E.T.S. de Ingeniería Industrial, Informática y de Telecomunicación

ENTORNO DE APRENDIZAJE DE DISPOSITIVOS DE LÓGICA PROGRAMABLE

Grado en Ingeniería
en Tecnologías Industriales

Trabajo Fin de Grado

Ismael Ezquerro Ezquerro

Francisco Javier Arregui San Martín

Pamplona, Jueves, 19 de Junio de 2014

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

INDICE

Capítulo 1.

Objetivos

Este proyecto se basa en el aprendizaje de sistemas digitales, concretamente, en la programación, mediante un lenguaje VHDL, de dispositivos de lógica programable, como son los FPGA. Para ello, se quieren alcanzar los siguientes objetivos:

- Introducción a los lenguajes de descripción de hardware (lenguajes HDL). Describir las principales características de estos lenguajes y realizar una comparación entre alguno de los lenguajes HDL más extendidos.

- Llevar a cabo un estudio comparativo entre universidades para escoger un lenguaje concreto y desarrollar un entorno de aprendizaje en torno a ese lenguaje de descripción de hardware y a la programación de dispositivos de lógica programable a través de él.

- Elaborar una serie de guiones de prácticas para el laboratorio, de modo que se afiancen los conceptos vistos en clase y se aprenda a realizar diseños de circuitos digitales mediante este lenguaje.

Los guiones de prácticas seguirán la misma estructura que el temario que se impartirá en las clases teóricas, de forma que los alumnos puedan intercalar las clases teóricas y las prácticas de laboratorio.

Los guiones de prácticas tendrán como objetivos:

- Puesta en práctica de los conocimientos que se han visto en las clases teóricas y conocer diversas formas de llevar a cabo el diseño de sistemas electrónicos digitales.

- Realización de una serie de diseños de circuitos digitales a través de dispositivos de lógica programable. Para ello se dispondrá de una tarjeta educacional con múltiples accesorios y en la que estará integrado un dispositivo de lógica programable.

- Manejo de un software de diseño enfocado, fundamentalmente, para realizar diseños con dispositivos de lógica programable.

Capítulo 2.

Introducción a los lenguajes HDL

2.1. Antecedentes

La aparición de los lenguajes de descripción de hardware se remonta a la década de los sesenta, cuando se produce una notable evolución en los procesos de fabricación de los circuitos integrados, que conlleva el desarrollo de los circuitos digitales hasta bien entrada la década de los ochenta. Durante aquel periodo el diseño se centraba en los niveles eléctricos para establecer parámetros y relaciones entre los componentes básicos al nivel de transistor. El método que se llevaba a cabo durante el diseño era básicamente manual, apenas se utilizaban alguna herramienta de simulación de esquemas eléctricos.

Conforme el tiempo pasaba, los procesos de diseño eran más complicados, cada vez aparecían más problemas de integración y de mantenimiento de los diseños. A finales de la década de los sesenta se llega a una situación de gran desfase entre tecnología y diseño. Las empresas están superadas por la situación, ya que la fabricación de circuitos de alta integración supone riesgos y costes imposibles de abordar por estas.

Una vez que se alcanza este punto, es cuando una serie de grupos comienzan a desarrollar los lenguajes de descripción de hardware. Se trataban de lenguajes que no irían dirigidos a la caracterización eléctrica de cada componente al nivel de transistor, si no que se centrarían en el funcionamiento lógico del sistema. Algunas empresas, como por ejemplo IBM o Texas Instruments, así como grupos de estudio en universidades, empezaron a buscar soluciones para el diseño de sistemas complejos. Sin embargo, estos lenguajes no se consolidaron por distintos motivos, como la falta de soporte o la falta de difusión.

Es entonces, a mitad de la década de los ochenta cuando aparecen los lenguajes VHDL y Verilog, que aprovechando el desarrollo de la tecnología que se había producido en los años anteriores consiguen consolidarse como herramientas fundamentales para el diseño y desarrollo de nuevos sistemas.

Actualmente, ambos lenguajes están normalizados y acaparan gran parte del mercado, apartando a los lenguajes gráficos o esquemáticos, que son soportados por herramientas de CAD. ***(REF. [1])***

2.2. Descripción de los lenguajes HDL

Los lenguajes HDL (Hardware Description Language) son herramientas de programación formal para describir el comportamiento y la estructura de sistemas usando un esquema textual.

Se trata de un lenguaje capaz de describir actividades que ocurren de forma simultánea y luego permitirnos describir módulos con acciones que serán evaluadas de forma secuencial. Además, nos permite la construcción de una estructura jerárquica, donde es posible combinar descripciones estructurales y de flujo de datos con descripciones de comportamiento, a su vez que posibilita algo tan fundamental en la descripción de sistemas electrónicos como es la modelación del concepto del tiempo.

2.2.1. Niveles de estilos (o de abstracción) *(REF. [2])*

El diseñador puede describir la operación del sistema mediante distintos niveles de estilos y estos se clasifican en:

- Comportamiento: describe qué es lo que el sistema debe hacer, detalla la función entrada-salida del diseño, sin profundizar la arquitectura o los registros empleados. Es el nivel de abstracción más alto.
- Algorítmico o funcional: describe cómo hacerlo, detalla las acciones a realizar para alcanzar los objetivos, como un algoritmo de software.
- Flujo de datos: describe la solución especificando los registros y la lógica que los une, pero sin incluir como solucionar esa lógica. Es una descripción de la arquitectura del sistema, pero no de la tecnología.
- Estructural: describe una red de compuertas y registros, añadiendo posibles esquemas de conexión. La descripción estructural es equivalente al diagrama esquemático el circuito. Es el menor nivel de abstracción.

Un diseño puede ser perfectamente descrito con una combinación de todos los estilos.

2.2.2. Ventajas de los HDLs *(REF. [1])*

- El lenguaje es totalmente independiente de la tecnología que se programe, el mismo modelo puede ser sintetizado en librerías de distintos vendedores. Esta independencia sobre la tecnología permite reutilizar el diseño en diferentes componentes, como pueden ser ASICs o FPGAs, con muy poco esfuerzo.
- Nos posibilita la opción de comprobar el funcionamiento del sistema durante el proceso de diseño sin tener que implementar el circuito físicamente. Esto nos permite probar la arquitectura del sistema para rectificar y realizar cambios en el diseño si fuesen necesarios.

- Resulta mucho más sencillo para una persona cualquiera comprender qué función realiza el diseño mediante HDL que mediante un esquemático basado en puertas. Por ejemplo una puerta and sería:

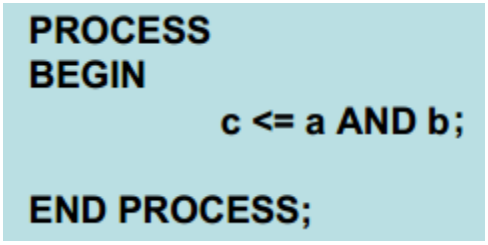

Figura 1: Ejemplo de puerta AND de dos entradas
(Imagen extraída de [3])

- Las herramientas de síntesis (sintetizadores RTL) tienen la capacidad de convertir una descripción realizada en HDL a puertas lógicas.
- No es necesario adecuar el circuito a cada dispositivo porque las herramientas de síntesis también se encargan de ello.
- Supone un ahorro de tiempo, ya que reduce el tiempo de diseño y facilita las correcciones del diseño debido a errores en el diseño o a cambios en las especificaciones.
- El lenguaje soporta jerarquía, un sistema puede ser modelado como un conjunto de componentes conectados entre sí y a su vez cada componente puede ser modelado como un conjunto de subcomponentes.
- Soporta metodologías de diseño distintas: top-down, bottom-up o mixtas.
- Soporta modelos de tiempos asíncronos y síncronos.
- El lenguaje es público.
- Estos lenguajes (VHDL y Verilog) son normas IEEE.
- No hay limitaciones impuestas por el lenguaje en cuanto a tamaño del diseño.

2.2.3. Inconvenientes *(REF. [1])*

- Supone un esfuerzo de aprendizaje, ya que se trata de un nuevo lenguaje y de una nueva metodología.
- Se necesitan nuevas herramientas como simuladores o sintetizadores de HDL.
- El diseño mediante estos lenguajes hace que se vaya perdiendo el enfoque físico del diseño a costa de aumentar la importancia sobre la funcionalidad del mismo.

En conclusión, hoy en día los lenguajes HDL están totalmente extendidos, están estandarizados y una vez que se adquiere un poco de experiencia con ellos, resultan bastante más eficiente que un lenguaje gráfico o esquemático.

2.2.4. Tipos de HDL

- ABEL HDL: es la abreviatura de Advanced Boolean Expression Language. Es un lenguaje de bajo nivel, permite definir un circuito a nivel de arquitectura (puertas básicas, ecuaciones lógicas, etc).
- AHDL: es la abreviatura de Altera Hardware Description Language (Lenguaje de Descripción de Hardware de Altera). Es un lenguaje de nivel medio, permite definir un circuito en modo jerárquico y el uso de descripciones de comportamiento.
- VHDL: es el acrónimo que representa la combinación de VHSIC (Very High Speed Integrated Circuit) y HDL (Hardware Description Language). Es un lenguaje utilizado para describir circuitos en alto nivel, el cual es reconocido como un estándar IEEE. Además, es un lenguaje que posee una sintaxis similar al ADA.
- Verilog HDL: de la mismo forma que VHDL, el Verilog es un lenguaje de alto nivel usado para modelar circuitos electrónicos complejos. Este también guarda similitud con otro lenguaje de programación como es C, aunque la gran diferencia entre ellos es que Verilog no ejecuta las sentencias de forma estrictamente lineal.

2.3. Comparativa entre VHDL y Verilog HDL

Una vez vistos los distintos lenguajes HDL más destacados, nos centraremos en los lenguajes de alto nivel de programación, que son VHDL y Verilog HDL. A continuación veremos en que difieren y en que se asemejan estos dos lenguajes de descripción de hardware.***(REF. [3])***

Capacidad

La descripción del hardware se puede modelar con la misma eficacia tanto en VHDL como en Verilog, aunque sí es cierto que las construcciones del modelado abarcan un rango ligeramente diferente dentro de los niveles de abstracción de comportamiento.

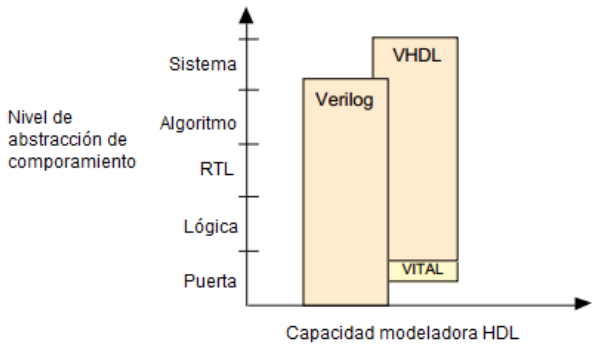

Figura 2: Gráfica de la capacidad modeladora de VHDL y Verilog HDL (Imagen extraída de [4])

Así pues, la elección de uno de ellos no se puede basar solamente en la capacidad técnica que poseen, ya que ofrecen prácticamente propiedades similares.

- **Tipos de datos**

En VHDL se pueden utilizar multitud de lenguaje y de tipos de datos definidos por el usuario. Esto significa que es necesaria la existencia de funciones de conversión para convertir datos de un tipo a otro. La elección de qué tipo de datos se va a utilizar debe tomarse con prudencia, sobre todo en ciertos tipos que pueden traernos problemas laboriosos de solucionar.

Todo esto hace que los modelos sean más fáciles de escribir, más fáciles de leer e interpretar y evitar aquellas funciones de conversión innecesarios que estorbarían en el código.

Los tipos de datos de Verilog, en comparación con VHDL, son muy simples y muy fáciles de usar, sin embargo, a diferencia de VHDL, todos los tipos de datos son definidos por Verilog y no por el usuario.

VHDL puede ser preferible por la multitud de lenguaje y tipos de datos que permite y Verilog por su simplicidad.

- **Reutilización del diseño**

Una característica de VHDL es que los procedimientos y las funciones pueden colocarse en un "paquete" de modo que están disponibles para cualquier otro diseño que desee utilizarlos.

Por otro lado tenemos Verilog, en el que el concepto de "paquete" no existe, las funciones y procedimientos que se usan en un modelo deben ser definidos en el módulo.

Para hacer que las funciones y procedimientos puedan ser accesibles desde un módulo diferente deben colocarse en sistemas de archivos diferentes y se deben incluir usando el directorio de compilación "include".

- **Facilidad para aprenderlos**

Si se empieza con un conocimiento nulo de los dos lenguajes, Verilog es probablemente más sencillo de captar y de entender. VHDL, al principio, puede parecer menos intuitivo, pero esta forma de programación lo hace más robusto y potente, aunque requiera un tiempo de aprendizaje superior. Además, permite modelar un mismo circuito de infinidad de maneras diferentes.

- **Facilidad para leerlos**

Este aspecto es más cuestión del estilo de programación y de la experiencia de usuario que de las características del lenguaje. VHDL es un lenguaje conciso y detallado, que se parece al lenguaje ADA, mientras que las instrucciones de Verilog se basan en C.

- **Llamadas a librerías, procedimientos y tareas**

En VHDL si que existe la posibilidad de usar librerías, aspecto muy útil, sobretodo, en grandes proyectos de diseño. Sin embargo, en Verilog no existe el concepto de librerías, debido a la naturaleza de su lenguaje.

VHDL también permite las llamadas a procedimientos concurrentes, pero Verilog tampoco permite llamadas a tareas concurrentes.

Capítulo 3.

Estudio comparativo sobre el aprendizaje de lenguajes HDL en las universidades

Una vez vistas las principales diferencias, entre VHDL y Verilog, en los aspectos propios del lenguaje que más nos interesan, se va a estudiar cual de los dos lenguajes está más extendido en la docencia universitaria, para que nos permita decantarnos por el más adecuado.

En la Tabla 1 se comparan diferentes universidades, para ver que lenguaje de descripción de hardware incluyen en los temarios de las asignaturas relacionadas con la electrónica digital.

Universidad	Universidad Politécnica de Valencia	Universidad de Granada	Universidad de Sevilla			Universidad de Zaragoza	
Titulación	Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación	Grado en Ingeniería de Tecnologías de Telecomunicación	Grado en Ingeniería en Electrónica Industrial	Grado en Ingeniería de Tecnologías Industriales	Grado en Ingeniería de las Tecnologías de Telecomunicación	Grado en Ingeniería Electrónica y Automática	Grado en Ingeniería d Tecnologías Industriales
Asignatura	Sistemas digitales programables	Sistemas electrónicos digitales	Diseño digital avanzado	Sistemas electrónicos digitales	Electrónica digital	Electrónica digital	Sistemas electrónicos digitales
Características de la asignatura	- Obligatoria - 4º semestre	- Obligatoria - 5º semestre	- Optativa - 8º semestre	- Optativa - 5º semestre	- Obligatoria - 5º semestre	- Obligatoria - 4º semestre	- Optativa - 7º semestre
Créditos ECTS	4,5	6	6	4,5	4,5	6	6
Lenguaje HDL	Verilog HDL	VHDL	VHDL	VHDL	VHDL	VHDL	VHDL
Horas de prácticas (HDL)	(-)	12	(-)	(-)	3	(-)	(-)
Bibliografía sobre HDL	- **A Verilog HDL primer.** J.Bashker, Second Edition, Star Galaxy Publishing, 1999. - **Verilog HDL: a guide to digital design and synthesis.** Samir Palnitkar, SunSoft Press, 1996. - **Verilog digital computer design : algorithms into hardware.** Mark Gordo Arnold, Prentice-Hall, 1998.	- **VHDL for logic synthesis.** Andrew Rushton, 3erd Edition, Wiley, 2011.	(-)	(-)	(-)	**- Electrónica digital. Aplicaciones y problemas con VHDL.** José Ignacio Artigas, Luís Ángel Barragán, Carlos Orrite e Isidro Urriza, Universidad de Zaragoza, 2004.	- **Electrónica digital. Aplicaciones y problemas con VHDL** José Ignacio Artigas, Luís Ángel Barragán, Carlos Orrite e Isidro Urriza, Universidad d Zaragoza, 2004.
Universidad	**Universidad de Nebrija**	**Universidad Politécnica de Catalunya**			**Universidad de Valencia**		**Universidad del País Vasco**
Titulación	Grado en Ingeniería en Tecnologías Industriales	Grado en Ingeniería de Sistemas Electrónicos	Grado en Ingeniería de Sistemas de Telecomunicación	Grado en Ingeniería Electrónica Industrial y Automática	Grado en Ingeniería Electrónica Industrial	Grado en Ingeniería Electrónica de Telecomunicación	Grado en Ingeniería Electrónica
Asignatura	Sistemas digitales	Diseño digital	Diseño digital	Sistemas digitales	Sistemas electrónicos digitales I	Sistemas electrónicos digitales I	Electrónica digital
Características de la asignatura	- Obligatoria* - 8º semestre	- Obligatoria - 3º semestre	- Obligatoria - 3º semestre	- Obligatoria - 6º semestre	- Obligatoria - 5º semestre	- Obligatoria - 3º semestre	- Obligatoria - 5º semestre
Créditos ECTS	6	6	6	6	6	6	6
Lenguaje HDL	VHDL	VHDL	VHDL	VHDL	VHDL	VHDL	VHDL
Horas de prácticas							

A simple vista destaca que el lenguaje HDL más empleado en un entorno educativo universitario a nivel de grado es, con mucha diferencia, el VHDL, independientemente de la titulación que se trate. De todas las universidades que han sido objeto de estudio, tan sólo una de ellas estudia únicamente Verilog HDL, mientras que otra presenta una introducción sobre dicho lenguaje, siendo el VHDL el que verdaderamente se estudia con cierta profundidad.

A pesar de que el aprendizaje del VHDL es más complejo y cuesta más tiempo dominarlo, las universidades nacionales se decantan por este lenguaje, ya que ofrece un abanico más amplio de posibilidades y herramientas que benefician el diseño de sistemas electrónicos digitales, lo que lo hace más potente.

En cuanto a las titulaciones estudiadas, además de la titulación de Grado en Ingeniería en Tecnologías Industriales, se han incluido en el análisis tanto Grados en Ingeniería Electrónica como Grados en Ingeniería en Tecnologías de Telecomunicaciones. Además de confirmar el predominio del VHDL, de esta forma se percibe como en el Grado en Ingeniería en Tecnologías Industriales las asignaturas, en donde se presentan los lenguajes HDL, se imparten en el séptimo y octavo semestre, tratándose siempre de asignaturas optativas.

Lo que sucede comúnmente en este grado es que se ve una primera asignatura sobre electrónica digital, que sí que es obligatoria, y posteriormente, para el alumnado que elige la mención en electrónica, es cuando se presentan, por primera vez, los lenguajes de descripción de hardware junto a una ampliación de los conocimientos en electrónica digital.

A su vez, en las otras dos titulaciones de grado analizadas, las asignaturas, en donde se ven los lenguajes HDL, tienen lugar durante el segundo (tercer y cuarto semestre) y el tercer curso (quinto y sexto semestre), teniendo siempre un carácter obligatorio. Incluso, en algunas universidades, se introducen en una asignatura obligatoria y se estudian con más profundidad en una optativa.

A continuación se detallará que lenguaje HDL se enseña en la Universidad Pública de Navarra y de qué forma se enfoca el aprendizaje de estos.

Titulación	**Grado en Ingeniería en Tecnologías Industriales**	**Grado en Ingeniería Eléctrica y Electrónica**	**Grado en Ingeniería en Tecnologías de Telecomunicación**	**Grado en Ingeniería en Tecnologías de Telecomunicación**
Asignatura	Electrónica digital	Electrónica digital	Sistemas digitales I	Diseño y test de sistemas electrónicos
Características de la asignatura	- Obligatoria - 5º semestre	- Obligatoria - 4º semestre	- Obligatoria - 2º semestre	- Obligatoria - 5º semestre
Créditos ECTS	6	6	6	6
Contenidos en lenguajes HDL	Introducción al VHDL	Introducción al VHDL	Introducción al VHDL	Diseño y verificación de sistemas digitales con lenguajes HDL

Tabla 2: Comparación de contenidos sobre lenguajes HDL en asignaturas de grado en la UPNA

En primer lugar, está el Grado en Ingeniería en Tecnologías de Telecomunicación, en el cual hay dos asignaturas, ambas obligatorias, en las que se estudian los lenguajes HDL. La primera que se imparte es sistemas digitales I, en la que solamente se introduce el VHDL en una práctica, haciendo más hincapié en el lenguaje esquemático y gráfico. Sin embargo, en el quinto semestre hay una asignatura en la que sí que se diseña y simula sistemas electrónicos mediante lenguajes HDL.

Es en esta asignatura donde se imparten los conocimientos necesarios para dominar el lenguaje y para poder llegar a diseñar sistemas digitales mediante esta herramienta.

Por otro lado, tanto en el Grado en Ingeniería Eléctrica y Electrónica como en el de Tecnologías Industriales, sólo se presenta una asignatura, electrónica digital, en donde se realiza una práctica de introducción al VHDL. Ocurre como en el caso anterior con la asignatura sistemas digitales I, en donde las prácticas se centran en el diseño a través de editores gráficos.

Se puede observar, que a diferencia del resto de universidades analizadas, en el Grado en Ingeniería en Tecnologías Industriales, la asignatura que incluye en su temario el aprendizaje de un HDL tiene carácter obligatorio. Se considera esencial conocer, aunque solo se introduzca, algún tipo de lenguaje HDL en una titulación como esta, independientemente de la mención que el alumno elija.

En general, en el grado no se estudian a fondo los lenguajes HDL, se centran más en el diseño de sistemas mediante lenguajes gráficos y esquemáticos, aunque si es verdad que en la mayoría se ve una introducción al VHDL.

Se va a realizar un estudio similar al que se ha llevado a cabo con los grados, pero en este caso el objeto de estudio será el máster universitario, de esta forma completaremos la visión sobre los lenguajes HDL que se tiene en las universidades españolas.

A continuación se podrán ver en la Tabla 3 los resultados.

Universidad	**Universidad de Zaragoza**	**Universidad Politécnica de Madrid**	**Universidad Politécnica de Cartagena**	**Universidad de Navarra**
Titulación	Máster en Ingeniería Electrónica	Máster en Ingeniería de Sistemas y Servicios	Máster en Tecnologías de la Información y Comunicaciones	Máster Universitario en In Telecomunicación
Asignatura	Diseño electrónico de sistemas empotrados en FPGA	Sistemas empotrados	Tendencias en el diseño de sistemas electrónicos	Sistemas de comunicación
Créditos ECTS	4	5	3	5
Lenguaje HDL	VHDL	VHDL	VHDL	VHDL
Bibliografía sobre HDL	(-)	**- VHDL for Logic Synthesis**, Andrew Rushton, Wiley; 2 edition, 1998 (ISBN: 978-0471983255).	**- Fundamentos de sistemas digitales**. 7ª Edición. Thomas Floyd. Prentice Hall, 2000. - **Problemas resueltos de Electrónica Digital.** 1ª Edición. Javier García Zubía, Ed. Thomson, 2003.	- **Essential VHDL: RTL Synt Right**, Sundar Rajan (ISBN: 0-0). - **The Designer's Guide to** J. Ashenden, 3erd Edition, Kaufmann Publishers, 199

Universidad	**Universidad de Alcalá**	**Universidad de Málaga**	**Universidad Complutense de Madrid**	**Universidad de Santiago d Compostela**
Titulación	Máster Universitario de Sistemas Electrónicos Avanzados, Sistemas Inteligentes	Máster Universitario en Inteligencia Ambiental: Sistemas electrónicos para entornos inteligentes	Máster en Nuevas Tecnologías Electrónicas y Fotónicas	Máster Universitario en Te Información
Asignatura	Diseño Electrónico	Diseño de Sistemas Empotrados basados en FPGA	Diseño de circuitos integrados	Diseño electrónico avanza
Créditos ECTS	6	3	6	6
Lenguaje HDL	VHDL	VHDL	VHDL	VHDL
Bibliografía sobre HDL	- **VHDL Lenguaje estándar de diseño electrónico**, L. Terés, Y.		- **The designer's guide to VHDL**. P. J.	- **VHDL Coding and Logic D**

Al igual que en las titulaciones de grado, en los másteres también se utiliza VHDL, pero en este caso la asignatura está destinada completamente al aprendizaje de un tipo de lenguaje de descripción de hardware. Es en este ámbito, donde se va a desarrollar el entorno de aprendizaje que vamos a desarrollar a lo largo del presente trabajo, en donde se verá a fondo el diseño de sistemas electrónicos mediante un lenguaje HDL.

No hay mucho más que decir sobre esta tabla, simplemente confirma el predominio del VHDL en la enseñanza universitaria en España.

Una vez visto el panorama nacional, en donde se ve perfectamente como el lenguaje VHDL es el más utilizado en asignaturas de grado, se va a realizar otro pequeño análisis sobre universidades estadounidenses. De este modo, se observará si en las universidades de Estados Unidos existe un predominio tan claro del VHDL sobre el Verilog HDL.

En la Tabla 4 se muestran los resultados obtenidos:

Universidad	**University of Central Florida**	**University of Illinois**	**Columbia University**	**University of Texas at Dallas**	**Boston University**	**Souther Universi**
Course	Field-Programmable Gate Array (FPGA) Design	Introduction to VLSI System Design	Computer hardware Design	Digital Circuits Lab	Advanced Digital Design with Verilog and FPGAs	Program Design
Créditos	3	(-)	3	3	4	4
Lenguaje HDL	Verilog HDL	VHDL	VHDL	Verilog HDL	Verilog HDL	Verilog
Bibliografía	- **Digital Design (Verilog): An Embedded Systems Approach Using Verilog**, Peter Ashenden. Patrick Schaumont,Springer, 2009	- **Principles of CMOS VLSI Design**, N. Weste and K. Eshraghian, Second Edition, Addison-Wesley, 1993.	(-)	- **Verilog® HDL: A Guide to Digital Design and Synthesis**, Samir Palnitkar, Second Edition	- **Advanced Digital Design with the Verilog HDL**. M.D. Ciletti (ISBN: 0136019285) - **Verilog Styles for Synthesis of Digital Systems**. David R. Smith, Paul Franzon (ISBN: 0201618605)	- **FPGA Verilog** Chu , W
Universidad	**Northwestern Polytechnic University**	**Indiana University-Purdue University Indianapolis**	**San José State University**	**Univesity of New Mexico**	**University of Arkansas**	**Lawrenc Technol Universi**
Course	**Verilog HDL & Digital Design**	Advanced Digital System Design	Digital Design with FPGAs	Hardware Design with VHDL	Digital Design	Digital E Digital E Laborat
Créditos	(-)	3	3	3	4	4
Lenguaje HDL	Verilog HDL	VHDL y Verliog HDL	Verilog HDL	VHDL	VHDL	VHDL
Bibliografía	- **VERILOG Digital System Design** by Zainalabedin Navabi. Publisher: McGraw-Hill,	**-Digital Systems Design**		- **RTL Harware Design**	**- Fundamentals of**	- **Fundar**

Los datos obtenidos difieren bastante del estudio de las universidades nacionales que se ha realizado, en este caso no destaca ningún lenguaje sobre el otro. A grandes rasgos, se puede afirmar que la mitad de las universidades estudiadas se inclinan por VHDL y la otra mitad por Verilog HDL e incluso hay algún caso en el que incluyen los dos en la misma asignatura.

Otra diferencia que resalta respecta a las universidades españolas, es que los cursos son de entre 3 y 4 créditos y son totalmente específicos en el diseño de sistemas electrónicos digitales.

Como se ha visto en la tabla anterior, las asignaturas son más extensas pero el temario de las asignaturas abarca conceptos de electrónica digital en general, en donde se tratan sus fundamentos básicos y se introduce, de forma breve, un lenguaje de descripción de hardware. Sin embargo, en las asignaturas de las universidades estadounidenses se invierten todos los créditos en estudiar y aprender un lenguaje HDL. Previamente los alumnos ya han cursado asignaturas donde aprenden los conocimientos de la electrónica digital, por lo que después están capacitados para cursar este tipo de asignaturas tan específicas.

Conclusión

Tras estudio del aprendizaje de lenguajes HDL tanto en universidades nacionales como estadounidenses, ha quedado demostrado que a pesar de que en E.E.U.U. no hay un predominio claro de un lenguaje, en España casi el 100% de las universidades estudiadas utilizan VHDL como lenguaje de descripción de hardware.

Por otro lado en la comparativa de los dos lenguajes se ha visto que VHDL es más laborioso de dominar, pero que ofrece muchas más posibilidades a la hora de diseñar sistemas digitales.

Por todo ello, VHDL es el lenguaje que se usará en el entorno de aprendizaje que desarrollaremos en este trabajo.

Capítulo 4.

Hardware

El hardware del que se dispone en el laboratorio, para llevar a cabo las prácticas que se van a desarrollar, es la tarjeta educacional DE2 de la compañía Altera. Se trata de una tarjeta de nivel intermedio, la cual posee un dispositivo FPGA Cyclone II EP2C35F672C6 con EPCS16.

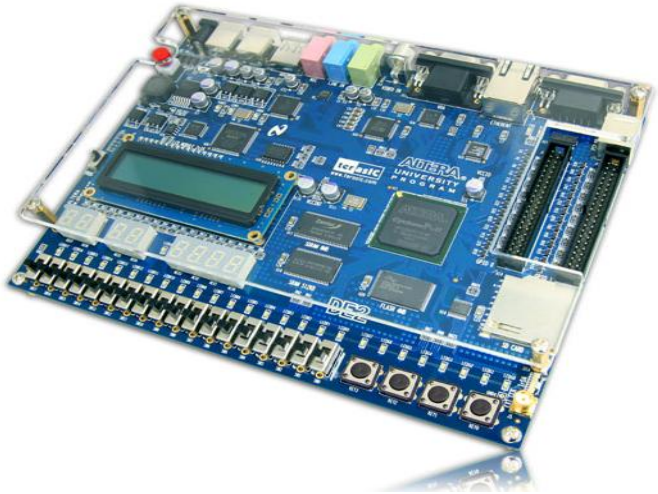

Figura 3: DE2 Education Board
(Imagen extraída del enlace Web de Terasic Technologies)

Algunas ventajas que nos proporciona este material:

- Altera oferta un programa universitario en el que se realizan descuentos y facilita el acceso a productos destinados para el uso académico a entidades académicas.
- Altera ofrece un software libre (QUARTUS II) para este tipo de aplicaciones, que se puede obtener mediante una descarga directa dese su página web.
- La tarjeta DE2 dispone del núcleo de procesador integrado NIOS II, que proporciona alta flexibilidad y velocidad de procesamiento.
- Nos ofrece la posibilidad de ampliar la memoria mediante el acoplamiento de tarjetas externas.
- Dispone de un gran número de LEDs, pulsadores, puertos de comunicación, displays, etc.
- Trae consigo el kit completo para la programación y configuración del sistema, además de un manual de usuario para ayudar a entender su funcionamiento.

El inconveniente más destacable de la DE2 es que los precios de otras tarjetas del mismo nivel, pertenecientes a otras compañías, son menores. Sin embargo, sí que es verdad, que ninguna de ellas oferta un abanico tan amplio de posibilidades a la hora de desarrollar un programa de prácticas para el alumnado universitario.

A continuación, se recogerá en una tabla las principales características de la tarjeta DE2:

Característica	Descripción
Dispositivo FPGA	- Cyclone II EP2C35F672C6 with EPCS16 16-Mbit serial configuration
RISC	- NIOS II
Dimensiones	- 8x6 pulgadas
Puertos de comunicación	- Line In/Out, Microphone In (24-bit Audio Codec). - Video In (NTSD/PAL/Multi-format) - RS-232 - Infrared port - PS/2 mouse or keyboard port - 10/100 Ethernet - USB 2.0 (type A and B)
Puertos de expansión	- 2x40 pines
Memoria	- 8 MB SDRAM - 512 KB SRAM - 4 MB Flash - SD memory card slot
Displays	- 8x7-segments display - 16x2 LCD display
Switches	- 18 interruptores - 4 pulsadores
LEDs	- 9 greeen LEDs - 18 red LEDs
Relojes	- 50 MHz clock - 27 MHz clock - External SMA clock input
Cable de descarga	- USB Blaster
Precio	- Academic: $269 - Commercial: $495

Tabla 5: Características de la tarjeta DE (Información extraída del enlace Web de Altera Corporation)

El software que se utilizará para llevar a cabo los diseños que se requieran en los guiones de prácticas será el Quartus II, que es una herramienta de software producida por Altera.

Este programa de diseño permite realizar diseños mediante varias herramientas y luego permite tanto llevar a cabo simulaciones temporales, como programar dispositivos de lógica programable

En este caso se realizarán todas las prácticas mediante la versión Quartus II Edición Web, se trata de una versión gratuita de este software que se puede descargar directamente desde la página oficial de Altera.

Esta versión está limitada por el número de dispositivos que es capaz de simular y programar. El FPGA que contiene la tarjeta DE2 es de la familia Cyclone II y es una de las familias de dispositivos que este programa puede configurar, por lo que con esta versión será suficiente.

Capítulo 5.

Guiones de prácticas

Práctica 1: Introducción

Es muy importante, para el desarrollo de las siguientes prácticas, que se asienten los conceptos mostrados en esta primera práctica, así como que se adquiera soltura en el uso del software Quartus II.

En esta primera práctica, se introducirá el software Quartus II y se realizarán diferentes tipos de ejercicios, todos ellos sencillos de realizar.

1. Objetivos

- Introducción al diseño de circuitos lógicos con código en VHDL mediante el software Quartus II.
- Simular los diseños.
- Programar y configurar el FPGA de la tarjeta DE2 con el diseño realizado.

2. Material

- Ordenador personal con el software QUARTUS II disponible.
- Tarjeta educacional DE2
- Guión de prácticas

3. Desarrollo de la práctica

3.1. Puertas lógicas

El circuito que se va a diseñar para introducir el software Quartus II es el siguiente:

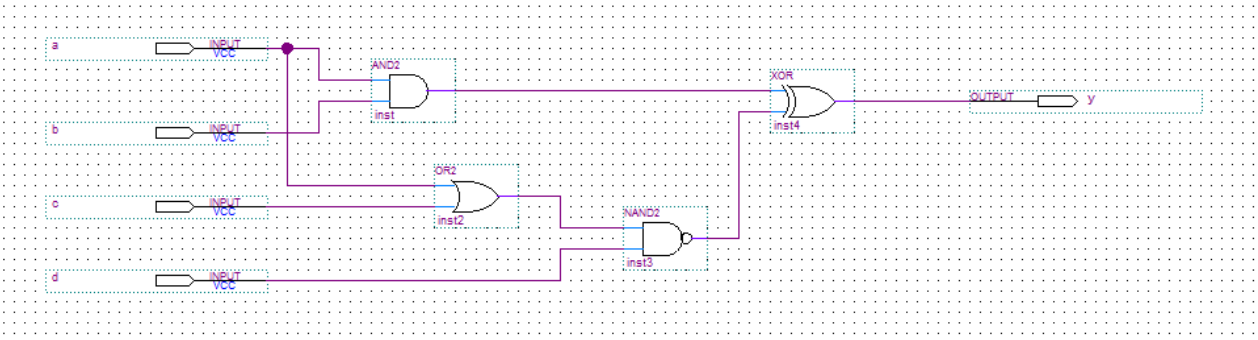

Figura 1: Esquemático ejercicio 3.1

Para llevar a cabo el diseño se seguirán los pasos que se mostrarán a continuación.

1º.- Crear el proyecto

Una vez inicializado el Quartus II, el primer paso es crear y definir un proyecto. Dentro de un proyecto se encuentran albergados una lista de archivos de diseño en un mismo directorio de trabajo.

Es muy importante que cada proyecto se encuentre en un directorio de trabajo diferente, ya que si hay varios de ellos en un mismo directorio, se pueden llegar a producir conflictos entre los proyectos a la hora de compilar.

Para crear el proyecto, se debe acudir al menú ***File*** y seleccionar la opción ***New Project Wizard***. Dentro de esta opción, nos aparecerán una serie de ventanas donde se podrá definir el proyecto mediante las operaciones que se detallarán a continuación.

En primer lugar aparecerá la siguiente ventana:

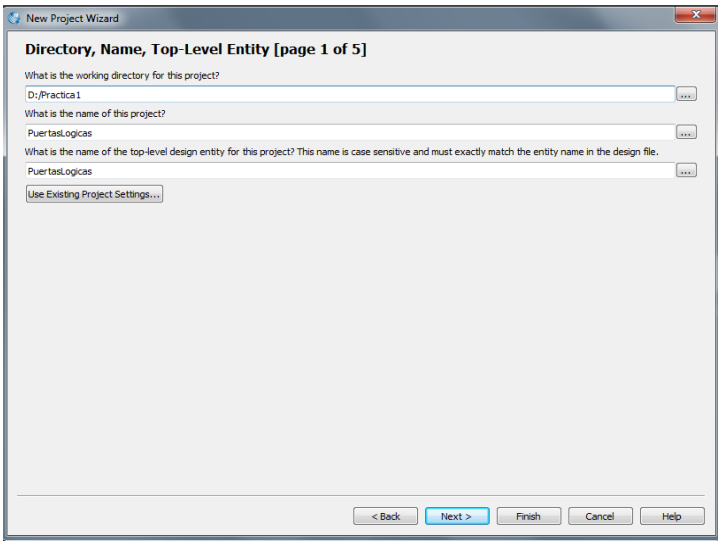

*Figura 2: Ventana 1 de 5 de la opción **New Project Wizard***

En este paso se selecciona el directorio de trabajo donde vamos a ubicar el proyecto, que como se ha comentado antes, cada proyecto se debe encontrar en un directorio distinto.

Además, también se le asigna un nombre al proyecto y al archivo de diseño que ocupa el lugar más alto en la jerarquía de archivos (top level entity), en este caso se nombran de igual forma.

Tras esta ventana nos aparece otra, en la que podremos incluir librerías y archivos de proyectos anteriores que sean de útiles para el proyecto actual.

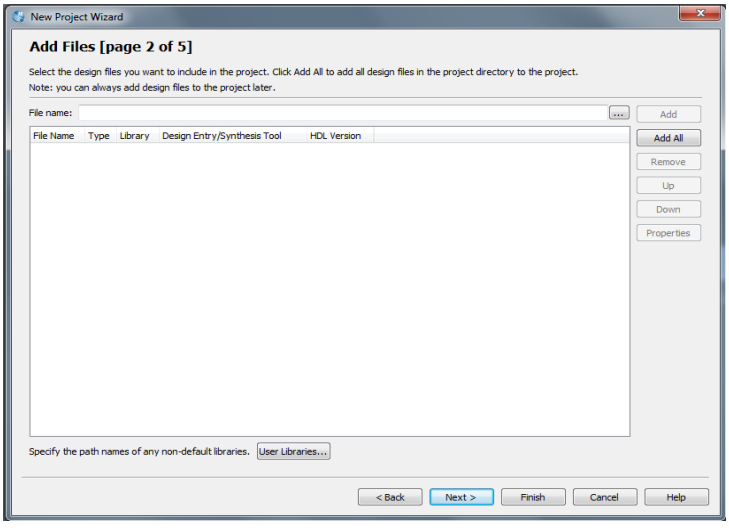

Figura 3: *Ventana 2 de 5 de la opción **New Project Wizard***

En la siguiente ventana aparece lo siguiente:

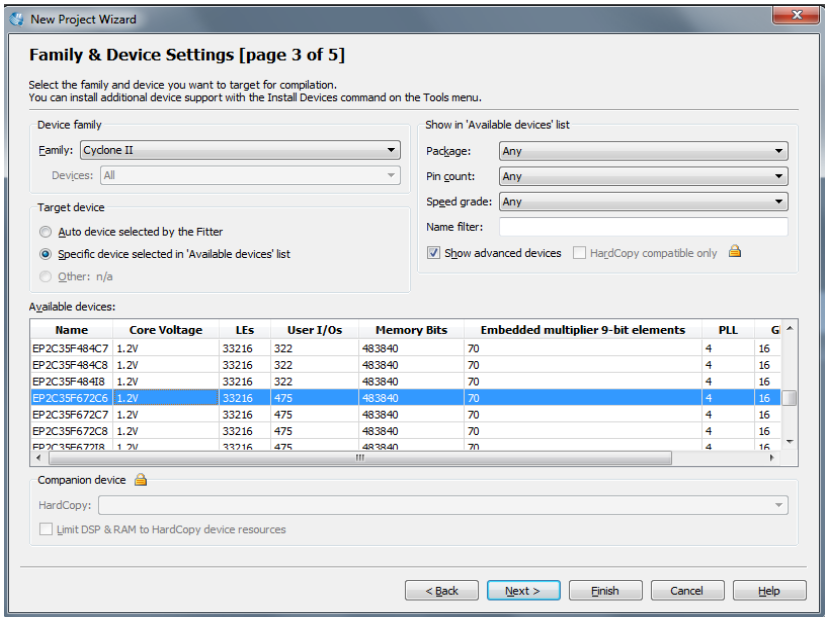

Figura 4: *Ventana 3 de 5 de la opción* ***New Project Wizard***

Aquí se tiene que seleccionar la familia de dispositivos y el dispositivo en concreto con el que se va a trabajar, este paso es importante, ya que si se elige un dispositivo distinto, no dejará volcar el diseño a la placa.

En este caso, se escoge la familia Cyclone II y el dispositivo EP2C35F672C6, que es el que se encuentra en la placa DE2, que es con la que se va a trabajar.

Se pueden especificar otras herramientas auxiliares para utilizar en el proyecto dentro de la siguiente ventana, pero ahora no es necesario, por lo que se pulsa directamente ***Next***.

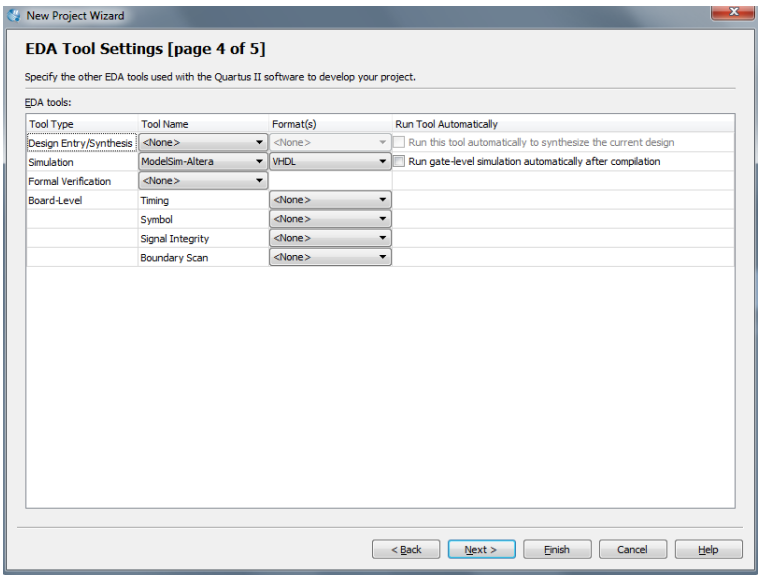

Figura 5: *Ventana 4 de 5 de la opción* ***New Project Wizard***

La última ventana nos presenta un resumen de todos los pasos anteriores, se comprueba que todo este correcto y se pulsa la tecla ***Finish***.

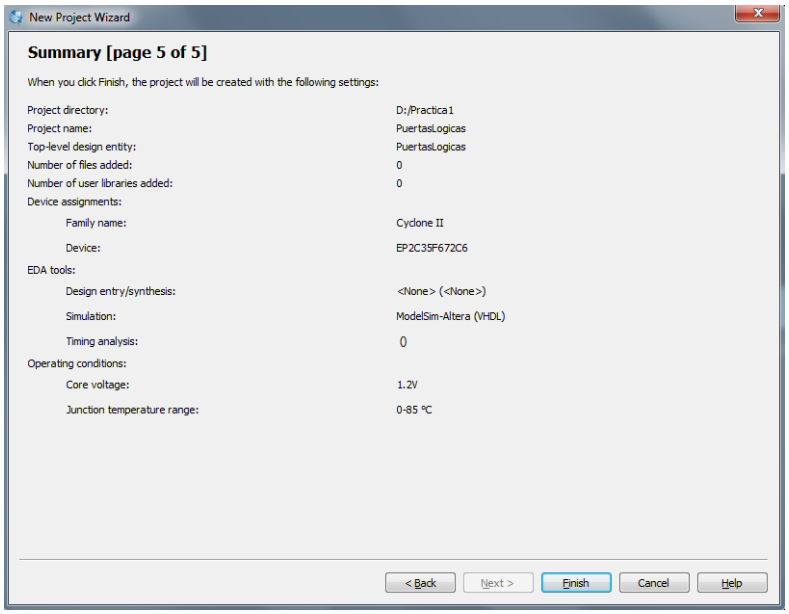

Figura 6: *Ventana 5 de 5 de la opción* ***New Project Wizard***

2º.- Crear el archivo VHDL

Hasta ahora, en cursos anteriores, sólo se ha trabajado con el Editor Gráfico de Quartus II, pero este software también incluye un Editor de Texto, el cual permite diseñar mediante varios lenguajes de programación. En estas prácticas se utiliza el lenguaje VHDL, que está extendido y aceptado en todo el mundo.

Para crear el Editor de Texto VHDL se accede mediante la opción ***New*** que está en el menú ***File***. Acto seguido aparecerá una ventana como esta:

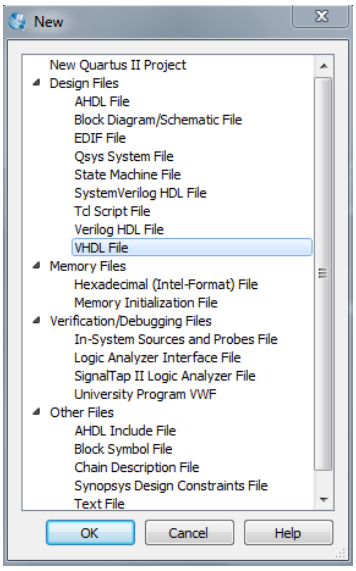

Figura 7: Creación de un nuevo archivo VHDL

Dentro de todas las opciones que se ofrecen, elegiremos ***VHDL File*** y se abrirá la siguiente ventana:

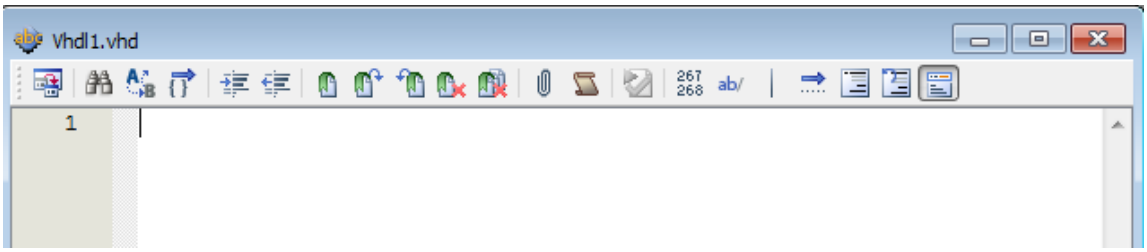

Figura 8: Aspecto de la ventana principal de un archivo VHDL

Una vez creado, lo primero que se va a hacer es guardar el archivo VHDL. Para ello dentro del menú ***File*** se clica en la opción ***Save As…***. Una vez que aparece la ventana, se introduce el nombre del archivo y el tipo, en este caso es importante que asegurar que la extensión es *.vhd*, para que el programa reconozca el lenguaje con el que se está programando.

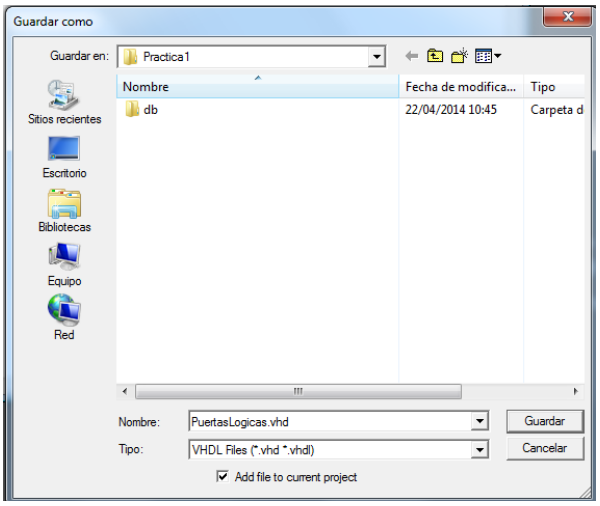

Figura 9: Guardado de archivo con extensión .vhd

Ahora, hay que escribir las líneas de código para implementar el circuito de la figura 1.

```
entity puertaslogicas is
   port( a,b,c,d: in bit;                    -- 4 entradas
         y: out bit);                        -- 1 salida
end puertaslogicas;

architecture ejemplo1 of puertaslogicas is   -- Cabecera del programa
begin                                        -- Empieza el programa
   process(a,b,c,d)                          -- Cabecera de un proceso
   begin                                     -- Empieza el proceso
     y<=(a and b)xor((a or c)nand d);        -- Sentencia para implementar el circuito lógico
         end process;                        -- Termina el proceso
end ejemplo1;                                -- Termina el programa
```

3º.- Compilar el diseño

Este proceso detecta los posibles errores que se han cometido en la sintaxis del diseño, sintetiza el diseño lógico, proporciona información temporal para llevar a cabo la simulación, ajusta el diseño al PLD seleccionado y genera un archivo con extensión .sof, necesario para el volcado del diseño a la tarjeta.

Antes de comenzar con la compilación, se debe tener en cuenta que en un mismo proyecto puede haber varios archivos con diferentes diseños, por lo que es necesario señalar al programa con cuál de ellos se quiere trabajar. Para ello, en la ***ventana Project Navigator*** (parte superior izquierda), dentro de la pestaña ***Files*** aparecen todos los archivos que contiene el proyecto, ahí se busca el diseño deseado y pulsando el botón derecho del ratón encima de él aparecen una serie de opciones, de todas ellas se elige ***Set as Top-Level Entity***.

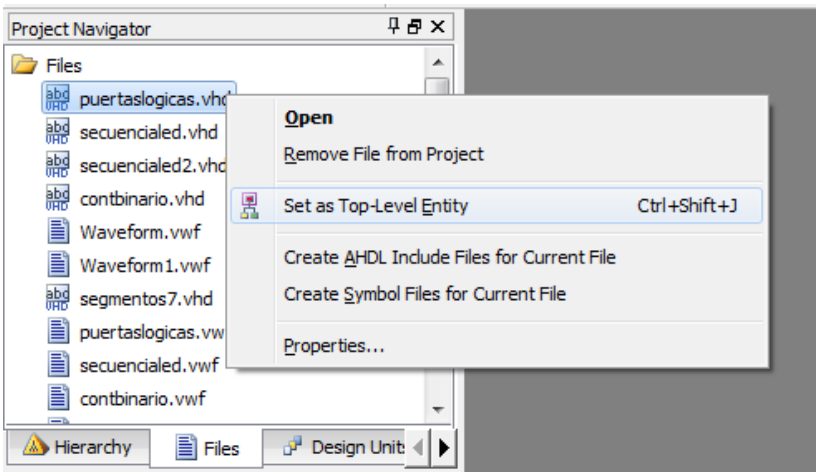

Figura 10: Asignación del nivel de jerarquía Top-Level-Entity

Ya seleccionado el archivo que se desea como Top-Level Entity, ya se puede proceder a su compilación.

Este paso es muy importante, ya que si no se realiza, el archivo con el que se trabajará será todo el rato el de mayor jerarquía dentro del proyecto y no aquel que el usuario quiera.

Legados a este punto, para iniciar la compilación, primero se guarda el diseño y después se accede al proceso a través de la opción ***Start Compilation*** en la pestaña ***Processing*** o clicando en el icono 

Cuando finaliza la compilación aparece una ventana en la que se muestra los posibles errores y advertencias acerca del diseño. Si no hay ningún error se puede continuar con el siguiente paso, pero si hubiese alguno hay que buscarlo y solucionarlo. Para ello, en la ventana de mensajes del compilador aparecen los errores en color rojo, solo es necesario clicar dos veces sobre ellos y el programa te lleva a la línea del Editor de Texto donde se encuentra el error.

En la figura 11 se muestra un ejemplo de la ventana de compilación que se acaba de mencionar. En este ejemplo se ha modificado alguna línea del código para provocar errores y ver como el programa los muestra, pero, para que quede claro, el código que se ha dado anteriormente está perfectamente elaborado y no contiene errores.

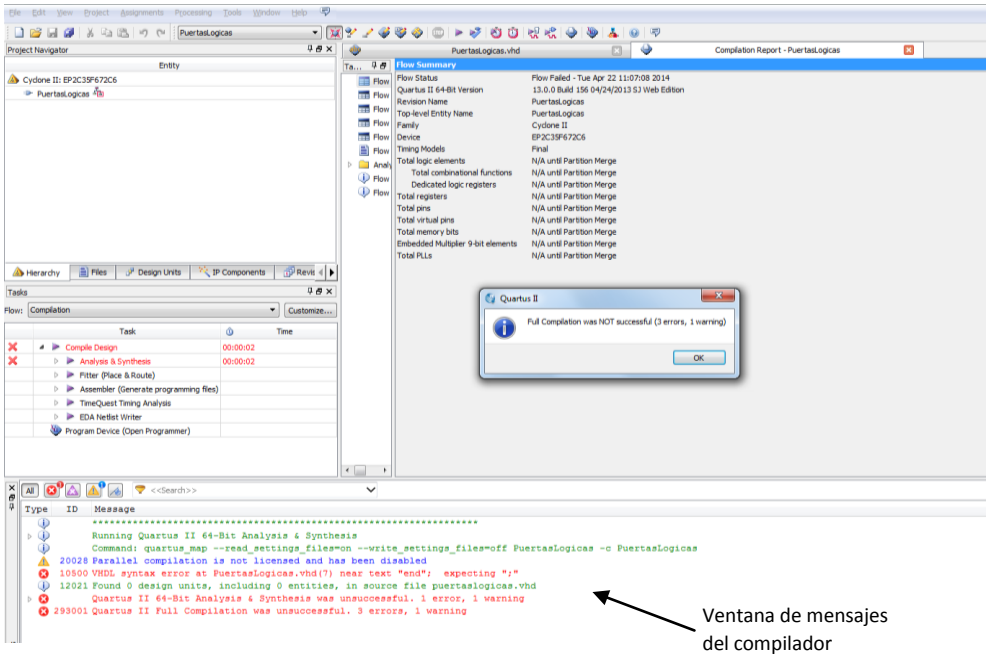

Figura 11: Compilación del diseño

4º.- Simulación

Una vez que se ha compilado el diseño sin errores y antes de volcar el diseño a la tarjeta, se va a comprobar el correcto funcionamiento de este mediante una simulación, la cual se lleva a cabo a través de un Editor de señales. Para abrir dicho Editor se abre el menú ***File*** y se crea un nuevo archivo mediante la opción ***New***, entonces aparece la siguiente ventana:

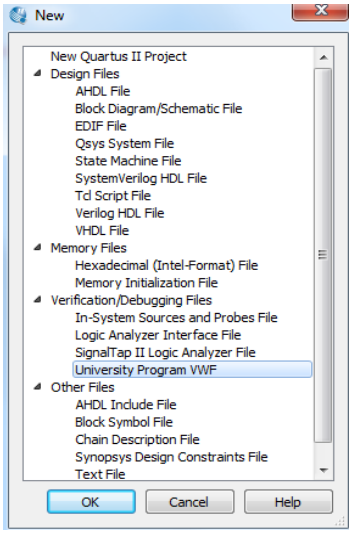

Figura 12: Creación de un nuevo archivo University Program VWF

Se elige la opción ***University Program VWF*** y se creará el Editor de Señales, el cual tiene es un entorno sencillo e intuitivo para trabajar, como se puede ver en la figura.

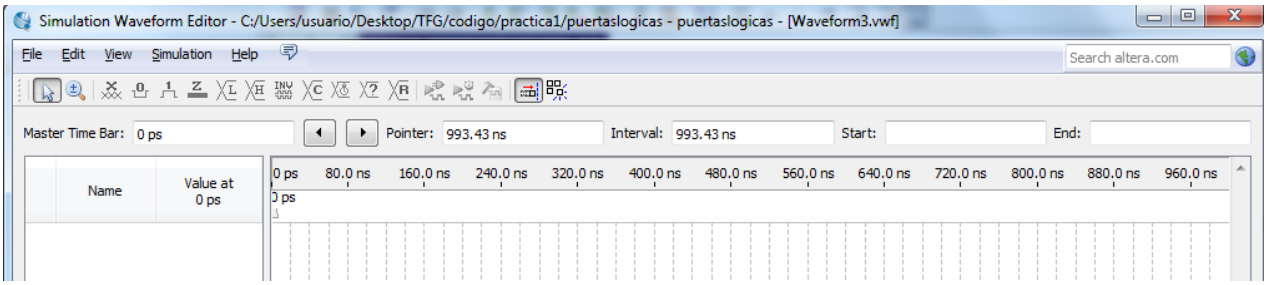

Figura 13: Aspecto de la ventana principal del Editor de Señales

Lo primero que se va a hacer es establecer el tiempo de análisis, para ello accedemos a la opción ***Set End Time…*** en la pestaña ***Edit***.

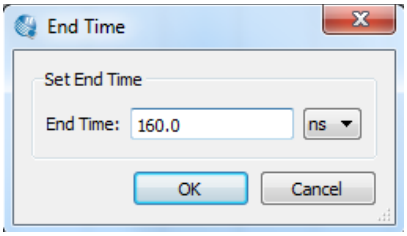

Figura 14: Establecimiento del tiempo de simulación

Ahora, se insertarán las señales en el editor, para así poder darles valores y realizar la simulación para ver si se obtienen los resultados esperados. Así pues, en la pestaña ***Insert*** dentro del menú ***Edit*** encontraremos la opción ***Insert Node or Bus…***, si clicamos en ella nos aparece la siguiente ventana:

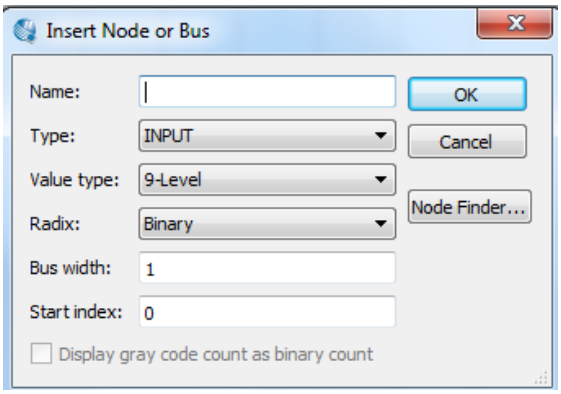

Figura 15: Ventana para insertar los canales

Dentro de esta ventana hay que pulsar la pestaña ***Node Finder…***, que lleva a la siguiente ventana. Una vez aquí, en la casilla ***Look in:*** se debe elegir el diseño que se quiere simular y, posteriormente, si se clica sobre la pestaña ***List*** aparecen todas las señales correspondientes a dicho diseño. Para incluir las señales en el editor hay que pasarlas a la ventana ***Selected Nodes:*** mediante la pestaña **>** o **>>**.

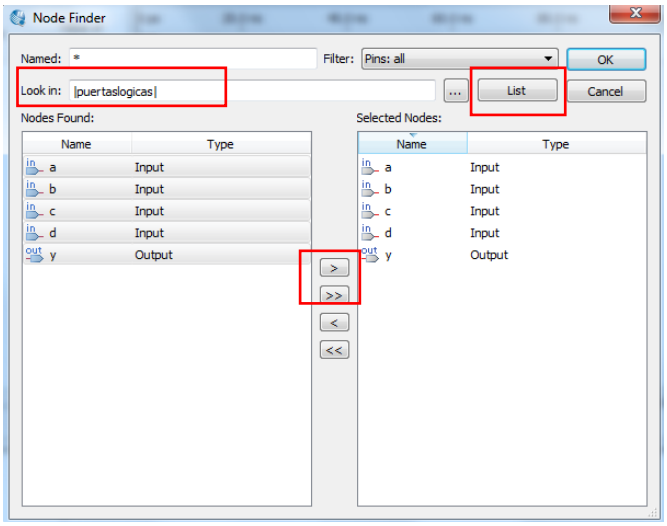

Figura 16: Elección de canales para la simulación

Ya tenemos las señales que se quieren simular en el editor, ahora se les darán valores para observar si los valores de salida son los deseados. Para ello hay una serie de herramientas con las que se pueden configurar las señales, algunas de las que más se utilizan son las siguientes:

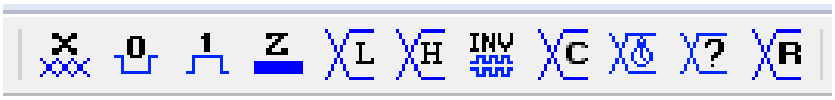

Figura 17: Barra de herramientas para configurar las señales

En esta caso utilizaremos la opción ***Count Value***  que sirve para asignar un periodo de onda, que lo elige el usuario, a la señal.

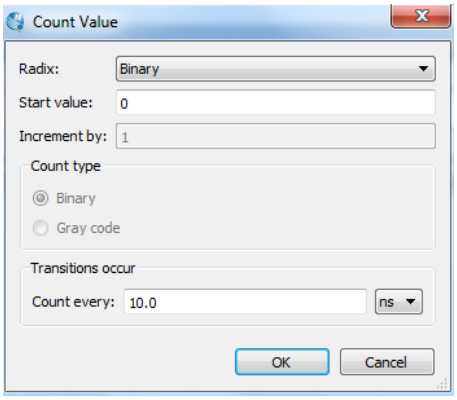

Figura 18: Elección del periodo de la señal

A continuación, con las señales con sus valores correspondientes, se procede a la simulación. El software ofrece dos opciones para la simulación y las dos se encuentran en el menú ***Simulation***, se trata de ***Run Functional Simulation***  y ***Run Timing Simulation***  . La primera opción se trata de una simulación ideal y la segunda realiza una simulación teniendo en cuenta los retrasos reales de las puertas lógicas, integrados, etc. utilizados en el diseño.

Una vez simulado, se mostrarán los resultados de la simulación y se comprobarán con los resultados calculados teóricamente.

5º.- Asignación de pines

Para volcar el diseño a la tarjeta, es necesario asignar a las variables del programa un pin específico de la propia tarjeta DE2. Para ello, se dispone de la herramienta ***Assignment Editor*** que se encuentra en la pestaña ***Assignments***, esta herramienta tiene el siguiente aspecto:

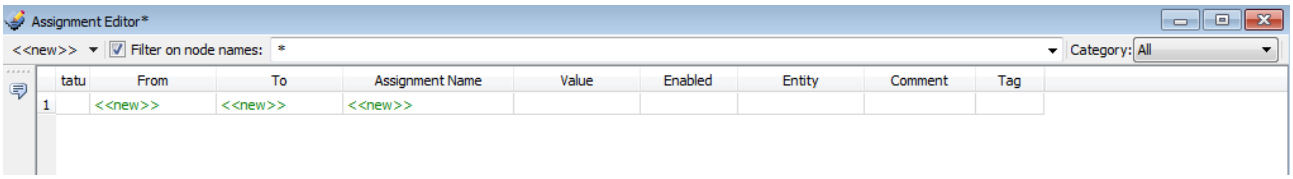

Figura 19: Ventana principal del Editor de Asignación de pines

Para encontrar los pines debemos hacer doble clic sobre la ventana debajo de la pestaña ***To***, en donde pone ***<<new>>***, entonces aparecerá el icono  , es ahí donde debemos clicar. A continuación, aparece la siguiente ventana:

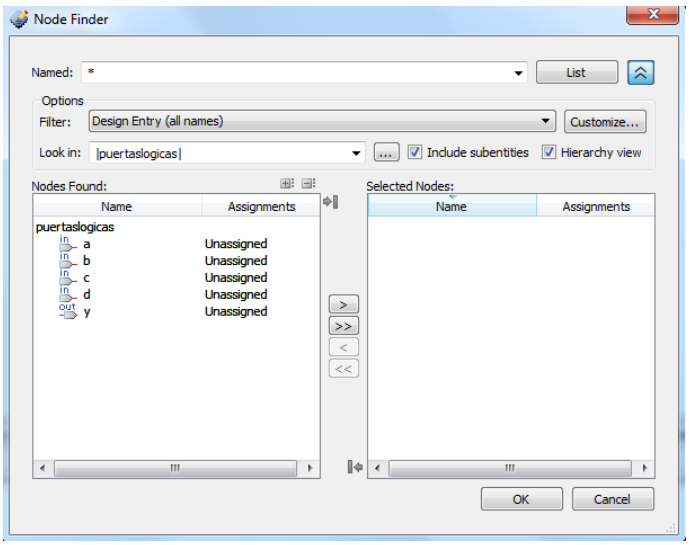

Figura 20: Elección de canales de entrada y salida

Es aquí donde se tienen que seleccionar los pines, si no nos salen directamente se deberá pulsar la pestaña ***List***. Una vez que aparezcan los pines debemos pasarlos a la ventana ***Selected Nodes***, mediante las pestañas **>** o **>>**, la primera solo pasa el pin seleccionado y la segunda pasa directamente todos los pines encontrados. Una vez que se tengan todos los pines seleccionados ya se puede pulsar ***Ok***.

Como se ve en la siguiente figura ya se han encontrado los pines, ahora hay que asignarles un pin específico de la tarjeta DE2.

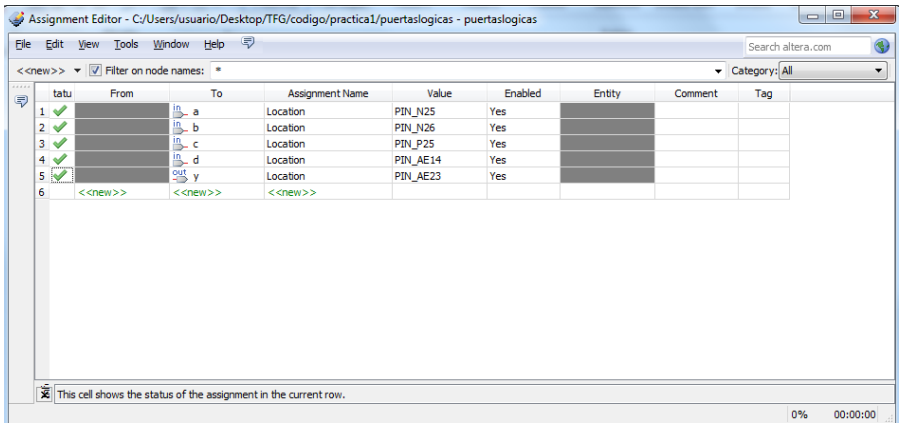

Figura 21: Asignación de pines

Para ello en la pestaña ***Assignment Name*** seleccionaremos la opción ***Location(Accepts wildcards/groups)***. Por último, en la pestaña Value, se les asignará el pin de la tarjeta DE2 que se desee, tan solo escribiendo el nombre de dicho pin en la casilla de esa columna. En este caso la asignación de pines corresponderá con la tabla 1.

Señal	Pin
a	PIN_N25
b	PIN_N26
c	PIN_P25
d	PIN_AE14
y	PIN_AE23

Tabla 1: Asignación de pines para el ejercicio 3.1.

Después de la asignación de pines, se debe guardar y posteriormente volver a compilar para que el dispositivo actualice la nueva configuración.

Una vez configurado, el software nos ofrece una herramienta que nos guarda la asignación de pines elaborada, en este caso, para otros diseños en los que puede ser útil. Esto se consigue accediendo a la opción ***Export Assignments…*** dentro del menú ***Assignments***, de esta manera se genera un archivo con extensión .csv, que se puede leer con Microsoft Excel. De la misma forma que se exporta el archivo, se puede importar en un proyecto nuevo, mediante la opción ***Import Assignments…*** situada en el mismo menú.

6º.- Programación y volcado

El dispositivo FPGA que viene integrado en la tarjeta DE2 debe ser programado para volcar el diseño sobre la tarjeta, esto se lleva a cabo mediante la herramienta ***Programmer***, que se encuentra en el menú ***Tools***, también tiene un icono para acceder a ella directamente: 

Antes de realizar la programación, se debe comprobar que el interruptor de la placa DE2 se encuentra en la posición RUN, este interruptor se encuentra en la parte izquierda de la place y tiene dos posiciones RUN y PROG.

Una vez comprobado eso ya se puede acceder a programar el dispositivo FPGA, como ya hemos dicho antes se utiliza la herramienta ***Programmer***, que tiene el siguiente aspecto:

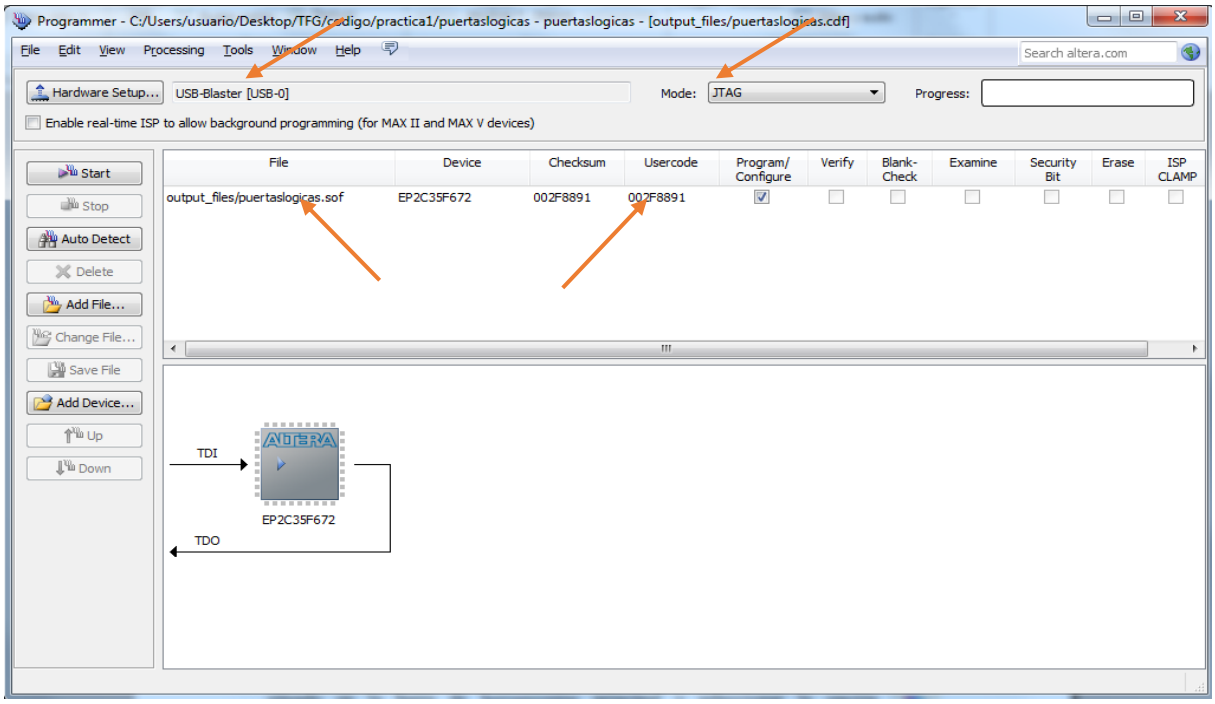

Figura 22: Ventana principal de la herramienta Programmer

Para la programación se necesita el archivo con extensión *.sof* que se ha generado durante la compilación y que permitirá el volcado del diseño. Además, como viene señalado en la figura, el modo de programación que vamos a utilizar es el modo JTAG, la casilla ***Program/Configure*** debe estar clicada y el hardware seleccionado debe ser USB-Blaster. Si no es así, hay que clicar sobre la opción ***Hardware Setup*** y elegir el hardware USB-Blaster como aparece en la siguiente ventana:

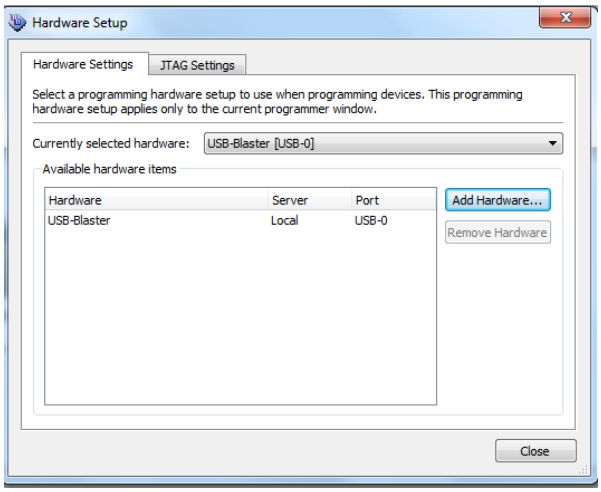

Figura 23: Elección del USB-Blaster

Una vez que se han seguido los pasos, ya se puede pulsar la pestaña ***Start*** y comprobar si nuestro diseño funciona correctamente en la tarjeta DE2.

Ejercicios

a) Obtener la tabla de verdad del circuito planteado.
b) Simular con todos los posibles estados de entradas y salidas y comparar los resultados con los obtenidos en la tabla de verdad.
c) Volcar el diseño a la tarjeta DE2.

3.2.- Secuencia de leds

Este segundo ejercicio consistirá en diseñar una secuencia de 6 leds, primero se encenderá el de la derecha del todo y cuando se pulse el pulsador, este se apagará y se encenderá el de su izquierda. Cuando vuelva a pulsarse el pulsador se apagará este y se encenderá el de su izquierda y así hasta que la secuencia llegue al sexto led. Cuando llegue al final, la secuencia deberá comenzar desde el principio otra vez. Además, se incluirá un botón de reset, que reiniciará la secuencia.

El código para este diseño es el siguiente:

```
library IEEE;                                          -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;                       -- Paquetes de la librería estándar de la IEEE

entity secuencialed is
   port (leds:out STD_LOGIC_VECTOR(5 downto 0);        -- 6 leds de salida
            pulsador: in STD_LOGIC;                    -- Canal de entrada del pulsador
            reset: in STD_LOGIC);                      -- Canal de entrada del reset
end secuencialed;

architecture secuencia of secuencialed is              -- Cabecera del programa
     signal cont: natural range 0 to 6;
     signal ledout: STD_LOGIC_VECTOR (5 downto 0);     -- Declaración de señales
begin
   process(pulsador,reset)                             -- Cabecera de proceso
   begin
      if reset='0' then                                -- Sentencia if para detectar la pulsación del reset
         cont<= 0;
      elsif pulsador'event and pulsador = '1' then     -- Sentencia elsif para detectar el flanco de subida de la
                                                       -- señal del pulsador
          if cont=6 then                               -- Sentencia if para detectar si cont debe reinicializarse a 1
            cont <= 0;                                 -- o seguir aumentando
          else
            cont <=cont + 1;
          end if;
      end if;
   end process;                                        -- Final de proceso
   process(cont)                                       -- Cabecera de proceso
   begin
       if cont=1 then                                  -- Sentencias if y elsif para detectar que led debe encenderse
         ledout<="000001";
       elsif cont=2 then
         ledout<="000010";
       elsif cont=3 then
         ledout<="000100";
       elsif cont=4 then
         ledout<="001000";
       elsif cont=5 then
         ledout<="010000";
       else
         ledout<="100000";
       end if;
       leds(5 downto 0) <= ledout(5 downto 0);         -- Traspaso de datos de la señal ledout a la señal de salida
    end process;                                       -- Final de proceso
 end secuencia;                                        -- Final de programa
```

Ejercicio

a) Simular con todos los estados posibles de entradas y salidas y comprobar los resultados
b) Volcar el diseño a la tarjeta DE2 y asignar a las señales los siguientes pines:

Señal	Pin
leds[0]	PIN_AE23
leds[1]	PIN_AF23
leds[2]	PIN_AB21
leds[3]	PIN_AC22
leds[4]	PIN_AD22
leds[5]	PIN_AD23
reset	PIN_G26
pulsador	PIN_N23

Tabla 2: Asignación de pines para el ejercicio 3.2.

4.- Ejercicios de diseño

4.1.- Secuencia de leds completada

Se trata de la misma secuencia que la del ejercicio anterior, pero en este caso se controlará mediante dos interruptores si la secuencia va hacia la izquierda, a la derecha o permanece quieta aunque se pulse el pulsador.

El diseño debe responder a esta combinación entre los dos interruptores:

Interruptor a	Interruptor b	Dirección
0	0	Izquierda
0	1	Quieto
1	0	Derecha
1	1	Quieto

Tabla 3: Tabla de verdad del ejercicio 4.1.

La asignación de pines será la siguiente:

Señal	Pin
a	PIN_N25
b	PIN_N26
leds[0]	PIN_AE23
leds[1]	PIN_AF23
leds[2]	PIN_AB21
leds[3]	PIN_AC22
leds[4]	PIN_AD22
leds[5]	PIN_AD23
reset	PIN_G26
pulsador	PIN_N23

Tabla 4: Asignación de pines del ejercicio 4.1.

4.2.- Contador binario

Se pide diseñar un contador binario de 0 (0000) a 15 (1111). El número aumentará cuando se pulse el pulsador y el resultado se mostrará a través de 4 leds.

La asignación de pines será la siguiente:

Señal	Pin
pulsador	PIN_N23
leds[0]	PIN_AE23
leds[1]	PIN_AF23
leds[2]	PIN_AB21
leds[3]	PIN_AC22

Tabla 5: Asignación de pines del ejercicio 4.2.

4.3.- Codificador 7 segmentos

Diseñar un programa que, introduciendo un número binario mediante 4 interruptores, muestre dicho número en dos displays 7 segmentos, un display para las unidades y otro para las decenas.

Aquí se muestra la posición y la numeración de cada uno de los segmentos del display que se va a utilizar para este diseño:

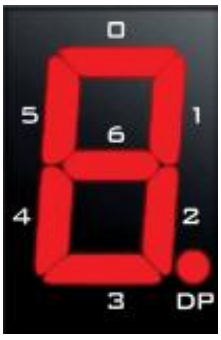

Figura 24: Display 7 segmentos

Al utilizar 4 interruptores como entrada, los números irán del 0 (0000) al 15 (1111).

Nota: Cabe destacar, que los segmentos del display se activan en bajo, por lo que cuando a un segmento le llegue un 0, este se activará, y si le llega un 1 se apagará.

La asignación de pines será la siguiente:

Señal	Pin
display1[0]	PIN_AF10
display1[1]	PIN_AB12
display1[2]	PIN_AC12
display1[3]	PIN_AD11
display1[4]	PIN_AE11
display1[5]	PIN_V14
display1[6]	PIN_V13
display2[0]	PIN_V20
display2[1]	PIN_V21
display2[2]	PIN_W21
display2[3]	PIN_Y22
display2[4]	PIN_AA24
display2[5]	PIN_AA23
display2[6]	PIN_AB24
num[0]	PIN_N25
num[1]	PIN_N26
num[2]	PIN_P25
num[3]	PIN_AE14

Tabla 6: Asignación de pines para el ejercicio 4.3.

Para cada uno de los ejercicios de diseño se tendrá que entregar:

- **El código VHDL con comentarios incluidos.**
- **La simulación del diseño con todos los posibles estados de entradas y salidas.**
- **El archivo con extensión *.csv*, que corresponde a la asignación de pines.**

Además, se deberá mostrar al profesor de prácticas que los ejercicios de diseño se han volcado a la tarjeta DE2 y que funcionan correctamente.

Resolución dráctica 1

A continuación se resolverán los diferentes ejercicios que se han planteado en esta práctica.

3.1.- Puertas lógicas

El circuito lógico que hay que implementar es el siguiente:

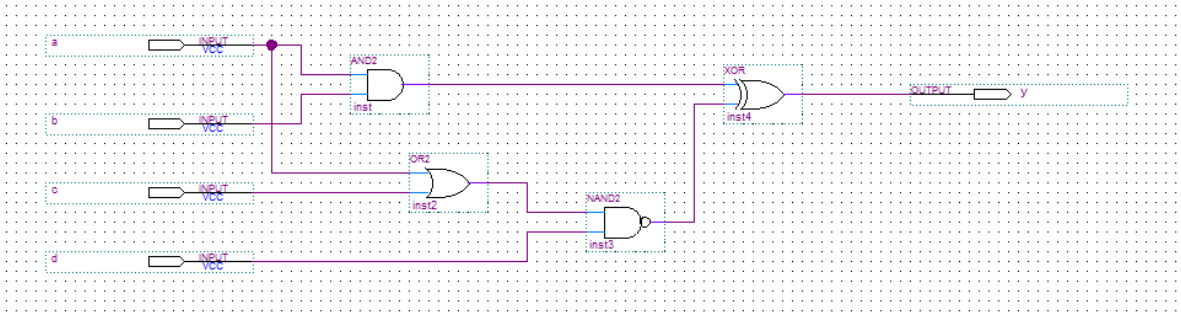

Figura 1: Esquemático del ejercicio 3.1.

a) Se ha obtenido la siguiente tabla de verdad:

a	b	c	d	y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Tabla 1: Tabla de verdad del ejercicio 3.1.

b) Ahora se procede a realizar la simulación y se obtienen estos resultados:

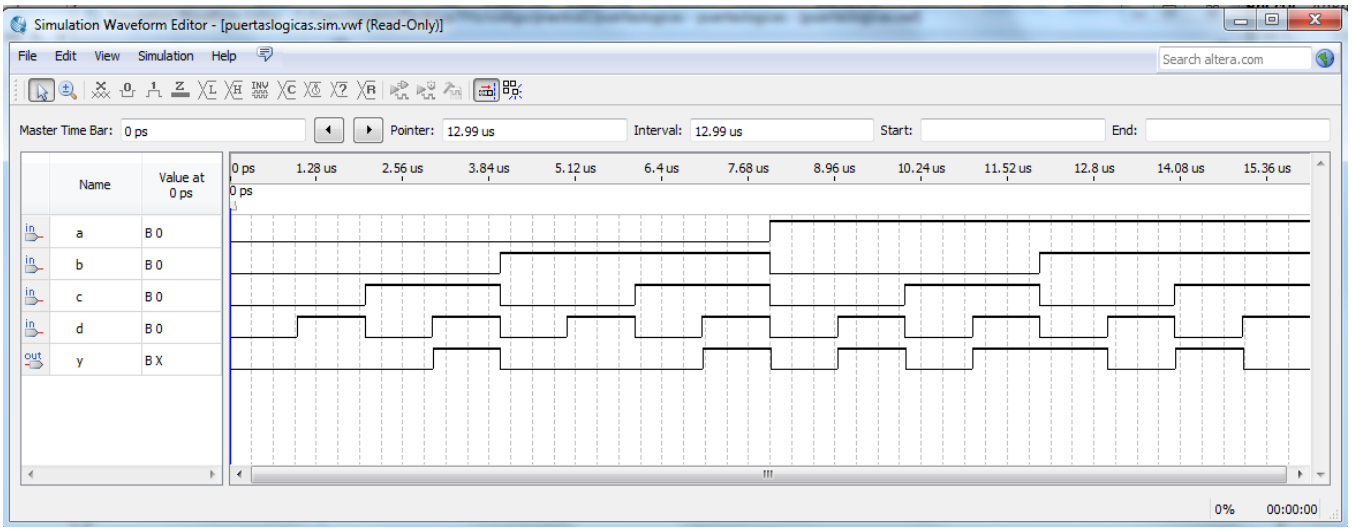

Figura 2: Resultados del circuito de puertas lógicas del ejercicio 3.1.

Si se analiza los resultados de la simulación, se observa que coinciden con la tabla de verdad, lo que implica que el código es correcto.

Ahora, se ha reducido el tiempo de simulación y se han llevado a cabo los dos tipos de simulaciones, ***Run Functional Simulation*** y ***Run Timing Simulation*** y se ha obtenido:

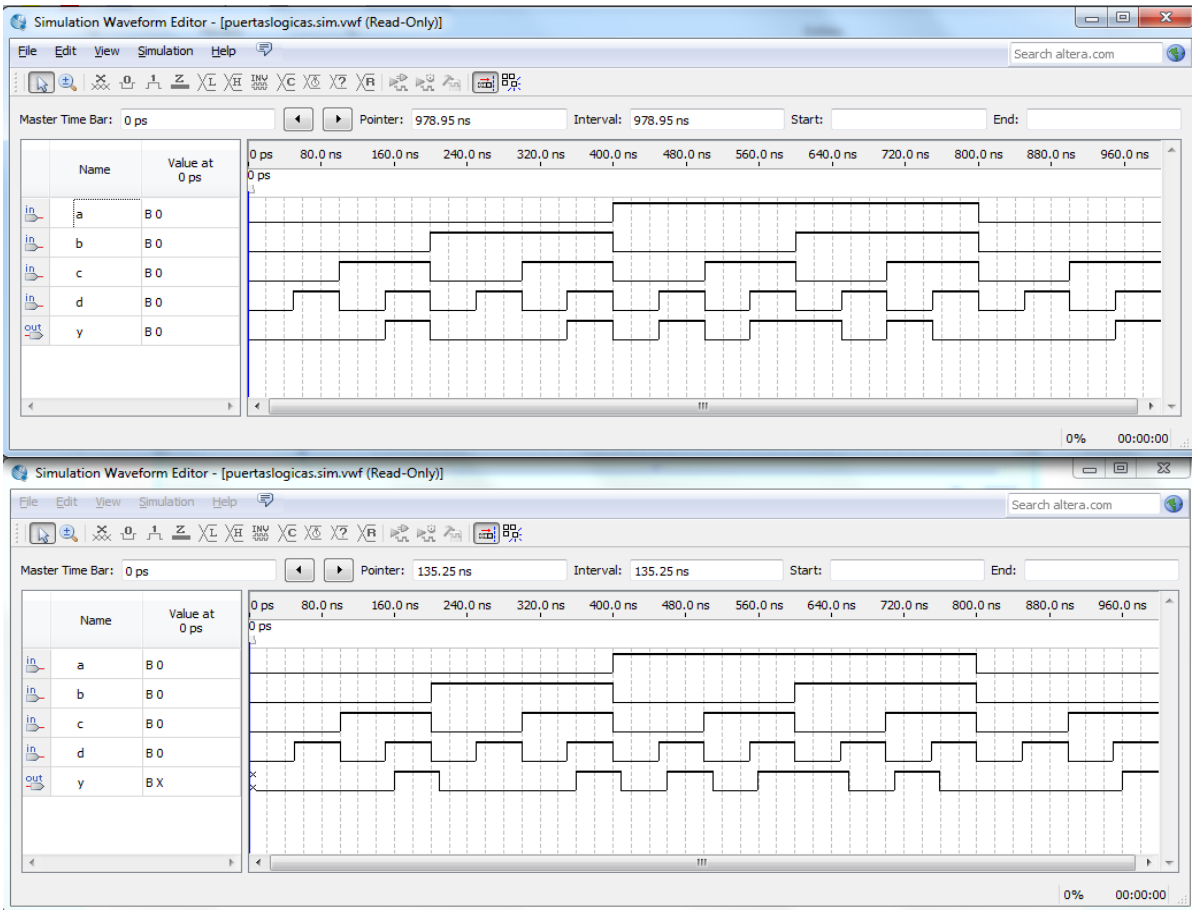

Figura 3: Comparativa entre las dos opciones de simulación

En la ventana de arriba se ha simulado mediante la opción ***Run Functional Simulation*** y se obtiene un resultado idéntico al anterior, pero en este caso con un tiempo de simulación menor. Sin embargo, en la ventana de abajo, la herramienta ***Run Timing Simulation*** realiza una simulación teniendo en cuenta los retrasos de las puertas lógicas. A diferencia que en la simulación anterior, que era de 16 us, esta última dura 1 us, por lo que se puede llegar apreciar que la salida *y* tiene un retraso de unos 10 ns.

3.2. Secuencia de leds

Se va a simular el código que se ha mostrado en el ejercicio 3.2., que implementa la secuencia de leds requerida en el ejercicio.

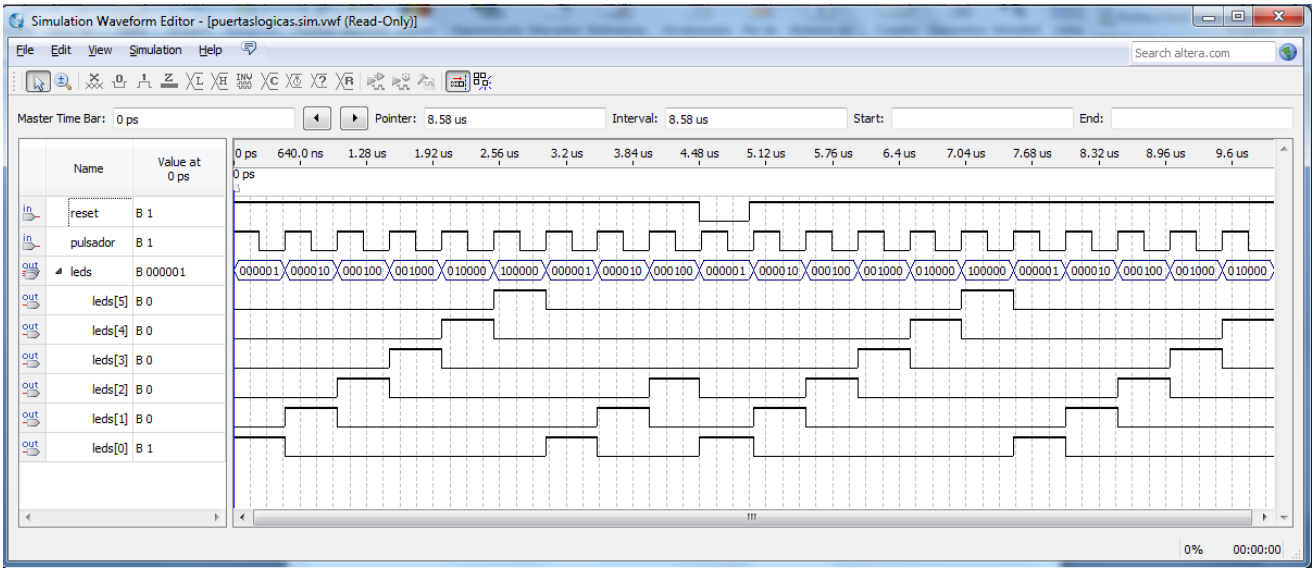

Figura 4: Resultados de la secuencia de leds del ejercicio 3.2.

En el cronograma mostrado se ve perfectamente como con cada pulsación el led encendido va avanzando hacia la izquierda, e incluso se aprecia como cuando el reset toma el valor'0', es decir el pulsador de reset está pulsado, la secuencia se reinicia y empieza de nuevo, encendiéndose el primer led.

4. Ejercicios de diseño

4.1. Secuencia de leds completada

El código que se ha escrito para realizar este ejercicio es el siguiente:

```
library IEEE;                                                    -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;                                 -- Paquetes de la librería estándar de la IEEE

entity secuencialed2 is
   Port (leds:out STD_LOGIC_VECTOR(5 downto 0);                  -- 6 leds de salida
         clk: in STD_LOGIC;                                      -- Canal de entrada del pulsador
          a: in STD_LOGIC;                                       -- Interruptor a
          b: in STD_LOGIC;                                       -- Interruptor b
          reset: in STD_LOGIC);                                  -- Canal de entrada del reset
end secuencialed2;

architecture izqdcha of secuencialed2 is                         -- Cabecera del programa
     signal conta: natural range 0 to 6;                         -- Declaración de señales
     signal ledout: STD_LOGIC_VECTOR (5 downto 0);
begin
   process(pulsador,reset,a,b)                                   -- Cabecera de proceso
   begin
           if reset='0' then                                     -- Sentencia if para detectar la pulsación del reset
              conta<=1;
           elsif pulsador'event and pulsador = '1' then -- Sentencia elsif para detectar el flanco de subida de la señal del
pulsador
              if b='0' then                                      -- Sentencia if para detectar si la secuencia permanece quieta o no
                 if a='0' then                                   -- Sentencia if para detectar si la secuencia va hacia la izq. o a la dcha.
                       if conta=6 then                           -- Sentencia if para detectar si cont debe inicializarse a 1
                          conta<= 1;                             -- o seguir aumentando
                       else
                          conta<=conta + 1;
                       end if;
                  else
                       if pulsador'event and pulsador = '1' then
                          if conta=1 then                        -- Sentencia if para detectar si cont debe reinicializarse a 6
                             conta <=6;                          -- o seguir disminuyendo
                          else
                             conta <=conta - 1;
                          end if;
                       end if;
                 end if;
              else
                 conta<=conta;
              end if;
           end if;
   end process;                                                  -- Final de proceso
   . . .
```

```
...
process(cont)                                   -- Cabecera de proceso
   begin
      if cont=1 then                            -- Sentencias if y elsif para detectar que led debe encenderse
        ledout<="000001";
      elsif cont=2 then
        ledout<="000010";
      elsif cont=3 then
        ledout<="000100";
      elsif cont=4 then
        ledout<="001000";
      elsif cont=5 then
        ledout<="010000";
      else
        ledout<="100000";
      end if;

      leds(5 downto 0) <= ledout(5 downto 0);   -- Traspaso de los datos de la señal ledout a la señal de salida
leds

   end process;                                 -- Final de proceso
 end izqdcha;                                   -- Final de programa
```

Una vez elaborado y compilado el código, se procede a simularlo. La siguiente figura muestra el cronograma de simulación:

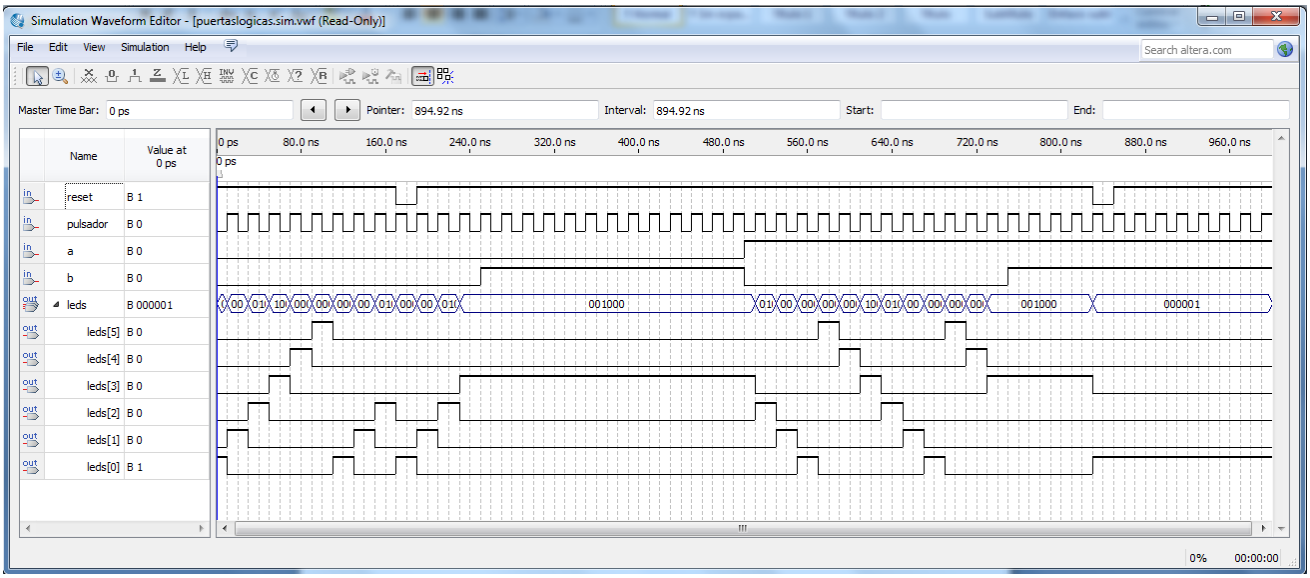

Figura 5: Resultados de la secuencia de leds del ejercicio 4.1.

Los resultados coinciden con lo esperado, si a y b son ’0’ la secuencia avanza hacia la izquierda, si a es ‘1’ y b ‘0’ avanza hacia la derecha y si b es ‘1’, independientemente de valor de a, la secuancia permanece quieta. Además la función del reset funciona perfectamente, reiniciando la secuencia cuando recibe el valor ‘0’.

4.2. Contador binario

Para implementar el contador binario que se pedía en este ejercicio se ha elaborado el siguiente código:

```
library IEEE;                                   -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;                -- Paquetes de la librería estándar de la IEEE

entity contbinario is
   port( pulsador: in STD_LOGIC;                -- Canal de entrada del pulsador
         leds: out STD_LOGIC_VECTOR (3 downto 0));  -- 4 leds de salida
end contbinario;

architecture binario of contbinario is          -- Cabecera del programa
   signal contador: STD_LOGIC_VECTOR (3 downto 0);  -- Declaración de señal
begin
   process(pulsador)                            -- Cabecera de proceso
   begin
      if pulsador='0' then                      -- Sentencia if para comprobar la pulsación del pulsador
         if contador<"1111" then                -- Sentencia if para detectar si el contador a llegado a 9
            contador<=contador + 1;             -- y debe reinicializarse o si debe seguir aumentando
         elsif contador="1111" then
            contador<="0000";
         end if;
      end if;
      leds<=contador;
   end process;                                 -- Final de proceso
end binario;                                    -- Final del programa
```

También se ha realizado la simulación del contador binario y el cronograma que se ha obtenido coincide perfectamente con el resultado esperado, como se puede ver a continuación.

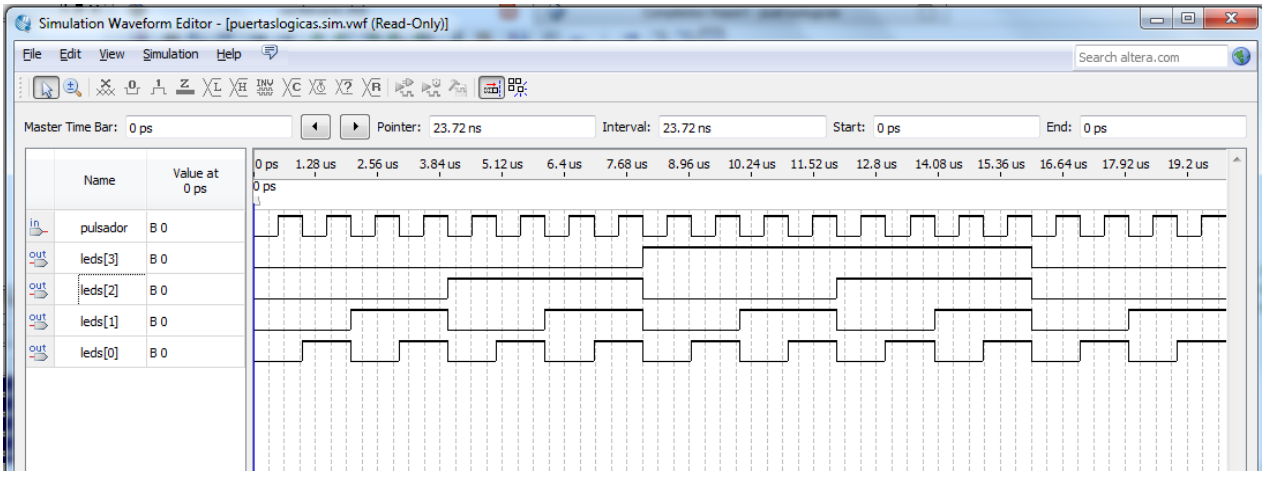

Figura 6: Resultados del contador binario del ejercicio 4.2.

Una observación importante es que en el cronograma se ve que las señales de los leds cambian cuando el pulsador cambia a 0, eso es debido a que el pulsador está normalmente a 1 y cuando se produce una pulsación cambia su estado 0.

4.3. Codificador 7 segmentos

La equivalencia entre los números binarios del 0 al 9 y sus correspondientes números cifrados para un display 7-segmentos es la siguiente:

Binario	7-segmentos
0	0000001
1	1001111
2	0010010
3	0000110
4	1001100
5	0100100
6	0100000
7	0001111
8	0000000
9	0001100

Tabla 2: Tabla de verdad del ejercicio 4.3

El código para diseñar el codificador 7-segmentos es el siguiente:

```
library IEEE;                                   -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;                   -- Paquetes de la librería estándar de la IEEE

entity segmentos7 is
port( num:in STD_LOGIC_VECTOR(3 downto 0);      -- 4 canales de entrada (interruptores)
      seg1,seg2: out STD_LOGIC_VECTOR(6 downto 0));  -- 2 vectores de salida de 7 bits cada uno
end segmentos7;

architecture codigosegmentos of segmentos7 is   -- Cabecera del programa
begin
...
```

```
...
  process (num)                                   -- Cabecera de proceso
  begin
    case num is                                   -- Sentencia case para pasar de numero binario
        when "0000"=>seg1<="1000000";             -- a numero en display 7-segmentos
                     seg2<="1000000";
        when "0001"=>seg1<="1111001";
                     seg2<="1000000";
        when "0010"=>seg1<="0100100";
                     seg2<="1000000";
        when "0011"=>seg1<="0110000";
                     seg2<="1000000";
        when "0100"=>seg1<="0011001";
                     seg2<="1000000";
        when "0101"=>seg1<="0010010";
                     seg2<="1000000";
        when "0110"=>seg1<="0000010";
                     seg2<="1000000";
        when "0111"=>seg1<="1111000";
                     seg2<="1000000";
        when "1000"=>seg1<="0000000";
                     seg2<="1000000";
        when "1001"=>seg1<="0011000";
                     seg2<="1000000";
        when "1010"=>seg1<="1000000";
                     seg2<="1111001";
        when "1011"=>seg1<="1111001";
                     seg2<="1111001";
        when "1100"=>seg1<="0100100";
                     seg2<="1111001";
        when "1101"=>seg1<="0110000";
                     seg2<="1111001";
        when "1110"=>seg1<="0011001";
                     seg2<="1111001";
        when "1111"=>seg1<="0010010";
                     seg2<="1111001";
    end case;
  end process;                                    -- Final de proceso
end codigosegmentos;                              -- Final de programa
```

A continuación se muestra la simulación y se observa como los resultados coinciden con la tabla.

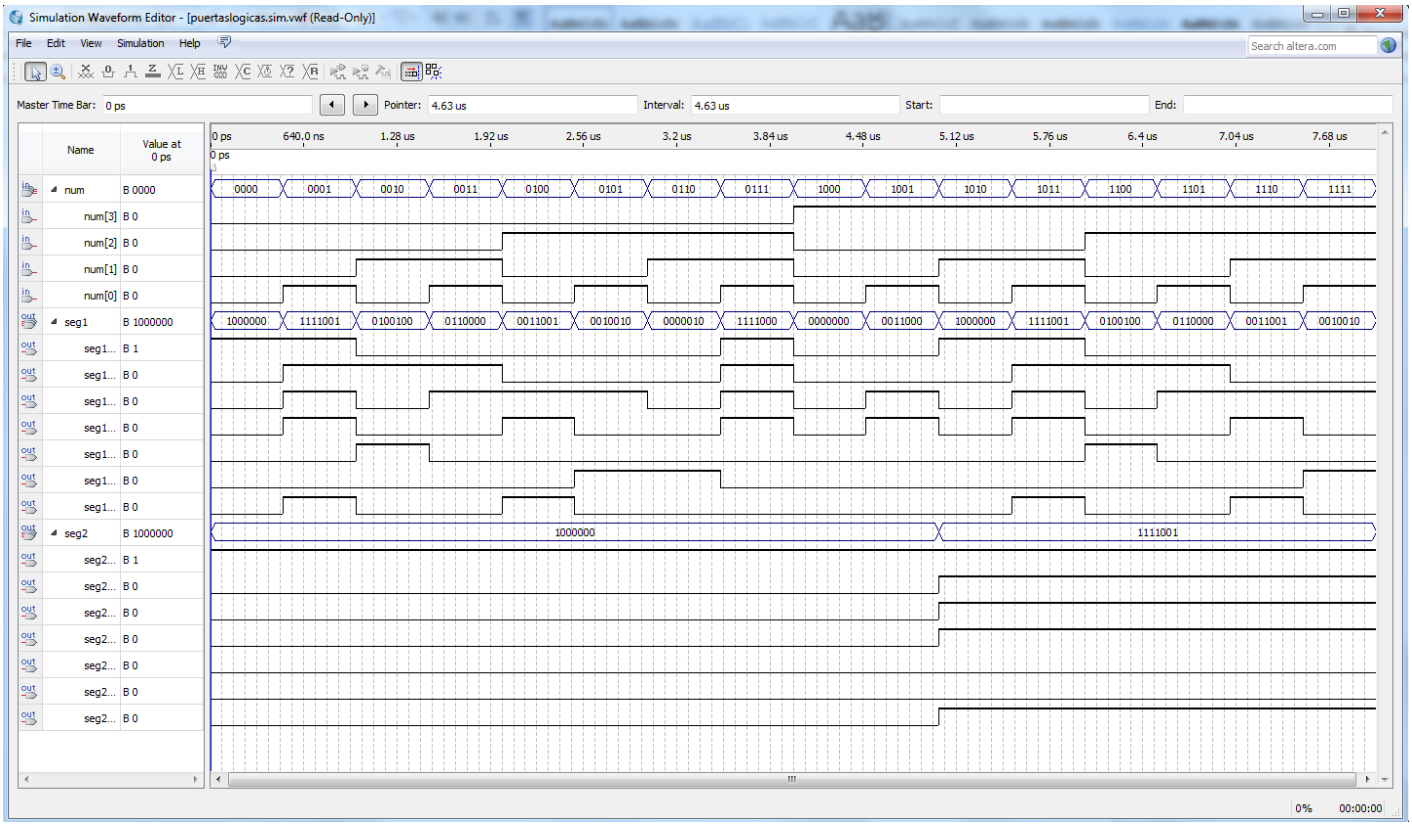

Figura 7: Resultados del codificador 7-segmentos del ejercicio 4.3.

num	seg1	seg2
0000	1000000	1000000
0001	1111001	1000000
0010	0100100	1000000
0011	0110000	1000000
0100	0011001	1000000
0101	0010010	1000000
0110	0000010	1000000
0111	1111000	1000000
1000	0000000	1000000
1001	0011000	1000000
1010	1000000	1111001
1011	1111001	1111001
1100	0100100	1111001
1101	0110000	1111001
1110	0011001	1111001
1111	0010010	1111001

Tabla 3: Tabla de verdad del ejercicio 4.3.

Práctica 2: Sistemas Lógicos Combinacionales

Un sistema combinacional es todo sistema digital cuyas salidas sean exclusivamente función del valor de sus entradas en un instante determinado, sin que de ninguna manera intervengan estados anteriores de las entradas o salidas.

Dentro de los sistemas combinacionales típicos tenemos:

- Lógicos:
  - Multiplexor y demultiplexor
  - Codificador y decodificador
  - Comparador
- Aritméticos:
  - Sumador

En este caso, esta práctica se centra en estudiar los diferentes sistemas combinacionales lógicos.

1. Objetivos

- Se verá cómo implementar, en lenguaje VHDL, multiplexores, demultiplexores/ decodificadores y comparadores.
- Utilizar estos elementos en diseños de circuitos digitales.

2. Material

- Ordenador personal con el software QUARTUS II disponible.
- Tarjeta educacional DE2
- Guión de prácticas

3. Desarrollo de la práctica

3.1. Multiplexor de 4 canales de entrada y 2 canales de control

A continuación, en la siguiente figura, se ve un esquema típico de un multiplexor de 4 canales de entrada y 2 canales de control. Este elemento es el que se va a implementar mediante lenguaje VHDL.

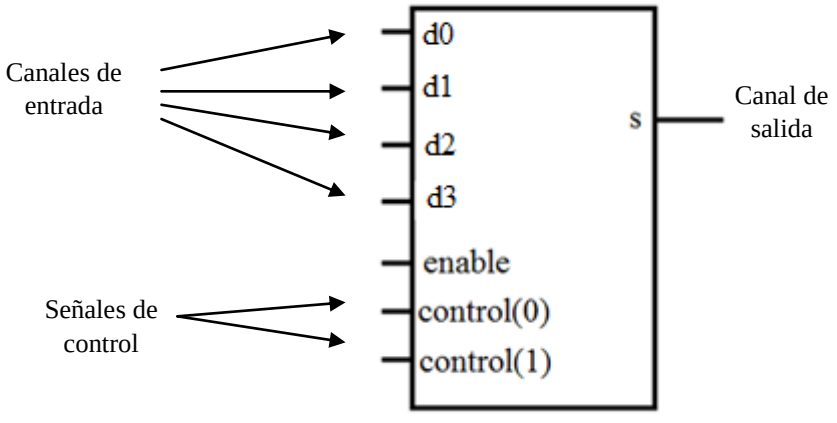

Figura 1: Multiplexor de 4 canales de entrada

El código es el siguiente:

```
library IEEE;                                          -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;                       -- Paquetes de la librería estándar de la IEEE

entity combinacional is port(
   d0,d1,d2,d3: in STD_LOGIC;                          -- 4 canales de entrada
   enable: in STD_LOGIC;                               -- Señal de enable
   control: in STD_LOGIC_VECTOR(1 downto 0);           -- 2 canales de control
   s: out STD_LOGIC);                                  -- 1 canal de salida
end combinacional;

architecture mult of combinacional is                  --Cabecera del programa
begin
 process(d0,d1,d2,d3,control,enable)                   --Cabecera de proceso
  begin
   if enable ='1' then                                 --Sentencia if para detectar si el enable está a 1
     s<='0';
    elsif enable='0' then                              --Sentencia elsif para detectar si el enable está a 0
     case control is                                   --Sentencia case para detectar que entrada se transmite a la salida
      when "00"=>s<=d0;
      when "01"=>s<=d1;
      when "10"=>s<=d2;
      when "11"=>s<=d3;
     end case;
   end if;
 end process;                                          --Final de proceso
end mult;                                              --Final del programa
```

Ejercicio

a) Simular el diseño y obtener el cronograma con todos los estados posibles de entradas y salidas.
b) Obtener la tabla de verdad y comprobar que coincide con los resultados de la simulación.

3.2. Multiplexor con sistema que gobierna las señales de control

Para controlar las señales del controlador se suele utilizar un sistema de conteo que va seleccionando los distintos canales de entrada secuencialmente cada cierto tiempo, este sistema cambia de canal a través de una señal de reloj. Así pues, dependiendo de la frecuencia de la señal de reloj se trabajará con un tiempo de multiplexado de los canales de entrada u otro, de esta forma, se consigue que la salida tome el valor de los diferentes canales de entrada cada cierto tiempo.

El sistema que se obtiene queda de la siguiente manera:

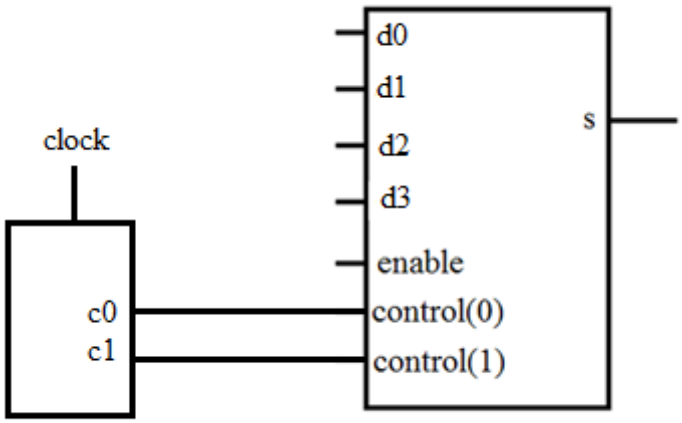

Figura 2: Multiplexor con sistema que gobierna las señales de control

El código para implementar el sistema descrito es el siguiente:

```
library IEEE;                                   -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;                -- Paquetes de la librería estándar de la IEEE

entity multiplexor2 is
   port( d0,d1,d2,d3: in STD_LOGIC;             -- 4 canales de entrada
         clock: in STD_LOGIC;                   -- Señal del reloj de 50 MHz
         enable: in STD_LOGIC;                  -- Señal de enable
         s: out STD_LOGIC);                     -- 1 canal de salida
end multiplexor2;

architecture multip of multiplexor2 is          --Cabecera del programa
   signal control: STD_LOGIC_VECTOR(1 downto 0); -- Declaración de señal
begin
   process(clock)                               -- Cabecera de proceso
   begin
      if clock'event and clock='1' then         -- Sentencia if para detectar el flanco de subida del reloj
         if control="11" then                   -- Sentencia if para comprobar si hay que reiniciar la señal
            control<="00";                      -- control o seguir aumentando su valor
         ...
```

```
        ...
        else
          control<=control+1;
        end if;
    end if;
  end process;                          -- Final de proceso
  process(d0,d1,d2,d3,control,enable)   --Cabecera de proceso
   begin
    if enable ='1' then                 --Sentencia if para detectar si el enable está a 1
        s<='0';
    elsif enable='0' then               --Sentencia elsif para detectar si el enable está a 0
        case control is                 --Sentencia case para detectar que entrada se transmite a la salida
          when "00"=>s<=d0;
          when "01"=>s<=d1;
          when "10"=>s<=d2;
          when "11"=>s<=d3;
        end case;
    end if;
  end process;                          --Final de proceso
end multip;                             --Final del programa
```

En este ejemplo se utiliza directamente la señal de reloj que ofrece la tarjeta DE2, que es tiene una frecuencia de 50 MHz. Esta frecuencia se puede cambiar y trabajar con el tiempo de multiplexado que se quiere, solamente es necesario implementar un divisor de frecuencia.

A continuación se muestra un divisor de frecuencia de 1 Hz:

```
entity divisor_frec is
   port( clock: in STD_LOGIC);                  -- Señal del reloj de 50 MHz
end divisor_frec;
architecture divisor1hz of divisor_frec is      -- Cabecera del programa
   signal cont1s: natural range 0 to 25000000;  -- Declaración de señal
begin
   process(clock)                               -- Cabecera de proceso
   begin
      if clock'event and clock = '1' then       -- Sentencia if para detecar el flanco de subida del reloj
          if cont1s=24999999 then               -- Sentencia if para comprobar si hay que reiniciar
             cont1s<=0;
          else
             cont1s<=cont1s+1;
          end if;
      end if;
   end process;                                 -- Final de proceso
end divisor1hz;                                 -- Final del programa
```

Ejercicio

a) Simular el ejemplo del multiplexor con el sistema que se le ha añadido y obtener el cronograma con todos los estados posibles de entradas y salidas.

3.3. Demultiplexor de 4 canales de salida y 2 canales de control

Aquí se muestra una imagen que representa el esquema típico de un demultiplexor de 4 canales de salida y 2 canales de control.

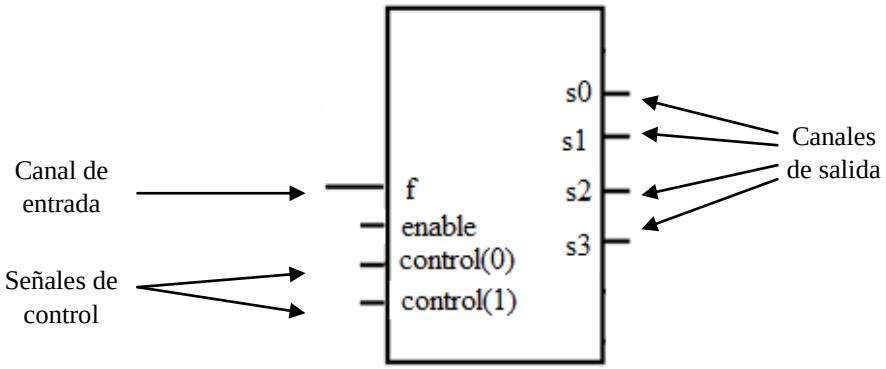

Figura 3: Demultiplexor de 4 canales de salida

El código para implementar el elemento descrito es:

```
library IEEE;                                          -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;                       -- Paquetes de la librería estándar de la IEEE
entity demultiplexor is
   port( enable: in STD_LOGIC;                         -- Señal de enable
         control: in STD_LOGIC_VECTOR(1 downto 0);     -- 2 canales de control
         f: in STD_LOGIC;                              -- 1 canal de entrada
         s0,s1,s2,s3: out STD_LOGIC);                  -- 4 canales de salida
end demultiplexor;

architecture demul of demultiplexor is                 -- Cabecera del programa
begin
  process(enable,control,f)                            -- Cabecera de proceso
  begin
         if enable='1' then                            --Sentencia if para detectar si el enable está a 1
            s0<='0';
            s1<='0';
            s2<='0';
            s3<='0';
         elsif enable='0' then                         --Sentencia elsif para detectar si el enable está a 0
         case control is                               --Sentencia case para detectar a que salida se transmite la entrada
            when "00"=>s0<=f;
            when "01"=>s1<=f;
            when "10"=>s2<=f;
            when "11"=>s3<=f;
         end case;
         end if;
         end process;                                  -- Final de proceso
end demul;                                             -- Final de programa
```

Ejercicio

b) Simular el diseño y obtener el cronograma con todos los estados posibles de entradas y salidas.
c) Obtener la tabla de verdad y comprobar que coincide con los resultados de la simulación.

3.4. Comparador de dos bits

En la figura se muestra un ejemplo general de un comparador de 2 palabras de n bits cada una, en el que hay tres salidas, las cuales indican si A es mayor que B, si A es menor que B o si ambas son iguales.

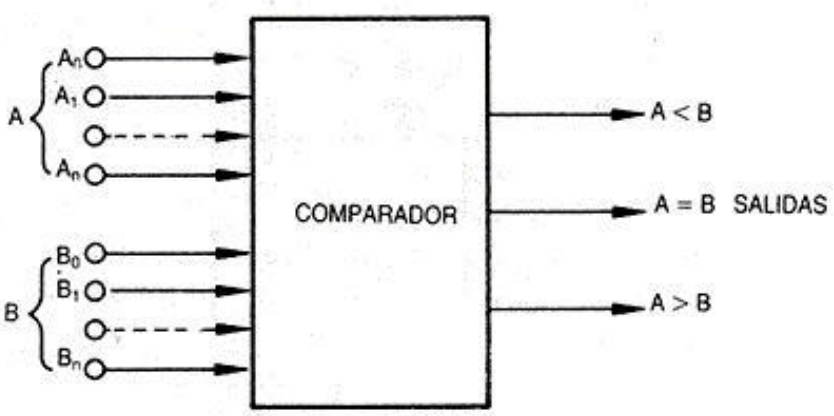

Figura 4: Comparador de dos palabras de n bits

En este ejemplo se va a ver un comparador tan sólo dos bits, el cual tiene dos canales de entrada, uno el bit 'a' y el otro el bit 'b', un canal de enable y tres salidas, las mismas que las del comparador general de la figura 3.

Se realizará primero este diseño sencillo para entender fácilmente como implementar un comparador mediante código VHDL

El código para realizar el comparador de dos bits es el siguiente:

```
library IEEE;                                   -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;                -- Paquetes de la librería estándar de la IEEE
entity comparador is
   port(  a,b: in STD_LOGIC;                    -- 2 canales de entrada
          enable: in STD_LOGIC;                 -- Señal de enable
          mayor,menor,igual: out STD_LOGIC);    -- 3 canales de salida
end comparador;
architecture comp2bits of comparador is         -- Cabecera del programa
begin
...
```

```
...
   process(enable,a,b)                          -- Cabecera de proceso
   begin
         if a>b then                            -- Sentencia if para detectar si el bit a es mayor que el bit b
            mayor<='1';
            menor<='0';
            igual<='0';
         elsif a<b then                         -- Sentencia elsif para detectar si el bit a es menor que el bit b
            mayor<='0';
            menor<='1';
            igual<='0';
         elsif a=b then                         -- Sentencia elsif para detectar si el bit a y el bit b son iguales
            mayor<='0';
            menor<='0';
            igual<='1';
         end if;
   end process;                                 -- Final de proceso
end comp2bits;                                  -- Final del programa
```

Ejercicio

a) Simular el diseño y obtener el cronograma con todos los estados posibles de entradas y salidas.
b) Obtener la tabla de verdad y comprobar que coincide con los resultados de la simulación.

4. Ejercicios de diseño

4.1. Demultiplexado de displays 7-segmentos

Se pide mostrar la palabra “HOLA” en 4 displays 7-segmentos, de manera que en cada display aparezca una letra. Además, los displays deberán estar demultiplexados, es decir, se utilizará una sola variable para transmitir los datos a los diferentes displays.

Realizar el diseño del ejercicio de modo que el tiempo de demultiplexado sea el adecuado para que en los displays parezca que se muestra constantemente la palabra y que no se perciba que las letras se muestran de una en una.

Para este ejercicio se deberá entregar el código VHDL con sus respectivos comentarios y el archivo con extensión *.csv*, que corresponde a la asignación de pines.

4.2. Comparador de 4 palabras de 4 bits

Implementar un comparador que compare 4 palabras (a,b,c,d) de 4 bits cada una y que muestre por un display 7-segmentos el valor, en decimal, de la mayor de ellas. En el caso de haya varias palabras que tengan el mayor valor, que se muestre dicho valor por el display como en un caso normal.

Probar los siguientes casos:

1) a=0000; b=1001; c=1100; d=0100
2) a=1000; b=1111; c=0010; d=1111
3) a=0111; b=0001; c= 0101; d=0011

Para este ejercicio se deberá entregar el código VHDL con sus respectivos comentarios, el cronograma de simulación con todos los posibles estados de entradas y salidas y el archivo con extensión *.csv*, que corresponde a la asignación de pines.

4.3. Detector de números primos

Completar el diseño del contador binario de la práctica anterior con un detector de números primos, de forma que el valor del contador se irá comparando para ver si es un número primo o no. Si es un número primo la salida del comparador se activará y se deberá encender un led verde y si no es número primo el led verde se mantendrá apagado.

Como el contador binario contaba hasta números de 4 bits, obviamente se detectarán los números primos comprendidos entre el 0 (0000) y el 15 (1111).

Para este ejercicio se deberá entregar el código VHDL con sus respectivos comentarios, el cronograma de simulación con todos los posibles estados de entradas y salidas y el archivo con extensión *.csv*, que corresponde a la asignación de pines.

Se deberá mostrar al profesor de prácticas que los ejercicios de diseño se han volcado a la tarjeta DE2 y que funcionan correctamente.

Resolución de la práctica 2

3.1. Multiplexor de 4 canales de entrada y 2 canales de control

Se ha simulado el diseño del multiplexor de 4 canales de entrada y 2 canales de control y se ha conseguido el siguiente cronograma:

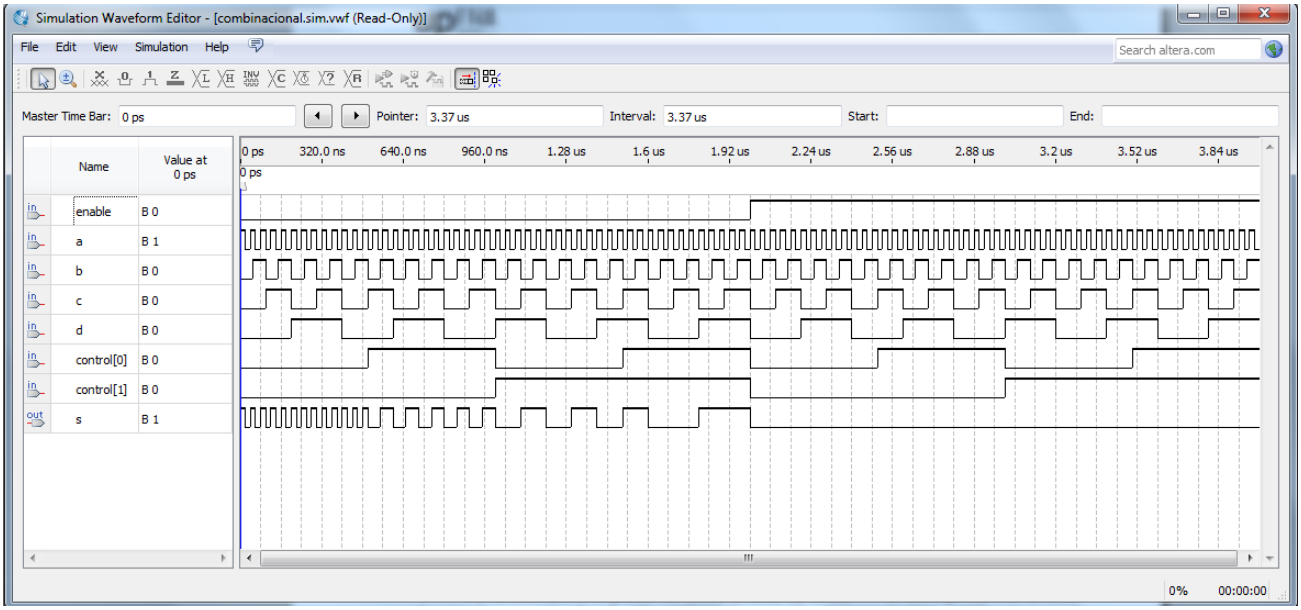

Figura 1: Resultados del multiplexor de 4 canales de entrada y 2 canales de control

Se puede observar que cuando el enable está a '1' el multiplexor no obedece a las entradas de control y que cuando está a '0' la salida toma el valor de la entrada que interpone la señal de control.

De aquí se obtiene la tabla de verdad y se ve que coincide con los resultados esperados.

enable	control(0)	control(1)	s
0	0	0	a
0	0	1	b
0	1	0	c
0	1	1	d
1	X	X	0

Tabla 1: Tabla de verdad del ejercicio 3.1.

3.2. Multiplexor con sistema que gobierna las señales de control

Se ha simulado el diseño del ejercicio 3.2. que se ha mostrado anteriormente con varios casos para ver que su funcionamiento es correcto y que el sistema que se le ha añadido controla que canal se ve a la salida.

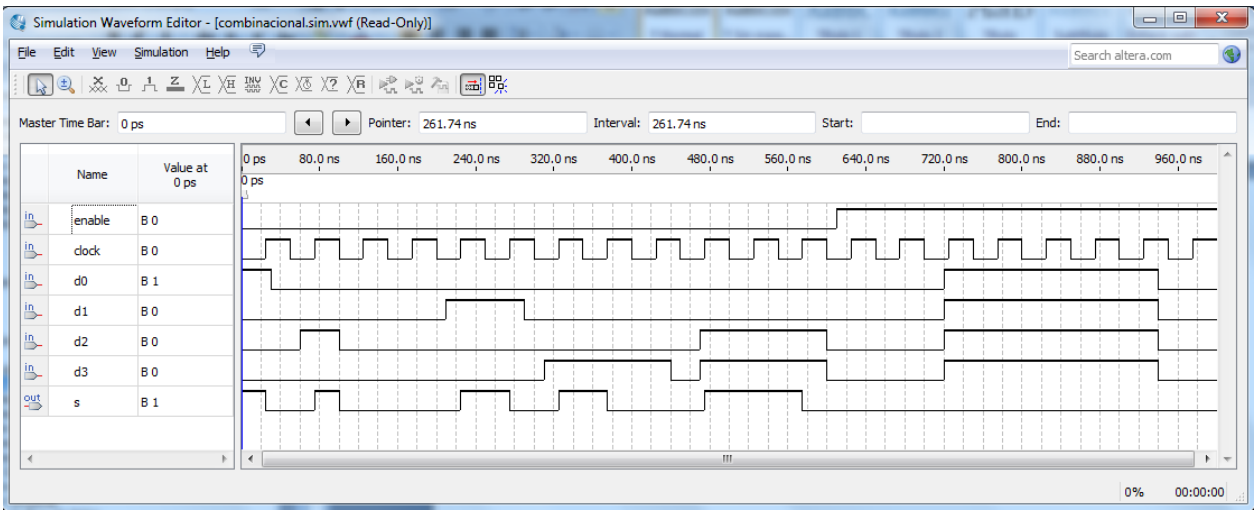

Figura2: Resultados del multiplexor del ejercicio 3.2.

Se observa claramente como con cada flanco de subida de la señal de reloj la señal de control aumenta y el canal que se ve a la salida cambia. Además, se ve como cuando el enable toma el valor '1' la salida es siempre '0' y no tiene en cuenta el valor ni de las señales de control ni de los canales de entrada.

3.3. Demultiplexor de 4 canales de salida y 2 canales de control

El cronograma que se ha obtenido tras la simulación del multiplexor de 4 canales de salida es el siguiente:

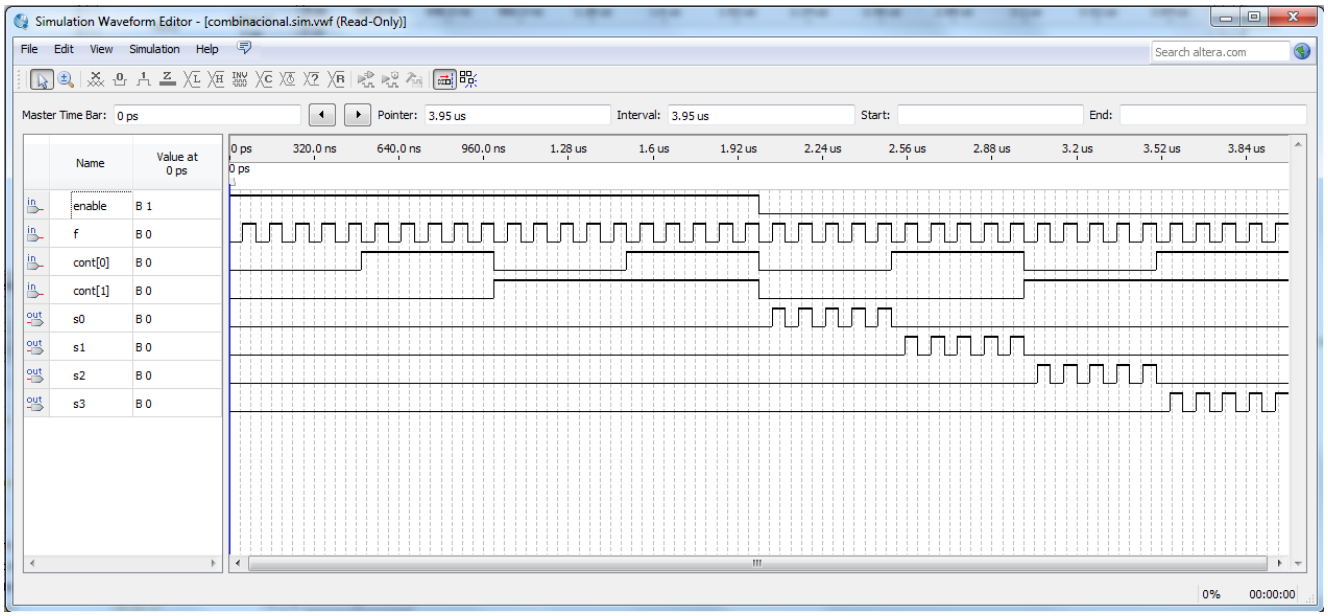

Figura 3: Resultados del demultiplexor del ejercicio 3.2.

Se observa como los resultados obtenidos son iguales que los calculados teóricamente.

enable	control(0)	control(1)	f	s0	s1	s2	s3
0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	1	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	1	0
0	1	1	0	0	0	0	0
0	1	1	1	0	0	0	1
1	X	X	X	0	0	0	0

Tabla 2: Tabla de verdad del ejercicio 3.2.

3.4. Comparador de dos bits

Se ha implementado un comparador de dos bits (a y b) y mediante el Editor de Señales se han obtenido los siguientes resultados:

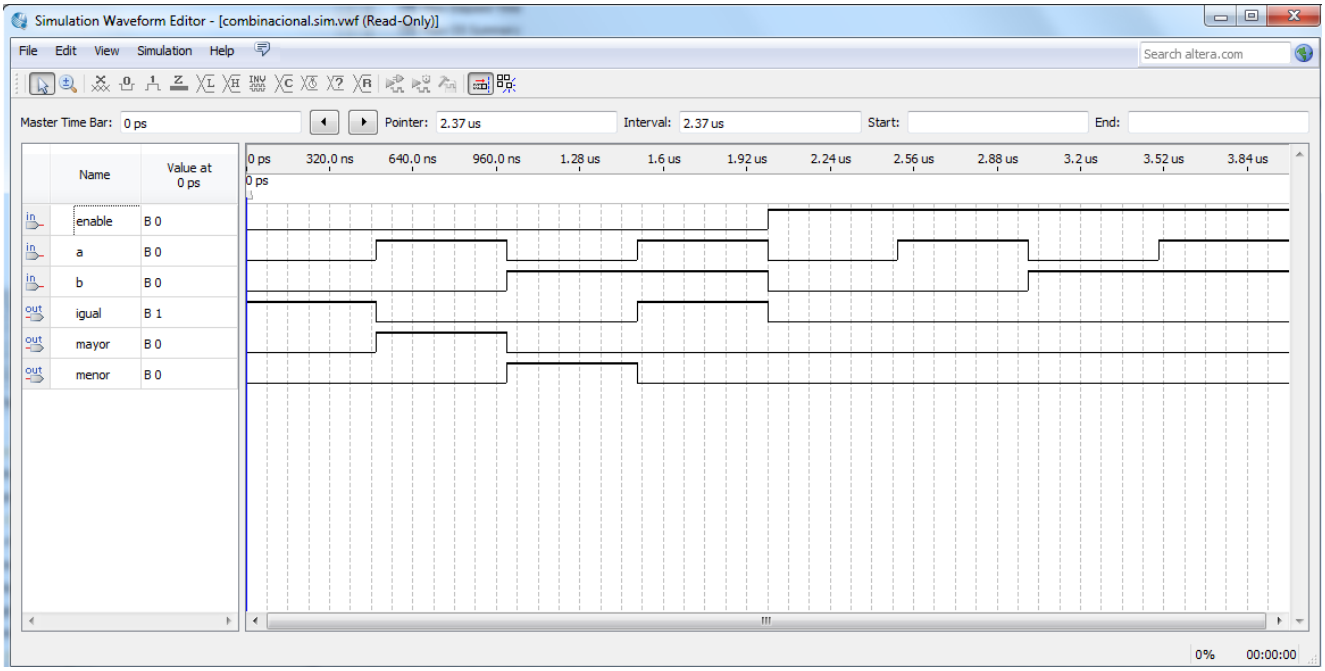

Figura 4: Resultados del comparador del ejercicio 3.3.

De la imagen anterior se ha sacado la siguiente tabla de verdad, en donde se comprueba que el comparador diseñado funciona correctamente.

enable	a	b	igual (a=b)	mayor (a>b)	menor (a<b)
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	1	0	0
1	X	X	0	0	0

Tabla 3: Tabla de verdad del ejercicio 3.3.

Ejercicios de diseño

4.1. Demultiplexado de displays 7-segmentos

En el caso de la tarjeta educacional DE2 no es necesario realizar una demultiplexación de los displays, ya que se puede acceder a cada uno de ellos simultáneamente sin necesidad de combinarlos para trasnimitir por un único medio de transmisión. Sin embargo, uno de los objetivos principales de esta práctica es aprender bien el funcionamiento de un demultiplexor, por lo que mediante este ejercicio se pretende simular el caso de que los cuatro displays que se utilizan compartan un mismo medio de transmisión, a través del cual reciben los datos.

Se ha elegido un tiempo de demultiplexación de 2 milisegundos, de esta forma parecerá que los 4 displays están constantemente mostrando y no se apreciará el que sólo muestran de uno en uno, este es el objetivo del demultiplexado de los displays.

Para llevar a cabo el diseño requerido se ha elaborado el siguiente diseño:

```
library IEEE;                                                    -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;                                 -- Paquetes de la librería estándar de la IEEE
entity mult7seg2milisegundos is
  port( display1,display2,display3,display4: out STD_LOGIC_VECTOR (6 downto 0);   -- 4 vectores de salida de 7 bits cada uno
        clock: in STD_LOGIC);                                    -- 1 canal de entrada para el reloj de 50 MHz
end mult7seg2milisegundos;
architecture multdisplay2ms of mult7seg2milisegundos is          -- Cabecera del programa
  signal cont2ms: natural range 0 to 399999;                     -- Declaración de señales
  signal letra: STD_LOGIC_VECTOR (6 downto 0);
begin
  process(clock)                                                 -- Cabecera de proceso
  begin
    if clock'event and clock = '1' then                          -- Sentencia if para detectar el flanco de subida del reloj
        if cont2ms=399999 then                                   -- Sentencia if para comprobar si hay que reiniciar
          cont2ms<=0;                                            -- el contador o seguir contando flancos
        else
          cont2ms<=cont2ms+1;
        end if;
    end if;
  end process;                                                   -- Final de proceso
  process(cont2ms)                                               -- Cabecera de proceso
  begin
    if cont2ms<99999 then                                        -- Sentencia if para detectar que letra debe mostrarse
        letra<="0001001";
    elsif cont2ms<199999 then
        letra<="1000000";
    elsif cont2ms<299999 then
        letra<="1000111";
    elsif cont2ms<399999 then
        letra<="0001000";
        ...
```

```
...
      end if;
   end process;                                    -- Final de proceso
   process(letra)                                  -- Cabecera de proceso
   begin
      if letra="0001001" then                      -- Sentencia if para detectar si el display1 debe mostrar
            display1<=letra;                       -- la "H" o no debe mostrar nada
      else
            display1<="1111111";
      end if;
      if letra="1000000" then                      -- Sentencia if para detectar si el display2 debe mostrar
            display2<=letra;                       -- la "O" o no debe mostrar nada
      else
            display2<="1111111";
      end if;
      if letra="1000111" then                      -- Sentencia if para detectar si el display3 debe mostrar
            display3<=letra;                       -- la "L" o no debe mostrar nada
      else
            display3<="1111111";
      end if;
      if letra<="0001000" then                     -- Sentencia if para detectar si el display4 debe mostrar
            display4<=letra;                       -- la "A" o no debe mostrar nada
      else
            display4<="1111111";
      end if;
   end process;                                    -- Final de proceso

end multdisplay2ms;                                -- Final de programa
```

4.2. Comparador de 4 palabras de 4 bits

El código para el comparador que se pide en este ejercicio es el siguiente:

```
library IEEE;                                      -- Librerí-a estándar de la IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;                   -- Paquetes de la librería estándar de la IEEE

entity comparador4palabras is
   port( a,b,c,d: in STD_LOGIC_VECTOR (3 downto 0);   -- 4 vectores de entrada e 4 bits cada uno
         display1,display2: out STD_LOGIC_VECTOR (6 downto 0));   -- 2 vectores de salida de 7 bits cada uno
end comparador4palabras;

architecture palabras4bits of comparador4palabras is   -- Cabecera del programa
   signal mayor: STD_LOGIC_VECTOR (3 downto 0);        -- Declaración de sñal
begin
...
```

```
...
  process(a,b,c,d)                          -- Cabecera de proceso
  begin
    if a>b then                             -- Sentencia if para comprobar si a es mayor, menor o igual que b
       if a>c then                          -- Sentencia if para comprobar si a es mayor, menor o igual que c
         if a>d then                        -- Sentencia if para comprobar si a es mayor, menor o igual que d
           mayor<=a;
         elsif a=d then
           mayor<=a;
         else
           mayor<=d;
         end if;
       elsif a=c then
         if a>d then                        -- Sentencia if para comprobar si a es mayor, menor o igual que d
           mayor<=a;
         elsif a=d then
           mayor<=a;
         else
           mayor<=d;
         end if;
       else
         if c>d then                        -- Sentencia if para comprobar si c es mayor, menor o igual que d
           mayor<=c;
         elsif c=d then
           mayor<=c;
         else
           mayor<=d;
         end if;
       end if;
    elsif a=b then
       if a>c then                          -- Sentencia if para comprobar si a es mayor, menor o igual que c
         if a>d then                        -- Sentencia if para comprobar si a es mayor, menor o igual que d
           mayor<=a;
         elsif a=d then
           mayor<=a;
         else
           mayor<=d;
         end if;
       elsif a=c then
         if a>d then                        -- Sentencia if para comprobar si a es mayor, menor o igual que d
           mayor<=a;
         elsif a=d then
           mayor<=a;
         else
           mayor<=d;
         end if;
       else
         if c>d then                        -- Sentencia if para comprobar si c es mayor, menor o igual que d
           mayor<=c;
       ...
```

```
        ...
          elsif c=d then
             mayor<=c;
          else
             mayor<=d;
          end if;
       end if;
    else
       if b>c then                          -- Sentencia if para comprobar si b es mayor, menor o igual que c
          if b>d then                       -- Sentencia if para comprobar si b es mayor, menor o igual que d
             mayor<=b;
          elsif a=d then
             mayor<=b;
          else
             mayor<=d;
          end if;
       elsif b=c then
          if b>d then                       -- Sentencia if para comprobar si b es mayor, menor o igual que d
             mayor<=b;
          elsif b=d then
             mayor<=b;
          else
             mayor<=d;
          end if;
       else
          if c>d then                       -- Sentencia if para comprobar si c es mayor, menor o igual que d
             mayor<=c;
          elsif c=d then
             mayor<=c;
          else
             mayor<=d;
          end if;
       end if;
   end if;
end process;                                -- Final de proceso
process (mayor)                             -- Cabecera de proceso
begin
   case mayor is                            -- Sentencia case para pasar de numero binario
       when "0000"=>display1<="1000000";    -- a numero en display 7-segmentos
                    display2<="1000000";
       when "0001"=>display1<="1111001";
                    display2<="1000000";
       when "0010"=>display1<="0100100";
                    display2<="1000000";
       when "0011"=>display1<="0110000";
                    display2<="1000000";
       when "0100"=>display1<="0011001";
                    display2<="1000000";
       when "0101"=>display1<="0010010";
                    display2<="1000000";
        ...
```

```
        ...
        when "0110"=>display1<="0000010";
                     display2<="1000000";
        when "0111"=>display1<="1111000";
                     display2<="1000000";
        when "1000"=>display1<="0000000";
                     display2<="1000000";
        when "1001"=>display1<="0011000";
                     display2<="1000000";
        when "1010"=>display1<="1000000";
                display2<="1111001";
        when "1011"=>display1<="1111001";
                     display2<="1111001";
        when "1100"=>display1<="0100100";
                     display2<="1111001";
        when "1101"=>display1<="0110000";
                     display2<="1111001";
        when "1110"=>display1<="0011001";
                     display2<="1111001";
        when "1111"=>display1<="0010010";
                     display2<="1111001";
    end case;
  end process;                                    -- Final de proceso
end palabras4bits;                                -- Final de programa
```

Tras realizar el diseño, se han realizado las simulaciones de los tres casos que se han pedido. En la siguiente figura se pueden ver los resultados.

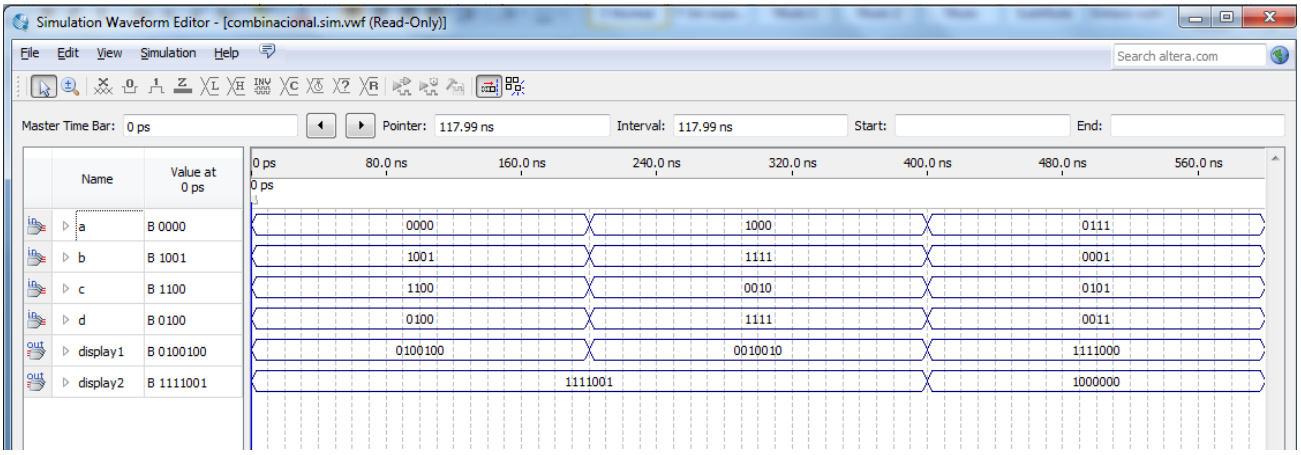

Figura 4: Resultados del comparador del ejercicio 4.2.

A continuación, en la siguiente tabla se recogen los tres casos propuestos anteriormente y además se incluyen los valores, obtenidos en la simulación, que deben tomar los displays en cada uno de los casos.

a	b	c	d	valor máximo	display1	display2
0000	1001	1100	0100	c	0100100	1111001
1000	1111	0010	1111	b y d	0010010	1111001
0111	0001	0101	0011	a	1111000	1000000

Tabla 4: Tabla de verdad del ejercicio 4.2.

4.3. Detector de números primos

Se ha realizado un comparador que detecta si el valor de la palabra de entrada es un número primo o no. Las palabras que son de 4 bits, por lo que el valor, en decimalo, de las palabras va desde el 1 hasta el 15, por lo que los números primos comprendidos entre esos valores son: 2, 3, 5, 7, 11 y 13.

Así pues, la tabla de verdad de nuestro circuito será la siguiente:

contador	ledverde
0000	0
0001	0
0010	1
0011	1
0100	0
0101	1
0110	0
0111	1
1000	0
1001	0
1010	0
1011	1
1100	0
1101	1
1110	0
1111	0

Tabla 5: Tabla de verdad del ejercicio 4.3.

Se ha realizado la simulación para comprobar si el diseño funciona correctamente, y tal y como se ve en la siguiente figura, los resultados obtenidos son idénticos a los que se esperaban.

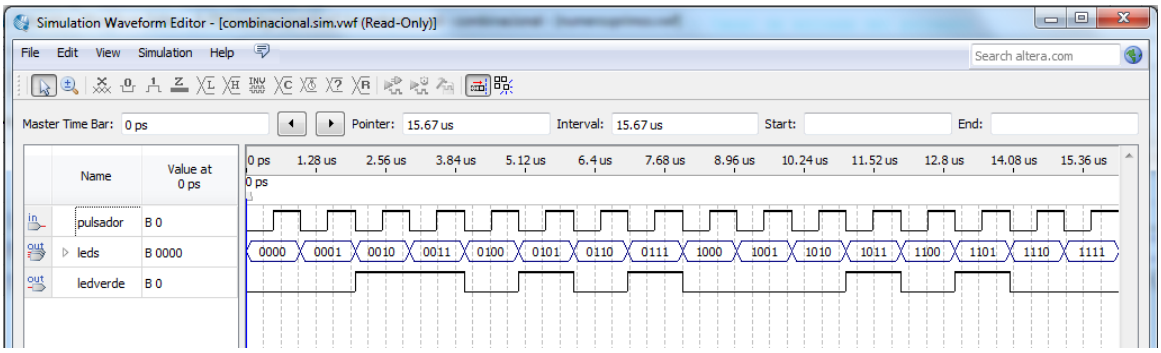

Figura 5: Resultados del detector de números primos del ejercicio 4.3.

Ahora se muestra el código que se ha realizado para cumplir con los requisitos que pedía el ejercicio:

```
library IEEE;                                          -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;                       -- Paquetes de la librería estándar de la IEEE
entity numerosprimos is
   port(pulsador: in STD_LOGIC;                        -- Canal de entrada del pulsador
        leds: out STD_LOGIC_VECTOR (3 downto 0);       -- 4 leds de salida para mostrar el número en binario
        ledverde: out STD_LOGIC);                      -- 1 led de salida para mostrar si el número es primo o no
end numerosprimos;
architecture detectorprimos of numerosprimos is        -- Cabecera del programa
        signal contador: STD_LOGIC_VECTOR (3 downto 0); -- Declaración de señal
begin
   process(pulsador)                                   -- Cabecera de proceso
   begin
     if pulsador='0' then                              -- Sentencia if para comprobar la pulsación del pulsador
        if contador<"1111" then                        -- Sentencia if para detectar si el contador a llegado a 9
          contador<=contador + 1;                      -- y debe reinicializarse o si debe seguir aumentando
        elsif contador="1111" then
          contador<="0000";
        end if;
     end if;
     leds<=contador;                                   -- Traspaso de datos de la señal contador a los canales de salida de leds
   end process;                                        -- Final de proceso

   process(contador)                                   -- Cabecera de proceso
   begin
     case contador is                                  -- Sentencia case para detectar los números primos
        when "0010"=>ledverde<='1';
        when "0011"=>ledverde<='1';
        when "0101"=>ledverde<='1';
        when "0111"=>ledverde<='1';
        when "1011"=>ledverde<='1';
        when "1101"=>ledverde<='1';
        when others=>ledverde<='0';
     end case;
        end process;                                   -- Final de proceso
end detectorprimos;                                    -- Final del programa
```

Práctica 3: Circuitos Aritméticos

1. Objetivos

- Diseñar diversos circuitos lógicos a través de los cuales conseguimos implementar operaciones aritméticas básicas con números binarios.
- Ver las diferentes formas de diseñar mediante lenguaje VHDL, que son las descripciones funcionales, estructurales y de flujo de datos.
- Utilizar para los diseños la definición y referencia de componentes, que harán nuestros diseños más sencillos y los dotarán de una jerarquía.

2. Material

- Ordenador personal con el software QUARTUS II disponible.
- Tarjeta educacional DE2
- Guión de prácticas

3. Desarrollo de la práctica

3.1. Sumador binario completo

El sumador completo es un circuito lógico que calcula la operación de suma de números binarios. Este circuito es capaz de sumar un bit más otro y además tiene en cuenta el acarreo anterior, por lo que combinando varios sumadores completos se pueden llevar a cabo sumas de números de más de un bit.

Para sumar dos bits, hay que seguir las reglas de la suma binaria:

Suma	Acarreo
0 + 0 = 0	0
0 + 1= 1	0
1 + 0= 1	0
1 + 1= 0	1

Tabla 1: Suma binario con acarreo

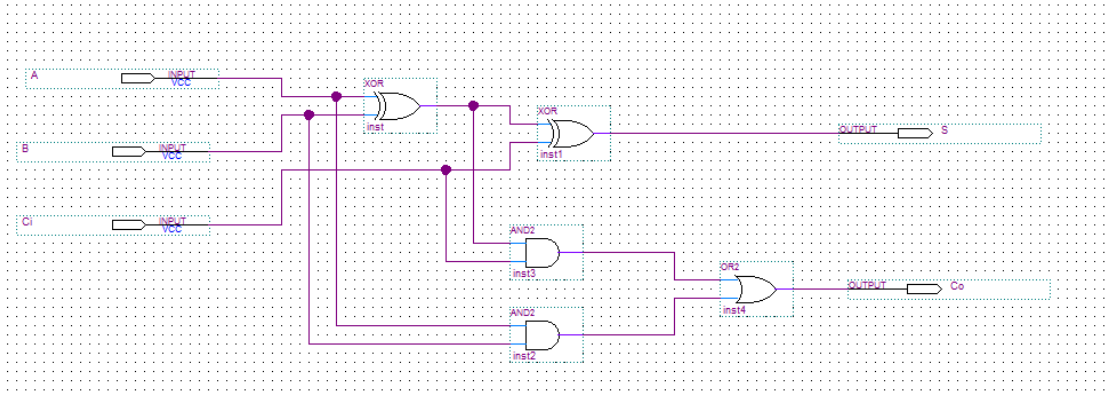

Figura 1: Circuito lógico del sumador completo

Las entradas A y B son los bits que se quieren sumar, la entrada Ci es la entrada del acarreo anterior, la salida S es el resultado de la suma de A y B, teniendo en cuenta el acarreo anterior, y Co es la salida del acarreo que genera dicha suma.

La tabla de verdad del sumador completo es:

A	B	Ci	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabla 2: Tabla de verdad del sumador completo

Una vez, que se ha visto el circuito lógico del sumador completo se va a proceder a diseñar dicho circuito, mediante lenguaje VHDL de tres maneras diferentes, mediante un algoritmo estructural, mediante uno funcional y mediante uno de flujo de datos. De esta forma se verán tres tipos de enfoque distintos de la programación con VHDL.

Descripción funcional

Se implementa el circuito describiendo la forma en que se comporta, es la que más se parece a los lenguajes de programación de software ya que se ejecuta secuencialmente. De esta forma es como se ha venido programando en las anteriores prácticas. Para que la descripción sea secuencial, las sentencias y funciones van dentro de los procesos, ya que dentro de ellos las sentencias se ejecutan una a una.

Siguiendo la descripción funcional se ha diseñado el sumador completo mediante el siguiente código:

```
library IEEE;                                   -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;                -- Paquetes de la librería estándar de la IEEE
entity aritmeticos is
   port( x: in STD_LOGIC;                       -- Bit de entrada
         y: in STD_LOGIC;                       -- Bit de entrada
         ci: in STD_LOGIC;                      -- Acarreo anterior
         s: out STD_LOGIC;                      -- Resultado de la suma
         acarreo: out STD_LOGIC);               -- Acarreo generado de la suma
end aritmeticos;
...
```

```
...
architecture sum1 of aritmeticos is                -- Cabecera de programa
begin
   process(x,y,ci)                                 -- Cabecera de proceso
   begin
      if x='0' and y='0' and ci='0' then           -- Sentencias if y elsif para obtener
           s<='0';                                 -- el resultado y el acarreo de la suma
           acarreo<='0';
      elsif x='0' and y='0' and ci='1' then
           s<='1';
           acarreo<='0';
      elsif x='0' and y='1' and ci='0' then
           s<='1';
           acarreo<='0';
      elsif x='0' and y='1' and ci='1' then
           s<='0';
           acarreo<='1';
      elsif x='1' and y='0' and ci='0' then
           s<='1';
           acarreo<='0';
      elsif x='1' and y='0' and ci='1' then
           s<='0';
           acarreo<='1';
      elsif x='1' and y='1' and ci='0' then
           s<='0';
           acarreo<='1';
      else
           s<='1';
           acarreo<='1';
      end if;
   end process;                                    -- Final de proceso
end sum1;                                          -- Final del programa
```

Descripción de flujo de datos

Esta descripción se comporta de forma totalmente contraria a la funcional, se basa en describir asignaciones concurrentes de señales, es decir ejecuta las sentencias en paralelo y no secuencialmente. En este caso los procesos no se utilizan.

En este caso concreto, para el sumador completo, el diseño es mucho más corto mediante descripción de flujo de datos que mediante descripción funcional. No siempre es así, para diseños de mayor envergadura este tipo de descripción suele ser muy engorrosa y laboriosa, por lo que en esos casos se utiliza el algoritmo funcional o el estructural, que veremos a continuación.

Aquí se muestra el código para ver cómo es mucho más corto y sencillo que el anterior.

```
library IEEE;                                          -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;                       -- Paquetes de la libreía estándar de la IEEE

entity sumador2 is
   port( x: in STD_LOGIC;                              -- Bit de entrada
         y: in STD_LOGIC;                              -- Bit de entrada
         ci: in STD_LOGIC;                             -- Acarreo anterior
         s: out STD_LOGIC;                             -- Resultado de la suma
         acarreo: out STD_LOGIC);                      -- Acarreo generado de la suma
end sumador2;

architecture sum2 of sumador2 is                       -- Cabecera del programa
begin
   s<=(x xor y) xor ci;                                -- Función para obtener el resultado de la suma
    acarreo<=(x and y) or ((x xor y) and ci);          -- Función para obtener el acarreo de la suma
end sum2;                                              -- Final del programa
```

Descripción estructural

Permite la realización de diseños jerárquicos, a partir de ahora será la que más se utilizará para diseñar los ejercicios, ya que simplifica mucho los diseños complejos

Para diseñar sistemas electrónicos mediante este tipo de descripción es necesario introducir un mecanismo que permite dotar de jerarquía al diseño, este mecanismo son los componentes. Un componente es una entidad que ha sido declarada en otro archivo de diseño o en alguna biblioteca, dicho archivo debe estar incluido en el proyecto en el que se está trabajando, si no el componente no podrá ser utilizado.

Es necesario declarar el componente que se va a usar, para ello se definirá en la parte declarativa de la arquitectura del programa que se está elaborando. La forma de declararlo es muy parecida a la de una entidad:

```
COMPONENT nombre IS
      GENERIC(lista_parametros);
      PORT(lista_de_puertos);
END COMPONENT nombre;
```

Una vez declarado, el componente ya puede ser usado en el programa, para ello se utiliza la siguiente estructura para referenciarlo:

```
referencia: COMPONENT nombre
            GENERIC MAP (parámetros)
            PORT MAP (puertos);
```

Se pueden declarar y referenciar tantos componentes como sea necesario para implementar el diseño deseado, siempre y cuando las entidades de dichos componentes estén incluidas en el proyecto en el cual se está elaborando el programa.

Como ya se ha dicho antes, mediante este mecanismo se consigue una descripción jerarquizada. Dicha jerarquía se muestra en la pestaña ***Herarchy*** del ***Project Navigator***, se ve claramente en la siguiente figura:

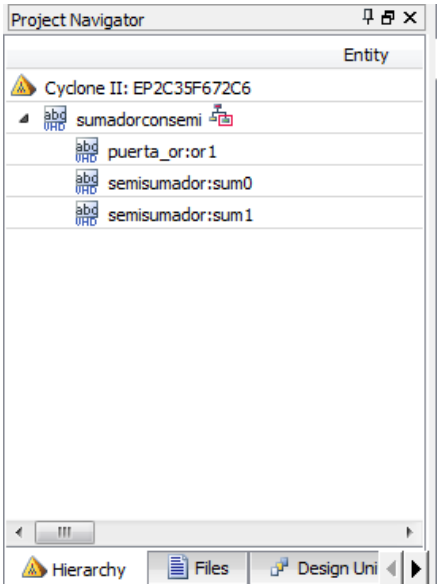

Figura 2: Jerarquía del diseño del sumador completo

Ahora, que ya se ha visto esta opción, se va a implementar el sumador completo utilizando distintos componentes. En este caso se diseñará mediante dos semisumadores y una puerta lógica or.

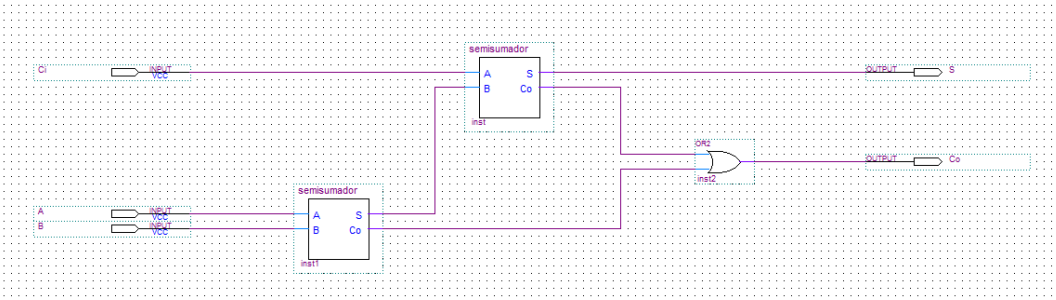

Figura 3: Sumador completo mediante semisumadores y puerta or

A diferencia del sumador completo, el semisumador suma dos bits pero no es capaz de tener en cuenta el acarreo anterior, pero mediante una combinación entre ellos se puede conseguir un sumador completo. La tabla de verdad del semisumador es muy sencilla:

A	B	S	Co
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabla 3: Tabla de verdad del semisumador

El circuito lógico del semisumador es el siguiente:

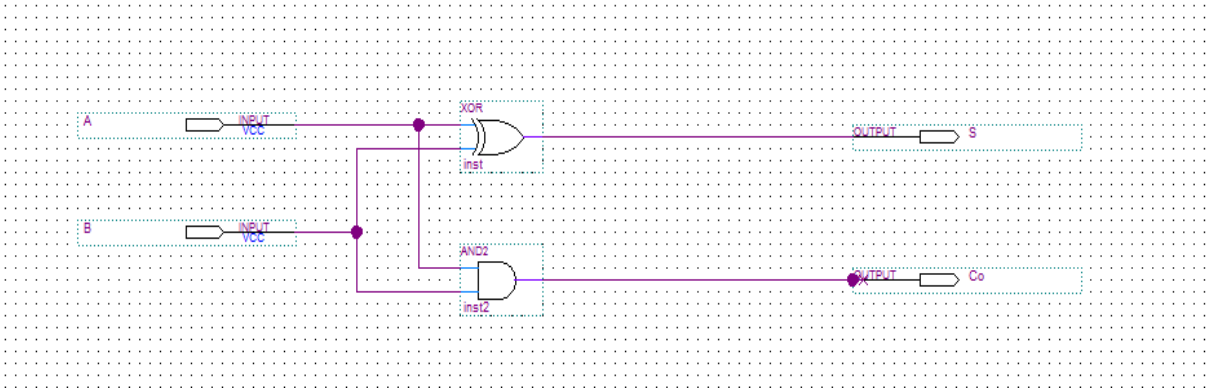

Figura 4: Circuito lógico del semisumador

Para obtener el semisumador mediante VHDL se ha elaborado el siguiente algoritmo de flujo de datos:

```
library IEEE;                                   -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;                -- Paquetes de la librería estándar de la IEEE

entity semisumador is
   port( a: in STD_LOGIC;                       - Bit de entrada
         b: in STD_LOGIC;                       -- Bit de entrada
         s: out STD_LOGIC;                      -- Resultado de la suma
         acarreo: out STD_LOGIC);               -- Acarreo de la suma
end semisumador;

architecture semi of semisumador is             -- Cabecera del programa
begin
   s<=a xor b;                                  -- Función para obtener el resultado de la suma
   acarreo<=a and b;                            -- Función para obtener el acarreo de la suma
end semi;                                       -- Final del programa
```

A continuación, se elaborará una entidad para la puerta lógica or y así poder incluir su componente en el diseño:

```
library IEEE;                                   -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;                -- Paquetes de la librería estándar de la IEEE
entity puerta_or is
   port( a: in STD_LOGIC;                       -- Señal de entrada
         b: in STD_LOGIC;                       -- Señal de entrada
         c: out STD_LOGIC);                     -- Señal de salida
end puerta_or;
architecture or1 of puerta_or is                -- Cabecera del programa
begin
   c<=a or b;                                   -- Función or
end or1;                                        -- Final del programa
```

Una vez que ya se tienen las entidades de los componentes que se necesitan elaboradas e incluidas en el proyecto, se realiza el diseño estructural del sumador completo mediante el siguiente código:

```
library IEEE;                                   -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;                -- Paquetes de la librería estándar de la IEEE
entity sumadorconsemi is
   port( x: in STD_LOGIC;                       -- Bit de entrada
         y: in STD_LOGIC;                       -- Bit de entrada
         ci: in STD_LOGIC;                      -- Bit de acarreo anterior
         s: out STD_LOGIC;                      -- Resultado de la suma
         co: out STD_LOGIC);                    -- Acarreo de la suma
end sumadorconsemi;
architecture sumadorsemi of sumadorconsemi is   -- Cabecera del programa
   component semisumador is                     -- Declaración del componente semisumador
      port( a: in STD_LOGIC;                    -- Bit de entrada del semisumador
         b: in STD_LOGIC;                       -- Bit de entrada del semisumador
         s: out STD_LOGIC;                      -- Resultado de la suma del semisumador
         acarreo: out STD_LOGIC);               -- Acarreo de la suma del semisumador
   end component;
   component puerta_or is                       -- Declaración del componente puerta_or
      port( a: in STD_LOGIC;                    -- Bit de entrada de la puerta or
         b: in STD_LOGIC;                       -- Bit de entrada de la puerta or
         c: out STD_LOGIC);                     -- Bit de salida de la puerta or
         end component;
         ...
```

```
            ...
   signal co0,co1,s0: STD_LOGIC;                -- Declaración de señal
begin
   sum0: semisumador                            -- Referencia de componente
      PORT MAP (x,y,s0,co0);                    -- Asignación de señales a los puertos
   sum1: semisumador                            -- Referencia de componente
      PORT MAP (ci,s0,s,co1);                   -- Asignación de señales a los puertos
   or1: puerta_or                               -- Referencia de componente
      PORT MAP (co1,co0,co);                    -- Asignación de señales a los puertos
end sumadorsemi;                                -- Final del programa
```

Ejercicio

Simular los distintos diseños del sumador completo y comprobar que los resultados coinciden.

4. Ejercicios de diseño

4.1. Sumador de números binarios de 4 bits

Realizar un sumador que sume dos números binarios de 4 bits cada uno a través de la descripción estructural, utilizando el sumador completo que se ha elaborado en el ejercicio 3.1. Los números de 4 bits se introducirán mediante 8 interruptores, 4 para cada número, otro interruptor introducirá el acarreo anterior y el resultado se mostrará a través de 5 leds rojos.

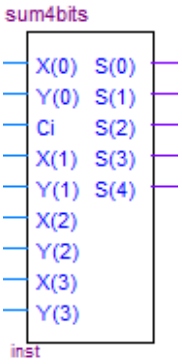

Figura 5: Sumador de 4 bits

Para comprobar el correcto funcionamiento del programa se pide simular los siguientes casos:

a) 3 + 4
b) 10 + 7
c) 14 + 2
d) 6 + 8 (Con acarreo anterior = '1')

Se deberá entregar el código VHDL con sus respectivos comentarios, el cronograma de simulación con los estados de entradas y salidas que se piden y el archivo con extensión *.csv*, que corresponde a la asignación de pines.

4.2. Sumador-Restador

Se pide diseñar un sumador-restador de números de 4 bits mediante descripción estructural, usando el sumador de números de 4 bits que se ha realizado en el ejercicio anterior y el menor número de puertas lógicas posibles. Se utilizará una señal que seleccione que operación vamos a utilizar, dicha señal será '0' cuando se desee sumar y '1' cuando se quiera restar. Para la resta se utilizará el complemento a dos (CA2), lo que significa que los números irán desde el 7 (0111 en CA2) hasta el -8 (1000 en CA2).

Para ello los números se introducirán de igual modo que antes, mediante un interruptor se elegirá la operación (Suma o Resta) y en este caso el resultado se mostrará a

través de dos displays 7-segmentos, el primero de ellos indicará el signo del número y el segundo la cifra.

El codificador de 7-segmentos está hecho en la práctica 1 pero habrá que modificarlo, ya que pueden aparecer números negativos en complemento a dos.

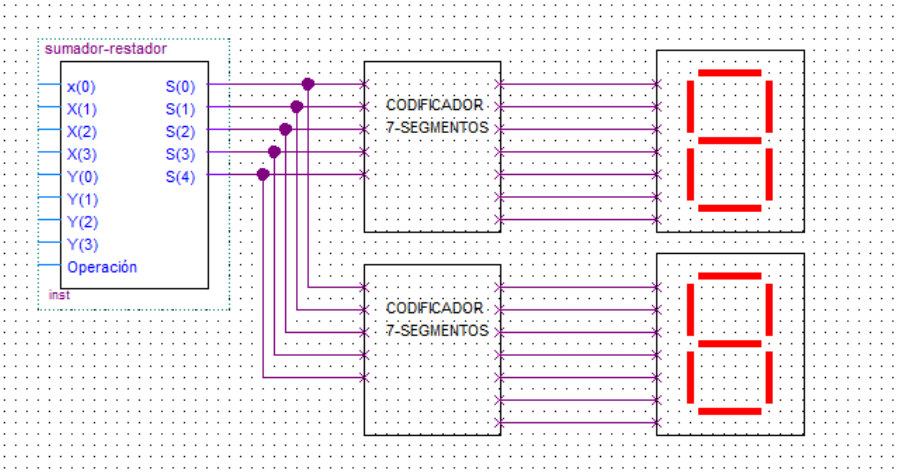

Figura 6: Sumador- Restador de 4 bits

En este ejercicio también se proponen los siguientes casos para la simulación:

a) 2 + 3
b) 5 – 4
c) 2 – 7
d) 6 – 8

Se deberá entregar el código VHDL con sus respectivos comentarios, el cronograma de simulación con los estados de entradas y salidas que se piden y el archivo con extensión *.csv*, que corresponde a la asignación de pines.

Se deberá mostrar al profesor de prácticas que los ejercicios de diseño se han volcado a la tarjeta DE2 y que funcionan correctamente.

Resolución de la práctica 3

3.1 Sumador binario completo

A continuación se muestran los cronogramas de las simulaciones de las tres formas distintas en las que se ha diseñado el sumador completo:

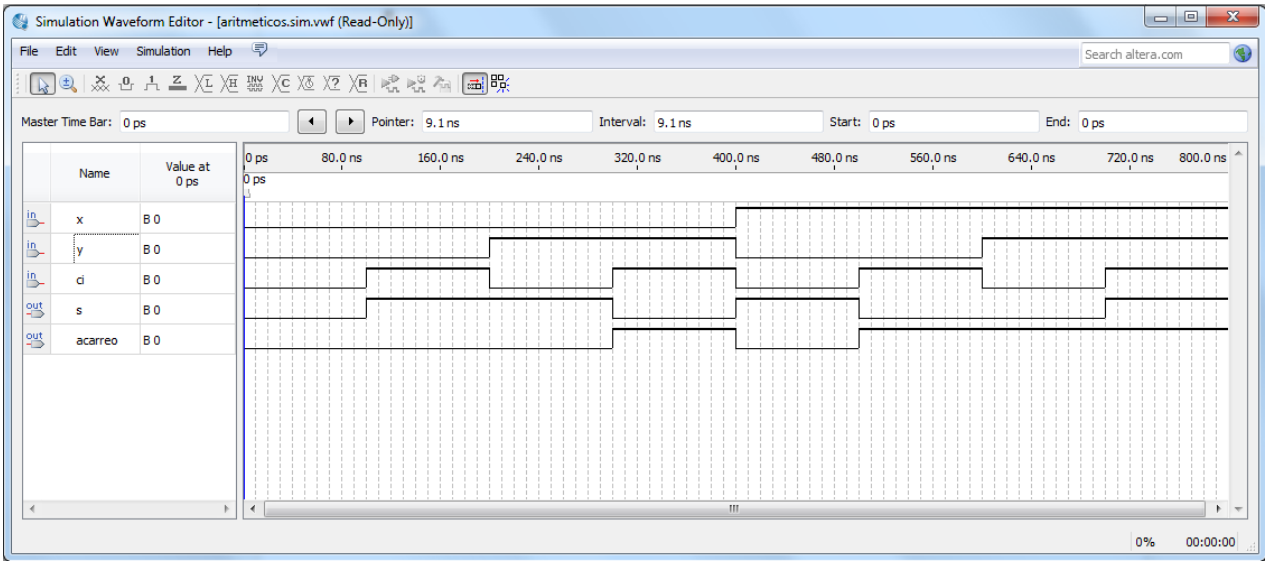

Figura 1: Resultados del diseño funcional del sumador completo

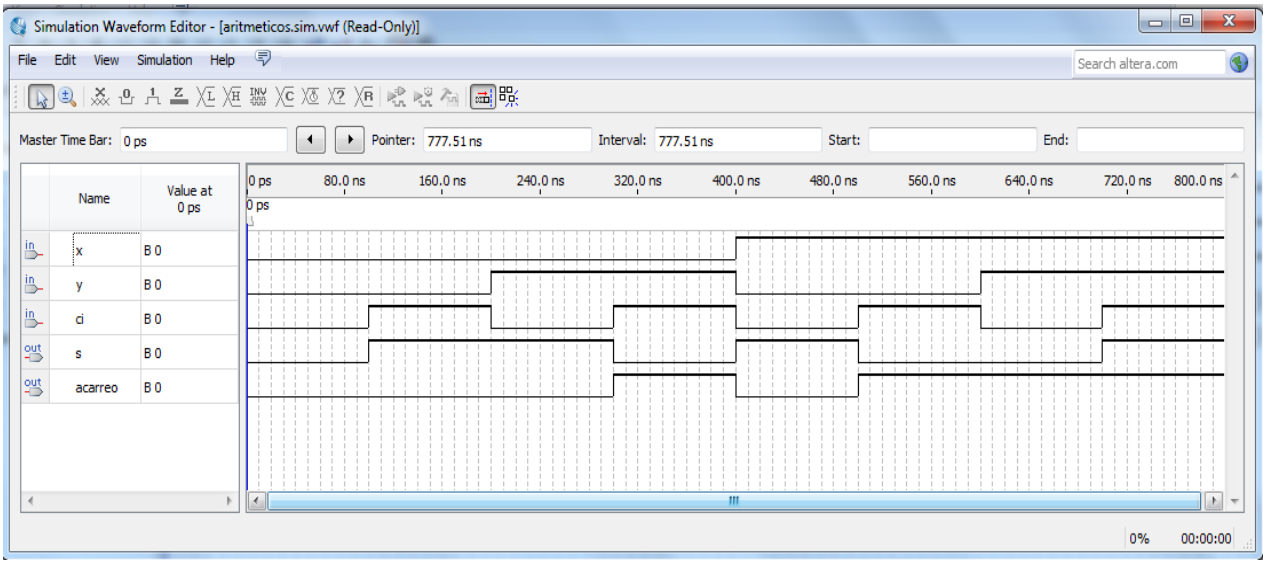

Figura 2 Resultados del diseño de flujo de datos del sumador completo

Antes de simular el algoritmo estructural del sumador completo, se va a simular por separado las dos entidades que lo conforman, el semisumador y la puerta lógica or, para ver si funcionan correctamente.

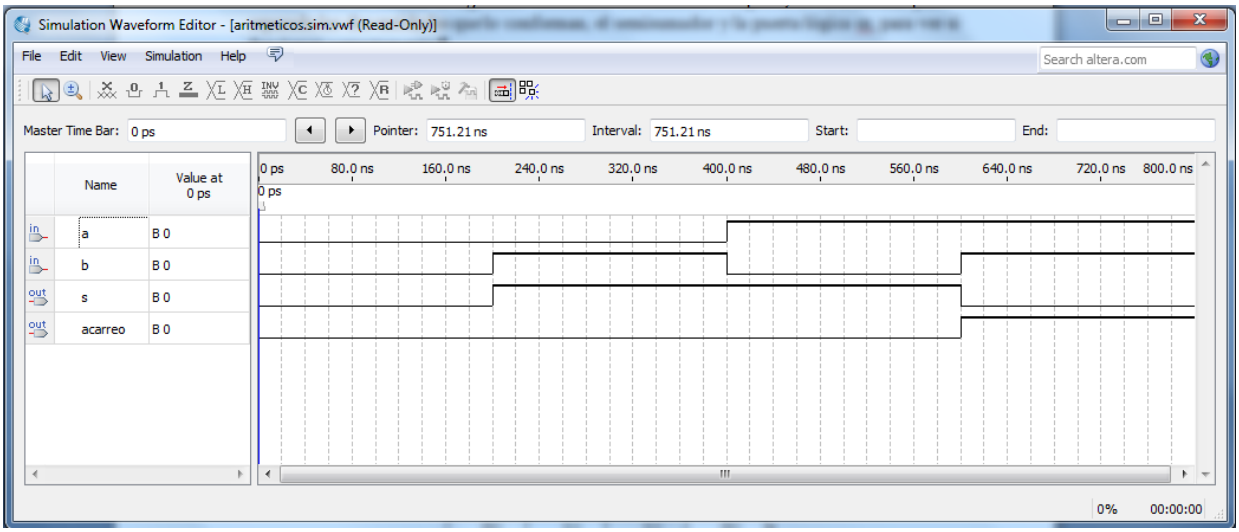

Figura 3: Resultados del semisumador

Se ve claramente que el cronograma de la simulación del código que implementa el semisumador coincide con la tabla de verdad.

A	B	S	Co
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabla 1: Tabla de verdad del semisumador

Por otro lado, está el código de la puerta or de dos entradas, cuyo cronograma también coincide con su correspondiente tabla de verdad.

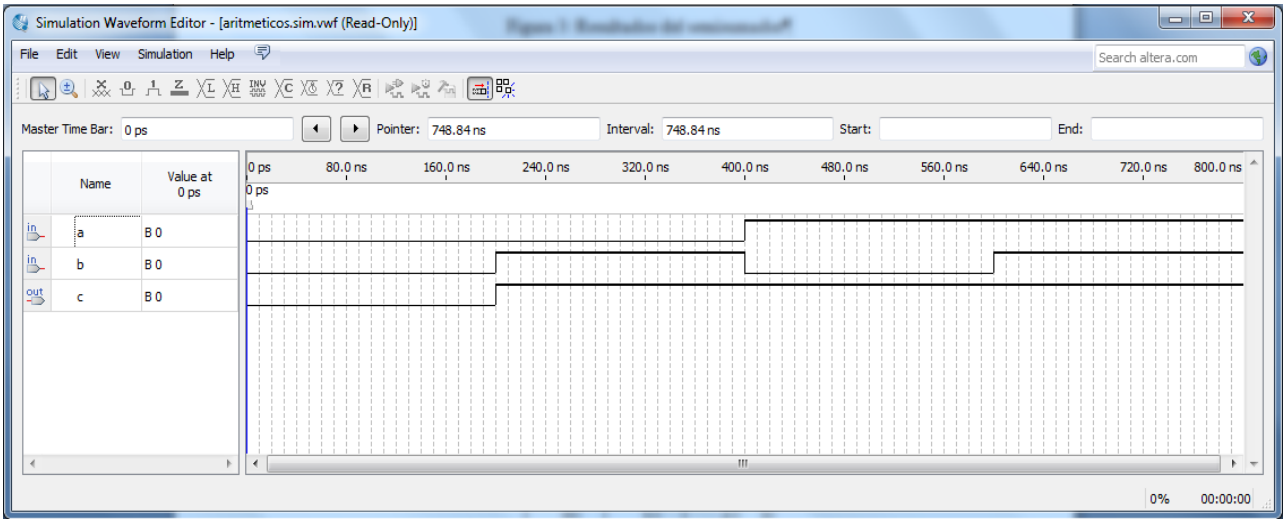

Figura 4: Resultados de la puerta lógica or de dos entradas

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 2: Tabla de verdad de la puerta lógica or de dos entradas

Una vez que se ha comprobado el correcto funcionamiento de los dos componentes que se van a utilizar en el algoritmo estructural para implementar el sumador completo, se procede a simular dicho código.

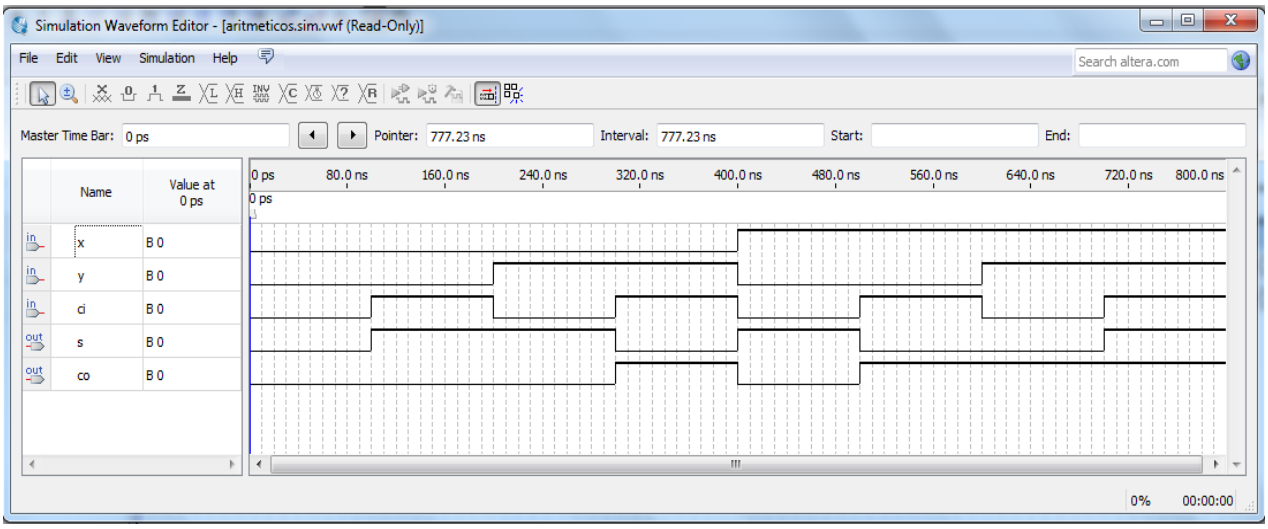

Figura 5: Resultados del diseño estructural del sumador completo

Comparando las tres simulaciones, se aprecia que son idénticas y que los tres tipos de descripciones dan los mismos resultados. Por lo que según el tipo de diseño que se tenga que hacer se utilizará convenientemente un tipo de diseño u otro. En este caso el algoritmo más corto y sencillo es el de flujo de datos, ya que utiliza únicamente dos sentencias en la arquitectura para implementar el diseño requerido.

4. Ejercicios de diseño

4.1. Sumador de números binarios de 4 bits

Para diseñar el sumador de números de 4 bits se pedía utilizar el sumador completo de 2 bits del ejercicio anterior, por lo que se debe utilizar la descripción estructural para el diseño. Para ello, se hará uso de la entidad del sumador completo y se declarará como componente. El circuito lógico del sumador que se requiere necesita 4 sumadores completos de 2 bits para poder ser implementado y se deben conectar de la siguiente manera:

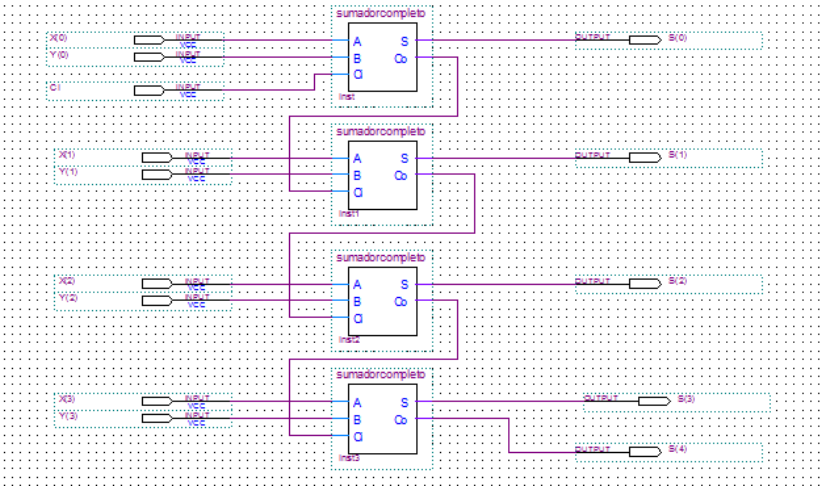

Figura 6: Circuito lógico del sumador de números de 4 bits

Para diseñar este circuito se ha elaborado el siguiente código:

```
library IEEE;                                   -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;                -- Paquetes de la librería estándar de la IEEE
entity sumador4bits is
   port( x: in STD_LOGIC_VECTOR (3 downto 0);   -- Señal de entrada de 4 bits
         y: in STD_LOGIC_VECTOR (3 downto 0);   -- Señal de entrada de 4 bits
         ci: in STD_LOGIC;                      -- Bit de acarreo anterior
         s: out STD_LOGIC_VECTOR (4 downto 0)); -- Señal de salida de 5 bits
end sumador4bits;
architecture sum4bits of sumador4bits is        -- Cabecera del programa
   component sumador2 is                        -- Declaración del componente del sumador completo
      port( x: in STD_LOGIC;                    -- Bit de entrada
         y: in STD_LOGIC;                       -- Bit de entrada
         ci: in STD_LOGIC;                      -- Bit de acarreo anterior
         s: out STD_LOGIC;                      -- Resultado de la suma
         acarreo: out STD_LOGIC);               -- Acarreo de la suma
   end component;

   signal co0,co1,co2: STD_LOGIC;               -- Declaración de señales
...
```

```
...
begin
   sum1: sumador2                          -- Referencia de componente
         PORT MAP (x(0),y(0),ci,s(0),co0);  -- Asignación de señales a los puertos
   sum2: sumador2                          -- Referencia de componente
         PORT MAP (x(1),y(1),co0,s(1),co1); -- Asignación de señales a los puertos
   sum3: sumador2                          -- Referencia de componente
         PORT MAP (x(2),y(2),co1,s(2),co2); -- Asignación de señales a los puertos
   sum4: sumador2                          -- Referencia de componente
         PORT MAP (x(3),y(3),co2,s(3),s(4)); -- Asignación de señales a los puertos

end sum4bits;                              -- Final del programa
```

Ahora, para comprobar su funcionamiento antes del volcado, se simularán los casos solicitados en el enunciado del ejercicio. Los resultados que se deberían obtener son:

a) 3 + 4 = 7 → 0011 + 0100 = 0111
b) 10 + 7 = 17 → 1010 + 0111 = 10001
c) 14 + 2 = 16 → 1110 + 0010 = 10000
d) 6 + 8 + 1 = 15 → 0110 + 1000 + 0001 = 1111

Se ve perfectamente que los resultados del cronograma coinciden con lo calculado teóricamente.

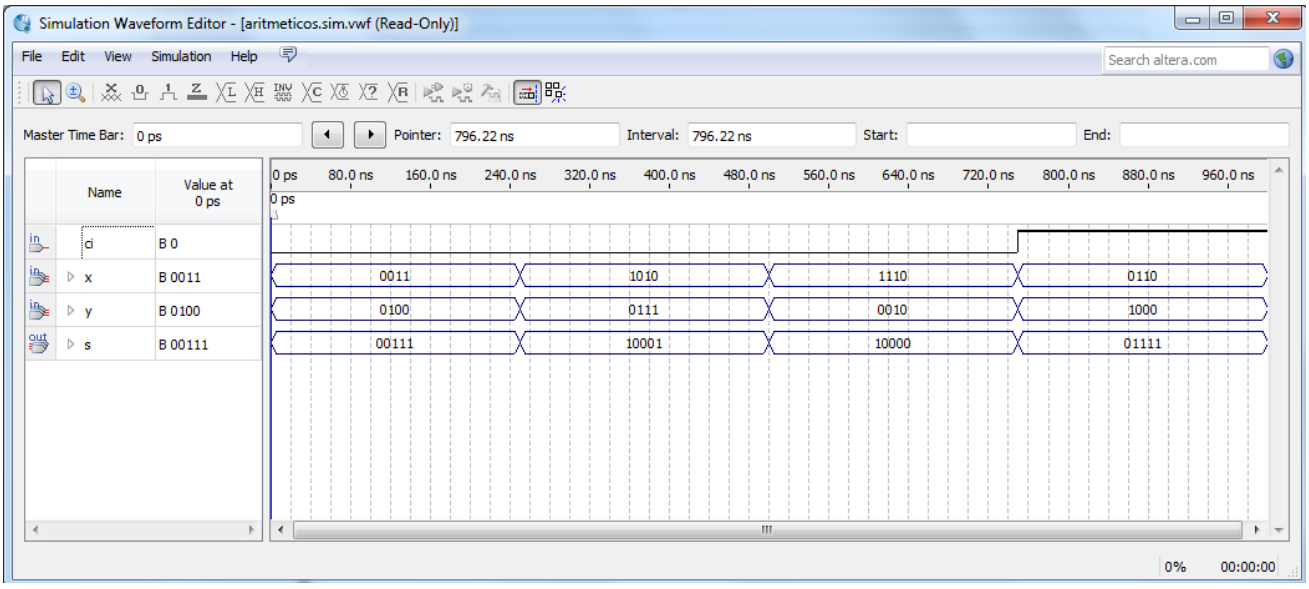

Figura 7: Resultados del sumador de números de 4 bits

4.2. Sumador-Restador

Para este ejercicio se necesita el sumador de números de 4 bits que se ha elaborado en el ejercicio 4.1 y además para conseguir restar mediante el complemento a dos se requieren cuatro puertas lógicas xor. El circuito lógico del sumador-restador que se debe implementar mediante lenguaje VHDL es el siguiente:

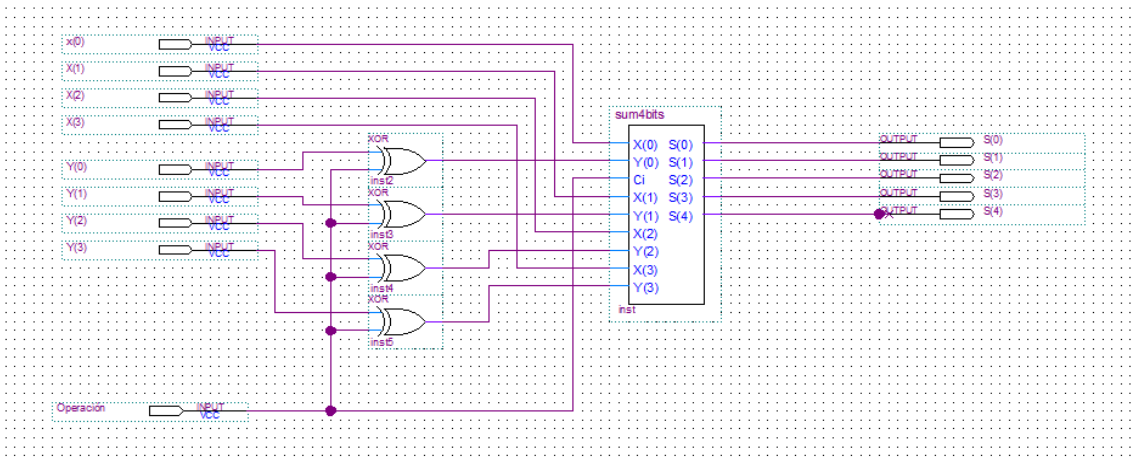

Figura 8: Circuito lógico del Sumador-Restador

Con este circuito, si se quiere sumar, la señal de entrada 'Operación' tiene que ser '0', para que las puertas xor no modifiquen el valor del número de entrada y así el sumador de 4 bits sume y actúe como si no estuviesen las puertas lógicas conectadas a sus entradas.

Sin embargo, cuando se quiera restar, la señal 'Operación' tendrá que ser '1', de este modo el número cambiará de signo a través de las puertas xor. Si sólo se haría esto, el número se convertiría en complemento a 1 (CA1), para que obtenerlo en CA2 se le debe sumar al CA1 un '1', por lo que de aquí se explica el porqué de que la señal de operación este también conectada al acarreo anterior Ci. De esta forma el minuendo de la resta no sufre modificaciones, mientras que al sustraendo se le cambia de signo a través del circuito que se acaba de detallar.

Para la implementación del diseño se necesitan 3 componentes distintos, el sumador de 4 bits, la puerta xor y el codificador de 7-segmentos. Ya se ha comprobado en el ejercicio anterior que el sumador de 4 bits funciona correctamente, ahora, antes de simular el código completo del sumador-restador, se va a comprobar la puerta xor y el codificador.

En primer lugar, aquí está el código para la puerta lógica xor:

```
library IEEE;                                  -- Librería estándar de la IEEE

use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;               -- Paquetes de la librería estándar de la IEEE

entity puerta_xor is
   port( a: in STD_LOGIC;                      -- Bit de entrada
         b: in STD_LOGIC;                      -- Bit de entrada
         c: out STD_LOGIC);                    -- Bit de salida
end puerta_xor;

architecture xor1 of puerta_xor is             -- Cabecera del programa
begin
   c<=a xor b;                                 -- Función xor
end xor1;                                      -- Final del programa
```

A continuación, se simula el código y se ve fácilmente como los resultados coinciden exactamente con la tabla de verdad de la puerta xor.

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 3: Tabla de verdad de la puerta xor

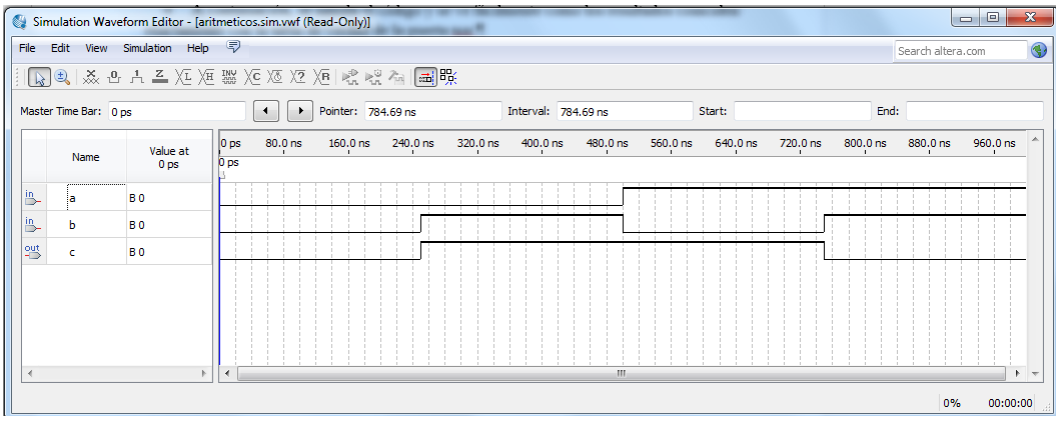

Figura 9: Resultados de la puerta xor

Ahora, se escribe elabora la entidad del codificador 7-segmentos, para luego poder incluirlo como componente en el diseño.

```
library IEEE;                                          -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;                          -- Paquetes de la librería estándar de la IEEE
entity seg7ca2 is
port( num:in STD_LOGIC_VECTOR(4 downto 0);             -- 4 interruptores de entrada para introducir el numero
        seg1,seg2: out STD_LOGIC_VECTOR(6 downto 0));  -- 2 vectores de salida de 7 bits cada uno
end seg7ca2;
architecture codigosegmento of seg7ca2 is              -- Cabecera del programa
begin
  process (num)                                        -- Cabecera de proceso
   begin
   case num is                                         -- Sentencia case para pasar de numero en CA2
     when "00000"=>seg1<="1000000";                    -- a numero en display 7-segmentos
          seg2<="1111111";
     when "00001"=>seg1<="1111001";
          seg2<="1111111";
     when "00010"=>seg1<="0100100";
          seg2<="1111111";
     when "00011"=>seg1<="0110000";
          seg2<="1111111";
     when "00100"=>seg1<="0011001";
          seg2<="1111111";
     when "00101"=>seg1<="0010010";
          seg2<="1111111";
     when "00110"=>seg1<="0000010";
          seg2<="1111111";
     when "00111"=>seg1<="1111000";
          seg2<="1111111";
     when "01000"=>seg1<="0000000";
          seg2<="0111111";
     when "01001"=>seg1<="1111000";
          seg2<="0111111";
     when "01010"=>seg1<="0000010";
          seg2<="0111111";
     when "01011"=>seg1<="0010010";
          seg2<="0111111";
     when "01100"=>seg1<="0011001";
          seg2<="0111111";
     when "01101"=>seg1<="0110000";
          seg2<="0111111";
     when "01110"=>seg1<="0100100";
          seg2<="0111111";
     when "01111"=>seg1<="1111001";
          seg2<="0111111";
     when others=>seg1<="1111111";
          seg2<="1111111";
   end case;
   end process;                                        -- Final de proceso
end codigosegmento;                                    -- Final de programa
```

Aquí se muestra la relación entre los números en CA2 y sus respectivos números en código 7-segmentos:

num	Seg1	Seg2
7 → 0111	1111000	1111111
6 → 0110	0000010	1111111
5 → 0101	0010010	1111111
4 → 0100	0011001	1111111
3 → 0011	0110000	1111111
2 → 0010	0100100	1111111
1 → 0001	1111001	1111111
0 → 0000	1000000	1111111
-1 → 1111	1111001	0111111
-2 → 1110	0100100	0111111
-3 → 1101	0110000	0111111
-4 → 1100	0011001	0111111
-5 → 1011	0010010	0111111
-6 → 1010	0000010	0111111
-7 → 1001	1111000	0111111
-8 → 1000	0000000	0111111

Tabla 4: Tabla de verdad del codificador 7-segmentos para números en CA2

Ahora, se realiza la simulación para observar como el codificador que se ha realizado nos da los mismos resultados que la tabla de verdad, por lo que ya se puede incluir este componente en nuestro diseño final.

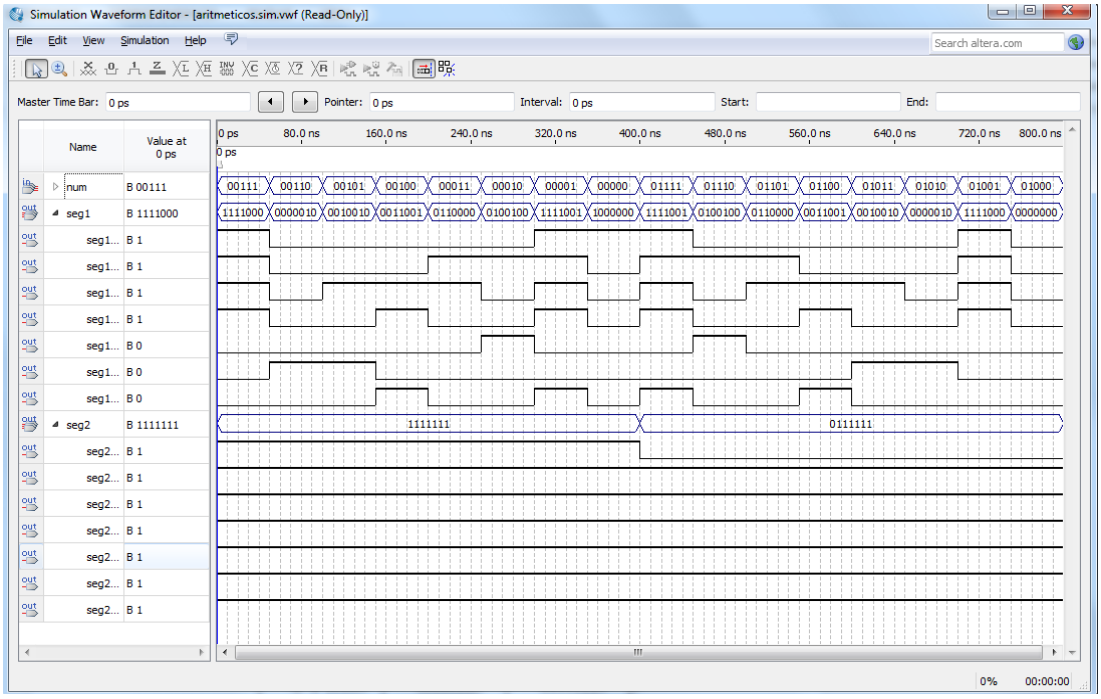

Figura 10: Resultados del codificador 7-segmentos para números en CA2

Llegados a este punto, en donde ya se ha comprobado el funcionamiento de todos los componentes que se van a utilizar en el diseño del sumador-restador, se elabora el algoritmo estructural.

```
library IEEE;                                          -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;                          -- Paquetes de la librería estándar de la IEEE
entity sumadorestador is
   port( num1: in STD_LOGIC_VECTOR (3 downto 0);       -- Número de entrada de 4 bits
         num2: in STD_LOGIC_VECTOR(3 downto 0);        -- Número de entrada de 4 bits
         operacion: in STD_LOGIC;                      -- Bit de selección de operación
         disp1,disp2: out STD_LOGIC_VECTOR (6 downto 0));  -- Señales de 7 bits para mostrar el resultado por los displays
end sumadorestador;

architecture sumres of sumadorestador is               -- Cabecera del programa
   component sumador4bits is                           -- Declaración de componente
      port( x: in STD_LOGIC_VECTOR (3 downto 0);       -- Señal de entrada de 4 bits
         y: in STD_LOGIC_VECTOR (3 downto 0);          -- Señal de entrada de 4 bits
         ci: in STD_LOGIC;                             -- Bit de acarreo anterior
         s: out STD_LOGIC_VECTOR (4 downto 0));        -- Señal de salida de 5 bits
   end component;
   component puerta_xor is                             -- Declaración de componente
      port( a: in STD_LOGIC;                           -- Bit de entrada
         b: in STD_LOGIC;                              -- Bit de entrada
         c: out STD_LOGIC);                            -- Bit de salida
   end component;
   component seg7ca2 is                                -- Declaración de componente
      port( num:in STD_LOGIC_VECTOR(4 downto 0);       -- 4 interruptores de entrada para introducir el numero
         seg1,seg2: out STD_LOGIC_VECTOR(6 downto 0));  -- 2 vectores de salida de 7 bits cada uno
   end component;

   signal y_ca2: STD_LOGIC_VECTOR (3 downto 0);        -- Declaración de señales
   signal result: STD_LOGIC_VECTOR (4 downto 0);

begin
   xor1: puerta_xor                                    -- Referencia de componente
         PORT MAP (num2(0),operacion,y_ca2(0));        -- Asignación de señales a los puertos
   xor2: puerta_xor                                    -- Referencia de componente
         PORT MAP (num2(1),operacion,y_ca2(1));        -- Asignación de señales a los puertos
   xor3: puerta_xor                                    -- Referencia de componente
         PORT MAP (num2(2),operacion,y_ca2(2));        -- Asignación de señales a los puertos
   xor4: puerta_xor                                    -- Referencia de componente
         PORT MAP (num2(3),operacion,y_ca2(3));        -- Asignación de señales a los puertos
   sumador: sumador4bits                               -- Referencia de componente
         PORT MAP (num1,y_ca2,operacion,result);       -- Asignación de señales a los puertos
   seg7: seg7ca2                                       -- Referencia de componente
         PORT MAP (result,disp1,disp2);                -- Asignación de señales a los puertos
end sumres;                                            -- Final del programa
```

Para este ejercicio se han pedido que se simularan una serie de casos, los resultados que deberían aparecer son los siguientes:

a) 3 + 4 = 7 → 0011 + 0100 = 0111
Display1 = 1111000
Display2 = 1111111

b) 5 – 4 = 1 → 0101 + CA2(0100) = 0101 + 1100 = **1**0001 >> 0001
Display1 = 1111001
Display2 = 1111111

c) 2 – 7 = -5 → 0010 + CA2(0111) = 0010 + 1001 = 1011
-5 = CA2(0101) = 1010 + 1 = 1011
Display1 = 0010010
Display2 = 0111111

d) 3 – 6 = -3 → 0011+ CA2(0110) = 0011 + 1010 = 1101
-3 = CA2(0011) = 1100 + 1 = 1101
Display1 = 0110000
Display2 = 0111111

Lo siguiente que se hará es simular estos casos y ver si coinciden con lo calculado anteriormente.

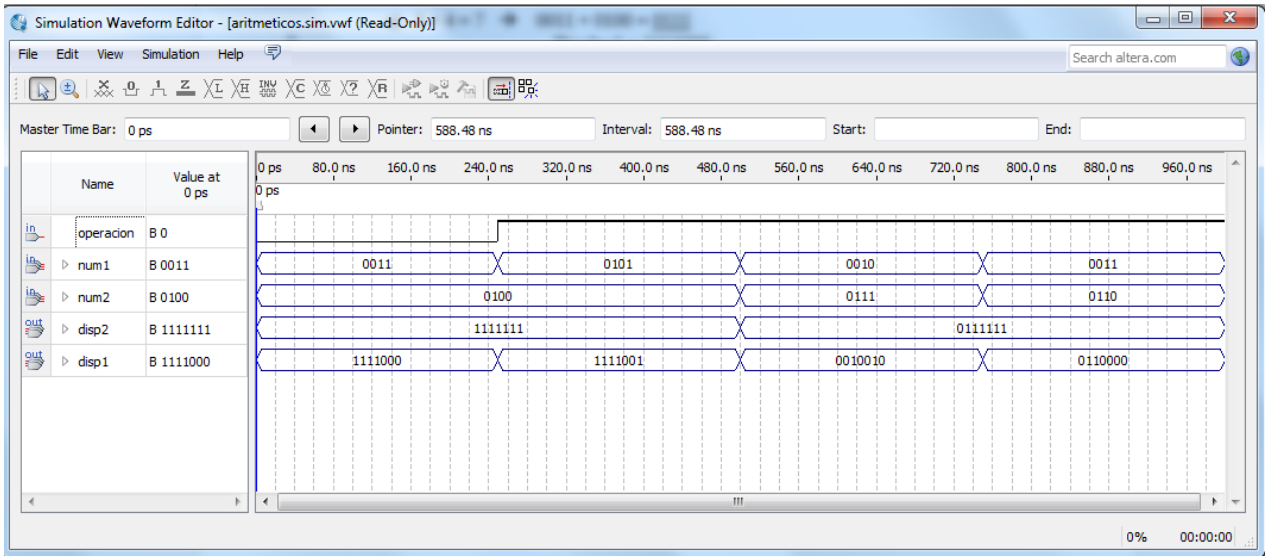

Figura 11: Resultados del Sumador-Restador de 4 bits

Como se puede observar, las señales del cronograma coinciden con los resultados esperados.

Práctica 4: Circuitos secuenciales y pantalla LCD

La gran diferencia entre los sistemas combinacionales y los secuenciales es que las salidas de estos últimos no dependen exclusivamente de los valores de las entradas en ese instante, sino que también dependen de los estados anteriores. El sistema secuencial más sencillo y el más utilizado es el biestable y es el que se va a ver en esta sesión.

Los biestables pueden ser síncronos o asíncronos, dependiendo si están gobernados por una señal de reloj (síncronos) o no (asíncronos). La mayoría de ellos, que son con los que se trabajará en la práctica, son síncronos.

1. Objetivos

- Ver cómo funcionan los sistemas secuenciales, en concreto los biestables, y aprender algunas de sus aplicaciones, como la de contadores.
- Elaborar y simular varios tipos diferentes de contadores y ver su comportamiento.
- Conocer el funcionamiento de una pantalla LCD y configurarla.
- Utilizar la pantalla LCD para mostrar el valor de uno de los contadores que se realizará en la práctica.

2. Materiales

- Ordenador personal con el software QUARTUS II disponible.
- Tarjeta educacional DE2
- Guión de prácticas

3. Desarrollo de la práctica

3.1. Biestable JK

El biestable JK es uno de los más utilizados y con el que se pueden implementar circuitos contadores y registros. En este caso, se introducirá el biestable JK síncrono disparado por flanco de subida y se incluirán las entradas Preset y Clear, que activarán en bajo.

A continuación se muestra un símbolo que lo representa:

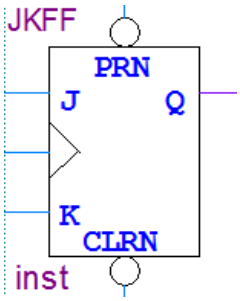

Figura 1: Biestable JK síncrono

Para realizar el símbolo anterior mediante lenguaje VHDL, se ha elaborado el siguiente código:

```
library IEEE;                                   -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;                   -- Paquetes de la librería estándar de la IEEE

entity biest_jk is
   port( j: in STD_LOGIC;                       -- Señal de entrada J
         k: in STD_LOGIC;                       -- Señal de entrada K
         clock: in STD_LOGIC;                   -- Señal de reloj
         preset: in STD_LOGIC;                  -- Señal de Preset
         clear: in STD_LOGIC;                   -- Señal de Clear
         q: out STD_LOGIC;                      -- Señal de salida
         notq: out STD_LOGIC);                  -- Señal de salida negada
end biest_jk;

architecture jksincrono of biest_jk is          -- Cabecera del programa
   signal estado: STD_LOGIC;                    -- Declaración de señales
    signal jk: STD_LOGIC_VECTOR (1 DOWNTO 0);
begin
   process(j,k,preset,clear,clock)              -- Cabecera de proceso
   begin
      if clock'event and clock='1' then         -- Sentencia if para detectar el flanco de subida de la señal de
reloj
         if preset='1' then                     -- Sentencia if para comprobar el estado del Preset
            if clear='1' then                   -- Sentencia if para comprobar el estado del Clear
               if j='0' and k='0' then          -- Sentencias if y elsif para comprobar el estado de
                  estado<=estado;               -- las entradas j y k y asignar un valor a la salida
               elsif j='0' and k='1' then
                  estado<='0';
               elsif j='1' and k='0' then
                  estado<='1';
               elsif j='1' and k='1' then
                  estado<= not estado;
               end if;
            else
               estado<='0';                     -- Señal estado a '0' si Clear='0'
            end if;
         else
            estado<='1';                        -- Señal estado a '1' si Preset='0'
         end if;
      end if;
      q<=estado;                                -- Traspasar los valores de estado a la salida q y notq
      notq<=not estado;
   end process;                                 -- Final de proceso
end jksincrono;                                 -- Final del programa
```

Se puede apreciar en el código como las salidas no solo dependen de las entradas, sino que también dependen de los estados anteriores. Además, se ve la función que desempeñan tanto el Preset como el Clear, poner la salida a '1' y a '0', respectivamente, sin importar los valores de las entradas J y K. Ambas señales se activan en bajo.

Todo ello activado por el flanco de subida de la señal de reloj, esto significa que las salidas no adquieren un nuevo valor hasta que se detecta el flanco de subida de la señal de reloj, es entonces cuando la salida toma el valor dependiendo las señales de entrada que haya en ese momento.

Si se simula el diseño anterior, se obtiene la tabla de verdad del biestable JK síncrono, que se puede observar a continuación.

J	K	Preset	Clear	Q(t+1)
0	0	1	1	Q(t)
0	1	1	1	0
1	0	1	1	1
1	1	1	1	Not(Q(t))
X	X	1	0	0
X	X	0	1	1

Tabla 1: Tabla de verdad del biestable JK síncrono

Se recomienda simular el código, con todos los posibles estados de entradas y salidas, para comprobar y conocer el correcto funcionamiento de un biestable JK síncrono.

3.2. Contador síncrono ascendente de módulo 4

Para implementar este tipo de circuitos contadores, al contrario que en la práctica anterior para los circuitos aritméticos, resulta más fácil y menos laborioso utilizar la descripción funcional, ya que de esta forma se entiende el código es mucho más intuitivo y sencillo de entender.

Así pues, en este ejercicio se ha realizado el diseño de un contador síncrono ascendente de 0 a 3, como el que se puede ver en la siguiente figura.

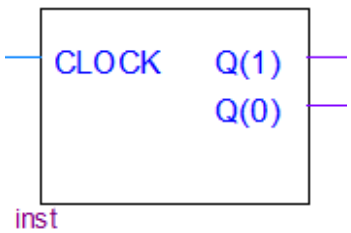

Figura 2: Contador síncrono ascendente de módulo 4

Como ya se ha comentado, en el código que se ha elaborado para implementar este contador se ha utilizado la descripción funcional, por lo que dicho código queda de la siguiente manera:

```
library IEEE;                                                    -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;                                    -- Paquetes de la librería estándar de la IEEE

entity cont_sinc_ascendente is
   port(  clock: in STD_LOGIC;                                   -- Señal de reloj
          q: out STD_LOGIC_VECTOR (1 DOWNTO 0) :="00");          -- Señal de salida
end cont_sinc_ascendente;

architecture contador_modulo4 of cont_sinc_ascendente is         -- Cabecera del programa
   signal contador: STD_LOGIC_VECTOR (1 DOWNTO 0):="00";         -- Declaración de señal
begin
   process(clock)                                                -- Cabecera de proceso
   begin
      if clock'event and clock='1' then                          -- Sentencia if para comprobar el flanco de subida del reloj
          if contador<"11" then                                  -- Sentencia if para detectar si el contador ha llegado a 3
             contador<=contador + 1;                             -- y debe reinicializarse o si debe seguir aumentando
          elsif contador="111" then
             contador<="00";
          end if;
      else
          contador<=contador;
      end if;
      q<=contador;                                               -- Traspaso de los valores de la señal contador a la salida
   end process;                                                  -- Final de proceso
end contador_modulo4;                                            -- Final del programa
```

Se pide simular el diseño en un cronograma mostrando todos los posibles casos de entradas y salidas, escogiendo una frecuencia de la señal de reloj óptima para que se vea claro el funcionamiento del contador.

3.3. Pantalla LCD

La pantalla LCD que incluye la tarjeta DE2 es el modelo CFAH1602B-TMC-JP, se trata de una LCD de 16x2 caracteres.

Posee 16 pines, como es típico en una LCD de estas características, y su distribución es la siguiente:

Pin Nº	Símbolo	Descripción
1	Vss	Ground
2	Vdd	Alimentación (5V)
3	VO	Operating Voltage (Variable)
4	RS	Si RS='1':Transmite Datos Si RS='0': Código de instrucciones
5	R/W	Si R/W='1': Leer Si R/W='0': Escribir
6	E	Enable
7	DBO	Bit 0 de datos
8	DB1	Bit 1 de datos
9	DB2	Bit 2 de datos
10	DB3	Bit 3 de datos
11	DB4	Bit 4 de datos
12	DB5	Bit 5 de datos
13	DB6	Bit 6 de datos
14	DB7	Bit 7 de datos
15	A	Power supply for LED backlight (+)
16	K	Power supply for LED backlight (-)

Tabla 2: Distribución de Pines en la pantalla LCD CFAH1602B-TMC-JP

Sin embargo, en la tarjeta DE2 se utilizan 13 pines para configurar y controlar esta pantalla LCD, la asignación de estos pines y sus funciones aparecen en la Anexo 1, en el apartado del módulo de la LCD. Es importante trabajar con esta asignación de pines, ya que es a través de estos como realmente se accederá a la LCD y no a través de los pines reales.

Para facilitar al alumno el uso de la pantalla LCD, se ha elaborado el código que inicializa, configura y deja preparada la LCD para escribir en ella. Para hacer uso de dicho código, solamente se ha de utilizar el componente denominado *lcd,* así pues, el archivo *lcd.vhd* se deberá incluir en el proyecto en el que se encuentre el diseño que necesite hacer uso de la pantalla LCD.

La forma de escribir en la LCD, mediante el componente facilitado, será caracter a caracter, es decir, se le asignará a cada posición, de las 32 (16x2) que tiene la pantalla en total, el carácter que se quiera escribir en ella en código ASCII.

A continuación se muestra la tabla de los diferentes caracteres que se pueden mostrar por pantalla con sus respectivos valores en código ASCII:

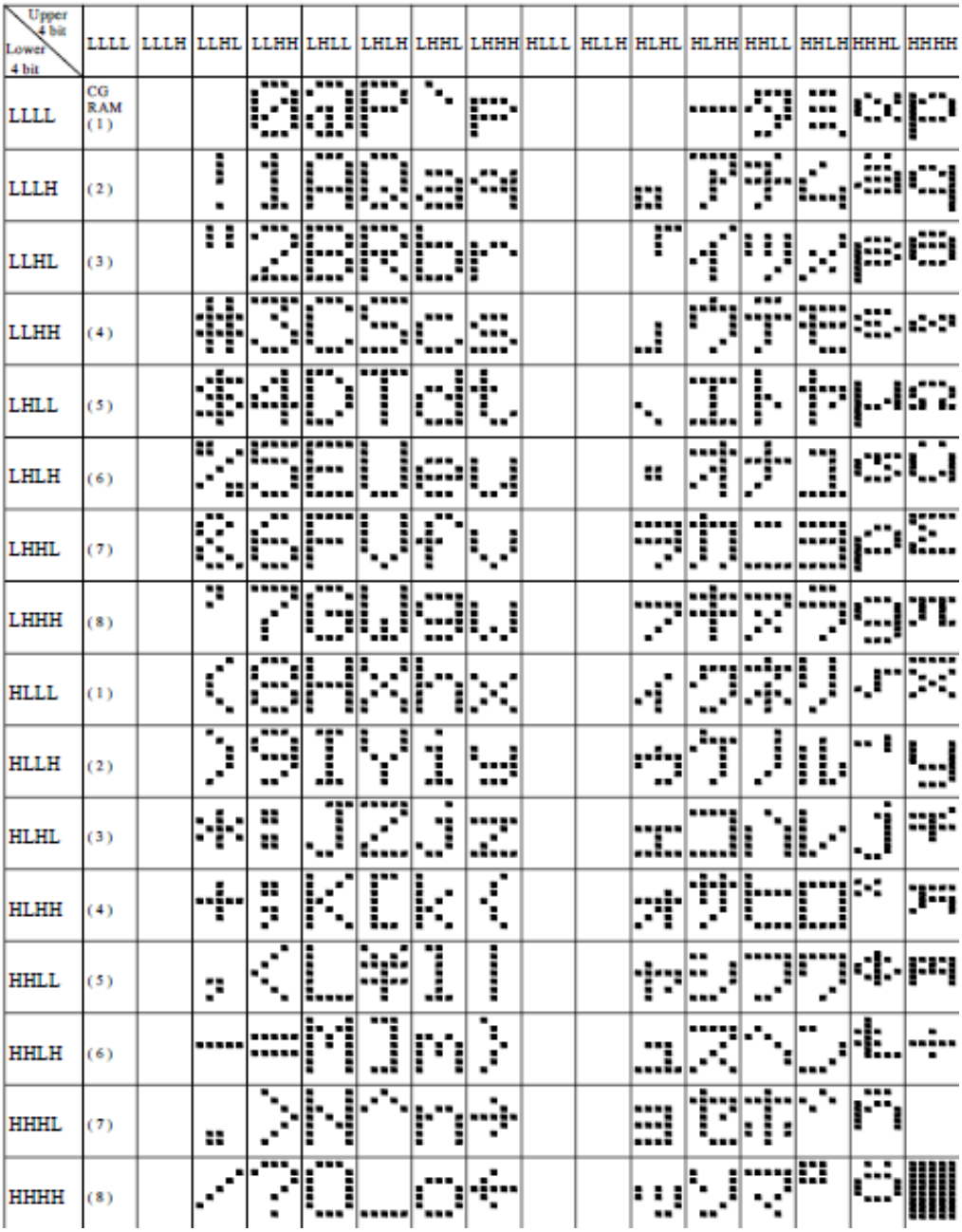

Tabla 3: Caracteres en código ASCII.

Un ejemplo de cómo utilizar la pantalla LCD con lo que se ha explicado anteriormente es el siguiente:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity ejemplo_lcd is
   port( clock: in STD_LOGIC;                          -- Reloj de 50MHz
         reset_lcd:in STD_LOGIC;
         lcd_rs,lcd_e,lcd_on: out STD_LOGIC;
         lcd_rw: buffer STD_LOGIC;
         datos: inout STD_LOGIC_VECTOR (7 DOWNTO 0):=x"00");
end ejemplo_lcd;

architecture ejemplo of ejemplo_lcd is
   component lcd is
   PORT( reset, clk_50Mhz: in STD_LOGIC;
          char1,char2,char3,char4,char5,char6,char7,char8,
          char9,char10,char11,char12,char13,char14,char15,
          char16,char17,char18,char19,char20,char21,char22,
         char23,char24,char25,char26,char27,char28,char29,
          char30,char31,char32: in STD_LOGIC_VECTOR (7 DOWNTO 0);
          LCD_RS, LCD_E, LCD_ON: out STD_LOGIC;
          LCD_RW: buffer STD_LOGIC;
         DATA_BUS: inout STD_LOGIC_VECTOR(7 DOWNTO 0));
   end component;

   signal char1,char2,char3,char4,char5,char6,char7,char8,
          char9,char10,char11,char12,char13,char14,char15,
          char16,char17,char18,char19,char20,char21,char22,
         char23,char24,char25,char26,char27,char28,char29,
          char30,char31,char32: STD_LOGIC_VECTOR (7 DOWNTO 0):=X"FE";

begin
   pantalla: lcd
         port map(reset_lcd,clock,char1,char2,char3,char4,char5,char6,char7,char8,
                  char9,char10,char11,char12,char13,char14,char15,
                  char16,char17,char18,char19,char20,char21,char22,
                  char23,char24,char25,char26,char27,char28,char29,
                  char30,char31,char32,lcd_rs,lcd_e,lcd_on,lcd_rw,datos);

   process(clock)
   begin

...                -- Dentro del proceso se le asigna a cada posición un caracter

   end process;
end ejemplo;
```

Como se puede ver claramente en el código, en la posición 1 de la pantalla se escribe a través de la señal char1, en la posición 2 a través de la señal char2 y así sucesivamente con todas las posiciones.

Se puede escribir en la pantalla LCD mediante otras muchas formas, pero se aconseja seguir la metodología que se acaba de mostrar para todos los diseños que incluyan LCD, ya que resulta sencillo e intuitivo usarla de esta manera.

Ejercicio:

a) ¿Por qué se inicializan todas las señales char (char1,char2, …) a X”FE”?
b) Realizar un programa en el que se muestre en la pantalla LCD los nombres de los integrantes del grupo de prácticas.
c) Comentar el código del programa.
d) Asignar los pines correctamente y volcar el diseño a la placa DE2

4. Ejercicios de diseño

4.1. Contador reversible

Se pide realizar un diseño de un contador reversible de módulo 10, que vaya de 0 a 9, utilizando la descripción (funcional, de flujo de datos o estructural) que se desee.

El contador deberá tener una **entrada de reloj**, que será el pulsador para incrementar o decrementar el valor de la salida, una **entrada de reset**, que será otro pulsador, una **entrada para elegir el sentido**, en este caso se utilizará un interruptor, y por último la **salida**, que se mostrará a través de 4 leds.

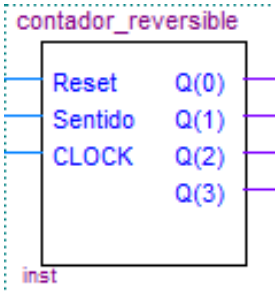

Figura 3: Contador reversible de módulo 10

Se deberá entregar el código VHDL con sus respectivos comentarios, el cronograma de simulación con todos los posibles estados de entradas y salidas, contrastándolo con la tabla de verdad, y el archivo con extensión *.csv*, que corresponde a la asignación de pines.

4.2. Contador prediseñado con muestra del resultado por pantalla LCD

4.2.1. Contador Prediseñado

Diseñar un contador prediseñado de módulo 10, que vaya de 0 a 9, utilizando la descripción (funcional, de flujo de datos o estructural) que se desee.

El contador deberá tener una entrada **enable**, que se controlará con un interruptor, una **entrada de reloj**, que será el pulsador para incrementar o decrementar el valor de la salida, una **entrada de reset**, que será otro pulsador, una **entrada de load**, que será un tercer pulsador, la **entrada para introducir el valor de la carga**, que se introducirá mediante 4 interruptores, una **entrada para elegir el sentido**, en este caso se utilizará otro interruptor, y por último la **salida**, que se mostrará a través de 4 leds.

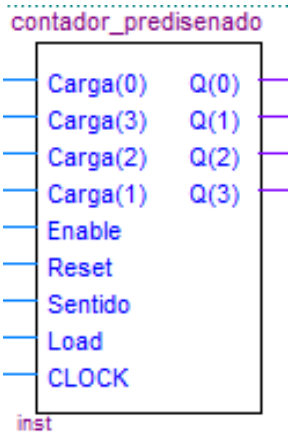

Figura 4: Contador prediseñado de módulo 10

Para comprobar que el contador funciona correctamente, se pide simular el diseño y que a la salida se muestre la siguiente secuencia:

0125676095|0125676095|012…

4.2.2. Mostrar el resultado del contador por pantalla LCD

Una vez que se haya realizado el diseño del contador prediseñado, en lugar de mostrar la salida mediante 4 leds, se deberá mostrar el valor del contador por la pantalla LCD.

En la primera línea se deberá mostrar el sentido del contador, si el contador esta en modo ascendente se deberá ver la palabra “Ascendente” y si va en sentido decreciente se mostrara “Descendente”. En la segunda línea se verá el valor de la salida mostrando la palabra “Contador = “ seguida del valor de la salida.

Se deberá entregar los dos códigos VHDL con sus respectivos comentarios, el cronograma de simulación del contador prediseñado con la secuencia especificada y los archivos con extensión *.csv*, que corresponden a la asignación de pines.

Se deberá mostrar al profesor de prácticas que los ejercicios de diseño se han volcado a la tarjeta DE2 y que funcionan correctamente.

Resolución de la práctica 4

3.1. Biestable JK

Se ha simulado el código, que aparece en el guión, para obtener el comportamiento de un biestable JK como este:

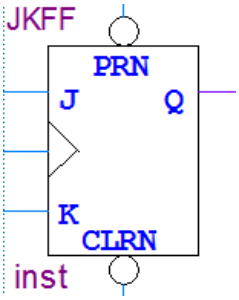

Figura 1: Biestable JK síncrono

A continuación se muestra el cronograma de simulación, en donde a la entrada clock se le ha asignado una frecuencia bastante más alta que a las entradas j y k, para así ver con más facilidad el funcionamiento del biestable JK.

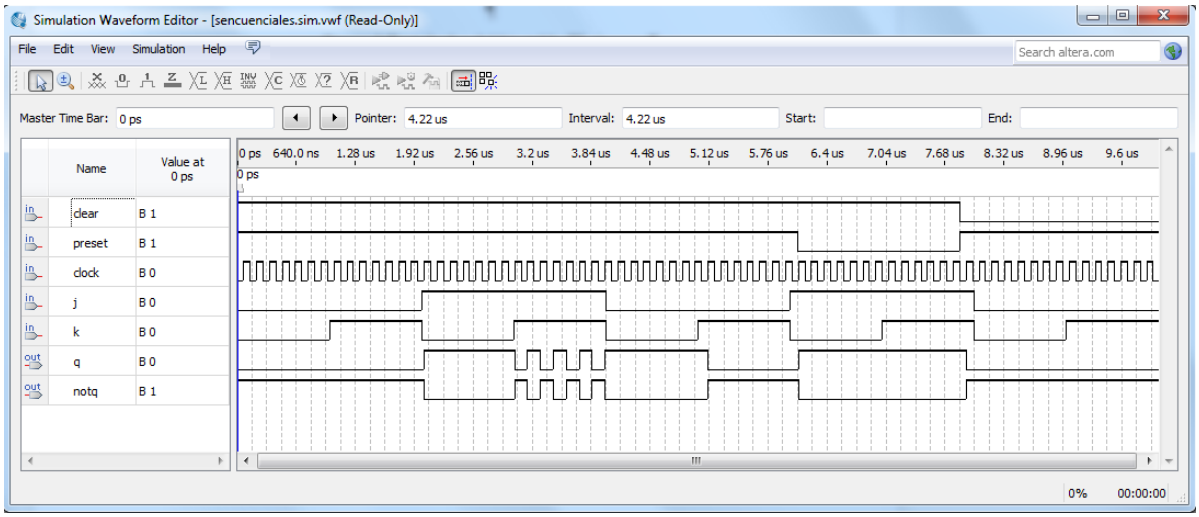

Figura 2:Restultados del biestable JK síncrono

Los resultados obtenidos de la simulación son correctos, las funciones reset y clear ponen la salida a '1' y a '0' respectivamente y la salida adopta el valor correcto dependiendo de las diferentes combinaciones de las entradas j y k, como aparece en la siguiente tabla.

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Not(Q(t))

Tabla 1: Tabla de verdad del biestable JK (sin preset y clear)

3.2. Contador síncrono ascendente de módulo 4

En este ejercicio se diseña un contador síncrono ascendente de módulo 4 muy simple, el cual tiene una única entrada, que es la señal de reloj, y dos salidas, que dan el valor del contador. Así pues, el contador irá aumentando su valor cuando la señal de reloj dé un pulso, hasta llegar al valor "11", pues con el siguiente pulso el valor del contador se reiniciará y volverá al "00". La tabla de verdad sería la siguiente:

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Not(Q(t))

Tabla 2: Tabla de verdad del contador del ejercicio 3.2.

Con la simulación del código se ve claramente el funcionamiento de este sencillo contador:

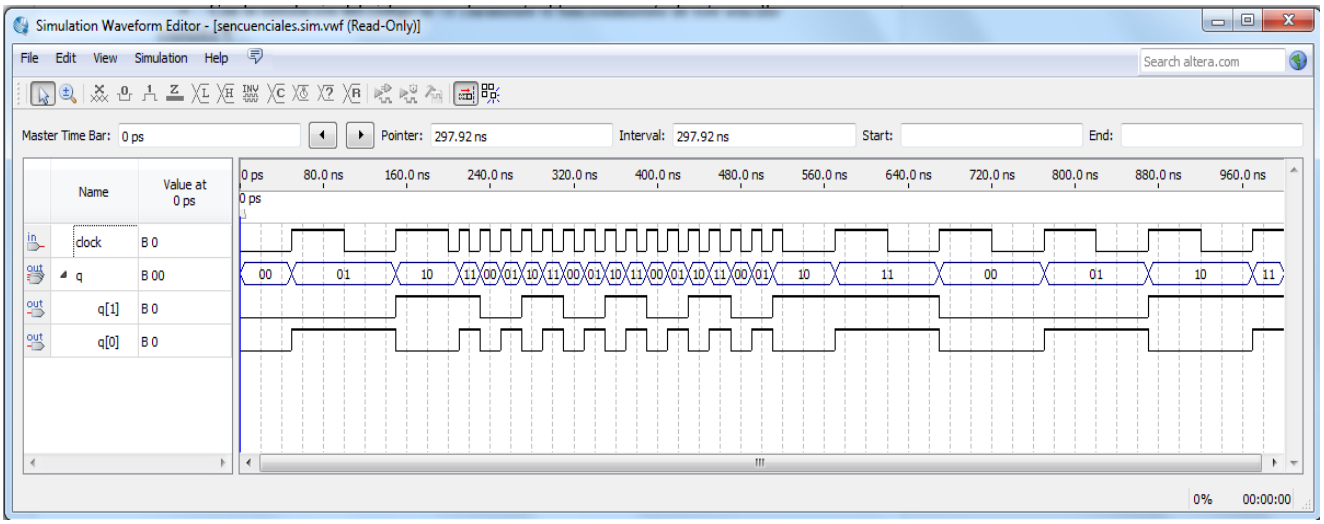

Figura 3: Resultados del contador síncrono ascendente de módulo 4

De nuevo, se ha jugado con el valor de la frecuencia del reloj para que se aprecie sin problemas el comportamiento del contador. En este caso se ha empezado con una frecuencia más lenta, luego se ha pasado a una bastante más rápida y por último se ha vuelto a una señal con la frecuencia más baja, de esta forma se ve como el valor de la salida va cambiando con diferentes señales de reloj y que no funciona con una solamente.

3.3. Pantalla LCD

a) Las señales char (char1,char2,…) se inicializan todas con el valor X"FE" porque en código ASCII ese valor corresponde a un espacio en blanco, de forma que si en el diseño no se utiliza, por ejemplo, la señal char14, en la posición 14 aparecerá en blanco debido al valor con el que se ha inicializado esa señal. Si no se le asignará un valor que correspondiese a un espacio en blanco, las posiciones que no se utilizaran mostrarían cualquier tipo de carácter.

b) A continuación se muestra el código, ya comentado, para mostrar por pantalla los nombres de los integrantes de cada grupo, en este caso mostraremos "ISMAEL".

```
library IEEE;                                                    -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;                                    -- Paquetes de la librería estándar de la IEEE

entity ejemplo_lcd is
   port(  clock: in STD_LOGIC;                                   -- Señal de entrada del reloj de 50 MHz
          reset_lcd:in STD_LOGIC;                                -- Señal de reset de la LCD
          lcd_rs,lcd_e,lcd_on: OUT STD_LOGIC;                    -- Señales de salida a los pines RS, Enable y ON
          lcd_rw: BUFFER STD_LOGIC;                              -- Señal de salida al pin R/W
          datos: inout STD_LOGIC_VECTOR (7 DOWNTO 0):=x"00");    -- Vector de salida de 8 bits de datos
end ejemplo_lcd;

architecture nombres_lcd of ejemplo_lcd is                       -- Cabecera del programa
   component lcd is                                              -- Declaración del componente lcd
      port( reset,clk_50Mhz: IN STD_LOGIC;                       -- Señal de reset y de reloj
          char1,char2,char3,char4,char5,char6,char7,char8,        -- Vectores de entrada para cada carácter
          char9,char10,char11,char12,char13,char14,char15,
          char16,char17,char18,char19,char20,char21,char22,
          char23,char24,char25,char26,char27,char28,char29,
          char30,char31,char32: in STD_LOGIC_VECTOR (7 DOWNTO 0);
          LCD_RS, LCD_E, LCD_ON: OUT STD_LOGIC;                  -- Bit RS, Enable y ON
          LCD_RW: BUFFER STD_LOGIC;                              -- Bit R/W
          DATA_BUS: INOUT          STD_LOGIC_VECTOR(7 DOWNTO 0)); -- Vector de salida para mostrar el carácter
   end component;

   signal char1,char2,char3,char4,char5,char6,char7,char8,        -- Declaración de señales
          char9,char10,char11,char12,char13,char14,char15,
           char16,char17,char18,char19,char20,char21,char22,
           char23,char24,char25,char26,char27,char28,char29,
           char30,char31,char32: STD_LOGIC_VECTOR (7 DOWNTO 0):=X"FE";
          ...
```

```
        ...
begin
  pantalla: lcd                                                    -- Referencia de componente
        port map(reset_lcd,clock,char1,char2,char3,char4,char5,    -- Asignación de señales a los puertos
                 char6,char7,char8,char9,char10,char11,char12,char13,
                 char14,char15,char16,char17,char18,char19,char20,char21,
                 char22,char23,char24,char25,char26,char27,char28,char29,
                 char30,char31,char32,lcd_rs,lcd_e,lcd_on,lcd_rw,datos);
  process(clock)                                                   -- Cabecera de proceso
  begin
    char6<=X"49";                                                  -- Mostrar el caracter "I" en la posición 6
    char7<=X"53";                                                  -- Mostrar el caracter "S" en la posición 7
    char8<=X"4D";                                                  -- Mostrar el caracter "M" en la posición 8
    char9<=X"41";                                                  -- Mostrar el caracter "A" en la posición 9
    char10<=X"45";                                                 -- Mostrar el caracter "E" en la posición 10
    char11<=X"4C";                                                 -- Mostrar el caracter "L" en la posición 11
  end process;                                                     -- Final de proceso
end nombres_lcd;                                                   -- Final del programa
```

4. Ejercicios de diseño

4.1. Contador reversible

Para este diseño se ha elegido la descripción funcional, ya que es más sencilla e intuitiva para este tipo de circuitos contadores.

Teniendo en cuenta la descripción escogida se ha elaborado el siguiente código para el diseño de un contador reversible de módulo 10:

```
library IEEE;                                          -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;                  -- Paquetes de la librería estándar de la IEEE
entity cont_reversible is
   port(  clock: in STD_LOGIC;                         -- Señal de reloj de 50 MHz
          sentido: in STD_LOGIC;                       -- Señal para indicar el sentido del contador
          reset: in STD_LOGIC;                         -- Señal de reset
          Q: out STD_LOGIC_VECTOR (3 DOWNTO 0));       -- Señal de salida
end cont_reversible;

architecture conta of cont_reversible is               -- Cabecera del programa
   signal contador: STD_LOGIC_VECTOR (3 downto 0);     -- Declaración de señal
begin
   process(clock)                                      -- Cabecera de proceso
   begin
      if reset='0' then                                -- Sentencia if para comprobar la pulsación del reset
          contador<="0000";                            -- Si se pulsa el pulsador de reset contador='0000'
      elsif clock'event and clock='1' then             -- Sentencia if para comprobar la pulsación del pulsador
          if sentido='0' then                          -- Sentencia if para comprobar el sentido del contador
             if contador<"1001" then                   -- Sentencia if para detectar si el contador a llegado a 9
                contador<=contador + 1;                -- y debe reinicializarse o si debe seguir aumentando su valor
             elsif contador="1001" then
                contador<="0000";
             end if;
          else
             if contador>"0000" then                   -- Sentencia if para detectar si el contador a llegado a 0
                contador<=contador - 1;                -- y debe reinicializarse o si debe seguir decrementando su valor
             elsif contador="0000" then
                contador<="1001";
             end if;
          end if;
      end if;
      Q<=contador;                                     -- Traspaso de datos de la señal contador a la salida Q
          end process;                                 -- Final de proceso
end conta;                                             -- Final del programa
```

Una vez que ya se ha obtenido el código, se procede a simularlo para comprobar si funciona correctamente.

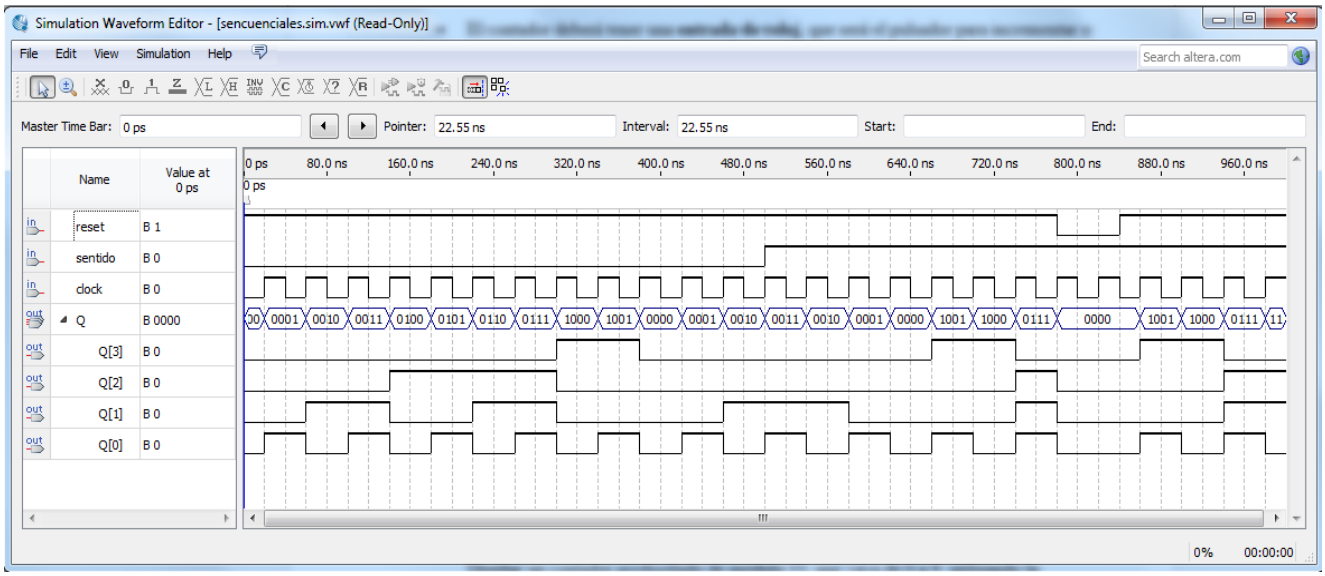

Figura 4: Resultados del contador reversible del ejercicio 4.1.

En el cronograma se ve que si la señal de entrada sentido está a '0' el contador aumenta su valor con cada pulsación y cuando está a '1' lo decrementa. Además si la señal de reset toma el valor '0', se reinicia el contador y la salida toma el valor "0000", hasta que la señal de reset vuelve a tomar el valor '1' y el contador sigue aumentando o decrementando su valor con cada pulsación, dependiendo del valor de la señal sentido.

La secuencia si el sentido es creciente debería ser:

0 → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 0 → 1 → …

La secuencia si el sentido es decreciente debería ser:

0 → 9 → 8 → 7 → 6 → 5 → 4 → 3 → 2 → 1 → 0 → 9 → …

A continuación, se muestra la tabla de verdad de este contador para se vea más fácil que los resultados de la simulación coinciden con lo esperado:

reset	sentido	Q3(t)	Q2(t)	Q1(t)	Q0(t)	Q3(t+1)	Q2(t+1)	Q1(t+1)	Q0(t+1)
1	0	0	0	0	0	0	0	0	1
1	0	0	0	0	1	0	0	1	0
1	0	0	0	1	0	0	0	1	1
1	0	0	0	1	1	0	1	0	0
1	0	0	1	0	0	0	1	0	1
1	0	0	1	0	1	0	1	1	0
1	0	0	1	1	0	0	1	1	1
1	0	0	1	1	1	1	0	0	0
1	0	1	0	0	0	1	0	0	1
1	0	1	0	0	1	0	0	0	0
1	1	0	0	0	0	1	0	0	1
1	1	0	0	0	1	0	0	0	0
1	1	0	0	1	0	0	0	0	1
1	1	0	0	1	1	0	0	1	0
1	1	0	1	0	0	0	0	1	1
1	1	0	1	0	1	0	1	0	0
1	1	0	1	1	0	0	1	0	1
1	1	0	1	1	1	0	1	1	0
1	1	1	0	0	0	0	1	1	1
1	1	1	0	0	1	1	0	0	0
0	X	X	X	X	X	0	0	0	0

Tabla 3: Tabla de verdad del contador reversible de módulo 10

4.2. Contador Prediseñado con muestra del resultado en pantalla LCD

4.2.1. Contador Prediseñado

Se ha diseñado el contador prediseñado de módulo 10 que aparece en la figura 5 mediante descripción funcional, por los motivos explicados anteriormente.

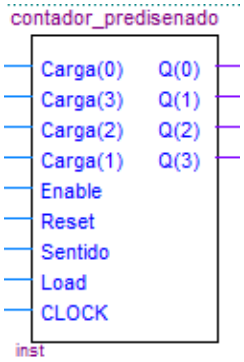

Figura 5: Contador prediseñado de módulo 10

Aquí se muestra el algoritmo funcional que se ha elaborado para diseñar el contador requerido en este ejercicio:

```
library IEEE;                                          -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;                          -- Paquetes de la librería estándar de la IEEE
entity cont_predisenado is
  port(   clock: in STD_LOGIC;                         -- Señal del pulsador
          enable: in STD_LOGIC;                        -- Señal del enable
          sentido: in STD_LOGIC;                       -- Señal para indicar el sentido del contador
          reset: in STD_LOGIC;                         -- Señal de reset
          load: in STD_LOGIC;                          -- Señal de load para cargar en el contador el valor de la señal carga
          carga: in STD_LOGIC_VECTOR (3 DOWNTO 0);     -- Señal que recoge el valor para cargar a través de la señal load
          Q: out STD_LOGIC_VECTOR (3 DOWNTO 0));       -- Señal de salida
end cont_predisenado;
architecture contador of cont_predisenado is           -- Cabecera del programa
  signal cont: STD_LOGIC_VECTOR (3 downto 0):="0000";  -- Declaración de señal
begin
  process(clock)                                       -- Cabecera de proceso
  begin
    if enable='0' then                                 -- Sentencia if para comprobar el enable
        cont<="0000";
    elsif reset='0' then                               -- Sentencia elsif para comprobar la pulsación de reset
        cont<="0000";                                  -- Si reset='0' la salida se reinicia a "0000" sino sigue contando
    elsif load='0' then                                -- Sentencia elsif para comprobar la pulsación de load
        cont<=carga;                                   -- Si load='0' la salida toma el valor de carga
    elsif clock'event and clock='1' then               -- Sentencia if para comprobar la pulsación del pulsador
        ...
```

```
        ...
        if sentido='0' then                    -- Sentencia if para detectar el sentido del contador
           if cont<"1001" then                 -- Si sentido='0' aumenta
              cont<=cont + 1;
           elsif cont="1001" then              -- Sentencia if para detectar si el contador a llegado a 9
              cont<="0000";                    -- y debe reinicializarse o si debe seguir aumentando
           end if;
        else                                   -- Si sentido='1' disminuye
           if cont>"0000" then                 -- Sentencia if para detectar si el contador a llegado a 0
              cont<=cont - 1;                  -- y debe reinicializarse o si debe seguir decrementando
           elsif cont="0000" then
              cont<="1001";
           end if;
        end if;
     end if;
     Q<=cont;                                  -- Traspaso de los datos de la señal cont a la salida Q
  end process;                                 -- Final de proceso
end contador;                                  -- Final del programa
```

Ahora se puede observar el cronograma que se ha obtenido tras la simulación del código que se acaba de mostrar.

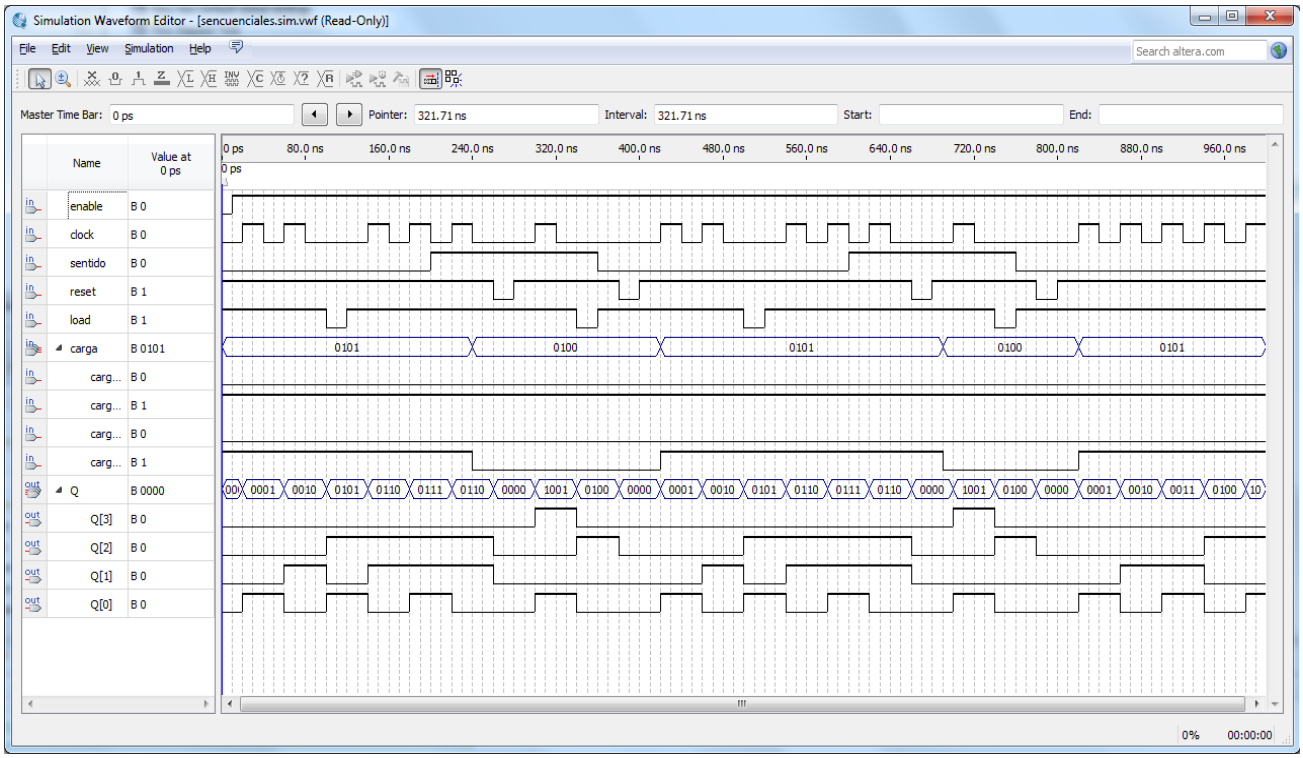

Figura 6: Resultados del contador prediseñado del ejercicio 4.2.1.

En este cronograma, como se ha propuesto en el enunciado del ejercicio, se ha simulado la siguiente secuencia:

0125676094|0125676094|…

- Los tres primeros valores (0,1 y 2) se consiguen con la señal de sentido a '0' para que el contador se comporte en modo ascendente.
- El 5 se consigue mediante la pulsación del load, asignando a la señal carga el valor "0101".
- Los dos siguientes valores (6 y 7) se consiguen de la misma forma que los tres primeros.
- Para el obtener el siguiente valor (6), se cambia el valor de la señal sentido a '0', entonces el contador pasa a modo descendente y pasa del 7 al 6.
- Después se tiene que conseguir el valor 0 a la salida, este valor se obtiene pulsando el reset, para que así el contador se reinicie.
- De nuevo con la señal sentido a '0' se pasa del 0 al 9.
- Ahora se cambia el valor de la carga a 4 y se vuelve a pulsar el pulsador de load, obteniendo el 4 a la salida.
- Por último para volver al primer valor de la secuencia para que vuelva a producirse, se pulsa el reset para conseguir que la salida sea 0.

4.2.2. Mostrar el resultado del contador por pantalla LCD

A continuación se presenta el código para conseguir mostrar por la pantalla LCD de la placa DE2 el valor de la salida del contador, en lugar de mostrarla mediante leds.

```
library IEEE;                                              -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;                              -- Paquetes de la librería estándar de la IEEE
entity contador_con_lcd is
   port(  pulsador: in STD_LOGIC;                          -- Señal del pulsador
          enable: in STD_LOGIC;                            -- Señal del enable del contador
          clock: in STD_LOGIC;                             -- Señal del reloj de 50 MHz
          sentido: in STD_LOGIC;                           -- Señal para indicar el sentido
          reset_cont: in STD_LOGIC;                        -- Señal del reset del contador
          reset_lcd:in STD_LOGIC;                          -- Señal del reset de la LCD
          load: in STD_LOGIC;                              -- Señal de load para cargar el valor de la señal carga
          carga: in STD_LOGIC_VECTOR (3 DOWNTO 0);         -- Señal que recoge el valor para cargar
          lcd_rs,lcd_e,lcd_on: OUT STD_LOGIC;              -- Bits de RS, Enable y ON de la LCD
          lcd_rw: BUFFER STD_LOGIC;                        -- Bit R/W de la LCD
          datos: inout STD_LOGIC_VECTOR (7 DOWNTO 0):=x"00");  -- Señal de datos de la LCD
end contador_con_lcd;
architecture contlcd of contador_con_lcd is                -- Cabecera del programa
   component lcd is                                        -- Declaración del componente lcd
          ...
```

```
        ...
    PORT( reset,clk_50Mhz: IN STD_LOGIC;                          -- Señal de reset y de reloj
          char1,char2,char3,char4,char5,char6,char7,char8,         -- Vectores de entrada para cada caracter
          char9,char10,char11,char12,char13,char14,char15,
          char16,char17,char18,char19,char20,char21,char22,
          char23,char24,char25,char26,char27,char28,char29,
          char30,char31,char32: in STD_LOGIC_VECTOR (7 DOWNTO 0):=X"FE";
          LCD_RS, LCD_E, LCD_ON: OUT STD_LOGIC;                   -- Bit RS, Enable y ON
          LCD_RW: BUFFER STD_LOGIC;                               -- Bit R/W
          DATA_BUS: INOUT STD_LOGIC_VECTOR(7 DOWNTO 0));          -- Vector de salida para llevar los datos a la LCD
  end component;
  component cont_predisenado is                                   -- Declaración del componente cont_predisenado
    port( clock: in STD_LOGIC;                                    -- Señal del pulsador
          enable: in STD_LOGIC;                                   -- Señal del enable
          sentido: in STD_LOGIC;                                  -- Señal para indicar el sentido del contador
          reset: in STD_LOGIC;                                    -- Señal de reset
          load: in STD_LOGIC;                                     -- Señal de load para cargar en el contador el valor
                                                                  -- de la señal carga
          carga: in STD_LOGIC_VECTOR (3 DOWNTO 0);                -- Señal que recoge el valor para cargar a través de
                                                                  -- la señal load
          Q: out STD_LOGIC_VECTOR (3 DOWNTO 0));                  -- Señal de salida
  end component;
  signal numero: STD_LOGIC_VECTOR (3 DOWNTO 0):=x"0";             -- Declaración de señales
  signal char1,char2,char3,char4,char5,char6,char7,char8,
          char9,char10,char11,char12,char13,char14,char15,
          char16,char17,char18,char19,char20,char21,char22,
          char23,char24,char25,char26,char27,char28,char29,
          char30,char31,char32: STD_LOGIC_VECTOR (7 DOWNTO 0):=X"FE";
begin
  contador: cont_predisenado                                      -- Referencia de componente
          port map( pulsador,enable,sentido,reset_cont, load,carga,numero); -- Asignación de señales a los puertos
  pantalla: lcd                                                   -- Referencia de componente
          port map( reset_lcd,clock,char1,char2,char3,char4,char5,  -- Asignación de señales a los puertos
                    char6,char7,char8,char9,char10,char11,char12,
                    char13,char14,char15,char16,char17,char18,
                    char19,char20,char21,char22,char23,char24,
                    char25,char26,char27,char28,char29,char30,
                    char31,char32,lcd_rs,lcd_e,lcd_on,lcd_rw,datos);
  process(numero)                                                 -- Cabecera de proceso
  begin
    if sentido='0' then                                           -- Sentencia if para comprobar la señal sentido
          char1<=X"41";                                           -- Si sentido='0' que se muestre en la línea 1
          char2<=X"53";                                           -- de la LCD la palabra "ASCENDENTE"
          char3<=X"43";
          char4<=X"45";
          char5<=X"4E";
          char6<=X"44";
          char7<=X"45";
          char8<=X"4E";
          ...
```

```
        ...
        char9<=X"54";
        char10<=X"45";
        char11<=X"FE";
    elsif sentido='1' then                    -- Si sentido='0' que se muestre en la línea 1
        char1<=X"44";                         -- de la LCD la palabra "DESCENDENTE"
        char2<=X"45";
        char3<=X"53";
        char4<=X"43";
        char5<=X"45";
        char6<=X"4E";
        char7<=X"44";
        char8<=X"45";
        char9<=X"4E";
        char10<=X"54";
        char11<=X"45";
    end if;
        char17<=X"43";                        -- Mostrar en la línea 2 de la LCD
        char18<=X"4F";                        -- "CONTADOR = " y el valor del contador
        char19<=X"4E";
        char20<=X"54";
        char21<=X"41";
        char22<=X"44";
        char23<=X"4F";
        char24<=X"52";
        char25<=X"FE";
        char26<=X"3D";
        char27<=X"FE";
        char28<= numero + X"30";
  end process;                                -- Final de proceso
end contlcd;                                  -- Final del programa
```

Práctica 5: Máquinas de Estados

Se conoce como máquina de estados a un modelo de comportamiento de un sistema con entradas y salidas, en donde las salidas dependen tanto de las señales de entrada actuales como también de las anteriores. Dentro de las máquinas de estados se conocen como máquinas de estados finitos si el conjunto de estados de la máquina es finito.

Se van a estudiar dos tipos de máquinas de estados finitos: la Máquina de Moore y la Máquina de Mealy.

Máquina de Moore: las salidas están determinadas por el estado actual únicamente y no dependen directamente de las entradas.

Máquina de Mealy: las salidas son generadas a partir del estado actual y del valor de las entradas

Con estos dos tipos de máquinas de estados realizaremos los diseños que se propondrán en esta práctica.

1. Objetivos

- Realizar diseños mediante circuitos secuenciales síncronos a través de la Máquina de Mealy y la Máquina de Moore.
- Ver como se implementan las máquinas de estados mediante lenguaje VHDL.

2. Materiales

- Ordenador personal con el software QUARTUS II disponible.
- Tarjeta educacional DE2
- Guión de prácticas

3. Desarrollo de la práctica

3.1. Ejemplo máquina de estados

Para diseñar una máquina de estados mediante lenguaje VHDL se debe seguir una determinada estructura en el diseño. Primero se realiza la codificación de los estados, después se codifica la transición de los estados, en donde también se asigna valores a las señales de salida, y, por último, se elabora una lógica que permite pasar de estado.

Codificación de estados

Para codificar los estados, se va a definir un tipo de dato, el cual tenga como valores los diferentes estados de la máquina. La definición del nuevo tipo de dato se realiza dentro de la arquitectura del código, en la zona donde se definen las señales.

Para llegar a crear un tipo de dato nuevo utiliza la opción ***TYPE***:

```
TYPE estado IS (estado1,estado2,estado3…);          -- Se define el tipo de dato "estado"

SIGNAL estado_actual,estado_siguiente: estado;      -- Se definen las señales estado_actual y
                                                    -- estado_siguiente como tipo "estado".
```

De esta forma al tipo de dato "estado" se le asigna como valores los diferentes estados que se requieran y posteriormente a las señales, que se van a usar para el estado actual y el estado siguiente, se definen de tipo "estado", para que así tengan como valores los diferentes estados de la máquina.

Codificación de la transición de estados

En esta parte del código es donde difieren la Máquina de Moore y la Máquina de Mealy. Ambas estructuras son similares pero a la hora de asignar los valores a las señales de salida es cuando se ve la diferencia entre ellas.

A continuación se muestra la estructura de la codificación de la transición de estados para Máquinas de Moore.

```
Moore: process(entrada,estado_actual)
begin
   case estado_actual is
      when estado1=>
           salidas<=valor;    -- Asignación de valores a las señales de salida para el estado1
           -- Cálculo del estado siguiente en función de las entradas
           ...
      when estado2=>
           salidas<=valor;    -- Asignación de valores a las señales de salida para el estado2
           -- Cálculo del estado siguiente en función de las entradas
           ...
      when estado_3=>
           salidas<=valor;    -- Asignación de valores a las señales de salida para el estado3
           -- Cálculo del estado siguiente en función de las entradas
           ...
           ...
   end case;
end process;
```

Ahora para la Máquina de Mealy:

```
Mealy: process(entrada,estado_actual)
begin
   case estado_actual is
      when estado1=>
            -- Cálculo de las señales de salida en función de las entradas
            -- Cálculo del estado siguiente en función de las entradas
            ...
      when estado2=>
            -- Cálculo de las señales de salida en función de las entradas
            -- Cálculo del estado siguiente en función de las entradas
            ...
      when estado_3=>
            -- Cálculo de las señales de salida en función de las entradas
            -- Cálculo del estado siguiente en función de las entradas
            ...
            ...
   end case;
end process;
```

Posteriormente, en este mismo ejercicio, se mostrará un ejemplo de una máquina de estados sencilla, en donde esto se entenderá mejor.

Lógica para pasar de estado

Como se trata de circuitos secuenciales síncronos, el estado debe cambiar con la señal de reloj, por lo que ahora se va a mostrar el proceso a través del cual se implementa esta función.

```
process(clock)
begin
   if clock'event and clock='1' then -- Se comprueba si ha llegado el flanco de subida del reloj
      estado_actual<=estado_siguiente;       -- Se pasa al estado siguiente
   else
      estado_actual<=estado_actual;
   end if;
end process;
```

A continuación se va a mostrar un pequeño ejemplo muy sencillo, que desarrolla un poco más la estructura de código que se acaba de explicar.

Este ejemplo corresponde a una Máquina de Mealy con tan sólo dos estados, una entrada y una salida, cuyo diagrama de estados es el siguiente:

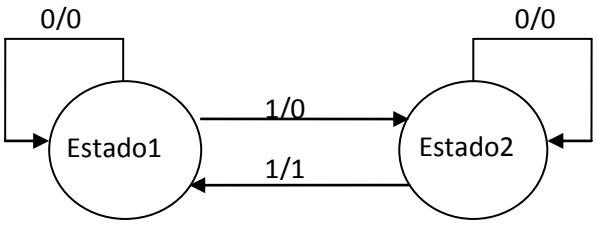

Figura 1: Diagrama de estados del ejemplo 3.1.

La tabla de transiciones también resulta muy sencilla, en ella se ve claramente como el estado y la salida dependen del estado actual y de la entrada.

Estado actual (Q(t))	Entrada	Estado siguiente (Q(t+1))	Salida
Estado1	0	Estado1	0
	1	Estado2	0
Estado2	0	Estado2	0
	1	Estado1	1

Tabla 1: Tabla de transiciones del ejemplo 3.1.

Ahora que ya se ha visto el comportamiento del sistema, se elabora el código para implementarlo.

```
library IEEE;                                   -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;                   -- Paquetes de la librería estándar de la IEEE

entity ejemplo is
   port( entrada: in STD_LOGIC;                 -- Señal de entrada
         reset,clock: in STD_LOGIC;             -- Señales de reset y de reloj
         salida: out STD_LOGIC);                -- Señal de salida
end ejemplo;

architecture estados of ejemplo is              -- Cabecera del programa
   type estado is (estado1,estado2);            -- Declaración de tipo de dato
   signal est_actual,est_siguiente: estado;     -- Declaración de señal
begin
...
```

```
...
process(clock,reset)                    -- Cabecera de proceso
begin
   if reset='0' then                    -- Sentencia if para detectar la pulsación del reset
      est_actual<=estado1;
      elsif clock'event and clock='1' then  -- Sentencia elsif para detectar el flanco de subida
      est_actual<=est_siguiente;        -- de la señal de reloj
      end if;
end process;                            -- Final de proceso

process(entrada,est_actual)             -- Cabecera de proceso
begin
   case est_actual is                   -- Sentencia case para comprobar el valor del estado actual
      when estado1=>                    -- Estado1 es el estado actual
           if entrada='0' then          -- Sentencia if para el cálculo del estado siguiente en función de las entradas
              salida<='0';              -- y para el cálculo de las señales de salida en función de las entradas
              est_siguiente<=estado1;
           elsif entrada='1' then
              salida<='0';
              est_siguiente<=estado2;
           end if;
      when estado2=>                    -- Estado2 es el estado actual
           if entrada='0' then          -- Sentencia if para el cálculo del estado siguiente en función de las entradas
              salida<='0';              -- y para el cálculo de las señales de salida en función de las entradas
              est_siguiente<=estado2;
           elsif entrada='1' then
              salida<='1';
              est_siguiente<=estado1;
           end if;
    end case;
end process;                            -- Final del proceso
end estados;                            -- Final del programa
```

Ejercicio

Simular el ejemplo que se acaba de mostrar y comprobar su comportamiento, que pase de un estado a otro correctamente y que la salida sea la esperada en cada momento.

Nota: Hay que tener en cuenta, que el cambio de estado se produce con un flanco de subida en la señal de reloj, mientras que la salida cambia su valor en el momento que la entrada cambia, por lo que las señales no tienen porque cambiar a la vez.

4. Ejercicios de diseño

4.1. Sistema de transferencia de energía

Se trata de implementar un sistema síncrono secuencial, por Máquina de Moore, que sea capaz de entregar diferentes tipos de señales en función de una señal de entrada.

Los tres tipos de señal de salida tienen las siguientes características:

- S1: Relación de aspecto 1 a 1 (1 periodo a '1', 1 periodo a '0').
- S2: Relación de aspecto 2 a 1 (2 periodo a '1', 1 periodo a '0').
- S3: Relación de aspecto 3 a 1 (3 periodo a '1', 1 periodo a '0').

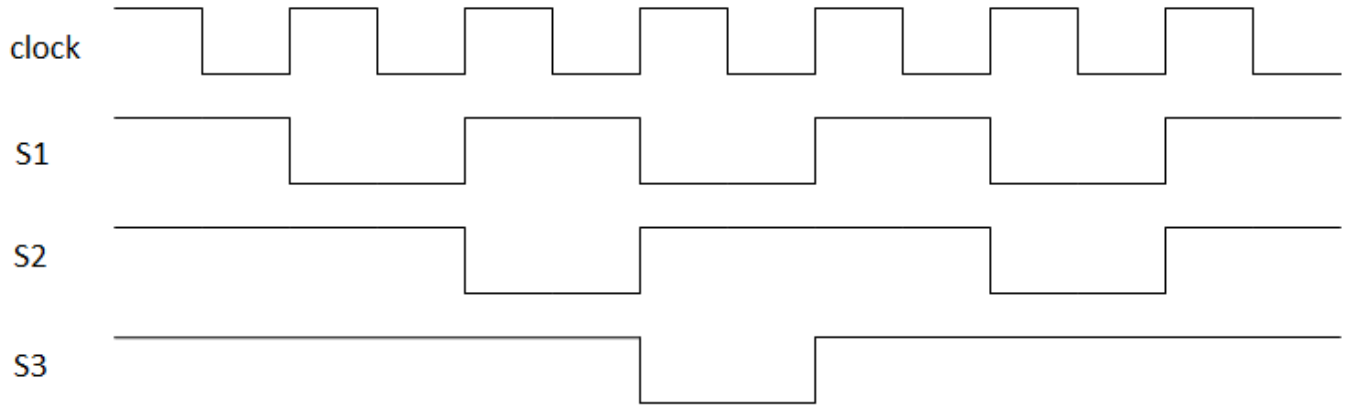

Figura 2: Formas de onda de la señal de reloj y de la señal de salida S

En un sistema donde la transferencia de energía está controlada por este tipo de señal significaría que la forma S1 entrega menos energía que la señal S2, y ésta menos energía que la señal S3.

Para controlar dicha transferencia de energía hay una señal de entrada I, tal que:

- Si la energía que se está entregando tiene que subir I = '1'.
- Si la energía que se está entregando tiene que bajar I='0'.

El circuito que se pide diseñar tendrá dos entradas I y clock (reloj) y una única salida S, con las tres formas de onda S1, S2 y S3 ya mencionadas. Es posible pasar de la forma de onda S1 a S2, pero no un salto brusco de S1 a S3. De igual forma es posible pasar de S3 a S2 pero no de S3 a S1. Evidentemente de S2 se puede pasar a S1 y/o S3 en función de la entrada I.

El cambio de una forma de onda se realizará cuando se haya concluido un ciclo completo de forma precedente.

Para el volcado a la tarjeta se utilizará un interruptor para la entrada I y un led para la salida S.

Se deberá entregar el código VHDL con sus respectivos comentarios, el diagrama de estados, la tabla de transiciones y el archivo con extensión *.csv*, que corresponde a la asignación de pines.

Nota: En lugar de utilizar la señal de reloj de 50 MHz directamente, se usará un divisor de frecuencia de 1 Hz, de esta forma en el volcado se apreciará perfectamente los valores de la salida S.

4.2. Cerradura Digital

En este ejercicio se deberá diseñar una cerradura digital que debe cumplir con las siguientes especificaciones:

- Se establecerá la combinación de apertura de la cerradura mediante 3 pulsadores y se mostrarán los 3 dígitos de la combinación a través de 3 displays 7-segmentos. Una vez que se tenga la contraseña deseada se pulsará el pulsador de "OK", para guardar la contraseña.
  Cuando se esté estableciendo la contraseña de apertura se deberá mostrar por la pantalla LCD lo siguiente: "ESTABLECE CONTRASENA NUEVA".
- Para abrir la cerradura se introducirá la combinación de la misma forma que para establecerla, mediante los mismos 3 pulsadores y se mostrará por los mismos 3 displays 7-segmentos. Una vez que se tenga la combinación que se desea introducir se pulsará de nuevo el pulsador de "OK".
  Una vez que se haya introducido la contraseña y pulsado el "OK", independientemente de si es correcta o no, los displays 7-segmentos deben borrar la contraseña introducida y mostrar todo ceros.
  Cuando se esté introduciendo la contraseña se deberá ver escrito en la pantalla LCD lo siguiente: "INTRODUCE CONTRASENA".
- Si la contraseña introducida coincide con la combinación de apertura que se ha establecido anteriormente, la cerradura se abrirá, 5 leds verdes se encenderán y aparecerá en la pantalla LCD lo siguiente: "CONTRASENA CORRECTA".
  Para cerrar la cerradura y poder abrirla de nuevo se pulsará el pulsador de "OK".
- Si la contraseña no coincide con la combinación establecida, la cerradura se mantendrá cerrada, 5 leds rojos se encenderán y aparecerá en la pantalla LCD lo siguiente: "CONTRASENA INCORRECTA".
  Para volver a introducir la contraseña e intentarlo de nuevo se pulsará el pulsador de "OK".
  Una vez que se ha introducido una combinación incorrecta,
- Una vez que se ha acertado la combinación de apertura y la cerradura se encuentra abierta, se podrá establecer de nuevo la combinación, si se desea cambiarla.
  Para ello, se utilizará un interruptor de "reset", de modo que en lugar de pulsar "OK" y cerrar la cerradura se pondrá el "reset" a '1' y se accederá a modificar la combinación de apertura como se ha explicado anteriormente.

Se pide implementar esta cerradura digital mediante Máquina de Mealy.

Se aconseja utilizar como componentes el diseño del contador binario, para introducir los 3 dígitos de las combinaciones, el del codificador 7-segmentos, para mostrar los dígitos por los displays, y el componente de la lcd, para mostrar por pantalla lo que se requiera. Todos ellos se han realizado en prácticas anteriores.

Se deberá entregar también el código VHDL con sus respectivos comentarios y el archivo con extensión *.csv*, que corresponde a la asignación de pines.

Nota: En lugar de utilizar la señal de reloj de 50 MHz directamente, se usará un divisor de frecuencia de 0,5 Hz, de esta forma en el volcado se apreciará perfectamente el funcionamiento de la cerradura.

Es importante tener en cuenta, que se debe mantener pulsado el pulsador "OK" hasta que el estado cambie, ya que, por definición, los sistemas secuenciales síncronos pasan de estado dependiendo del valor de la entrada cuando se produce un flanco de subida en la señal de reloj. Por lo que, si un pulso no coincide con el flanco de subida no será detectado.

Se deberá mostrar al profesor de prácticas que los ejercicios de diseño se han volcado a la tarjeta DE2 y que funcionan correctamente.

3. Resolución de la práctica 5

3.1. Ejemplo máquina de estados

Se ha simulado el diseño que se ha mostrado anteriormente y se ha obtenido el siguiente cronograma:

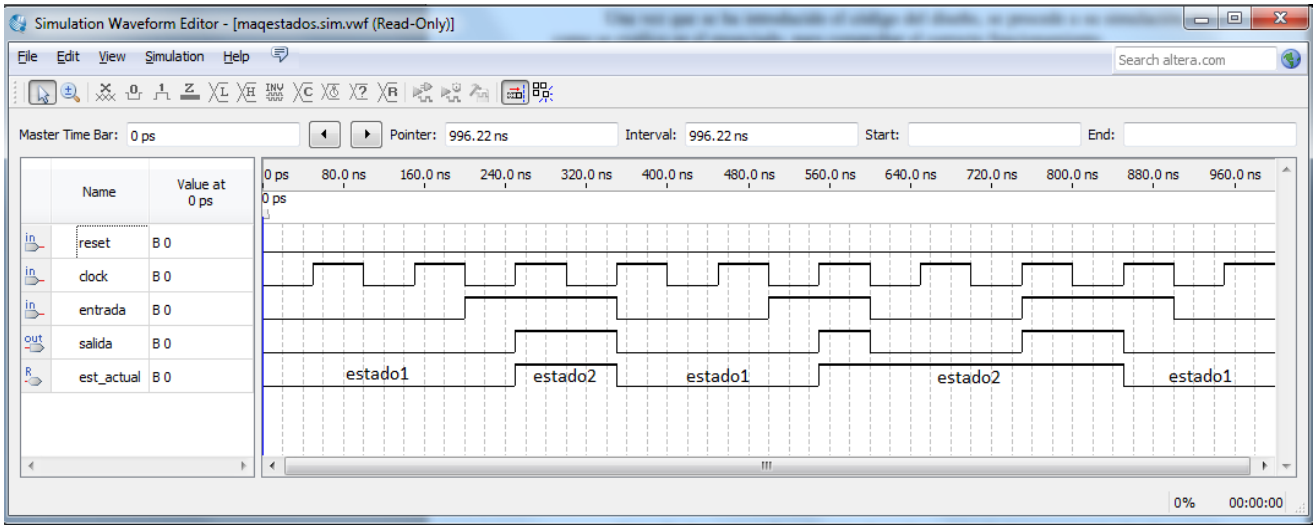

Figura 1: Resultados del ejemplo del ejercicio 3.1.

En la figura 1 se pueden observar todos los casos posibles del diseño, cuando estado actual es el estado1 la salida es '0', independientemente del valor de la entrada, sin embargo, cuando el estado es el estado2 la salida es '0' si la entrada es '0' y la salida es '1' si la entrada es '1'.

La salida cambia con el valor de la señal de entrada, mientras el cambio de estado se produce siempre cuando se produce un flanco de subida en la señal de reloj.

4. Ejercicios de diseño

4.1. Sistema de transferencia de energía

En primer lugar, se van a definir los estados por los que atraviesa el sistema:

q0: es el '1' de S1
q1: es el '0' de S1
q2: es el primer '1' de S2
q3: es el segundo '1' de S2
q4: es el '0' de S2
q5: es el primer '1' de S3
q6: es el segundo '1' de S3
q7: es el tercer '1' de S3
q8: es el '0' de S3

Una vez que se tienen definidos los estados se elabora la de transiciones y el diagrama de estados.

Estado actual (Q(t))	I	Estado siguiente (Q(t+1))	S
q0	0	q1	1
	1	q1	1
q1	0	q0	0
	1	q2	0
q2	0	q3	1
	1	q3	1
q3	0	q4	1
	1	q4	1
q4	0	q0	0
	1	q5	0
q5	0	q6	1
	1	q6	1
q6	0	q7	1
	1	q7	1
q7	0	q8	1
	1	q8	1
q8	0	q2	0
	1	q5	0

Tabla 1: Tabla de transiciones del ejercicio 4.1.

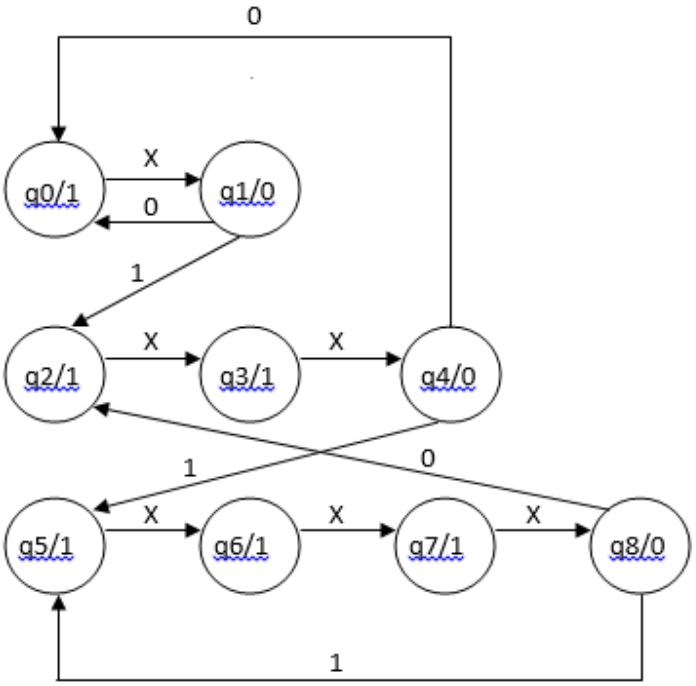

Figura 2: Diagrama de estados del ejercicio 4.1.

A continuación se muestra el código que se ha elaborado para implementar este sistema:

```
library IEEE;                                   -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;                   -- Paquetes de la librería estándar de la IEEE
entity prob_energia is
   port(  I: in STD_LOGIC;                      -- Señal de entrada
          clock: in STD_LOGIC;                  -- Señal de reloj
          S: out STD_LOGIC);                    -- Señal de salida
end prob_energia;
architecture maquina_estados of prob_energia is -- Cabecera del programa
   type estado is (q0,q1,q2,q3,q4,q5,q6,q7,q8); -- Declaración de tipo de dato
   signal est_actual,est_siguiente: estado;     -- Declaración de señales
   signal cont: integer range 0 to 125000000;
   signal clk_1Hz: STD_LOGIC;
begin
process(clock)                                  -- Cabecera de proceso
begin
   if clock'event and clock='1' then            -- Sentencia if para detectar el flanco de subida del reloj de 50 MHz
      if cont=24999999 then                     -- Sentencia if para generar una señal de 1 Hz de frecuencia
          clk_1Hz<=not clk_1Hz;
            cont<=0;
      else
          cont<=cont+1;
      end if;
   end if;
   ...
```

```
    ...
  if clk_1Hz'event and clk_1Hz='1' then          -- Sentencia if para detectar el flanco de subida del reloj de 1 Hz
    est_actual<=est_siguiente;                   -- y actualizar la señal del estado actual
  else
    est_actual<=est_actual;
  end if;
end process;                                     -- Final de proceso
process(I,est_actual)                            -- Cabecera de proceso
begin
  case est_actual is                             -- Sentencia case para comprobar el valor del estado actual
    when q0=>                                    -- Cuando el estado actual sea q0 ejecuta las siguientes sentencias
        S<='1';
        est_siguiente<=q1;
    when q1=>                                    -- Cuando el estado actual sea q1 ejecuta las siguientes sentencias
        S<='0';
        if I='0' then                            -- Sentencia if para ver que valor tiene la entrada
          est_siguiente<=q0;                     -- y pasar a un estado o a otro
        elsif I='1' then
          est_siguiente<=q2;
        end if;
    when q2=>                                    -- Cuando el estado actual sea q2 ejecuta las siguientes sentencias
        S<='1';
        est_siguiente<=q3;
    when q3=>                                    -- Cuando el estado actual sea q3 ejecuta las siguientes sentencias
        S<='1';
        est_siguiente<=q4;
    when q4=>                                    -- Cuando el estado actual sea q4 ejecuta las siguientes sentencias
        S<='0';
        if I='0' then                            -- Sentencia if para ver que valor tiene la entrada
          est_siguiente<=q0;                     -- y pasar a un estado o a otro
        elsif I='1' then
          est_siguiente<=q5;
        end if;
    when q5=>                                    -- Cuando el estado actual sea q5 ejecuta las siguientes sentencias
        S<='1';
        est_siguiente<=q6;
    when q6=>                                    -- Cuando el estado actual sea q6 ejecuta las siguientes sentencias
        S<='1';
        est_siguiente<=q7;
    when q7=>                                    -- Cuando el estado actual sea q7 ejecuta las siguientes sentencias
        S<='1';
        est_siguiente<=q8;
    when q8=>                                    -- Cuando el estado actual sea q8 ejecuta las siguientes sentencias
        S<='0';
        if I='0' then                            -- Sentencia if para ver que valor tiene la entrada
          est_siguiente<=q2;                     -- y pasar a un estado o a otro
        elsif I='1' then
          est_siguiente<=q5;
        end if;
  end case;
end process;                                     -- Final del proceso
end maquina_estados;                             -- Final del programa
```

4.2. Cerradura Digital

A continuación se mostrará el código elaborado para implementar la cerradura digital:

```
library IEEE;                                                    -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;                                    -- Paquetes de la librería estándar de la IEEE
entity cerradura is
  port( ok,pulsador1,pulsador2,pulsador3: in STD_LOGIC;          -- Señales de entrada de los pulsadores
        clock: in STD_LOGIC;                                     -- Señal del reloj de 50 MHz
        reset: in STD_LOGIC;                                     -- Señal del interruptor de reset
        mostrardisplay1,mostrardisplay2,
        mostrardisplay3: out STD_LOGIC_VECTOR (6 DOWNTO 0);      -- Señales de salida de 7 bits para los 3 displays 7-segmentos
        leds_verdes,leds_rojos: out STD_LOGIC_VECTOR (4 DOWNTO 0):="00000";   -- Señales de salida de 5 bits para los 5 leds
                                                                 -- rojos y los 5 verdes
        lcd_rs,lcd_e,lcd_on: OUT STD_LOGIC;                      -- Bits de RS, Enable y ON de la LCD
        lcd_rw: BUFFER STD_LOGIC;                                -- Bit R/W de la LCD
        datos: inout STD_LOGIC_VECTOR (7 DOWNTO 0):=x"00");      -- Señal de datos de la LCD
end cerradura;
architecture ejemplo of cerradura is                             -- Cabecera del programa
  component contbinario is                                       -- Declaración del componente del contador binario
    port( reset:in STD_LOGIC;                                    -- Canal de entrada del reset
        pulsador: in STD_LOGIC;                                  -- Canal de entrada del pulsador
        leds: out STD_LOGIC_VECTOR (3 downto 0));                -- 4 leds de salida
  end component;
  component segmentos7_0a9 is                                    -- Declaración del componente del codificador 7-segmentos
    port( num:in STD_LOGIC_VECTOR(3 downto 0);          -- 4 interruptores de entrada para introducir el numero en binario
        seg1: out STD_LOGIC_VECTOR(6 downto 0));                 -- 2 vectores de salida de 7 bits cada uno
  end component;
  component lcd is                                               -- Declaración del componente lcd
    PORT( reset,clk_50Mhz: IN STD_LOGIC;                         -- Señal de reset y de reloj
        char1,char2,char3,char4,char5,char6,char7,char8,          -- Vectores de entrada para cada caracter
        char9,char10,char11,char12,char13,char14,char15,
        char16,char17,char18,char19,char20,char21,char22,
        char23,char24,char25,char26,char27,char28,char29,
        char30,char31,char32: in STD_LOGIC_VECTOR (7 DOWNTO 0):=X"FE";
        LCD_RS, LCD_E, LCD_ON: OUT STD_LOGIC;                    -- Bit RS, Enable y ON
        LCD_RW: BUFFER STD_LOGIC;                                -- Bit R/W
         DATA_BUS: INOUT STD_LOGIC_VECTOR (7 DOWNTO 0));         -- Vector de salida para llevar los datos a la LCD
  end component;
  type estado is (meter_contra_nueva,introducir_contra,          -- Declaración de tipo de dato
                  contra_bien,contra_mal);
  signal est_actual,est_siguiente: estado:=meter_contra_nueva;   -- Declaración de las señales del estado actual y estado siguiente
  signal borrar: STD_LOGIC :='1';                                -- Declaración de la señal para reiniciar los contadores a 0
  signal cont: integer range 0 to 50000000;                      -- Declaración de la señal del contador para el divisor de frec.
  signal clk_0_5Hz: STD_LOGIC :='0';                             -- Declaración de la señal del reloj de 0,5 Hz
  signal num1,num2,num3, numero1,                                -- Declaración de las señales de los valores de los contadores
        numero2,numero3: STD_LOGIC_VECTOR (3 DOWNTO 0);          -- para establecer la combinación de apertura
  signal dig1,dig2,dig3, digito1,                                -- Declaración de las señales de los valores de los contadores
        digito2,digito3: STD_LOGIC_VECTOR (3 DOWNTO 0);          -- para introducir la contraseña
        ...
```

```
        ...
    signal display1_meter_contra,display2_meter_contra,      -- Declaración de las señales para mostrar en los displays 7-segmentos
           display3_meter_contra: STD_LOGIC_VECTOR (6 DOWNTO 0);-- los dígitos de la combinación que está estableciendo
    signal display1_introducir_contra,display2_introducir_contra,-- Declaración de las señales para mostrar en los displays 7-seg.
           display3_introducir_contra:STD_LOGIC_VECTOR(6 DOWNTO 0);-- los dígitos de la contraseña que se están introduciendo
    signal pulsador1_1,pulsador2_1,pulsador3_1: STD_LOGIC; -- Declaración de las señales de los pulsadores para aumentar el valor
                                                           -- del contador para establecer la combinación de apertura
    signal pulsador1_2,pulsador2_2,pulsador3_2: STD_LOGIC; -- Declaración de las señales de los pulsadores para aumentar el valor
                                                           -- del contador para introducir la contraseña
    signal char1,char2,char3,char4,char5,char6,char7,char8,          -- Declaración de las señales de los caracteres de la LCD
           char9,char10,char11,char12,char13,char14,char15,
           char16,char17,char18,char19,char20,char21,char22,
           char23,char24,char25,char26,char27,char28,char29,char30,
           char31,char32: STD_LOGIC_VECTOR (7 DOWNTO 0):=X"FE";
begin
    contador1_meter_contra: contbinario                          -- Referencia de componente
           port map(borrar,pulsador1_1,numero1);                 -- Asignación de señales a los puertos
    contador2_meter_contra: contbinario                          -- Referencia de componente
           port map(borrar,pulsador2_1,numero2);                 -- Asignación de señales a los puertos
    contador3_meter_contra: contbinario                          -- Referencia de componente
           port map(borrar,pulsador3_1,numero3);                 -- Asignación de señales a los puertos
    display7seg1_meter_contra: segmentos7_0a9                    -- Referencia de componente
           port map(numero1,display1_meter_contra);              -- Asignación de señales a los puertos
    display7seg2_meter_contra: segmentos7_0a9                    -- Referencia de componente
           port map(numero2,display2_meter_contra);              -- Asignación de señales a los puertos
    display7seg3_meter_contra: segmentos7_0a9                    -- Referencia de componente
           port map(numero3,display3_meter_contra);              -- Asignación de señales a los puertos
    contador1_introducir_contra: contbinario                     -- Referencia de componente
           port map(borrar,pulsador1_2,digito1);                 -- Asignación de señales a los puertos
    contador2_introducir_contra: contbinario                     -- Referencia de componente
           port map(borrar,pulsador2_2,digito2);                 -- Asignación de señales a los puertos
    contador3_introducir_contra: contbinario                     -- Referencia de componente
           port map(borrar,pulsador3_2,digito3);                 -- Asignación de señales a los puertos
    display7seg1_introducir_contra: segmentos7_0a9               -- Referencia de componente
           port map(digito1,display1_introducir_contra);         -- Asignación de señales a los puertos
    display7seg2_introducir_contra: segmentos7_0a9               -- Referencia de componente
           port map(digito2,display2_introducir_contra);         -- Asignación de señales a los puertos
    display7seg3_introducir_contra: segmentos7_0a9               -- Referencia de componente
           port map(digito3,display3_introducir_contra);         -- Asignación de señales a los puertos
    pantalla: lcd                                                -- Referencia de componente
           port map('1',clock,char1,char2,char3,char4,char5,      -- Asignación de señales a los puertos
                    char6,char7,char8,char9,char10,char11,char12,
                    char13,char14,char15,char16,char17,char18,
                    char19,char20,char21,char22,char23,char24,
                    char25,char26,char27,char28,char29,char30,
                    char31,char32,lcd_rs,lcd_e,lcd_on,lcd_rw,datos);
process(clock)                                                   -- Cabecera de proceso
begin
    if clock'event and clock='1' then                            -- Sentencia if para detectar el flanco de subida de la señal de
                                                                 --reloj de 50 MHz
      if cont=49999999 then                                      -- Sentencia if para comprobar si ha pasado 1 segundo
           clk_0_5Hz<=not clk_0_5Hz;                             -- Si ha pasado un seguindo se cambia el valor de la señal del
           ...                                                   -- reloj de 0.5 Hz
```

```
        ...
        cont<=0;                                        -- y se reinicia el contador
     else
        cont<=cont+1;                                   -- Si aún no ha pasado el segundo se sigue aumentando el contador
     end if;
  end if;
  if clk_0_5Hz'event and clk_0_5Hz='1' then         -- Sentencia if para detectar el flanco de subida de la señal de reloj de 0,5 Hz
     est_actual<=est_siguiente;                     -- Si ha llegado el flanco de subida la señal est_actual toma el valor de
  else                                              -- la señal de est_siguiente
     est_actual<=est_actual;                        -- Si no ha llegado el flanco permanece igual
  end if;
end process;                                        -- Final de proceso
process(est_actual,ok,pulsador1,pulsador2,pulsador3,reset)   -- Cabecera de proceso
begin
  case est_actual is                                -- Sentencia case para ver cual es el estado actual
     when meter_contra_nueva=>          -- Cuando el estado actual es en el que hay que establecer la combinación de apertura:
        borrar<='1';                                -- Se habilita el contador para establecer la combinacion
        leds_rojos<="00000";                        -- Los leds rojos están apagados
        leds_verdes<="00000";                       -- Los leds verdes están apagados
        char1<=X"45";char2<=X"53";char3<=X"54";char4<=X"41";    -- Se muestra por la primera línea de la pantalla LCD:
        char5<=X"42";char6<=X"4C";char7<=X"45";char8<=X"43";    --"ESTABLECE"
        char9<=X"45";char10<=X"FE";
        char17<=X"43";char18<=X"4F";char19<=X"4E";char20<=X"54";-- Se muestra por la segunda línea de la pantalla LCD:
        char21<=X"52";char22<=X"41";char23<=X"53";char24<=X"45";-- "CONTRASENA NUEVA"
        char25<=X"4E";char26<=X"41";char27<=X"FE";char28<=X"4E";
        char29<=X"55";char30<=X"45";char31<=X"56";char32<=X"41";
        pulsador1_1<=pulsador1;                     -- Se pasa el valor de la señal de entrada del pulsador1 a la señal
                                                    -- que aumenta el valor del primer dígito de la combinación
        pulsador2_1<=pulsador2;                     -- Se pasa el valor de la señal de entrada del pulsador2 a la señal
                                                    -- que aumenta el valor del segundo dígito de la combinación
        pulsador3_1<=pulsador3;                     -- Se pasa el valor de la señal de entrada del pulsador3 a la señal
                                                    -- que aumenta el valor del tercer dígito de la combinación
        mostrardisplay1<=display1_meter_contra;     -- Se pasa el valor que se obtiene del codificador 7-segmentos del
                                                    -- primer dígito a la señal de salida que va al display1
        mostrardisplay2<=display2_meter_contra;     -- Se pasa el valor que se obtiene del codificador 7-segmentos del
                                                    -- segundo dígito a la señal de salida que va al display2
        mostrardisplay3<=display3_meter_contra;     -- Se pasa el valor que se obtiene del codificador 7-segmentos del
                                                    -- tercer dígito a la señal de salida que va al display3
        num1<=numero1;                      -- Se guarda el valor del primer dígito en una señal, para después poder
                                                    -- compararlo con el primer dígito de la contraseña que se introduzca
        num2<=numero2;                      -- Se guarda el valor del segundo dígito en una señal, para después poder
                                            -- compararlo con el segundo dígito de la contraseña que se introduzca
        num3<=numero3;                      -- Se guarda el valor del segundo dígito en una señal, para después poder
                                                    -- compararlo con el tercer dígito de la contraseña que se introduzca
        if ok='0' then                      -- Sentencia if para comprobar si el pulsador de "OK" está pulsado o no
          mostrardisplay1<="1000000";       -- Si "OK" está pulsado se borran los displays 7-segmentos con la combinación
          mostrardisplay2<="1000000";               -- de apertura establecida
          mostrardisplay3<="1000000";
          est_siguiente<=introducir_contra;         -- y el estado siguiente será el estado donde se introduzca la contraseña
        ...
```

```
    ...
    elsif ok='1' then                                   -- Si "OK" aún no se ha pulsado se permanece en el estado actual
      est_siguiente<=meter_contra_nueva;
    end if;
when introducir_contra=>                                -- Cuando el estado actual es en el que hay que introducir la contraseña:
    borrar<='1';                                        -- Se habilita el contador para introducir la contraseña
    leds_rojos<="00000";                                -- Los leds rojos están apagados
    leds_verdes<="00000";                               -- Los leds verdes están apagados
    char1<=X"49"; char2<=X"4E"; char3<=X"54"; char4<=X"52";  -- Se muestra por la primera línea de la pantalla LCD:
    char5<=X"4F"; char6<=X"44"; char7<=X"55"; char8<=X"43";  -- "INTRODUCE"
    char9<=X"45";char10<=X"FE";
    char17<=X"43"; char18<=X"4F"; char19<=X"4E"; char20<=X"54";-- Se muestra por la segunda línea de la pantalla LCD:
    char21<=X"52"; char22<=X"41"; char23<=X"53"; char24<=X"45";-- "CONTRASENA"
    char25<=X"4E";char26<=X"41";char27<=X"FE";char28<=X"FE";
    char29<=X"FE";char30<=X"FE";char31<=X"FE";char32<=X"FE";
    pulsador1_2<=pulsador1;                             -- Se pasa el valor de la señal de entrada del pulsador1 a la señal
                                                        -- que aumenta el valor del primer dígito de la contraseña
    pulsador2_2<=pulsador2;                             -- Se pasa el valor de la señal de entrada del pulsador2 a la señal
                                                        -- que aumenta el valor del segundo dígito de la contraseña
    pulsador3_2<=pulsador3;                             -- Se pasa el valor de la señal de entrada del pulsador3 a la señal
                                                        -- que aumenta el valor del tercer dígito de la contraseña
    mostrardisplay1<=display1_introducir_contra;        -- Se pasa el valor que se obtiene del codificador 7-segmentos del
                                                        -- primer dígito a la señal de salida que va al display1
    mostrardisplay2<=display2_introducir_contra;        -- Se pasa el valor que se obtiene del codificador 7-segmentos del
                                                        -- segundo dígito a la señal de salida que va al display2
    mostrardisplay3<=display3_introducir_contra;        -- Se pasa el valor que se obtiene del codificador 7-segmentos del
                                                        -- tercer dígito a la señal de salida que va al display3
    dig1<=digito1;                                  -- Se guarda el valor del primer dígito en una señal, para después poder
                                                    -- compararlo con el primer dígito de la combinación que se ha establecido
    dig2<=digito2;                                  -- Se guarda el valor del segundo dígito en una señal, para después poder
                                                    -- compararlo con el segundo dígito de la combinación que se ha establecido
    dig3<=digito3;                                  -- Se guarda el valor del tercer dígito en una señal, para después poder
                                                    -- compararlo con el tercer dígito de la combinación que se ha establecido
    if ok='0' then                                  -- Sentencia if para comprobar si el pulsador de "OK" está pulsado o no
      mostrardisplay1<="1000000";                   -- Si "OK" está pulsado se borran los displays 7-segmentos con
      mostrardisplay2<="1000000";                       -- la contraseña introducida
      mostrardisplay3<="1000000";
      if dig1=num1 and dig2=num2 and dig3=num3 then-- Sentencia if para comparar los dígitos
        est_siguiente<=contra_bien;                      -- de la combinación de apertura y la contraseña introducida
      else                                          -- si coinciden el estado siguiente será contraseña correcta
        est_siguiente<=contra_mal;                  -- si no coinciden el estado siguiente será contraseña incorrecta
      end if;
    elsif ok='1' then                               -- Si "OK" aún no se ha pulsado se permanece en el estado actual
      est_siguiente<=introducir_contra;
    end if;
when contra_bien=>                                  -- Cuando el estado actual es contraseña correcta:
    borrar<='0';                                    -- Se inhabilitan los contadores
    leds_verdes<="11111";                           -- Se encienden los leds verdes
    dig1<="0000";                                   -- Se reinician las señales en las que se han guardado los dígitos
    dig2<="0000";                                    -- de la contraseña introducida
    dig3<="0000";
    ...
```

```
        ...
        char1<=X"43"; char2<=X"4F"; char3<=X"4E"; char4<=X"54";   -- Se muestra por la primera línea de la pantalla LCD:
        char5<=X"52"; char6<=X"41"; char7<=X"53"; char8<=X"45";   -- "CONTRASENA"
        char9<=X"4E";char10<=X"41";
        char17<=X"43"; char18<=X"4F"; char19<=X"52";char20<=X"52";-- Se muestra por la segunda línea de la pantalla LCD:
        char21<=X"45";char22<=X"43";char23<=X"54"; char24<=X"41";-- "CORRECTA"
        char25<=X"FE";char26<=X"FE";char27<=X"FE";char28<=X"FE";
        char29<=X"FE";char30<=X"FE";char31<=X"FE";char32<=X"FE";
        if ok='0' then                                  -- Sentencia if para comprobar si el pulsador de "OK" está pulsado o no
          est_siguiente<=introducir_contra;-- Si "OK" está pulsado el estado siguiente será donde se introduzca la contraseña
        elsif reset='1' then                                    -- Si "OK" no está pulsado y el interruptor de "reset" está a '1'
          est_siguiente<=meter_contra_nueva;     -- el estado siguiente será donde se establezca la combinación de apertura
        else                                                    -- Si "OK" no está pulsado y "reset" es '0'
          est_siguiente<=contra_bien;                           -- se permanece en el estado actual
        end if;
    when contra_mal=>                                           -- Cuando el estado actual es contraseña incorrecta:
        borrar<='0';                                            -- Se inhabilitan los contadores
        leds_rojos<="11111";                                    -- Se encienden los leds rojos
        dig1<="0000";                                           -- Se reinician las señales en las que se han guardado los dígitos
        dig2<="0000";                                           -- de la contraseña introducida
        dig3<="0000";
        char1<=X"43"; char2<=X"4F"; char3<=X"4E";char4<=X"54";    -- Se muestra por la primera línea de la pantalla LCD:
        char5<=X"52";char6<=X"41";char7<=X"53";char8<=X"45";      -- "CONTRASENA"
        char9<=X"4E";char10<=X"41";
        char17<=X"49"; char18<=X"4E"; char19<=X"43"; char20<=X"4F";-- Se muestra por la segunda línea de la pantalla LCD:
        char21<=X"52";char22<=X"52";char23<=X"45";char24<=X"43";-- "INCORRECTA"
        char25<=X"54";char26<=X"41";char27<=X"FE";char28<=X"FE";
        char29<=X"FE";char30<=X"FE";char31<=X"FE";char32<=X"FE";
        if ok='0' then                                  -- Sentencia if para comprobar si el pulsador de "OK" está pulsado o no
          est_siguiente<=introducir_contra;-- Si "OK" está pulsado el estado siguiente será donde se introduzca la contraseña
        elsif ok='1' then                               -- Si "OK" aún no se ha pulsado se permanece en el estado actual
          est_siguiente<=contra_mal;
        end if;
    end case;
end process;                                                    -- Final de proceso
end ejemplo;                                                    -- Final del programa
```

Capítulo 6

Conclusiones y líneas futuras

6.1. Conclusiones

La realización de este trabajo ha servido para desarrollar un entorno educativo para dispositivos de lógica programable, en el que el principal objetivo es aprender un lenguaje de descripción de hardware.

La comparativa entre los lenguajes HDL más extendidos y el estudio comparativo entre universidades han resultado imprescindibles para decidir entre las diferentes opciones que se encuentran dentro de los lenguajes de descripción de hardware.
21 universidades nacionales y 16 universidades estadounidenses han sido objeto del estudio comparativo para aportar información en la decisión de que lenguaje HDL se debía escoger. De este modo, con toda la información obtenida, ha resultado mucho más sencillo elegir la opción idónea para un entorno como el que se ha desarrollado a lo largo de este trabajo.

En cuanto a los guiones de prácticas realizados, se ha seguido el temario típico de un curso de electrónica digital, de forma que todos los conocimientos adquiridos en ese curso se enfoquen ahora hacia un lenguaje de descripción de hardware y a la programación de dispositivos de lógica programable.

El contenido de de los guiones se puede adaptar a otro tipo de asignaturas, en donde sólo se vea parte del temario desarrollado.

El software Quartus II y la tarjeta educacional DE2 de Altera son dos herramientas muy potente, con las que se pueden desarrollar un serie de aplicaciones muy interesantes relacionadas con el entorno educativo que se ha desarrollado.

Una vez finalizado el trabajo, desde el punto de vista del autor, el lenguaje VHDL ha sido una elección correcta. A pesar de que es un lenguaje que requiere un esfuerzo considerable para llegar a aprenderlo y dominarlo, resulta un lenguaje intuitivo y no muy complicado a la hora de leerlo e interpretarlo, por lo que encaja perfectamente en el entorno desarrollado.

6.2. Líneas futuras

De cara al futuro, los guiones de prácticas realizados se pueden adaptar a tecnologías más avanzadas. En este caso se ha trabajado en torno al hardware y al software del que se disponía en el laboratorio, pero en un futuro, realizando estudios de mercado cada cierto tiempo, si se cree oportuno se puede modernizar el material de laboratorio e ir adaptando el entorno educativo que se ha desarrollado en este trabajo

Respecto a los guiones de prácticas, se ha seguido un mismo formato para todos ellos, cada guión contiene un apartado donde se detallan los objetivos de la práctica, tiene una primera parte donde se introduce el bloque teórico con algunos ejemplos y, por último, están los ejercicios de diseño, que es donde se le exigirá al alumno que utilice los conocimientos adquiridos. Se ha seguido este formato en concreto, pero los guiones quedan abiertos a posibles modificaciones y mejoras que puedan aportar tanto el profesorado como los alumnos.

El contenido de los guiones es una introducción al lenguaje VHDL y a la programación de dispositivos programables, por lo que es posible profundizar mucho más en ello. Tanto el lenguaje VHDL, como la tarjeta DE2 y el Quartus II son lo suficientemente potentes como para llevar a cabo diseños de sistemas más complejos que los que se han implementado en las prácticas.

Por otra parte, además de profundizar en VHDL, resultaría muy interesante añadir unas prácticas de introducción a otro lenguaje de descripción de hardware, concretamente Verilog HDL. Al igual que VHDL, Verilog es un lenguaje de descripción de hardware estandarizado y muy potente, además, una vez aprendido un lenguaje como VHDL, no resultaría tan complicado como empezar sin tener ningún conocimiento previo sobre lenguajes HDL.

Capítulo 7

Presupuesto

El presupuesto se dividirá en el coste de los materiales, el valor de la mano de obra y la cuantía de las obligaciones sociales. El cálculo final se realizará mediante la suma de los tres apartados.

1. COSTE DE MATERIALES

CONCEPTO	CANTIDAD	PRECIO UNITARIO	TOTAL
Tarjeta Educacional DE2 (Altera)	1	198.43 € (269 $)	**199 €** (269 $)

2. COSTE DE LA MANO DE OBRA/SALARIO

CONCEPTO	Nº HORAS	SUELDO/HORA	TOTAL
Graduado en Ingeniería en Tecnologías Industriales	180	20 €	**3.600 €**

3. OBLIGACIONES SOCIALES

CONCEPTO	PORCENTAJE	VALOR
Contingencias Generales	28,30 %	1.019 €
Desempleo	8,30 %	299 €
Fondo de Garantía Salarial	0,20 %	7€
Formación Profesional	0,70 %	25€
Base de Accidente de Trabajo	1,65 %	59€
TOTAL	**39,15 %**	**1.409 €**

- **Cálculo total del presupuesto**

1. COSTE DE MATERIALES…….. 199 €
2. COSTE DE LA MANO DE OBRA/SALARIO……. 3.600 €
3. OBLIGACIONES SOCIALES……. 1.409 €

TOTAL……. **5.208 €** *

El total del presupuesto asciende a la cantidad de *Cinco mil doscientos ocho Euros* (I.V.A. no incluido).

*Este valor no incluye el I.V.A.

Capítulo 8

Bibliografía

Referencias bibliográficas

[1] Síntesis y descripción de circuitos digitales utilizando VHD. Francisco Javier Torres Valle. Universidad Autónoma de Guadalajara.

[2] Apuntes de la asignatura "Taller de diseño de lógica programable". Universidad Nacional del Centro de la Provincia de Buenos Aires. Guillermo Jaquenod.

[3] Apuntes de la asignatura "Laboratorio de estructura de ordenadores". Universidad Complutense de Madrid. José Jaime Ruz Ortiz.

[4] VHDL & Verilog compared and contrasted plus modeled example written in VHDL, Verilog and C. Douglas J. Smith. VeriBest Incorporated (1996).

[5] The Designer's Guide to VHDL. Peter J. Ashenden. Morgan Kaufmann Publishers (Third Edition 2008).

[6] DE2 User Manual. Guía de usuario de la tarjeta educacional DE2 de Altera.

[7] Diseño de sistemas digitales con VHDL. Felipe Machado Sánchez, Susana Borromeo López y Cristina Rodríguez Sánchez. Departamento de Tecnología Electrónica de la Universidad Rey Juan Carlos (2011).

[8] Introducción al lenguaje VHDL. Miguel Ángel Freire Rubio. Departamento de sistemas electrónicos y de control de la Universidad Politécnica de Madrid.

[9] VHDL, lenguaje para descripción y modelado de circuitos. Fernando Pardo Carpio. Ingeniería Informática de la Universitat de Valencia (1997).

[10] Proyecto final de carrera: "Entorno educativo para el aprendizaje de sistemas digitales". Juan María Pérez Azpeitia.

[11] Quartus II Introduction. Guíade introducción a Quartus II de Altera.

[12] Datasheet del módulo LCD GDM1602A de la tarjeta DE2.

[13] Getting started with Altera's DE2 board.

Referencias en la red

- Altera Corportation
  www.altera.com

- Terasic Technologies
  www.terasic.com

Anexos

Anexo 1: Distribución de pines de la tarjeta educacional DE2

Toda la información de este anexo esta extraída del manual de usuario de la tarjeta educacional DE2.

A continuación se mostrará la distribución de los pines de la tarjeta educacional DE2. Aquí sólo se mostrarán los pines correspondientes a los elementos que se han utilizado a la hora de realizar los diseños de los guiones de prácticas, la asignación de pines del resto de elementos se puede encontrar en el manual de usuario d la tarjeta DE2.

- **Interruptores:**

Signal Name	FPGA Pin No.	Description
SW[0]	PIN_N25	Toggle Switch[0]
SW[1]	PIN_N26	Toggle Switch[1]
SW[2]	PIN_P25	Toggle Switch[2]
SW[3]	PIN_AE14	Toggle Switch[3]
SW[4]	PIN_AF14	Toggle Switch[4]
SW[5]	PIN_AD13	Toggle Switch[5]
SW[6]	PIN_AC13	Toggle Switch[6]
SW[7]	PIN_C13	Toggle Switch[7]
SW[8]	PIN_B13	Toggle Switch[8]
SW[9]	PIN_A13	Toggle Switch[9]
SW[10]	PIN_N1	Toggle Switch[10]
SW[11]	PIN_P1	Toggle Switch[11]
SW[12]	PIN_P2	Toggle Switch[12]
SW[13]	PIN_T7	Toggle Switch[13]
SW[14]	PIN_U3	Toggle Switch[14]
SW[15]	PIN_U4	Toggle Switch[15]
SW[16]	PIN_V1	Toggle Switch[16]
SW[17]	PIN_V2	Toggle Switch[17]

Tabla 1: Asignación de pines de los interruptores

- **Pulsadores:**

Signal Name	FPGA Pin No.	Description
KEY[0]	PIN_G16	Pushbutton[0]
KEY[1]	PIN_N23	Pushbutton[1]
KEY[2]	PIN_P23	Pushbutton[2]
KEY[3]	PIN_W26	Pushbutton[3]

Tabla 2: Asignación de pines de los pulsadores

- **LEDs:**

La placa DE2 dispone de de 27 LEDs, 18 son de color rojo y 9 son de color verde.

Signal Name	FPGA Pin No.	Description
LEDR[0]	PIN_AE23	LED Red[0]
LEDR[1]	PIN_AF23	LED Red[1]
LEDR[2]	PIN_AB21	LED Red[2]
LEDR[3]	PIN_AC22	LED Red[3]
LEDR[4]	PIN_AD22	LED Red[4]
LEDR[5]	PIN_AD23	LED Red[5]
LEDR[6]	PIN_AD21	LED Red[6]
LEDR[7]	PIN_AC21	LED Red[7]
LEDR[8]	PIN_AA14	LED Red[8]
LEDR[9]	PIN_Y13	LED Red[9]
LEDR[10]	PIN_AA13	LED Red[10]
LEDR[11]	PIN_AC14	LED Red[11]
LEDR[12]	PIN_AD15	LED Red[12]
LEDR[13]	PIN_AE15	LED Red[13]
LEDR[14]	PIN_AF13	LED Red[14]
LEDR[15]	PIN_AE13	LED Red[14]
LEDR[16]	PIN_AE12	LED Red[15]
LEDR[17]	PIN_AD12	LED Red[17]
LEDG[0]	PIN_AE22	LED Green[0]
LEDG[1]	PIN_AF22	LED Green[1]
LEDG[2]	PIN_W19	LED Green[2]
LEDG[3]	PIN_V18	LED Green[3]
LEDG[4]	PIN_U18	LED Green[4]
LEDG[5]	PIN_U17	LED Green[5]
LEDG[6]	PIN_AA20	LED Green[6]
LEDG[7]	PIN_Y18	LED Green[7]
LEDG[8]	PIN_Y12	LED Green[8]

Tabla 3: Asignación de pines de los LEDs

- **Displays 7-segmentos:**

En la tarjeta se pueden encontrar 8 displays 7-segmentos. A continuación se muestra la posición y la numeración de cada uno de los segmentos que forman el display 7-segmentos.

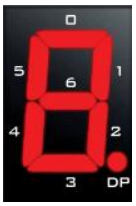

Figura 24: Display 7 segmentos

Signal Name	FPGA Pin No.	Description
HEX0[0]	PIN_AF10	Seven Segment Digit 0[0]
HEX0[1]	PIN_AB12	Seven Segment Digit 0[1]
HEX0[2]	PIN_AC12	Seven Segment Digit 0[2]
HEX0[3]	PIN_AD11	Seven Segment Digit 0[3]
HEX0[4]	PIN_AE11	Seven Segment Digit 0[4]
HEX0[5]	PIN_V14	Seven Segment Digit 0[5]
HEX0[6]	PIN_V13	Seven Segment Digit 0[6]
HEX1[0]	PIN_V20	Seven Segment Digit 1[0]
HEX1[1]	PIN_V21	Seven Segment Digit 1[1]
HEX1[2]	PIN_W21	Seven Segment Digit 1[2]
HEX1[3]	PIN_Y22	Seven Segment Digit 1[3]
HEX1[4]	PIN_AA24	Seven Segment Digit 1[4]
HEX1[5]	PIN_AA23	Seven Segment Digit 1[5]
HEX1[6]	PIN_AB24	Seven Segment Digit 1[6]
HEX2[0]	PIN_AB23	Seven Segment Digit 2[0]
HEX2[1]	PIN_V22	Seven Segment Digit 2[1]
HEX2[2]	PIN_AC25	Seven Segment Digit 2[2]
HEX2[3]	PIN_AC26	Seven Segment Digit 2[3]
HEX2[4]	PIN_AB26	Seven Segment Digit 2[4]
HEX2[5]	PIN_AB25	Seven Segment Digit 2[5]
HEX2[6]	PIN_Y24	Seven Segment Digit 2[6]
HEX3[0]	PIN_Y23	Seven Segment Digit 3[0]
HEX3[1]	PIN_AA25	Seven Segment Digit 3[1]
HEX3[2]	PIN_AA26	Seven Segment Digit 3[2]
HEX3[3]	PIN_Y26	Seven Segment Digit 3[3]
HEX3[4]	PIN_Y25	Seven Segment Digit 3[4]
HEX3[5]	PIN_U22	Seven Segment Digit 3[5]
HEX3[6]	PIN_W24	Seven Segment Digit 3[6]
HEX4[0]	PIN_U9	Seven Segment Digit 4[0]
HEX4[1]	PIN_U1	Seven Segment Digit 4[1]
HEX4[2]	PIN_U2	Seven Segment Digit 4[2]
HEX4[3]	PIN_T4	Seven Segment Digit 4[3]
HEX4[4]	PIN_R7	Seven Segment Digit 4[4]
HEX4[5]	PIN_R6	Seven Segment Digit 4[5]
HEX4[6]	PIN_T3	Seven Segment Digit 4[6]
HEX5[0]	PIN_T2	Seven Segment Digit 5[0]
HEX5[1]	PIN_P6	Seven Segment Digit 5[1]
HEX5[2]	PIN_P7	Seven Segment Digit 5[2]
HEX5[3]	PIN_T9	Seven Segment Digit 5[3]
HEX5[4]	PIN_R5	Seven Segment Digit 5[4]
HEX5[5]	PIN_R4	Seven Segment Digit 5[5]
HEX5[6]	PIN_R3	Seven Segment Digit 5[6]
HEX6[0]	PIN_R2	Seven Segment Digit 6[0]
HEX6[1]	PIN_P4	Seven Segment Digit 6[1]
HEX6[2]	PIN_P3	Seven Segment Digit 6[2]

HEX6[3]	PIN_M2	Seven Segment Digit 6[3]
HEX6[4]	PIN_M3	Seven Segment Digit 6[4]
HEX6[5]	PIN_M5	Seven Segment Digit 6[5]
HEX6[6]	PIN_M4	Seven Segment Digit 6[6]
HEX7[0]	PIN_L3	Seven Segment Digit 7[0]
HEX7[1]	PIN_L2	Seven Segment Digit 7[1]
HEX7[2]	PIN_L9	Seven Segment Digit 7[2]
HEX7[3]	PIN_L6	Seven Segment Digit 7[3]
HEX7[4]	PIN_L7	Seven Segment Digit 7[4]
HEX7[5]	PIN_P9	Seven Segment Digit 7[5]
HEX7[6]	PIN_N9	Seven Segment Digit 7[6]

Tabla 4: Asignación de pines de los displays 7-segmentos

- **Entradas de reloj:**

La tarjeta DE2 permite trabajar con un reloj interno de 27 MHz, con un reloj interno de 50 MHz y con un reloj externo.

Signal Name	FPGA Pin No.	Description
CLOCK_27	PIN_D13	27 MHz clock input
CLOCK_50	PIN_N2	50 MHz clock input
EXT_CLOCK	PIN_P26	External (SMA) clock input

Tabla 5: Asignación de pines de las entradas de reloj

- **Pantalla LCD:**

Signal Name	FPGA Pin No.	Description
LCD_DATA[0]	PIN_J1	LCD Data[0]
LCD_DATA[1]	PIN_J2	LCD Data[1]
LCD_DATA[2]	PIN_H1	LCD Data[2]
LCD_DATA[3]	PIN_H2	LCD Data[3]
LCD_DATA[4]	PIN_J4	LCD Data[4]
LCD_DATA[5]	PIN_J3	LCD Data[5]
LCD_DATA[6]	PIN_H4	LCD Data[6]
LCD_DATA[7]	PIN_H3	LCD Data[7]
LCD_RW	PIN_K4	LCD Read/Write Select, 0 = Write, 1 = Read
LCD_EN	PIN_K3	LCD Enable
LCD_RS	PIN_K1	LCD Command/Data Select, 0 = Command, 1 = Data
LCD_ON	PIN_L4	LCD Power ON/OFF
LCD_BLON	PIN_K2	LCD Back Light ON/OFF

Tabla 6: Asignación de pines de la pantalla LCD

Anexo 2: Código del archivo *lcd.vhd*

```
library IEEE;                                                    -- Librería estándar de la IEEE
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;                                    -- Paquetes de la librería estándar de la IEEE

ENTITY lcd IS
   PORT(reset,clk_50Mhz: IN STD_LOGIC;                           -- Señal de reset y de reloj
          char1,char2,char3,char4,char5,char6,char7,char8,        -- Vectores de entrada para cada caracter
          char9,char10,char11,char12,char13,char14,char15,
          char16,char17,char18,char19,char20,char21,char22,
          char23,char24,char25,char26,char27,char28,char29,
          char30,char31,char32: in STD_LOGIC_VECTOR (7 DOWNTO 0);
          LCD_RS, LCD_E, LCD_ON: OUT STD_LOGIC;                  -- Bit RS, Enable y ON
          LCD_RW: BUFFER STD_LOGIC;                              -- Bit R/W
          DATA_BUS: INOUT STD_LOGIC_VECTOR(7 DOWNTO 0));         -- Vector de salida para configurar la pantalla y mostrar el carácter
END lcd;
ARCHITECTURE config_lcd OF lcd IS                                -- Cabecera del programa
   TYPE STATE_TYPE IS ( HOLD,FUNC_SET,DISPLAY_ON,                -- Delcaración de una variable creada por el usuario
                        MODE_SET,WRITE_CHAR1,WRITE_CHAR2,WRITE_CHAR3,
                        WRITE_CHAR4,WRITE_CHAR5,WRITE_CHAR6,WRITE_CHAR7,
                        WRITE_CHAR8, WRITE_CHAR9, WRITE_CHAR10, WRITE_CHAR11,
                        WRITE_CHAR12,WRITE_CHAR13,WRITE_CHAR14,WRITE_CHAR15,
                        WRITE_CHAR16,WRITE_CHAR17,WRITE_CHAR18,WRITE_CHAR19,
                        WRITE_CHAR20,WRITE_CHAR21,WRITE_CHAR22,WRITE_CHAR23,
                        WRITE_CHAR24,WRITE_CHAR25,WRITE_CHAR26,WRITE_CHAR27,
                        WRITE_CHAR28,WRITE_CHAR29,WRITE_CHAR30,WRITE_CHAR31,
                        WRITE_CHAR32,LINE_2,RETURN_HOME, TOGGLE_E, RESET1,
                        RESET2,RESET3,DISPLAY_OFF,DISPLAY_CLEAR);
   SIGNAL state, next_command: STATE_TYPE;                       -- Declaración de señales
   SIGNAL DATA_BUS_VALUE: STD_LOGIC_VECTOR(7 DOWNTO 0);
   SIGNAL CLK_COUNT_400HZ: STD_LOGIC_VECTOR(19 DOWNTO 0);
   SIGNAL CLK_400HZ: STD_LOGIC;
BEGIN
   LCD_ON <= '1';                                                -- Bit On a '1'
   DATA_BUS <= DATA_BUS_VALUE WHEN LCD_RW = '0' ELSE "00000000";     -- Traspaso de los valores de la señal DATA_BUS_VALUE
                                                                     -- a la salida DATA_BUS
   PROCESS(clk_50Mhz)                                            -- Cabecera de proceso
   BEGIN
     IF RESET = '0' THEN                                         -- Sentencia if para comprobar si hay que resetear el contador o no
          CLK_COUNT_400HZ <= X"00000";
          CLK_400HZ <= '0';
     ELSIF CLK_50MHZ'EVENT AND CLK_50MHZ = '1'then
          IF CLK_COUNT_400HZ < X"0F424" THEN                     -- Sentencia if para pasar de una frecuencia de 50 MHz a una de 400 Hz
             CLK_COUNT_400HZ <= CLK_COUNT_400HZ + 1;
          ELSE
            CLK_COUNT_400HZ <= X"00000";
            CLK_400HZ <= NOT CLK_400HZ;
          END IF;
     END IF;
          ...
```

```
        ...
    END PROCESS;                                    -- Final de proceso
    PROCESS (CLK_50Mhz, reset)                      -- Cabecera de proceso
    BEGIN
      IF reset = '0' THEN                           -- Sentencia if para comprobar si hay que resetear la LCD o no
          state <= RESET1;
          DATA_BUS_VALUE <= X"38";
          next_command <= RESET2;
          LCD_E <= '1';
          LCD_RS <= '0';
          LCD_RW <= '0';
      ELSIF clk_400Hz'EVENT AND clk_400Hz = '1' THEN -- Sentencia para detectar el flanco de subida de la señal de 400 Hz
-- Configurarla para trasnmitir 8 bits y para utilizar las dos líneas del display con una resolución de 5x8
          CASE state IS                             -- Sentencia case para ejecutar las sentencias de los diferentes estados
             WHEN RESET1 =>                         -- Estado RESET1 --> se inicializa o resetea la LCD
                    LCD_E <= '1';
                    LCD_RS <= '0';
                    LCD_RW <= '0';
                    DATA_BUS_VALUE <= X"38";
                    state <= TOGGLE_E;
                    next_command <= RESET2;
             WHEN RESET2 =>                         -- Estado RESET2 --> se inicializa o resetea de nuevo la LCD
                    LCD_E <= '1';
                    LCD_RS <= '0';
                    LCD_RW <= '0';
                    DATA_BUS_VALUE <= X"38";
                    state <= TOGGLE_E;
                    next_command <= RESET3;
             WHEN RESET3 =>                         -- Estado RESET3 --> se inicializa o resetea de nuevo la LCD
                    LCD_E <= '1';
                    LCD_RS <= '0';
                    LCD_RW <= '0';
                    DATA_BUS_VALUE <= X"38";
                    state <= TOGGLE_E;
                    next_command <= FUNC_SET;
-- En la hoja de características de la pantalla LCD aparece que hay que mandar DATA_BUS_VALUE <= X"38" 3 veces para
-- inicializar correctamente la LCD
             WHEN FUNC_SET =>
-- Se especifica el número de líneas de la LCD (2) y la resolución (5x8)
                    LCD_E <= '1';
                    LCD_RS <= '0';
                    LCD_RW <= '0';
                    DATA_BUS_VALUE <= X"38";
                    state <= TOGGLE_E;
                    next_command <= DISPLAY_OFF;
-- Turn off Display and Turn off cursor
             WHEN DISPLAY_OFF =>
                    LCD_E <= '1';
                    LCD_RS <= '0';
                    LCD_RW <= '0';
                    DATA_BUS_VALUE <= X"08";
                    state <= TOGGLE_E;
                    next_command <= DISPLAY_CLEAR;
                    ...
```

```
                ...
-- Turn on Display and Turn off cursor
        WHEN DISPLAY_CLEAR =>
                LCD_E <= '1';
                LCD_RS <= '0';
                LCD_RW <= '0';
                DATA_BUS_VALUE <= X"01";
                state <= TOGGLE_E;
                next_command <= DISPLAY_ON;
-- Turn on Display and Turn off cursor
        WHEN DISPLAY_ON =>
                LCD_E <= '1';
                LCD_RS <= '0';
                LCD_RW <= '0';
                DATA_BUS_VALUE <= X"0C";
                state <= TOGGLE_E;
                next_command <= MODE_SET;
-- Configurar el Write Mode para incremente automáticamente la dirección después de escribir un caracter
-- y que dicho incremento sea de una posición a la derecha
        WHEN MODE_SET =>
                LCD_E <= '1';
                LCD_RS <= '0';
                LCD_RW <= '0';
                DATA_BUS_VALUE <= X"06";
                state <= TOGGLE_E;
                next_command <= WRITE_CHAR1;
-- Escribir un caracter ASCII en la posición 1 de la LCD
        WHEN WRITE_CHAR1 =>
                LCD_E <= '1';
                LCD_RS <= '1';
                LCD_RW <= '0';
                DATA_BUS_VALUE <= char1;
                state <= TOGGLE_E;
                next_command <= WRITE_CHAR2;
-- Escribir un caracter ASCII en la posición 2 de la LCD
        WHEN WRITE_CHAR2 =>
                LCD_E <= '1';
                LCD_RS <= '1';
                LCD_RW <= '0';
                DATA_BUS_VALUE <= char2;
                state <= TOGGLE_E;
                next_command <= WRITE_CHAR3;
-- Escribir un caracter ASCII en la posición 3 de la LCD
        WHEN WRITE_CHAR3 =>
                LCD_E <= '1';
                LCD_RS <= '1';
                LCD_RW <= '0';
                DATA_BUS_VALUE <= char3;
                state <= TOGGLE_E;
                next_command <= WRITE_CHAR4;
                ...
```

```
        ...
WHEN WRITE_CHAR4 =>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char4;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR5;
WHEN WRITE_CHAR5 =>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char5;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR6;
WHEN WRITE_CHAR6 =>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char6 ;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR7;
WHEN WRITE_CHAR7 =>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char7;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR8;
WHEN WRITE_CHAR8 =>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char8;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR9;
WHEN WRITE_CHAR9 =>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char9;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR10;
WHEN WRITE_CHAR10 =>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char10;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR11;
        ...
```

```
                ...
        WHEN WRITE_CHAR11 =>
                LCD_E <= '1';
                LCD_RS <= '1';
                LCD_RW <= '0';
                DATA_BUS_VALUE <= char11;
                state <= TOGGLE_E;
                next_command <= WRITE_CHAR12;
        WHEN WRITE_CHAR12 =>
                LCD_E <= '1';
                LCD_RS <= '1';
                LCD_RW <= '0';
                DATA_BUS_VALUE <= char12;
                state <= TOGGLE_E;
                next_command <= WRITE_CHAR13;
        WHEN WRITE_CHAR13 =>
                LCD_E <= '1';
                LCD_RS <= '1';
                LCD_RW <= '0';
                DATA_BUS_VALUE <= char13;
                state <= TOGGLE_E;
                next_command <= WRITE_CHAR14;
        WHEN WRITE_CHAR14 =>
                LCD_E <= '1';
                LCD_RS <= '1';
                LCD_RW <= '0';
                DATA_BUS_VALUE <= char14;
                state <= TOGGLE_E;
                next_command <= WRITE_CHAR15;
        WHEN WRITE_CHAR15 =>
                LCD_E <= '1';
                LCD_RS <= '1';
                LCD_RW <= '0';
                DATA_BUS_VALUE <= char15;
                state <= TOGGLE_E;
                next_command <= WRITE_CHAR16;
        WHEN WRITE_CHAR16 =>
                LCD_E <= '1';
                LCD_RS <= '1';
                LCD_RW <= '0';
                DATA_BUS_VALUE <= char16;
                state <= TOGGLE_E;
                next_command <= LINE_2;
-- Se cambia de la línea 1 a la línea 2 de la LCD
        WHEN LINE_2=>
                LCD_E <= '1';
                LCD_RS <= '0';
                LCD_RW <= '0';
                DATA_BUS_VALUE <= X"C0";
                state <= TOGGLE_E;
                next_command <= WRITE_CHAR17;
                ...
```

```
        ...
WHEN WRITE_CHAR17 =>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char17;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR18;
WHEN WRITE_CHAR18 =>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char18;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR19;
WHEN WRITE_CHAR19=>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char19;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR20;
WHEN WRITE_CHAR20=>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char20;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR21;
WHEN WRITE_CHAR21=>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char21;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR22;
WHEN WRITE_CHAR22=>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char22;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR23;
WHEN WRITE_CHAR23=>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char23;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR24;
        ...
```

```
        ...
WHEN WRITE_CHAR24=>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char24;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR25;
WHEN WRITE_CHAR25=>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char25;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR26;
WHEN WRITE_CHAR26=>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char26;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR27;
WHEN WRITE_CHAR27 =>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char27;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR28;
WHEN WRITE_CHAR28 =>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char28;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR29;
WHEN WRITE_CHAR29 =>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char29;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR30;
WHEN WRITE_CHAR30=>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= char30;
        state <= TOGGLE_E;
        next_command <= WRITE_CHAR31;
        ...
```

```
                ...
            WHEN WRITE_CHAR31=>
                LCD_E <= '1';
                LCD_RS <= '1';
                LCD_RW <= '0';
                DATA_BUS_VALUE <= char31;
                state <= TOGGLE_E;
                next_command <= WRITE_CHAR32;
            WHEN WRITE_CHAR32 =>
                LCD_E <= '1';
                LCD_RS <= '1';
                LCD_RW <= '0';
                DATA_BUS_VALUE <= char32;
                state <= TOGGLE_E;
                next_command <= RETURN_HOME;
-- Se vuelve a la posición 1 de la línea 1 de la LCD
            WHEN RETURN_HOME =>
                LCD_E <= '1';
                LCD_RS <= '0';
                LCD_RW <= '0';
                DATA_BUS_VALUE <= X"80";
                state <= TOGGLE_E;
                next_command <= WRITE_CHAR1;
-- Los siguientes dos estados tienen lugar al final de cada uno de los demás estados
-- SE utilizan para dar tiempo a que la LCD ejecute las oredenes que se le da mediante los otros estados
            WHEN TOGGLE_E =>                    -- Estado TOGGLE_E --> Conmutación del Enable
                LCD_E <= '0';
                state <= HOLD;
            WHEN HOLD =>                        -- Estado HOLD --> Retención del estado durante un periodo
                state <= next_command;
        END CASE;
    END IF;
  END PROCESS;                                  -- Final de proceso
END config_lcd;                                 -- Final del programa
```

Anexo 3: Glosario

- **ABEL HDL:** Acrónimo ingles de **A**dvanced **B**oolean **E**xpression **L**anguage. Se trata de un lenguaje de descripción de hardware para programar dispositivos de lógica programable.

- **ADA:** Es un lenguaje de programación multipropósito orientado a objetos.

- **AHDL:** Acrónimo inglés de **A**ltera **H**ardware **D**escription **L**anguage. Es un lenguaje de descripción de hardware creado por Altera.

- **Arquitectura (Architecture):** Es la parte del código de VHDL en donde se realiza la descripción detallada de la estructura interna del sistema o de su comportamiento.

- **ASIC:** Acrónimo inglés de **A**pplication-**S**pecific **I**ntegrated **C**ircuit, circuito integrado para aplicaciones específicas. Es un circuito integrado fabricado para un uso concreto y no para el uso general.

- **CAD (computer-aided design):** Acrónimo en inglés de **C**omputer-**A**ided **D**esign, en castellano se conoce como diseño asistido por computadora, se trata de una serie de herramientas computacionales que se basan básicamente en programas de dibujo 2D y modelado 3D. Son muy utilizadas en ámbitos de ingeniería, arquitectura y diseño.

- **Componente (Component):** Se trata de una entidad que ha sido declarada en otro diseño que permite realizar diseños jerárquicos.

- **Entidad (Entity):** Parte del código de VHDL en donde se definen las entradas y as salidas que tiene el diseño.

- **FLASH:** Es el tipo de memorias que se utilizan en las memorias USB, ya que permiten la lectura y escritura en varias posiciones en una misma operación.

- **FPGA:** Acrónimo inglés de **F**ield **P**rogrammable **G**ate **A**rray. Es un dispositivo semiconductor que contiene bloques de lógica que puede ser programado mediante lenguajes de descripción de hardware.

- **Programación JTAG:** Acrónimo inglés de **J**oint **T**est **A**ction **G**roup. Mediante esta programación los datos se envían directamente al dispositivo de lógica programable y este guarda esa configuración mientras no se interrumpa la alimentación.

- **Librería (Library):** Lugar donde se guarda la información relacionada con un diseño determinado y que puede proporcionar servicios a programas totalmente independientes a ella.

- **Niveles de abstracción:** Se refieren al grado de detalle en que un sistema es modelado, entiendo como un nivel alto una estructura global del sistema y como un nivel bajo una visión mucho más detallada.

- **PLD:** Acróonimo inglés de **P**rogrammable **L**ogic **D**evice. Se trata de un dispositivo de lógica programable formado por múltiples bloques lógicos.

- **Proceso (Process):** Sentencia que ejecuta de forma secuencial todas las instrucciones que lo forman, de este modo se puede alternar programación concurrente y secuencial en un mismo diseño.

- **PS/2:** Conector o puerto utilizados para conectar teclados y ratones.

- **RISC:** Acrónimo ingles de **R**educed **I**nstruction **S**et **C**omputer, en castellano se conoce como computador de instrucciones reducidas. Es un tipo de arquitectura computacional que se utiliza generalmente en el diseño de microprocesadores y microcontroladores.

- **RS-232:** Estándar que recoge una norma para el intercambio de una serie de datos binarios entre dos equipos.

- **RTL:** Acrónimo inglés de **R**egistered **T**ransfer **L**anguage. Es un lenguaje de programación de bajo nivel muy parecido al lenguaje ensamblador.

- **SDRAM:** Acrónimo inglés de **S**ynchronous **D**ynamic **R**andom-**A**ccess **M**emory. Se conoce por este nombre a una familia de memorias dinámicas de acceso aleatorio, que tienen una interfaz síncrona.

- **SRAM:** Acrónimo inglés de **S**tatic **R**andom **A**cces **M**emory. Es un tipo de memoria volátil con las mismas características que las SDRAM, pero no poseen una interfaz síncrona.

- **USB Blaster:** Circuito integrado en la tarjeta DE2 para su programación a través de un cable USB conectado al ordenador.