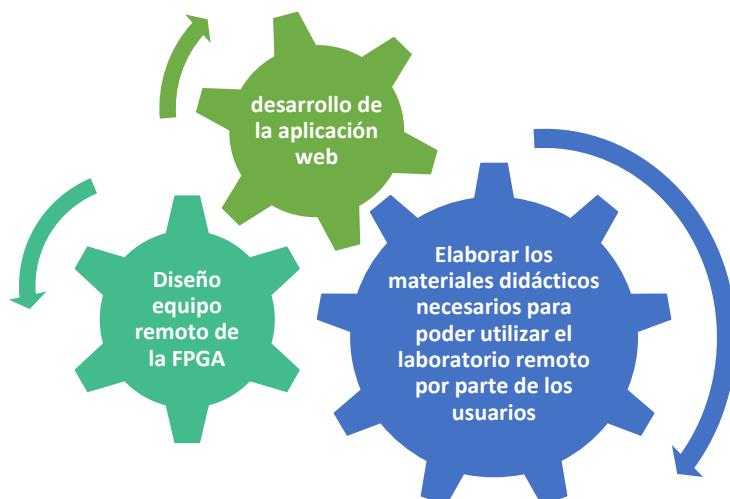


Figura 3.1: Metodología de trabajo

### 3.2. ANÁLISIS DE ALTERNATIVAS

Una de las primeras decisiones a tomar antes de acometer el desarrollo de un laboratorio remoto es decidir entre las diferentes tecnologías existentes para la implantación de este tipo de equipos [24]. Las principales tecnologías de diseño utilizadas son las basadas en **escritorio remoto** y **las basadas en servicio web**.

Para estudiar las diferencias entre estas tecnologías se ha realizado la *TABLA 3.1*, donde se analiza una serie de factores, con la finalidad de valorar la mejor tecnología para implantar el diseño del laboratorio remoto de la FPGA IceZUM Alhambra

**TABLA 3.1: ESTUDIO DE ALTERNATIVAS: LABORATORIO REMOTO FPGA IceZUM Alhambra**

| TECNOLOGÍAS DE DISEÑO |  |  |
|-----------------------|--|--|
| Factores evaluados    | Escritorio remoto  | Programación web   |
|                       | Son estaciones de trabajo conectadas a la FPGA que abren una conexión al escritorio remoto | Constituidos, por un lado cliente y otro servidor. Implementados con los lenguajes de programación php, SQL, html y Java |
|                       | Facilidad de creación  | más sencilla   |
|                       | Control por usuarios   | control sobre la estación de trabajo   |
|                       | Modos de operación (local o remoto)  | un modo operación  |
|                       | Monitorización de resultados   | por cámara web   |
|                       | Control de accesos   | disponen de servicio   |
|                       | Software libre   | no se dispone de código abierto  |
|                       | Planificación  | proporciona servicios de planificación   |
| VALOR*                | 17   | 22   |

\*Ponderación Rojo = 1; Amarillo = 2; Verde = 3. A mayor valor mejor

Según la *TABLA 3.1*, la mejor tecnología de diseño para implementar el laboratorio remoto objeto de este proyecto, es la basada en la programación web. Uno de los principales motivos para decantarse por esta tecnología y no la otra, es porque existe un código abierto sobre el cual comenzar el diseño, por otro lado, la autora de este trabajo dispone de conocimientos de los lenguajes de programación requeridos para para el desarrollo web.

También hay que indicar que los laboratorios remotos basados en servicio web nos ofrece una serie de ventajas sobre los basados en escritorio remoto [24]:

- ❖ Los estudiantes no tienen control sobre la estación de trabajo y la FPGA, evitando que realicen actividades peligrosas para el equipo.
- ❖ Modularidad del diseño.
- ❖ Guardan Registro del uso

### 3.3. DISEÑO ARQUITECTÓNICO

---

En el epígrafe 2.1.3 Arquitectura de un laboratorio remoto, se ha explicado brevemente las distintas partes de las cuales está compuesto un laboratorio remoto quedando esquematizadas en la Figura 2.3 donde se muestra el esquema general de la arquitectura de un LABORATORIO REMOTO.

En este epígrafe se procede a explicar la arquitectura desarrollada para la realizar el laboratorio remoto de la FPGA IceZUM Alhambra.

Como ya se ha dicho el uso del laboratorio implica una arquitectura cliente (usuario) –servidor (laboratorio). En este tipo de arquitectura se encuentran dos partes diferenciadas:

- ❖ Cliente (usuario): es la parte local donde se encuentra el equipo del usuario, el navegador web y las aplicaciones informáticas que utilice para generar los archivos a subir al servidor (IceStudio, Apio, o cualquier compilador de verilog)
- ❖ Servidor (laboratorio): es la parte remota, donde se tendrá el equipo en el cual se encuentra instalado el laboratorio remoto, con los servidores de web, de base de datos y de video necesarios para el uso remoto del equipo. En este equipo estará instalada la página web (desarrollada con php, html, css, JavaScript y SQL y el software necesario para la síntesis y configuración de la FPGA IceZUM Alhambra, en nuestro caso se instalará el IceStudio. Ya que lleva integradas la herramientas toolchain-icestorm.

En la siguiente imagen se realiza un esquema de la arquitectura del sistema:

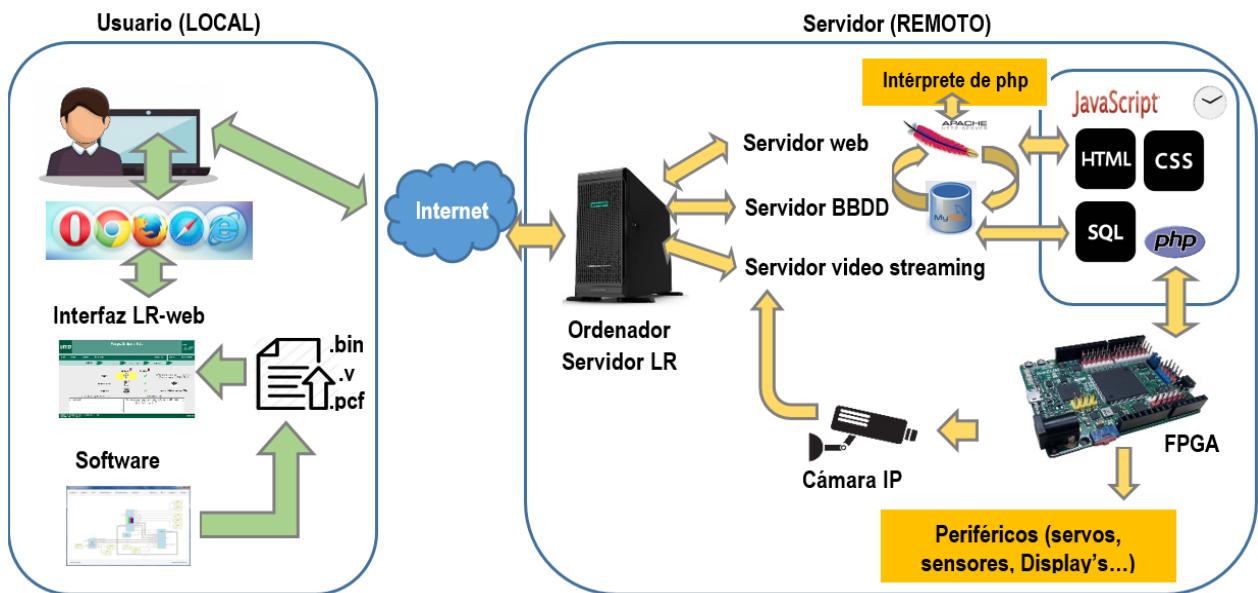


Figura 3.2: Arquitectura del Laboratorio Remoto FPGA IceZUM Alhambra

## 3.4. DISEÑO DE LABORATORIO REMOTO DE LA FPGA LIBRE ICEZUM ALHAMBRA

### 3.4.1. ANÁLISIS DE ALTERNATIVAS

El montaje del laboratorio se puede plantear de tres formas diferentes:

- ❖ Conectar la placa sin ningún periférico
- ❖ Configurar el montaje para realizar una serie de prácticas determinadas
- ❖ Configurar el montaje con la finalidad de dar la opción al usuario de plantear diferentes combinaciones, para ello se conectarán todos los periféricos que se puedan en la placa.

Para estudiar las principales diferencias entre estas opciones se ha realizado la TABLA 3.2, donde se analiza una serie de factores, con la finalidad de valorar cual sería el mejor montaje a diseñar para el laboratorio remoto de la FPGA IceZUM Alhambra.

A la vista del análisis realizado se llega a la conclusión que aunque es interesante implementar un laboratorio con una variedad de periféricos conectados porque aumentaría la versatilidad del equipo, esto complicaría el uso de laboratorio remoto por parte de usuarios inexpertos en esta clase de dispositivos.

Se opta por realizar un montaje en el que se conecten una serie limitada de periféricos de fácil uso, por parte de los alumnos, pero que a la vez de la posibilidad de combinar varios dispositivos para implementar diseños más sofisticados. IceStudio permite el diseño sencillo de circuitos digitales, en una interfaz amigable.

**TABLA 3.2: DISEÑO SISTEMA REAL: MONTAJE LABORATORIO  
MODO DE OPERACIÓN**

| Descripción        |                            | Solo la FPGA IceZUM Alhambra  | La FPGA con periféricos prestablecidos por las prácticas a realizar                                 | La FPGA con una serie de periféricos que permitan realizar diversos tipos de prácticas  |
|--------------------|----------------------------|---|---|---|
| Factores evaluados | Facilidad del montaje      | Solo se conecta al servidor remoto la placa IceZUM sin ningún periférico.   | Se conecta a la placa los periféricos necesarios para realizar las prácticas programadas.           | Se conecta a la placa una serie de periféricos de diferente tipología para permitir la implementación de diferentes configuraciones   |
|                    | Valor didáctico            | Más sencillo, solo es conectar la placa.  | Se monta, simplemente, lo que hace falta para las prácticas, no hay que pensar en las alternativas. | Es el más complejo, se intentaría ocupar todos los pines de la placa con la finalidad de aumentar la versatilidad   |
|                    | Versatilidad               | Sería bajo, solo se podría implementar configuraciones en las que interviniieran los LEDs que incorpora la FPGA IceZUM Alhambra | Se aumenta el valor didáctico porque el usuario puede controlar elementos externos.                 | El valor sería alto, se tendrían más opciones y combinaciones para la configuración de la FPGA IceZUM Alhambra  |
|                    | Facilidad de Uso           | Baja, solo se podrían usar los LED internos.  | Media, se puede hacer algo más que en caso anterior, aunque estaría limitado.                       | ALTA, es la opción que más opciones presenta, se puede adaptar a diferentes niveles de aprendizaje  |
|                    | Interacción con el usuario | Alta, sería fácil de usar   | Media, al tener periféricos instalados, se complicaría la programación                              | BAJA, la compilación se podría complicar mucho, sobre todo si se quiere realizar comunicaciones en serie  |
|                    | Nivel de aprendizaje       | Baja, solo se tendría acceso a los LED integrados en la placa de entrenamiento  | Media, el usuario puede utilizar varios periféricos y realizar diseños más interactivos             | Alta, aunque complica bastante la programación, y podría aparecer el riesgo de que un usuario inexperto utilizara los periféricos de forma inapropiada aumentando el riesgo de fallo de la FPGA |
|                    | Dificultad de programación | Baja  | Media   | Alto, se puede interactuar con un mayor variedad de periféricos, y realizar programaciones más complejas  |
| VALOR*             |                            | 13  | 14  | 13  |

\*Ponderación Rojo = 1; Amarillo = 2; Verde = 3. A mayor valor mejor

### 3.4.2. IMPLEMENTACIÓN DEL LABORATORIO REMOTO

#### Pines de la Icezum Alhambra

La placa Icezum Alhambra tiene 4 tipos de pines para conectarse al exterior, resumidos en este diagrama:

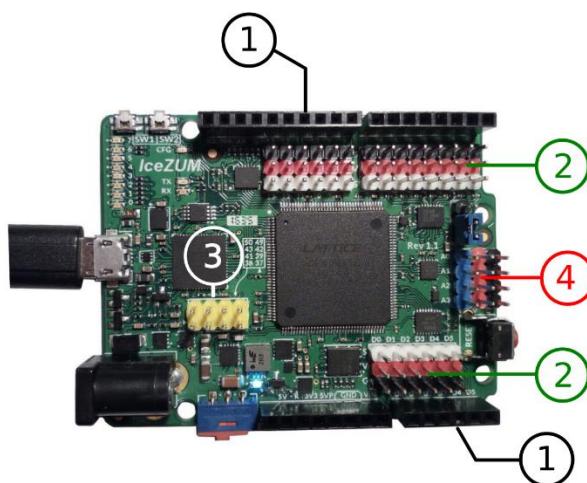


Figura 3.3: Pines de la IceZUM Alhambra.

Fuente: FPGAwars

Estos conectores son:

1. **Conectores de I/O hembra de 5 Voltios:** Son los típicos conectores de Arduino que sirven para conectar cables macho y shields de expansión. Se encuentran en la parte superior e inferior de la placa, como se muestra en la figura. En total hay 31 pines, de los cuales 20 son de I/O= de 5v. Los pines de I/O superiores se denominan D13 - D0 en Icestudio, y los inferiores DD5 - DD0. Todos estos pines tienen la característica de que son de 5v.
2. **Conectores de I/O macho de 5 Voltios, con alimentación y masa:** Son 20 pines de I/O, machos, duplicados de los 20 pines I/O de los conectores hembra de 5v. Cada uno de estos pines incluye uno adicional de alimentación (+5v) y masa (GND). En total son tres pines separados por colores. El pin blanco contiene la señal de I/O, el rojo la alimentación de 5v y el negro la masa (GND). Tienen esta disposición para poder conectar directamente servos y sensores. Se denominan igual que sus duplicados hembra: D0 - D13 para los de la parte superior y DD0 - DD5 en la parte inferior.
3. **Conectores de I/O macho de 3.3 Voltios:** Son 8 pines macho dispuestos en dos filas de 4 pines, de color amarillo. Los 8 pines están conectados directamente a pines de la FPGA, que trabajan a 3.3V. Los pines hembra y macho anteriores están conectados a la FPGA a través de conversores de nivel. Se denotan con la nomenclatura GP0 - GP7.
4. **Conectores macho para entradas analógicas:** La Icezum Alhambra tiene 4 entradas analógicas situadas en la parte derecha, accesibles a través de pines macho, Cada entrada analógica (Azul)

tiene un pin adicional de alimentación (rojo, 5v) y otro de masa (negro, GND). Estas tiras de 3 pines permiten conectar directamente sensores de Luz o potenciómetros. La FPGA NO tiene entradas analógicas, por lo que en la Icezum Alhambra hay un conversor Analógico-digital (A/D) que se conecta a la FPGA a través de un bus I2C. Por ello, para leer las entradas analógicas es preciso incluir en nuestros diseños de un circuito capaz de leer el bus I2C

### **Leds externos**

En el laboratorio remoto, se han conectado **LEDs externos** a los Conectores de I/O macho de 5 Voltios, con alimentación y masa. En concreto se han colocado 4 a los conectores D13, D12, D11 y D10.

Los conectores de 5v están conectados a los de la FPGA a través de unos conversores de nivel (3.3v  $\leftrightarrow$  5v). Para que el LED se ilumine con más intensidad se realiza la conexión a través de un transistor. El esquema de conexión es el siguiente:

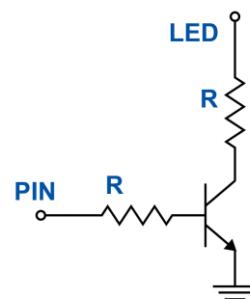


Figura 3.4: Esquema de conexión LED externo.

Al enviarse un 1 por el pin de la FPGA, se activa el transistor, que funciona como un interruptor, conectando la resistencia a GND y cerrando el circuito.

### **Servos**

En el laboratorio remoto, se han instalado Servos conectados a los Conectores de I/O macho de 5 Voltios, con alimentación y masa. En concreto se han colocado 2 a los conectores DD0 y DD1.

Los servos son actuadores cuyo eje de salida se puede fijar a una posición concreta. Se usan mucho en robots educativos y de investigación, para implementar pinzas o articulaciones. Se presenta el controlador más simple para moverlos: El servoBit, que permite posicionar el servo en dos posiciones, según el bit recibido por su entrada

Para llevar la cabeza del servo a una posición determinada hay que enviar una señal de control especial. Se trata de un pulso de 5v, de anchura variable, que se envía periódicamente, cada 20ms (frecuencia de 50Hz)

El ancho del pulso es el que determina la posición a la que se moverá el servo. Los valores dependen del modelo de servo, pero típicamente un pulso de 1ms lleva el servo al extremo derecho y uno de 2ms al izquierdo. Valores entre 1ms y 2ms lo posicionarán linealmente entre 0 y 180 grados

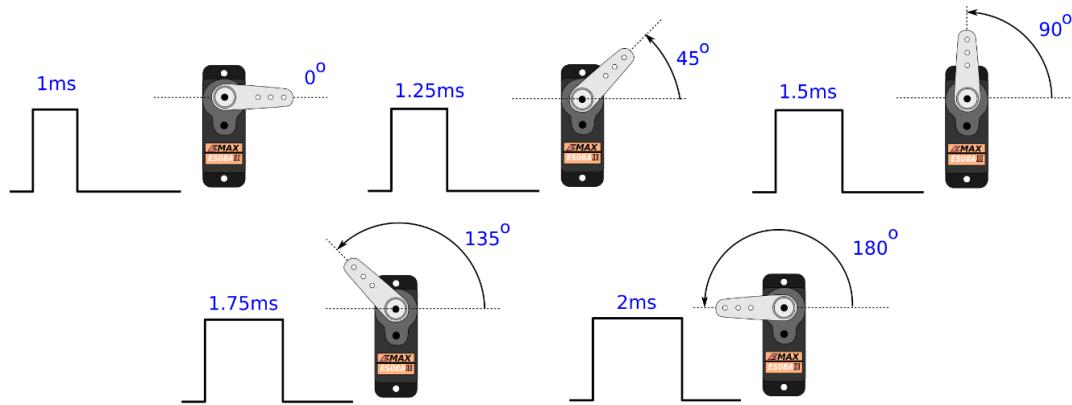


Figura 3.5: Posición servos.

Fuente: FPGAwars

Los valores exactos dependen del modelo de servo utilizado. Lo importante es saber que el ángulo en el que se posiciona el servo es directamente proporcional al tiempo del ancho del pulso, y que este pulso se tiene que repetir periódicamente. Si se deja de repetir, el servo deja de ejercer fuerza y se puede mover con la mano a cualquier posición

Los servos se pueden colocar en cualquier posición dentro de su rango de 180 grados. Sin embargo, se va a utilizar el más sencillo: dos posiciones, separadas un ángulo de 90 grados. El controlador será mínimo: un bloque que tiene una única entrada y que según su valor mueve el servo a una posición (0) o a otra (1).

Al recibir un bit a 0 por su entrada, se posicionará en la derecha. La posición absoluta depende de cómo esté colocada su cabeza. Al recibir un 1 se moverá 90 grados en sentido antihorario

Cada modelo de servo tiene unas características de la señal de control diferentes, por eso para poder usar los servos instalados en el laboratorio remoto hay que cargar la colección IceZUM LR UNED, que se puede descargar de la página web del Laboratorio Remoto

### Sensores de infrarrojos

En el laboratorio remoto, se han conectado 2 sensores infrarrojos, justo debajo del recorrido que realizan los Servos, éstos están conectados a los Conectores de I/O macho de 5 Voltios, con alimentación y masa. En concreto **se han colocado a los conectores DD2 y DD3**.

Los sensores de infrarrojos (IR) permiten detectar la presencia de un objeto que está delante de ellos, a un centímetro aproximadamente. Constan de un emisor y un receptor.

El emisor es un LED infrarrojo. Es un LED que en vez de emitir luz visible, la emite en el espectro infrarrojo, que no es visible. El emisor está constantemente emitiendo, cuando el sensor está alimentado. Esto se puede observar mirando el sensor a través de la cámara del teléfono móvil: se puede apreciar un color azulado/violeta.

El receptor es un fototransistor, que se activa cuando recibe luz infrarroja. La detección de objetos se hace por reflexión. La luz infrarroja que está constantemente generando el emisor se refleja en el objeto y llega al receptor, que nos devuelve un 1 (objeto detectado). Si por el contrario no hay objeto, la luz infrarroja se pierde y no regresa al receptor. En este caso el sensor nos devuelve un 0 (Objeto no detectado)

Estos sensores también se pueden utilizar para distinguir el blanco del negro en una superficie plana, y construir así robots que puedan por ejemplo seguir un camino negro. Cuando el sensor está sobre blanco, la luz del emisor se refleja completamente, devolviendo un 1. Cuando está sobre la parte negra, no hay reflexión. El negro absorbe la luz, y no la refleja, por lo que no llega nada al receptor y devuelve un 0.

## ***Esquema de conexión***

En la Figura 3.7: Esquema de conexión del Laboratorio Remoto IceZUM Alhambra. Figura 3.7, se muestra el esquema de conexión implementado en el Laboratorio IceZUM Alhambra. Se puede ver que se ha dejado espacio para una posible ampliación del equipo, en esta ampliación (Figura 3.6) se podría conectar un display LCD y sensores analógicos.

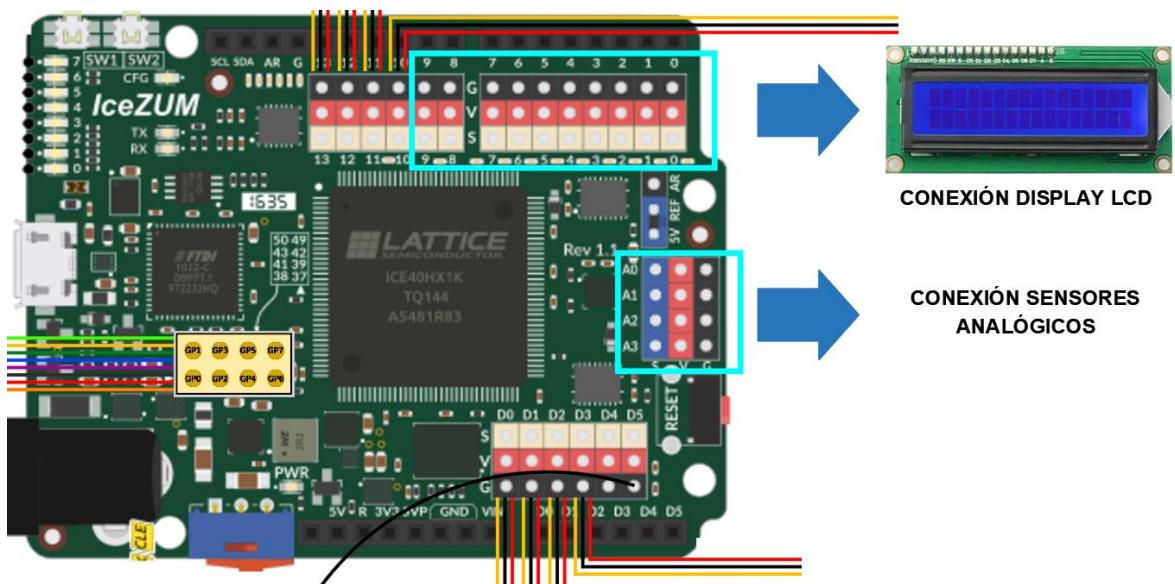


Figura 3.6: Futuras ampliaciones Laboratorio Remoto IceZUM Alhambra.

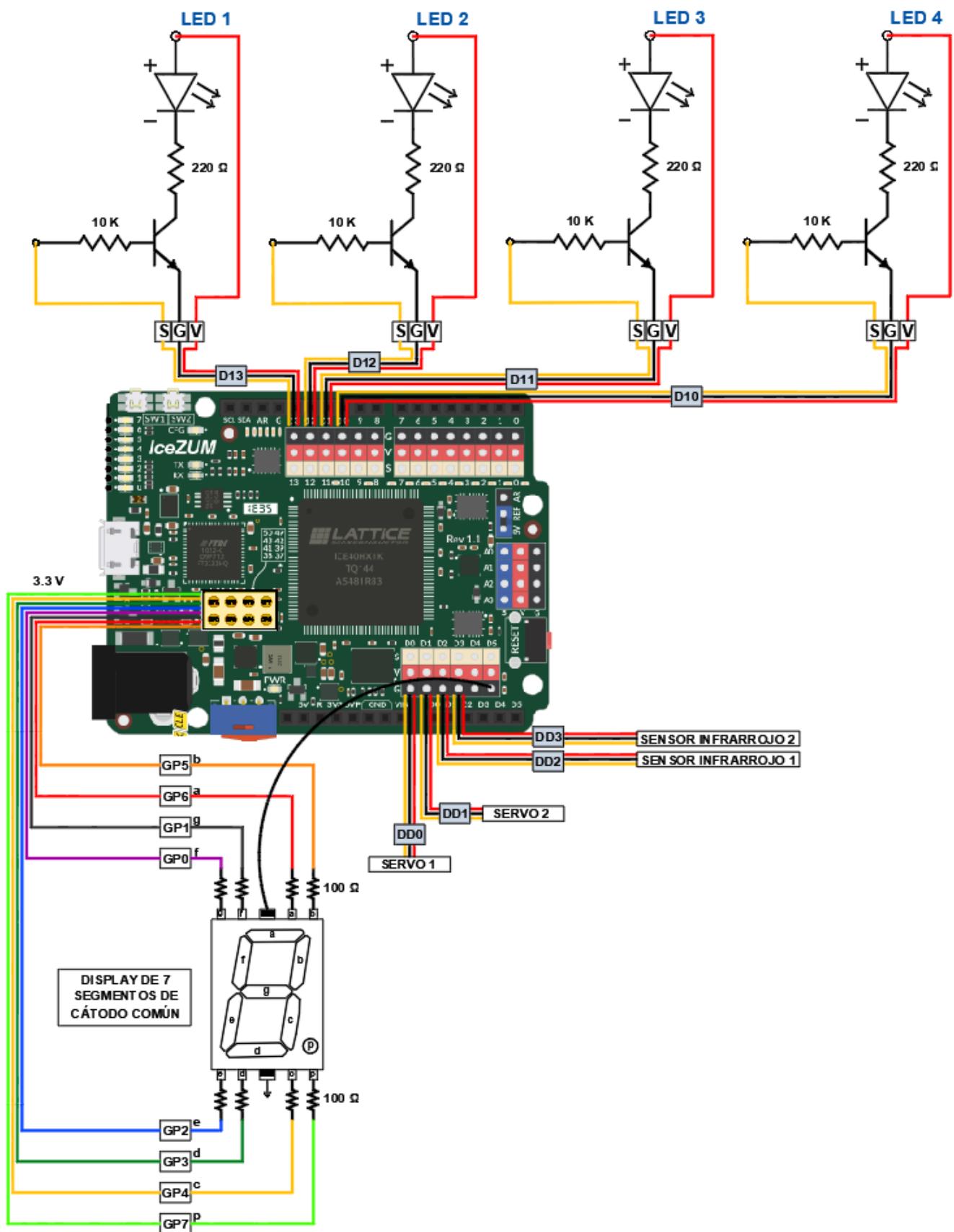


Figura 3.7: Esquema de conexión del Laboratorio Remoto IceZUM Alhambra.

### 3.4.3. CÁMARA IP

Con la finalidad de ver el resultado de las prácticas a realizar en el laboratorio remoto, se instalará una cámara que permita ver el montaje y su funcionamiento al usuario del laboratorio remoto.

Por otro lado, la instalación de la cámara puede plantear un par de alternativas posibles a la hora de implementar la página web:

- ❖ Visualización de la imagen proporcionada por la cámara después de haberse realizado la carga de la placa
- ❖ Visualización de la imagen proporcionada por la cámara a la vez que se realiza la carga de la misma

Para estudiar las principales diferencias entre estas opciones se ha realizado la TABLA 2.1, donde se analiza una serie de factores, con la finalidad de valorar cual sería la mejor opción para el laboratorio remoto de la FPGA IceZUM Alhambra.

**TABLA 3.3: ALTERNATIVAS DE ACCESO A MONITORIZACIÓN DE RESULTADOS CÁMARA**

|                           | <b>Descripción</b>           | <b>No hay monitorización</b>  | <b>Después de la carga</b>   | <b>Mientras se carga</b>   |
|---------------------------|------------------------------|---|--|--|
|                           |                              | Esta opción no se valora, por no tener sentido, es necesario poder ver el resultado de la carga en la placa. De otra forma, este laboratorio carecería de sentido | Se programa la web de forma que se accede a la imagen generada por la cámara IP una vez se ha cargado con éxito la placa | Se programa la web de forma que se accede a la imagen generada por la cámara IP mientras se carga la placa |
| <b>Factores evaluados</b> | <b>Facilidad de creación</b> |   | dificultad media, hay que implementar bucles   | dificultad baja, no es necesaria la implementación de bloques  |
|                           | <b>Valor didáctico</b>       |   | No se ve la evolución de la carga, solo el resultado si es correcto  | Se puede ver la evolución de la carga de la placa mientras se produce                                      |
|                           | <b>Interacción</b>           |   | Menor interacción, se ve la placa si la carga es correcta  | Mayor interacción, se ve lo que pasa cuando no se carga bien, y da error                                   |
|                           |                              |   | <b>6</b>   | <b>9</b>   |

\*Ponderación **Rojo = 1**; **Amarillo = 2**; **Verde = 3**. A mayor valor mejor

Se considera que la mejor opción es implementar una página donde se puede ver como se carga la placa, porque mejora la interacción del usuario con el equipo.

## 3.5. DESARROLLO DE UN ENTORNO WEB PARA CONTROL REMOTO

El entorno web del laboratorio está basado en el *Laboratorio de FPGA's*<sup>3</sup> instalado en el Departamento de Ingeniería Eléctrica, Electrónica, Control, Telemática y Química Aplicada a la Ingeniería de la UNED.

### 3.5.1. ANÁLISIS DE ALTERNATIVAS

Antes de comenzar con la implementación de la web, se ha analizado las ventajas y desventajas que presentaba los distintos modos de operación en el diseño del flujo de trabajo en la página web

Para estudiar las principales diferencias entre estas opciones plantadas se ha realizado la TABLA 3.4, donde se analiza una serie de factores, con la finalidad de valorar cual sería el mejor diseño para flujo de trabajo de web del laboratorio remoto de la FPGA IceZUM Alhambra.

TABLA 3.4: DISEÑO DE FUNCIONAMIENTO DE LA WEB – MODO DE OPERACIÓN

|                    |                       | Local   | Remoto   | Ambos  |
|--------------------|-----------------------|---|--|--|
| Factores evaluados | Facilidad de creación | Más sencillo, hay que programar menos páginas   | hay que implementar más páginas para mostrar los pasos de la compilación remota  | es el más complejo de implementar, puesto que realiza los dos tipos de compilación |
|                    | Valor didáctico       | El usuario solo tendría que subir el archivo .bin y ver el resultado en la cámara web. No tendría la opción de ver el proceso de síntesis del bitstream | el usuario subiría los archivos .v y .pcf, y a partir de ellos se generaría de forma remota el archivo .bin, sería capaz de entender el proceso de síntesis y diferenciarlo del de carga de la placa | el usuario tendría la opción de hacerlo de las dos formas                          |

<sup>3</sup> [http://www.ieec.uned.es/Labs\\_remotos.htm](http://www.ieec.uned.es/Labs_remotos.htm). Dicho equipo, es un desarrollo basado en Xilinx Spartan 3AN FPGA para llevar a cabo prácticas de programación VHDL a distancia con visualización a través de cámara web.

TABLA 3.4: DISEÑO DE FUNCIONAMIENTO DE LA WEB – MODO DE OPERACIÓN

|                     |   |   |                           |
|---------------------|---|---|---------------------------|
| <b>Versatilidad</b> | Poca versatilidad, es preciso tener instalado la tool machine en el ordenador | se tiene la opción de compilar en verilog el programa en cualquier programa, no hace falta instalar las herramientas Icestorm o IceStudio en el ordenador del usuario | ofrece todas las opciones |
| <b>VALOR*</b>       | 5   | 6   | 7                         |

\*Ponderación **Rojo = 1**; **Amarillo = 2**; **Verde = 3**. A mayor valor mejor

Tal y como se desprende de la tabla anterior la opción más completa es la implementación de una web que nos dé la opción de la compilación local y la compilación remota, al contar con un diseño previo, resulta bastante sencillo implementar esta opción para este caso, así que es la que se desarrollará en los siguientes apartados.

### 3.5.2. PERFILES DE ACCESO

Se definen dos perfiles de acceso al laboratorio:

#### Perfil de usuario

Se trata un para acceso general, compilación de diseños y uso del programador con supervisión de la placa IceZUM Alhambra con la cámara IP. Este será el que usen los alumnos para la realización de prácticas, con generación de informes de actividad. Este perfil carece de privilegios para ver los reportes ni para modificar ningún aspecto del funcionamiento del laboratorio, aunque si puede acceder a los archivos que genera la ejecución de los ejecutables del proyecto *Icestorm*, utilizados para realizar la compilación remota y la carga del bitstream para la configuración de la FPGA.

#### Perfil Administrador

Este perfil se utiliza para la gestión del laboratorio, modificación de los usuarios con privilegios, generación de informes y supervisión de la actividad de los alumnos. Para poder utilizar este perfil se necesita tener privilegios y estar incluido en la tabla de '*administracion*' de la base de datos. Si se accede un usuario administrador se entra en la parte de gestión del laboratorio.

En esta parte se podrá:

- ❖ ver los registros creados referentes a la actividad de los alumnos
- ❖ responder y enviar mensajes,
- ❖ hacer y restaurar copias de seguridad,

- ❖ añadir o eliminar nuevos administradores
- ❖ supervisar la actividad de la placa IceZUM Alhambra través de la cámara IP,
- ❖ y entrar al laboratorio como un usuario más para verificar el funcionamiento del laboratorio o desarrollo y comprobación de nuevas prácticas y efectuar pruebas antes de ponerlas a disposición de los alumnos.

Para la utilización del Laboratorio Remoto, por parte de los usuarios generales del laboratorio, sin privilegios de administrador se ha elaborado un **Manual de Usuario adjunto como apéndice** a esta memoria.

Por otro lado para la gestión del laboratorio por parte de los administradores del mismo, también se ha realizado un **Manual de Administrador que se adjunta como apéndice** a esta memoria.

### 3.5.3. PLANIFICACIÓN

Para gestionar los accesos al laboratorio remoto se ha mantenido la arquitectura de acceso libre que estaba implementada en el código usado como base para elaborar la web del presente trabajo.

Con acceso libre se quiere decir que no se realiza reserva de hora o día para la utilización del laboratorio remoto, con la finalidad de no forzar a los alumnos a tener un tiempo fijado o limitaciones para hacer las prácticas.

Al permitir el servidor accesos múltiples y las herramientas de *IceStorm* ejecuciones simultáneas de los comandos, no se ha considerado necesario un sistema de reservas, sino un sistema abierto.

Para no sobrecargar el equipo servidor, y dado que esta compartido con otros laboratorios y el portal del departamento, para verificar el comportamiento del equipo, se ha limitado el acceso a diez usuarios trabajando y otros diez en espera, ya que más de ese número sería una espera demasiado larga. Para usar el programador, como es lógico suponer, no se puede permitir a más de una persona el uso para evitar conflictos en la programación del dispositivo. Por lo tanto, existen dos colas diferentes aunque la forma de gestionarlas es la misma, a través de sus respectivas tablas en la base de datos.

### 3.5.4. DIAGRAMA DEL PROCESO

A continuación se muestra el diagrama del proceso que sigue la página web

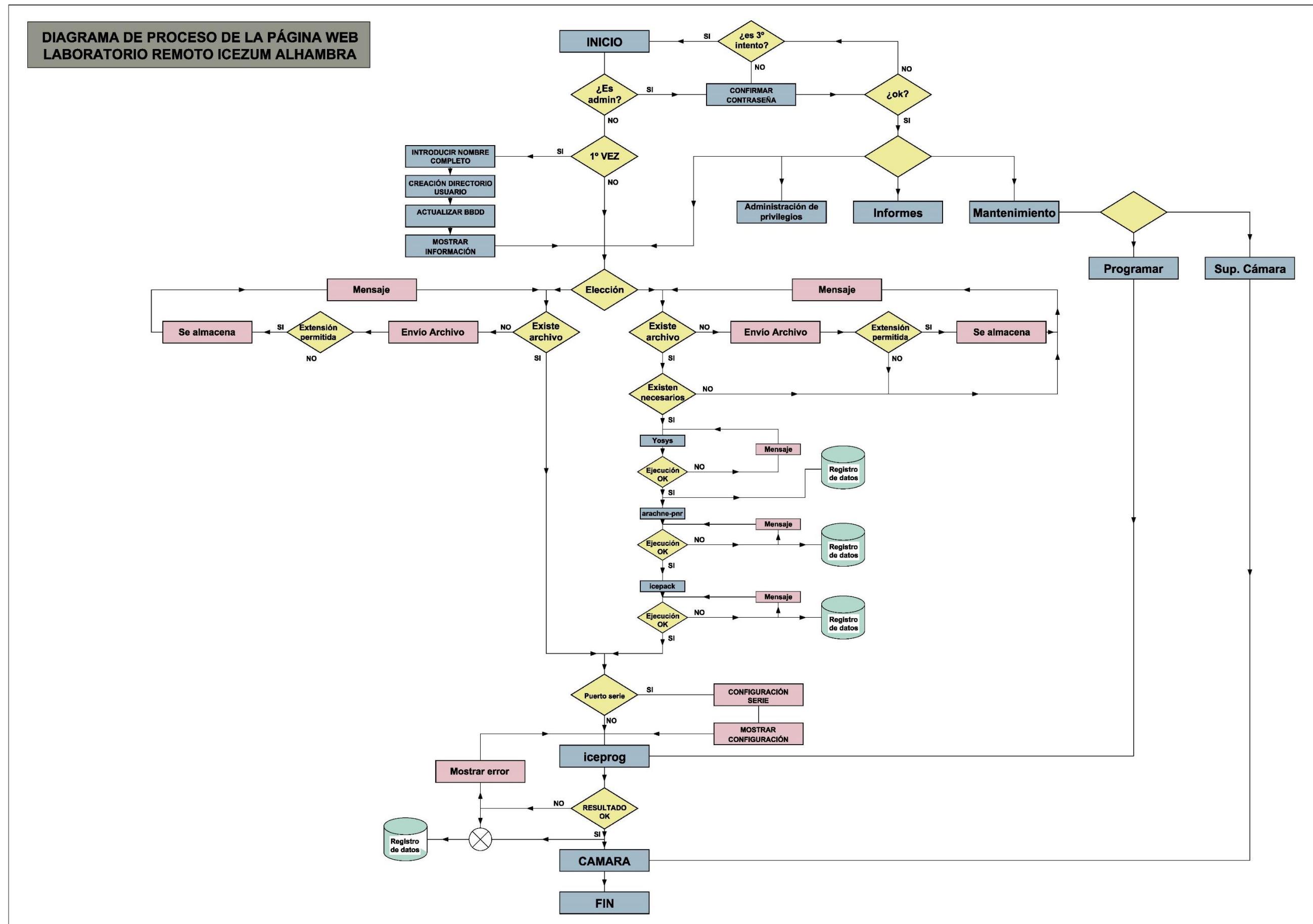


Figura 3.8: Diagrama de proceso de la página web - Laboratorio Remoto IceZUM Alhambra

Una vez se accede se introduce el identificador, si no es la primera vez, se pasa directamente a la elección de práctica, donde el usuario puede elegir entre la compilación local o la compilación remota.

#### **Compilación remota**

En el caso de elegir la compilación remota, se pasa al envío de archivos y la comprobación de la extensión y el tamaño; no están permitidos archivos diferentes de los necesarios para la compilación ni de tamaño superior a 1Mbits, tamaño más que suficiente para cualquier proyecto.

Las extensiones permitidas son .v (archivo con código verilog) y prf (asociación de etiquetas con los pines de la FPGA).

Una vez recibidos y comprobados los archivos se continúa con la compilación remota de forma secuencial pasando por cada una de las etapas, hasta generar el archivo bitstream

#### **Compilación local**

Si se elige la compilación remota, se pasa directamente a una hoja donde se sube el archivo con extensión .bin, una vez subido el archivo se pasa a cargar el archivo bitstream a la FPGA.

A partir de aquí el proceso es común para ambos tipos de compilación, se configura el puerto serie si es necesario y se programa la FPGA, una vez se pulsa el botón "programar placa" se puede ver la evolución del programa en esta misma página.

Si el usuario que accede a la web es identificado como administrador, se comprueba la contraseña, y si esta es correcta, se entra en la página de administrador desde donde se puede administrar privilegios, imprimir informes o mantener la web.

Si el usuario es la primera vez que accede a la página web, se pasa a registrarla, y se redirige a la página de información.

Toda la actividad del usuario en el laboratorio queda registrada de manera que se pueda revisar datos tales como los intentos realizados para cada práctica, los errores obtenidos en cada una, el tiempo empleado dentro del laboratorio, etc.

### **3.5.5. ESTRUCTURA DE LA PROGRAMACIÓN**

Un esquema con la estructura de las páginas que forman el laboratorio remoto se puede ver en la siguiente figura:

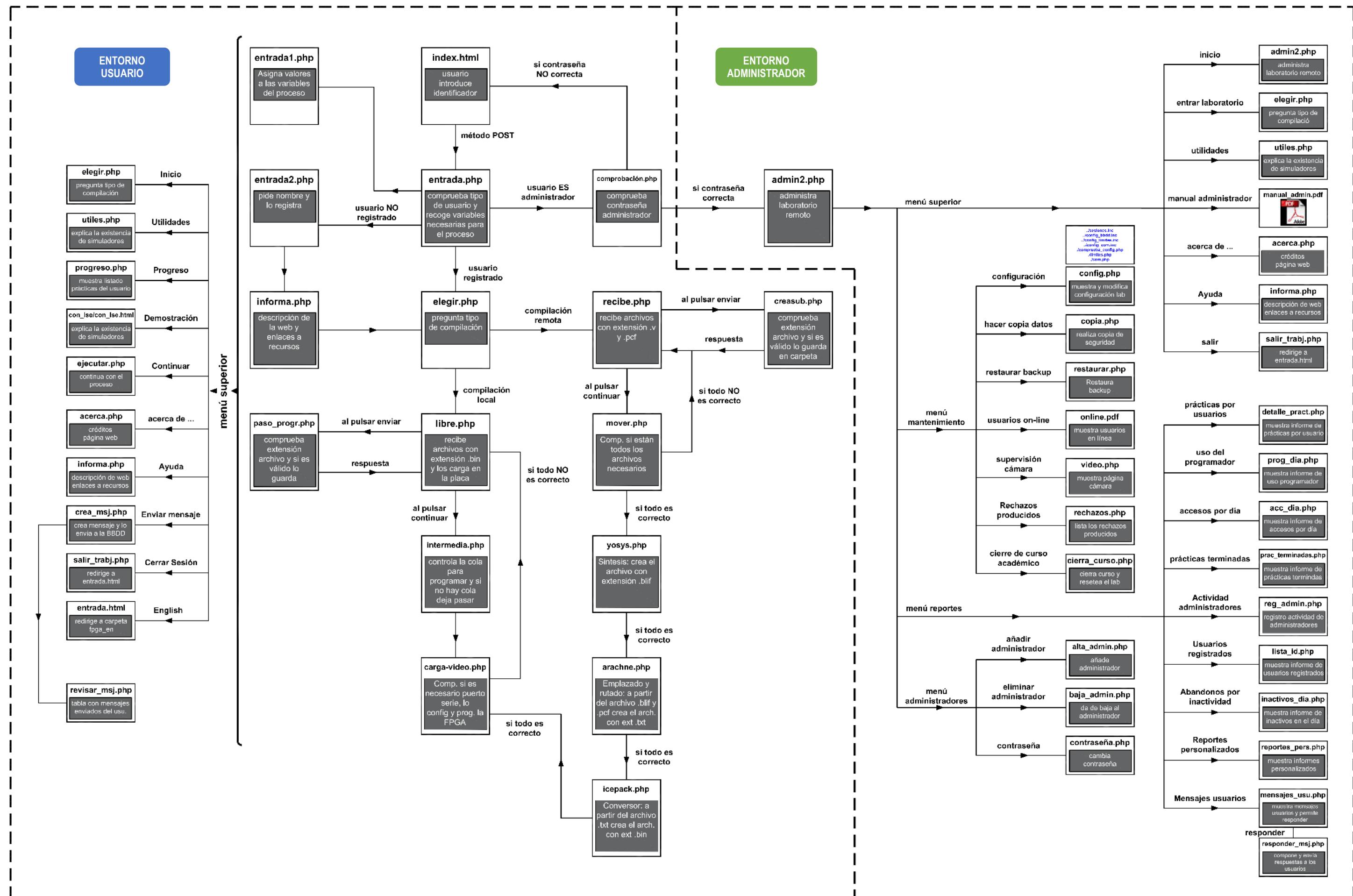


Figura 3.9: Estructura programación de la página web - Laboratorio Remoto IceZUM Alhambra

Como lenguaje de programación para el desarrollo de la interfaz web se ha elegido PHP por varias razones:

- ❖ Se necesitaba poder ejecutar las aplicaciones externas de IceStudio, requisito que cumple sobradamente PHP.
- ❖ PHP configurado como módulo de apache funciona con gran velocidad de ejecución y sin crear demoras en la maquina donde se ejecuta, corriendo sin problema en cualquier sistema operativo sin tener que modificar el código escrito.
- ❖ Es de código abierto, lo que facilita la detección y corrección de problemas y bugs que puedan aparecer, ofreciendo gran estabilidad al sistema.
- ❖ Tiene diferentes niveles de seguridad configurables en el archivo php.ini.
- ❖ Puede interactuar con muchos motores de bases de datos sin tener que instalar nuevas expansiones, aunque se puede instalar el modulo ODBC para las situaciones que lo requieran.
- ❖ Se pueden añadir fácilmente módulos
- ❖ Es ejecutado en el lado del servidor, lo que no provoca carga en la transferencia de datos en la conexión.

Para el funcionamiento tanto de la compilación remota como de la carga del archivo bitstream para la configuración de la fpga, se debe instalar el **IceStudio** en el propio equipo donde se encuentra la página web, de esta forma las llamadas a los ejecutables de “**toolchain-icestorm**” que realiza la web a través de la programación en PHP se hacen apuntando a la localización de dichos ejecutables.

La función exec de PHP es la que se utiliza para ejecutar las herramientas *icestrom* integradas en el *Icestudio*. Esta función nos permite ejecutar un programa externo, a continuación se muestra la ejecución del *iceprog.exe*, que es el ejecutable de las herramientas *icestorm* que carga el archivo bitstream para configurar la FPGA:

```
<?php
$archivo_cargado=$_SESSION['archivo']; //se refiere al archivo que se ha subido o compilado
$carpeta=$_SESSION['proyecto']; //se refiere a la carpeta de los archivos subidos o compilados
$sin_ext=$_SESSION['usuario']; // se refiere a la carpeta del usuario del Laboratorio Remoto
$ruta_carga='./fpga/'.$sin_ext.'/'.$carpeta.'/'.$_SESSION['archivo'];
$envio='C://Users/XXXX/.icesudio/api/packages/toolchain-icestorm/bin/iceprog '.$ruta_carga;
exec($envio,$comando,$codigo)
?>
```

| Descripción   |
|---|
| <pre>string exec ( string \$command [, array &amp;\$output [, int &amp;\$return_var ]] )</pre>  |
| exec() ejecuta el <b>comando</b> dado.  |
| Parámetros  |
| <p><b>command</b><br/>El comando que será ejecutado.</p> <p><b>output</b><br/>Si el argumento <b>output</b> está presente, entonces el array especificado será llenado con cada línea de la salida del comando. El espacio en blanco extra, como <code>\n</code>, no es incluido en este array. Note que si el array ya contiene algunos elementos, exec() anexará sus resultados al final del array. Si no desea que la función anexe los elementos, use <a href="#">unset()</a> sobre el array antes de pasársela a exec().</p> <p><b>return_var</b><br/>Si el argumento <b>return_var</b> está presente junto con el argumento <b>output</b>, entonces el status de retorno del comando ejecutado será escrito en esta variable.</p> |

Figura 3.10. Descripción y parámetros de la función exec (PHP).

Fuente: <http://php.net/manual/es/function.exec.php>

Además de PHP se ha usado Javascript para realizar dos tareas fundamentales, una la de temporizar la permanencia de un usuario en una página. Si se mantiene sin hacer nada durante cinco minutos es borrada la sesión y dirigido a la página de entrada para dejar sitio a otro usuario. A los cuatro minutos aparece una ventana emergente que le avisa acompañada de un sonido. La forma de conseguirlo es añadiendo las líneas de código con llamadas a las funciones correspondientes acompañadas del tiempo a las que tienen que ejecutarse:

```
setTimeout ('avivar()', 250000);
setTimeout ('cerrar()', 295000);
```

No son exactamente cinco minutos porque ese es el tiempo que tiene configurado PHP como límite para borrar las sesiones de usuarios que han abandonado el laboratorio de forma irregular.

La otra función importante de javascript es detectar el sistema operativo y el navegador del usuario que accede. Esta tarea es necesaria para poder dirigir correctamente al usuario hacia la página de descarga del plugin de vlc usado para reproducir el streaming de video, los usuarios de Windows necesitan descargar un archivo diferente que los usuarios de sistemas Unix o Mac OS.

También hay otras partes de código javascript pero no tienen relevancia en cuanto al funcionamiento del laboratorio se refiere, son usados para mostrar imágenes al pasar por un ícono, por ejemplo al pasar el ratón por los iconos de interrogación de las páginas de compilación remota aparecen imágenes con aclaraciones.

### 3.5.6. ÁRBOL DE DIRECTORIOS

El laboratorio remoto está integrado en el portal de nuevas tecnologías para la enseñanza de ingeniería. Este portal está contenido en una carpeta llamada “laboratorios” en la unidad C:\ del disco duro del ordenador servidor. Dentro de él, en subdirectorios separados, están los diferentes laboratorios que se han implementado para la realización de prácticas remotas en diferentes dispositivos.

En el caso que nos ocupa, el subdirectorio que contiene el laboratorio remoto de la FPGA IceZUM Alhambra se llama **fpga\_icezum**. En dicha carpeta se encuentran todos los archivos necesarios para el funcionamiento de la página web, y el trabajo realizado por los usuarios.

Para almacenar el trabajo realizado por los usuarios se crea la carpeta “fpga”, que es la que se utilizará para realizar la copia de seguridad, dentro de la carpeta “fpga”, con el identificador que cada usuario dará la primera vez que entre al laboratorio remoto, se creará un directorio de trabajo local para él, y dentro de este subcarpetas con el nombre de la práctica en la que trabaje, archivos enviados al servidor y los generados por la compilación remota al ejecutar las herramientas icestorm para sintetizar el bitstream. De esta manera se facilita la localización de las prácticas realizadas por cualquier usuario partiendo del identificador.

Además de los archivos creados en la compilación remota, en cada carpeta de usuario se crea un archivo en formato XML con el mismo nombre que el identificador, donde se almacenan los accesos producidos, tanto el día de acceso como el tiempo de permanencia en el laboratorio, prácticas elegidas y las operaciones realizadas con su consiguiente resultado, por ejemplo ejecución de una etapa y si el resultado ha sido correcto o no.

Por ejemplo, para localizar la práctica 2 del usuario 'usulab1', habría que acceder al directorio general 'fpga' y busca la carpeta 'usulab1' y allí estaría la práctica 2 con todos los archivos.

### 3.5.7. BASE DE DATOS

Se ha creado una base de datos con ocho tablas, llamada **fpga\_izezum**, la función de cada tabla es la siguiente:

#### Entradas

Esta tabla se utiliza para llevar un registro de usuarios que han accedido al laboratorio, de modo que cuando un usuario accede por primera vez, no estará incluido en la tabla, y entonces se le solicitará el nombre completo y se guardará junto con el identificador para un posterior uso en los informes que se generen. Cuando accede nuevamente su identificador estará en la tabla porque se habrá guardado

#### Sesiones

Guarda toda la información referente a la sesión de usuario. PHP asigna un número aleatorio y único para identificar cada acceso que se produce al servidor web. Ese número de sesión es guardado en la tabla y es un elemento clave para relacionar esta tabla con las demás. Junto a este número se

guardan todas las variables usadas a lo largo de todo el proceso y mientras dure la conexión al servidor. Cuando se abandona el laboratorio de forma regular, mediante el ícono de salida presente en cada página, se borra la entrada de la tabla que coincide con el número de sesión. Si la salida se ha producido de forma no regular, cuando pasa un tiempo definido en la configuración de PHP automáticamente se borra para limpiar los accesos no útiles y que aparezcan en las tablas las permanencias reales.

### **Administración**

---

Guarda los identificadores de aquellos que serán usuarios con privilegios del laboratorio junto con sus contraseñas. Al producirse un acceso, el identificador introducido es buscado primero en esta tabla, y de encontrarse, el usuario será considerado administrador y direccionado a la parte de gestión, previa comprobación de la contraseña asociada. Si no se encuentra en esta tabla, es considerado usuario normal, y comprobada la existencia en la tabla 'entradas' y sigue el proceso normal de entrada.

### **Programar**

---

Se usa para gestionar las peticiones de uso del programador. Según se van produciendo las peticiones, en caso de ser más de una, cuando se ha terminado la ejecución del comando 'iceprog.exe' se van añadiendo filas a la tabla, y el orden de espera será el orden en que se han insertado las filas. Como se ha indicado antes, el identificador de sesión es el enlace con la tabla que guarda las sesiones de PHP y la que relaciona la sesión con el usuario. Conforme van saliendo usuarios del laboratorio, las filas van disminuyendo y cuando un usuario llega a la primera fila pasa a cargar el archivo bitstream para configurar la FPGA.

### **Esperas**

---

Se usa para gestionar las solicitudes de entrada al laboratorio, en caso de que produzcan más peticiones de las que se permiten simultáneamente. Por razones de carga al servidor se han limitado los accesos a diez, sobrepasado ese número se crea una lista de espera de otros diez y si se produjera la circunstancia de que hubiera más peticiones serían rechazados. El orden de la lista es el mismo con el que se van insertado los registros en la tabla, y según vayan saliendo los usuarios, la cola de espera va avanzando y van pasando a la página de elección de la práctica cuando se ha llegado a la primera posición de la tabla y existen menos de diez usuarios trabajando.

### **Colas**

---

Cada vez que se produce una entrada que desborda el límite de accesos, se inserta un registro en esta tabla con fecha y hora producido, el usuario que ha sido enviado a esperar y si la espera se produce en la página de entrada o en la espera para programar. La función de esta tabla es tener una referencia de cuantos accesos hay que no se pueden servir para poder elevar el número de entradas permitidas o poder gestionarlas de otra forma.

## **Rechazos**

Su finalidad es guardar un registro de las veces que algún usuario es rechazado por estar completo tanto el límite de usuarios trabajando como la cola de espera. No se almacena el nombre del usuario rechazado porque no se considera relevante a quien se ha rechazado, solo se guarda en qué momento se ha producido para posterior análisis en el caso en que sean numerosos y sea necesario una distribución de los accesos de los alumnos o el aumento del límite.

## **Mensajes**

Se utiliza para gestionar los mensajes que los usuarios pueden enviar a los administradores/profesores, bien para consultas o comentarios, los generados automáticamente cuando se produce un acceso duplicado del mismo identificador y aquellos que los gestores del laboratorio/profesores pueden enviar a los usuarios/alumnos respectivamente.

Cuando es un mensaje en dirección usuario/gestor, se almacena la fecha de inserción del registro y cuando es leído se marca como tal y se inserta la fecha de lectura y la respuesta ofrecida, si la hay. Si es en dirección contraria puede haber dos casos, que sea un solo destinatario o sea a todos los usuarios; la diferencia en cuanto a la base de datos se refiere, es la forma en que se da como leído, que en el primer caso es cuando acceda el usuario destinatario y en el segundo es a los dos días de enviado el mensaje.

Desde mysql se tiene la facilidad de hacer un volcado de la generación de las tablas, que en caso de necesidad se puede restaurar con solo ejecutar un comando.

| Tabla                      | Columnas  |
|----------------------------|---|
| fpga_icezum administracion | usuario : varchar(15), contrasena : varchar(15)   |
| fpga_icezum sesiones       | sesion_id : varchar(26), validez : int(11), valor : text  |
| fpga_icezum programar      | id : int(2), sesion_id : varchar(26)  |
| fpga_icezum colas          | sesion_id : varchar(26), dia : varchar(10), hora : int(11), usuario : varchar(30), tipo : varchar(10), duracion : int(11)   |
| fpga_icezum esperas        | id : int(2), sesion_id : varchar(26)  |
| fpga_icezum entradas       | identificador : varchar(30), nombre : varchar(40)   |
| fpga_icezum rechazos       | id : int(11), fecha-hora : timestamp  |
| fpga_icezum mensajes       | Idmensaje : int(5), IdUsuario : varchar(10), mensaje : varchar(150), fecha_insercion : varchar(10), leido : varchar(2), fecha_leitura : varchar(10), lector : varchar(10), respuesta : varchar(150) |

Figura 3.11: Estructura de la base de datos del Laboratorio Remoto FPGA IceZUM Alhambra

## 3.6. DESARROLLO DE MATERIALES DIDÁCTICOS

El desarrollo de los materiales didácticos es una de las piezas clave de un laboratorio remoto, ya que el alumno tiene que realizar las prácticas en su casa sin ayuda del profesor, por ello los materiales facilitados al alumno deben de ser claros y desarrollados para sea capaz de realizar las prácticas en su casa y no quedarse atascado en alguna de las etapas del proceso.

Se ha planteado el proyecto para que los alumnos realicen la programación de un dispositivo lógico FPGA de forma remota, de manera que no entrañe dificultad alguna, ya que la única herramienta de la que disponen es internet.

En primer lugar, antes de proponer las prácticas a realizar, se analiza qué se puede enseñar con el Laboratorio Remoto que se ha implementado. En dicho análisis se plantearán las posibilidades de aprendizaje que ofrece la aplicación remota.

### 3.6.1. ALTERNATIVAS DE APRENDIZAJE.

Una de las mayores ventajas de las FPGAs es la capacidad que tienen para ser reprogramadas. Es en esta flexibilidad donde se encuentra el mayor atractivo de esta tecnología, permitiendo obtener velocidades hardware con flexibilidad software, ya que puede utilizarse el mismo hardware para varias aplicaciones cambiando exclusivamente su programación interna.

Esta tecnología nos ofrece un cambio de paradigma: hardware que puede modificarse vía software. De la misma manera que un ordenador puede escribir datos en una memoria, ese mismo equipo puede grabar un determinado circuito dentro de un chip, y cambiarlo tantas veces como se quiera. El circuito se modifica internamente, sin la necesidad de que haya cambios físicos externos, de ahí su gran utilidad tanto a nivel industrial como educativo.

La aplicación remota objeto de esta memoria, nos ofrece la posibilidad de mejorar, los aspectos prácticos de la enseñanza de:

- Algebra de Boole → puertas lógicas → Lógica combinacional
- Lenguaje HDL – Verilog
- Diseño de circuitos Digitales (con ICESTUDIO y con VERILOG)
- Tecnología y Organización de Computadores, entre otras.

Las prácticas a realizar dependerán de diferentes factores como los conocimientos previos del alumno, los objetivos del aprendizaje, el tiempo disponible, las herramientas disponibles (accesibilidad, facilidad de uso...), dificultad o criterios de evaluación. En la TABLA 3.5 se realiza un análisis de cada uno de ellos:

TABLA 3.5: ANÁLISIS DE LOS FACTORES QUE INFLUYEN EN EL DISEÑO DE LAS PRÁCTICAS

|                    |                                  |   |
|--------------------|----------------------------------|---|
| Factores evaluados | Conocimientos previos del alumno | <p>Dependiendo del conocimiento previo del alumno, se podrán plantear prácticas con un nivel de dificultad mayor o menor, estos conocimientos dependerán principalmente del curso en el que se encuentren y de los estudios que estén realizando.</p> <p>Aunque el Laboratorio Remoto IceZUM Alhambra sirve para introducir conceptos básicos como el <b>Algebra de Boole</b>, <b>puertas lógicas</b> o la <b>lógica combinacional</b>, para diseñar las prácticas <b>se va a suponer que los alumnos cuentan con estos conocimientos</b>.</p>  |
|                    | Objetivos del aprendizaje        | <p>Los objetivos del aprendizaje dependerán de la asignatura en la que se engloben las prácticas, de los conceptos teóricos que se quieran reforzar o de las herramientas disponibles para poder realizarlas.</p> <p>Como el Laboratorio remoto IceZUM Alhambra se encontrará instalado en el departamento de Ingeniería Eléctrica, Electrónica, Control, Telemática y Química Aplicada a la Ingeniería de la UNED, <b>las prácticas tratarán del diseño de circuitos digitales</b>.</p>  |
|                    | Tiempo disponible                | <p>El tiempo disponible es un factor muy importante a tener en cuenta, ya que disponer de más tiempo permitiría profundizar más en los conceptos estudiados y afianzar las ideas principales. En un principio dado el carácter a distancia de las enseñanzas que se imparten en la UNED, se podría pensar que un alumno dispone de un tiempo ilimitado para realizar las prácticas, pero normalmente estos alumnos compatibilizan sus estudios con otras actividades.</p> <p>Proponer un guion de prácticas demasiado extenso, podría producir una baja participación a consecuencia de la desmoralización del alumnado. Por esta <b>razón se propondrán prácticas con una duración acotada, para evitar un bajo rendimiento</b>.</p>   |
|                    | Herramientas disponibles         | <p>La <b>accesibilidad</b> a las diferentes herramientas para realizar las prácticas es un factor decisivo a la hora de plantear un guion de prácticas, de ello dependerá qué tipo de prácticas realizar o como plantearlas.</p> <p>Las FPGAs Libres como es el caso de la <b>IceZUM Alhambra cuentan con herramientas de acceso libre</b>, que permiten realizar el diseño, la síntesis y la programación de la FPGA. Por otro lado, la <b>aplicación web desarrollada en el presente trabajo permite programar la IceZUM Alhambra de una forma remota</b>.</p> <p>La <b>facilidad de uso</b> de las herramientas permitirá su uso por parte de estudiantes con un nivel más básico.</p> <p>En este caso, el laboratorio remoto nos ofrece dos caminos:</p> <ul style="list-style-type: none"> <li>❖ <b>diseñar el circuito digital en verilog</b> y realizar la compilación remota desde la aplicación web, y programar la placa para comprobar los resultados</li> <li>❖ <b>diseñar el circuito con IceStudio</b> y exportar el diseño a un archivo bitstream y subirlo a la web, para programar la placa y comprobar los resultados</li> </ul> <p>Dependiendo del nivel de los alumnos se podrá optar por un camino u otro.</p> <p>Hay que tener en cuenta que <b>los estudiantes de primer curso seguramente desconocen los denominados lenguajes de descripción hardware (HDL)</b>, y aprender un lenguaje de este tipo no se puede abordar en un cuatrimestre junto con los contenidos propios de una asignatura de introducción a la electrónica digital o diseño de circuitos digitales.</p> <p>Icestudio permite desarrollar algunos componentes en Verilog y encapsular diseños en otros más complejos. Esta propiedad se podría utilizar para diseñar previamente módulos que los estudiantes puedan necesitar para llevar a cabo las prácticas propuestas, de esta forma solo tendrían que seleccionar y conectar los módulos necesarios para implementar la solución de cada una de las prácticas. <b>La facilidad de uso de las herramientas disponibles se puede adaptar fácilmente a los conocimientos previos de los alumnos</b>.</p> |

TABLA 3.5: ANÁLISIS DE LOS FACTORES QUE INFLUYEN EN EL DISEÑO DE LAS PRÁCTICAS

|                         |  |
|-------------------------|--|
| Dificultad              | <p>La dificultad de las prácticas a realizar, siempre es un parámetro a considerar antes de abordar el diseño de un guión de prácticas, lo normal es <b>optar por prácticas con un nivel de dificultad creciente</b>, para facilitar la compresión de los conceptos trabajados.</p> <p>Si los alumnos no cuentan con conocimientos de electrónica digital se pueden plantear las prácticas de forma que se aborde el aprendizaje de circuitos digitales desde el interfaz de Icestudio, y centrarse en la <b>comprensión del pensamiento hardware en el diseño lógico de circuitos digitales</b>. Subiendo el nivel de complejidad paulatinamente.</p> <p>Si los alumnos cuentan ya con conocimientos de electrónica digital, se puede abordar el <b>aprendizaje del lenguaje HDL Verilog</b>, preparando prácticas en las que los alumnos tengan que utilizar este lenguaje para realizar diseños clásicos, que se puedan sintetizar y programar, esto <b>permitiría a los alumnos comprender el proceso de configuración de la FPGA</b></p>  |
| Criterios de evaluación | <p>Para poder cumplir con la finalidad de las prácticas es importante plantearse desde el principio cuales van a ser los criterios de evaluación de las prácticas: como el ritmo de trabajo, interpretación de resultados conclusiones ...</p> <p>El laboratorio remoto implementado permite realizar un seguimiento del uso de mismo por parte de los alumnos, el profesor puede acceder a informes que especifiquen las veces que un alumno ha accedido a la aplicación, el número de intentos realizados, y cuáles de ellos han tenido éxito y cuáles no. Estas herramientas podrían ser utilizadas como instrumentos de evaluación.</p> <p>Los criterios de evaluación podrían ser los siguientes:</p> <ul style="list-style-type: none"> <li>❖ <b>Completa las prácticas</b> en tiempo y forma</li> <li>❖ <b>Comprende el proceso</b> de compilación y configuración de la FPGA</li> <li>❖ <b>Diferencia entre Hardware y Software</b></li> <li>❖ <b>Implementa circuitos que dan solución a los problemas planteados</b> integrando los conocimientos adquiridos sobre componentes discretos y lenguajes de descripción hardware y haciendo uso de los recursos bibliográficos y herramientas informáticas a su alcance.</li> <li>❖ <b>Genera documentación</b> correctamente redactada, clara y precisa sobre el trabajo realizado en el laboratorio</li> </ul> |

### 3.6.2. PROPUESTA DE PRÁCTICAS A REALIZAR.

Se elabora un manual de prácticas con objeto de facilitar y servir de orientación en el proceso de aprendizaje del diseño de circuitos digitales y la programación bajo lenguaje de descripción hardware Verilog, simulación y síntesis de circuitos digitales sobre FPGAs, para ello se mostrarán aspectos relevantes como la estructura de una FPGA y las características relevantes del Verilog y del entorno de Icestudio.

Si se piensa en la complejidad que ya de por si conllevan el diseño, la programación y la simulación en el proceso del Diseño Digital, sumado a la complejidad, ya no sólo para realizar el proceso de síntesis sobre un dispositivo sino también para comprender y utilizar las herramientas existentes para síntesis, es comprensible que frecuentemente no se cubran los aspectos referentes a la síntesis e implementación práctica en las asignaturas troncales referentes a esta materia, por falta de tiempo.

El manual de prácticas contempla los diversos procesos necesarios para el diseño, programación e implementación de circuitos digitales en estos dispositivos. En especial se explora las posibilidades de la FPGA IceZUM Alhambra y de todos sus componentes, mostrando ejemplos completos y proponiendo ejercicios en los que el alumno podrá dar utilidad a circuitos digitales genéricos para aplicaciones particulares. El objetivo de las prácticas es que el alumno culmine el proceso de diseño, programación y simulación de los circuitos digitales con el proceso de utilización de herramientas de síntesis, la implementación y por último la verificación de un dispositivo real sin necesitar demasiado tiempo para familiarizarse con los complejos entornos de síntesis.

Una vez analizados los diferentes factores que se han considerado antes de plantear el diseño del guion de las prácticas, se propone a continuación las prácticas a realizar:

---

## PARTE I. DISEÑO DE CIRCUITOS DIGITALES - ICESTUDIO

### PRACTICA 1. ENCENDIENDO LEDS.

Esta práctica el alumno practicará encendiendo los leds internos de la IceZUM Alhambra, con la finalidad de que familiarice con el dispositivo y el entorno de trabajo

### PRACTICA 2. PERIFÉRICOS

En esta práctica se introducen los periféricos conectados a la FPGA IceZUM Alhambra, con el objetivo de que el alumno se familiarice con su uso.

### PRACTICA 3. MULTIPLEXACIÓN

Introducción al diseño de circuitos con multiplexores, y aplicación práctica de la lógica combinacional en el uso de dispositivos digitales de uso común

### PRACTICA 4. DISPLAY DE 7 SEGMENTOS

La finalidad de esta práctica es que el alumno utilice un display de 7 segmentos para mostrar el estado de un circuito digital

---

## PARTE 2. INTRODUCCIÓN AL LENGUAJE HDL – VERILOG

### PRACTICA 5. CONTADOR

Esta práctica consiste en diseñar un contador conectado a los LEDs internos de la placa IceZUM Alhambra

### PRACTICA 6. PRESCALER

En esta práctica se diseñará un prescaler de N bits para hacer parpadear un led a diferentes frecuencias

## PRACTICA 7. REGISTRO DE 4 BITS

La finalidad de esta práctica es realizar el diseño de un registro de 4 bits en lenguaje HDL-verilog

## PRACTICA 8. REGISTRO DE DESPLAZAMIENTO

Para finalizar se diseña en lenguaje HDL verilog un registro de desplazamiento, haciendo uso de los componentes diseñados en las anteriores prácticas

---

Las prácticas propuestas van aumentando en dificultad progresivamente y están dirigidas a alumnos que no sean necesariamente tienen porque ser de especialidades de informática o ingeniería electrónica o relacionados con ellas.

Se ha elaborado una [Guía de Prácticas](#) donde se explica cada una de las prácticas enumeradas con más detalle, dicho documento se adjunta a esta memoria como apéndice.

### 3.6.3. RECURSOS DE APOYO PEDAGÓGICOS

En la web se han añadido recursos de apoyo pedagógicos para ayudar al usuario del laboratorio a su utilización, tanto en la compilación local como en la remota, todos ellos adjuntos como apéndices a esta memoria.

---

#### Manual de uso IceStudio

Icestudio es una herramienta para diseño y síntesis de circuitos digitales en FPGAs libres, creada por Jesús Arroyo. Está programada en nodejs. Es software libre y multiplataforma. Corre en los sistemas GNU/Linux, Mac OS y Windows. La interfaz ofrece una biblioteca de objetos ya creados, tales como:

- ❖ ELEMENTOS BÁSICOS: Pines de entrada, de salida, comentarios, constantes y módulos con las entradas y salidas programables en Verilog.
- ❖ BIT: Se puede añadir un bit tanto a 1 como a 0.
- ❖ CONFIGURACIÓN: En este menú desplegable se puede encontrar configuraciones pull-up para los pines y buffers tri-estado.
- ❖ LÓGICA: En este menú desplegable se pueden encontrar dispositivos tales como multiplexores, demultiplexores, módulos para controlar visualizadores de 7 segmentos, todo tipo de puertas lógicas y algunos tipos de flip-flops.

En el [Apéndice 05 Manual de Uso IceStudio](#), se adjunta un documento que pretende seguir de guía para el uso de IceStudio. En él se explica la instalación del programa y las principales herramientas que presenta.

### **Manual verilog**

---

Verilog HDL Es utilizado para diseñar sistemas electrónicos, verilog soporta el diseño, prueba e implementación de circuitos analógicos, digitales y de señales mixta a diferentes niveles de complejidad.

Posee una sintaxis similar a la del lenguaje de programación C. El lenguaje tiene un preprocesador como C, y la mayoría de palabras reservadas de control como while, if entre otras son similares.

Verilog es uno de los HDL más utilizados y permite descripciones abstractas y representaciones en bajo nivel es decir puede describir sistemas digitales en base a compuertas, e incluso en base a transistores.

Permite que en un diseño se puedan usar diferentes niveles de descripción de sistemas digitales en un mismo ambiente, las diferentes descripciones pueden ser simuladas para verificar el funcionamiento y además pueden ser sintetizadas es decir traducidas a la interconexión de componentes básicas de un dispositivo programable.

Además permite la descripción estructural del diseño en base a componentes básicas, y descripciones más abstractas que se enfocan en la conducta del sistema. La conducta puede describirse mediante expresiones lógicas y también empleando procedimientos.

Un diseño basado en descripciones funcionales o de comportamiento puede resultar lento y de gran tamaño. Las descripciones en niveles estructurales permiten un ahorro de los circuitos lógicos para maximizar la velocidad, minimizar el tamaño y más bajo costo.

En el [Apéndice 06 Manual Verilog](#), se adjunta un documento que pretende servir de guía de aprendizaje para el alumno con ejemplos para el diseño HDL usando Verilog.

### **Manual de uso Icarus Verilog + GTKWave**

---

Los lenguajes HDL como Verilog son utilizados generalmente para programar FPGA (Field Programmable Gate Array). Estos dispositivos programables contienen bloques de lógica cuya interconexión y funcionalidad puede ser configurada mediante uno de estos lenguajes de descripción especializados (HDL). Se trata precisamente de programar hardware.

Cuando se trabaja con FPGAs se está realizando hardware y hay que tener cuidado. Se podría escribir un código que por ejemplo tuviera un cortocircuito. Y podría ocurrir que las herramientas de síntesis no avisen con un warning. Y al ser cargado en la FPGA, esta se podría estropear parcialmente.

Por ello, es preciso simular el código que se diseñe. Y una vez se comprueba que funciona correctamente es cuando se carga en la FPGA.

Icarus Verilog es un compilador de lenguaje Verilog desarrollado para GNU/Linux, pero que puede funcionar correctamente en Windows gracias a la pila MinGW (Minimal GNU for Windows). MinGW

es el mismo compilador que utiliza, por ejemplo, el IDE para desarrollo en lenguaje C Code::Blocks. Ya sea en sus versiones para Linux como en sus versiones para Windows, Icarus Verilog es liberado bajo la licencia GNU GPL.

Es un compilador liviano e incluye el simulador GTKWave para verificar el funcionamiento de un diseño.

En el [Apéndice 07 Manual de uso icarus verilog + GTKWave](#) se explica el proceso de instalación y configuración de Icarus Verilog en Windows, así como el uso básico del mismo a través de un programa (diseño) de prueba.

### 3.6.4. EJEMPLO DE APLICACIÓN DEL LABORATORIO REMOTO

En la página web se han añadido dos vídeos que muestran la utilización de la aplicación, uno utilizando la compilación local y otro utilizando la compilación remota

#### Compilación local

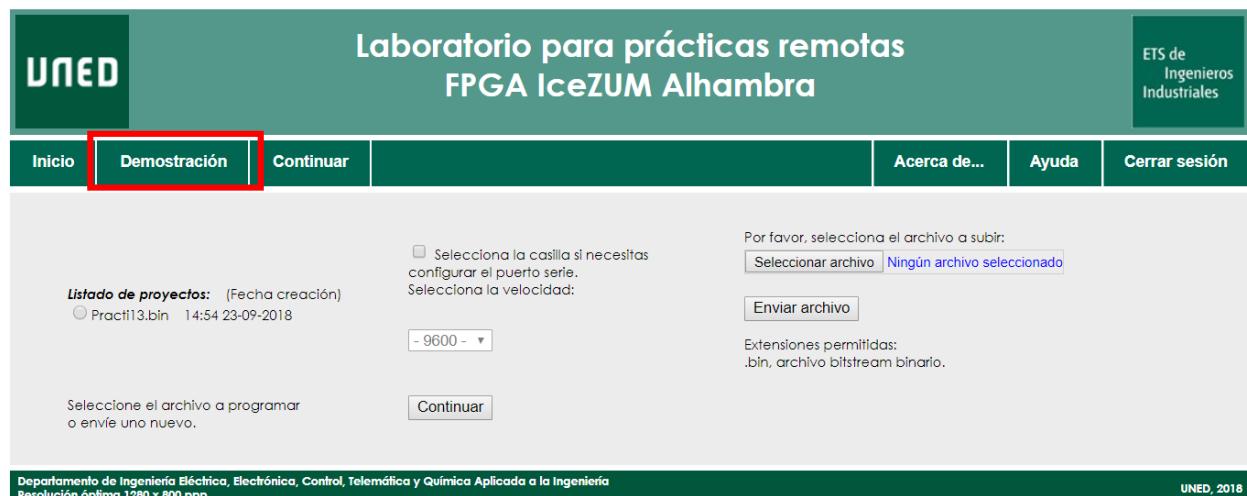


Figura 3.12: Acceso a video de demostración desde la página de compilación local

Accediendo a compilación local en el menú superior izquierdo al clicar en demostrador se accede a una página donde se muestra un vídeo en que se puede ver un ejemplo de aplicación de la compilación local.

### Compilación remota



Figura 3.13: Acceso a video de demostración desde la página de compilación remota

Accediendo a compilación remota en el menú superior izquierdo al clicar en demostrador se accede a una página donde se muestra un vídeo en que se puede ver un ejemplo de aplicación de la compilación Remota.

### 3.7. INSTALACIÓN Y MONTAJE DEL LABORATORIO REMOTO

Para la instalación y montaje del laboratorio en su ubicación definitiva se ha elaborado un [Manual de instalación y montaje adjunto a esta memoria como apéndice](#), en dicho manual se explican los pasos a seguir para montar el laboratorio, así como la habilitación de la web y del servidor.

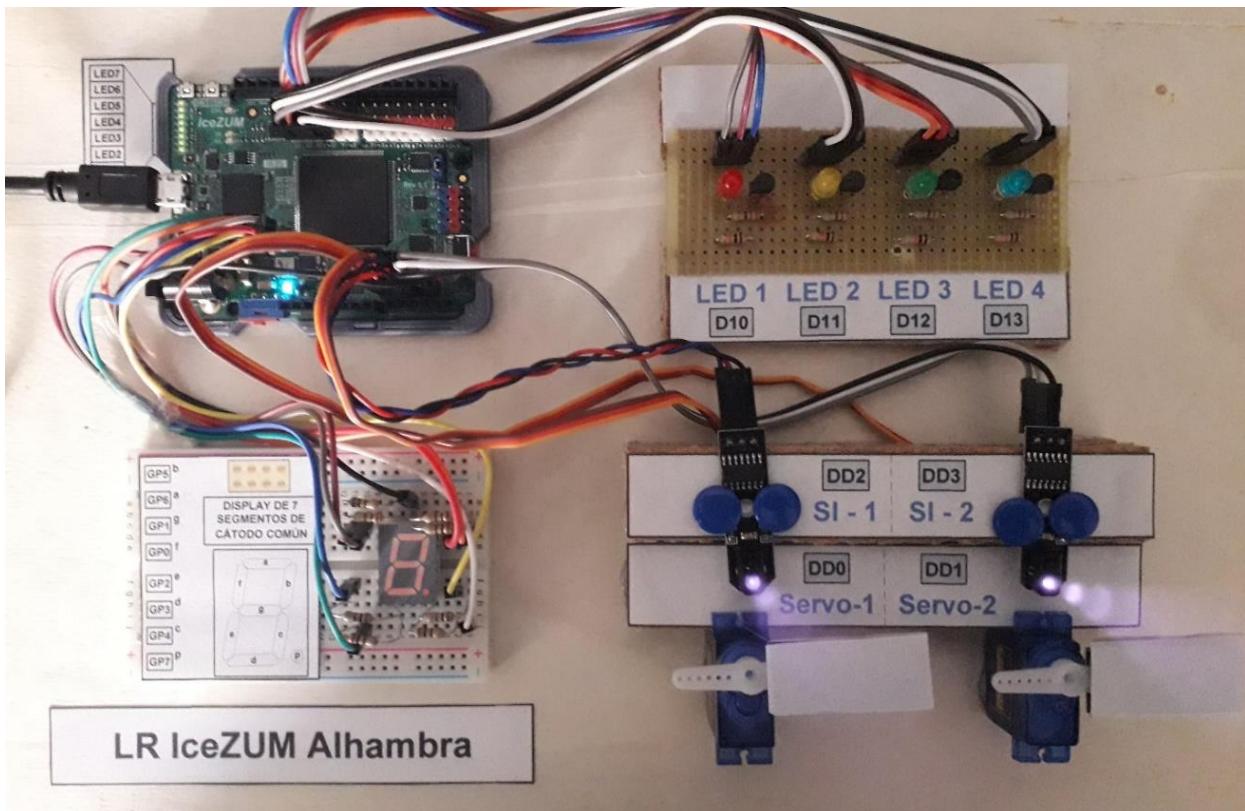


Figura 3.14. Montaje Laboratorio Remoto FPGA IceZUM Alhambra

## 4. PRESUPUESTO DEL LABORATORIO REMOTO

### A.-EQUIPAMIENTO - MATERIALES

| Código   | Ud | Descripción                                       | Medición | Precio          | Importe  |
|--|----|---|----------|-----------------|----------|
| A.01   | ud | ICEZUM Alhambra board I                           | 1        | 49.90 €         | 49.90 €  |
| A.02   | ud | Cámara IP, con interfaz ethernet DCS-2120, D-Link | 1        | 148.49 €        | 148.49 € |
| A.03   | ud | Cable Usb De Datos Usb A Mini Usb                 | 1        | 4.95 €          | 4.95 €   |
| A.04   | ud | Resistencia de carbono, 100 ohm, 2W, +/- 5%       | 8        | 0.33 €          | 2.64 €   |
| A.05   | ud | Cables de conexión Macho-Hembra 20CM              | 28       | 0.05 €          | 1.40 €   |
| A.06   | ud | Resistencias carbono 10 K ohmios, 2W, +/-5%, 1    | 4        | 0.41 €          | 1.64 €   |
| A.07   | ud | Resistencia de carbono, 270 ohm, 2W, +/- 5%       | 4        | 0.36 €          | 1.44 €   |
| A.08   | ud | Transistor NPN BC547B, con encapsulado TO-92      | 4        | 0.24 €          | 0.96 €   |
| A.09   | ud | LED 5 MM ESTÁNDAR Varios colores                  | 4        | 0.12 €          | 0.48 €   |
| A.10   | ud | IR Tracking SigueLineas                           | 2        | 3.50 €          | 7.00 €   |
| A.11   | ud | Tower Pro Micro Servo SG90 1.8Kg/9g/0.12seg       | 2        | 2.98 €          | 5.96 €   |
| A.12   | ud | Display de 7 segmentos, cátodo común, de 12,7 mm  | 1        | 1.85 €          | 1.85 €   |
| A.13   | ud | Placa Protoboard 82x55mm                          | 1        | 3.47 €          | 3.47 €   |
| A.14   | ud | Platina para soldar circuito para Prototipos      | 1        | 0.89 €          | 0.89 €   |
| <b>TOTAL CAPÍTULO A. EQUIPAMIENTO - MATERIALES</b> |    |   |          | <b>231.07 €</b> |          |

### B.-SOFTWARE

| Código                            | Ud | Descripción   | Medición | Precio | Importe    |
|-----------------------------------|----|---------------|----------|--------|------------|
| B.01                              | ud | IDE ICESTUDIO | 1        | - €    | - €        |
| <b>TOTAL CAPÍTULO B.-SOFTWARE</b> |    |               |          |        | <b>- €</b> |

### C.-MANO DE OBRA

| Código                                | Ud | Descripción              | Medición | Precio  | Importe           |
|---------------------------------------|----|--------------------------|----------|---------|-------------------|
| C.01                                  | h  | Configuración del Equipo | 4        | 25.00 € | 100.00 €          |
| C.02                                  | h  | Diseño                   | 80       | 25.00 € | 2 000.00 €        |
| C.03                                  | h  | Programación             | 80       | 25.00 € | 2 000.00 €        |
| C.04                                  | h  | Pruebas                  | 40       | 25.00 € | 1 000.00 €        |
| <b>TOTAL CAPÍTULO C.-MANO DE OBRA</b> |    |                          |          |         | <b>5 100.00 €</b> |

### RESUMEN DEL PRESUPUESTO

| CAPÍTULO                                 | EUROS             |
|--|-------------------|
| A.-EQUIPAMIENTO - MATERIALES             | 231.07 € 4.33%    |
| B.-SOFTWARE                              | - € 0.00%         |
| C.-MANO DE OBRA                          | 5 100.00 € 95.67% |
| <b>PRESUPUESTO DE EJECUCIÓN MATERIAL</b> | <b>5 331.07 €</b> |
| Gastos Generales                         | 13% 693.04 €      |
| Beneficio Industrial                     | 10% 533.11 €      |
|  | <b>6 557.22 €</b> |
| IVA                                      | 21% 1 377.02 €    |
| <b>PRESUPUESTO TOTAL</b>                 | <b>7 934.23 €</b> |

*El presupuesto del Laboratorio Remoto IceZUM Alhambra asciende a la cantidad de siete mil novecientos treinta y cuatro euros con veintitrés céntimos.*

## 5. CONCLUSIONES

En este proyecto se ha desarrollado un laboratorio remoto para poder manipular desde cualquier sitio con conexión a internet la FPGA IceZUM Alhambra proporcionando al estudiante de ingeniería industrial la posibilidad de aprender a programar FPGAs sin desplazarse. Sobre la base del trabajo realizado en el mismo pueden establecerse las siguientes conclusiones:

- El conjunto de las tareas que se han realizado; diseño de Laboratorio Remoto, desarrollo de la aplicación web y elaboración de materiales didácticos, constituyen el Laboratorio Remoto de la FPGA libre IceZUM Alhambra.
- Los Laboratorios Remotos han evolucionado sustancialmente en los últimos años, implantándose en cada vez más centros educativos. Sin embargo todavía queda mucho trabajo por hacer, puesto que se conocen pocas experiencias en centros que no sean de educación superior, probablemente debido a la falta de formación del personal en relación a este tipo de dispositivos.
- Las FPGAs se encuentran actualmente inmersos en su particular “revolución industrial”. Desde que en 2015 el ingeniero austriaco Clifford Wolf hiciera ingeniería inversa de la familia ICE40 de Lattice, creando el primer sintetizador de FPGAs libre de la historia se han desarrollado diferentes tarjetas entrenadoras que utilizan la FPGA Lattice iCE40, una de ellas es la IceZUM Alhambra, que cuenta con su IDE, Icestudio.
- Los lenguajes HDL se caracterizan por ser poco intuitivos y más complejos que los lenguajes de programación textual más habituales en la programación de software. Para que los estudiantes puedan entender mejor el diseño de los circuitos digitales, se puede utilizar la IDE, Icestudio, una herramienta libre creada por Jesús Arroyo. Icestudio que permite diseñar gráficamente circuitos digitales, sin necesidad de utilizar código de descripción hardware, permitiendo introducir a la electrónica digital a alumnos más jóvenes.
- A pesar de que se puede pensar que el uso de Icestudio limita las posibilidades de programación de la FPGA, no es así, ya que permite la introducción de código en bloques, lo que facilita el aprendizaje del lenguaje HDL Verilog. Por ello es más intuitivo para los estudiantes que comienzan su aprendizaje en este tipo de programación: se trata de una herramienta sencilla con un gran potencial, y lo más importante es software libre.
- Con las FPGAs se tiene la posibilidad de programar y manipular diferentes tipos de elementos en la industria. Es interesante que los estudiantes de ingeniería industrial conozcan la capacidad que tienen estos dispositivos, programándolos y utilizándolos, mediante lenguajes HDL. Esta idea se puede transmitir mediante el Laboratorio Remoto desarrollado en este proyecto.
- Durante la valoración de las posibilidades que ofrece la FPGA IceZUM Alhambra desde el punto de vista técnico, tecnológico y educativo, para desarrollar la experiencia a ser realizada por los estudiantes con acceso remoto, ha resultado complejo sintetizar la gran variedad de posibilidades que se pueden llevar a cabo. Por otro lado el hecho que los desarrolladores estén actualmente trabajando en el proyecto de la FPGA libre IceZUM Alhambra facilita el trabajo,

porque proporcionan tutoriales e información muy didáctica al respecto. Sin embargo limita las aplicaciones actuales del Laboratorio Remoto, puesto que aún se están desarrollando componentes para las comunicaciones serie con la FPGA o para poder realizar la lectura de sensores, aplicaciones con las que se podría ampliar el Laboratorio Remoto en un futuro próximo.

- Durante el desarrollo de la interfaz web fue clave el conocimiento de las herramientas Icestorm, desarrolladas por Clifford Wolf en su trabajo de ingeniería inversa, ya que son los ejecutables que la aplicación web utiliza para compilar y programar la FPGA IceZUM Alhambra desde el servidor.
- El entorno web del laboratorio está basado en el Laboratorio de FPGA's instalado en el Departamento de Ingeniería Eléctrica, Electrónica, Control, Telemática y Química Aplicada a la Ingeniería de la UNED. Dicho equipo es un desarrollo basado en Xilinx Spartan 3AN FPGA para llevar a cabo prácticas de programación VHDL a distancia con visualización a través de cámara web. Contar con este ejemplo ha resultado de gran ayuda a la hora de diseñar e implementar el Laboratorio Remoto de FPGA Libre IceZUM Alhambra que ofrecerá ahora otra opción diferente de aprendizaje permitiendo realizar prácticas de programación Verilog y el uso de la IDE Icestudio para el diseño de circuitos digitales. Estas herramientas, siendo de código libre, resultan muy atractivas para su uso académico debido a su sencillez y al potencial educativo que presentan.
- Desarrollar todos los materiales didácticos para el uso del Laboratorio Remoto por parte de los estudiantes ha sido un trabajo tedioso, ya que ha sido necesario documentar adecuadamente el uso del laboratorio, y las herramientas requeridas para su empleo. Es importante que toda esta documentación esté bien elaborada, puesto que durante el uso del Laboratorio Remoto, será la única ayuda que tenga el estudiante para realizar las prácticas.
- Desarrollar la guía de prácticas ha resultado muy interesante porque se ha tenido la oportunidad de probar el laboratorio remoto y comprobar el uso remoto de la aplicación, y probar los diferentes resultados en la FPGA.
- Ha sido complicado y a la vez motivador trabajar en la implementación del Laboratorio Remoto de un dispositivo del que actualmente se están desarrollando aplicaciones, y todos los avances que se realizan son nuevas aportaciones.

Como futuras líneas de trabajo se podrían proponer las siguientes:

- Debido a que todavía se están desarrollando herramientas para el uso de la FPGA IceZUM Alhambra, en un futuro próximo se podría ampliar el laboratorio remoto instalando una pantalla LCD en los pines de 5V que quedan libres y sensores en los pines analógicos. Esta ampliación sería fácil de implementar ya que solo supondría la conexión de dichos periféricos en la IceZUM Alhambra
- Por otro lado se podría implementar la comunicación serie con el dispositivo a través de la interfaz web, y aumentar con esto la interacción del Laboratorio Remoto con el usuario.

- Actualmente los desarrolladores de las herramientas libres sobre las que se apoya el diseño de la aplicación web, que permiten la compilación y programación de la FPGA, están trabajando en nuevas versiones de los paquetes yosys, nextpnr y los proyectos icesstorm y trellis. En consecuencia, el paquete 'toolchain-icesstorm' quedará obsoleto y aparecerán los paquetes 'toolchain-yosys', 'toolchain-ice40' y toolchain-ecp5. Como línea de trabajo futuro se plantea la actualización de las herramientas utilizadas actualmente en la aplicación web por las herramientas que surjan de estos futuros desarrollos.

## 6. BIBLIOGRAFÍA

- [1] J. R. H. Luis Rosado, «Aportaciones didácticas de los laboratorios virtuales y remotos en la enseñanza de la física,» p. 11, 2005.
- [2] C. C. B. a. C. J. Ko, Creating Web-based Laboratories, Springer, 2005.
- [3] M. y. O. Castro, «Servicios para Plataformas Educativas: Laboratorios y Aplicaciones,» de *IX Congreso de Tecnologías Aplicadas a la Enseñanza Electrónica*, TAAE, 2010.
- [4] Y. R. O. y. A. D. Velasquez, «Technology Used for the Implementation of Remote Laboratories,» *International Journal of Applied Engineering Research*, nº 11, p. 5, 2016.
- [5] J. Garcia-Zubía, «Estrategias de diseño de laboratorios remotos,» *IEEE*, 2008.
- [6] A. J. C. R. A. B. H. Hoyer, «A Multiuser Virtual-Reality Environment for a Tele-Operated Laboratory,» *IEEE Transactions on Education*, vol. 47, nº 1, February 2004.
- [7] C. Arguedas Matarrita, «Diseño y desarrollo de un Laboratorio Remoto para la enseñanza,» 2017.
- [8] M. V. J. y. V. J. Lopez, «Generación de páginas web Interactivas,» de *Desarrollo web en entorno servidor*, Madrid, RA-MA, 2014, pp. 232-234.
- [9] J. Zofio, «Lenguaje de marcas, hojas de estilos y scripts,» de *Aplicaciones web*, Madrid, Macmillan Iberia, 2014, p. 19.
- [10] E. Blanco, «Fundamentos de Informática en Entornos Bioinformáticos,» de *Gestión de Datos con MySQL*, Barcelona, UOC, 2013, p. 172.
- [11] J. Villada, «Instalación y Configuración del Software de Servidor web,» de *Conceptos Básicos de Sistemas de Servidores*, Granada, IC, 2015, pp. 37-38.
- [12] M. E. A. J. R. y. F. P. Francisco Charte, «Uso de dispositivos FPGA como apoyo a la enseñanza de asignatura de arquitectura de ordenadores,» *Enseñanza y Aprendizaje de Ingeniería de Computadores*, nº 7, p. 16, 2017.
- [13] J. G. Gomez, «Obijuan,» [En línea]. Available: <https://github.com/Obijuan/>. [Último acceso: 23 Agosto 2018].
- [14] L. V. M. y. otros, *FPGA. Nociones básicas e implementación*, Madrid: Departamento de Ingeniería Informática, 2004.
- [15] M. N. T. J. M. C. F. G. M. H. y. N. L. J. Aguirre Echanove, Diseño de sistemas digitales mediante lenguajes de descripción de hardware, Sevilla, España, 2005.
- [16] A. y. P. M. F. Aparicio Olmedo, Estudio comparativo de los lenguajes HDL y su aplicación en la implementación del laboratorio de sistemas digitales avanzados, Riobamba, Ecuador: [ebook], 2014.

- [17] Es.wikibooks.org, «Programación en VHDL - Wikilibros,» 2018. [En línea]. Available: [https://es.wikibooks.org/wiki/Programaci%C3%B3n\\_en\\_VHDL](https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_VHDL).
- [18] Es.wikibooks.org., «Programación en Verilog - Wikilibros,» 2018. [En línea]. Available: [https://es.wikibooks.org/wiki/Programaci%C3%B3n\\_en\\_Verilog](https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Verilog).
- [19] C. Wolf, «Project IceStorm,» Marzo 2015. [En línea]. Available: <http://www.clifford.at/icestorm/>. [Último acceso: Agosto 2018].
- [20] J. Gonzalez, «IceZUM Alhmabra board,» [En línea]. Available: <https://github.com/FPGAwars/icezum/wiki..> [Último acceso: Agosto 2018].
- [21] L. Semiconductor, *iCE40TM LP/HX Family Data Sheet*, 2017.
- [22] J. G. a. J. Arroyo, «IceStudio,» [En línea]. Available: <https://github.com/FPGAwars/icestudio>. [Último acceso: Agosto 2018].
- [23] S. Williams, «Icarus Verilog,» [En línea]. Available: <http://iverilog.icarus.com/>. [Último acceso: Agosto 2018].
- [24] S. M.-G. y. M. C.-G. Luis Joya-Guirado, «Diseño de laboratorio remoto abierto para electrónica digital,» *DYNA New Technologies*, p. 13, 2016.

## **APÉNDICES**

---

## A01-MANUAL DE INSTALACIÓN Y MONTAJE

---

## ÍNDICE DE CONTENIDOS

---

|   |   |
|---|---|
| 1 INTRODUCCIÓN.....                               | 3 |
| 2 REQUISITOS.....                                 | 3 |
| 3 INSTALACIÓN.....                                | 3 |
| 4 INSTALACIÓN DRIVERS DE LA ICEZUM ALHAMBRA ..... | 5 |
| 4.1 SELECCIÓN DE FPGA .....                       | 5 |
| 4.2 COMPROBAR LA INSTALACIÓN.....                 | 6 |

## ÍNDICE DE ILUSTRACIONES

---

|   |   |
|---|---|
| <i>Figura 4.1: Menú herramientas/Drivers, Icestudio.....</i>      | 5 |
| <i>Figura 4.2: Selección Placa .....</i>                          | 6 |
| <i>Figura 4.3: Menú Archivo/Ejemplos/1. Basic, Icestudio.....</i> | 6 |
| <i>Figura 4.4: Ejemplo 01.One LED, Icestudio .....</i>            | 7 |
| <i>Figura 4.5: Verificación 01.One LED, Icestudio .....</i>       | 7 |
| <i>Figura 4.6: Sintetizado 01.One LED, Icestudio .....</i>        | 8 |
| <i>Figura 4.7: Carga 01.One LED, Icestudio .....</i>              | 8 |
| <i>Figura 4.8: LED 0 encendido, FPGA IceZUM Alhambra.....</i>     | 8 |

## 1 INTRODUCCIÓN

El presente manual de instalación es una guía para instalar la web del Laboratorio remoto IceZUM Alhambra en el servidor.

## 2 REQUISITOS

La instalación del laboratorio requiere que en el servidor se encuentre instalado el **software IceStudio**, que lleva incluido apio, que será la herramienta que se utilice para la generación del bitstream y posterior programación de la FPGA IceZUM Alhambra.

Se omite aquí la instalación del servidor web y del servidor MySQL, ya que no necesariamente tienen que ser los usados.

**El servidor deberá proporcionar acceso a la base de datos y tener soporte para PHP.**

## 3 INSTALACIÓN

- 1- Copia de los archivos del laboratorio remoto IceZUM Alhambra en la carpeta destino del servidor.** Los archivos que componen el laboratorio se encuentran en un archivo comprimido, el cual deberá ser descomprimido antes de copiarlo al directorio del servidor.
- 2- Configuración de la base de datos.**
  - a. Arrancar el gestor de bases de datos**
  - b. Acceder al administrador de bases de datos**
  - c. Crear una base de datos con el nombre `fpga_icezum` y un usuario administrador llamado “laboratorio” con contraseña “`fpga`”**
  - d. Importar el fichero “`fpga_icezum.sql`”**
- 3- Acceder al laboratorio remoto como usuario “admin” y contraseña “admin”** y en el apartado de “Mantenimiento”, en el botón “configuración” se cambian el resto de parámetros necesarios, como son el acceso a la base de datos, host, nombre, usuario y contraseña, límite que queremos imponer para el numero máximo de usuarios simultáneos, el puerto serie que se va a usar y las direcciones de correo electrónico a donde se van a mandar los avisos generados por el laboratorio.
- 4- Esta configuración se puede realizar de forma manual modificando los siguientes archivos**

|  |
|--|
| <b>'config_bbdd.ini'</b>   |
| <pre>&lt;?php \$host = "localhost"; /* nombre del host */ \$bdatos = "fpga_icezum"; /* nombre de la base de datos */ \$user = "laboratorio"; /* nombre de usuario de acceso */ \$passwd = "fpga"; /* password de acceso */ ?&gt;</pre> |
| <b>'config_limites.ini'</b>  |
| <pre>&lt;?php \$limite_trabajo=10; //límite de usuarios trabajando \$limite_esperas=10; //límite de usuarios en espera ?&gt;</pre>   |
| <b>'config_com.ini'</b>  |
| <pre>&lt;?php \$com="COM9"; //PUERTO SERIE ?&gt;</pre>   |

## 5- Configuración de archivos para compilación remota

|   |
|---|
| <b>'yosys.php' – línea 204</b>  |
| <pre>\$comando='C://Users/USER/.icestudio/api/packages/toolchain-icestorm/bin/yosys -p "synth_ice40 -blif ./fpga/'.\$ruta_carga.'.'.\$sin_ext.'.'.\$sin_ext.'.blif" ./fpga.'/'.\$ ruta_carga.'.'.\$sin_ext.'.'.\$sin_ext.'.v';</pre>  |
| <b>'arachne.php' – línea 124</b>  |
| <pre>\$envio='C://Users/USER/.icestudio/api/packages/toolchain-icestorm/bin/arachne-pnr -d 1k -p ./fpga.'/'.\$ ruta_carga.'.'.\$sin_ext.'.'.\$sin_ext.'.pcf ./fpga.'/'.\$ ruta_carga.'.'.\$sin_ext.'.'.\$sin_ext.'.blif -o ./fpga.'/'.\$ ruta_carga.'.'.\$sin_ext.'.'.\$sin_ext.'.txt';</pre> |
| <b>'icepack.php' – línea 130</b>  |
| <pre>\$envio='C://Users/USER/.icestudio/api/packages/toolchain-icestorm/bin/icepack ./fpga.'/'.\$ ruta_carga.'.'.\$sin_ext.'.'.\$sin_ext.'.txt ./fpga.'/'.\$ ruta_carga.'.'.\$sin_ext.'.'.\$sin_ext.'.bin';</pre>   |

## 6- Configuración de archivo para programación la FPGA

|   |
|---|
| <b>'prog_pass2.php' línea 95</b>  |
| <pre>\$envio='C://Users/USER/.icestudio/api/packages/toolchain-icestorm/bin/iceprog '.'/'.\$ ruta_carga;'</pre> |

## 7- Configuración video

|  |
|--|
| <b>'./admin/video.php' línea 82</b>  |
| <pre>&lt;img src="http://62.204.201.150:30/videostream.cgi?user=admin&amp;pwd=Dieec_01" id="webcam" border="1" /&gt;</pre> |
| <b>'carga-video.php' línea 95</b>  |
| <pre>&lt;img src="http://62.204.201.150:30/videostream.cgi?user=admin&amp;pwd=Dieec_01" id="webcam" border="1" /&gt;</pre> |

## 8- Configuración reset

```
'reset.php' línea 21
$envio='C:\\\\Users\\\\USER\\\\icestudio\\\\asio\\\\packages\\\\toolchain-icestorm\\\\bin\\\\iceprog leds.bin'
'reset2.php' línea 21
$envio='C:\\\\Users\\\\USER\\\\icestudio\\\\asio\\\\packages\\\\toolchain-icestorm\\\\bin\\\\iceprog leds.bin'
'salir_trabj.php' línea 11
$envio='C:\\\\Users\\\\USER\\\\icestudio\\\\asio\\\\packages\\\\toolchain-icestorm\\\\bin\\\\iceprog leds.bin'
```

## 4 INSTALACIÓN DRIVERS DE LA ICEZUM ALHAMBRA

Para cargar en la placa los circuitos sintetizados, es necesario que se configure los drivers para ello clicamos habilitar en el menú **Herramientas/Drivers**.

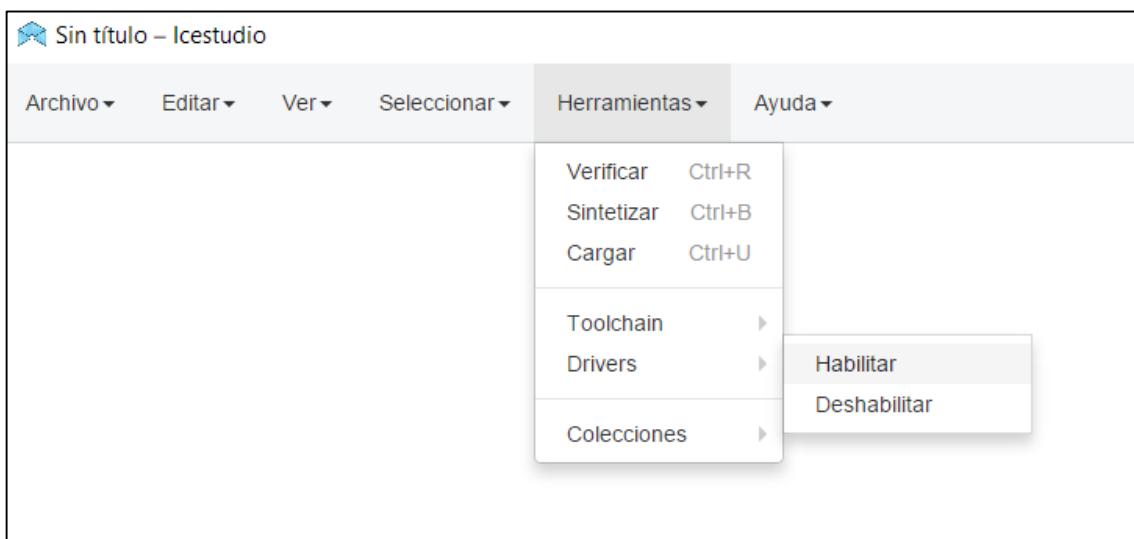


Figura 4.1: Menú herramientas/Drivers, Icestudio

Los pasos a seguir dependerán del sistema operativo del equipo y de la FPGA libre se utilice.

### 4.1 SELECCIÓN DE FPGA

Icestudio permite trabajar con varios tipos de placas, para seleccionar la placa con la que se va a trabajar se accede al menú **Seleccionar/Placa**, y se elige la que corresponda:

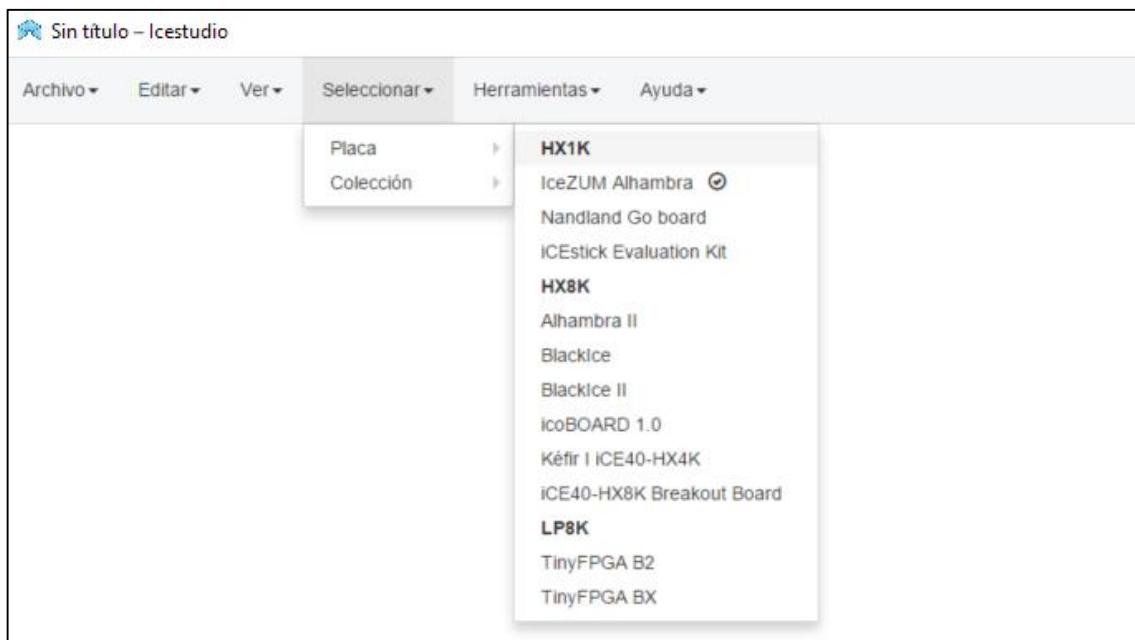


Figura 4.2: Selección Placa

## 4.2 COMPROBAR LA INSTALACIÓN

Para comprobar que la instalación y la configuración del programa han sido correctas, se carga un ejemplo. Para ello se entra en el menú **Archivo/Ejemplos/1.Basic** y se selecciona *One LED*

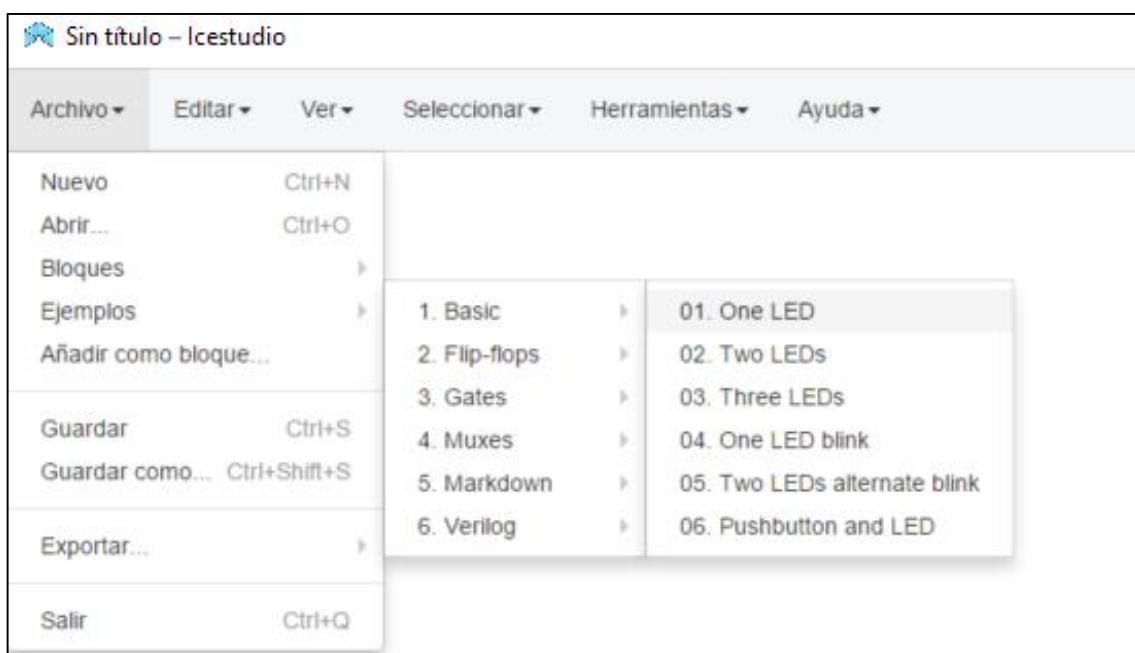


Figura 4.3: Menú Archivo/Ejemplos/1. Basic, Icestudio

Y se abrirá la siguiente ventana:

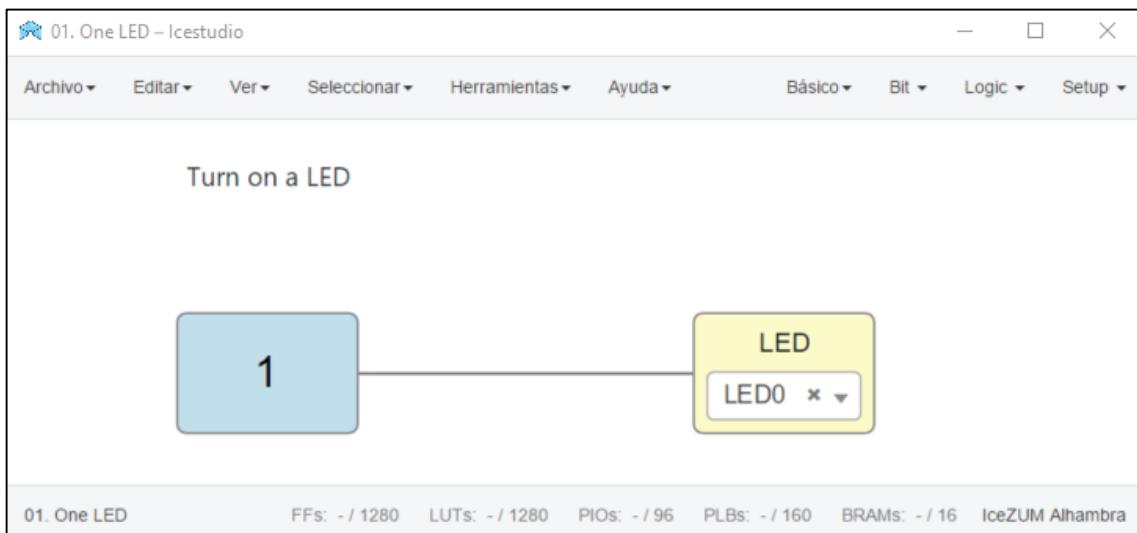


Figura 4.4: Ejemplo 01.One LED, Icestudio

Con la placa conectada al equipo, se accede al menú herramientas/verificar y se verifica el diseño:

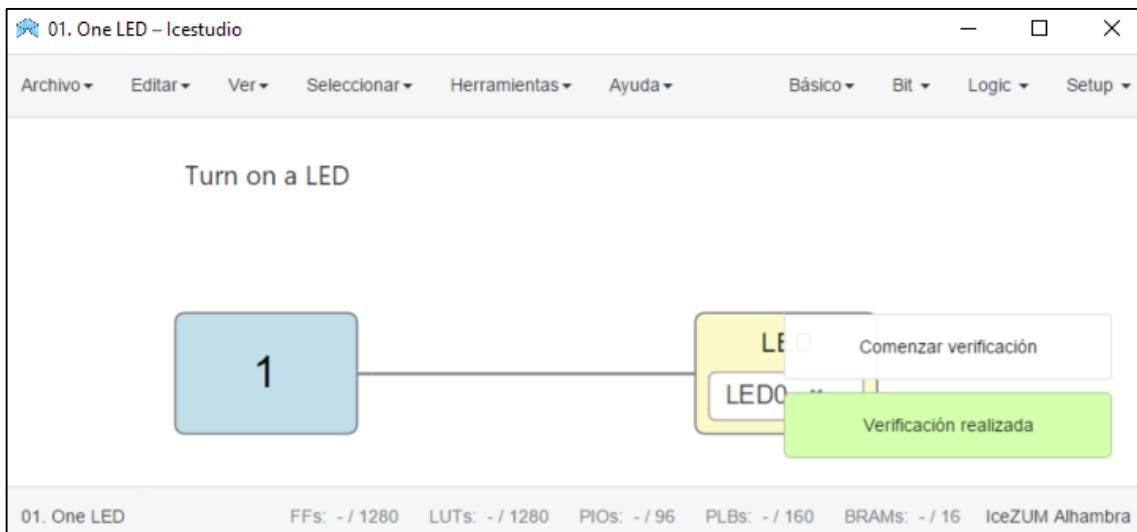


Figura 4.5: Verificación 01.One LED, Icestudio

Una vez verificado el diseño se procede a sintetizarlo en el menú Herramientas/sintetizar:

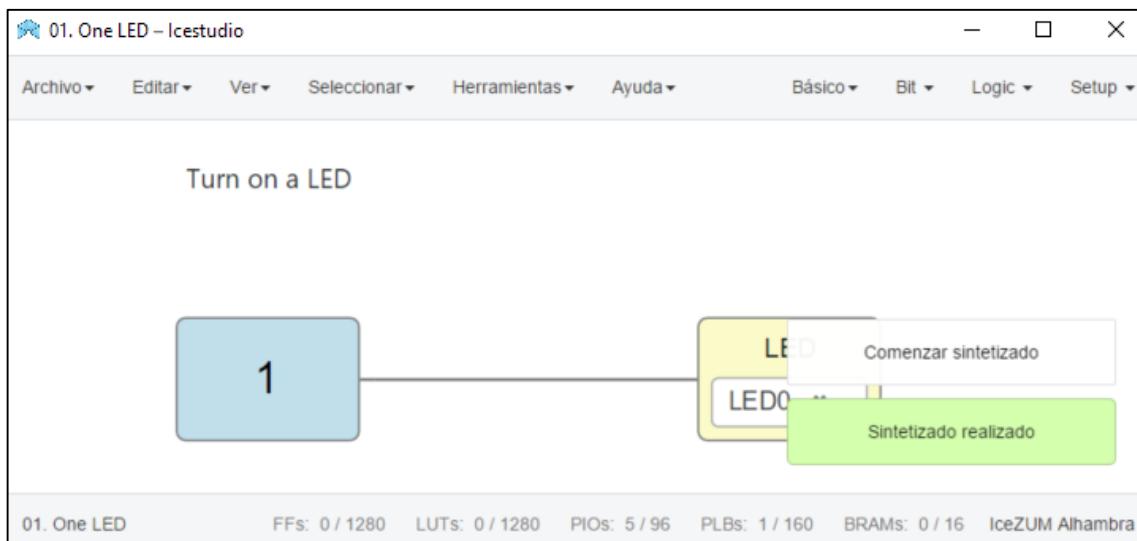


Figura 4.6: Sintetizado 01.One LED, Icestudio

Y para finalizar se carga en la placa en el menú Herramienta/cargar:

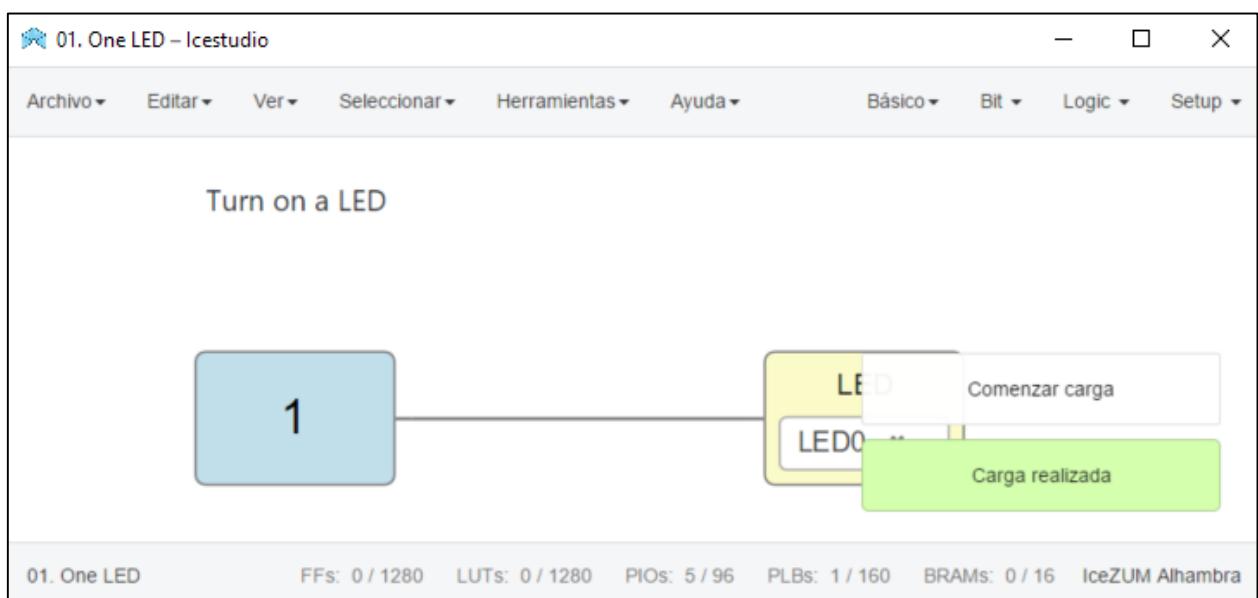


Figura 4.7: Carga 01.One LED, Icestudio

Si todo ha ido bien se podrá ver el LED 0 encendido en la placa:

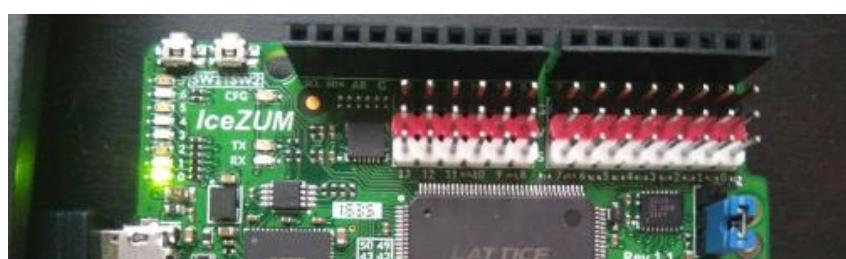


Figura 4.8: LED 0 encendido, FPGA IceZUM Alhambra.  
Fuente: <https://icestudio.readthedocs.io/en/latest/source/installation.html>

Es posible que la carga no vaya bien, y se Icestudio informe de un error, las posibles causas podrían ser:

- La placa no está conectada al USB → Conectarla
- La placa está conectada a otro puerto USB que no está configurado → conectarla al puerto USB correcto
- La placa no se ha conectado correctamente al USE → revisar conexiones
- El cable USB utilizado no es el adecuado → utilizar un cable de datos
- Problemas con los drivers → Reinstala los Drivers, en Windows a veces hay que intentarlos varias veces.

---

## A02-MANUAL DE USUARIO

---

## ÍNDICE DE CONTENIDOS

---

|  |    |
|--|----|
| 1 INTRODUCCIÓN .....                             | 3  |
| 2 ACCESO LABORATORIO REMOTO ICEZUM ALHAMBRA..... | 3  |
| 3 ELECCIÓN DE PRÁCTICA A REALIZAR.....           | 4  |
| 4 FORMAS DE TRABAJO.....                         | 5  |
| 4.1 COMPILACIÓN LOCAL .....                      | 5  |
| 4.2 COMPILACIÓN REMOTA.....                      | 6  |
| 5 PROGRAMACIÓN FPGA.....                         | 9  |
| 6 LIMITACIONES DE USO.....                       | 9  |
| 7 PÁGINA DE UTILIDADES .....                     | 10 |
| 8 MENSAJES ADMINISTRADORES .....                 | 11 |
| 9 ACCESOS CON IDENTIFICADORES DUPLICADOS.....    | 12 |

## ÍNDICE DE ILUSTRACIONES

---

|   |    |
|---|----|
| <i>Figura 1: Entrada a laboratorio remoto IceZUM Alhambra.....</i>                                  | 3  |
| <i>Figura 2: Solicitud de Nombre completo, solo primer acceso. ....</i>                             | 4  |
| <i>Figura 3: Hoja de ayuda e información del LR IceZUM Alhambra, solo primer acceso. ....</i>       | 4  |
| <i>Figura 4: Hoja de selección tipo de compilación LR IceZUM Alhambra. ....</i>                     | 5  |
| <i>Figura 5: Página de envío de archivos bitstream, LR IceZUM Alhambra. ....</i>                    | 6  |
| <i>Figura 6: Proceso de compilación LR IceZUM Alhambra.....</i>                                     | 7  |
| <i>Figura 7: Página Yosys –Síntesis-, compilación remota LR IceZUM Alhambra.....</i>                | 7  |
| <i>Figura 8: Página Arachne –Emplazado y enrutado-, compilación remota LR IceZUM Alhambra....</i>   | 8  |
| <i>Figura 9: Página Icepack –Generación bitstream-, compilación remota LR IceZUM Alhambra. ....</i> | 8  |
| <i>Figura 10: Página Icepack –bitstream generado-, compilación remota LR IceZUM Alhambra.....</i>   | 8  |
| <i>Figura 11: Página Iceprog –Programar-, LR IceZUM Alhambra.....</i>                               | 9  |
| <i>Figura 12: Página Utilidades-Consulta Registros generados, LR IceZUM Alhambra. ....</i>          | 10 |
| <i>Figura 13: Envío de mensajes, LR IceZUM Alhambra.....</i>  | 11 |
| <i>Figura 14: Respuestas de mensajes, LR IceZUM Alhambra.....</i>                                   | 11 |
| <i>Figura 15: Usuario ya identificado, LR IceZUM Alhambra.....</i>                                  | 12 |

## 1 INTRODUCCIÓN

El presente manual de usuario es una guía de utilización del Laboratorio Remoto IceZUM Alhambra, dicho laboratorio es una plataforma web a través de la cual se puede compilar archivos en leguaje HDL hasta llegar a un archivo binario programable en una FPGA. La aplicación web proporciona durante todo el proceso resultados y registros.

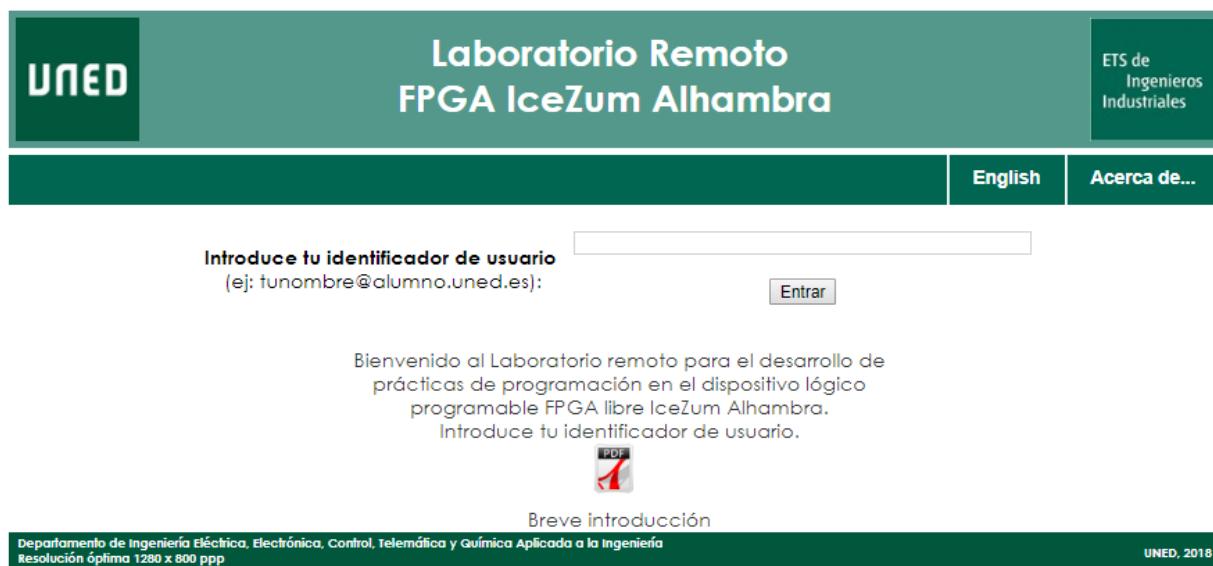
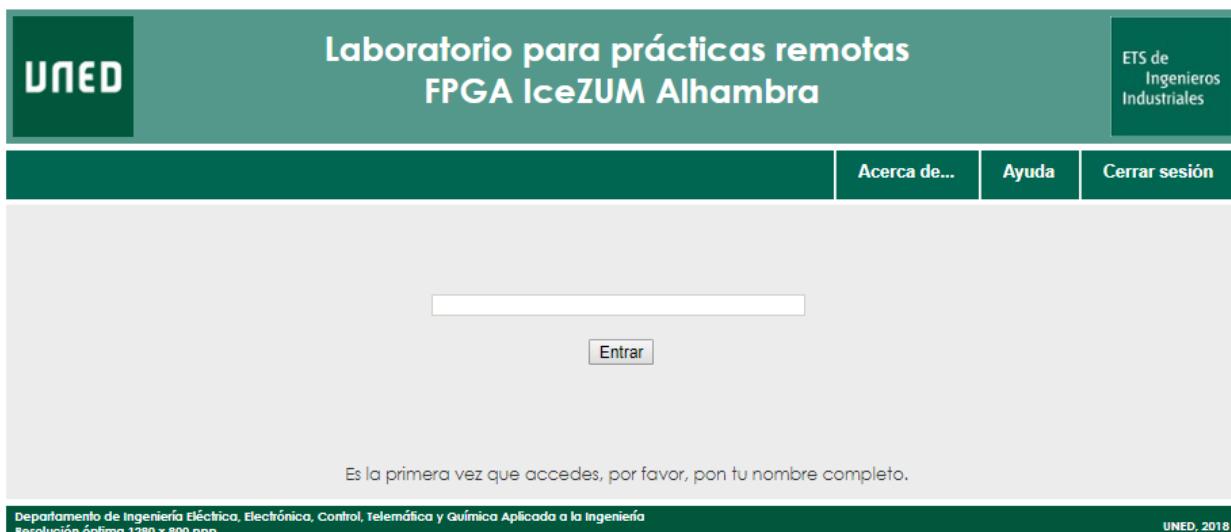


Figura 1: Entrada a laboratorio remoto IceZUM Alhambra.

## 2 ACCESO LABORATORIO REMOTO ICEZUM ALHAMBRA

El acceso se realiza con un identificador que puede ser elegido libremente o ser asignado por el profesor (debe tener menos de 10 caracteres).

Una vez introducido el identificador, la primera vez que se accede se solicita el nombre completo para un mejor reconocimiento del trabajo realizado por cada alumno y evitar posibles errores por similitud de identificadores.



Laboratorio para prácticas remotas  
FPGA IceZUM Alhambra

ETS de Ingenieros Industriales

Acerca de... Ayuda Cerrar sesión

Entrar

Es la primera vez que accedes, por favor, pon tu nombre completo.

Departamento de Ingeniería Eléctrica, Electrónica, Control, Telemática y Química Aplicada a la Ingeniería  
Resolución óptima 1280 x 800 ppp

UNED, 2018

Figura 2: Solicitud de Nombre completo, solo primer acceso.

Cuando se finaliza el registro, se informa brevemente del proceso que se sigue en el laboratorio para la compilación de los archivos HDL, paso que se obvia en los siguientes accesos que realice el usuario.



Laboratorio Remoto FPGA  
IceZUM Alhambra

ETS de Ingenieros Industriales

Inicio Atrás... Acerca de... Enviar mensaje Cerrar sesión

A continuación se ofrece una pequeña descripción de la página y enlaces con recursos relacionados con la FPGA IceZUM Alhambra. Si ya la has leído puedes [continuar](#), además puedes volver aquí desde cualquier página mediante el enlace de ayuda del menú superior.

**Normas de funcionamiento.**

El objetivo de este laboratorio remoto es facilitar la realización de prácticas de asignaturas de programación de dispositivos lógicos programables. Aquí usamos la FPGA libre Ice ZUM Alhambra basada en ICE40HX1K-TQ144 de Lattice, cuya hoja de características se adjunta más abajo. Debido a que hay múltiples accesos a la página, el tiempo de permanencia está un tanto limitado; en caso de estar inactivo en una página aparecerá un mensaje advirtiendo del hecho y si no se realiza ninguna acción, a los 3 minutos se termina la sesión y se guarda lo realizado hasta ese momento. Se recomienda salir siempre con el enlace de "Cerrar sesión" ya que así se cierra correctamente la sesión, se guarda registro del trabajo realizado y a los demás usuarios se les evita una espera innecesaria.

Figura 3: Hoja de ayuda e información del LR IceZUM Alhambra, solo primer acceso.

A esta página también se podrá acceder desde el **menú superior, clicando en ayuda**.

### 3 ELECCIÓN DE PRÁCTICA A REALIZAR

Al presionar en continuar se accede a la página de elección de práctica a realizar, a esta página es a la que se accederá directamente cuando el usuario acceda en ocasiones posteriores. A partir de aquí el proceso toma dos caminos diferentes según se elija una forma de trabajar u otra.

Hola usuario6 : Seleccione el tipo de práctica a realizar

No tiene mensajes

[Compilación local](#) Realizada en el PC de usuario y envío del archivo \*.bin

[Compilación remota](#) Realizada en el laboratorio, enviando archivos de código

Departamento de Ingeniería Eléctrica, Electrónica, Control, telemática y Química Aplicada a la Ingeniería  
Resolución óptima 1280 x 800 ppp

UNED, 2018

Figura 4: Hoja de selección tipo de compilación LR IceZUM Alhambra.

Cualquier página de la aplicación web tiene dos zonas de trabajo, la parte superior con el título de la página donde se encuentra y asociado a la barra de menú, que se mantienen invariables a lo largo de todo el proceso. Debajo de la barra horizontal de menú se encuentra la zona de trabajo donde va apareciendo las diferentes etapas por las que se va pasando. Para abandonar el laboratorio es necesario hacerlo usando el enlace “**Cerrar sesión**” disponible en todas las páginas para guardar registro de actividad y prácticas realizadas y no haya perdida de información.

## 4 FORMAS DE TRABAJO

En el uso del laboratorio se distinguen dos formas de trabajo, compilación remota, que consiste en obtener el archivo bitstream (.bit) en el ordenador personal, y subir dicho archivo para ver los resultados programando la FPGA. Por otro lado, la compilación local permite al usuario programar un archivo en HDL (.v) y junto con la asignación de pines de la FPGAn (.pcf) subir dichos archivos para que el proceso de compilación se realice en el servidor.

### 4.1 COMPILACIÓN LOCAL

La compilación local consiste en generar el archivo birstream desde el **software IceStudio**. Dicho software permite diseñar circuitos digitales de diferentes niveles de complejidad.

En este caso se deberá subir el archivo (.bin) y si la extensión es la permitida, se pasará a programar directamente. Para enviar los archivos se utiliza un formulario donde se indica la ruta del archivo y se pulsa el botón de envío. Solo se permiten extensiones de tipo \*.bin, que son las generadas para la programación de la FPGA.

The screenshot shows a web-based remote laboratory interface. At the top, there's a header with the UNED logo and the text 'Laboratorio para prácticas remotas' and 'FPGA IceZUM Alhambra'. On the right, it says 'ETS de Ingenieros Industriales'. Below the header is a navigation bar with links for 'Inicio', 'Demostración', 'Continuar', 'Acerca de...', 'Ayuda', and 'Cerrar sesión'. The main content area is divided into two main sections:

- SELECCIÓN DE PROYECTOS:** This section contains a list of projects ('Listado de proyectos') with their creation dates: 'leds.blm 16:13 23-09-2018' and 'setbit.blm 14:49 23-09-2018'. It also includes a note to select a project file or upload a new one, and a dropdown menu for baud rate selection ('- 9600 -'). Buttons for 'Continuar' and 'A programar' are present.
- FORMULARIO DE SELECCIÓN Y ENVÍO DE ARCHIVOS:** This section is enclosed in a red box. It asks for a file to be selected for upload ('Por favor, selecciona el archivo a subir:'), with a note that 'Ningún archivo seleccionado' is currently selected. A 'Enviar archivo' button is available. It specifies that '.bin' files are allowed ('Extensiones permitidas: .bin, archivo bitstream binario').

At the bottom of the page, there's a footer with the text 'Departamento de Ingeniería Eléctrica, Electrónica, Control, Telemática y Química Aplicada a la Ingeniería' and 'Resolución óptima 1280 x 800 ppp', along with the year 'UNED, 2018'.

Figura 5: Página de envío de archivos bitstream, LR IceZUM Alhambra.

Clicando en continuar se accederá a la página de programación de la placa, donde se podrá visualizar una imagen en tiempo real del Laboratorio Remoto.

## 4.2 COMPILACIÓN REMOTA

En el caso de que no se disponga de un ordenador dedicado, no se desee instalar la aplicación IceStudio o se acceda desde varios sitios diferentes, se puede usar la compilación remota. Para ello se tendrán que subir los archivos en código HDL - verilog (.v), y el archivo de asignación de pines (.pcf). El proceso de envío de archivos es igual que en el caso anterior, con la salvedad que las extensiones permitidas son todas las relacionadas con el diseño del proyecto.

El proceso de compilación de un archivo en lenguaje Verilog hasta la generación del archivo binario para grabar en la FPGA comprende las siguientes etapas

- Se parte de los ficheros fuente en verilog (.v) y la asociación de etiquetas con los pines de la placa, archivo con extensión .pcf
- Síntesis del archivo, con el programa Yosys (IceStorm Tools), a partir del archivo en verilog (.v) de generan el fichero netlist (.blif)
- Emplazado y rutado, arachne-pnr (IceStorm Tools). A partir del archivo .pcf y del netlist (.blif) se genera el bitstream en formato ASCII (.txt)
- Con Icepack se crea el bitstream binario (.bin)
- Envío del fichero binario (.bin) con Iceprog al dispositivo y comprobación de resultados.

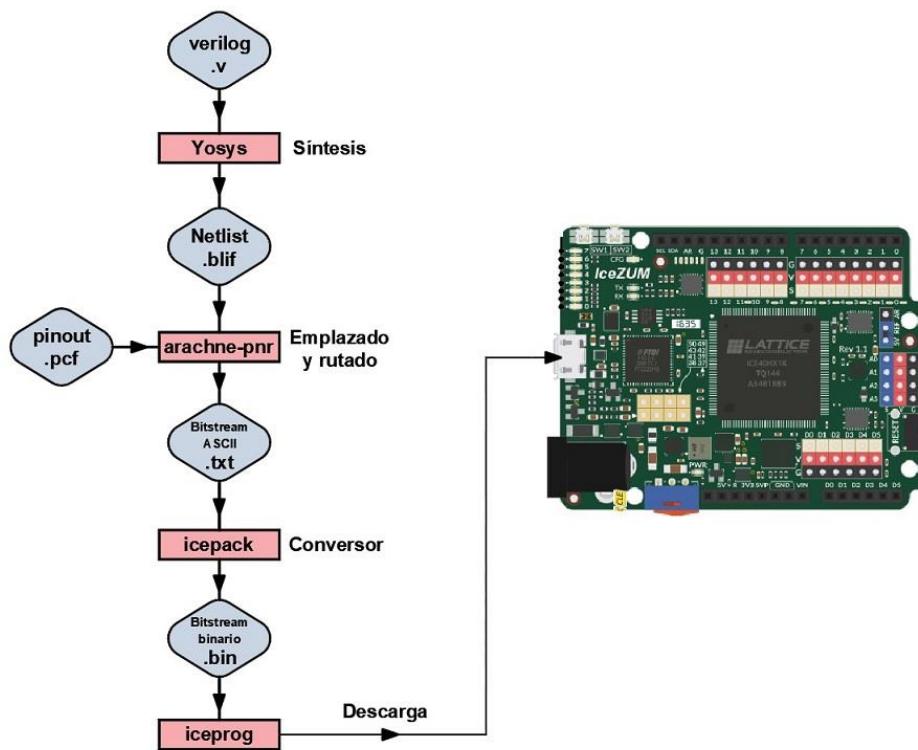


Figura 6: Proceso de compilación LR IceZUM Alhambra.

Dicho proceso se ha emulado en el diseño la web, que permite ir visualizando el avance en la compilación, hasta que se genera el archivo bitstream y se pasa a la página de programar.

Figura 7: Página Yosys –Síntesis-, compilación remota LR IceZUM Alhambra.

**arachne: emplazado y enrutado**

ETS de Ingenieros Industriales

|  |          |          |            |   |              |       |               |  |
|--|----------|----------|------------|---|--------------|-------|---------------|--|
| Inicio   | Atrás... | Ejecutar | Utilidades |   | Acerca de... | Ayuda | Cerrar sesión |  |
|  |          |          |            | yosys ➔ arachne ➔ icepack ➔ iceprog ➔   |              |       |               |  |
| <b>Proceso</b> <b>Estado</b><br>yosys  ✓ Encontrados archivos de etapa anterior. Todo correcto<br>arachne-pnr  ✘<br>icepack  ✘ |          |          |            |   |              |       |               |  |
| asociación de etiquetas con pines FPGA<br>Asegúrate que la asociación de pines es correcta:<br>set_io A 99                     |          |          |            | Comando a enviar<br>./toolchain-icestorm/bin/arachne-pnr -d 1k -p<br>./maria/setbit/setbit.pcf<br>./maria/setbit/setbit.blif -o-<br>./maria/setbit/setbit.txt |              |       |               |  |

Departamento de Ingeniería Eléctrica, Electrónica, Control, Telemática y Química Aplicada a la Ingeniería  
 Resolución óptima 1280 x 800 ppp

UNED, 2018

Figura 8: Página Arachne –Emplazado y enrutado-, compilación remota LR IceZUM Alhambra.

**icepack: generación de bitstream**

ETS de Ingenieros Industriales

|  |          |          |            |  |              |       |               |  |
|--|----------|----------|------------|--|--------------|-------|---------------|--|
| Inicio   | Atrás... | Ejecutar | Utilidades |  | Acerca de... | Ayuda | Cerrar sesión |  |
|  |          |          |            | yosys ➔ arachne ➔ icepack ➔ iceprog ➔  |              |       |               |  |
| <b>Proceso</b> <b>Estado</b><br>yosys  ✓ Encontrados archivos de etapa anterior. Todo correcto<br>arachne-pnr  ✓<br>icepack  ✘ |          |          |            |  |              |       |               |  |
| asociación de etiquetas con pines FPGA<br>set_io A 99  |          |          |            | Comando a enviar<br>./toolchain-icestorm/bin/icepack<br>./maria/setbit/setbit.txt<br>./maria/setbit/setbit.bin |              |       |               |  |

Departamento de Ingeniería Eléctrica, Electrónica, Control, Telemática y Química Aplicada a la Ingeniería  
 Resolución óptima 1280 x 800 ppp

UNED, 2018

Figura 9: Página Icepack –Generación bitstream-, compilación remota LR IceZUM Alhambra.

**icepack: generación de bitstream**

ETS de Ingenieros Industriales

|   |          |          |            |  |              |       |               |  |
|---|----------|----------|------------|--|--------------|-------|---------------|--|
| Inicio  | Atrás... | Ejecutar | Utilidades |  | Acerca de... | Ayuda | Cerrar sesión |  |
|   |          |          |            | yosys ➔ arachne ➔ icepack ➔ iceprog ➔  |              |       |               |  |
| <b>Proceso</b> <b>Estado</b><br>yosys  ✓ Están todas las etapas completadas. Puedes pasar a PROGRAMAR<br>arachne-pnr  ✓<br>icepack  ✓ |          |          |            |  |              |       |               |  |
| asociación de etiquetas con pines FPGA<br>set_io A 99   |          |          |            | Comando a enviar<br>./toolchain-icestorm/bin/icepack<br>./maria/setbit/setbit.txt<br>./maria/setbit/setbit.bin |              |       |               |  |

Departamento de Ingeniería Eléctrica, Electrónica, Control, Telemática y Química Aplicada a la Ingeniería  
 Resolución óptima 1280 x 800 ppp

UNED, 2018

Figura 10: Página Icepack –bitstream generado-, compilación remota LR IceZUM Alhambra.

## 5 PROGRAMACIÓN FPGA

Sea cual sea la forma elegida para trabajar con el laboratorio, una vez que se tiene el archivo (.bin) generado por la propia aplicación o enviado desde el ordenador del alumno, se pasa a programar si esta libre el programador.

Se accede a una página donde se puede visualizar el laboratorio remoto. En dicha página se puede programar la FPGA, o Resetearla, por si surgiera algún problema.

Cuando se abandone la página se reseteará igualmente. En dicha página hay un cuadro de texto que informará del éxito de la programación, o del Reset, así como de si hubiese algún problema.

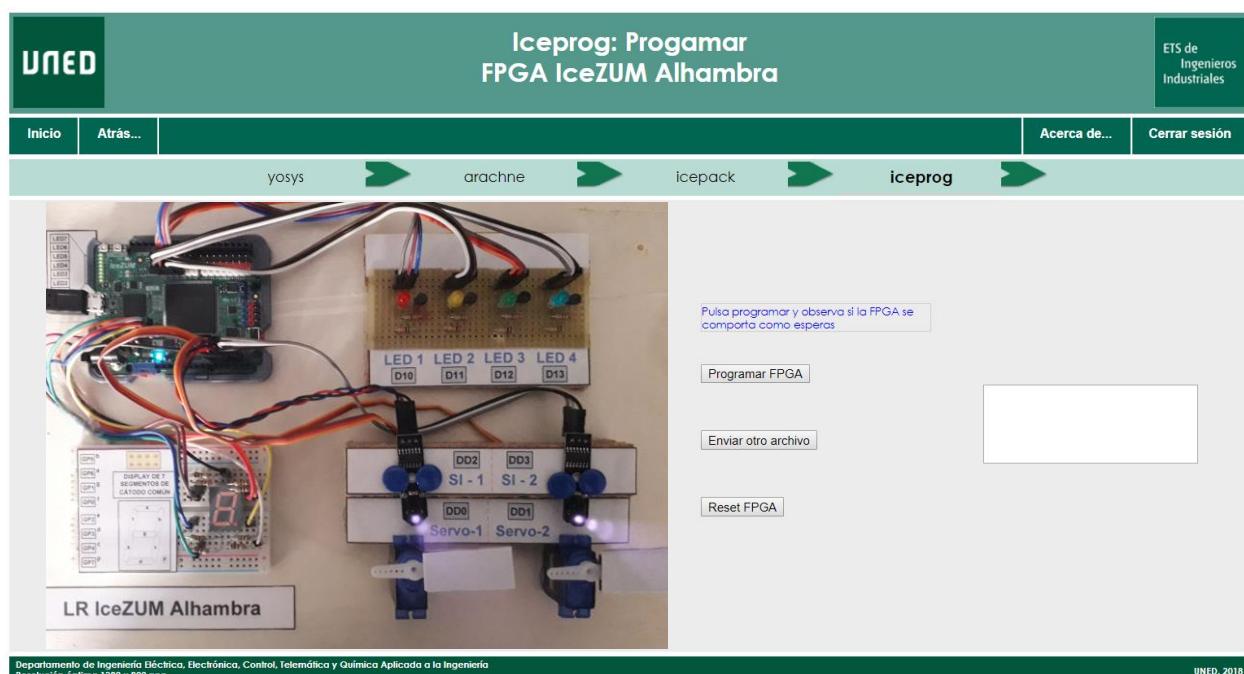


Figura 11: Página Iceprog –Programar-, LR IceZUM Alhambra.

## 6 LIMITACIONES DE USO

El laboratorio tiene limitado el tiempo de permanencia. Si un usuario está trabajando, cambia de página, comprueba registros generados o mantiene una actividad de cualquier forma no se verá afectado por ninguna limitación, pero si se mantiene inactivo en cualquiera de las páginas más de cinco minutos automáticamente es dirigido a la página de entrada al laboratorio para dejar sitio a otro usuario que quiera realizar prácticas. El trabajo realizado durante su estancia no se pierde y cuando vuelva a entrar podrá continuar con el proceso en el punto donde se quedó en la vez anterior.

## 7 PÁGINA DE UTILIDADES

Existe la posibilidad de consultar los registros creados durante la ejecución de los comandos de compilación desde la página de utilidades, accesible desde el menú superior.

Se han añadido dos enlaces a unos programas gratuitos para la comprobación de la sintaxis del archivo HDL y que además añade marcas de tiempo y cronogramas para la simulación temporal del archivo. Una vez que se ha ejecutado esta aplicación, con la otra se pueden generar gráficas y cronogramas completos de todas las señales internas que intervienen en la ejecución del código escrito, teniendo la posibilidad de comprobar cómo se comportara una vez programado en la FPGA (Icarus Verilog y GtkWave Profesor). En **Ayuda** se puede consultar un manual de instalación de estos programas, con un ejemplo de uso.



**Utilidades para simulación y síntesis.**

ETS de Ingenieros Industriales

Inicio
Atrás...
Acerca de...
Ayuda
Cerrar sesión

**Icarus Verilog**

Es un programa de licencia libre, que se puede usar tanto en plataformas windows como linux/unix.Icarus verilog crea un fichero ejecutable a partir del código Verilog. Al ejecutarlo se realiza la simulación. Los resultados se vuelcan a un fichero .vcd que se visualiza con la herramienta GTKWave. Esto nos permite inspeccionar las señales para comprobar su correcto funcionamiento

[Página oficial](#)

**GTKWave**

Este programa, igual que el anterior es de código abierto y permite la visualización de gráficas de tiempo para el análisis de diseños mediante volcados de verilog.

[Página oficial](#)

**Elige la práctica de la que quieras ver los registros creados.**

setbit ▾

Seleccionar práctica

Departamento de Ingeniería Eléctrica, Electrónica, Control, Telemática y Química Aplicada a la Ingeniería
UNED, 2018



**Busqueda de registros generados**

ETS de Ingenieros Industriales

Inicio
Atrás...
Utilidades
Acerca de...
Ayuda
Cerrar sesión

**Registros generados**

|                             |                         |
|-----------------------------|-------------------------|
| <a href="#">setbit.blif</a> | Registro de yoys        |
| <a href="#">setbit.txt</a>  | Registro de arachne-pnr |
| <a href="#">setbit.bin</a>  | icepack                 |

Se han encontrado 3 registros de setbit.

Figura 12: Página Utilidades-Consulta Registros generados, LR IceZUM Alhambra.

## 8 MENSAJES ADMINISTRADORES

Desde la página de información accesible desde la barra de menú en 'Ayuda', se tiene la posibilidad de enviar mensajes para realizar consultas o plantear dudas sobre el proceso a los Administradores. Por medio de un formulario con un cuadro de texto que permite hasta 150 caracteres. Hay que tener en cuenta que no es un buzón de sugerencias, por lo que se debe ser conciso y sobre el tema que nos ocupa.

Laboratorio Remoto FPGA  
IceZUM Alhambra

A continuación se ofrece una pequeña descripción de la página y enlaces con recursos relacionados con la FPGA IceZUM Alhambra. Si ya la has leído puedes [continuar](#), además puedes volver aquí desde cualquier página mediante el enlace de [casa](#) en el menú superior.

**Normas de funcionamiento.**

El objetivo de este laboratorio remoto es facilitar la realización de prácticas de asignaturas de programación de dispositivos lógicos programables. Aquí usamos la FPGA libre Ice ZUM Alhambra basada en iCE40HX1K-TQ144 de Lattice, cuya hoja de características se adjunta más abajo. Debido a que hay múltiples accesos a la página, el tiempo de permanencia está un tanto limitado; en caso de estar inactivo en una página aparecerá un mensaje advirtiendo del hecho y si no se realiza ninguna acción, a los 3 minutos se termina la sesión y se guarda lo realizado hasta ese momento. Se recomienda salir siempre con el enlace de "Cerrar sesión" ya que así se cierra correctamente la sesión, se guarda registro del trabajo realizado y a los demás usuarios se les evita una espera innecesaria.

Laboratorio para prácticas remotas  
FPGA IceZUM Alhambra

Mensajes antiguos

Enviar

Puedes escribir un mensaje de hasta 150 caracteres.

Departamento de Ingeniería Eléctrica, Electrónica, Control, Telemática y Química Aplicada a la Ingeniería  
Resolución óptima 1280 x 800 ppp

UNED, 2018

Figura 13: Envío de mensajes, LR IceZUM Alhambra.

Después de escribir un mensaje, cuando vuelva a acceder al laboratorio será informado del estado en que se encuentra, y si ha sido leído se mostrará la respuesta.

Elección del tipo  
de práctica a realizar

Hola maria : Seleccione el tipo de práctica a realizar

Tiene 12 mensaje/s sin responder  
No tiene mensajes

Compilación local

Realizada en el PC de usuario y envío del archivo \*.t

Compilación remota

Realizada en el laboratorio, enviando archivos de c

Tiene 11 mensaje/s sin responder  
No tiene mensajes

Respuesta número 1:  
Leído el 26-09-2018 por admin y la respuesta fue: Inténtalo otra vez, variando el valor de los parámetros

Cor

Cor

Departamento de Ingeniería Eléctrica, Electrónica, Control, Telemática y Química Aplicada a la Ingeniería  
Resolución óptima 1280 x 800 ppp

UNED, 2018

Figura 14: Respuestas de mensajes, LR IceZUM Alhambra.

También existe la posibilidad de recibir mensajes de los Administradores o profesores, que en este caso pueden aparecer de dos formas, como mensaje privado, si hay un solo destinatario del mensaje o público si es para todos los registrados. Los mensajes públicos serán usados para avisos referentes al funcionamiento del laboratorio y estarán visibles hasta las 12:00 horas del segundo día siguiente a su inserción, luego no se mostrara más.

Al clicar en 'mensaje' se abre una ventana nueva con el mensaje recibido, que al cerrar actualiza el estado del mensaje para que no vuelva a aparecer el aviso, aunque puede ser recuperado desde la página de envío de mensajes, en mensajes antiguos, ver Figura 13.

## 9 ACCESOS CON IDENTIFICADORES DUPLICADOS

En el caso de que dos usuarios utilicen el mismo identificador para acceder a la web en el mismo momento, será advertido de la situación. Tendrá la opción desde esta página de enviar un correo al administrador para informar de la situación.

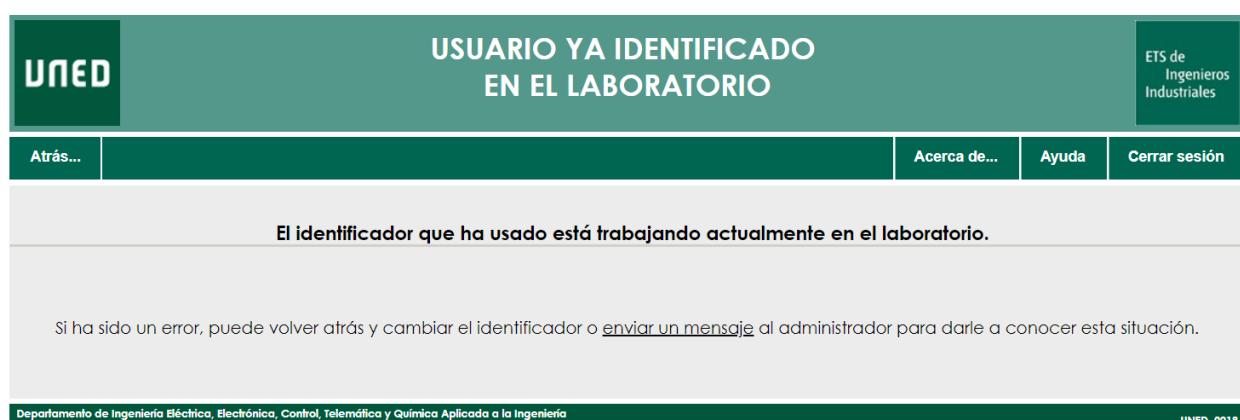


Figura 15: Usuario ya identificado, LR IceZUM Alhambra.

---

## A03-MANUAL DE ADMINISTRADOR

---

## ÍNDICE DE CONTENIDOS

---

|                                     |   |
|-------------------------------------|---|
| 1. INTRODUCCIÓN .....               | 3 |
| 2. ESTRUCTURA DE LA PÁGINA .....    | 3 |
| 2.1. MENÚ SUPERIOR .....            | 3 |
| 2.2. MENÚ LATERAL IZQUIERDO .....   | 4 |
| 3. MANTENIMIENTO .....              | 5 |
| 4. GENERACIÓN DE REPORTES .....     | 6 |
| 5. GESTIÓN DE ADMINISTRADORES ..... | 8 |

## ÍNDICE DE ILUSTRACIONES

---

|   |   |
|---|---|
| <i>Figura 1: Página principal administrador .....</i>                 | 3 |
| <i>Figura 2: Acceso de administradores al Laboratorio Remoto.....</i> | 4 |
| <i>Figura 3: Menú mantenimiento .....</i>                             | 5 |
| <i>Figura 4: Ventana de configuración .....</i>                       | 5 |

## 1. INTRODUCCIÓN

El presente manual explica el funcionamiento del laboratorio remoto cuando se accede al mismo como administrador, lo que permite, realizar el mantenimiento, generar reportes, limitar el uso, crear y eliminar administradores, o cambiar contraseñas.

## 2. ESTRUCTURA DE LA PÁGINA

Cuando la aplicación web se instala en el servidor se crea un usuario genérico para acceso inicial que puede ser eliminado posteriormente, una vez se haya creado uno o varios administradores más.

La página web reconoce a los usuarios administrador, y antes de acceder a la página de gestión solicita la contraseña de acceso.

Gestión del Laboratorio de FPGA Ice ZUM Alhambra

ETS de Ingenieros Industriales

Inicio Entrar a Laboratorio Utilidades Manual Administrador Acerca de... Ayuda Cerrar sesión

Mantenimiento

Configuración

Hacer copia datos

Restaurar backup

Usuarios on-line

Supervisión cámara

Rechazos producidos

Cierre curso académico

Reportes

Prácticas por usuarios

Uso del programador

Accesos por día

Prácticas terminadas

Actividad administradores

Usuarios registrados

Abandonos por inactividad

Reportes personalizados

Mensajes de usuarios

Administradores

Añadir Administrador

Eliminar Administrador

Contraseña

Página principal de gestión del Laboratorio

Elija alguna opción de los apartados del menú lateral o puede realizar prácticas desde el acceso Entrar a Laboratorio del menú superior.

Número de usuarios conectados: 1.

Número de usuarios en espera: 0.

Número de mensajes sin leer: 14.

Última copia de seguridad: Hace más de una semana que no se hace un backup.

Departamento de Ingeniería Eléctrica, Electrónica, Control, Telemática y Química Aplicada a la Ingeniería  
Resolución óptima 1280 x 800 ppp

UNED, 2018

Figura 1: Página principal administrador

### 2.1. MENÚ SUPERIOR

Una vez en la página principal, desde el menú superior se tiene acceso al laboratorio como un usuario normal. Si se opta por entrar al laboratorio se puede regresar a esta página desde el menú superior con la opción “Administración”, disponible solo para administradores.

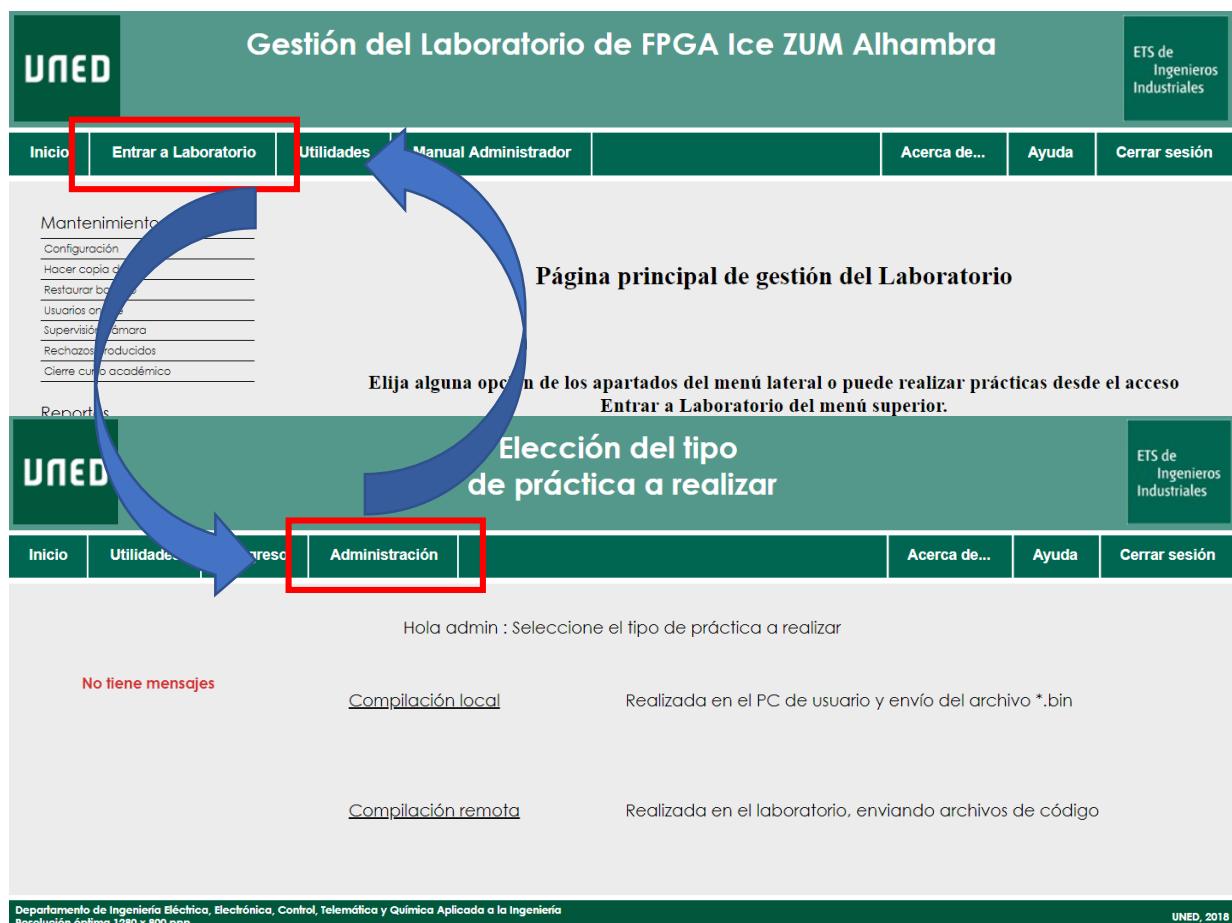


Figura 2: Acceso de administradores al Laboratorio Remoto

Los administradores no son contabilizados a la hora del límite propuesto de entradas, aunque siguen teniendo limitación de tiempo de inactividad en cada página.

En la página inicial de administración se muestra un resumen del estado actual del laboratorio, información sobre el número de usuarios conectados y los que están en espera si se da esta circunstancia, mensajes que han sido recibidos y que todavía no han sido contestados, así como del estado de las copias de seguridad periódicas que se realizan, con aviso si no se están realizando correctamente o si la fecha en que se hizo es demasiado antigua.

## 2.2. MENÚ LATERAL IZQUIERDO

Las diferentes acciones que se pueden realizar, se encuentran en el menú lateral izquierdo:

- Mantenimiento
- Reportes
- Administradores

### 3. MANTENIMIENTO

**Configuración:** Se ha previsto una forma de modificar el acceso a la base de datos, en el caso de que se quiera cambiar por avería del equipo servidor, modificación de la aplicación o reinstalación.

Aunque se hace la comprobación de si la base de datos que se introduce existe, hay que tener en cuenta que en el caso de que se varíen los datos y los nuevos introducidos no sean correctos el laboratorio dejara de funcionar y se tendrá que editar a mano el archivo de configuración "config\_bbdd.inc" del directorio raíz.

Otro apartado configurable del laboratorio es el límite de usuarios que pueden trabajar a la vez en página y los que pueden esperar. También es configurable el puerto serie al que se conecta la FPGA IceZUM Alhambra.

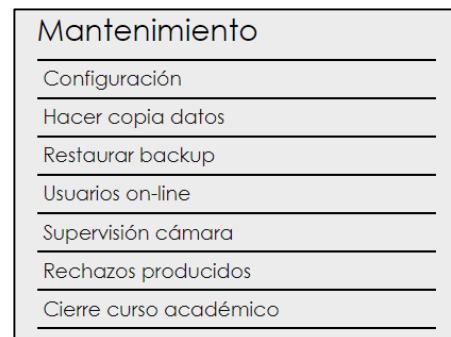


Figura 3: Menú mantenimiento

Figura 4: Ventana de configuración

**Hacer copia de datos:** Permite realizar una copia del directorio donde se almacena el trabajo realizado por los alumnos. Aunque está programado que se realice una copia cada semana.

**Restaurar backup:** Al igual que en el caso anterior, se puede restaurar la última copia realizada en caso de ser necesario, teniendo en cuenta que los datos existentes serán sobrescritos sin pedir confirmación.

**Usuarios on-line:** En la página principal de administración se muestra un número de personas conectadas, y accediendo a usuarios on-line se puede ver que usuarios son y qué actividad están realizando.

**Supervisión de cámara:** Desde este enlace se tiene acceso a la camara IP, sin interferir en la actividad que se esté realizando en ese momento para comprobacion de funcionamiento o en el caso de consulta de algún alumno.

**Rechazos producidos:** Informa de los rechazos que se han producido, con esta información se podrá valorar si es necesario aumentar el límite de usuarios.

**Cierre del curso académico:** Realiza el cierre del curso académico, realiza un reporte y resetea la base de datos.

## 4. GENERACIÓN DE REPORTES

La aplicación web permite generar reportes basados en los registros de actividad que los usuarios han ido haciendo en el laboratorio en los diferentes accesos que han realizado. De los más usuales se ha añadido su propio enlace:

**Prácticas por usuarios:** En este apartado están listadas todas las prácticas realizadas, agrupadas por usuario y ordenadas por número de práctica. Se indica el día y la hora que se ha accedido, la etapa en la que se ha trabajado y el resultado de la ejecución, si es que se ha llegado a realizar.

**Uso del programador:** Si se desea conocer el uso que ha tenido el programador, en este enlace se puede consultar información relativa al usuario que ha accedido, la practica sobre la que ha trabajado y el resultado de la programación, todo ello agrupado por días

**Accesos por día:** Si lo que se desea es saber quién ha accedido en un día concreto, se puede saber el identificador del usuario, el tiempo que ha permanecido en el laboratorio, la práctica en la que ha estado trabajando y la etapa que ha completado en el acceso.

**Prácticas terminadas:** Es de gran utilidad para la evaluación del trabajo realizado por los alumnos saber quién ha terminado alguna o varias prácticas y cuál ha sido o han sido. Se da información del día y la hora en que se ha completado.

**Actividad administradores:** Aunque no tiene finalidad docente, también se guarda registro de la actividad que han generado los accesos de los administradores al laboratorio, con menor cantidad de detalles que los precedentes pero con la suficiente información como para saber lo que se ha hecho durante el acceso.

También se guarda el hecho de realizar una copia de seguridad o restaurarla. En el caso de que un administrador realice prácticas, este trabajo será reflejado en los registros anteriores como parte generada por el propio laboratorio.

**Usuarios registrados:** Para conocer el listado completo de usuarios que están registrados en el laboratorio, incluidos los administradores, se muestra una lista de identificadores y nombres completos asociados.

**Abandonos por inactividad:** Se ha considerado interesante guardar información de los abandonos que se producen en el laboratorio debido a inactividad de los usuarios, ya que de ser un usuario que se repite con frecuencia puede ser un problema de conocimiento del funcionamiento, de errores de programación o de conexión, y se puede ofrecer ayuda o consultarle los problemas que le puedan surgir.

Aun en el caso de que los reportes expuestos hasta ahora no satisfagan las necesidades personales de un administrador, existe la posibilidad de filtrar toda esta información y hacerla más selectiva y reducir la cantidad a lo estrictamente necesario. Se empieza filtrando un nombre de usuario, después se elige un criterio de agrupamiento de entre los siguientes:

- Todos los accesos producidos.
- Accesos de una fecha.
- Practicas empezadas.
- Practicas terminadas.
- Uso del programador.

Dependiendo de cuál se elija, se habilitan otras casillas de selección para seguir completando el filtro, por ejemplo en el caso de seleccionar cualquiera de las opciones de prácticas se habilita la posibilidad de elegir una práctica en concreto, o si se quiere elegir una fecha se muestra una caja de texto para introducir la fecha de búsqueda.

### ***Envío de mensajes***

---

Desde este apartado de reportes se accede también a todos los mensajes enviados por los usuarios y responder a los que todavía no lo están. Un mensaje puede estar en dos estados, con lectura confirmada o sin confirmar, aunque a su vez puede tener respuesta o no. Para los mensajes generados automáticamente por el laboratorio frente a un acceso duplicado, no es necesario publicar ninguna respuesta, solo con aceptar que se ha leído el mensaje es suficiente.

También es posible desde la misma página enviar mensajes a un usuario determinado, para avisarle de un error que está cometiendo por ejemplo, o a todos los registrados, por tareas de mantenimiento de la plataforma o algo similar.

El aviso a los usuarios llega de la misma manera cualquiera que sea el tipo, pero el segundo tipo tiene una validez determinada; como no se sabe de antemano cuantos usuarios van a entrar al laboratorio ni en qué periodo de tiempo, se ha fijado una validez del aviso de dos días, y a partir de las 12:00 horas del segundo día desde su inserción dejara de aparecer el aviso. Es por ello que su uso estará limitado a dar a conocer tareas de mantenimiento, cierres temporales de alguna parte del laboratorio, cambios acometidos, etc...

Cuando un mensaje sea leído se avisara al usuario que lo escribió la próxima vez que acceda al laboratorio, mostrándole la respuesta ofrecida si la hay.

De todos los reportes predefinidos se pueden generar documentos en formato PDF, pulsando en el botón que existe en todas las páginas.

## 5. GESTIÓN DE ADMINISTRADORES

Por último, está el apartado para poder añadir nuevos administradores del laboratorio, o eliminar aquellos que no sean necesarios. Se debe tener en cuenta que no hay diferentes niveles de privilegios, es decir, todos los que figuren en la base de datos como usuarios con acceso a la parte de gestión tendrán los mismos privilegios.

Para añadir un nuevo usuario con privilegios se accede mediante el menú lateral a **añadir administrador**, donde hay que indicar el identificador que se va a usar, el nombre completo para facilitar la identificación posterior y la contraseña. Si el identificador ya está en uso el advertido y obligado a cambiarlo.

Para la eliminación será necesario introducir el identificador que se desea eliminar, y tras aceptar y confirmar la eliminación, el registro es borrado de la base de datos.

También existe la posibilidad de cambiar la contraseña, en el caso de que se tengan sospechas de acceso indebidos o por seguridad, sin motivo alguno. Es necesario introducir la contraseña actual y la nueva dos veces para confirmar errores en la escritura.

---

## A04-GUÍA DE PRÁCTICAS

---

## ÍNDICE DE CONTENIDOS

---

|  |    |
|--|----|
| 1. INTRODUCCIÓN .....                                    | 3  |
| 2. OBJETIVOS .....                                       | 3  |
| 3. DOCUMENTACIÓN COMPLEMENTARIA .....                    | 3  |
| PARTE I. DISEÑO DE CIRCUITOS DIGITALES - ICESTUDIO ..... | 4  |
| I.1 FPGA ICEZUM ALHAMBRA .....                           | 4  |
| I.2 ENTORNO DE DESARROLLO ICESTUDIO .....                | 7  |
| I.3 PRÁCTICAS PROPUESTAS .....                           | 7  |
| PRACTICA 1. ENCENDIENDO LEDS .....                       | 8  |
| PRACTICA 2. PERIFÉRICOS .....                            | 10 |
| PRACTICA 3. MULTIPLEXACIÓN .....                         | 15 |
| PRACTICA 4. DISPLAY DE 7 SEGMENTOS .....                 | 17 |
| PARTE 2. INTRODUCCIÓN AL LENGUAJE HDL – VERILOG .....    | 19 |
| II.1 LENGUAJE DE DESCRIPCIÓN HARDWARE – VERILOG .....    | 19 |
| II.2 HERRAMIENTAS DE SIMULACIÓN .....                    | 19 |
| II.3 HERRAMIENTAS DE COMPILACIÓN .....                   | 20 |
| II.4 FLUJO DE TRABAJO .....                              | 21 |
| II.5 PRACTICAS PROPUESTAS .....                          | 21 |
| PRACTICA 5. CONTADOR .....                               | 22 |
| PRACTICA 6. PRESCALER .....                              | 24 |
| PRACTICA 7. REGISTRO DE 4 BITS .....                     | 26 |
| PRACTICA 8. REGISTRO DE DESPLAZAMIENTO .....             | 28 |
| 4. BIBLIOGRAFÍA .....                                    | 32 |
| SOLUCIONES PRÁCTICAS .....                               | 33 |
| SOLUCIÓN PRÁCTICA 1 – ENCENDIENDO LEDS .....             | 33 |
| SOLUCIÓN PRÁCTICA 2 – PERIFÉRICOS .....                  | 34 |
| SOLUCIÓN PRÁCTICA 3 – MULTIPLEXACIÓN .....               | 36 |
| SOLUCIÓN PRÁCTICA 4 – DISPLAY DE 7 SEGMENTOS .....       | 38 |
| SOLUCIÓN PRÁCTICA 5 – CONTADOR .....                     | 39 |
| SOLUCIÓN PRÁCTICA 6 – PRESCALER .....                    | 39 |
| SOLUCIÓN PRÁCTICA 7 – REGISTRO DE 4 BITS .....           | 40 |
| SOLUCIÓN PRÁCTICA 8 – REGISTRO DESPLAZAMIENTO .....      | 41 |

## ÍNDICE DE ILUSTRACIONES

---

|  |    |
|--|----|
| <i>Figura 1: Placa IceZUM Alhambra.....</i>  | 5  |
| <i>Figura 2: Pinout de la placa ICEZUM Alhambra.....</i>                                 | 6  |
| <i>Figura 3: Conectores de la placa ICEZUM Alhambra.....</i>                             | 10 |
| <i>Figura 4: Esquema de conexión LED externo.....</i>                                    | 11 |
| <i>Figura 5: Posición servos.....</i>  | 12 |
| <i>Figura 6: Servo Tower Prp SG-90.....</i>  | 13 |
| <i>Figura 7: Esquema display de 7 segmentos.....</i>                                     | 17 |
| <i>Figura 8: Esquema de conexión del display de 7 segmentos a los pines de FPGA.....</i> | 18 |
| <i>Figura 9: Herramientas usadas para crear el bitstream .....</i>                       | 20 |
| <i>Figura 10: Esquema COUNTER – pines de conexión.....</i>                               | 23 |
| <i>Figura 11: Prescaler.....</i>   | 24 |
| <i>Figura 12: Esquema PRESCALER – pines de conexión.....</i>                             | 25 |
| <i>Figura 13: Registro de N bits.....</i>  | 26 |
| <i>Figura 14: Esquema REGISTRO DE 4 BITS – pines de conexión.....</i>                    | 27 |
| <i>Figura 15: Esquema REGISTRO DE DESPLAZAMIENTO.....</i>                                | 28 |
| <i>Figura 16: Diagrama de bloques componente desplazamiento.v.....</i>                   | 29 |
| <i>Figura 17: Registro de desplazamiento – pines de conexión.....</i>                    | 30 |

## 1. INTRODUCCIÓN

Las prácticas que a continuación se detallan se realizarán sobre la plataforma Laboratorio Remoto FPGA libre IceZUM Alhambra, dicha plataforma permite programar desde casa la FPGA IceZUM Alhambra.

La presente guía de prácticas se elabora con objeto de facilitar y servir de orientación en el proceso de aprendizaje del diseño de circuitos digitales y la programación bajo lenguaje de descripción hardware Verilog, simulación y síntesis de circuitos digitales sobre FPGAs, para ello se muestran aspectos relevantes como la estructura de una FPGA y las características relevantes de lenguaje HDL Verilog y del entorno de Icestudio.

## 2. OBJETIVOS

El objetivo de las prácticas es que el alumno culmine el proceso de diseño, programación y simulación de los circuitos digitales con el proceso de utilización de herramientas de síntesis, la implementación y por último la verificación de un dispositivo real sin necesitar demasiado tiempo para familiarizarse con los complejos entornos de síntesis. Para ello deberá:

- Implementar circuitos que den solución a los problemas planteados
- Usar Icestudio
- Comprender el proceso de compilación, simulación y configuración de la FPGA
- Usar el lenguaje de descripción hardware – Verilog

## 3. DOCUMENTACIÓN COMPLEMENTARIA

Manual de usuario Laboratorio Remoto de la FPGA IceZUM Alhambra

Manual de Uso IceStudio

Manual Verilog

Manual de Uso Icarus Verilog+GTKWave

**Toda esta documentación se encuentra en la página de Ayuda del Laboratorio remoto IceZUM Alhambra.**

# PARTE I. DISEÑO DE CIRCUITOS DIGITALES - ICESTUDIO

## I.1 FPGA ICEZUM ALHAMBRA

IceZUM es una FPGA diseñada con los mismos patrones que Arduino, por lo que se puede utilizar para su ampliación todos los “escudos” (shields) que existen en el mercado para Arduino y es un dispositivo de los denominados de Open Hardware.

El hardware de la placa IceZUM Alhambra cuenta con los siguientes dispositivos:

- ❖ **FPGA:** Lattice iCE40HX1K-TQ144 [2]. En esta FPGA hay 1280 Celdas lógicas, formadas cada una de ellas por una Look-up-table (LUT) y un flip-flop, 64 Kbits de memoria RAM y un PLL. Esta es una FPGA realmente pequeña y de bajo coste, con recursos lógicos reducidos pero suficientes para la realización de sistemas de nivel de complejidad reducido.
- ❖ **RELOJ:** Reloj MEMS a 12 MHz.
- ❖ **INTERRUPTOR:** Cuenta con un pequeño interruptor para desactivar los pines.
- ❖ **REGULADOR DE VOLTAJE:** Permite alimentar la placa con un rango de 6 a 17 voltios.
- ❖ **CORRIENTE DE ENTRADA:** Permite hasta una corriente de 3 amperios, útil para alimentar actuadores como, por ejemplo, motores de robots.
- ❖ **PINES:** Cuenta con 20 pines de Entrada/Salida a 5V y 10 a 3.3V.
- ❖ **INTERFAZ:** Programación mediante un USB micro-B, FTDI 2232H para programación e interfaz UART con un PC.
- ❖ **LEDS:** La placa cuenta con 8 leds que el usuario puede utilizar en sus programas.
- ❖ **PULSADORES:** Dispone de dos pulsadores utilizables por el usuario.
- ❖ **ANALÓGICO:** Cuenta con cuatro entradas analógicas conectadas a un convertidor analógico/digital ADC QFN16, con una precisión de 12 bits, conectado a la FPGA con un bus I2C propio.
- ❖ **PROTECCIÓN:** La placa cuenta con protección contra cortocircuitos y polaridad inversa.

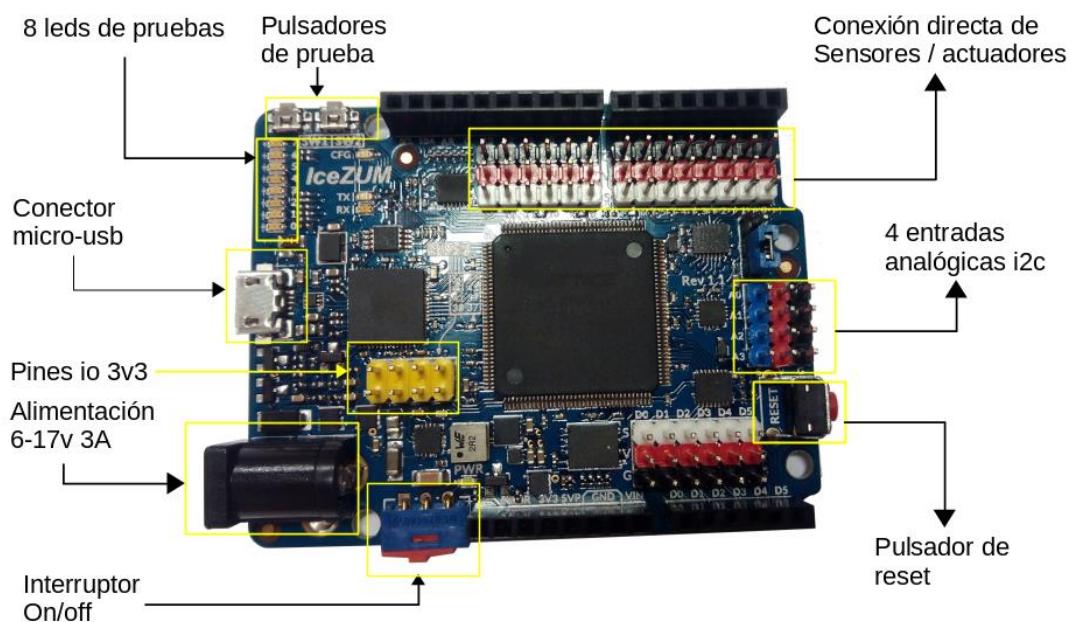


Figura 1: Placa IceZUM Alhambra  
Fuente: FPGAwars

Además de todo esto, el fabricante proporciona datos sobre la placa, tales como el pinout, el PCF (fichero donde aparecen los pines restringidos de la placa y su uso), la hoja de características técnicas de la placa (datasheet), o reglas de diseño de la placa.

# iCEZUM

## Alhambra

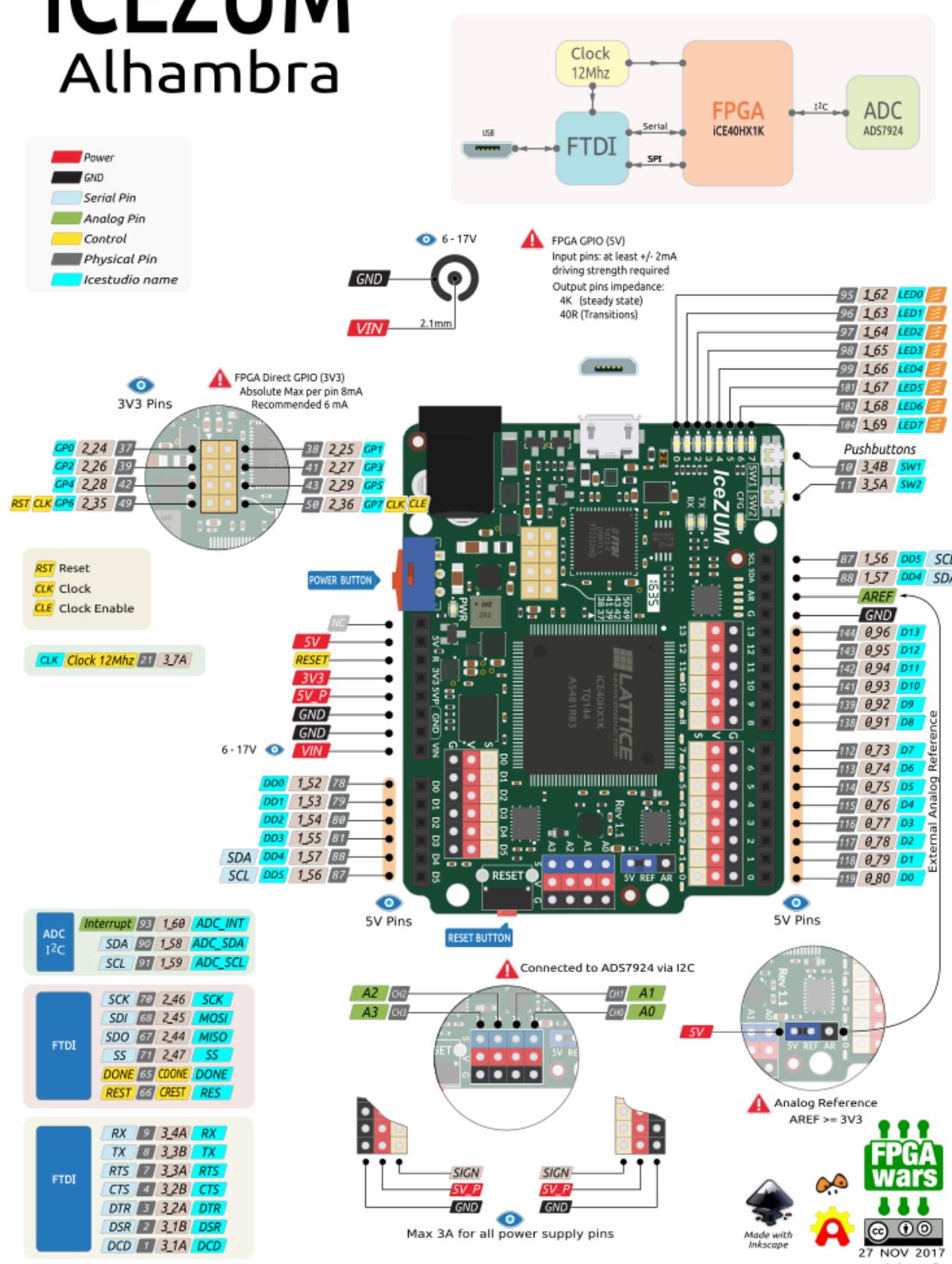


Figura 2: Pinout de la placa ICEZUM Alhambra.  
Fuente: FPGAwars

Por otra parte, el entorno de desarrollo que suministra el fabricante de la placa es el denominado IceStudio. Cabe destacar de él su máxima simplicidad, ofreciendo una interfaz que permite instalar todas las herramientas necesarias, tanto para la síntesis de HDL, como los drivers FTDI (Future Technology Devices International, circuito USB 2.0 a UART) necesarios para programar la IceZUM.

## I.2 ENTORNO DE DESARROLLO ICESTUDIO

Icestudio es una herramienta para diseño y síntesis de circuitos digitales en FPGAs libres, creada por Jesús Arroyo. Está programada en nodejs. Es software libre y multiplataforma. Corre en los sistemas GNU/Linux, Mac OS y Windows. La interfaz ofrece una biblioteca de objetos ya creados, tales como:

- ❖ ELEMENTOS BÁSICOS: Pines de entrada, de salida, comentarios, constantes y módulos con las entradas y salidas programables en Verilog.
- ❖ BIT: Se puede añadir un bit tanto a 1 como a 0.
- ❖ CONFIGURACIÓN: En este menú desplegable se puede encontrar configuraciones pull-up para los pines y buffers tri-estado.
- ❖ LÓGICA: En este menú desplegable se pueden encontrar dispositivos tales como multiplexores, demultiplexores, módulos para controlar visualizadores de 7 segmentos, todo tipo de puertas lógicas y algunos tipos de flip-flops.

Antes de continuar se recomienda al alumno leer con atención el [MANUAL DE ICESTUDIO](#) que se encuentra en la página web del laboratorio.

En dicho manual se explica cómo instalar el programa y las órdenes principales para comenzar a usarlo.

## I.3 PRÁCTICAS PROPUESTAS

En esta primera parte se realizarán cuatro prácticas con Icestudio, con la finalidad de que el alumno comience a diseñar circuitos sencillos en la FPGA, mejorando su intuición y comenzando con el lenguaje HDL - verilog

## PRACTICA 1. ENCENDIENDO LEDS

### **Bits constantes**

Utilizando Icestudio.

**Construir un circuito que encienda el LED 3.**

**En archivo/exportar/bitstream, exportar a archivo con extensión .bin**

**Cargar en la placa y comprobar**

### **Superposición de circuitos**

El pensamiento hardware tiene dos características que le diferencian del pensamiento algorítmico:

- Se piensa en espacio: los circuitos ocupan un espacio físico
- Las cosas suceden en paralelo

Esto permite combinar circuitos, y hacer que diferentes acciones ocurran simultáneamente.

A esta propiedad se le llama **superposición de circuitos**. Sólo es aplicable a circuitos que son independientes. Aplicando esta superposición, se consigue que un circuito tenga la suma de comportamientos de sus integrantes.

**Construir un circuito en que se encienda los LEDs 0,2,4,6**

**En archivo/exportar/bitstream, exportar a archivo con extensión .bin**

**Cargar en la placa y comprobar**

**Comentar resultados**

### **LEDs Parpadeantes**

Para conseguir que un LED parpadee hay que enviar las siguiente secuencia de bits (1-0-1-0-1-0 ... ) a dicho. Para ello se diseñará un bloque en lenguaje HDL – Verilog que haga esa función.

Icestudio ofrece la posibilidad de crear bloques donde introducir código. De esta forma se pueden personalizar los diseños.

Para realizar esta parte de la práctica, se seguirán los siguientes pasos:

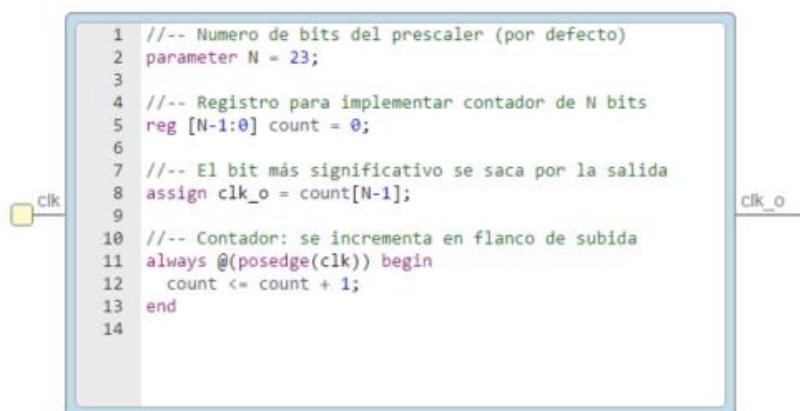
### PASO 1. Ir a menú **Básico/ código**

**PASO 2.** En ventana que aparece introducir la siguiente información, y clicar **OK**:

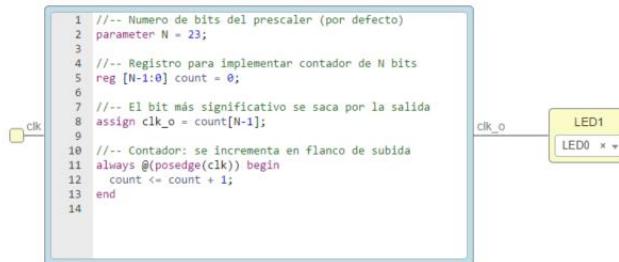
- puertos de entrada: clk
- puerto de salida: clk\_o
- parámetros

|                                  |
|----------------------------------|
| Introduce los puertos de entrada |
| clk                              |
| Introduce los puertos de salida  |
| clk_o                            |
| Introduce los parámetros         |
|                                  |
| OK CANCELAR                      |

**PASO 3.** En el bloque que aparece, introducir el siguiente código:



### PASO 4. Conectar el LED 0



### PASO 5. Comprobar y Cargar en la Placa

Variando el parámetro **N** del bloque de código se puede conseguir que el LED parpadee a diferentes velocidades.

**Hacer un circuito digital en el 4 LEDs de la icezum alhambra parpadeen cada uno a una velocidad, según esta tabla:**

**LED 0: N = 20**

**LED 2: N = 22**

**LED 4: N = 24**

**LED 6: N = 26**

**En archivo/exportar/bitstream, exportar a archivo con extensión .bin**

**Cargar en la placa y comprobar**

**Comentar resultados**

## PRACTICA 2. PERIFÉRICOS

### Introducción

Hay muchos periféricos, tanto de entrada como de salida, y que permiten realizar diseños muy variados relacionados con la robótica.

En esta práctica se verán LEDs Externos, servos y sensores infrarrojos.

### Pines de la Icezum Alhambra

La placa Icezum Alhambra tiene 4 tipos de pines para conectarse al exterior, resumidos en este diagrama:

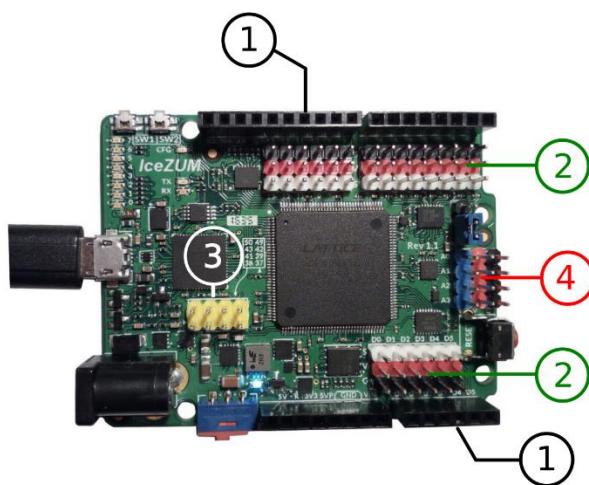


Figura 3: Conectores de la placa ICEZUM Alhambra.

Fuente: FPGAwars

Estos conectores son:

1. **Conectores de I/O hembra de 5 Voltios:** Son los típicos conectores de Arduino que sirven para conectar cables macho y shields de expansión. Se encuentran en la parte superior e inferior de la placa, como se muestra en la figura. En total hay 31 pines, de los cuales 20 son de I/O= 5v. Los pines de I/O superiores se denominan D13 - D0 en Icestudio, y los inferiores DD5 - DD0. Todos estos pines tienen la característica de que son de 5v.
2. **Conectores de I/O macho de 5 Voltios, con alimentación y masa:** Son 20 pines de I/O, machos, duplicados de los 20 pines I/O de los conectores hembra de 5v. Cada uno de estos pines incluye uno adicional de alimentación (+5v) y masa (GND). En total son tres pines separados por colores. El pin blanco contiene la señal de I/O, el rojo la alimentación de 5v y el negro la masa (GND). Tienen esta disposición para poder conectar directamente servos y sensores. Se denominan igual que sus duplicados hembra: D0 - D13 para los de la parte superior y DD0 - DD5 en la parte inferior.
3. **Conectores de I/O macho de 3.3 Voltios:** Son 8 pines macho dispuestos en dos filas de 4 pines, de color amarillo. Los 8 pines están conectados directamente a pines de la FPGA, que trabajan a 3.3V. Los pines hembra y macho anteriores están conectados a la FPGA a través de conversores de nivel. Se denotan con la nomenclatura GP0 - GP7.

- 4. Conectores macho para entradas analógicas:** La Icezum Alhambra tiene 4 entradas analógicas situadas en la parte derecha, accesibles a través de pines macho. Cada entrada analógica (Azul) tiene un pin adicional de alimentación (rojo, 5v) y otro de masa(negro, GND). Estas tiras de 3 pines permiten conectar directamente sensores de Luz o potenciómetros. La FPGA NO tiene entradas analógicas, por lo que en la Icezum Alhambra hay un conversor Analógico-digital (A/D) que se conecta a la FPGA a través de un bus I2C. Por ello, para leer las entradas analógicas es preciso incluir en nuestros diseños de un circuito capaz de leer el bus I2C

### **LEDs externos**

En el laboratorio remoto, se han conectado LEDs externos a los Conectores de I/O macho de 5 Voltios, con alimentación y masa. En concreto se han colocado 4 a los conectores **D13, D12, D11 y D10**.

Los conectores de 5v están conectados a los de la FPGA a través de unos conversores de nivel (3.3v <> 5v). Para que el LED se ilumine con más intensidad se realiza la conexión a través de un transistor. El esquema de conexión es el siguiente:

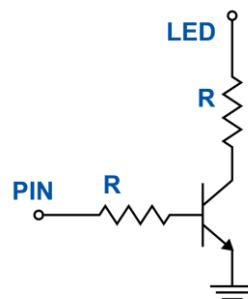


Figura 4: Esquema de conexión LED externo.

Al enviarse un 1 por el pin de la FPGA, se activa el transistor, que funciona como un interruptor, conectando la resistencia a GND y cerrando el circuito.

Teniendo en cuenta que su funcionamiento es similar al de los LEDs internos se pide:

**Construir un circuito en que se encienda los 4 LEDs externos**

**En archivo/exportar/bitstream, exportar a archivo con extensión .bin**

**Cargar en la placa y comprobar**

**Construir un circuito con un LED externo que parpadee**

**En archivo/exportar/bitstream, exportar a archivo con extensión .bin**

**Cargar en la placa y comprobar**

**Comentar resultados**

## Controlando Servos

En el laboratorio remoto, se han instalado Servos conectados a los Conectores de I/O macho de 5 Voltios, con alimentación y masa. En concreto se han colocado 2 a los conectores **DD0** y **DD1**.

Los servos son actuadores cuyo eje de salida se puede fijar a una posición concreta. Se usan mucho en robots educativos y de investigación, para implementar pinzas o articulaciones. Se presenta el controlador más simple para moverlos: El servoBit, que permite posicionar el servo en dos posiciones, según el bit recibido por su entrada

Para llevar la cabeza del servo a una posición determinada hay que enviar una señal de control especial. Se trata de un pulso de 5v, de anchura variable, que se envía periódicamente, cada 20ms (frecuencia de 50Hz)

El ancho del pulso es el que determina la posición a la que se moverá el servo. Los valores dependen del modelo de servo, pero típicamente un pulso de 1ms lleva el servo al extremo derecho y uno de 2ms al izquierdo. Valores entre 1ms y 2ms lo posicinal linealmente entre 0 y 180 grados

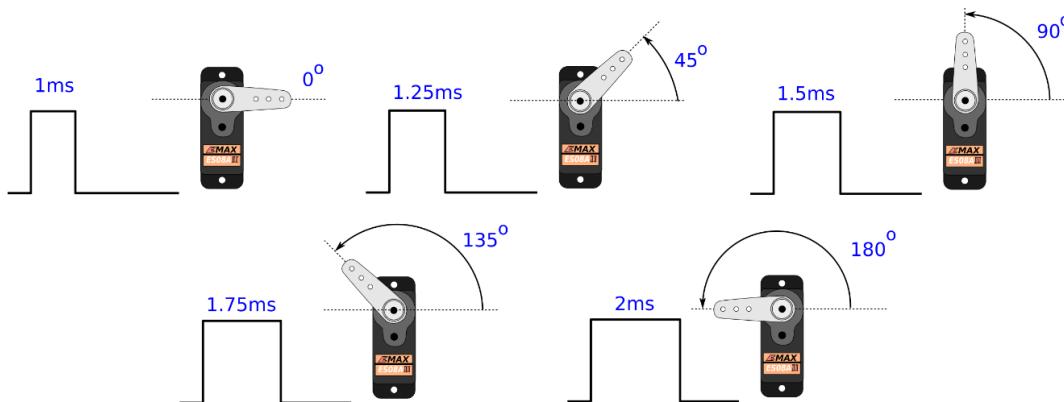


Figura 5: Posición servos.  
Fuente: FPGAwars

Los valores exactos dependen del modelo de servo utilizado. Lo importante es saber que el ángulo en el que se posiciona el servo es directamente proporcional al tiempo del ancho del pulso, y que este pulso se tiene que repetir periódicamente. Si se deja de repetir, el servo deja de ejercer fuerza y se puede mover con la mano a cualquier posición

Los servos se pueden colocar en cualquier posición dentro de su rango de 180 grados. Sin embargo, se va a utilizar el más sencillo: dos posiciones, separadas un ángulo de 90 grados. El controlador será mínimo: un bloque que tiene una única entrada y que según su valor mueve el servo a una posición (0) o a otra (1).

Al recibir un bit a 0 por su entrada, se posicionarán en la derecha. La posición absoluta depende de cómo esté colocada su cabeza. Al recibir un 1 se moverá 90 grados en sentido antihorario

Cada modelo de servo tiene unas características de la señal de control diferentes, por eso para poder usar los servos instalados en el laboratorio remoto hay que cargar la colección **IceZUM\_LR\_UNED**, que se puede descargar de la página web del Laboratorio Remoto.

Para cargar la colección ir a Icestudio desde la opción Herramientas/Colecciones/Añadir, si esta opción no funcionara, también se podría descomprimirla directamente dentro de la carpeta **Collections que se encuentra en equipo - OS(C:) - usuarios - USER - .icestudio**, el proceso es equivalente.

En esta colección encontraremos el controlador para el servo: TowerPro SG-90, que es que está instalado en Laboratorio Remoto



Figura 6: Servo Tower Prp SG-90.

Fuente: FPGAwars

En el menú **Varios/Servos/ServoBit-90**. Se puede seleccionar el controlador del servo.

Se pide:

**Hacer un circuito digital conectando el servo instalado en la icezum alhambra para que se mueva entre las dos posiciones 0 y 1, enviando la secuencia (0 – 1 – 0 – 1 ...) a la entrada del controlador, (utilizar bloque que hace parpadear LEDs) tomando N = 24.**

**En archivo/exportar/bitstream, exportar a archivo con extensión .bin**

**Cargar en la placa y comprobar**

**Comentar resultados**

### Sensores de infrarrojos

En el laboratorio remoto, se han conectado 2 sensores infrarrojos, justo debajo del recorrido que realizan los Servos, éstos están conectados a los Conectores de I/O macho de 5 Voltios, con alimentación y masa. En concreto se han colocado a los conectores **DD2 y DD3**.

Los sensores de infrarrojos (IR) permiten detectar la presencia de un objeto que está delante de ellos, a un centímetro aproximadamente. Constan de un emisor y un receptor.

El emisor es un LED infrarrojo. Es un LED que vez de emitir luz visible, la emite en el espectro infrarrojo, que no es visible. El emisor está constantemente emitiendo, cuando el sensor está alimentado. Esto se puede observar mirando el sensor a través de la cámara del teléfono móvil: se puede apreciar un color azulado/violeta.

El receptor es un fototransistor, que se activa cuando recibe luz infrarroja. La detección de objetos se hace por reflexión. La luz infrarroja que está constantemente generando el emisor se refleja en el objeto y llega al receptor, que nos devuelve un 1 (objeto detectado). Si por el contrario no hay objeto, la luz infrarroja se pierde y no regresa al receptor. En este caso el sensor nos devuelve un 0 (Objeto no detectado)

Estos sensores también se pueden utilizar para distinguir el blanco del negro en una superficie plana, y construir así robots que puedan por ejemplo seguir un camino negro. Cuando el sensor está sobre blanco, la luz del emisor se refleja completamente, devolviendo un 1. Cuando está

sobre la parte negra, no hay reflexión. El negro absorbe la luz, y no la refleja, por lo que no llega nada al receptor y devuelve un 0.

Se pide:

**Hacer un circuito digital conectando un servo instalado en la icezum alhambra para que se mueva entre las dos posiciones 0 y 1, enviando la secuencia (0 – 1 – 0 – 1 ...) a la entrada del controlador, (utilizar bloque que hace parpadear LEDs) tomando N = 24. Empleando la propiedad de superposición: activar el sensor infrarrojo para que se active uno de los LEDs externos.**

**En archivo/exportar/bitstream, exportar a archivo con extensión .bin**

**Cargar en la placa y comprobar**

**Comentar resultados**

## PRACTICA 3. MULTIPLEXACIÓN

### ***Introducción***

Cuando se tienen dos circuitos independientes, que funcionan correctamente de forma separada, se pueden cargar en la misma FPGA y funcionarán los dos a la vez, aprovechando el paralelismo que hay en el hardware, tal y como se ha realizado en ejercicios anteriores.

Pero si los circuitos no son independientes y acceden a la misma salida, hay que utilizar la multiplexación, que consiste en la combinación de dos o más canales de información en un solo medio de transmisión usando un dispositivo llamado multiplexor.

Los multiplexores permiten combinar circuitos que usan las mismas salidas.

### ***Multiplexor 2 a 1***

Un multiplexor de 2 a 1 tiene 2 canales de entrada, una entrada de selección y una salida. Cuando por la entrada de selección hay un 0, por la salida sale el canal 0 y cuando la entrada de selección es 1, lo que sale son los bits del canal 1

### ***Multiplexor de 4 a 1***

Los multiplexores 4 a 1 tienen 4 canales de entrada y 2 bits de selección. El funcionamiento es el mismo que el de los multiplexores 2:1, pero ahora se selecciona entre los cuatro canales. El valor de los dos bits de selección determina cuál de los canales es el que se saca por la salida. Por cada uno de estos canales llegan bits genéricos, que pueden estar a 1 ó 0.

### ***Multiplexor en IceStudio***

Como ya se ha dicho en apartados anteriores, IceStudio ofrece la posibilidad de crear bloques donde introducir código. De esta forma se pueden personalizar los diseños.

Para crear un bloque **multiplexor 2:1** en IceStudio, se seguirán los siguientes pasos:

#### **PASO 1. Ir a menú *Básico/ código***

#### **PASO 2. En ventana que aparece introducir la siguiente información, y clicar **OK**:**

- puertos de entrada: i0, i1, sel
- puerto de salida: o
- parámetros

Introduce los puertos de entrada

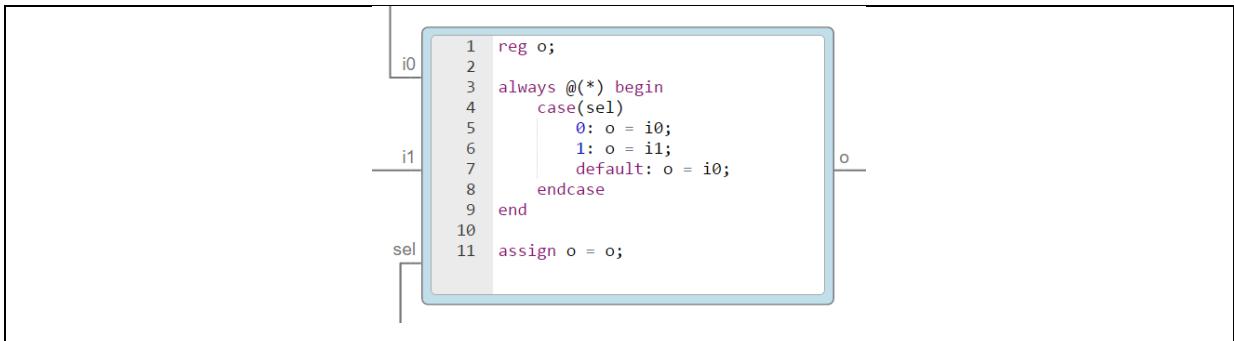
i0, i1, sel

Introduce los puertos de salida

o

Introduce los parámetros

#### **PASO 3. En el bloque que aparece, introducir el siguiente código:**



**Se pide:**

Diseñar un limpiaparabrisas a dos velocidades. Usar un servo para simular un limpiaparabrisas de un coche, que se mueva a dos velocidades: lenta ( $N=25$ ) y rápida ( $N=23$ ).

En la entrada de selección (sel) colocar la secuencia (0 – 1 – 0 – 1 ...) a la entrada del controlador, (utilizar bloque que hace parpadear LEDs) tomando  $N = 27$ .

Coneectar el LED0 a sel para controlar el estado de la secuencia de bits

En archivo/exportar/bitstream, exportar a archivo con extensión .bin

Cargar en la placa y comprobar

Implementa un bloque con el código de un multiplexor 4:1

Diseña un circuito con un LED en cuatro modos

MODO 1 → Encendido (bit 1)

MODO 2 → Apagado (bit 2)

MODO 3 → PARPADEO LENTO ( $N=20$ )

MODO 4 → PARPADEO RÁPIDO ( $N=22$ )

En las entrada de selección colocar la secuencia (0 – 1 – 0 – 1 ...) a la entrada del controlador, (utilizar bloque que hace parpadear LEDs) tomando  $N = 26$ , y  $N = 27$

Poner un LED de control en cada entrada de selección para conocer el estado.

En archivo/exportar/bitstream, exportar a archivo con extensión .bin

Cargar en la placa y comprobar

Comentar resultados

## PRACTICA 4. DISPLAY DE 7 SEGMENTOS

### *Introducción*

Los displays de 7 segmentos están compuestos por 8 LEDs independientes, 7 de ellos tienen una forma alargada para representar las líneas de los dígitos, y uno para representar el punto

Para dibujar los dígitos es necesario nombrar los diferentes segmentos, la nomenclatura estándar es usar las letras a, b, c, d, e, f, y g para los segmentos y p para el punto

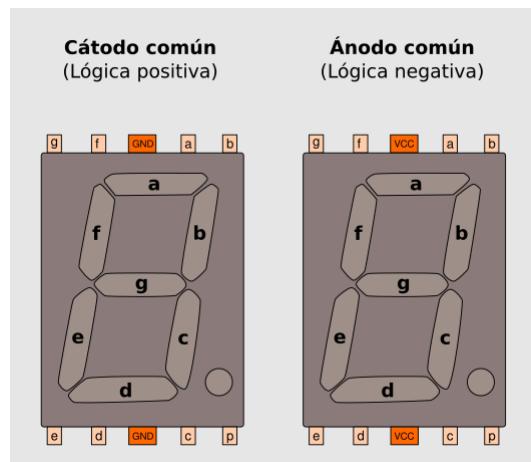


Figura 7: Esquema display de 7 segmentos.

Fuente: FPGAwars

Los displays de 7 segmentos vienen en un encapsulado con 10 patas, 5 en la parte superior y 5 en la inferior. Con lógica positiva: si se envía un 1 el segmento se enciende, y si se envía un 0 se apaga. Los displays que funcionan de esta manera se denominan de cátodo común. Pero también existen los de lógica negativa: con un 0 se encienden y con un 1 se apagan. Se denominan de ánodo común.

Los patillajes son iguales para ambos tipos de displays, con la diferencia de que en unos las patas del medio se conectan a GND (cátodo común) y en otros a VCC (ánodo común)

En el laboratorio remoto se ha conectado un display de 7 segmentos de cátodo común (lógica positiva) a la icezum Alhambra a través de los pines de 3.3v: GP7 - GP0. Utilizando una resistencia de 100ohm. El segmento a está conectado al bit de mayor peso (GP7) y el segmento g al de menor (GP0)

Como cada segmento es un LED independiente, el esquema de conexión es el mismo que se usa para conectar LEDs externos a los pines de 3.3v de la Icezum Alhambra:

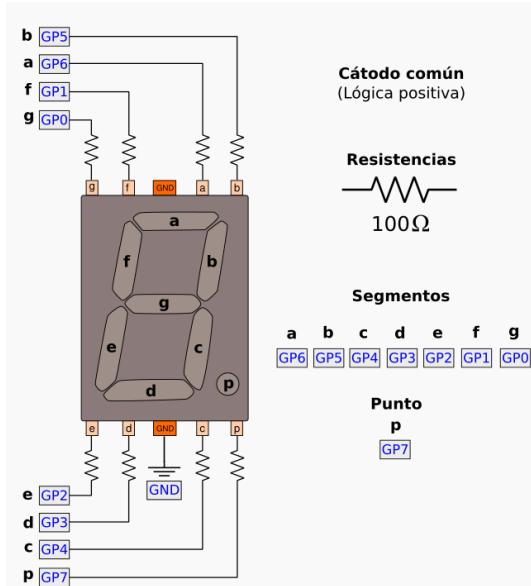


Figura 8: Esquema de conexión del display de 7 segmentos a los pines de FPGA  
Fuente: FPGAwars

**Se pide:**

**Utilizando el display de siete segmentos pon un “0”**

**En archivo/exportar/bitstream, exportar a archivo con extensión .bin**

**Cargar en la placa y comprobar**

**Realizando un diseño similar al de la práctica 2.4, se pide:**

**Utilizando el display de siete segmentos, diseñar un circuito que muestre un “0” cuando el sensor infrarrojo no detecta nada y un “1” cuando el sensor infrarrojo detecta algo.**

**Cargar en la placa y comprobar**

**Comentar resultados**

## PARTE 2. INTRODUCCIÓN AL LENGUAJE HDL – VERILOG

### II.1 LENGUAJE DE DESCRIPCIÓN HARDWARE – VERILOG

**Verilog HDL** Es utilizado para diseñar sistemas electrónicos, verilog soporta el diseño, prueba e implementación de circuitos analógicos, digitales y de señales mixta a diferentes niveles de complejidad.

Posee una sintaxis similar a la del lenguaje de programación C. El lenguaje tiene un preprocesador como C, y la mayoría de palabras reservadas de control como while, if entre otras son similares.

Verilog es uno de los HDL más utilizados y permite descripciones abstractas y representaciones en bajo nivel es decir puede describir sistemas digitales en base a compuertas, e incluso en base a transistores.

Permite que en un diseño se puedan usar diferentes niveles de descripción de sistemas digitales en un mismo ambiente, las diferentes descripciones pueden ser simuladas para verificar el funcionamiento y además pueden ser sintetizadas es decir traducidas a la interconexión de componentes básicas de un dispositivo programable.

Además permite la descripción estructural del diseño en base a componentes básicas, y descripciones más abstractas que se enfocan en la conducta del sistema. La conducta puede describirse mediante expresiones lógicas y también empleando procedimientos.

Un diseño basado en descripciones funcionales o de comportamiento puede resultar lento y de gran tamaño. Las descripciones en niveles estructurales permiten un ahorro de los circuitos lógicos para maximizar la velocidad, minimizar el tamaño y más bajo costo.

**En el MANUAL VERILOG el alumno podrá encontrar una guía de aprendizaje con ejemplos para el diseño HDL usando Verilog.**

### II.2 HERRAMIENTAS DE SIMULACIÓN

Los lenguajes HDL como Verilog son utilizados generalmente para programar FPGA (Field Programmable Gate Array). Estos dispositivos programables contienen bloques de lógica cuya interconexión y funcionalidad puede ser configurada mediante uno de estos lenguajes de descripción especializados (HDL). Se trata precisamente de programar hardware.

Cuando se trabaja con FPGAs se está realizando hardware y hay que tener cuidado. Se podría escribir un código que por ejemplo tuviera un cortocircuito. Y podría ocurrir que las herramientas de síntesis no avisen con un warning. Y al ser cargado en la FPGA, esta se podría estropear parcialmente.

Por ello, es preciso simular el código que se diseñe. Y una vez se comprueba que funciona correctamente es cuando se carga en la FPGA.

**Icarus Verilog** es un compilador de lenguaje Verilog desarrollado para GNU/Linux, pero que puede funcionar correctamente en Windows gracias a la pila MinGW (Minimal GNU for Windows). MinGW es el mismo compilador que utiliza, por ejemplo, el IDE para desarrollo en lenguaje C

Code::Blocks. Ya sea en sus versiones para Linux como en sus versiones para Windows, Icarus Verilog es liberado bajo la licencia GNU GPL.

Es un compilador liviano e **incluye el simulador GTKWave** para verificar el funcionamiento de un diseño.

En el **MANUAL DE USO ICARUS VERILOG + GTKWave** se explica el proceso de instalación y configuración de Icarus Verilog en Windows, así como el uso básico del mismo a través de un programa (diseño) de prueba.

## II.3 HERRAMIENTAS DE COMPILACIÓN

Como herramienta de compilación se utilizará la web del laboratorio remoto IceZUM Alhambra, eligiendo la opción de compilación remota, en la que el alumno se tendrá que enviar los archivos con extensiones .v (archivo con código verilog) y prf (asociación de etiquetas con los pines de la FPGA).

En la siguiente figura se muestran las diferentes herramientas usadas en las etapas, y las extensiones de los archivos que se van generando:

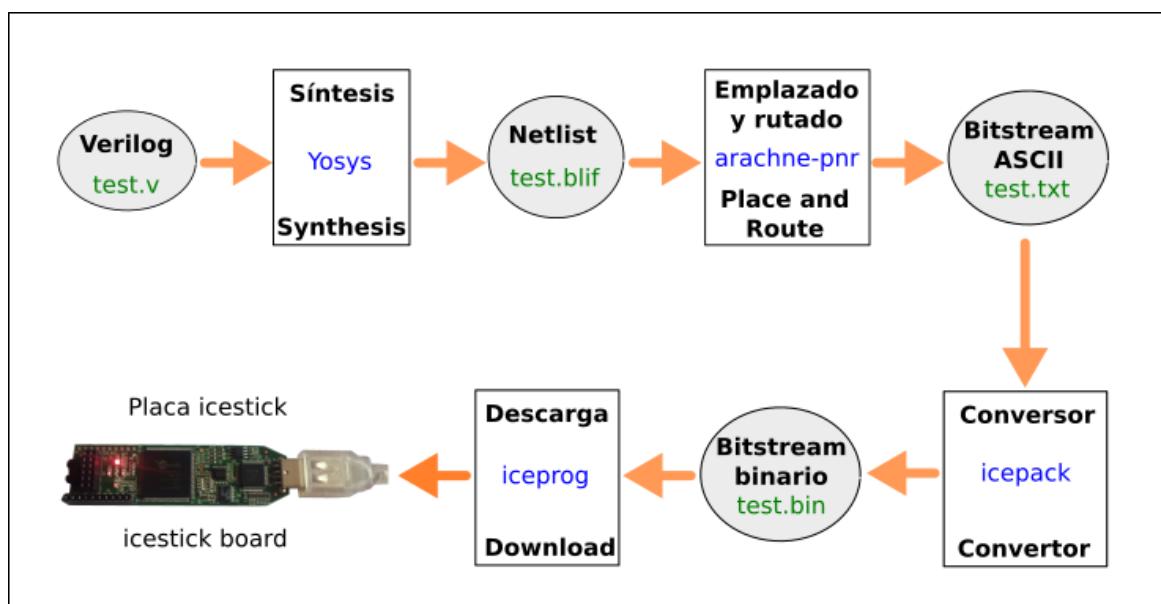


Figura 9: Herramientas usadas para crear el bitstream  
Fuente: <https://github.com/Obijuan/open-fpga-verilog-tutorial>

Se parte de los ficheros fuente en **verilog (.v)**. Usando el sintetizador **Yosys**, se generan los ficheros **netlist (.blif)**. El emplazado y rutado se realiza con **arachne-pnr**, generándose el bitstream en formato **ASCII (.txt)**. Con **icepack** se crea el bitstream **binario (.bin)** que finalmente se envía a la FPGA con **iceprog**.

En la línea de comandos, los pasos a seguir para llevar el fichero test.v hasta la FPGA serían:

```

yosys -p "synth_ice40 -blif test.blif" test.v
arachne-pnr -d 1k -p test.pcf test.blif -o test.txt
icepack test.txt test.bin
iceprog test.bin
  
```

## II.4 FLUJO DE TRABAJO

---

- ❖ **Diseño:** Lápiz y papel o la pizarra
- ❖ **Escritura de HDL:** Por el momento las herramientas soportan únicamente el lenguaje Verilog. Para generar los archivos se pueden utilizar cualquier editor que soporte Verilog
- ❖ **Simulación:** Puede realizarse utilizando Icarus Verilog + GTKWave. Icarus verilog crea un fichero ejecutable a partir del código Verilog. Al ejecutarlo se realiza la simulación. Los resultados se vuelcan a un fichero .vcd que se visualiza con la herramienta Gt\_kwave. Esto nos permite inspeccionar las señales para comprobar su correcto funcionamiento
- ❖ **Síntesis:** Se realiza utilizando Yosys.
- ❖ **Ruteado:** Se realiza utilizando Arachne-pnr. Para hacer el mapeo es necesario un fichero con extensión .pcf, en el cual se asociarán las etiquetas utilizadas por el usuario con los pines de la FPGA.
- ❖ **Generación de Bitstream:** IceStorm, que también provee de algunas utilidades para guardar el bitstream directamente en la FPGA a través de un convertidor USB-Serie.

En el **MANUAL DE USUARIO** del Laboratorio Remoto IceZUM Alhambra se explica el proceso a seguir para realizar la síntesis y programación de la FPGA.

## II.5 PRACTICAS PROPUESTAS

---

A continuación se explican las prácticas propuestas.

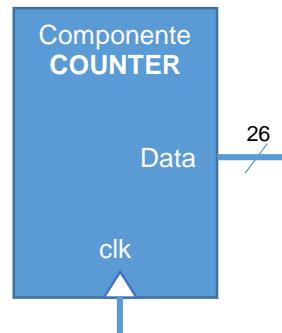
## PRACTICA 5. CONTADOR

### Introducción

Esta práctica consiste en diseñar un contador conectado a los LEDs internos de la placa IceZUM Alhambra.

Un contador es un circuito secuencial que puede almacenar información.

Almacenará un número que se incrementará con cada tic del reloj. Como la señal del reloj de la placa IceZUM Alhambra es de 12Mhz, se deberá utilizar un contador de 26 bits, **se usará los 4 bits más significativos para mostrarlos en los LEDs** de la placa.



### Descripción Hardware

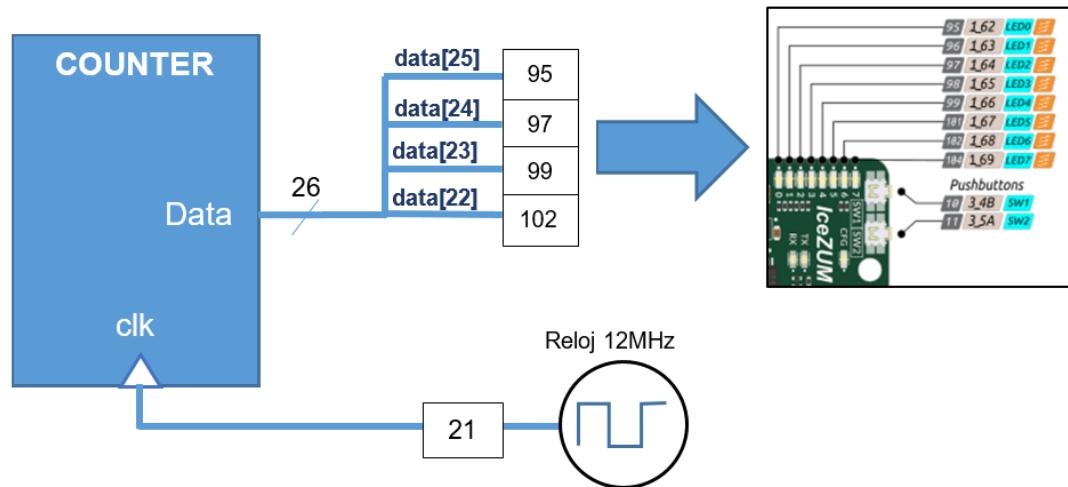
El contador tendrá una **entrada clk**, que es un cable, y una salida data de 26 bits que nos devolverá el valor del contador. La salida es un registro de 26 bits, que almacena el valor de la cuenta.

El funcionamiento del componente se describe dentro del bloque **always @(posedge clk)**, que indica que todo lo que está dentro de ese bloque sólo se evaluará cada vez que llegue un flanco de subida en la **señal clk**. Cada vez que llega uno, se incrementa en una unidad el registro data (y saldrá por la salida del componente).

Cualquier contador, con independencia de su número de bits, se construye de esta manera. Se puede cambiar el número de bits, cambiando el valor del **registro data** (p.e. de 25 a 19 para que tenga 20 bits)

### Síntesis en la FPGA

Los 4 bits de mayor peso (data[25], data[24], data[23] y data[22]) los conectaremos a los pines de la fpga donde están los leds. El reloj entra por el pin 21, que lo conectaremos a la señal de reloj clk de nuestro contador



*Figura 10: Esquema COUNTER – pines de conexión.*

La asociación entre pines de nuestro componente y pines de la fpga se establece en el fichero **contador.pcf**.

```
set_io data[25] 96  
set_io data[24] 97  
set_io data[23] 98  
set_io data[22] 99  
set io clk 21
```

## Realizar la compilación remota en el laboratorio remoto y programar la FPGA IceZUM Alhambra

**Cambiar el contador de 26 a 20 bits, y comentar los resultados**

## PRACTICA 6. PRESCALER

### Introducción

Los prescalers sirven para ralentizar las señales de reloj. Por la entrada entra una señal de reloj de frecuencia  $f$  y por la salida se obtiene una de frecuencia menor. En esta práctica se diseñará un prescaler de  $N$  bits para hacer parpadear un led a diferentes frecuencias

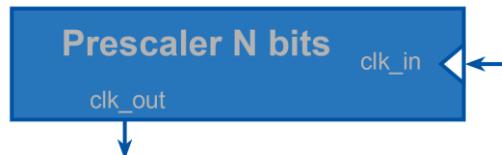


Figura 11: Prescaler.

Para un prescaler de  $N$  bits, las fórmulas que relacionan las frecuencias y períodos de entrada con los de salida son:

$$\text{Frecuencia entrada} = f_{in} \quad \text{Frecuencia salida} = f_{out} = \frac{f_{in}}{2^N}$$

$$\text{Período entrada} = T_{in} \quad \text{Período salida} = T_{out} = 2^N \cdot T_{in}$$

### Frecuencias y períodos del prescaler

La frecuencia de entrada al prescaler en la placa IceZUM Alhambra es de 12Mhz. Aplicando la fórmula anterior:

**Construir una tabla con períodos y frecuencias para prescalers con diferente número de bits (N).**

| Bits (N) | Frecuencia | Período    |
|----------|------------|------------|
| 1        | 6 MHz      | 0.167 usec |
| ....     | ....       | ....       |
| ....     | ....       | ....       |
|          |            |            |

Teniendo en cuenta que el ojo humano tiene una frecuencia de refresco de unos 25Hz. Esto significa que frecuencias superiores no se aprecian. Si se hace parpadear el led con una frecuencia superior, se aprecia como si siempre estuviese encendido (no se ve parpadear)

**Determinar el número de bits que deberá tener el prescaler para poder apreciar el parpadeo de lo LED.**

### Descripción del hardware

El código de un prescaler es similar al de un contador, sin embargo hay introducir la novedad de que el prescaler es paramétrico, de esta forma el número de bits estará determinado por un parámetro  $N$ , que permitirá sintetizar prescalers de diferentes tamaños cambiando el valor del parámetro  $N$

**Definir un registro de N bits que incremente en cada flanco de subida de la señal del reloj de entrada. El bit más significativo se deberá conectar directamente a la salida clk\_out**

### Síntesis en la FPGA

La señal de la placa IceZUM Alhambra de 12 MHZ se introduce a clk\_in a través del pin 21 de la FPGA. La salida clk\_out se envía directamente al led D1 (pin 99), para que parpadee a su misma frecuencia.

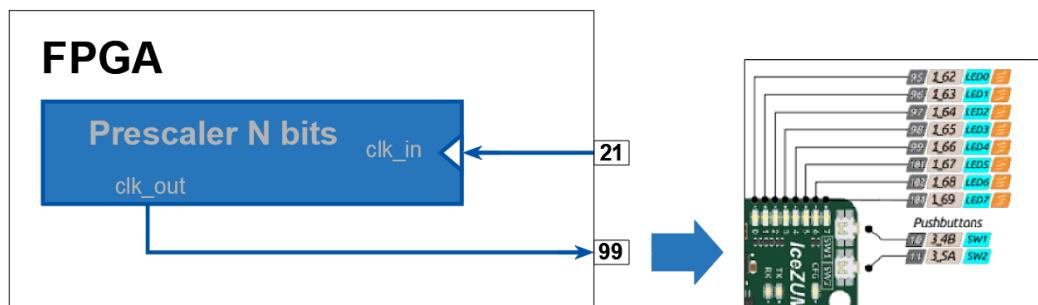


Figura 12: Esquema PRESCALER – pines de conexión.

La asociación entre pines de nuestro componente y pines de la fpga se establece en el fichero **prescaler.pcf**:

```
set_io clk_out 99
set_io clk_in 21
```

**Realizar la compilación remota en el laboratorio remoto y programar la FPGA IceZUM Alhambra**

Si todo ha ido bien el led 4 empezará a parpadear

**Modificar el prescaler para valores de N = 18, 19, 20 y 21.**

**Cambiar led 4 a otro led**

**Sintetizar y descargar en la FPGA IceZUM Alhambra.**

**Comentar resultados.**

## PRACTICA 7. REGISTRO DE 4 BITS

### Introducción

Los registros permiten almacenar información, desde 1 hasta N bits. Se utilizan para implementar procesadores, realizar segmentación, almacenamiento de resultados intermedios, etc.

Un registro básico captura los datos de la entrada, en flanco de subida o bajada del reloj, los almacena y los saca por la salida. Su esquema es:

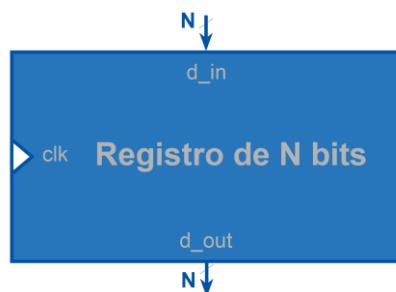


Figura 13: Registro de N bits

Un registro de 4 bits inicialmente está inicializado a 0. Por la salida dout salen 4 bits a 0, que se convierten en 4 bits a 1 al pasar por el inversor. En el siguiente flanco de subida del reloj, se captura este nuevo valor 4'b1111 en el registro. Al salir por dout, se vuelve a invertir obteniéndose 4 bits a 0, como al principio.

El resultado es la siguiente secuencia 0000, 1111, 0000, 1111 ... que cambia con cada flanco de subida del reloj.

Si la salida **dout** se conecta a los leds, estos se encenderán y apagarán con la frecuencia del reloj.

### Descripción del hardware

La implementación de un registro es verilog es extremadamente sencilla. Basta con este código:

```
always @(posedge(clk_base))
dout <= din;
```

**Utilizar un registro de 4 bits para hacer parpadear 4 leds de la placa IceZUM Alhambra**

**Para poder apreciar el parpadeo, incluye un prescaler de 22 bits**

En la figura siguiente se muestra el diseño a implementar en la FPGA:

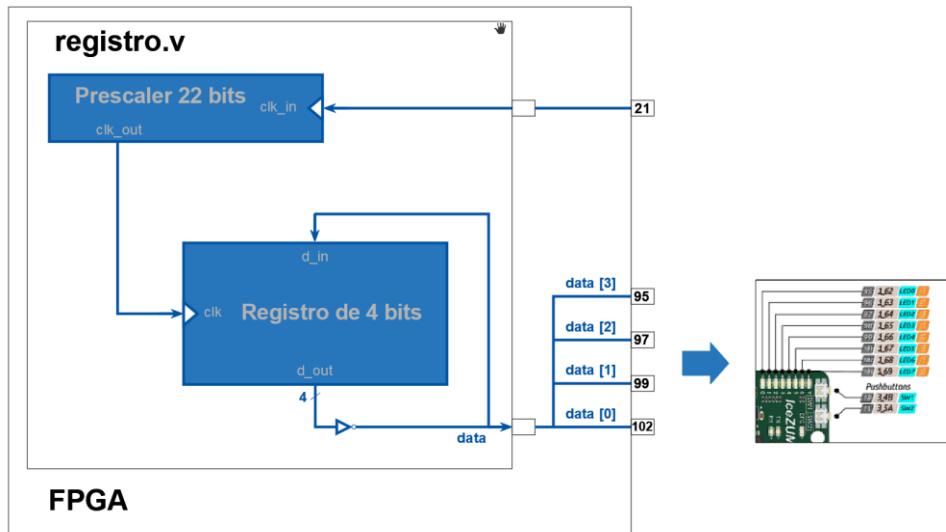


Figura 14: Esquema REGISTRO DE 4 BITS – pines de conexión.

### Síntesis en la FPGA

La asociación entre pines de nuestro componente y pines de la fpga se establece en el fichero **registro.pcf**:

```
set_io data[3] 99
set_io data[2] 98
set_io data[1] 97
set_io data[0] 96
set_io clk 21
```

### Realizar la compilación remota en el laboratorio remoto y programar la FPGA IceZUM Alhambra

Si todo ha ido bien los leds 0, 2, 4 y 6 comenzarán a parpadear

**Cambiar frecuencia de parpadeo de los leds**

**Sintetizar y descargar en la FPGA IceZUM Alhambra.**

**Comentar resultados.**

## PRACTICA 8. REGISTRO DE DESPLAZAMIENTO

### Introducción

Los registros de desplazamiento permiten almacenar un valor y desplazarlo. Se utilizan para convertir la información de paralelo a serie (y vice-versa) para usar en las comunicaciones síncronas. Las comunicaciones a través de SPI, I2C, etc, se implementan con estos registros. También permiten realizar las operaciones de multiplicar por 2 y dividir entre 2 para número enteros.

El registro de desplazamiento utilizado será como el siguiente:

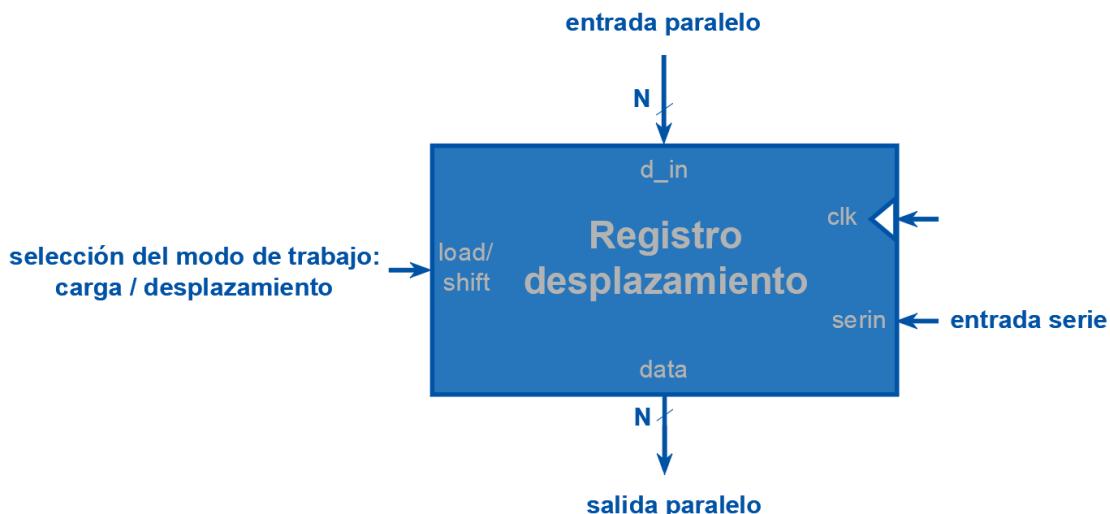


Figura 15: Esquema REGISTRO DE DESPLAZAMIENTO.

La salida del registro es de N bits. Tiene una entrada en paralelo de N bits, que permite cargar en el registro con un valor nuevo. La señal **load\_shift** permite determinar el modo de funcionamiento: cuando está a 0 se realiza la carga de un valor nuevo al llegar un flanco de subida de reloj. Cuando está a 1 se realiza un desplazamiento hacia la izquierda en el flanco de subida del reloj.

En este desplazamiento el bit más significativo se pierde, y el nuevo se lee de la entrada **serin** (serial input). Si el valor inicial almacenado es 1001, la señal **load\_shift** está a 1, **serin** está a 0 y llega un flanco de subida de reloj, obtendremos el valor: 0010. En el siguiente flanco (si **serin** sigue a 0) obtendremos 0100, luego 1000 y luego 0000.

Un registro de desplazamiento de 4 bits puede rotar una secuencia de bits y mostrarlos por los 4 leds de la placa IceZUM Alhambra. La secuencia obtenida por los leds dependerá del valor inicial cargado en el registro.

El diagrama de bloques del componente desplazamiento es:

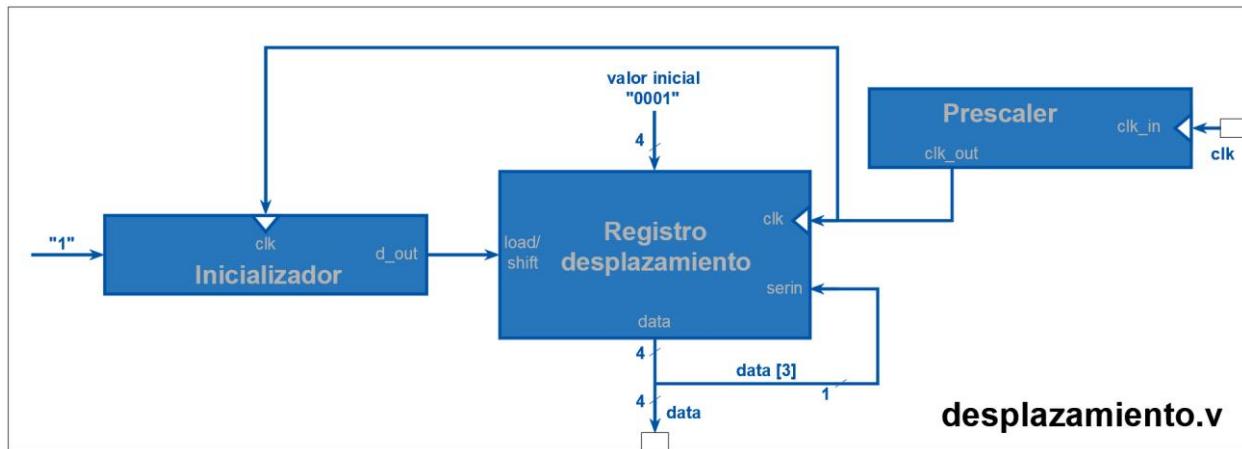


Figura 16: Diagrama de bloques componente desplazamiento.v

El componente principal es un registro de desplazamiento de 4 bits. Por su entrada de reloj se conecta el reloj de la placa IceZUM Alhambra a través de un prescaler, para disminuir su frecuencia y poder ver la rotación de los bits en los leds.

El bit más significativo del registro (data [3]) se conecta directamente a la entrada serin, de forma que se consiga la rotación de bits (el más significativo pasa a ser el de menor peso)

Por la entrada paralelo se introduce el valor inicial, que por defecto será el 0001. Al rotar aparecerá la secuencia 0010, 0100, 1000 y 0001.

La carga inicial se realiza usando un inicializador, que presentaría el siguiente código:

```
//-- init.v
module init(input wire clk, output ini);
reg ini = 0;
always @ (posedge(clk)) ini <= 1;
endmodule
```

El inicializador está conectado a la entrada load\_shift, de manera que inicialmente está a 0 y al llegar el primer flanco de reloj pasa a '1', cargándose el valor inicial y pasando a modo desplazamiento. En el resto de flancos se realizará el desplazamiento (y no la carga).

### Descripción del hardware

El registro de desplazamiento se describe con muy pocas líneas. Es un proceso que depende del flanco de subida del reloj.

```
always @ (posedge(clk_pres)) begin
if (load_shift == 0) //-- Load mode
data <= INI;
else
data <= {data[2:0], serin};
end
```

En el modo carga se saca el valor inicial (INI) por la salida. En el de desplazamiento se sacan los tres bits menos significativos y se añade **serin** como menos significativo. Esto se hace con

el operador de concatenación {}: A los tres cables definidos por data[2:0] se le añade un cuarto cable: **serin**

El componente final **desplazamiento** tendrá **2 parámetros**: el **número de bits del prescaler** (NP), que determinará la velocidad de rotación de los bits y el **valor inicial** (INI) a cargar, que determinará la secuencia. Por defecto, este valor inicial será 0001.

El registro de desplazamiento se puede implementar directamente como un proceso en el propio componente, en vez de hacer como un componente separado (diseño jerárquico).

**Utilizar el registro de desplazamiento para generar una secuencia de 4 estados en los leds de la placa IceZUM Alhambra**

### Síntesis en la FPGA

Para sintetizarlo en la fpga se conectan las salidas data a los leds, y la entrada de reloj a la de la placa IceZUM Alhambra.

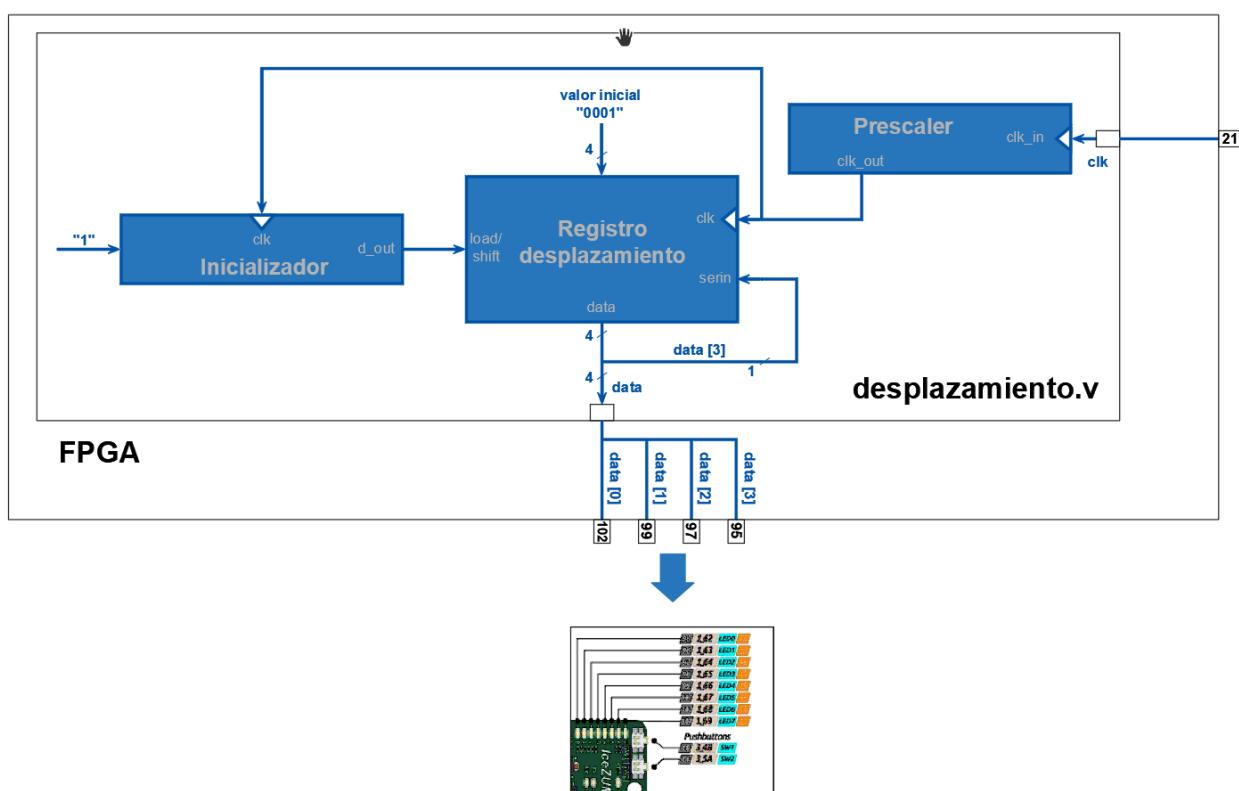


Figura 17: Registro de desplazamiento – pines de conexión.

La asociación entre pines de nuestro componente y pines de la fpga se establece en el fichero **desplazamiento.pcf**:

```
set_io data[3] 96
set_io data[2] 97
set_io data[1] 98
set_io data[0] 99
set_io clk 21
```

**Realizar la compilación remota en el laboratorio remoto y programar la FPGA IceZUM Alhambra**

Si todo ha ido bien los leds 0, 2, 4 y 6 comenzarán a realizar la secuencia de 4 estados

**Cambiar el valor del prescaler para que la rotación sea más rápida y el valor inicial del registro, para que salga otra secuencia.**

**Sintetizar y descargar en la FPGA IceZUM Alhambra.**

**Comentar resultados.**

## 4. BIBLIOGRAFÍA

<https://github.com/Obijuan/open-fpga-verilog-tutorial/wiki>

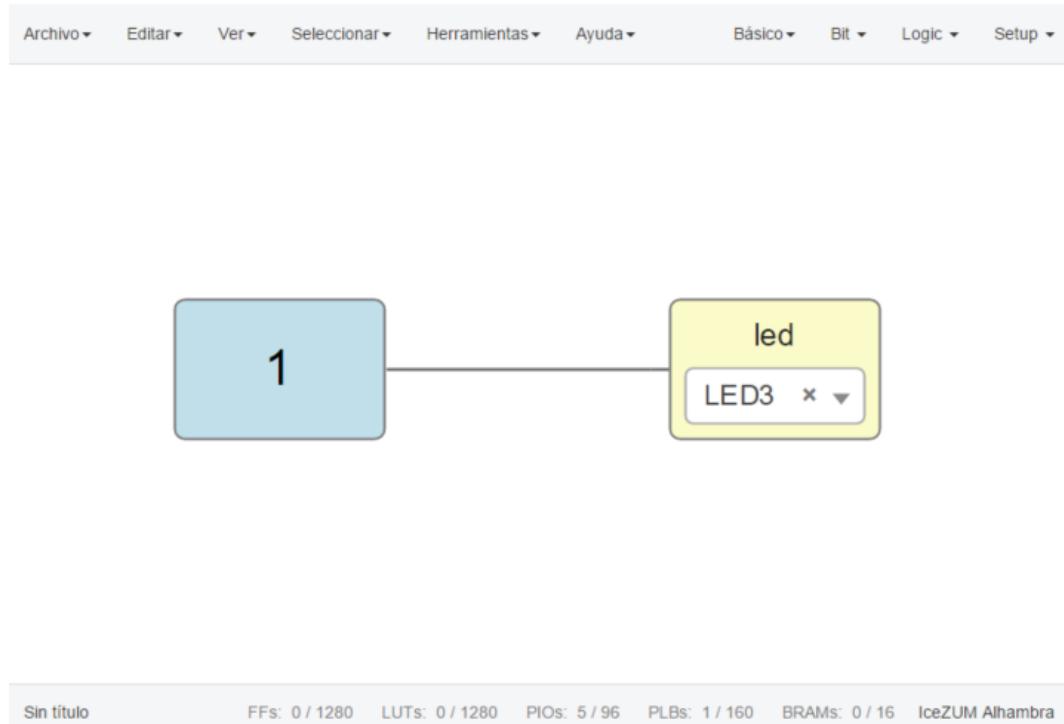
<https://www.linuxito.com/windows/1088-primeros-pasos-con-icarus-verilog-en-windows>

Guía de Instalación y Uso de Icarus Verilog y GtkWave Profesor: Claudio Estévez Auxiliar: Luis Alberto Herrera April 15, 2011

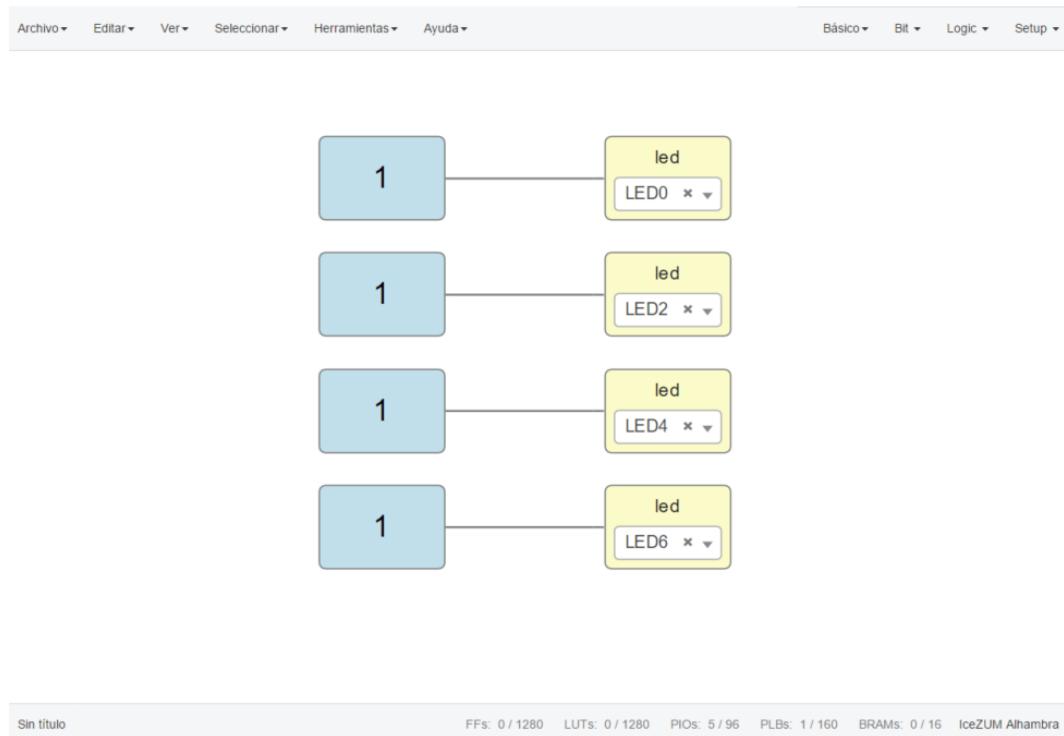
# SOLUCIONES PRÁCTICAS

## SOLUCIÓN PRÁCTICA 1 – ENCENDIENDO LEDS

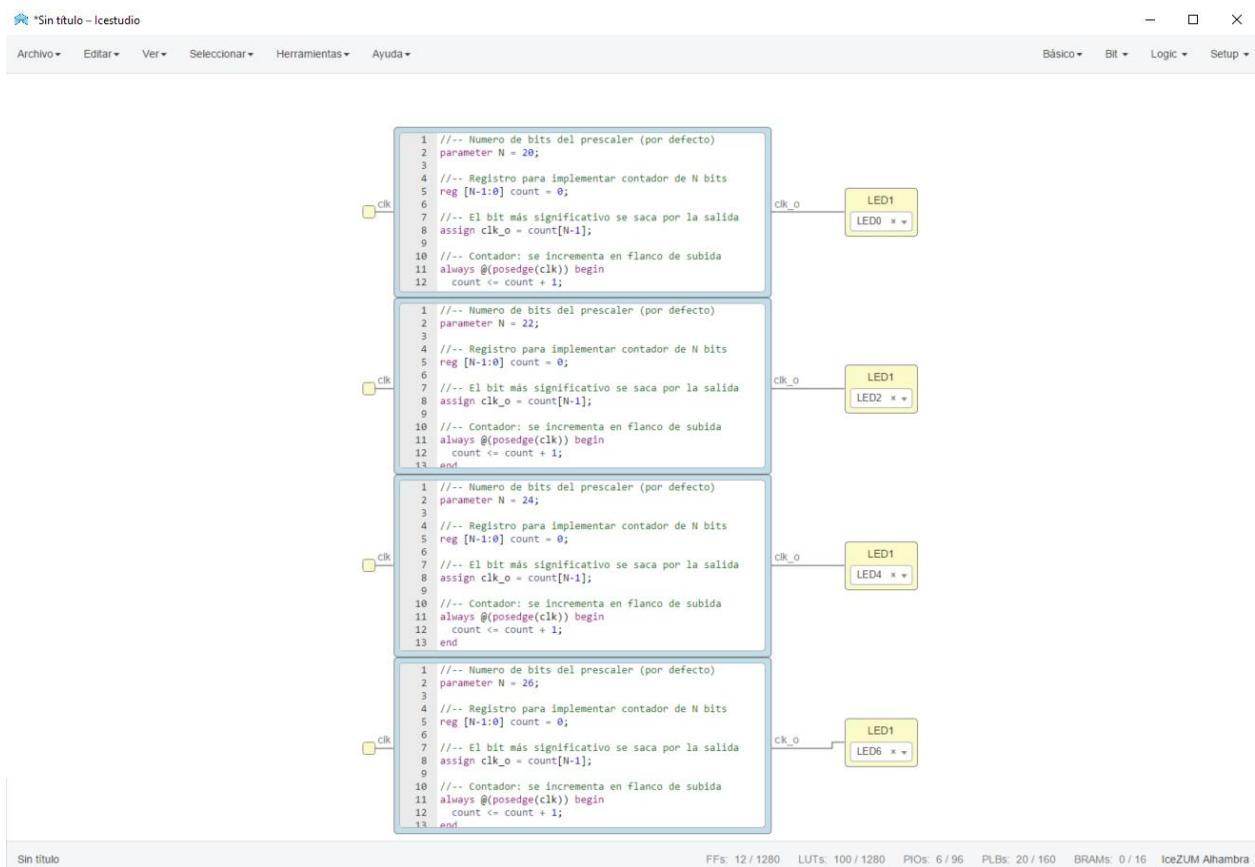
### *Bits constantes*



### *Superposición de circuitos*

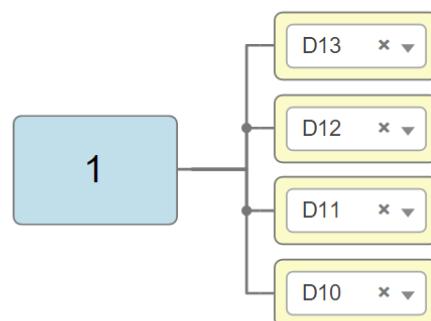


## **Leds parpadeantes**



## SOLUCIÓN PRÁCTICA 2 – PERIFÉRICOS

## **LEDs Externos 1**



Practica 02.1

EEs - / 1280

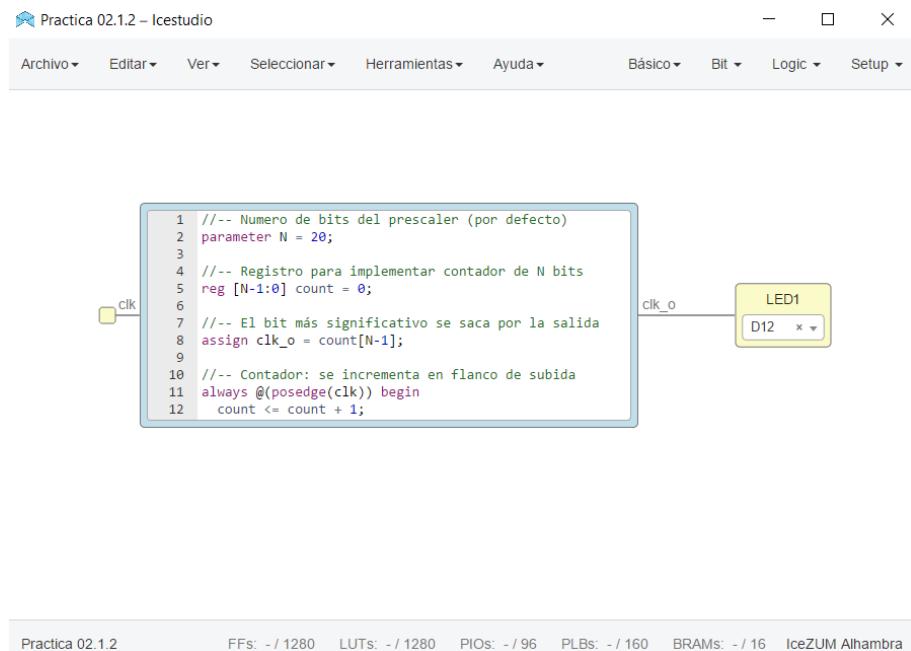
PIOS: - / 96

PI Rs. - / 160

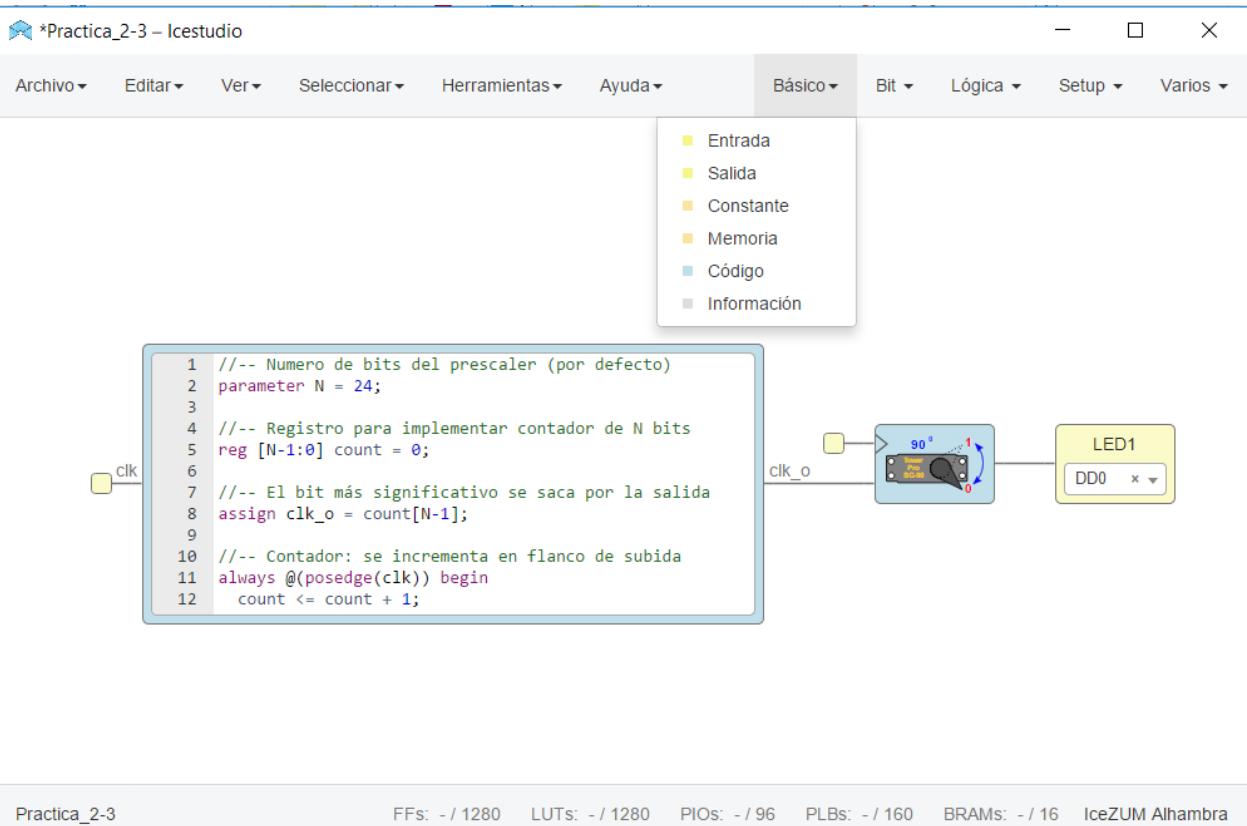
BRAMs

6 IceZI UM Alhambra

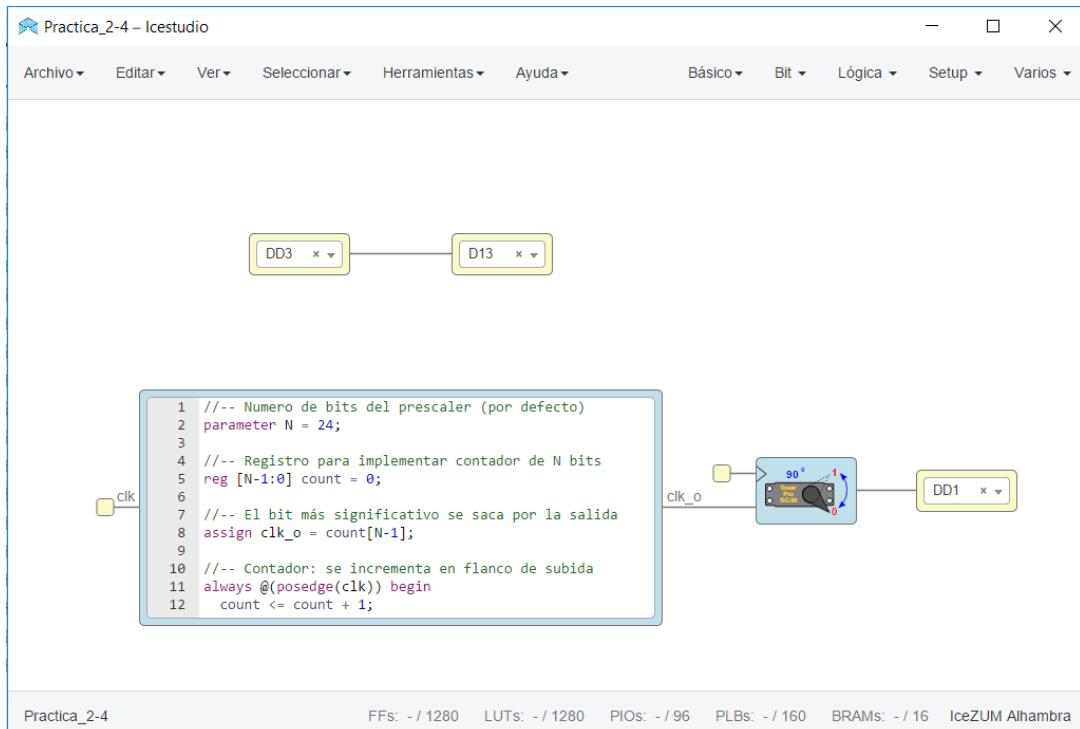
## LEDs Externos 2



## Controlando Servos

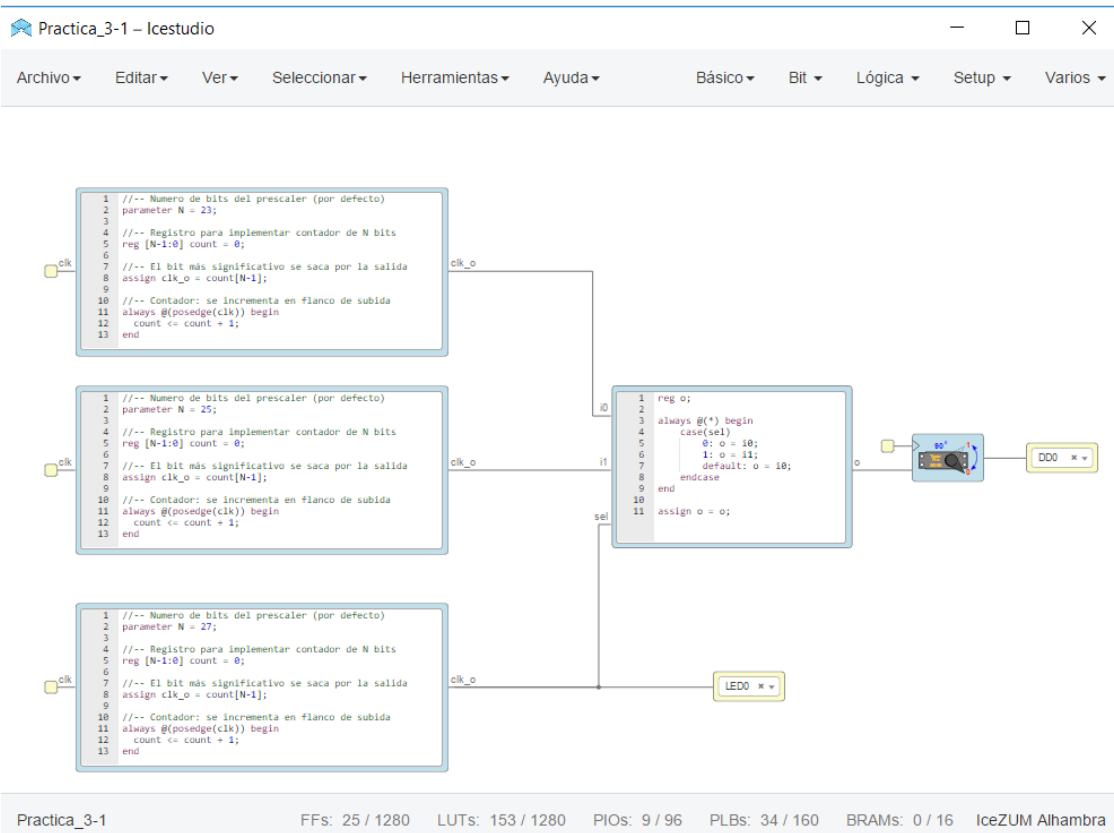


## Infrarrojos

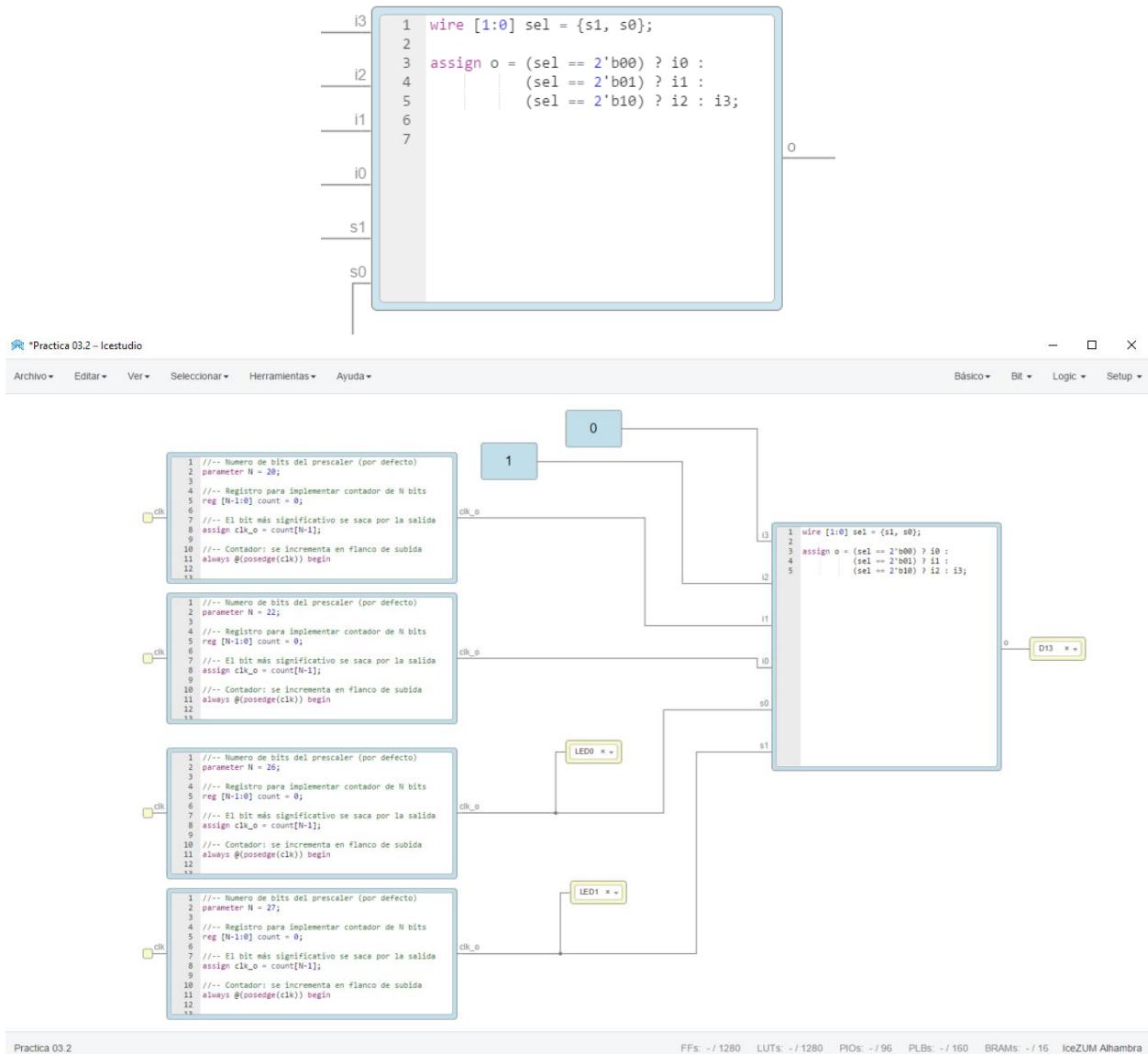


## SOLUCIÓN PRÁCTICA 3 – MULTIPLEXACIÓN

### Multiplexor 2 a 1

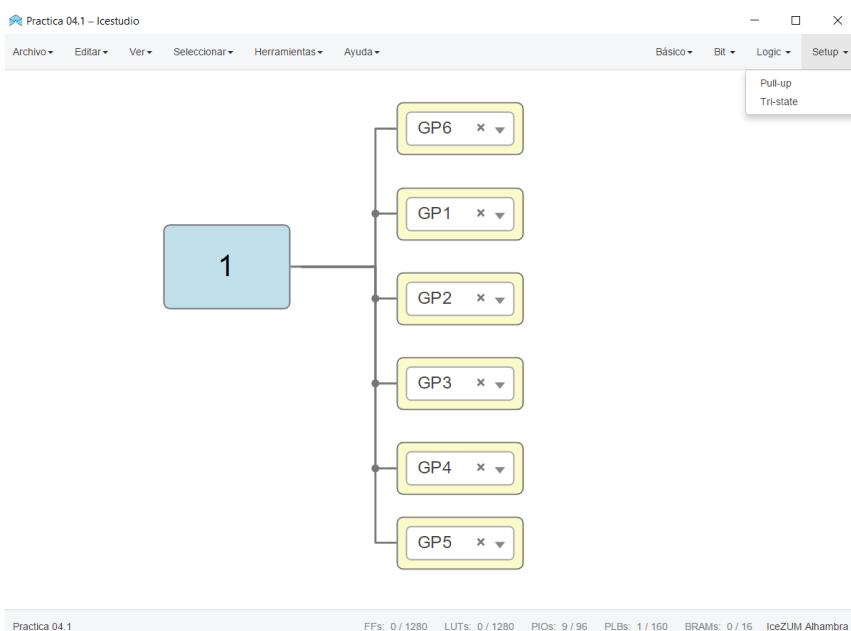


## Multiplexor 4 a 1



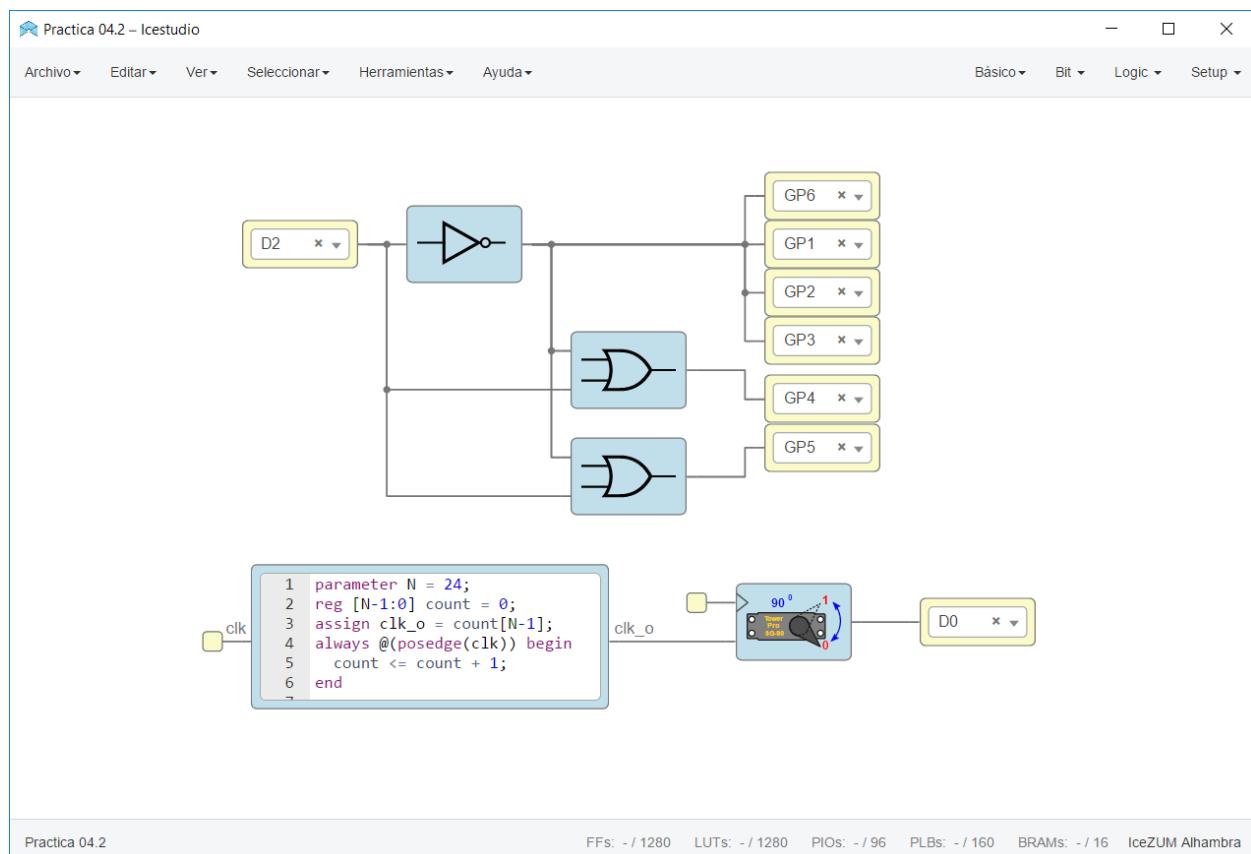
## SOLUCIÓN PRÁCTICA 4 – DISPLAY DE 7 SEGMENTOS

### Número



Practica 04.1 FFs: 0 / 1280 LUTs: 0 / 1280 PIOs: 9 / 96 PLBs: 1 / 160 BRAMs: 0 / 16 IceZUM Alhambra

### Decodificador



Practica 04.2 FFs: - / 1280 LUTs: - / 1280 PIOs: - / 96 PLBs: - / 160 BRAMs: - / 16 IceZUM Alhambra

## SOLUCIÓN PRÁCTICA 5 – CONTADOR

```
//-----
//-- Entrada: señal de reloj
//-- Salida: contador de 26 bits
//-----
module contador(input clk, output [25:0] data);
wire clk;

//-- La salida es un registro de 26 bits, inicializado a 0
reg [25:0] data = 0;

//-- Sensible al flanco de subida
always @(posedge clk) begin
//-- Incrementar el registro
data <= data + 1;
end
endmodule

set_io data[25] 96
set_io data[24] 97
set_io data[23] 98
set_io data[22] 99
set_io clk 21
```

## SOLUCIÓN PRÁCTICA 6 – PRESCALER

```
 //-- prescaler.v
//-- clk_in: señal de reloj de entrada
//-- clk_out: Señal de reloj de salida, con menor frecuencia
module prescaler(input clk_in, output clk_out);
wire clk_in;
wire clk_out;

//-- Numero de bits del prescaler (por defecto)
parameter N = 22;

//-- Registro para implementar contador de N bits
reg [N-1:0] count = 0;

//-- El bit más significativo se saca por la salida
assign clk_out = count[N-1];

//-- Contador: se incrementa en flanco de subida
always @(posedge(clk_in)) begin
count <= count + 1;
end
endmodule

set_io clk_out 99
set_io clk_in 21
```

## SOLUCIÓN PRÁCTICA 7 – REGISTRO DE 4 BITS

```
//-- registro4bit.v
module registro4b(input wire clk, output wire [3:0] data);

//-- Bits para el prescaler
parameter N = 22;

//-- Reloj principal (prescalado)
wire clk_base;

//-- Datos del registro
reg [3:0] dout = 0;

//-- Cable de entrada al registro
wire [3:0] din;

//-- Instanciar el prescaler
prescaler #(.N(N))
PRES (
.clk_in(clk),
.clk_out(clk_base)
);

//-- Registro
always @(posedge(clk_base))
dout <= din;
//-- Puerta NOT entra la salida y la entrada
assign din = ~dout;

//-- Sacar datos del registro por la salida
assign data = dout;

endmodule

module prescaler(input clk_in, output clk_out);
wire clk_in;
wire clk_out;

parameter N = 22;

reg [N-1:0] count = 0;
assign clk_out = count[N-1];

always @(posedge(clk_in)) begin
count <= count + 1;
end

endmodule

set_io data[3] 99
set_io data[2] 98
set_io data[1] 97
set_io data[0] 96
set_io clk 21
```

## SOLUCIÓN PRÁCTICA 8 – REGISTRO DESPLAZAMIENTO

```
module shift4(input wire clk, output reg [3:0] data);

//-- Parametros del secuenciador
parameter NP = 21; //-- Bits del prescaler
parameterINI = 1; //-- Valor inicial a cargar en el registro

//-- Reloj de salida del prescaler
wire clk_pres;

//-- Shift / load. Señal que indica si el registro //-- se carga o desplaza
//-- shift = 0: carga //-- shift = 1: desplaza
reg load_shift = 0;

//-- Entrada serie del registro
wire serin;

//-- Instanciar el prescaler de N bits
prescaler #(.N(NP))
pres1 (
    .clk_in(clk),
    .clk_out(clk_pres)
);

//-- Inicializador
always @(posedge(clk_pres)) begin
    load_shift <= 1;
end

//-- Registro de desplazamiento
always @(posedge(clk_pres)) begin
    if (load_shift == 0) //-- Load mode
        data <=INI;
    else
        data <= {data[2:0], serin};
end

//-- Salida de mayor pero se re-introduce por la entrada serie
assign serin = data[3];

endmodule

module prescaler(input clk_in, output clk_out);
wire clk_in;
wire clk_out;

parameter N = 22;

reg [N-1:0] count = 0;

assign clk_out = count[N-1];

always @(posedge(clk_in)) begin
    count <= count + 1;
end

endmodule
```

```
set_io data[3] 96
set_io data[2] 97
set_io data[1] 98
set_io data[0] 99
set_io clk 21
```

---

## A05-MANUAL DE USO ICESTUDIO

---

## ÍNDICE DE CONTENIDOS

---

|  |    |
|--|----|
| 1. INTRODUCCIÓN .....                      | 3  |
| 2. INSTALACIÓN DEL SOFTWARE .....          | 3  |
| 2.1. DESCARGA DEL INSTALADOR .....         | 3  |
| 2.2. ANTES DE EJECUTAR EL INSTALADOR ..... | 3  |
| 2.3. INSTALAR LA TOOLCHAIN .....           | 5  |
| 2.4. INSTALAR LOS DRIVERS .....            | 6  |
| 2.5. SELECCIÓN DE FPGA .....               | 6  |
| 2.6. COMPROBAR LA INSTALACIÓN .....        | 7  |
| 3. ENTORNO DE TRABAJO .....                | 10 |
| 4. MENÚ .....                              | 11 |
| 4.1. ARCHIVO .....                         | 11 |
| 4.2. EDITAR .....                          | 12 |
| 4.3. VER .....                             | 13 |
| 4.4. SELECCIONAR .....                     | 14 |
| 4.5. HERRAMIENTAS .....                    | 15 |
| 4.6. AYUDA .....                           | 16 |
| 5. MENÚ DE BLOQUES .....                   | 17 |
| 5.1. BÁSICO .....                          | 17 |
| 5.2. BLOQUES .....                         | 18 |
| 6. ÁREA DE DISEÑO .....                    | 18 |
| 6.1. CAMBIAR TAMAÑO DE LOS BLOQUES .....   | 18 |
| 6.2. EXAMEN DE BLOQUE .....                | 18 |
| 6.3. DETECCIÓN DE ERRORES VERILOG .....    | 18 |
| 6.4. DESHACER / REHACER .....              | 19 |
| 6.5. TOMA UNA IMAGEN DE LA PANTALLA .....  | 19 |
| 7. COLECCIONES .....                       | 19 |
| 7.1. LA COLECCIÓN POR DEFECTO .....        | 19 |
| 7.2. AÑADIR COLECCIONES .....              | 20 |
| 7.3. SELECCIÓN DE COLECCIÓN .....          | 20 |
| 7.4. ELIMINAR COLECCIÓN .....              | 21 |
| 8. COMENTARIOS .....                       | 21 |
| 9. BIBLIOGRAFÍA .....                      | 22 |

## ÍNDICE DE ILUSTRACIONES

---

|   |    |
|---|----|
| <i>Figura 2.1: Página de descarga IceStudio Septiembre 2018.</i> .....                                    | 3  |
| <i>Figura 2.2: Pasos para instalar IceStudio en diferentes sistemas operativos. Septiembre 2018.</i> .... | 5  |
| <i>Figura 2.3: Menú herramientas/Toolchain, Icestudio</i> .....   | 5  |
| <i>Figura 2.4:Avisos de Icestudio.</i> .....  | 6  |
| <i>Figura 2.5: Menú herramientas/Drivers, Icestudio.</i> .....  | 6  |
| <i>Figura 2.6:Selección Placa.....</i>  | 7  |
| <i>Figura 2.7:Menú Archivo/Ejemplos/1. Basic, Icestudio</i> .....   | 7  |
| <i>Figura 2.8: Ejemplo 01.One LED, Icestudio</i> .....  | 8  |
| <i>Figura 2.9: Verificación 01.One LED, Icestudio</i> .....   | 8  |
| <i>Figura 2.10: Sintetizado 01.One LED, Icestudio</i> .....   | 9  |
| <i>Figura 2.11: Carga 01.One LED, Icestudio</i> .....   | 9  |
| <i>Figura 2.12: LED 0 encendido, FPGA IceZUM Alhambra.</i> .....  | 9  |
| <i>Figura 3.1: Entorno de trabajo Icestudio.</i> .....  | 11 |
| <i>Figura 4.1: Menú archivo.</i> .....  | 11 |
| <i>Figura 4.2: Menú editar.</i> .....   | 12 |
| <i>Figura 4.3 Menú ver</i> .....  | 13 |
| <i>Figura 4.4: Menú Seleccionar</i> .....   | 14 |
| <i>Figura 4.5: Placas disponibles en versión 0.3.3.</i> .....   | 14 |
| <i>Figura 4.6: Menú Herramientas</i> .....  | 15 |
| <i>Figura 4.7: Menú ayuda</i> .....   | 16 |
| <i>Figura 5.1: Menú Básico</i> .....  | 17 |
| <i>Figura 7.1: Colección por defecto, Icestudio</i> .....   | 20 |
| <i>Figura 8.1: Bloque información, Icestudio</i> .....  | 21 |

## 1. INTRODUCCIÓN

Icestudio es una herramienta para diseño y síntesis de circuitos digitales en FPGAs libres programada en nodejs. Es software libre y multiplataforma. Corre en los sistemas GNU/Linux, Mac OS y Windows.

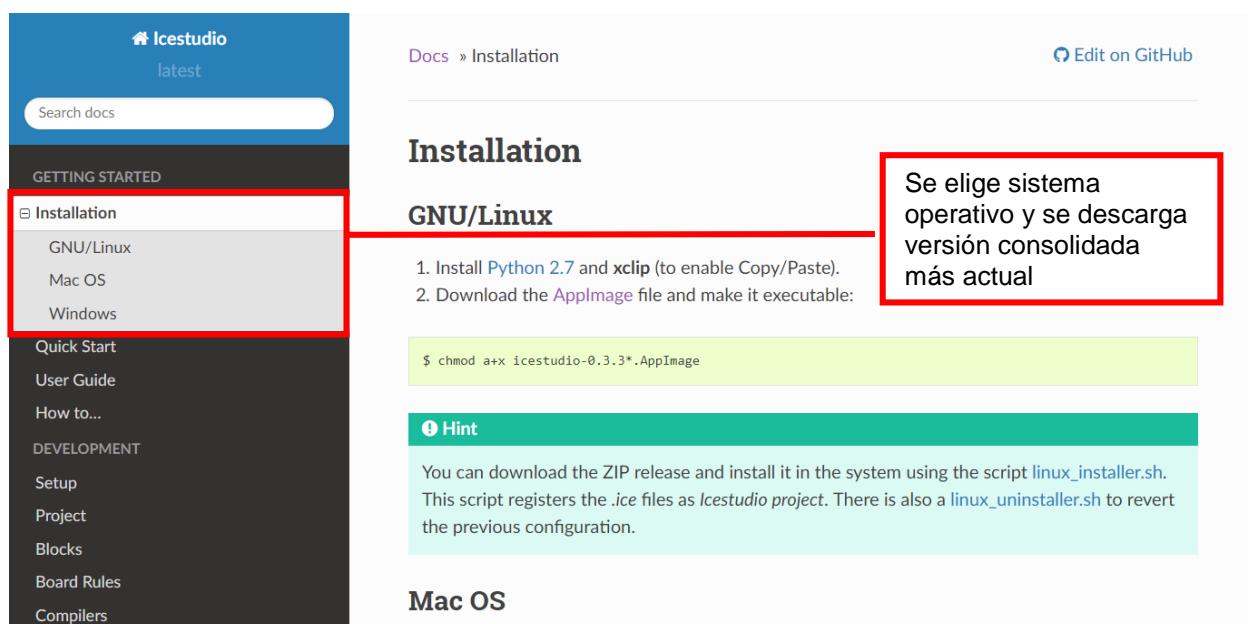
## 2. INSTALACIÓN DEL SOFTWARE

A continuación se explica el proceso a seguir para la instalación del Software.

### 2.1. DESCARGA DEL INSTALADOR

Se descarga del siguiente enlace el instalador de Icestudio.

<https://icestudio.readthedocs.io/en/latest/source/installation.html>



The screenshot shows the Icestudio documentation page for the 'latest' version. The left sidebar has a 'GETTING STARTED' section with a 'Installation' menu item, which is highlighted with a red box. This menu has three options: 'GNU/Linux', 'Mac OS', and 'Windows'. Below this are sections for 'Quick Start', 'User Guide', and 'How to...'. Under 'DEVELOPMENT', there are sections for 'Setup', 'Project', 'Blocks', 'Board Rules', and 'Compilers'. The main content area is titled 'Installation' and has a sub-section for 'GNU/Linux'. It contains two numbered steps: '1. Install Python 2.7 and xclip (to enable Copy/Paste). 2. Download the AppImage file and make it executable:'. Below these steps is a green code block with the command '\$ chmod a+x icestudio-0.3.3\*.AppImage'. There is also a 'Hint' section with a note about downloading a ZIP release and using the script 'linux\_installer.sh'. A red box highlights the 'GNU/Linux' section, and a callout box points to it with the text: 'Se elige sistema operativo y se descarga versión consolidada más actual'.

Figura 2.1: Página de descarga Icestudio Septiembre 2018.

Fuente: <https://icestudio.readthedocs.io/en/latest/source/installation.html>

### 2.2. ANTES DE EJECUTAR EL INSTALADOR

Dependerá del sistema operativo, es necesario seguir los pasos que se indican en la web.

## GNU/Linux

1. Install [Python 2.7](#) and [xclip](#) (to enable Copy/Paste).
2. Download the [AppImage](#) file and make it executable:

```
$ chmod a+x icestudio-0.3.3*.AppImage
```

### ! Hint

You can download the ZIP release and install it in the system using the script [linux\\_installer.sh](#). This script registers the .ice files as *Icestudio project*. There is also a [linux\\_uninstaller.sh](#) to revert the previous configuration.

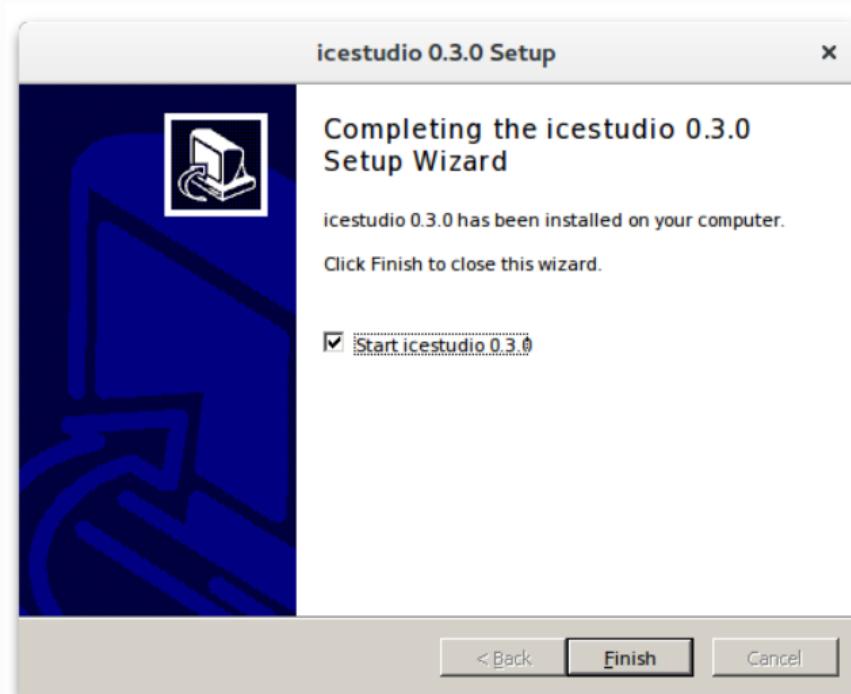
## Mac OS ☺

1. Install [Python 2.7](#).
2. Install [Homebrew](#).
3. Download and execute the [DMG package](#).



## Windows

1. Download and execute the Windows installer.



### Note

Python 2.7 will be installed automatically if it is not installed. This installer registers the .ice files as *Icestudio project*.

Figura 2.2: Pasos para instalar IceStudio en diferentes sistemas operativos. Septiembre 2018.  
Fuente: <https://icestudio.readthedocs.io/en/latest/source/installation.html>

## 2.3. INSTALAR LA TOOLCHAIN

Una vez instalado el programa, antes de comenzar a usarlo hay que instalar la ToolChain. En el menú **herramientas/Toolchain** clicamos Instalar

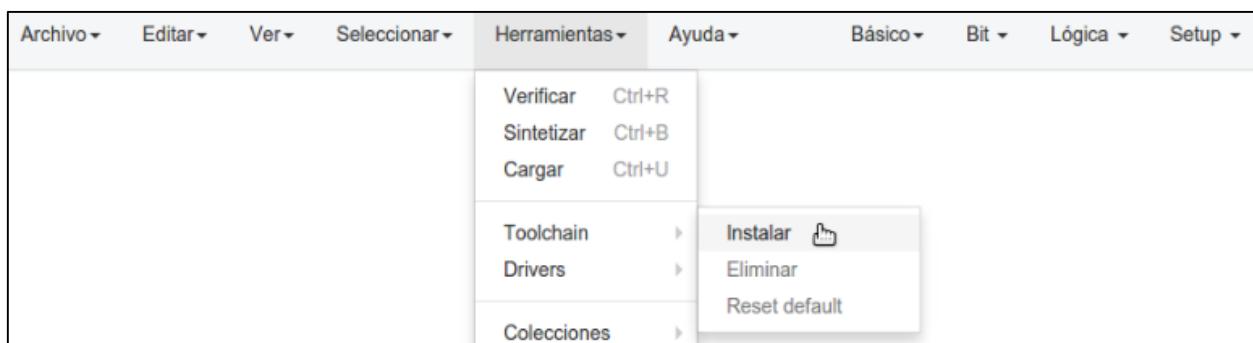


Figura 2.3: Menú herramientas/Toolchain, Icestudio  
Fuente: <https://github.com/Obijuan/digital-electronics-with-open-FPGAs-tutorial/>

Al terminar la instalación debe aparecer una ventana indicando que todo ha ido bien, y dando el aviso para que se configuren los drivers.



Figura 2.4:Avisos de Icestudio.

Fuente: <https://github.com/Obijuan/digital-electronics-with-open-FPGAs-tutorial/>

## 2.4. INSTALAR LOS DRIVERS

Para cargar en la placa los circuitos sintetizados, es necesario que se configure los drivers para ello clicamos habilitar en el menú **Herramientas/Drivers**.



Figura 2.5: Menú herramientas/Drivers, Icestudio

Los pasos a seguir dependerán del sistema operativo del equipo y de la FPGA libre se utilice.

## 2.5. SELECCIÓN DE FPGA

Icestudio permite trabajar con varios tipos de placas, para seleccionar la placa con la que se va a trabajar se accede al menú **Seleccionar/Placa**, y se elige la que corresponda:

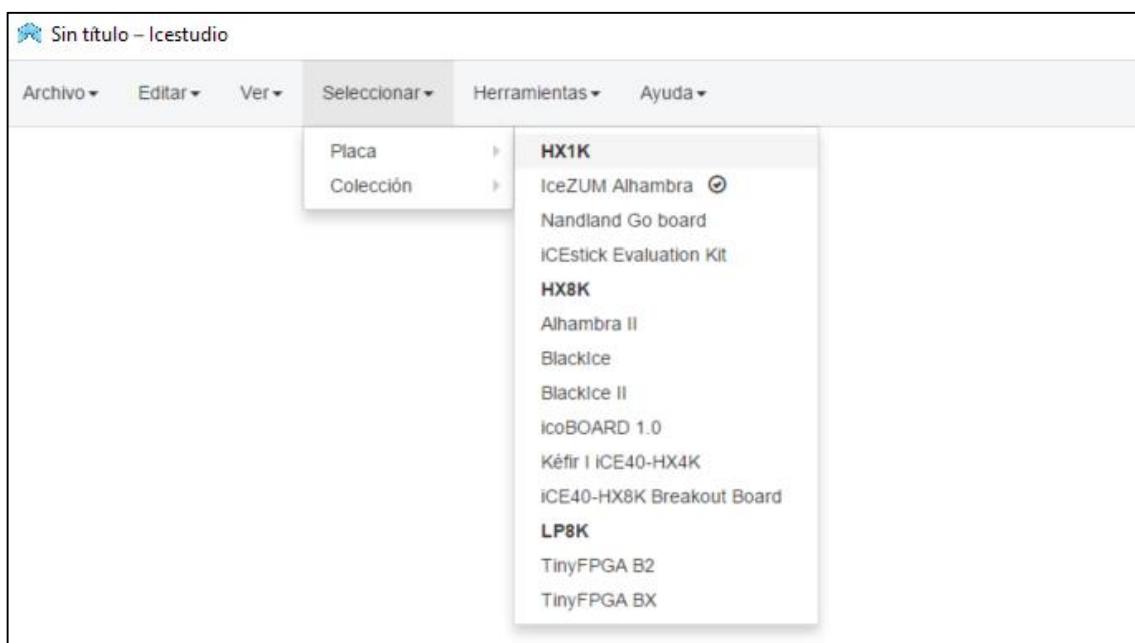


Figura 2.6: Selección Placa

## 2.6. COMPROBAR LA INSTALACIÓN

Para comprobar que la instalación y la configuración del programa han sido correctas, se carga un ejemplo. Para ello se entra en el menú **Archivo/Ejemplos/1.Basic** y se selecciona *One LED*

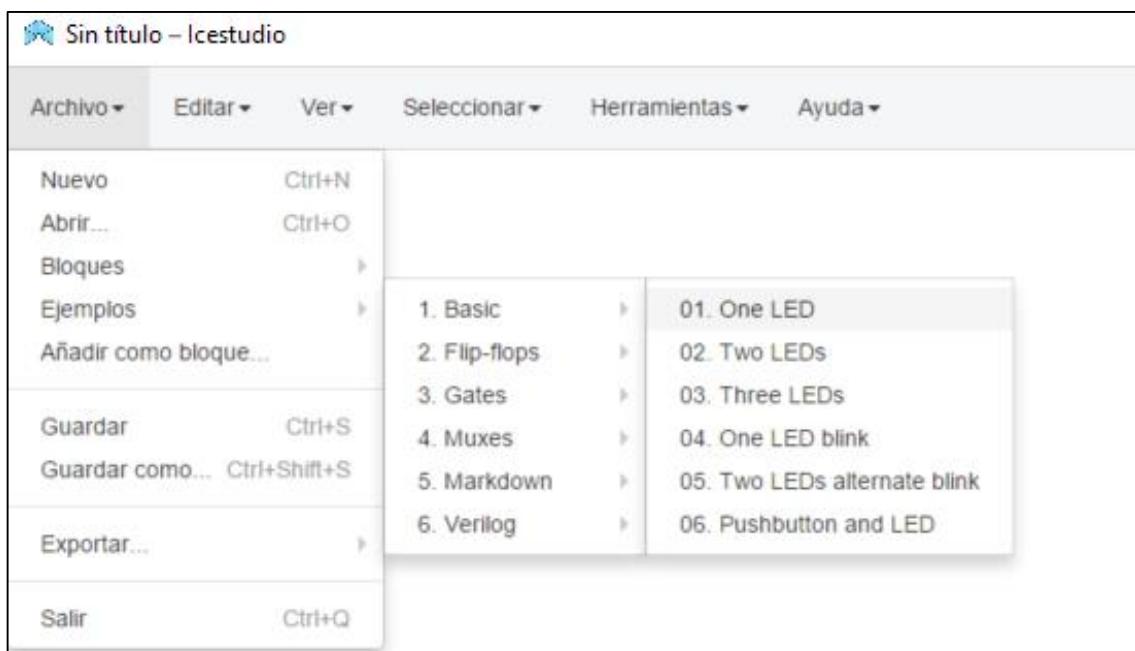
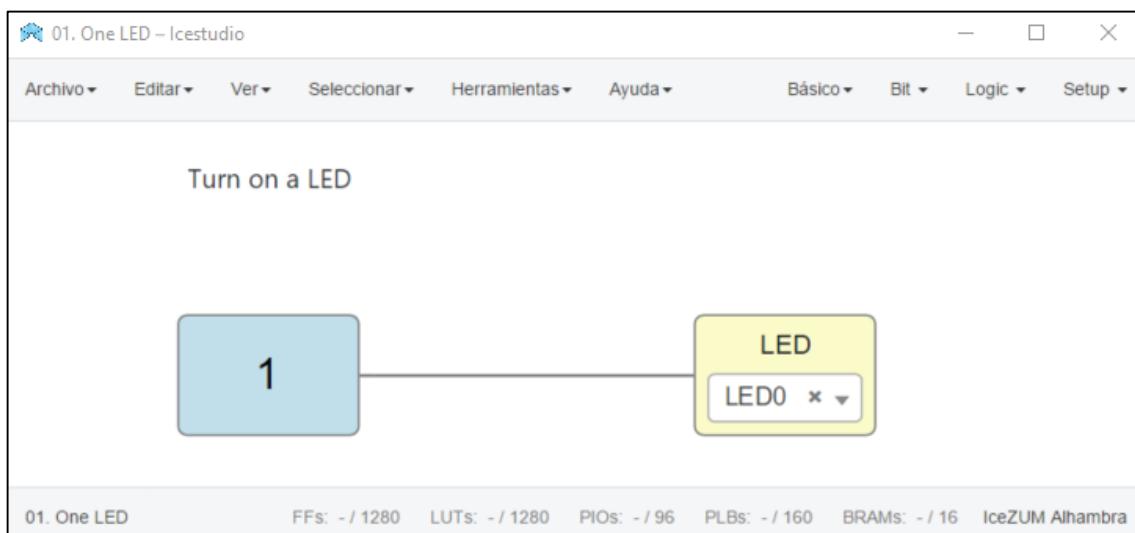


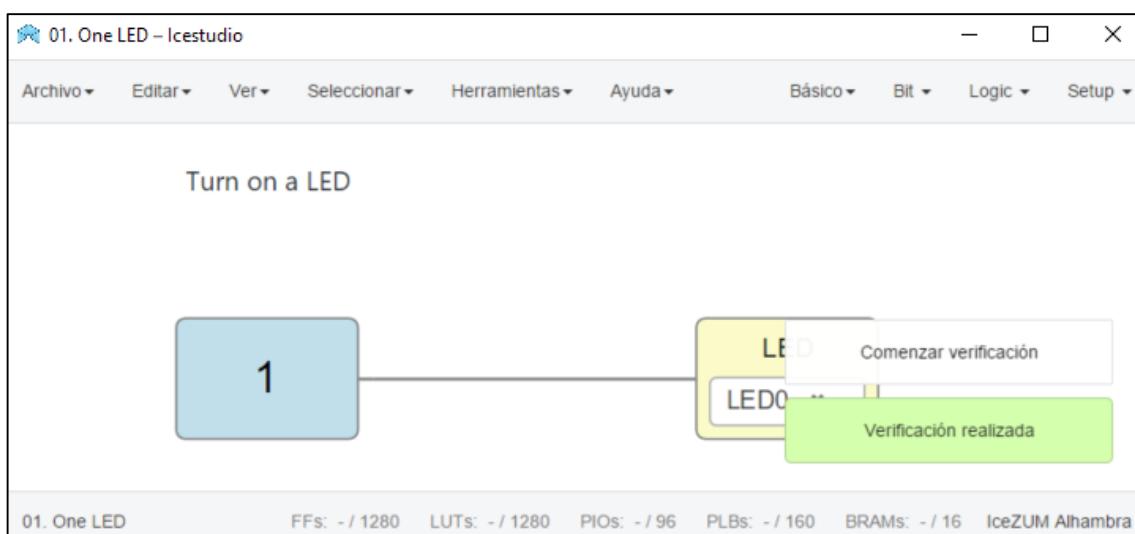
Figura 2.7: Menú Archivo/Ejemplos/1. Basic, Icestudio

Y se abrirá la siguiente ventana:



*Figura 2.8: Ejemplo 01.One LED, Icestudio*

Con la placa conectada al equipo, se accede al menú herramientas/verificar y se verifica el diseño:



*Figura 2.9: Verificación 01.One LED, Icestudio*

Una vez verificado el diseño se procede a sintetizarlo en el menú Herramientas/sintetizar:

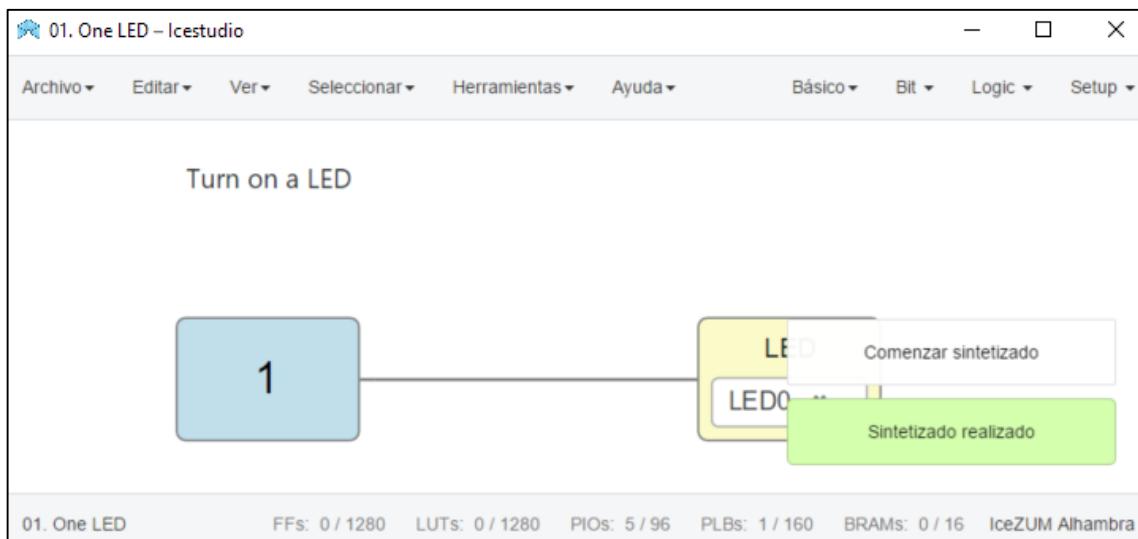


Figura 2.10: Sintetizado 01.One LED, Icestudio

Y para finalizar se carga en la placa en el menú Herramienta/cargar:

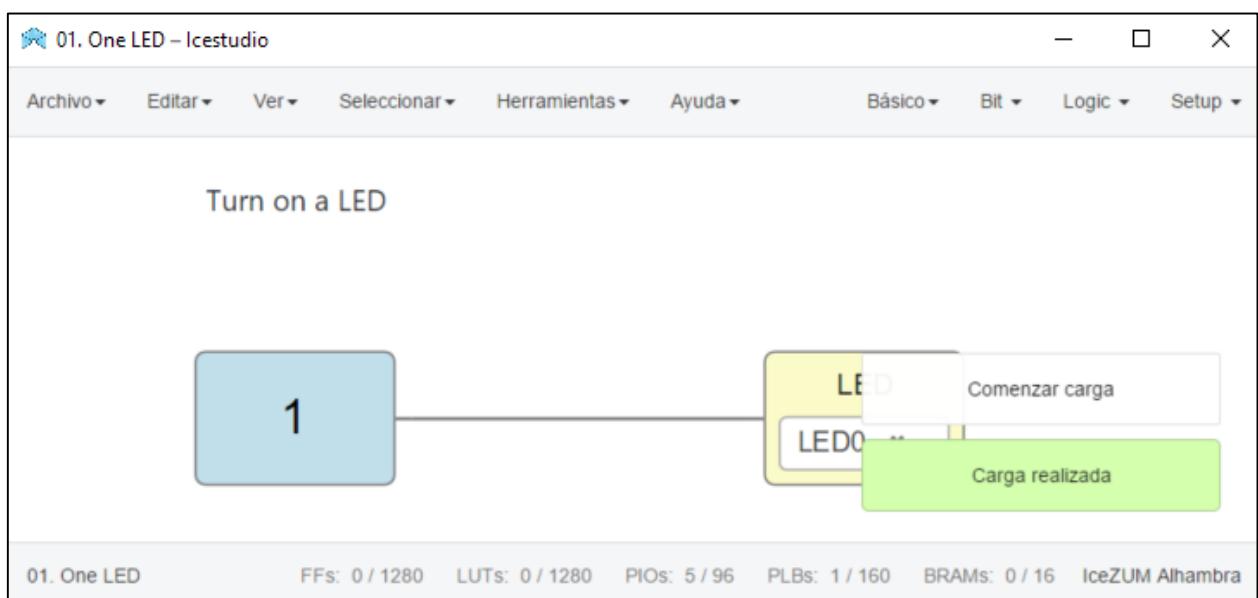


Figura 2.11: Carga 01.One LED, Icestudio

Si todo ha ido bien se podrá ver el LED 0 encendido en la placa:



Figura 2.12: LED 0 encendido, FPGA IceZUM Alhambra.  
Fuente: <https://icestudio.readthedocs.io/en/latest/source/installation.html>

Es posible que la carga no vaya bien, y se Icestudio informe de un error, las posibles causas podrían ser:

- La placa no está conectada al USB → Conectarla
- La placa está conectada a otro puerto USB que no está configurado → conectarla al puerto USB correcto
- La placa no se ha conectado correctamente al USE → revisar conexiones
- El cable USB utilizado no es el adecuado → utilizar un cable de datos
- Problemas con los drivers → Reinstala los Drivers, en Windows a ves hay que intentarlos varias veces.

### 3. ENTORNO DE TRABAJO

Al arrancar el programa (doble clic sobre el ícono del programa), aparece la pantalla de trabajo del programa donde se pueden diferenciar diferentes elementos:

- Menú: Barra superior izquierda, desde donde se podrá acceder a diferentes opciones, de forma que hay un menú principal (Archivo, Edición, Ver,...) y de cada una de las opciones anteriores “cuelgan” otras órdenes que se verán a continuación.
- Menú de bloques: Barra superior derecha, desde este menú se podrá acceder a los distintos bloques que ofrece el programa para configurar los proyectos.
- Nombre proyecto: En la esquina inferior izquierda, podremos ver el nombre del proyecto que coincide con el nombre del archivo de IceStudio con extensión .ice
- Información de la Placa: en la parte inferior derecha de la ventana del programa, se muestra información de la placa que se esté utilizando.
- Área de diseño, la zona más amplia es el área gráfica de diseño, que corresponde a la zona en la que se crean, modifican y se actúa sobre el diseño. En esta zona se pueden encontrar bloques de distintas características:
  - Bloque código: es un bloque, que permite al usuario diseñar sus propios módulos con código verilog.
  - Bloque constante: bloques que adquieren valor constante, se accede a ellos desde la barra de menú bloques
  - Bloques I/O: bloque de entrada o salidas

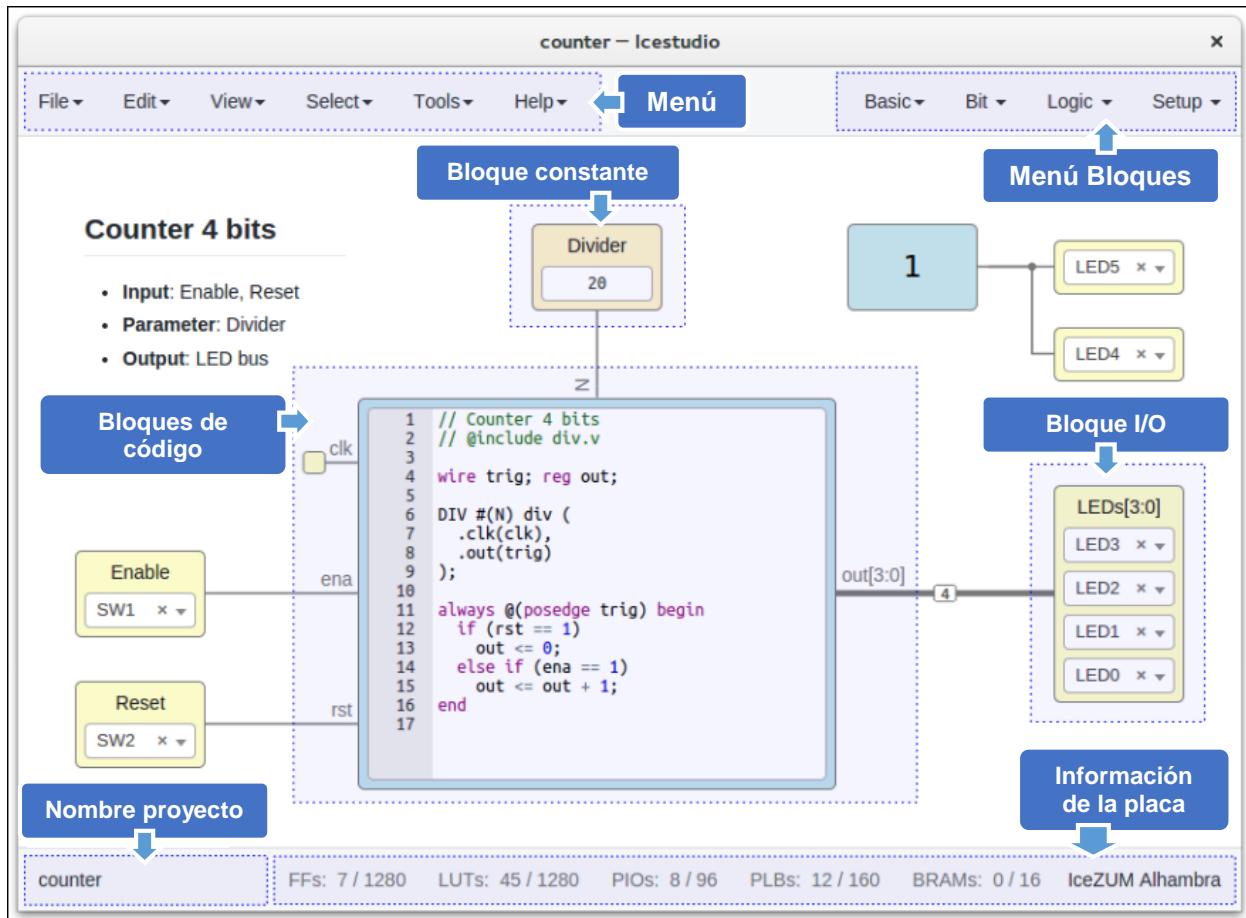


Figura 3.1: Entorno de trabajo Icestudio.  
Fuente: <https://icestudio.readthedocs.io/en/latest/source/userguide.html>

## 4. MENÚ

### 4.1. ARCHIVO



Figura 4.1: Menú archivo.

| Acción                | Descripción   | Abreviaciones |
|-----------------------|---|---------------|
| Nuevo                 | Se crea una nueva ventana   | Ctrl+N        |
| Abrir...              | Abre un proyecto  | Ctrl+O        |
| Bloques               | Abre los bloques de la colección seleccionada   |               |
| Ejemplos              | Abre los ejemplos de la colección seleccionada  |               |
| Añadir como Bloque... | Añade un proyecto como bloque   |               |
| Guarda                | Guarda el proyecto actual   | Ctrl+S        |
| Guarda como...        | Guarda el proyecto actual con un Nuevo nombre   | Ctrl+Shift+S  |
| Exportar...           | Exporta el proyecto a otros archivos de diferente extensión: <b>Verilog, PCF, Testbench, GTKWave, BLIF, ASC and Bitstream</b> |               |
| Salir                 | Cierra la aplicación  | Ctrl+Q        |

## 4.2. EDITAR



Figura 4.2: Menú editar.

| Acción   | Descripción                         | Abreviaciones            |
|----------|-------------------------------------|--------------------------|
| Deshacer | Deshace el último cambio            | Ctrl+Z                   |
| Rehacer  | Rehace la última acción de deshacer | Ctrl+Y /<br>Ctrl+Shift+Z |
| Corta    | Corta los bloques seleccionados     | Ctrl+X                   |
| Copia    | Copia los bloques seleccionados     | Ctrl+C                   |

| Acción               | Descripción   | Abreviaciones |
|----------------------|---|---------------|
| Pega                 | Pega los bloques seleccionados  | Ctrl+V        |
| Selecciona todo      | Selecciona todos los bloques  | Ctrl+A        |
| Ajustar contenido    | Ajusta el contenido al área gráfica   | Ctrl+1        |
| Preferencias         | Preferencias del usuario: Idioma, Reglas de la placa, colecciones extras.<br><b>Reglas de la placa:</b> Habilita o deshabilita globalmente las reglas de la placa. Estas reglas permiten automatizar tareas tales como conexiones de puerto predeterminadas o valores de pin predeterminados. |               |
| Información proyecto | Permite añadir información del proyecto; Versión, descripción, autor ...  |               |

### 4.3. VER



Figura 4.3 Menú ver

| Acción                      | Descripción  |
|-----------------------------|--|
| PCF                         | Muestra el archive pcf de la placa                     |
| Pinout                      | Muestra el SVG pinout de la placa                      |
| Datasheet                   | Abre un navegador web con la hoja de datos de la placa |
| Reglas de la placa          | Muestra la reglas de la placa                          |
| Información de la colección | Muestra el archivo README de la colección actual       |
| Toolchain output            | Muestra el resultado del último comando ejecutado      |
| Recursos de la FPGA         | Muestra/ Oculta los recursos de FPGA utilizados        |

## 4.4. SELECCIONAR

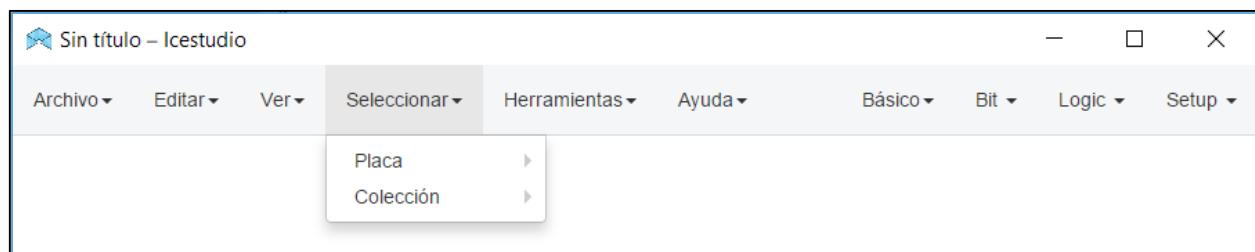


Figura 4.4: Menú Seleccionar

### Seleccionar/ Placa

Se puede seleccionar la placa con la que se está trabajando, en la versión 0.3.3, se pueden encontrar las siguientes:

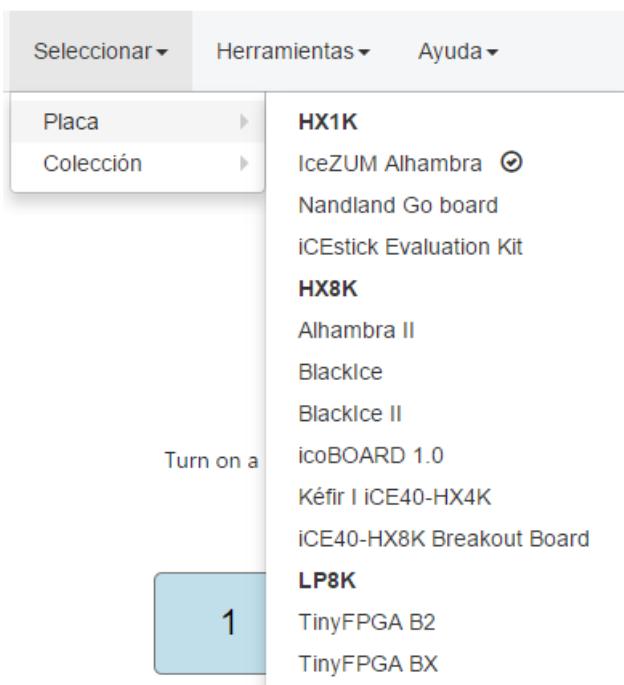


Figura 4.5: Placas disponibles en versión 0.3.3

Cuando se selecciona una placa, todas las combinaciones de bloques de I / O se actualizan y se restablecen sus valores actuales.

### Seleccionar/ Colección

Seleccionar colección, se puede cargar colecciones, o seleccionar la colección que se instala por defecto. La colección seleccionada será la colección de trabajo.

Una colección está compuesta por bloques y ejemplos. Cuando se selecciona una colección, las siguientes secciones se actualizan con el contenido de la colección:

- Archivo> Bloques
- Archivo> Ejemplos
- Bloques de menú

La colección predeterminada siempre está disponible y contiene los bloques y ejemplos distribuidos dentro de la aplicación.

## 4.5. HERRAMIENTAS

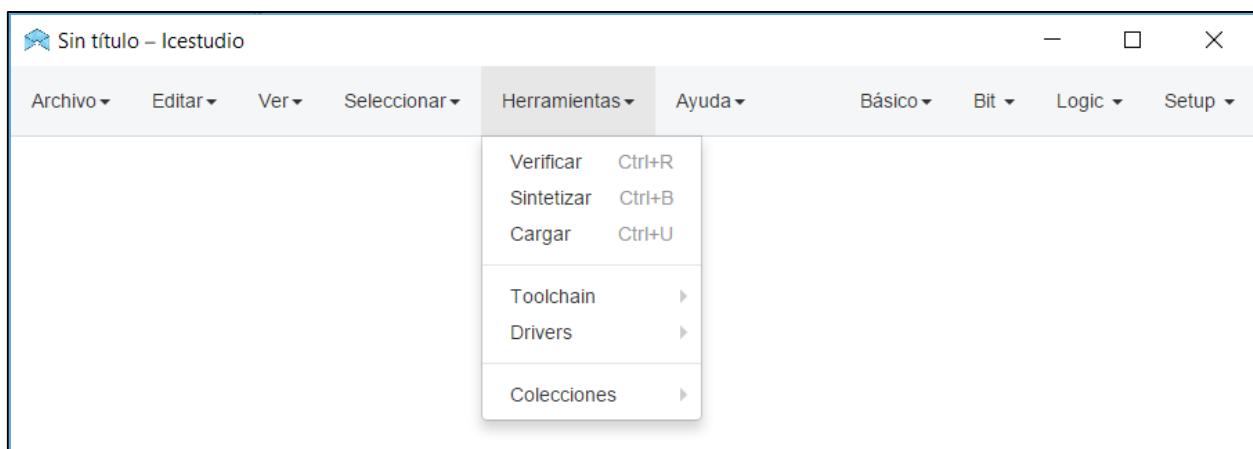


Figura 4.6: Menú Herramientas

| Acción              | Descripción  |  | Abreviación |
|---------------------|--|--|-------------|
| Verificar           | Chequea el código Verilog generado   |  | Ctrl+R      |
| Sintetizar          | Sintetiza el bitstream del diseño  |  | Ctrl+B      |
| Cargar              | Sintetiza (si es necesario) y carga el bitstream a la FPGA   |  | Ctrl+U      |
| <b>Toolchain</b>    |  |  |             |
| Instalar/Actualizar | Instalar un virtualenv, apio y los paquetes apio requeridos. Requiere Python 2.7                       |  |             |
| Eliminar            | Elimina los archivos de toolchain  |  |             |
| Reset default       | Restaura la toolchain predeterminada en Icestudio  |  |             |
| Versión Apio        | Muestra la versión actual de apio  |  |             |
| <b>Drivers</b>      |  |  |             |
| Habilita            | Inicia la configuración de los controladores FTDI. Cada sistema operativo tiene un proceso diferente   |  |             |
| Deshabilita         | Revierte la configuración de los controladores FTDI. Cada sistema operativo tiene un proceso diferente |  |             |

| Acción        | Descripción                                     | Abreviación |
|---------------|---|-------------|
| Colecciones   |   |             |
| Añadir        | Agrega un archivo ZIP con una o más colecciones |             |
| Recargar      | Recarga todas las colecciones de los archivos   |             |
| Eliminar      | Elimina la colección seleccionada               |             |
| Eliminar todo | Elimina todas las colecciones                   |             |

## 4.6. AYUDA

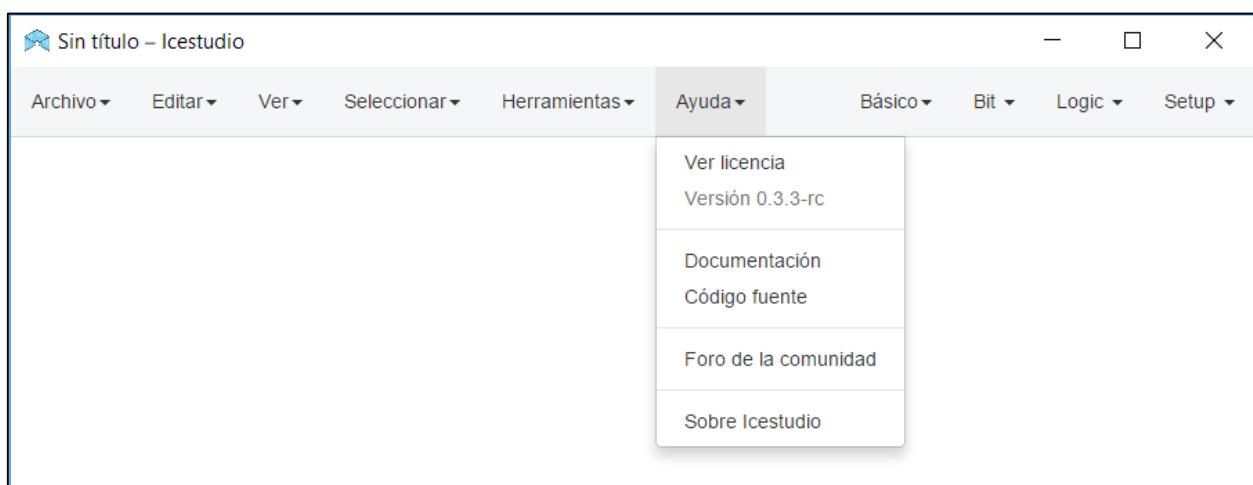


Figura 4.7: Menú ayuda

| Acción               | Descripción  |
|----------------------|--|
| Ver licencia         | Abre la licencia de Icestudio en un navegador web      |
| Versión              | Muestra la versión actual de Icestudio                 |
| Documentación        | Abre la documentación de Icestudio en un navegador web |
| Código fuente        | Abre el código fuente de Icestudio en un navegador web |
| Foro de la comunidad | Abre el foro FPGAwars en un navegador web              |
| Sobre Icestudio      | Información sobre la aplicación                        |

## 5. MENÚ DE BLOQUES

### 5.1. BÁSICO

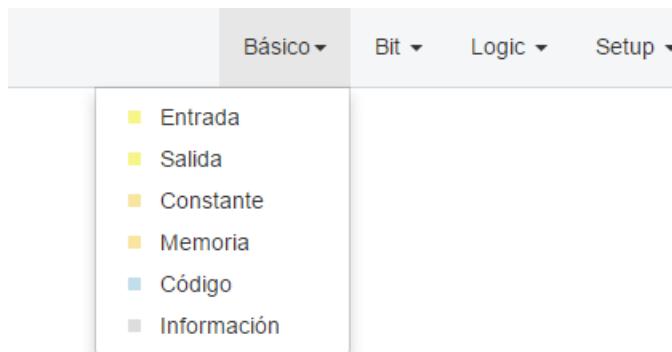


Figura 5.1: Menú Básico

Contiene los siguientes bloques básicos:

- Entrada: muestra un diálogo para insertar el nombre y tipo del bloque de entrada.
- Salida: muestra un diálogo para insertar el nombre y el tipo del bloque de salida.
- Constante: muestra un cuadro de diálogo para insertar el nombre y el tipo del bloque constante.
- Memoria: muestra un cuadro de diálogo para insertar el nombre y el tipo del bloque de memoria.
- Código: muestra un diálogo para insertar los puertos y parámetros del bloque de código.
- Información: crea un bloque de cuadro de texto vacío.

Los puertos de entrada y salida se pueden configurar en virtual. Los puertos virtuales se usan para proyectos independientes de FPGA. Además, se pueden configurar como un bus agregando la notación [x: y] al nombre del puerto.

Los bloques de memoria y constantes se pueden configurar en local. Los parámetros locales no están expuestos cuando el proyecto se agrega como un bloque.

Se pueden crear múltiples bloques de entrada, salida, constante y de memoria usando el separador de coma. Por ejemplo: x, y, z creará 3 bloques con esos nombres. Los valores de los puertos de I/O FPGA se establecen en el cuadro combinado de bloques. Estos valores se pueden establecer buscando y también se pueden deshacer haciendo clic en la cruz. Hacer doble clic sobre la entrada, la salida, la constante o el bloque de memoria permite modificar el nombre y el tipo del bloque. En la definición de puertos de bloques de código, también se pueden crear múltiples puertos de entrada y salida y parámetros utilizando el separador de coma.

## 5.2. BLOQUES

---

Contiene todos los bloques almacenados y ordenados por categorías. Este menú es generado cuando se inicia la aplicación. Puede mostrar la colección predeterminada o cualquier colección instalada.

## 6. ÁREA DE DISEÑO

Esta es la zona principal, donde se colocan los bloques y se conectan entre sí con los cables.

- **Zoom** → El zoom se realiza con la rueda del mouse.
- **Seleccionar** → La selección de un bloque se realiza con el botón izquierdo del mouse. Los bloques se pueden seleccionar / deseleccionar individualmente usando Shift + clic izquierdo. Además, se pueden seleccionar varios bloques mediante un cuadro de selección. Una selección se cancela cuando se hace clic con el botón izquierdo en el fondo.
- **Mover bloques** → Cualquier selección de bloques o bloque se puede mover en el diseño usando el botón izquierdo del mouse sobre el bloque o la selección. También se puede mover una selección de bloques con las teclas de flecha.

### 6.1. CAMBIAR TAMAÑO DE LOS BLOQUES

---

Los bloques de memoria, código e información se pueden cambiar de tamaño con la herramienta de cambio de tamaño en la esquina inferior derecha del bloque.

### 6.2. EXAMEN DE BLOQUE

---

Los bloques no básicos se pueden leer solo examinados haciendo doble clic en el bloque con el botón izquierdo del mouse. Esta es una acción recursiva. Para volver atrás, haga clic en el enlace de atrás o presione la tecla Atrás. Durante el examen, la panorámica, el zoom y la navegación de códigos están habilitados. También la acción 'Ajustar contenido'.

### 6.3. DETECCIÓN DE ERRORES VERILOG

---

Los errores de verificación, compilación y carga se capturan y se muestran en el diseño con una notificación.

- Si el error proviene de un bloque Code, se marca en rojo y se establece una anotación en línea:
- Si el error proviene de un bloque Constante, está marcado en rojo.
- Si el error proviene de un bloque genérico, está marcado en rojo.

## 6.4. DESHACER / REHACER

---

Icestudio permite deshacer / rehacer las siguientes acciones:

- Agregar o eliminar un bloque.
- Agregar o quitar un cable.
- Mover un bloque o una selección de bloque.
- Editar un de I / O: nombre, tipo y valor.
- Editar un bloque Constante: nombre, tipo y valor.
- Editar un bloque de memoria: nombre, tipo, formato y valor.
- Editar un bloque de código: puertos, parámetros y contenido.
- Editar un bloque de información: escriba y contenido.
- Cambiar el tablero.
- Cambiar el idioma.
- Cambiar la información del proyecto: al deshacer / rehacer un cambio de información del proyecto, aparecerá una notificación en la que se puede hacer clic para acceder a la sección de información del proyecto.

## 6.5. TOMA UNA IMAGEN DE LA PANTALLA

---

Tomar una instantánea png de la aplicación es tan fácil como presionar Ctrl + P. Aparecerá un cuadro de diálogo para guardar el nombre y la ruta de la imagen capturada.

## 7. COLECCIONES

Los bloques usados en IceStudio no forman parte de la aplicación, sino que se encuentran en las colecciones mencionadas en apartados anteriores.

### 7.1. LA COLECCIÓN POR DEFECTO

---

Al instar Icestudio, se añade la colección por defecto (default), que contiene unos pocos bloques y algunos ejemplos. Las tres opciones situadas en el menú superior derecho de la barra superior pertenecen a esta colección por defecto: Bit, Lógica y Setup

Y también los ejemplos del menú Archivo. Cuando se cambia a una colección nueva, estas dos partes: los bloques y los ejemplos se actualizarán



Figura 7.1: Colección por defecto, Icestudio  
Fuente: <https://github.com/Obijuan/digital-electronics-with-open-FPGAs-tutorial>

Desde el menú Seleccionar/Colección se puede cambiar a cualquiera de las otras colecciones que estén instaladas en Icestudio.

Para obtener más información sobre una colección, se puede acceder a menú Ver/Información de la colección, aparecerá una ventana con la información de la colección

## 7.2. AÑADIR COLECCIONES

Las colecciones se crean en archivos comprimidos, y es así como se deben añadir. Para añadir una colección se clica en el menú Herramientas/Colecciones/Añadir. Se abre una ventana, donde se seleccionará el archivo comprimido que contiene la colección a añadir. Antes de incorporarse, Icestudio solicitará el nombre de la colección. Una vez se haya añadido aparecerá una notificación de que el proceso ha sido correcto, y la colección ha sido añadida.

## 7.3. SELECCIÓN DE COLECCIÓN

Para seleccionar una colección se accede al menú Seleccionar/Colección, además de la colección por defecto, aparecerá las colecciones que se hayan añadido. Al clicar sobre cualquiera de ellas, cambiaremos de la colección actual (por defecto) a la seleccionada.

Al cargarse la nueva colección aparece una notificación indicando que se ha seleccionado una nueva colección y las opciones del menú de la parte superior derecha se actualizan a los de la nueva colección.

El menú Básico es común a todas las colecciones. Siempre estará ahí

Se puede cambiar de una colección a otra siempre que se quiera. Según la colección seleccionada se tendrán unos bloques u otros, así como los ejemplos

## 7.4. ELIMINAR COLECCIÓN

Opcionalmente se puede eliminar las colecciones que no se vayan a usar. En el menú **Herramientas/Colecciones/Eliminar** aparecen todas las colecciones que están instaladas, se hace click sobre la que se quiere eliminar, y a continuación aparecerá un mensaje de confirmación. Se selecciona OK, y aparece una notificación indicando que la colección ha sido eliminada.

## 8. COMENTARIOS

Icestudio ofrece la opción de añadir comentarios al diseño colocando bloques de información, para ello clicamos en menú **Básico/Información**, aparecerá un bloque gris, con el cursor encima se mueve hasta la posición donde se quieran poner los comentarios y se hace click.



Figura 8.1: Bloque información, Icestudio

El tamaño del bloque se cambia pinchando en su esquina inferior derecha y arrastrándola a la nueva posición.

Para escribir en su interior se hace click dentro y se escriben los comentarios.

Este tipo de bloque se puede editar y cambiar en cualquier momento.

### Cambiar formato bloques información

En los bloques de información se puede incluir código html para resaltar textos o incluir imágenes.

## 9. BIBLIOGRAFÍA

<https://github.com/Obijuan/open-fpga-verilog-tutorial/wiki>

<https://icestudio.readthedocs.io/en/latest/source/userguide.html>

---

## A06-MANUAL VERILOG

---

## ÍNDICE DE CONTENIDOS

---

|   |    |
|---|----|
| 1. INTRODUCCIÓN.....  | 3  |
| 2. NIVELES DE ABSTRACCIÓN EN VERILOG .....                  | 3  |
| 3. PALABRAS RESERVADAS .....                                | 4  |
| 4. DESCRIPCIÓN DE UN DISEÑO VERILOG.....                    | 5  |
| 5. REGLAS DE CONECTIVIDAD DE PUERTOS EN VERILOG .....       | 6  |
| 6. USO DE DECLARACIONES BLOQUEADORAS Y NO BLOQUEADORAS..... | 6  |
| 7. ELEMENTOS BÁSICOS DEL LENGUAJE VERILOG .....             | 7  |
| 7.1. COMENTARIOS.....                                       | 7  |
| 7.2. USO DE MAYÚSCULAS .....                                | 7  |
| 7.3. IDENTIFICADORES.....                                   | 7  |
| 7.4. NÚMEROS .....  | 7  |
| 7.5. TIPOS DE DATOS .....                                   | 9  |
| 7.6. TIPOS DE OPERADORES EN VERILOG .....                   | 10 |
| 7.7. PRECEDENCIA DE LOS OPERADORES.....                     | 13 |
| 8. MÓDULOS EN VERILOG.....                                  | 13 |
| 8.1. CARACTERÍSTICAS.....                                   | 13 |
| 8.2. ESTRUCTURA.....  | 13 |
| 8.3. INTERFAZ DEL MODULO .....                              | 14 |
| 8.4. ASIGNACIONES .....                                     | 14 |
| 8.5. TEMPORIZACIONES.....                                   | 15 |
| 9. MÓDULOS Y JERARQUÍAS .....                               | 16 |
| 9.1. CONEXIONADO.....                                       | 16 |
| 9.2. TIPOS DE ESTRUCTURAS DE MODULOS .....                  | 17 |
| 10. ESTRUCTURAS DE CONTROL.....                             | 17 |
| 10.1. SENTENCIAS CONDICIONALES IF – ELSE .....              | 17 |
| 10.2. SENTENCIA CASE .....                                  | 18 |
| 10.3. SENTENCIA CASEZ Y CASEX .....                         | 18 |
| 10.4. SENTENCIAS DE BUCLE .....                             | 19 |
| 11. FUNCIONES DEL SISTEMA .....                             | 20 |
| 12. DIRECTIVAS PARA EL COMPILADOR .....                     | 24 |
| 12.1. DEFINE .....  | 24 |
| 12.2. INCLUDE.....  | 24 |
| 12.3. IFDEF .....   | 25 |
| 12.4. TIMESCALE .....                                       | 25 |
| 13. PROCESOS DE VERILOG .....                               | 25 |
| 13.1. EVENTOS DE NIVEL:.....                                | 27 |
| 14. TESTBENCH.....  | 28 |
| 14.1. ESTRUCTURA DE UN TESTBENCH.....                       | 28 |
| 14.2. MÓDULO TEST.....                                      | 29 |
| 15. TAREAS EN VERILOG .....                                 | 33 |
| 16. FUNCIONES EN VERILOG .....                              | 34 |
| 17. ALGEBRA BOOLEANA .....                                  | 35 |
| 18. SISTEMAS COMBINACIONALES .....                          | 37 |
| 19. BIBLIOGRAFÍA .....                                      | 41 |

## ÍNDICE DE ILUSTRACIONES

---

|  |    |
|--|----|
| <i>Figura 4.1 Descripción interfaz</i> .....   | 6  |
| <i>Figura 7.1: Definición de las señales de interfaz y nodos internos. Net4 se declara como reg</i> .....  | 10 |
| <i>Figura 7.2: Definición de las señales de interfaz y nodos internos. Net4 se declara como wire</i> ..... | 10 |
| <i>Figura 9.1: Llamada a un módulo</i> .....   | 16 |
| <i>Figura 13.1 Procesos initial y always</i> .....   | 26 |
| <i>Figura 13.2: Error de asignación de valores a variables tipo wire. B) Código corregido.</i> .....       | 27 |
| <i>Figura 14.1: Estructura de un Test-nench</i> .....  | 28 |
| <i>Figura 14.2: Generación de la señal de reset (asíncrona)</i> .....                                      | 31 |
| <i>Figura 14.3: Generación de la señal de reset (síncrona)</i> .....                                       | 31 |
| <i>Figura 14.4: Generación de la señal de reloj</i> .....  | 32 |

## 1. INTRODUCCIÓN

**Verilog HDL** Es utilizado para diseñar sistemas electrónicos, verilog soporta el diseño, prueba e implementación de circuitos analógicos, digitales y de señales mixta a diferentes niveles de complejidad.

Posee una sintaxis similar a la del lenguaje de programación C. El lenguaje tiene un preprocesador como C, y la mayoría de palabras reservadas de control como while, if entre otras son similares.

Verilog es uno de los HDL más utilizados y permite descripciones abstractas y representaciones en bajo nivel es decir puede describir sistemas digitales en base a compuertas, e incluso en base a transistores.

Permite que en un diseño se puedan usar diferentes niveles de descripción de sistemas digitales en un mismo ambiente, las diferentes descripciones pueden ser simuladas para verificar el funcionamiento y además pueden ser sintetizadas es decir traducidas a la interconexión de componentes básicas de un dispositivo programable.

Además permite la descripción estructural del diseño en base a componentes básicas, y descripciones más abstractas que se enfocan en la conducta del sistema. La conducta puede describirse mediante expresiones lógicas y también empleando procedimientos.

Un diseño basado en descripciones funcionales o de comportamiento puede resultar lento y de gran tamaño. Las descripciones en niveles estructurales permiten un ahorro de los circuitos lógicos para maximizar la velocidad, minimizar el tamaño y más bajo costo.

## 2. NIVELES DE ABSTRACCIÓN EN VERILOG

Verilog soporta el diseño de un circuito a diferentes niveles de abstracción, entre los que destacan:

- a) **Nivel de puerta.**- Corresponde a una descripción a bajo nivel del diseño, también denominada modelo estructural. El diseñador describe el diseño mediante el uso de primitivas lógicas (AND, OR, NOT, etc...), conexiones lógicas y añadiendo las propiedades de tiempo de las diferentes primitivas. Todas las señales son discretas, pudiendo tomar únicamente los valores “0”, “1”, “X” o “Z” (siendo “X” estado indefinido y “Z” estado de alta impedancia).
- b) **Nivel de transferencia de registro o nivel RTL.**- Los diseños descritos a nivel RTL especifican las características de un circuito mediante operaciones y la transferencia de datos entre registros. Mediante el uso de especificaciones de tiempo las operaciones se realizan en instantes determinados. La especificación de un diseño a nivel RTL le confiere la propiedad de diseño sintetizable, por lo que hoy

en día una moderna definición de diseño a nivel RTL es todo código sintetizable se denomina código RTL.

- c) **Nivel de comportamiento (Behavioral).**- La principal característica de este nivel es su total independencia de la estructura del diseño. El diseñador, más que definir la estructura, define el comportamiento del diseño. En este nivel, el diseño se define mediante algoritmos en paralelo. Cada uno de estos algoritmos consiste en un conjunto de instrucciones que se ejecutan de forma secuencial. La descripción a este nivel pude hacer uso de sentencias no sintetizables, y su uso se justifica en la realización de los denominados **testbenches** (definidos más adelante).
- d) **Nivel Algorítmico.**- Similar a un programa de alto nivel en C.

### 3. PALABRAS RESERVADAS

Son palabras de léxico propio del lenguaje Verilog con algún significado especial en las expresiones que no pueden ser utilizados fuera del contexto para lo que están reservadas, se describen a continuación:

| PALABRAS RESERVADAS |              |                 |                      |          |
|---------------------|--------------|-----------------|----------------------|----------|
| Always              | Endfunction  | Join            | pullup               | Supply1  |
| And                 | Endgenerate  | Large           | Pulsestyle_onevent   | table    |
| Assign              | Endmodule    | Liblist         | Pulsestyle_onedetect | Task     |
| automatic           | Endprimitive | Localparam      | Rcmos                | Time     |
| Begin               | Endspecify   | Macromodule     | Real                 | Tran     |
| Buf                 | Endtable     | Médium          | Realtime             | Tranif0  |
| Bufif0              | Endtask      | Module          | Reg                  | Tranif1  |
| Bufif1              | Event        | Nand            | Reléase              | Tri      |
| Case                | For          | Negedge         | Repeat               | Tri0     |
| Casex               | Forcé        | Nmos            | Rnmos                | Tri1     |
| Casez               | Forever      | Nor             | Rpmos                | Triand   |
| Cell                | Fork         | Not             | Rtran                | Trior    |
| Cmos                | Function     | Noshowcancelled | Rtrainf0             | Trireg   |
| Config              | Generate     | Notif0          | Rtrainf1             | Unsigned |
| Deassign            | Genvar       | Notif1          | Scalared             | Use      |
| Default             | Highz0       | Or              | Signed               | Vectored |
| Defparam            | Highz1       | Output          | Showcancelled        | Wait     |
| Desing              | If           | Parameter       | Small                | Wand     |
| Disable             | Ifnone       | Pmos            | Specify              | Weak0    |
| Edge                | Initial      | Posedge         | Specparam            | Weak1    |
| Else                | Instance     | Primitive       | Strength             | While    |
| End                 | Inout        | Pull0           | Strong0              | Wire     |
| Endcase             | Input        | Pull1           | Strong1              | Wor      |
| Endconfig           | Integer      | Pulldown        | Supply0              | Xnor     |
|                     |              |                 |                      | Xor      |

## 4. DESCRIPCIÓN DE UN DISEÑO VERILOG

La descripción de un diseño en Verilog comienza con la sentencia de declaración del módulo:

### 1. Declaración de módulo:

Indica el inicio de la definición de módulos. Esta es estrictamente necesaria. Sintaxis:

**module** <nombre\_de\_módulo> <(definición señales de interfaz, lista de puertos>;

### 2. Declaración de puerto (declaración de entradas y salidas):

Indica la dirección, ancho y nombre del puerto. Sintaxis:

in/out/inout <[MSB:LSB]> nombre\_de\_puerto;

### 3. Declaración de registros y cables:

Indica el ancho y nombre del registro o cable. Sintaxis:

reg/wire <[MSB:LSB]> nombre\_de\_registro/nombre\_de\_cable;

### 4. Instancias de componentes:

Instancia de subbloque o compuerta. El nombre de la instancia debe ser único. Sintaxis:

subbloque/nombre\_módulo/compuerta nombre\_instancia(lista de puertos conectividad)

### 5. Assign:

Asignación de valores a una conexión (wire). Sintaxis:

assign nombre\_de\_conexion = <#delay>nombre\_del\_registro/nombre\_de\_conexión;

### 6. Cuerpo del módulo:

Es el corazón del código HDL, contiene la descripción comportamental o estructural de toda la lógica combinacional y secuencial. Incluye las declaraciones always e initial, expresiones lógicas y aritméticas, los comandos case y muchos otros.

### 7. Declaración de fin de módulo:

Indica el fin de la definición de un módulo. Esta es estrictamente necesaria. Sintaxis:

endmodule.

### Ejemplo de los componentes de descripción:

```
module nombre_de_módulo <(lista_de_puertos>;
    in nombre_de_puerto;
    out nombre_de_puerto;
    inout nombre_de_puerto;
    reg/wire <[MSB:LSB]> nombre_de_registro/nombre_de_cable;
        assign nombre_deConexion = nombre_del_registro/nombre_de_conexión;
    endmodule
```

La Figura 4.1 muestra las diferentes partes en la descripción de un diseño.

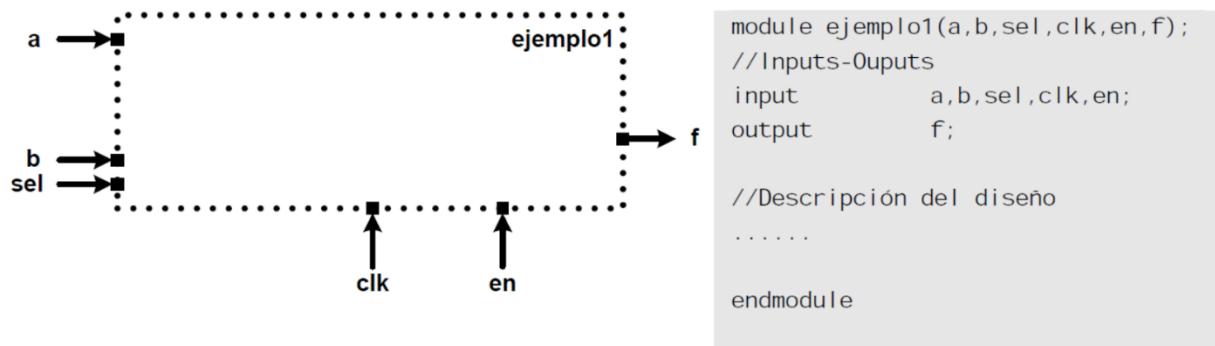


Figura 4.1 Descripción interfaz

Fuente: <http://www.iuma.ulpgc.es/~nunez/clases-FdC/verilog/Verilog%20Tutorial%20v1.pdf>

## 5. REGLAS DE CONECTIVIDAD DE PUERTOS EN VERILOG

Cuando se conectan elementos de hardware considere lo siguiente:

- Registros (registers) o redes (nets) externos se deben conectar únicamente a entradas (inputs) o redes internas.
- Salidas (outputs), registros (registers) o redes (nets) internos se deben conectar únicamente a redes (nets) externas.
- Entrada/salidas (inouts) deben ser conectados únicamente a redes (nets) externas.

## 6. USO DE DECLARACIONES BLOQUEADORAS Y NO BLOQUEADORAS

Las **asignaciones bloqueadoras** son utilizadas para especificar comportamientos combinacionales. Esto significa que al argumento de la derecha se le asigna al argumento de la izquierda en exactamente en el orden en que se escriben las asignaciones.

Su sintaxis es: → **variable = expresión;**

Por otro lado, las **asignaciones no bloqueadoras** son aquellas que se ejecutan en paralelo, esto significa que se ejecutan independientemente de su orden de aparición, y permiten especificar comportamientos secuenciales.

Su sintaxis es: → **variable <= expresión;**

El tipo de sentencia que se utiliza tiene un impacto directo sobre el circuito sintetizado, de forma que el programa escrito en el HDL podría no comportarse tal y como se esperaba que lo hiciera. Es por ello que al programar se deben observar las siguientes reglas de oro:

- Nunca mezcle asignaciones bloqueadoras y no bloqueadoras en el mismo bloque de código.
- Use sentencias no bloqueadoras para describir Flip-Flops y hardware similar, pero use sentencias bloqueadoras cuando el código describa lógica combinacional, es decir que el orden de las asignaciones sí cambia el resultado.

## 7. ELEMENTOS BÁSICOS DEL LENGUAJE VERILOG

Antes de comenzar es preciso conocer algunos elementos básicos del lenguaje Verilog.

### 7.1. COMENTARIOS

Los comentarios van precedidos de los caracteres **//**, cuya información hasta el final de la línea es ignorado. También, pueden ir entre **/\*** y **\*/**, donde la información entre estos caracteres es ignorada, la diferencia con la anterior alternativa, es que en esta segunda puede haber más de una línea. Ejemplos:

```
// Esto es un comentario de una línea  
/* Esto es un comentario de varias líneas */
```

### 7.2. USO DE MAYÚSCULAS

Verilog es sensible al uso de mayúsculas. Se recomienda el uso únicamente de minúsculas.

### 7.3. IDENTIFICADORES

Los identificadores en Verilog deben comenzar con un carácter, pueden contener cualquier letra de la **a** a la **z**, caracteres numéricos además de los símbolos “**\_**” y “**\$**”. El tamaño máximo es de 1024 caracteres.

```
reg A;  
reg A0;  
reg data_i;
```

### 7.4. NÚMEROS

Los números en Verilog pueden especificarse en decimal, hexadecimal, octal o binario. Los números negativos se representan en complemento a 2 y el carácter “**\_**” puede utilizarse para una representación más clara del número. La sintaxis para la representación de un número es la siguiente.

**<TAMAÑO> <BASE> <VALOR>**

- a) **Tamaño:** Es el número de bits expresado en decimal de la cantidad que viene a continuación. Este dato es opcional, en caso de no darse, por defecto el valor es de 32 bits.
- b) **Base:** indica la base en la que se va a expresar el valor. Es opcional, por defecto es decimal.

'b Base binaria  
 'd Base decimal  
 'h Base hexadecimal  
 'o Base octal

- c) **Valor:** Verilog tiene 4 valores que cualquier señal puede tomar:

| VALORES DE SEÑALES |  |
|--------------------|--|
| VALOR              | SIGNIFICADO  |
| 0                  | Un valor binario de cero. Corresponde a cero voltios.  |
| 1                  | Un valor binario de 1. Dependiendo de la tecnología de fabricación, puede corresponder a +5V, +3.3V, o algún otro valor positivo.                                  |
| X                  | Un valor cualquiera los valores x no son ni 0 o 1, y debería ser tratado como un valor desconocido.  |
| Z                  | Un valor de alta impedancia de un bufer de tres estados cuando la señal de control no está definida. Corresponde a un wire que no está conectado, o está flotando. |

Los números negativos se especifican poniendo el signo “-“ delante del tamaño de la constante, tal y como se especifica en el siguiente ejemplo:

-8'd2 → 11111110

La representación interna de los números negativos en Verilog se realiza en complemento a 2.

| Entero             | Almacenado como                    | Descripción                   |
|--------------------|------------------------------------|-------------------------------|
| 1                  | 00000000000000000000000000000001   | Unzised 32 bits               |
| 8'hAA              | 10101010                           | Sized hex                     |
| 6'b10_0011         | 100011                             | Sized binary                  |
| 'hF                | 0000000000000000000000000000001111 | Unzised hex 32 bits           |
| 6'hCA              | 001010                             | Valor truncado                |
| 6'hA               | 001010                             | Relleno de 0's a la izquierda |
| 16'bzzzzzzzzzzzzzz | zzzzzzzzzzzzzz                     | Relleno de z's a la izquierda |
| 8'bxxxxxxx         | xxxxxxxx                           | Relleno de x's a la izquierda |
| 8'b10000001        | 00000001                           | Relleno de 0's a la izquierda |

## 7.5. TIPOS DE DATOS

Existen en Verilog **dos tipos de datos principalmente**:

- **Nets.** Representan conexiones estructurales entre componentes. No tienen capacidad de almacenamiento de información. De los diferentes tipos de nets el más usado es el tipo **wire**.
- **Registers.** Representan variables con capacidad de almacenar información. De los diferentes los más utilizados son el tipo **reg** y el tipo **integer** (estos últimos solo en la construcción de los **testbenches**).

La sintaxis para declarar estas variables es la siguiente.

```
<TIPO> [<MSB> : <LSB>] <NOMBRE>;
```

- **Tipo:** existen varios tipos de variables, aunque las más destacadas son reg y wire.
  - reg:** Representan variables con capacidad de almacenar información.
  - wire:** Representan conexiones estructurales entre componentes. No tienen capacidad de almacenamiento.
  - integer:** Registro de 32 bits.
  - real:** Registro capaz de almacenar números en coma flotante
  - time:** Registro sin signo de 64 bits.
- **MSB y LSB:** Por defecto estas variables son de un sólo bit, aunque es posible definir vectores de bits.
- **Nombre:** Indica el nombre de la variable.

A continuación, se muestran varios ejemplos de variables.

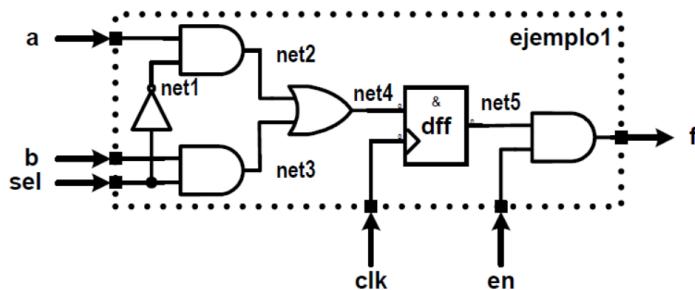
```
reg[5:0] data;      // Registro de 6 bits, donde data[0] es el bit menos significativo
wire outA;          // Net de un bit
integer numA;        // Registro de 32 bits
reg[31:0] numB;    // Registro de 32 bits
```

Las señales de interfaz y los nodos internos se declaran de la siguiente forma:

- **Inputs.** El tipo de las señales de entrada NO SE DEFINEN, por defecto se toman como **wire**.
- **Outputs.** Las salidas pueden ser tipo **wire** o **reg**, dependiendo si tienen capacidad de almacenamiento de información. OJO, en Verilog, un nodo tipo **wire** puede atacar a una salida.
- **Nodos internos.** Siguen la misma filosofía que las salidas.

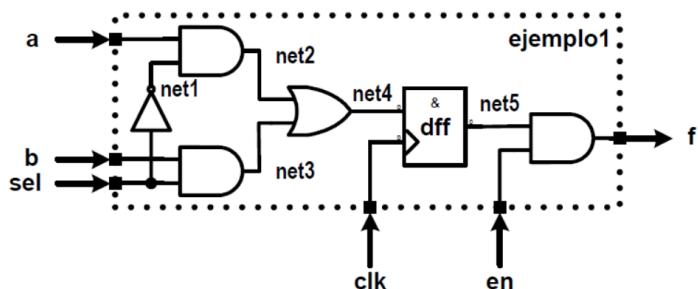
En las siguientes figuras se introduce la definición de los nodos, tanto **wire** como **reg**, se puede observar que en cada uno de los diseños net4 se ha declarado de forma diferente, en el

primer caso se declara de tipo **reg** y en el segundo de tipo **wire**. Si observamos el diseño final es idéntico, la diferencia está en la forma de definir el código



```
module ejemplo1(a,b,sel,clk,en,f);
//Inputs-Outputs
input a,b,sel,clk,en;
output f;
wire f;
//Descripción de los nodos internos
reg net5;
wire net1,net2,net3,net4;
// Descripción del diseño
endmodule
```

Figura 7.1: Definición de las señales de interfaz y nodos internos. Net4 se declara como reg  
Fuente: <http://www.iuma.ulpgc.es/~nunez/clases-FdC/verilog/Verilog%20Tutorial%20v1.pdf>



```
module ejemplo1(a,b,sel,clk,en,f);
//Inputs-Outputs
input a,b,sel,clk,en;
output f;
wire f;
//Descripción de los nodos internos
reg net4,net5;
wire net1,net2,net3;
// Descripción del diseño
endmodule
```

Figura 7.2: Definición de las señales de interfaz y nodos internos. Net4 se declara como wire  
Fuente: <http://www.iuma.ulpgc.es/~nunez/clases-FdC/verilog/Verilog%20Tutorial%20v1.pdf>

## 7.6. TIPOS DE OPERADORES EN VERILOG

Los operadores aritméticos y/o lógicos se utilizan para construir expresiones. Estas expresiones se realizan sobre uno o más operandos. Estos últimos pueden ser nodos, registros constantes, etc...

### OPERADORES ARITMÉTICOS

De un operando:

|     |            |                      |
|-----|------------|----------------------|
| “+” | a = +8”h01 | Resultado: a = 8”h01 |
| “-” | a = -8”h01 | Resultado: a = 8”hFF |

### De dos operandos:

|      |  |
|------|--|
| “+”  | $a = b + c;$   |
| “-”  | $a = b - c;$   |
| “*”  | $a = b * c;$   |
| “/”  | $a = b / c;$ Resultado: Se devuelve la parte entera de la división |
| “%*” | $a = b \% c;$ Resultado: Se devuelve el módulo de la división      |

El resultado del módulo toma el signo del primer operando.

Si algún bit del operando tiene el valor “x”, el resultado completo es “x”.

Los números negativos se representan en complemento a 2.

### OPERADORES RELACIONALES

|      |          |                          |
|------|----------|--------------------------|
| “<”  | $a < b$  | a es menor que b         |
| “>”  | $a > b$  | a es mayor que b         |
| “<=” | $a <= b$ | a es menor o igual que b |
| “>=” | $a >= b$ | a es mayor o igual que b |

El resultado que se devuelve es:

|   |  |
|---|--|
| 0 | si la condición no se cumple                     |
| 1 | si la condición se cumple                        |
| x | si alguno de los operandos tiene algún bit a “x” |

### OPERADORES DE IGUALDAD

|      |           |  |
|------|-----------|--|
| “==” | $a == b$  | a es igual a b                           |
| “!=” | $a !=$    | b a es diferente de b                    |
| “==” | $a === b$ | a es igual a b, incluyendo “x” y “z”     |
| “!=” | $a !== b$ | a es diferente a b, incluyendo “x” y “z” |

Los operandos se comparan bit a bit, rellenando con ceros para ajustar el tamaño en caso de que no sean de igual ancho. Estas expresiones se utilizan en condicionales. El resultado es:

|   |   |
|---|---|
| 0 | si se cumple la igualdad  |
| 1 | si no se cumple la igualdad   |
| x | únicamente en las igualdades “==” o “!=” si algún operando contiene algún bit a “x” o “z” |

### OPERADORES LÓGICOS

|      |                 |
|------|-----------------|
| “!”  | negación lógica |
| “&&” | AND lógica      |
| “  ” | OR lógica       |

Las expresiones con “&&” o “||” se evalúan de izquierda a derecha. Estas expresiones se utilizan en condicionales. El resultado es:

|   |  |
|---|--|
| 0 | si la relación es falsa                          |
| 1 | si la relación es verdadera                      |
| x | si algún operando contiene algún bit a “x” o “z” |

## OPERADORES BIT A BIT (BIT-WISE)

---

|     |              |
|-----|--------------|
| “~” | negación     |
| “&” | AND          |
| “ ” | OR           |
| “^” | OR exclusiva |

Los citados operadores pueden combinarse obteniendo el resto de operaciones, tales como  $\sim\&$  (AND negada),  $\sim^$  (OR exclusiva negada), etc... El cálculo incluye los bits desconocidos (x) en la siguiente manera:

$$\begin{aligned}\sim x &= x \\ 0 \& x &= 0 \\ 1 \& x &= x \& x = x \\ 1 | x &= 1 \\ 0 | x &= x | x = x \\ 0 ^ x &= 1 ^ x = x ^ x = x\end{aligned}$$

## OPERADORES DE REDUCCIÓN

---

|              |     |  |
|--------------|-----|--|
| “&”          | AND | Ejemplo: $\&a$ Devuelve la AND de todos los bits de a.               |
| “ $\sim\&$ ” | AND | negada Igual que el anterior.  |
| “ ”          | OR  | Ejemplo: $ a$ Devuelve la OR de todos los bits de a.                 |
| “ $\sim $ ”  | OR  | negada Igual que el anterior.  |
| “^”          | OR  | exclusiva. Ej: $^a$ Devuelve la OR exclusiva de todos los bits de a. |
| “ $\sim^$ ”  | OR  | exclusiva negada Igual que el anterior                               |

Estos operadores de reducción realizan las operaciones bit a bit sobre un solo operando. Las operaciones negadas operan como AND, OR o EXOR pero con el resultado negado.

## OPERADORES DE DESPLAZAMIENTO

---

|      |                            |                        |
|------|----------------------------|------------------------|
| “<<” | Desplazamiento a izquierda | Ejemplo: $a = b << 2;$ |
| “>>” | Desplazamiento a derecha   | Ejemplo: $a = b >> 2;$ |

Notas: Los bits desplazados se rellenan con ceros.

## OPERADOR DE CONCATENACIÓN

---

“{}” Concatenación de operandos. Los operandos se separan por comas.

Ejemplos:

{a,b[3:0],c} Si a y c son de 8 bits, el resultado es de 20 bits  
{3{a}} Es equivalente a {a,a,a}  
{b,3{c,d}} Es equivalente a {b,c,d,c,d,c,d}

Nota: No se permite la concatenación con constantes sin tamaño.

## OPERADOR CONDICIONAL

---

“?” El formato de uso de dicho operador es (condición) ? trae\_expr : false\_expr

Ejemplos:

out = (enable) ? a : b; La salida out toma el valor de a si enable está a 1, en caso contrario toma el valor de b.

**Son los tipos de operadores más usados en el lenguaje Verilog.**

## 7.7. PRECEDENCIA DE LOS OPERADORES

---

El orden de precedencia de los operadores es el siguiente:

- Unary, Multiplicación, División, Módulo
- Suma, Resta, Desplazamientos
- Relación, Igualdad
- Reducción
- Lógicos
- Condicional

## 8. MÓDULOS EN VERILOG

### 8.1. CARACTERÍSTICAS

---

Las características principales de los módulos se pueden recoger en los siguientes puntos.

- Cada módulo dispone de una serie de entradas y salidas, por las cuales se puede interconectar otros módulos, aunque puede no tener entradas ni salidas, como es el caso de los testbenches.
- No existen variables globales.
- Fuera de los módulos solo hay directivas del compilador, que afectarán a partir del punto en donde aparecen.
- A pesar de que se pueden realizar varias simulaciones concurrentes, en general se suele tener un único módulo que emplea los módulos previamente definidos.
- Cada módulo puede describirse de forma arquitectural o de comportamiento.

### 8.2. ESTRUCTURA

---

En Verilog un sistema digital está compuesto por la interconexión de un conjunto de módulos. La estructura general de estos módulos es la siguiente:

```
module <nombre> (<señales>);
  input, output <declaración de señales>
  <funcionalidad del módulo>
endmodule
```

## 8.3. INTERFAZ DEL MODULO

---

Los argumentos del módulo pueden ser de tres tipos, estos argumentos comunicarán el interior o funcionalidad del módulo con otros elementos del propio diseño.

- **Input:** Entradas del módulo, cuyo tipo son wire.
- **Output:** Salidas del módulo. Dependiendo del tipo de asignación que las genere serán wire si proceden de una asignación continua y reg si proceden de una asignación procedural.
- **Inout:** Son a la vez entradas y salidas. Únicamente, son de tipo wire.

## 8.4. ASIGNACIONES

---

Existen dos maneras de asignar valores en Verilog, de forma continua o procedural.

### **ASIGNACIÓN CONTINUA**

---

La asignación continua se utiliza exclusivamente para modelar lógica combinacional, donde no se necesita de una lista de sensibilidad para realizar la asignación. La variable a la que se realiza la asignación continua sólo puede ser declarada de tipo net, es decir wire. Y la asignación sólo puede ser declarada fuera de cualquier proceso y nunca dentro de bloques always o initial. La sintaxis es la siguiente.

```
assign variable <#delay> = asignación
```

Un ejemplo de la asignación continua podría ser el siguiente.

```
assign A = B & C;
```

En el ejemplo no se ha añadido una temporización en la asignación, por lo tanto en cada ciclo se revisará dicha sentencia. Estas asignaciones también se ejecutan de forma secuencial las cuales se pueden controlar mediante la temporización.

### **ASIGNACIÓN PROCEDURAL**

---

La asignación procedural es la que se ha visto hasta el momento, donde a las variables se le asigna un valor dentro de un proceso always o initial, el tipo de variable a la que se le asigna el valor puede ser de cualquier tipo. Su sintaxis es la siguiente.

```
<#delay> variable = asignación
```

Un ejemplo de esta asignación es el siguiente.

**A = B & C;**

## 8.5. TEMPORIZACIONES

Las temporizaciones se consiguen mediante el uso de retardos. El retardo se especifica mediante el símbolo # seguido de las unidades de tiempo del retardo. Un ejemplo podría ser el siguiente, donde en el instante 0 se ejecutan las dos primeras sentencias, cinco unidades de tiempo más tarde se realiza la tercera asignación y cuatro más tarde se ejecutan las dos últimas.

```
initial
begin
    clk = 0;
    rst = 0;
#5 rst = 1;
#4 clk = 1;
    rst = 0;
end
```

En las temporizaciones existen dos tipos de asignaciones procedurales.

- **Con bloqueo:** La asignación se realiza antes de proceder con la siguiente.
- **Sin bloqueo:** El término se evalúa en el instante actual, pero no se asigna hasta finalizar dicho instante.

Con el siguiente ejemplo se entiende de mejor manera las asignaciones:

```
initial
begin
    // Asignación con bloqueo
    a = 5;
    #1 a = a + 1;
    b = a + 1;
    // Asignación sin bloqueo
    a = 5;
    #1 a <= a + 1;
    b = a + 1;
end
```

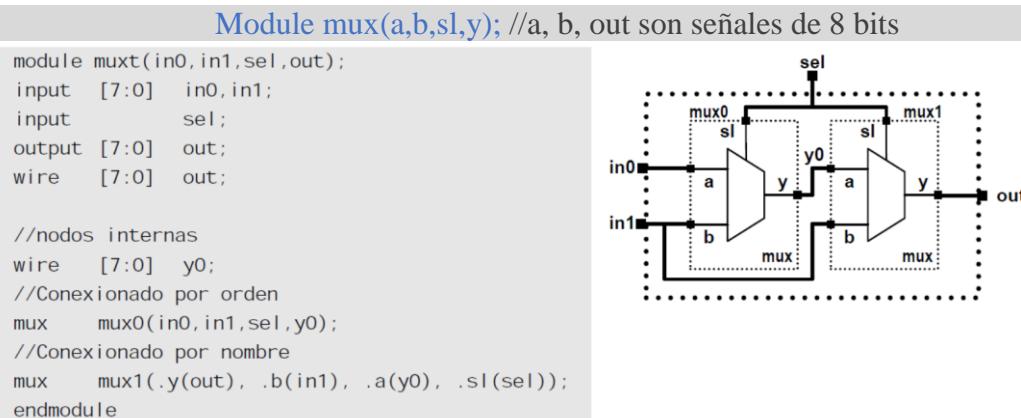
En el ejemplo, inicialmente se realiza una asignación con bloqueo, donde el resultado será a = 6 y b = 7, ya que a se ha calculado previamente. Mientras que en la asignación sin bloqueo a y b serán 6, ya que a no ha sido calculado.

## 9. MÓDULOS Y JERARQUÍAS

El identificador module se utiliza para la definición de módulos/diseños. Estos módulos pueden utilizarse para construir diseños de mayor complejidad, creando lo que se denomina un diseño con jerarquía. La manera de incluir un módulo en un diseño es:

```
master_name instante_name(port_list);
```

La figura siguiente muestra un ejemplo de un módulo, denominado **muxt**, formado por dos módulos, denominados **mux**, y cuya declaración de interfaz es la que a continuación se muestra:



*Figura 9.1: Llamada a un módulo*

Fuente: <http://www.iuma.ulpgc.es/~nunez/clases-FdC/verilog/Verilog%20Tutorial%20v1.pdf>

### 9.1. CONEXIONADO

El conexionado de un módulo dentro de un diseño se puede realizar de dos formas: **por orden** (conexión implícita) o **por nombre** (conexión explícita).

En el conexionado por orden, utilizado en mux0 en el ejemplo anterior, las señales utilizadas en la llamada al bloque se asignan a las señales internas por **orden**. En ese caso, si se tiene en cuenta la declaración del módulo:

```
module mux(a,b,sl,y); //a, b, out son señales de 8 bits
```

y la llamada al mismo:

```
mux mux0(in0,in1,sel,y0);
```

se producen la siguientes asignaciones:

|          |
|----------|
| in0 → a  |
| in1 → b  |
| sl → sel |
| y → y0   |

En el conexionado por nombre, utilizado en mux1 en el ejemplo anterior, se debe especificar, además de la señal, el puerto al que va conectado siguiendo la siguiente sintaxis:

```
.(puerto) señal
```

## 9.2. TIPOS DE ESTRUCTURAS DE MODULOS

Un módulo puede tener estructura compacta, todo está definido en la propia declaración del módulo:

```
module blink4(input wire clk, output wire [3:0] data);
```

También se podría haber hecho así:

```
module blink4(input clk, output [3:0] data);
    wire clk;
    wire data;
```

o así:

```
module blink4(clk, data);
    input clk;
    output data;
    wire clk;
    wire [3:0] data;
```

Esta última forma se utiliza para definir componentes paramétricos.

## 10. ESTRUCTURAS DE CONTROL

En este apartado se presentan las estructuras más comunes para la descripción de comportamiento en Verilog. A continuación se presentan cada una de ellas por separado indicando, además de un ejemplo, si son sintetizables o no.

### 10.1. SENTENCIAS CONDICIONALES IF – ELSE

La sentencia condicional if – else controla la ejecución de otras sentencias y/o asignaciones (estas últimas siempre procedurales). El uso de múltiples sentencias o asignaciones requiere el uso de begin – end. La sintaxis de la expresión es:

```
if (condición) //Condición simple
    sentencias;
```

```
if (condición) //Condición doble sentencias;  
else  
    sentencias;
```

```
if (condición1) // Múltiples condiciones sentencias;  
else if (condición2)  
    sentencias;  
....  
else  
    sentencias;
```

## 10.2. SENTENCIA CASE

La sentencia case evalúa una expresión y en función de su valor ejecuta la sentencia o grupos de sentencias agrupadas en el primer caso en que coincida. El uso de múltiples sentencias o asignaciones requiere el uso de **begin – end**. En caso de no cubrir todos los posibles valores de la expresión a evaluar, es necesario el uso de un caso por defecto (default). Este caso se ejecutará siempre y cuando no se cumplan ninguno de los casos anteriores. La sintaxis de la expresión es:

```
case (expresión)  
    <caso1>: sentencias;  
    <caso2>: begin  
        sentencias;  
        sentencias;  
    end  
    <caso3>: sentencias;  
    ....  
    ....  
    <default>: sentencias;  
endcase
```

## 10.3. SENTENCIA CASEZ Y CASEX

Estas sentencias corresponden a versiones especiales del case y cuya particularidad radica en que los valores lógicos z y x se tratan como valores indiferentes.

**casez:** usa el valor lógico z como valor indiferente

**casex:** toma como indiferentes tanto el valor z como el valor lógico x

A continuación se muestra de un ejemplo del uso del casez.

**casez (opcode)**

```
4'b1zzz: out = a; // Esta sentencia se ejecuta siempre que el bit más significativo sea 1.  
4'b01?: out = b; // El símbolo ? siempre se considera como “indiferente”, luego en este  
caso es equivalente a poner 4'b01zz  
4'b001?: out = c;  
default: $display("Error en el opcode");  
endcase
```

## 10.4. SENTENCIAS DE BUCLE

Todas las sentencias de bucles en Verilog deben estar contenidas en bloques procedurales (**initial o always**). Verilog soporta cuatro sentencias de bucles.

### **SENTENCIA FOREVER**

El bucle **forever** se ejecuta de forma continua, sin condición de finalización. Su sintaxis es:

```
forever <sentencias>
```

En caso de englobar más de una sentencia o asignación, éstas se deben incluir entre **begin – end**.

```
initial  
begin  
clk = 0;  
forever #5 clk = !clk;  
end
```

### **SENTENCIA REPEAT**

El bucle **repeat** se ejecuta un determinado número de veces, siendo este número su condición de finalización. Su sintaxis es:

```
repeat (<número>) <sentencias>
```

En caso de englobar más de una sentencia o asignación, éstas se deben incluir entre **begin – end**.

```
if (opcode == 10) //Realización de una operación de rotación  
repeat (8) begin  
temp = data[7];  
data = {data <<1,temp};  
end
```

## **SENTENCIA WHILE**

El bucle **while** se ejecuta mientras la condición que evalúa sea cierta. La condición que se especifica expresa la condición de ejecución del bucle. Su sintaxis es:

**while (<expresión>) <sentencias>**

En caso de englobar más de una sentencia o asignación, éstas se deben incluir entre **begin – end**.

```
loc = 0;  
if (data == 0) //Ejemplo de cálculo del bit más significativo loc = 32;  
else while (data[0] == 1'b0) begin  
    loc = loc + 1;  
    data = data >> 1;  
end
```

## **BUCLE FOR**

El bucle **for** es similar a aquellos utilizados en los lenguajes de programación. Su sintaxis es:

**for (<valor inicial>,<expresión>,<incremento>) <sentencias>**

En caso de englobar más de una sentencia o asignación, éstas se deben incluir entre **begin – end**.

```
for (i=0, i<64, i=i+1) //Inicialización de memoria RAM  
ram[i] = 0;
```

# **11. FUNCIONES DEL SISTEMA**

Existen acciones de bajo nivel disponibles denominadas funciones de sistema, suelen variar de una implementación a otra. Enumeraremos varias funciones más usas:

## **\$FINISH**

Si no se indica lo contrario la simulación es indefinida, como esto no suele ser deseable, esta función indica el final de la simulación.

## **\$TIME**

Esta función retorna el instante de tiempo en el que se encuentra la simulación.

## **\$RANDOM**

Esta función retorna un valor aleatorio entero de 32 bits cada vez que se invoca.

Se le puede suministrar como argumento una variable con la semilla que controla la generación de la misma secuencia aleatoria en el caso de repetir la simulación. Esto es, los valores obtenidos de la invocación repetida de esta función son aleatorios entre sí, pero será la misma secuencia si se ejecuta de nuevo.

### \$display y \$write

```
$display( P1, P2, P3, ...);  
$write( P1, P2, P3, ...);
```

Estas dos funciones de sistema permiten imprimir por la salida estándar, mensajes y variables del sistema. Ambas tienen una funcionalidad similar salvo porque **\$display** coloca un salto de línea al final.

Si el argumento es una variable se imprime (en formato decimal), si es una cadena se imprime tal cual salvo que tenga caracteres de escape:

TABLA: CARACTERES DE ESCAPE

| CARÁCTER | CARACTERÍSTICA |
|----------|----------------|
| \n       | Salto de línea |
| \t       | Tabulador      |
| \\       | Carácter \     |
| %%       | Carácter %     |

O indicadores de formato:

TABLA: INDICADORES

| CARÁCTER | CARACTERÍSTICA            |
|----------|---------------------------|
| %d       | formato decimal (defecto) |
| %h       | formato hexadecimal       |
| %b       | formato binario           |
| %o       | formato octal             |
| %t       | formato tiempo            |
| %c       | carácter ASCII            |

En este caso, a continuación de la cadena, deben de aparecer tantos argumentos como formatos se especifiquen.

Por omisión el tamaño del formato es el máximo de la variable correspondiente, pudiendo ajustarse al tamaño actual de la variable colocando un cero después del %.

Así por ejemplo:

```
reg [15:0] A;
initial begin
    A=10;
    $display ("%b %0b %d %0d", A, A, A, A);
end
```

Imprimiría: → 0000000000001010 1010 22210 10

#### **\$FDISPLAY Y \$FWRITE**

```
$fdisplay( fd, P1, P2, P3, ...);
$fwrite( fd, P1, P2, P3, ...);
```

Estos dos comandos son similares a los del apartado anterior salvo que permiten almacenar el resultado en un fichero, cuyo descriptor (fd) se le da como primer argumento. Las funciones para manipular la apertura y cierre del fichero son:

| Declaración   | Ejemplo   |
|---|---|
| <code>fd=\$fopen( nombre del fichero);<br/>\$fclose( fd );</code> | <code>integer fd;<br/>initial fd=\$fopen("resultados.dat");<br/>initial begin for(i=0;i&lt;100;i=i+1) begin<br/>#(100) \$fdisplay(fd,"%d %h",A,B);<br/>end<br/>\$fclose(fd);<br/>\$finish;<br/>end</code> |

#### **\$MONITOR Y \$FMONITOR**

```
$monitor( P1, P2, P3, ...);
$fmonitor( fd, P1, P2, P3, ...);
```

Estas funciones permiten la monitorización continua: se ejecutan una única vez, registrando aquellas variables que deseamos ver su cambio. De forma que cada vez que se produzca un cambio en una variable registrada, se imprimirá la línea completa.

#### **\$MONITOROFF:**

Esta función detiene la monitorización de variables que se ejecuta debido al comando `$monitor`. Ejemplo:

```
$monitoroff;
```

#### **\$MONITORON:**

Esta función habilita la monitorización de variables deshabilitada al ejecutar `$monitoroff`. Ejemplo:

```
$monitoron;
```

## \$DUMPFILE Y \$DUMPVARS

---

Un formato de almacenamiento de los datos de una simulación Verilog es el VCD (Verilog Change Dump). Este formato se caracteriza porque se almacena el valor de un conjunto de variables cada vez que se produce un cambio. Suele ser empleado por los visualizadores y post-procesadores de resultados.

Para crear este fichero son necesarias dos acciones: abrir el fichero donde se almacenaran los resultados y decir que variables se desea almacenar.

```
Initial
begin
$dumpfile("file1.vcd");
$dumpvars;
End
```

En el ejemplo anterior se desea crear un fichero cuyo nombre es file1.vcd, donde se desean almacenar todas las variables del diseño (que es lo que se almacena en el caso de no proporcionar parámetros).

Pueden restringirse las variables almacenadas empleando:

```
$dumpvars(levels,name1,name2,...)
```

Siendo levels el número de niveles de profundidad en la jerarquía que se desean almacenar, y name1, name2... las partes del diseño que se desean almacenar.

Si el primer argumento es 0, significa que se desean todas las variables existentes por debajo de las partes indicadas. Un 1 significa las variables en esa parte, pero no el contenido de módulos instanciados en su interior.

## \$READMEMB Y \$READMEMH

---

Permite leer la información contenida en un fichero en una memoria.

```
$readmemh(fname,array,start index,stop index);
$readmemb(fname,array,start index,stop index);
```

Donde fname: es el nombre de un fichero que contiene:

Datos binarios (\$readmemb) o hexadecimales (\$readmemh) (sin especificadores de tamaño o base). Pudiendo emplearse, x o z separadores: espacios, tabulaciones, retornos de carro. La información se debe de almacenar en una memory array, como por ejemplo:

```
reg [7:0] mem [0:1023];
```

El resto de argumentos son opcionales:

Ejemplo

```
initial $readmemh(file.dat,mem,0);
initial $readmemh(file.dat,mem,5);
initial $readmemh(file.dat,mem,511,0);
```

En el primer ejemplo se comienza a llenar desde el índice 0, en el segundo se comienza desde el índice 5. En el último caso se comienza por 511 y descendiendo hasta 0.

Si el número de datos del fichero es diferente al tamaño reservado o especificado para llenar el simulador da un error.

### **\$FOPEN**

---

Esta función permite abrir un fichero, devolviendo, sobre una variable definida como **integer** el identificador del fichero (file\_id). Ejemplo:

```
id = $fopen("hola.txt"); // "id" debe estar definido como integer. Se crea el fichero "hola.txt"
```

### **\$FCLOSE**

---

Esta función permite cerrar un fichero. El fichero a cerrar se especifica mediante el identificador. Ejemplo:

```
$fclose(id)
```

## **12. DIRECTIVAS PARA EL COMPILADOR**

Verilog ofrece un conjunto de directivas de compilación que permiten obtener diferentes códigos a partir de una única descripción. A continuación se detallan las más comunes. La sintaxis para la definición de una directiva es la siguiente.

```
'directiva <nombre> <valor>
```

### **12.1. DEFINE**

---

La directiva **define** permite definir un valor.

```
'define TD 1
.....
assign #TD data = A;
```

### **12.2. INCLUDE**

---

La directiva **include**, como su nombre indica permite incluir un fichero, el cual puede contener, por ejemplo, la definición de otros módulos.

```
'include "sumador.v"
```

### 12.3. IFDEF

La directiva **ifdef** permite compilar un código siempre y cuando se haya definido el símbolo al que referencia.

```
'define SIMULATION
.....
always @(posedge clk or posedge rst) if(rst)
data <= 0;
else
'ifdef SIMULATION data <= A;

'else
data <= B;
'endif
```

### 12.4. TIMESCALE

La directiva timescale permite definir las unidades de tiempo con las que se va a trabajar. La sintaxis de esta directiva es la siguiente. Donde la unidad de tiempo es mayor o igual a la resolución y los valores que pueden tomar los enteros son 1, 10 y 100 y la unidad de medida puede ser: "s", "ms", "us", "ns", "ps" y "fs".

```
'timescale <unidad de tiempo> / <resolucion>
```

Un ejemplo de esta directiva es la siguiente.

```
'timescale 1ns/100ps
```

En el ejemplo, la unidad de tiempo empleada es el nanosegundo y cualquier retraso fraccionario se redondeará al primer decimal debido a la resolución especificada.

## 13. PROCESOS DE VERILOG

El concepto de procesos que se ejecutan en paralelo es una de las características fundamentales del lenguaje, siendo ese uno de los aspectos diferenciales con respecto al lenguaje procedural como el lenguaje C.

Toda descripción de comportamiento en lenguaje Verilog debe declararse dentro de un proceso, aunque existe una excepción que trataremos a lo largo de este apartado. Existen en Verilog dos tipos de procesos, también denominados **bloques concurrentes**.

## **INITIAL**

Este tipo de proceso se ejecuta una sola vez comenzando su ejecución en tiempo cero. Este proceso NO ES SINTETIZABLE, es decir no se puede utilizar en una descripción RTL. Su uso está íntimamente ligado a la realización del **testbecnh**.

## **ALWAYS**

Este tipo de proceso se ejecuta continuamente a modo de bucle. Tal y como su nombre indica, se ejecuta siempre. Este proceso es totalmente sintetizable. La ejecución de este proceso está controlada por una temporización (es decir, se ejecuta cada determinado tiempo) o por eventos. En este último caso, si el bloque se ejecuta por más de un evento, al conjunto de eventos se denomina lista sensible. La sintaxis de este proceso es:

**always <temporización> o <@(lista sensible)>**

En la siguiente figura se muestran dos ejemplos de utilización de los procesos **initial** y **always**. De estos ejemplos se pueden apuntar las siguientes anotaciones:

```
initial
begin
    clk = 0;
    reset = 0;
    enable = 0;
    data = 0;
end
```

```
always @ (a or b or sel)
begin //Sobra en este caso
    if (sel == 1)
        y = a;
    else
        y = b;
end //Sobra en este caso
```

Figura 13.1 Procesos *initial* y *always*.

Fuente: <http://www.iuma.ulpgc.es/~nunez/clases-FdC/verilog/Verilog%20Tutorial%20v1.pdf>

- **Begin, end:** Si el proceso engloba más de una asignación procedural (=) o más de una estructura de control (if-else, case, for, etc...), estas deben estar contenidas en un bloque delimitado por **begin** y **end**.
- **Initial:** Se ejecuta a partir del instante cero y, en el ejemplo, en tiempo 0 (no hay elementos de retardo ni eventos, ya los trataremos), si bien las asignaciones contenidas entre **begin** y **end** se ejecutan de forma secuencial comenzando por la primera. En caso de existir varios bloques **initial** todos ellos se ejecutan de forma concurrente a partir del instante inicial.
- **Always:** En el ejemplo, se ejecuta cada vez que se produzcan los eventos variación de la variable a o variación de b o variación de **sel** (estos tres eventos conforman su lista de sensibilidad) y en tiempo 0. En el ejemplo, el proceso **always** sólo contiene una estructura de control por lo que los delimitadores **begin** y **end** pueden suprimirse.

Todas las asignaciones que se realizan dentro de un proceso **initial** o **always** se deben de realizar sobre variables tipo **reg** y NUNCA sobre nodos tipo **wire**. En la siguiente figura se muestra un ejemplo de asignación errónea sobre nodos tipo **wire** en un proceso **initial**.

```
wire      clk,reset;
reg       enable,data;
initial
begin
  clk = 0;           //Error
  reset = 0;         //Error
  enable = 0;
  data = 0;
end
```

```
reg      clk,reset;
reg      enable,data;
initial
begin
  clk = 0;
  reset = 0;
  enable = 0;
  data = 0;
end
```

Figura 13.2: Error de asignación de valores a variables tipo wire. B) Código corregido.

En general, la asignación dentro de un proceso **initial** o **always** tiene la siguiente sintaxis:

**variable = f(wire,reg,constante numérica)**

La variable puede ser tanto una variable interna como una señal del interfaz del módulo. La asignación puede ser de un nodo tipo **wire**, de una variable tipo **reg**, de una constante numérica o una función de todas ellas.

### 13.1. EVENTOS DE NIVEL:

Se distinguen dos tipos de eventos:

#### Eventos de nivel

Este evento se produce por el cambio de valor de una variable simple o de una lista sensible. Ejemplos:

**TABLA EVENTO DE NIVEL**

| Evento   | Descripción  |
|--|--|
| <b>Always @(x)<br/>Z &lt;= x + y</b>               | Cada vez que el valor de la señal “x” Cambia, el proceso se evalúa.  |
| <b>Always @(x or y or z)<br/>T &lt;= x + y + z</b> | Cada vez que el valor de la señal “x” o “y” o “z” Cambia el proceso se evalúa. En este caso x, y, y z conforman la lista sensible. |

#### Eventos de flanco:

Se producen por la combinación de flancos de subida y/o bajada de una señal simple o de una lista sensitiva.

TABLA EVENTOS DE FLANCOS

| Evento  | Descripción   |
|---|---|
| <b>Always @(posedge clk or posedge irt)</b><br><b>Z &lt;= z + y</b> | Cada vez que se produce un flanco de subida de clk o de irt, el proceso se evalúa                     |
| <b>Always @(posedge clk or negedge irt)</b><br><b>Z &lt;= z + y</b> | Cada vez que se produce un flanco de subida de clk o un flanco de bajada de irt, el proceso se evalúa |

## 14. TESTBENCH

Un componente descrito en Verilog **no se puede simular directamente**. Es necesario escribir un “**banco de pruebas**” o **testbench** que indique qué cables conectar a sus pines, qué valores de prueba enviar y comprobar que por sus salidas salen resultados correctos. Este banco de pruebas es un fichero también en Verilog.

El propósito de un **testbench** no es otro que verificar el correcto funcionamiento de un Módulo /diseño. La escritura de un **testbench** es casi tan compleja como la realización en RTL del módulo a verificar, a partir de ahora la denominaremos DUT (*Desing Under Test*). Una de las ventajas que presenta la escritura de un **testbench** es la posibilidad de no tener que ser sintetizable y, por tanto RTL.

Para escribir un **testbench** el diseñador debe tener siempre presente las especificaciones de diseño, en la que quedan reflejadas las funciones del diseño y, por tanto, las funciones a verificar.

### 14.1. ESTRUCTURA DE UN TESTBENCH

La estructura de un **testbench** es la que se refleja a continuación:

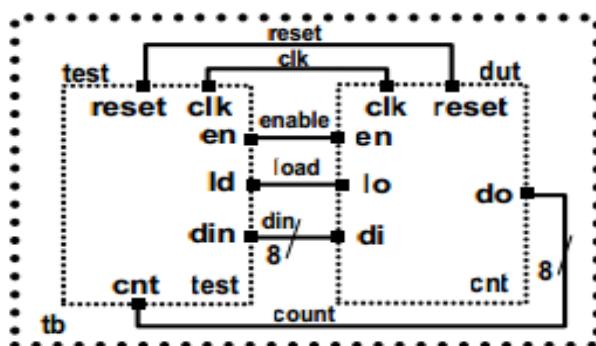


Figura 14.1: Estructura de un Test-nench

Fuente: <http://www.iuma.ulpgc.es/~nunez/clases-FdC/verilog/Verilog%20Tutorial%20v1.pdf>

```
module tb;
//nodos internas
wire [7:0] din, count;
wire reset,clk,enable,load;
//Dut
cnt dut(.clk (clk),
          .reset (reset),
          .di (din),
          .en (enable),
          .lo (load),
          .do (count));
//Test
test test(.clk (clk),
          .reset (reset),
          .en (enable),
          .din (din),
          .ld (load),
          .cnt (count));
initial
begin
    $shm_open("waves.shm");
    $shm_probe(tb,"AS");
end
endmodule
```

Se compone de:

- **Módulo dut:** Diseño a verificar
- **Módulo test:** Módulo generador/analizador. Este módulo es el encargado de generar las señales de entrada al módulo DUT y de analizar sus salidas. Su realización se hace a nivel de comportamiento (behavioral) y no necesariamente utilizando código RTL. Tal y como se indicó anteriormente, la realización de este módulo implica el conocer en detalle el diseño a verificar.
- **Módulo tb:** Este módulo incluye los módulos anteriores. Se caracteriza por no tener entradas ni salidas. Corresponde a una declaración estructural del conexionado de los módulos dut y test. En el ejemplo de la se ha incluido un proceso (initial) cuya única finalidad es la de abrir una base de datos con las formas de ondas del testbench.

## 14.2. MÓDULO TEST

El módulo test será el encargado de proporcionar los estímulos de entrada al **dut** y de analizar sus salidas. Su realización se hace a nivel de comportamiento (behavioral) y no

necesariamente utilizando código RTL. Tal y como se indicó anteriormente, la realización de este módulo implica el conocer en detalle el diseño a verificar. En este apartado se realizará la verificación del contador, denominado *cnt*, que se muestra en la figura anterior, por lo que en primer lugar se ha de conocer las especificaciones de este módulo.

Las especificaciones del módulo *cnt* son:

- Funcionamiento general. Se trata de un contador de 8 bits con señal de habilitación de contaje (enable) y señal de carga (load). El contador no es cíclico, por lo que al llegar al valor 8'HFF debe detenerse.
- Señal de reset. Cuando se activa se produce el reset asíncrono del contador a 0.
- Señal de enable. El contador realiza el contaje cuando la señal de enable está activa. En caso contrario el contaje queda detenido.
- Señal de load. La señal de load habilita la carga del contaje inicial, presente en din. Esta señal no tiene efecto si el contaje está habilitado. Una vez cargado, el contador debe comenzar el contaje con el valor cargado.

### ***Interfaz de entrada/salida***

---

La descripción del módulo de **test** comienza con la definición de las señales de interfaz. Éstas deben ser iguales, pero de sentido contrario, a las señales de interfaz del módulo **dut**. A continuación se muestra la descripción del interfaz del módulo test para nuestro ejemplo.

```
module test(clk,reset,en,lo,di,do);
    //Señales globales
    input clk,reset;
    //Entradas
    input [7:0] di;
    input en;
    input lo;
    //Salidas
    output [7:0] do;
    reg [7:0] do;
    ...
endmodule
```

### ***5.2.2 Generación de estímulos***

---

Para la generación de estímulos se utilizan los procesos **initial** o **always**. La elección del tipo de proceso viene determinada por las características de las señales a generar. Para aclarar este punto veamos la generación de diferentes señales. Dependiendo de la naturaleza de la señal, éstas se pueden generar de forma síncrona o asíncrona. La generación asíncrona se basa en el uso de retardos para activar y desactivar las señales. La generación síncrona se basa en el uso de eventos, disparados por flancos (normalmente

de la señal de reloj) para activar y desactivar las señales. En este último caso hay que hacer notar un detalle muy importante. **Todas las señales que se activen de forma síncrona se harán con un retardo.**

- Señal de reset (asíncrona). La siguiente figura muestra la generación de una señal de reset de forma asíncrona. Esta señal se inicializa a '0' en tiempo 0. Despues de 35 unidades de tiempo se activa a '1' y tras 50 unidades de tiempo, respecto al último evento, se pone de nuevo a '0'.

```
initial
begin
    reset = 1'b0;
    #35
    reset = 1'b1;
    #50
    reset = 1'b0
end
```

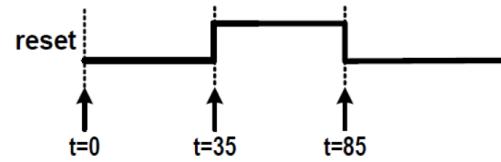


Figura 14.2: Generación de la señal de reset (asíncrona).

Fuente: <http://www.iuma.ulpgc.es/~nunez/clases-FdC/verilog/Verilog%20Tutorial%20v1.pdf>

- Señal de reset (síncrona). A continuación se muestra la generación de la señal de reset de forma síncrona. Hay que destacar que la activación/desactivación de esta señal se controla mediante eventos de, en este caso, de la señal de reloj. Tras su inicialización, se activa después de dos flancos de reloj y con un retardo de una unidad de tiempo. Se desactiva al siguiente flanco de reloj.

```
initial
begin
    reset = 1'b0;
    repeat(2)
        @(posedge clk) #1;
        reset = 1'b1;
        @(posedge clk) #1;
        reset = 1'b0;
end
```

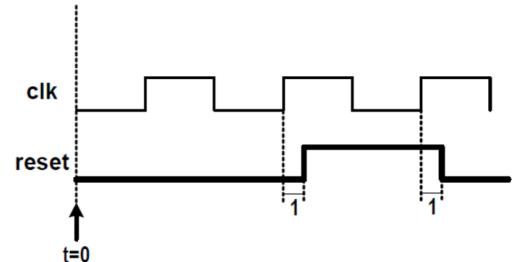


Figura 14.3: Generación de la señal de reset (síncrona).

Fuente: <http://www.iuma.ulpgc.es/~nunez/clases-FdC/verilog/Verilog%20Tutorial%20v1.pdf>

- Señal de reloj. La señal de reloj, por sus características, se genera usando otro tipo de procesos, generalmente always. La muestra la generación de una señal de reloj de 5 unidades de tiempo de semiperiodo. Hacer notar que para que la señal de reloj se genere de forma correcta hay que inicializar la variable clk a cero.

```

`define      SPER 5
...
always #`SPER clk = !clk;
initial
begin
  clk = 1'b0
end

```

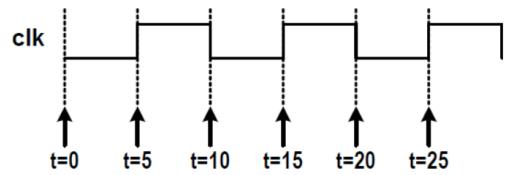


Figura 14.4: Generación de la señal de reloj  
Fuente: <http://www.iuma.ulpgc.es/~nunez/clases-FdC/verilog/Verilog%20Tutorial%20v1.pdf>

- Señales de control/datos. Al igual que ocurría con el reset, estas señales pueden ser asíncronas o síncronas. En este último caso habrá que tener en cuenta la señal de reloj para generarlas.

Hay que destacar como en el módulo de test presentado existen dos procesos que se ejecutan en paralelo. El primero de ellos corresponde al **always** para la generación de la señal de reloj. El segundo corresponde al bloque **initial**. Debido a la existencia de un bloque **always** y para poder parar la simulación se hace necesario el uso del comando **\$finish** que, al ejecutarse, para la simulación.

Ha continuación se muestra un test para el contador que estamos desarrollando.

```

//Señal de reloj
always #`SPER clk = !clk;
//Reset, señales de control y datos
initial
begin
  reset    = 1'b0;
  #35
  reset    = 1'b1;
  en       = 1'b0;
  ld       = 1'b0;
  din      = 8'b0;
  #50
  reset    = 1'b0
//Enable count
  @(posedge clk) #1;
  en       = 1'b1;
//Dejar 30 ciclos contando
  repeat(30)
    @(posedge clk) #1;
//Deshabilitar conteo durante 10 ciclos
  en       = 1'b0;
  repeat(10)
    @(posedge clk) #1;
//Volvemos a habilitar conteo
  @(posedge clk) #1;
  en       = 1'b1;

```

```
//Habilitamos carga de nuevo dato
@(posedge clk) #1;
ld      = 1'b1;
din    = 8'h4A;
@(posedge clk) #1;
ld      = 1'b0;
//Esperamos 10 ciclos de reloj y repetimos la operación
//deshabilitando el contaje
@(posedge clk) #1;
en      = 1'b0;
ld      = 1'b1;
din    = 8'hF0;
@(posedge clk) #1;
en      = 1'b1;
//Habilitamos 30 ciclos de reloj y finalizamos
repeat(30)
@(posedge clk) #1;
$finish;
end
```

## 15. TAREAS EN VERILOG

Las tareas son usadas en la mayor parte de los lenguajes de programación, conocidas comúnmente como procedimientos o subrutinas. Las tareas tienen la finalidad de reducir el código del diseño y sus llamadas se realizan en tiempo de compilación. Las tareas ofrecen la posibilidad de permitir parámetros de entrada y/o salida usando para ello nodos declarados en el módulo que realiza las llamadas. Las tareas:

- Se definen en el módulo en el que se utilizan, si bien pueden estar definidas en un fichero aparte y ser incluidas mediante la directiva **include** de Verilog.
- Pueden incluir retardos del tipo **#delay**, **posedge** o **negedge**.
- Pueden tener cualquier número de entradas/salidas.
- La definición de entradas y/o salidas marcan el orden en que éstas deben pasarse a la tarea.
- Las variables que se declaran dentro de una tarea son locales a dicha tarea.
- Las tareas pueden usar y/o asignar valores a cualquier señal declarada como global en el diseño.
- Una tarea puede llamar a otra tarea o función.
- Pueden utilizarse para modelar tanto lógica combinacional como secuencial.
- La llamada a una tarea no se puede realizar dentro de una expresión.

La sintaxis y la llamada de una tarea es la que se muestra a continuación.

| Descripción de una tarea   | Ejemplo: de una tarea   | Ejemplo: llamada a una tarea   |
|--|---|--|
| <pre>task &lt;name&gt; //Nombre de la tarea inputs; //Entradas outputs; //Salidas &lt;internal&gt;; //Variables internas begin     código de la tarea; end endtask</pre> | <pre>task logic_oper; input [7:0] numa; input [7:0] numb; output [7:0] out_and; output [7:0] out_or; begin     out_and = numa &amp; numb;     out_or = numa   numb; end endtask</pre> | <pre>module operador(a,b); input [7:0] a,b; reg [7:0] aandb; reg [7:0] aorb; ... always @( a or b)     logia_oper(a,b,aandb,aorb); ... endmodule</pre> |

## 16. FUNCIONES EN VERILOG

Una función en Verilog se asemeja a una tarea con las siguientes salvedades:

- No pueden incluir elementos de retardo.
- Pueden tener cualquier número de entradas pero sólo una salida.
- Solo se pueden utilizar para modelar lógica combinacional.
- Una función puede llamar a otra función pero no a una tarea.

La sintaxis y la llamada de una función es la que se muestra a continuación.

| Descripción de una función  | Ejemplo: de una función   | Ejemplo: llamada función  |
|---|---|---|
| <pre>task &lt;name&gt; //Nombre de la tarea inputs; //Entradas &lt;internal&gt;; //Variables internas begin     código de la función; end endfunction</pre> | <pre>function logic_oper; input [7:0] numa; input [7:0] numb; begin     logic_oper = numa &amp; numb; end endfunction</pre> | <pre>module operador(a,b); input [7:0] a,b; wire [7:0] aandb; ... assign aandb = logia_oper(a,b); ... endmodule</pre> |

## 17. ALGEBRA BOOLEANA

### Buffer lógico:

Su salida (S) es exactamente el mismo valor que tiene en su entrada (A)

| Representación gráfica | Código Verilog  |
|------------------------|---|
|                        | <pre>module compigualdad (input a, output r); assign r= a; end module</pre> |

### Inversor lógico NOT:

Su salida (S) es el valor contrario al que tiene en su entrada (A)

| Representación gráfica | Código Verilog   |
|------------------------|--|
|                        | <pre>module compNOT (input a, output r); assign r= ~ a; end module</pre> |

### Puerta lógica OR:

entrega a su salida (S) un valor de 1 si uno de sus valores de entrada (A) o (B) es 1 o ambos son 1 y un valor de 0 si sus valores de entrada de (A) y (B) son 0

| Representación gráfica | Código Verilog  |
|------------------------|---|
|                        | <pre>module compOR (r,a,b); input a,b; output r; assign r= a   b; endmodule</pre> |

### Puerta lógica NOR:

tiene todas sus entradas (A,B) un valor de 0 y como salida (S) nos da un 1 lógico, es decir que cuando hay un valor en 1 lógico siempre va tener un 0 lógico en sus salidas.

| Representación gráfica | Código Verilog  |
|------------------------|---|
|                        | <pre>module compNOR (r,a,b); input a,b; output r; assign r= ~(a   b); endmodule</pre> |

### Puerta lógica AND:

envía un valor de 1 a su salida (S) cuando los valores de entrada de (A) y (B) son 1 en caso uno de los valores de (A) o (B) es cero su valor a la salida "S" es cero

| Representación gráfica | Código Verilog   |
|------------------------|--|
|                        | <pre>module compAND (r,a,b);   input a,b;   output r;   assign r= a &amp; b; endmodule</pre> |

### Puerta lógica NAND:

Envía un valor de 1 a su salida (S) cuando los valores de entrada de (A) y (B) son 0, si uno de los valores de (A) o (B) es 1 su valor a la salida "S" es 1.

| Representación gráfica | Código Verilog  |
|------------------------|---|
|                        | <pre>module compNAND (r,a,b);   input a,b;   output r;   assign r= ~ (a &amp; b); endmodule</pre> |

### Puerta lógica OR Exclusiva XOR:

disyunción exclusiva envía a su salida (S) un valor de 1 si exactamente uno de los valores de (A) o (B) es uno (pero no ambos) de dos condiciones

| Representación gráfica | Código Verilog   |
|------------------------|--|
|                        | <pre>module compXOR (r,a,b);   input a,b;   output r;   assign r= a ^ b; endmodule</pre> |

### Puerta lógica NOR Exclusiva XNOR:

debe tener en todas sus entradas (A,B) un igual valor sea de 1 o 0 y como salida (S) nos da un 1 lógico, es decir que cuando todas sus entradas son diferentes de 1 o 0 lógico siempre va tener un 0 lógico en sus salidas

| Representación gráfica | Código Verilog  |
|------------------------|---|
|                        | <pre>module compXNOR (r,a,b);   input a,b;   output r;   assign r= ~ (a ^ b); endmodule</pre> |

## 18. SISTEMAS COMBINACIONALES

### Medio SUMADOR:

En el medio sumador existen dos entradas (A,B), son los dígitos a sumar, dos salidas (R, C) que son la respuesta (R) y el acarreo (C), donde lo último tiene mayor peso

| Representación gráfica | Código Verilog  |
|------------------------|---|
|                        | <pre>module MedioS(input a,b, output r,c );   assign r= a ^ b;   assign c= a&amp;b; endmodule</pre> |

### SUMADOR COMPLETO:

El sumador completo es la unión de los dos medios sumadores aquí se suman 3 bits, los dos bits de entrada (A, B) se suman en un medio sumador, la respuesta (R) se vuelve a sumar en un medio sumador con la tercera entrada (C), los acarreos de los dos medios sumadores se los suma y se tiene un acarreo total

| Representación gráfica | Código Verilog  |
|------------------------|---|
|                        | <pre>module SumaC (input a,b,Cin output r,Cout);   assign r= (a ^ b ^ Cin);   assign Cout= ((a ^ b) &amp; Cin)   (a &amp; b); endmodule</pre> |

### MEDIO RESTADOR:

Un medio retador es un circuito combinacional que sustrae dos bits y produce su diferencia e indica si ha tomado un préstamo.

| Representación gráfica | Código Verilog   |
|------------------------|--|
|                        | <pre>module MedioR( input m,s, output r,p );   assign r= (~ m) &amp; s;   assign p= m ^ s; endmodule</pre> |

### RESTADOR COMPLETO:

Intervienen tres bits en esta operación de resta, esto es cuando consideramos a más de los bits a restar un bit de préstamo, es decir, se ha efectuado una operación de resta en una columna anterior que ha generado dicho bit, el circuito se convierte en un restador completo.

| Representación gráfica | Código Verilog   |
|------------------------|--|
|                        | <pre>module ResctaC( input a,b,cin , output s, cout ); assign s = b ^ a ^ cin; assign cout = ((~a)&amp;cin)   ((~a)&amp;b)  (b &amp; cin); endmodule</pre> |

### CODIFICADOR:

Un codificador es un circuito combinacional que permite pasar un conjunto de información sin codificar (dígitos decimales, octales, hexadecimales, etc) en un conjunto que responda a la estructura de un código (BCD, exs3, binario, etc).

El circuito que actué como codificador tendrá N salidas cuando se requiera codificar M entradas, teniendo un relación de  $2^N$  entradas lo que será igual a M salidas, de tal manera podemos decir que M informaciones diferentes se pueden representar mediante grupos de N bits.

### CODIFICADOR PARA DÍGITOS DE BASE OCHO

| Representación gráfica | Código Verilog   |
|------------------------|--|
| <br>                   | <pre>module Codificador( input a,b,c,d,e,f,g,h, output s0,s1,s2); assign s0 = e   f   g   h; assign s1 = c   d   g   h; assign s2 = b   d   f   h; endmodule</pre> |

### DECODIFICADOR:

Los decodificadores son circuitos que realizan la función inversa a los circuitos codificadores, es decir, toman la información codificada en n bits y generan la información original, para ellos usaremos el ejemplo hecho de codificador en el apartado anterior, para que dé binario decodifique a octal.

| Representación gráfica | Código Verilog   |
|------------------------|--|
| <br>                   | <pre>module Decodificador( input a,b,c, output s0,s1,s2,s3,s4,s5,s6,s7 ); assign s0=(~a)&amp;(~b)&amp;(~c); assign s1=(~a)&amp;(~b)&amp;c; assign s2=(~a)&amp;b&amp;(~c); assign s3=(a)&amp;b&amp;c; assign s4=a&amp;(~b)&amp;(~c); assign s5=a&amp;(~b)&amp;c; assign s6=a&amp;b&amp;(~c); assign s7=a&amp;b&amp;c; endmodule</pre> |

### MULTIPLEXOR:

Es un circuito combinacional el cual posee un conjunto de entradas tanto de datos como de control y un solo pin de salida. El trabajo que debe de realizar el multiplexor es el de pasar la información que se encuentra en la entrada de datos, donde los pines se irán activando de acuerdo al dato que se le envié en sus pines de control para que así la información pueda pasar desde su entrada a la salida única que posee este dispositivo.

En la siguiente grafica se mostrara de qué manera se representa un multiplexor de 4 x 1 con línea de activación.

| Representación gráfica | Código Verilog   |
|------------------------|--|
| <br>                   | <pre> module multiflexor(   input [1:0] c,   input [3:0] e,   output reg s);   always @ (c, e)     case (c)       2'b00: s = e[0];       2'b01: s = e[1];       2'b10: s = e[2];       default: s = e[3];     endcase endmodule </pre> |

## 19. BIBLIOGRAFÍA

IEEE, Verilog Grupo Verilog Normalización. <http://www.verilog.com>

NUÑEZ. Tutorial Verilog., Las Palmas-España. <http://www.iuma.ulpgc.es/~nunez/clases-FdC/verilog/Verilog%20Tutorial%20v1.pdf>

[http://es.wikibooks.org/wiki/Programaci%C3%B3n\\_en\\_Verilog/M%C3%B3dulos](http://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Verilog/M%C3%B3dulos).

MATPIC., Tutorial Verilog., 2017

<http://www.matpic.com/esp/vhdl/verilog.html>

<https://www.linuxito.com/windows/1088-primeros-pasos-con-icarus-verilog-en-windows>

---

## A07-MANUAL DE USO ICARUS VERILOG + GTKWAVE

---

## ÍNDICE DE CONTENIDOS

---

|                          |   |
|--------------------------|---|
| 1. INTRODUCCIÓN .....    | 3 |
| 2. INSTALACIÓN .....     | 3 |
| 3. UTILIZACIÓN .....     | 4 |
| 3.1. COMPILEAR .....     | 5 |
| 3.2. EJECUCIÓN .....     | 6 |
| 3.3. VISUALIZACIÓN ..... | 6 |
| 4. BIBLIOGRAFÍA .....    | 8 |

## ÍNDICE DE ILUSTRACIONES

---

|  |   |
|--|---|
| <i>Figura 2.1: comprobación instalación Icarus Verilog .....</i> | 4 |
| <i>Figura 3.1: detalle de la consola de Windows .....</i>        | 6 |
| <i>Figura 3.2 Ventana GTKWave .....</i>                          | 7 |
| <i>Figura 3.3: Señales en GtkWave .....</i>                      | 7 |

## 1. INTRODUCCIÓN

Los lenguajes HDL como Verilog son utilizados generalmente para programar FPGA (Field Programmable Gate Array). Estos dispositivos programables contienen bloques de lógica cuya interconexión y funcionalidad puede ser configurada mediante uno de estos lenguajes de descripción especializados (HDL). Se trata precisamente de programar hardware.

Cuando se trabaja con FPGAs se está realizando hardware y hay que tener cuidado. Se podría escribir un código que por ejemplo tuviera un cortocircuito. Y podría ocurrir que las herramientas de síntesis no avisen con un warning. Y al ser cargado en la FPGA, esta se podría estropear parcialmente.

Por ello, es preciso simular el código que se diseñe. Y una vez se comprueba que funciona correctamente es cuando se carga en la FPGA.

**Icarus Verilog** es un compilador de lenguaje Verilog desarrollado para GNU/Linux, pero que puede funcionar correctamente en Windows gracias a la pila MinGW (Minimal GNU for Windows). MinGW es el mismo compilador que utiliza, por ejemplo, el IDE para desarrollo en lenguaje C Code::Blocks. Ya sea en sus versiones para Linux como en sus versiones para Windows, Icarus Verilog es liberado bajo la licencia GNU GPL.

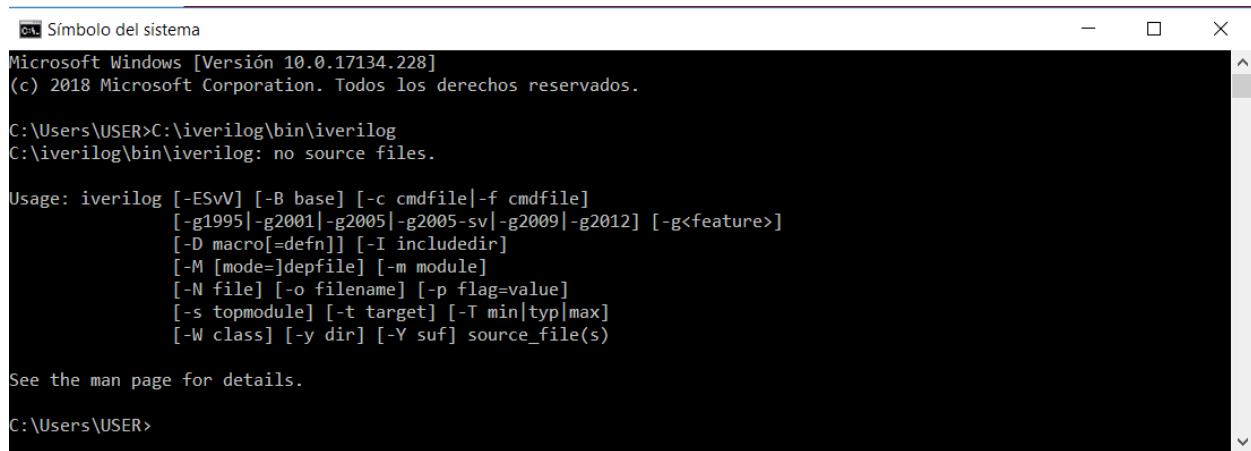
Es un compilador liviano e **incluye el simulador GTKWave** para verificar el funcionamiento de un diseño. A continuación se explica el proceso de instalación y configuración de Icarus Verilog en Windows, así como el uso básico del mismo a través de un programa (diseño) de prueba.

## 2. INSTALACIÓN

La instalación de Icarus Verilog y GtkWave es extremadamente sencilla, basta con bajarse la última versión del instalador desde [bleyer.org/icarus/](http://bleyer.org/icarus/). Al ejecutar el instalador debe aceptarse la licencia del programa (no hay problemas, es software libre) y utilizar todas las opciones que vienen por defecto. Para comprobar la instalación, simplemente abrir una consola de Windows y ejecutar **iverilog** en el directorio donde se haya instalado Icarus verilog:

```
C:\iverilog\bin\iverilog
```

Si la salida dice **"no source files"**, significa que ha sido instalado correctamente.



```
Símbolo del sistema
Microsoft Windows [Versión 10.0.17134.228]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Users\USER>C:\iverilog\bin\iverilog
C:\iverilog\bin\iverilog: no source files.

Usage: iverilog [-ESvV] [-B base] [-c cmdfile|-f cmdfile]
                [-g1995|-g2001|-g2005|-g2005-sv|-g2009|-g2012] [-g<feature>]
                [-D macro[=defn]] [-I includedir]
                [-M [mode=]depfile] [-m module]
                [-N file] [-o filename] [-p flag=value]
                [-s topmodule] [-t target] [-T min|typ|max]
                [-W class] [-y dir] [-Y suf] source_file(s)

See the man page for details.

C:\Users\USER>
```

Figura 2.1: comprobación instalación Icarus Verilog

### 3. UTILIZACIÓN

Una vez Icarus Verilog ha sido instalado correctamente, a continuación se diseña un “hola mundo” que servirá de ejemplo para explicar el uso de Icarus Verilog.

Los programas verilog se separan en módulos, un módulo en Verilog es un bloque funcional que posee entradas, salidas y una lógica interna. Pueden pensarse como bloques en un diagrama de bloques de un circuito. Existen dos tipos de módulos: de comportamiento y de estructura. Ambos tipos pueden tener el mismo funcionamiento pero difieren en la forma en que se escriben.

A continuación para mostrar el proceso de simulación utilizando Icarus Verilog se diseña un sencillo código que consiste en activar un bit, (en la FPGA, se representa encendiendo un LED). Para implementar y testear un “hola mundo”, utilizaremos el siguiente código:

```
//-- Fichero ActivaBit.v
module ActivaBit(output A);
wire A;

assign A = 1;

endmodule
```

Lo guardamos en el directorio de trabajo como **ActivaBit.v**

Por otro lado para poder simular, necesitamos implementar el módulo de testeo y el testbench (banco de pruebas)

Es necesario escribir un banco de pruebas que indique qué cables conectar a sus pines, qué valores de prueba enviar y comprobar que por sus salidas salen resultados correctos. Este banco de pruebas es un fichero también en Verilog.

El fichero se llamará **ActivaBit\_tb.v**. Se usa el sufijo \_tb para indicar que se trata de un banco de pruebas (TestBench). Este banco de pruebas NO ES SINTETIZABLE. Es un código que sólo vale para la SIMULACIÓN. Lo que sintetizamos es el componente **ActivaBit.v**.

El banco de pruebas tiene 3 elementos: el componente ActivaBit, un cable que hemos llamado A y un bloque de comprobación (que sería el equivalente al polímetro en el mundo real). El código sería el siguiente:

```
/**- Fichero ActivaBit_tb.v
/**- Modulo para el test bench
module setbit_tb;
wire A;
ActivaBit SB1 (
    .A (A)
);

/**- Comenzamos las pruebas (bloque de comprobacion)
initial begin

    /**- Definir el fichero donde volcar resultados simulacion
    $dumpfile("ActivaBit_tb.vcd");

    /**- Volcar todos los datos a ese fichero (al finalizar la simulacion)
    $dumpvars(0, ActivaBit_tb);

    /**- Pasadas 20 unidades de tiempo comprobamos si el cable está a '1'
    /**- En caso de no estar a 1, se informa del problema, pero la
    /**- simulacion no se detiene
    # 20 if (A != 1)
        $display("---->¡ERROR! Salida no esta a 1");
    else
        $display("Componente ok!");

    /**- Terminar la simulacion tras 10 unidades de tiempo desde la comprobacion
    # 10 $finish;
end
endmodule
```

### 3.1. COMPILEAR

---

Una vez se tienen los dos archivos diseñados; **ActivaBit.v**, que contiene el código a comprobar y **ActivaBit\_tb.v** que contiene el código necesario para la simulación, se procede a compilar el diseño, para ello abriremos la consola y cambiaremos al directorio donde se encuentran los archivos a compilar (utilizando el comando cd):

```
C:\Users\USER> cd Directorio_trabajo
```

Para compilar utilizaremos el siguiente comando:

```
C:\iverilog\bin\iverilog -o ActivaBit ActivaBit.v ActivaBit_tb.v
```

La opción –o indica que la salida sea al archivo **ActivaBit**. Este archivo no es ejecutable, sino que debe ejecutarse mediante el simulador **vvp**. Este es el simulador de Icarus Verilog, el

cual produce los resultados de la simulación en forma de ceros y unos para cada variable modelada, en función del tiempo. Esto es, precisamente, señales.

### 3.2. EJECUCIÓN

La simulación se ejecuta mediante el comando:

```
C:\iverilog\bin\vvp ActivaBit
```

Este proceso genera el archivo **ActivaBit\_tb.vcd**, el cual contiene los resultados de la simulación.

### 3.3. VISUALIZACIÓN

El formato de salida del archivo **ActivaBit\_tb.vcd** no es fácil de comprender. Sin embargo, es posible abrirlo con GTKWave para generar las gráficas de tiempo de cada señal. Para visualizar las señales se utiliza el comando gtkwave pasándole como parámetro el archivo vcd generado en el paso anterior.

```
C:\iverilog\gtkwave\bin\gtkwave ActivaBit_tb.vcd
```

The screenshot shows a Windows command prompt window. The text is annotated with red boxes and arrows to identify the steps:

- A red box surrounds the path `C:\Users\USER>cd verilog`. An arrow points from this box to a red box labeled "Cambio a directorio de trabajo".
- A red box surrounds the command `C:\Users\USER>C:\iverilog\bin\iverilog -o ActivaBit ActivaBit.v ActivaBit_tb.v`. An arrow points from this box to a red box labeled "Compilación".
- A red box surrounds the command `C:\Users\USER>C:\iverilog\bin\vvp ActivaBit`. An arrow points from this box to a red box labeled "Simulación".
- A red box surrounds the command `C:\Users\USER>C:\iverilog\gtkwave\bin\gtkwave ActivaBit_tb.vcd`. An arrow points from this box to a red box labeled "Visualización".

```
Símbolo del sistema - C:\iverilog\gtkwave\bin\gtkwave ActivaBit_tb.vcd
Microsoft Windows [Versión 10.0.17134.228]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Users\USER>cd verilog
Cambio a directorio de trabajo

C:\Users\USER>C:\iverilog\bin\iverilog -o ActivaBit ActivaBit.v ActivaBit_tb.v
Compilación

C:\Users\USER>C:\iverilog\bin\vvp ActivaBit
VCD info: dumpfile ActivaBit_tb.vcd opened for output.
Componente ok!

C:\Users\USER>C:\iverilog\gtkwave\bin\gtkwave ActivaBit_tb.vcd
Simulación

GTKWave Analyzer v3.3.71 (w)1999-2016 BSI
[0] start time.
[30] end time.

Visualización
```

Figura 3.1: detalle de la consola de Windows

Al ejecutar el comando gtkwave se abre la siguiente ventana

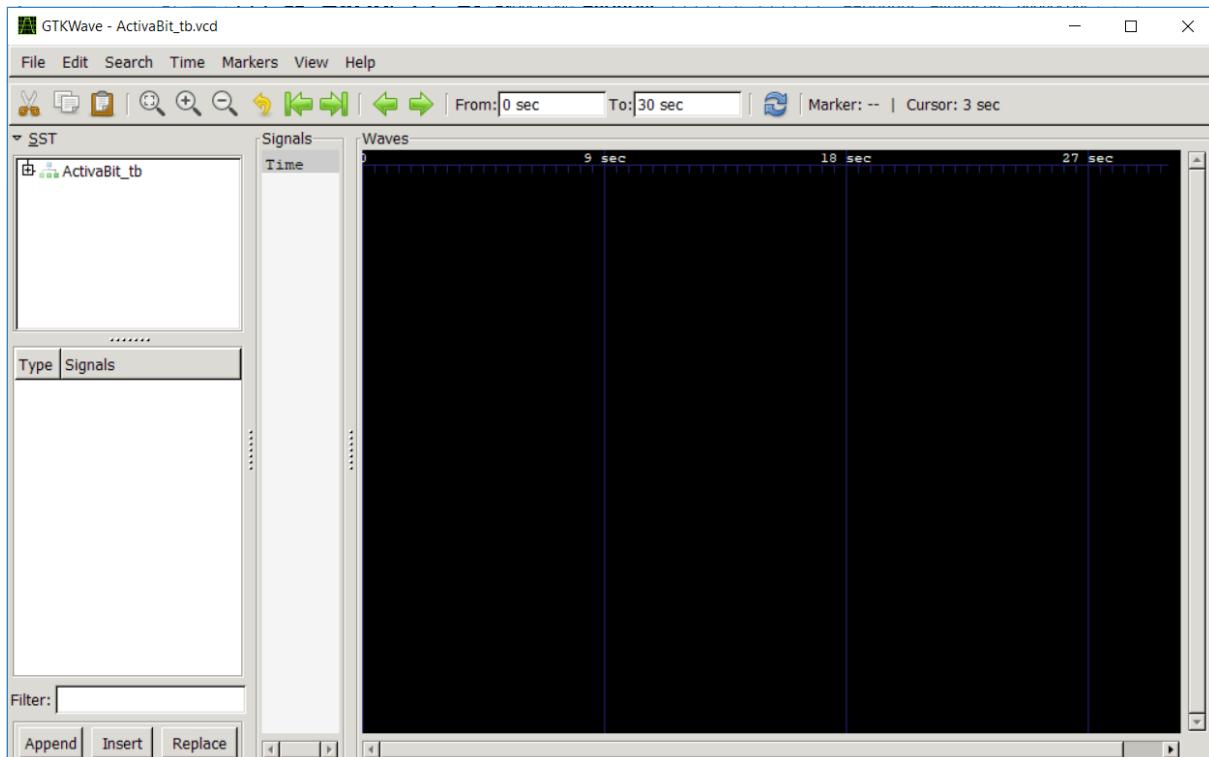


Figura 3.2 Ventana GTKWave

Una vez abierta es necesario agregar las variables para verlas en el diagrama de tiempo. Aparecerá a la derecha una jerarquía de módulos, se hace click en ActivaBit\_tb y abajo aparecerán las señales de ese modulo. Y las arrastramos a la caja inmediatamente a la derecha bajo la etiqueta Signals obteniendo una figura como la siguiente

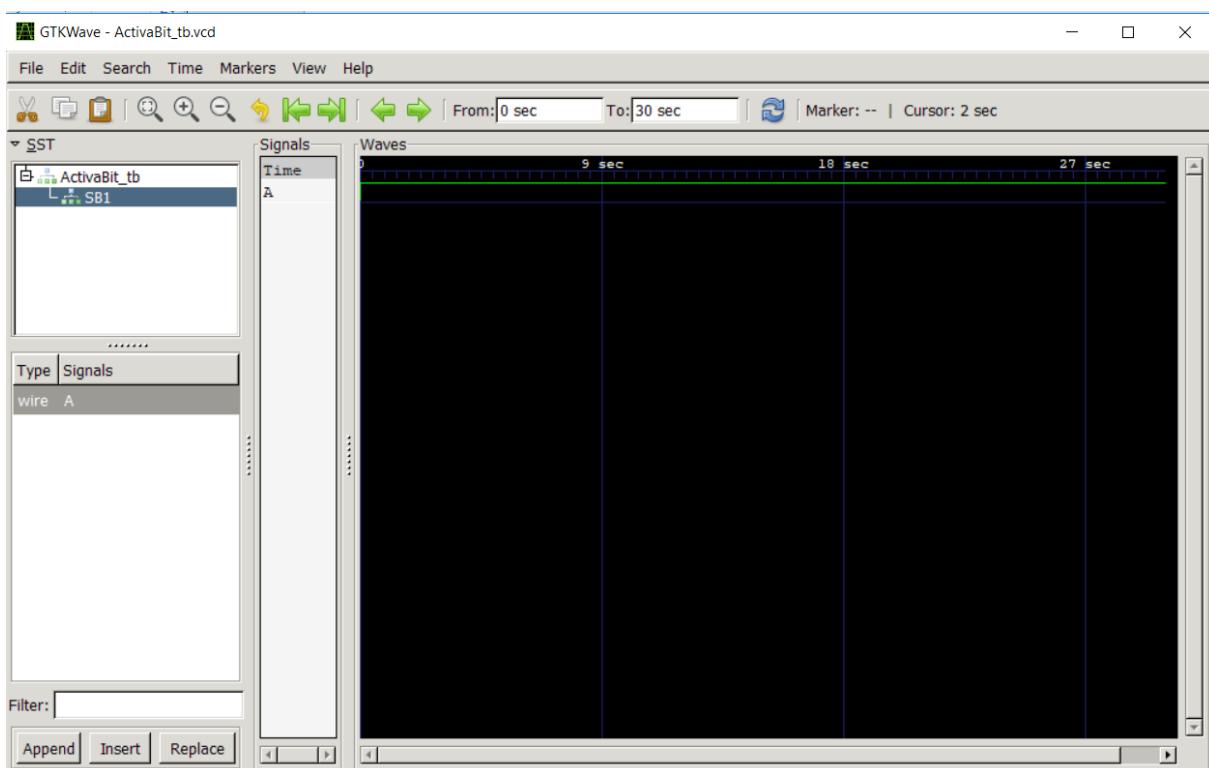


Figura 3.3: Señales en GtkWave

Se comprueba que se comporta como se esperaba: la señal del cable A está siempre a '1'. Se ignora las unidades de tiempo que pone, que por defecto está en segundos. Para nosotros serán unidades de tiempo. Tras 30 unidades, la simulación termina

Cuando se hace el testeo de un circuito, es necesario compilar-simular-graficar con cierta frecuencia. En este caso GTKWave tiene la opción de recargar las gráficas desde la salida de la simulación desde el menú "File > Reload Waveform". Esto permite refrescar los diagramas de temporizado de señales de forma rápida.

## 4. BIBLIOGRAFÍA

<https://github.com/Obijuan/open-fpga-verilog-tutorial/wiki>

<https://www.linuxito.com/windows/1088-primeros-pasos-con-icarus-verilog-en-windows>

Guía de Instalación y Uso de Icarus Verilog y GtkWave Profesor: Claudio Estévez Auxiliar: Luis Alberto Herrera April 15, 2011