Metodología PBL en modo colaborativo aplicada al diseño de un SoC

P. Ruiz-de-Clavijo, J. Juan, J. Viejo, M.J. Bellido, E. Ostúa, y D. Guerrero
E.T.S.Ingeniería Informática, Dpto. Tecnología Electrónica - Universidad de Sevilla, Sevilla 41012, Spain
paulino@dte.us.es, jjchico@dte.us.es, julian@us.es, bellido@us.es, ostua@dte.us.es y guerre@dte.us.es

***Resumen*—Dado el carácter principalmente práctico en las asignaturas de los másteres universitarios la metodología PBL es ampliamente utilizada. Este trabajo presenta una experiencia docente de varios años, donde se aplica PBL, pero uniendo a todos los alumnos en un único grupo/equipo para la realización de un proyecto conjunto. Para este fin, se utilizan las técnicas y herramientas usadas en el desarrollo de software y hardware abierto. Este modelo aporta a la docencia dos características: gran interactividad tanto dentro el equipo de trabajo como con el profesor y, el aprendizaje de herramientas y metodologías demostradas de éxito, relacionadas con el desarrollo de proyectos de gran magnitud.**

I. Introducción

En la última década con la implantación de los nuevos planes de estudio adaptados al Espacio Europeo de Educación Superior (EEES) se ha producido un cambio notable en la enseñanza universitaria aumentando considerablemente el peso de la clases de tipo práctico. Este tipo de enseñanza práctica es ideal para abordarlo con metodologías de aprendizaje basadas en proyectos (PBL) y específicamente en el ámbito de la electrónica e ingenierías, el uso de estas metodologías está bastante consolidado.

Este tipo de metodologías, con origen en los años 60 en el ámbito de la medicina, se han ido expandiendo a muchas disciplinas y actualmente está aceptada como una solución eficiente, cuyo principal hito es el concepto de aprender realizando. Hoy en día es ampliamente utilizada a nivel universitario en el ámbito de estudios de ingeniería [1]. En el ámbito de la enseñanza de la electrónica, donde se ubica este trabajo, se pueden encontrar multitud de iniciativas donde se aplican diversas variantes este tipo de métodos. En la mayoría de las situaciones uno de los principales inconvenientes para la utilización de métodos activos de enseñanza es el número de alumnos matriculados. Aún con grupos numerosos existen experiencias como el trabajo presentado en [2] donde se consigue aplicar el método PBL en una asignatura básica de electrónica con cerca de 400 alumnos.

Este trabajo se enmarca dentro de la asignatura Diseño y Aplicaciones de Procesadores Avanzados (DAPA) del Máster Universitario en Ingeniería de Computadores y Redes impartido en la Escuela Técnica Superior de Ingeniería Informática de la Universidad de Sevilla. Este máster no presenta el problema indicado anteriormente respecto al número de alumnos matriculados, facilitando la aplicación y gestión de la asignatura usando metodologías PBL. A pesar de no contar con un número elevado de alumnos inscritos, existen otros desafíos al afrontar la docencia en este máster. El principal aspecto a considerar es la diversidad de titulaciones técnicas de origen de los alumnos. Con este variado perfil de los alumnos, los conocimientos básicos son heterogéneos. Así, en este entorno el primer objetivo al usar este tipo de metodología es facilitar el intercambio de conocimientos mediante el trabajo en equipo [3].

Esta asignatura se imparte a lo largo del primer cuatrimestre y se ha divido en dos grandes bloques. En el primero se intenta homogeneizar, en la medida de lo posible, los conocimientos de diseño digital de los alumnos. Para ello se repasan o enseñan (según el perfil de cada alumno) los principales conceptos de diseño digital usando HDL Verilog. Concretamente se realizan 7 sesiones de laboratorio de 2 horas donde se desarrollan diversos ejemplos de circuitos digitales llegando a implementarlos en las placas de desarrollo tipo FPGA disponibles. El segundo bloque de esta asignatura es el objeto de este trabajo. En él se plantea el desarrollo de un proyecto en equipo, concretamente, un SoC que debe desarrollarse en HDL Verilog realizándose la implementación y testado de este SoC en una placa de desarrollo basada en FPGA. El proyecto se afronta con un único equipo formado por todos los alumnos y el profesor. Para facilitar la evaluación, el proyecto se divide en dos partes principales, en la primera se desarrolla de manera colaborativa la mayor parte del SoC y se testea un primer ejemplo. Tras finalizar esta primera parte cada alumno termina el desarrollo del SoC de manera individual.

Para facilitar la lectura, se resume la organización de este trabajo como sigue: en la siguiente sección se tratan conceptos generales sobre métodos de aprendizaje y desarrollo colaborativo. Además se justifica el uso de ciertas metodologías relacionadas en el software/hardware abierto, enumerando las tecnologías en las que se apoyará la docencia. Antes de presentar detalladamente como se ha aplicado metodología al desarrollo del SoC, en la sección III se describe de manera exhaustiva el SoC propuesto a los alumnos. Una vez descrito el SoC, en la sección IV se presenta como se afronta este desarrollo de manera colaborativa y la metodología utilizada. En la última sección se discuten los resultados obtenidos en los últimos tres cursos en los que se ha ido desarrollando esta experiencia docente.

II. Aprendizaje colaborativo y desarrollo colaborativo

El desarrollo del proyecto planteado se aborda con actividades realizadas en el laboratorio y basadas en procesos colaborativos de resolución problemas. En la diversa literatura existente se acepta el hecho de como el trabajo colaborativo facilita la construcción del conocimiento, favorece el aprendizaje y la atribución de sentido al mismo, características que

difícilmente se producen en la interacción profesor-alumno [4]. Por ello planteamos el papel del profesor como encargado de promover la autonomía e intercambio de conocimientos de los estudiantes. En este proceso es fundamental el traspaso de parte de la responsabilidad sobre el aprendizaje del profesor a los alumnos.

II-A. Software abierto y hardware abierto

La globalización del conocimiento, surgido tras la expansión de Internet, ha llevado a la aparición de nuevos modos de organización de grupos de trabajos disgregados geográficamente y formados por personas muy diversas. Estas últimas décadas han demostrado el éxito de estos nuevos modelos de organización y desarrollos colaborativos, cuya máxima expresión y origen del mismo, es el desarrollo del software libre. Todo este movimiento, cuyo origen se puede datar en 1985 ha dado sus frutos en multitud de software libre donde, proyectos como OpenOffice/FreeOffice, Wikipedia, Firefox y GNU/Linux son considerados buques insignia. Actualmente existen Webs llamadas forjas de software (GitHub, SourceForge, etc.) que son incubadoras de desarrolladores donde colaboran en los proyectos allí alojados. Como muestra de la magnitud de este tipo de desarrollos colaborativos, SourceForge cuenta con más de 2 millones de desarrolladores en más de 200 millones de proyectos activos.

El software libre ha contribuido fuertemente al desarrollo de metodologías que persiguen, principalmente, obtener productos de forma colaborativa en grupos diversos, no muy organizados y deslocalizados geográficamente. De forma paralela, todas las nuevas metodologías organizativas surgidas, las herramientas desarrolladas, etc. han ido aplicándose a otras disciplinas obteniéndose multitud de movimientos que han desembocado en una nueva cultura que apuesta por el libre conocimiento y que aporta diverso material a la comunidad bajo licencias libres como son galerías fotográficas, libros, diseños 3D y en lo que atañe al trabajo que se presenta, el hardware abierto (open hardware).

Actualmente el desarrollo electrónico digital se afronta desde lenguajes de descripción de hardware de alto nivel (HDL). El ciclo de vida de este tipo de productos es similar al de un producto software, por ello el diseño de hardware comparte en una fuerte medida los procesos y metodologías de desarrollo del software. Dada esta similitud la aplicación de cualquier metodología de desarrollo software e incluso el uso de las mismas herramientas que las usadas en el software es perfectamente viable. Partiendo de este principio, basta con estudiar metodologías de desarrollo de software orientadas a facilitar la organización de equipos de trabajo en modo colaborativo, e intentar exportar el proceso al hardware. Prueba de esto, es el gran número de proyectos de hardware abierto, siendo algunos de los más conocidos Arduino [5], Raspberry Pi [6] y OpenRISC [7].

No obstante, para el desarrollo del proyecto planteado se pretende fomentar las relaciones de colaboración entre los estudiantes y con el propio docente, lo cual implica un fuerte intercambio de conocimientos entre los componentes del equipo. Se pretende lograr que el desarrollo y la colaboración no se vea limitada a las 2 horas semanales presenciales facilitando así el trabajo desde cualquier lugar y en cualquier momento.

Para tal fin, utilizaremos las mismas herramientas y tecnologías usadas en el desarrollo de software y hardware abierto, las cuales son universalmente usadas en el desarrollo de todo tipo de proyectos. En la siguiente sección se justifican las escogidas.

II-B. Tecnologías y herramientas utilizadas

En el desarrollo de cualquier proyecto la aplicación de una determinada metodología conlleva el uso de ciertas herramientas asociadas a la misma, cuyo objetivo es facilitar su aplicación. Todas las herramientas en las que se apoyan las metodologías de desarrollo ligadas al software/hardware abierto, generalmente, también son software libre, y cubren todo el ciclo de vida del producto (desarrollo y mantenimiento). Destacan principalmente: sistemas de control de versiones (SCV), sistemas de reporte de incidencias (bugtracker), bases de conocimiento (wikis) y sistemas de auto-documentación.

Actualmente también existen gestores de proyectos que integran todas las herramientas anteriores y más, pero dada la duración del proyecto se ha considerado no usar en su conjunto ninguno de estos gestores. En el desarrollo que nos atañe en la asignatura se ha optado por el uso de dos herramientas: un sistema de control de versiones y un wiki, este último usado como base de conocimiento, punto de encuentro y lugar de discusión.

El uso de unas determinadas herramientas para un desarrollo no implica que todos los miembros de equipo las utilicen de igual modo. Por este motivo hay que considerar que un aspecto crítico en el desarrollo colaborativo es la denominada guía de estilos. En los desarrollos libres disponibles en Internet no es posible contribuir sin usar correctamente la guía de estilos asociada al proyecto. Estas guías de estilo no son más que una serie de normas que consiguen que todas las partes del proyecto sean homogéneas, mejorando con ello la comprensión de las aportaciones de cada miembro del equipo permitiendo el avance del proyecto de forma fluida.

Vistos estos aspectos fundamentales, para comenzar el desarrollo del SoC se utiliza un wiki con una descripción completa: conjunto de instrucciones, arquitectura, periféricos, etc. Los alumnos pueden interactuar en este wiki al estar abierto a la edición. Así, realizan correcciones, cambios menores en la arquitectura, y todo el equipo (incluido el profesor) puede discutir sobre cualquier detalle. A la hora de escoger el wiki a usar nos encontramos con multitud de posibilidades, muchas de ellas integradas en diversos gestores de proyectos. Para minimizar los problemas de mantenimiento de un sitio Web optamos por usar DokuWiki [8] ya que este gestor de contenidos está pensado para realizar despliegues rápidos y es adecuado para nuestro propósito.

Además del wiki, también se usa un SCV que permite a muchos desarrolladores trabajar de forma simultánea en una copia de trabajo obtenida de un repositorio. Concretamente se ha usado los últimos cursos *git* [9] aunque en los dos primeros años de impartición de la asignatura se utilizó *subversion*. *Subversion* está actualmente en desuso y está siendo sustituido paulatinamente por sistemas más avanzados como *git* o *mercurial*. Optar por *git* es debido a su modo de funcionamiento distribuido que tiene innumerables ventajas frente a los que operan de forma centralizada como *subversion*.

Al ser un SCV distribuido, las copias locales del repositorio remoto se comportan como repositorios locales sin ningún tipo de limitación, conteniendo toda la información del historial de cambios. La alta flexibilidad que ofrece esta herramienta hace que aparezcan multitud de métodos organizativos para el trabajo sobre el repositorio. Aunque *git* favorece el uso de ramas para soportar el trabajo paralelo sobre un mismo proyecto, por simplicidad hemos optado por el uso de una única rama para todo el equipo de trabajo. De todos modos, el uso de una única rama no es una imposición, de hecho, algunos los alumnos terminan creando ramas extras para albergar su trabajo, normalmente los que realizan aportaciones extra al desarrollo.

Resumidamente podemos decir que el uso de *git* facilita el control de cambios, versiones y permite el seguimiento del trabajo de cada desarrollador, y este seguimiento, es utilizado por el profesor como herramienta para la evaluación.

III. Especificación y arquitectura del SoC propuesto

En esta sección se presentan la arquitectura y todos los componentes del SoC. El mínimo trabajo exigido a cada alumno es conseguir sintetizarlo en alguna de las placas de desarrollo disponibles. Concretamente se pone a disposición del alumnado placas del fabricante Digilent con FGPA de Xilinx [10]. Esto obliga a usar los entornos de desarrollo de Xilinx. Independientemente de la tecnología final la descripción del procesador se realiza siguiendo una metodología clásica top-down. El SoC está compuesto por un único procesador conectado a una serie de periféricos.

Comenzando por el procesador, la descripción comienza desde el nivel más alto que es el conjunto de instrucciones del procesador (ISP) y está inspirado en la serie de microcontroladores AVR del fabricante ATMEL [11]. Así, el procesador propuesto es un procesador de 8 bits con arquitectura hardvard con un reducido número de instrucciones (RISC) de longitud fija (16 bits). Dispone de una memoria de datos de 256 palabras de 8 bits y una memoria de programa de 256 palabras de 16 bits. Además se ha dotado de un bus de entrada y salida de 8 bits con 256 direcciones. El conjunto de instrucciones propuesto se muestra en la Tabla I, donde la primera columna representa los 5 bits, de ancho fijo, establecidos para el código de operación. Además, se puede ver como no todas las combinaciones de operación se utilizan.

A nivel de arquitectura, en la Figura 1 se muestra el diagrama de bloques del procesador donde se pueden ver en la parte derecha las 5 conexiones que forman el bus de entrada y salida. En la misma figura se presenta la división clásica, en dos bloques, realizada en este tipo de diseños digitales: unidad de proceso (datos) y unidad de control. Básicamente la unidad de control es la máquina de estados que activa secuencialmente cada una de las señales de los componentes de la unidad de datos, con el fin de ejecutar cada una de las instrucciones soportadas por el procesador.

Una propuesta inicial de la unidad de datos se presenta en la Figura 2. A lo largo de los diferentes cursos se ha ido cambiando dicha unidad de datos. Así, la mostrada es la utilizada en el curso 15/16 y está realizada con todos los buses multiplexados. En otros cursos también se han

Tabla I. Conjunto de instrucciones del procesador

Cód.Op.	Sintaxis	Efecto
00000	ST (Rd),Rf	$MEM[Rd] \leftarrow Rf$
00001	LD Rd,(Rf)	$Rd \leftarrow MEM[Rf]$
00010	STS addr,rd	$MEM[addr] \leftarrow Rd$
00011	LDS Rd,addr	$Rd \leftarrow MEM[addr]$
00100	CALL addr	$MEM[SP] \leftarrow PC, SP \leftarrow SP-1, PC \leftarrow addr$
00101	RET	$PC \leftarrow MEM[SP+1], SP \leftarrow SP+1$
00110	BRxx addr	$xx : PC \leftarrow addr$
00111	JMP addr	$PC \leftarrow addr$
01000	ADD Rd,Rf	$Rd \leftarrow Rd + Rf$
01001	-	-
01010	SUB Rd,Rf	$Rd \leftarrow Rd - Rf$
01011	CP Rd,Rf	$Rd - Rf$
01100	-	-
01101	-	-
01110	-	-
01111	MOV Rd,Rf	$Rd \leftarrow Rf$
10000	-	-
10001	-	-
10010	CLC	$SR[C] \leftarrow 0$
10011	SEC	$SR[C] \leftarrow 1$
10100	ROR Rd	$Rd \leftarrow SHR(Rd, C)$
10101	ROL Rd	$Rd \leftarrow SHL(Rd, C)$
10110	-	-
10111	STOP	-
11000	ADDI Rd,data	$Rd \leftarrow Rd + data$
11001	-	-
11010	SUBI Rd,data	$Rd \leftarrow Rd - data$
11011	CPI Rd,data	$Rd - data$
11100	-	-
11101	IN Rd,port_addr	$Rd \leftarrow PORT\_IN[port\_addr]$
11110	OUT port_addr,Rd	$PORT\_OUT[port\_addr] \leftarrow Rd$
11111	LDI Rd,data	$Rd \leftarrow data$

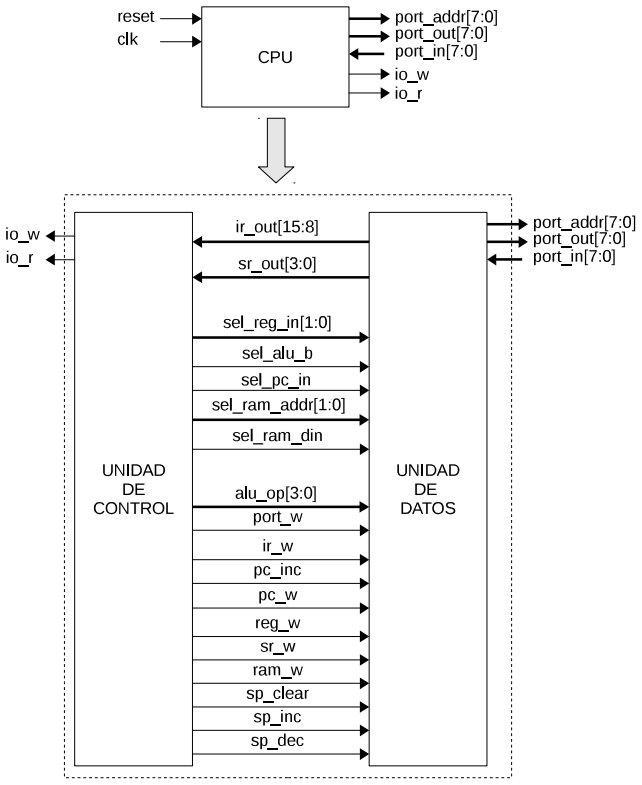

Figura 1. Arquitectura del procesador.

usado unidades de datos con buses compartidos o mixtos, obteniéndose versiones diferentes del procesador en cada curso

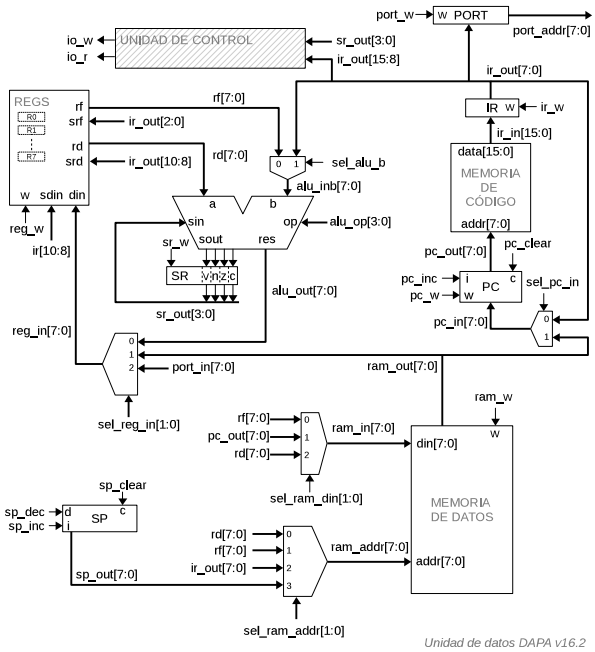

Figura 2. Propuesta inicial de unidad de datos.

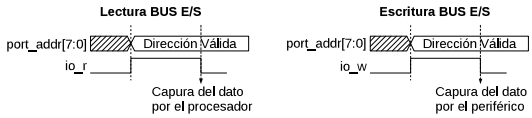

Figura 3. Ciclo de bus de entrada y salida.

académico.

El SoC se completa conectando al procesador unos controladores para los periféricos existentes en las placas FPGA de test. Por ello, además de la arquitectura propuesta se especifica el ciclo de bus de entrada/salida para conectar los periféricos y que estos operen correctamente. El ciclo de bus usado es simple, está sincronizado con el reloj y los datos se leen o escriben en el flanco siguiente de reloj tras la activación de la señal de lectura o escritura (señales *io_r*, *io_w*). Este comportamiento se describe mediante diagramas temporales en la Figura 3.

En el módulo top del SoC se interconectan los controladores de los periféricos. Estos periféricos ya están diseñados, por lo que sólo debe realizarse la integración. Al ser pocos periféricos, se propone utilizar una decodificación parcial del bus de direcciones de entrada/salida. De esta forma los periféricos quedan conectados a las direcciones de puertos mostrados en la Tabla II. Los periféricos suministrados a los alumnos han sido diseñados para reducir la lógica de interconexión para este procesador. Por ello, todos disponen de una señal de habilitación (*enable*) independiente de otra señal de lectura o escritura según sea su función. Concretamente los periféricos son los siguientes:

- Controlador de botones (Figura 4).
- Controlador de display de 7 segmentos (Figura 5). El tipo de placas utilizadas, este display es de 4 dígitos, por ello el controlador permite la escritura de 2 palabras de 8 bits. Estas palabras se seleccionan mediante las señales *w_lsb* y *w_msb*.
- Controlador de leds (Figura 6).
- Controlador de conmutadores (Figura 7).

Tabla II. DIRECCIONES DE PERIFÉRICOS EN EL BUS DE ENTRADA/SALIDA.

Periférico	Tipo	Dirección E/S
Controlador de LEDS	Salida	0x01
Controlador de display 7 segmentos	Salida	0x02 y 0x04
Controlador de conmutadores	Entrada	0x01
Controlador de pulsadores	Entrada	0x02

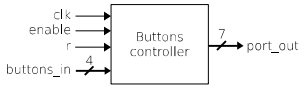

Figura 4. Controlador de pulsadores.

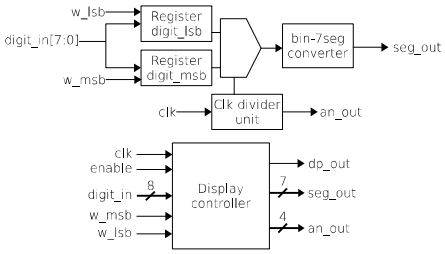

Figura 5. Controlador de display de 7 segmentos.

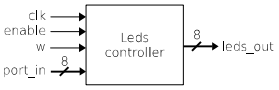

Figura 6. Controlador de leds.

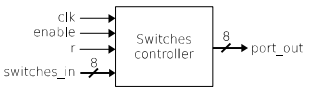

Figura 7. Controlador de conmutadores.

Finalmente todo el sistema es sintetizado y testado usando algunos programas de test. Todo este proceso se describe en la siguiente sección.

IV. METODOLOGÍA DE DESARROLLO

Las clases de laboratorio consisten en sesiones de dos horas y se dispone de 7 sesiones para completar esta parte de la asignatura. En la primera sesión se pone a disposición del alumnado los recursos disponibles para el desarrollo del SoC. Entre ellos está la descripción completa del diseño mostrado en la sección anterior. Toda esta información se centraliza en el wiki de la asignatura, el cual es editable en algunas secciones por los alumnos. Cabe recordar que este desarrollo se afronta tras la finalización de la primera parte de la asignatura, por tanto, se parte de unos conocimientos mínimos de HDL y de diseño digital. Como objetivo final, el SoC completo debe

sintetizarse y testarse en alguna de las placas de desarrollo disponibles y de manera individual por cada alumno.

Como punto de partida se usa el desarrollo del procesador. Así, el primer día de clase y en la primera hora se presenta el SoC a desarrollar usando la documentación disponible en el wiki. En este wiki el alumno encuentra el siguiente contenido:

- Conjunto de instrucciones del procesador. Este conjunto es un requisito inmutable, es decir, los alumnos no pueden cambiar las instrucciones propuestas.
- Propuesta de arquitectura del SoC y esquema detallado de los componentes que formarán el procesador y su interconexión.
- Descripción exhaustiva de algunos componentes.
- Descripción de la entrada/salida: ciclo de bus y periféricos.
- Conjunto mínimo de programas a simular y ejecutar.
- Documentación sobre la guía de estilos y repositorio *git* compartido.

En esta primera clase se presta especial atención a la guía de estilos, la cual es propuesta para facilitar el intercambio de trabajo entre todo el equipo de desarrollo. Esta guía incluye aspectos funcionales, de organización de contenidos y de sintaxis de ficheros. Se resume en los siguientes puntos:

- Se establece el flanco positivo de reloj para todos los componentes.
- Todas las señales de inicialización serán síncronas y activas en alto.
- Todas las unidades funcionales deben estar acompañadas de un *testbench*.
- Todos los ficheros se ubicarán bajo una carpeta llamada *rtl*.
- Los nombres de ficheros deben estar en minúsculas.
- El nombre del fichero del *testbench* debe ser el mismo que el del componente, pero terminado en *_tb*.
- Cada fichero debe tener en la primera línea el nombre del autor.
- Todas las conexiones estructurales se deben hacer por nombre.
- Se debe usar correctamente la tabulación para hacer el código legible y se decide utilizar 4 espacios en lugar de tabuladores.
- Los nombres de señales, variables y constantes deben tener relación con su finalidad. Se deben evitar nombres genéricos.
- Usar exactamente los mismos nombres de señales que los existentes en la descripción estructural del SoC mostrada en los esquemas.
- Los nombres de los módulos, señales y variables deben estar en minúsculas. Los nombres de las constantes se pondrán en mayúsculas.

Componentes de la unidad de datos

Componente	Ficheros	Desarrollador/es
PORT	port.v, port_tb.v	JAKOB
PC	pc.v, pc_tb.v	CRISTINA
SP	sp.v, sp_tb.v	ROYLAN
ALU	alu.v, alu_tb.v	RICARDO
IR	ir.v, ir_tb.v	JAKOB
SR	sr.v, sr_tb.v	ROYLAN
REGS	regs.v, regs_tb.v	DANIEL
MEM DATOS	ram.v, ram_tb.v	JESUS
MUX4	mux4.v, mux4_tb.v	ANTONIO
MUX2	mux2.v, mux2_tb.v	ANTONIO

Uniones estructurales de componentes

Componente	Ficheros	Desarrollador/es	Descripción
u_datos	u_data.v	Unidad de datos	CRISTINA
u_control	u_control.v	Unidad de control	INDIVIDUAL
cs2016	cs2016.v	Procesador completo	ROYLAN
–	soc.v	Conexiones con Basys2	sin asignar

Unidad de memoria

Programa	Fichero	Descripción	Desarrollador
programa1	memory_1.v	Programa con 2 instrucciones	sin asignar
programa2	memory_2.v	Programa con entrada y salida	sin asignar
programa3	memory_3.v	Programa aritmético	sin asignar
programa4	memory_4.v	Programa subrutinas	sin asignar

Figura 8. Asignación inicial de componentes.

- Se recomienda el uso de comentarios en las partes donde existan muchas bifurcaciones o procesos.

En esta primera clase se realiza una discusión de las guías de estilos en general, haciendo referencia a proyectos reales. Tras esta discusión, se realiza una asignación inicial de trabajo. A cada alumno se le propone la realización de un componente con la finalidad de comenzar el trabajo con la herramienta *git*. Llegados a este punto comienza el desarrollo del proyecto y se divide en las siguientes fases:

1. Asignación de los componentes a los alumnos.
2. Simulación del primer programa y corrección de errores.
3. Interconexión de periféricos y síntesis del SoC.
4. Terminación de la unidad de control con todas las instrucciones y ejecución de todos los programas propuestos.

La primera fase del desarrollo comienza en la segunda mitad de la primera clase. El objetivo de esta primera fase es familiarizar a los alumnos con el uso de la herramienta de control de versiones *git* y se realiza de forma individual. Así, los alumnos obtienen del repositorio *git* una versión de partida con varios ficheros que forman parte del SoC. Estos ficheros corresponden a los componentes y tienen definido en HDL Verilog el nombre del módulo, sus entradas y sus salidas, pero el cuerpo está vacío. Como apoyo, en el wiki se dispone de tablas con todos estos componentes donde el profesor los asigna a diferentes alumnos. En la Figura 8 se muestra una captura del wiki con la asignación realizada en el curso 15/16.

Ya se indicó que por simplicidad en el uso de *git* se propone trabajar sobre una única rama. En esta primera fase de desarrollo no se suelen producir conflictos, ya que cada alumno tiene ficheros diferentes asignados. Al trabajar sobre una única rama, en el momento de subir los cambios puede que *git* le

obligue a mezclar sus cambios con los del repositorio remoto, si algún compañero ha subido sus cambios antes. Realizar este procedimiento de mezcla de versiones sirve para introducir a los alumnos que desconozcan *git* en el uso de esta herramienta.

En la segunda fase cada alumno debe completar la unidad de control del microprocesador para que sea capaz de ejecutar las instrucciones mostradas en el Código 1. En esta fase el objetivo es corregir todos los problemas existentes en los componentes e interconexiones realizadas en la fase anterior, pero con una peculiaridad, la unidad de control no se debe compartir, cada alumno desarrolla una versión propia. Esta parte del trabajo es colaborativo en la corrección de errores del resto de componentes. Los alumnos al ir corrigiendo posibles errores existentes, van actualizando el repositorio *git*. Aquí suelen surgir algunos conflictos que deben de solucionar.

En el programa mostrado en el Código 1 sólo hay 3 instrucciones y hace uso de la entrada/salida. Concretamente se activan los leds escribiendo el dato AA en el periférico que lo controla. Por ello, en este punto es necesario conseguir integrar al menos el controlador de leds y conectarlo correctamente al procesador. En la integración de los periféricos también se hace uso de *git* ya que los periféricos desarrollados por el profesor están disponibles en un repositorio de GitHub y deben incluirlo en su proyecto. Esta fase termina cuando por simulación se verifica la correcta ejecución del programa. Una vez conseguido este primer objetivo se marca mediante una etiqueta en el repositorio.

Código 1. Primer programa de prueba

```
LDI  R0,0xAA
OUT  0x01,R0
STOP
```

La tercera fase tiene como principal objetivo introducir a los alumnos el procedimiento de entrada/salida usado en el procesador. Para ello deben integrar el resto de periféricos ya diseñados, y deben integrarlos sin realizar cambios en ellos. La forma de testar el resultado es ejecutar los dos programas (Código 1 y 2) sobre la placa de desarrollo. El segundo programa ya hace uso de la entrada mediante la instrucción *IN*. Tras su ejecución se transfiere el dato leído desde los conmutadores a los leds. Tener que ejecutar los programas en la placa de desarrollo implica que deben corregirse todos los fallos de síntesis existentes en los componentes. De nuevo las correcciones vuelven a subirse y compartirse en el repositorio y todos los alumnos deben mantener actualizados sus diseños con las correcciones que surjan.

La cuarta y última fase es abordada individualmente por cada alumno. Deben terminar la unidad de control para que sea capaz de ejecutar todas las instrucciones propuestas, además deben ejecutar correctamente otros programas como el mostrado en el Código 3. Estos programas ya incluyen saltos condicionales y llamadas a subrutinas. Aunque esta parte es individual, las instrucciones restantes hacen uso de componentes no testados previamente (la ALU, la memoria, etc), por ello aparecen nuevos errores a corregir y se deben subir al repositorio compartido.

Código 2. Programa testeo de la entrada y salida.

```
LDI R0,0xAA
OUT 0x01,R0
IN  R1,0x01
OUT 0x02,R1
STOP
```

Código 3. Programa con cálculo aritmético y saltos

```
  LDI  R0,0x08
  LDI  R1,0x10
  LDI  R2,0x00
BUCLE:
  SUBI R1,0x01
  BRZS FIN
  ADD  R2,R0
  JMP  BUCLE
FIN:
  OUT  0x02,R2
  STOP
```

Llegados a este estadio del desarrollo algunos alumnos optan por realizar aportaciones propias y suelen ser diversas. En estas aportaciones el uso de *git* también facilita la tarea. Se propone crear ramas de trabajo con propuestas originales y no planteadas en el diseño inicial. Habitualmente, los alumnos añaden al SoC diferentes capacidades como: nuevos periféricos, controlador de interrupciones, nuevas instrucciones, etc. En la sección de resultados se presentan algunos ejemplos de aportaciones realizadas en lo últimos cursos.

Respecto a la evaluación, esta segunda parte de la asignatura pesa un 50 % del total de la asignatura. El requisito mínimo para aprobar esta parte es conseguir la ejecución de los tres programas presentados (Códigos 1, 2 y 3). Estos programas incluyen sólo un subconjunto pequeño de las instrucciones del procesador. A partir de este punto, los alumnos deben continuar implementando las restantes instrucciones y probándolas mediante test de simulación y programas de ejemplo ejecutados en la placa de desarrollo. En este estado la nota máxima alcanzable es un 8. Para obtener más nota se propone la aportación de nuevas características al SoC como pueden ser nuevas instrucciones, periféricos o algún tipo de aplicación del mismo.

V. Resultados y conclusiones

De manera general, destacamos como durante los sucesivos años académicos de impartición de la asignatura se ha conseguido equilibrar el trabajo colaborativo y responsabilidad individual en la carga de trabajo que la asignatura supone a los alumnos. Además, los estudiantes aprecian especialmente el planteamiento realizado como proyecto único y la forma de afrontar el desarrollo colaborativo del SoC considerándolo relevante para su futura práctica profesional. Igualmente, valoran de manera muy positiva desde el punto de vista técnico las herramientas utilizadas como son *git* y el wiki.

Respecto a las tareas docentes el profesor puede realizar un seguimiento continuo de la evolución del desarrollo, pudiéndose tomar medidas correctivas con antelación. Principalmente en la primera parte del desarrollo, el profesor sigue continuamente las confirmaciones (*commits*) hechas en el repositorio. Estos son una gran fuente de información, ya que además, puede consultar el código y preparar con antelación la siguiente sesión de laboratorio donde afrontar los posibles errores detectados. Uno de los principales problemas suele ser que los alumnos no siguen la guía de estilos y además, esta situación es recurrente en todos los cursos académicos. Esto conlleva muchos problemas de integración con los compañeros. Con

Tabla III. RESUMEN ESTADÍSTICO OBTENIDO DEL REPOSITORIO EN EL CURSO 2013/2014.

Autor	Confirmaciones	Líneas de código
paulino	44 (42.2 %)	1144 (35.6 %)
yamrodest	13 (12.6 %)	604 (18.6 %)
carlopmol	14 (13.6 %)	412 (12.7 %)
mjdominguez	5 (4.8 %)	316 (9.8 %)
alfromzap	5 (4.8 %)	201 (6.2 %)
salpulgom	5 (4.8 %)	179 (5.5 %)
indlancru1	5 (4.8 %)	138 (4.2 %)
tomrubcam1	4 (3.9 %)	76 (2.3 %)
manpesgon	3 (2.9 %)	75 (2.3 %)
eugdepsan	2 (1.9 %)	61 (1.9 %)
joshurrom	3 (2.9 %)	48 (1.4 %)
Totales	103 (100.0 %)	3254 (100.0 %)

Tabla IV. RESUMEN ESTADÍSTICO OBTENIDO DEL REPOSITORIO EN EL CURSO 2014/2015.

Autor	Confirmaciones	Líneas de código
paulino	88 (41.5 %)	1641 (39.2 %)
javromlem	50 (23.6 %)	1215 (29.0 %)
juadommor2	8 (3.8 %)	318 (7.6 %)
marara	6 (2.8 %)	316 (7.6 %)
danpercho	1 (0.5 %)	150 (3.6 %)
dieramjim	7 (3.3 %)	131 (3.1 %)
abdel	5 (2.4 %)	120 (2.9 %)
josbonrod1	2 (0.9 %)	104 (2.5 %)
diaromcoy	1 (0.5 %)	80 (1.9 %)
edepercas	39 (18.4 %)	56 (1.3 %)
enrmorayu	5 (2.4 %)	54 (1.3 %)
Totales	202 (100.0 %)	4185 (100.0 %)

Tabla V. RESUMEN ESTADÍSTICO OBTENIDO DEL REPOSITORIO EN EL CURSO 2015/2016.

Autor	Confirmaciones	Líneas de código
paulino	19 (34.62 %)	1865 (46.3 %)
jakob	8 (15.38 %)	379 (9.1 %)
jesus	7 (13.46 %)	234 (5.6 %)
jackzkay	6 (11.54 %)	185 (4.4 %)
daniel	3 (5.77 %)	487 (11.7 %)
ricardo	2 (3.85 %)	266 (6.4 %)
roylan	2 (3.85 %)	245 (5.9 %)
antonio	2 (3.85 %)	185 (4.4 %)
ricardo	1 (1.92 %)	5 (0.1 %)
jesus	1 (1.92 %)	94 (2.3 %)
cristina	1 (1.92 %)	218 (5.2 %)
Totales	52 (100.0 %)	4163 (100.0 %)

todos estos datos, al principio de cada sesión de laboratorio el profesor realiza una exposición de los problemas detectados en el estado actual de desarrollo y solicita cambios a los alumnos involucrados.

Como ejemplos de seguimientos de cursos anteriores en las Tablas III, IV y V se presentan estadísticas de los tres últimos cursos. Estas tablas muestran el número de confirmaciones y líneas de código aportados por cada desarrollador. Estos resultados sirven como indicadores de la actividad desarrollada por cada alumno. Así, el número de confirmaciones indica los alumnos que avanzan mas rápido en el desarrollo, ya que van aportando más componentes y corrigiendo errores. El número de líneas de código suele ser alto para aquellos alumnos que realizan aportaciones propias, ya que añaden ficheros no existentes en el diseño original.

Para un mejor análisis, el historial de *git* es una gran fuente de información sobre la evolución del desarrollo del trabajo y la actividad de cada alumno. En la Figura 9 se muestra el historial de desarrollo del curso 15/16 donde se pueden realizar

Figura 9. Seguimiento con *git* durante el curso 2015/16.

observaciones y extraer algunas conclusiones como:

- La mitad inferior de la figura muestra gran cantidad de trabajo colaborativo. Todos los alumnos se ven obligados a ir mezclando sus cambios con los de los compañeros.
- Existe una etiqueta donde se genera una rama que indica cuando el primer programa ya se ejecuta correctamente en las placas de desarrollo. Este es un punto de inflexión ya que comienza un desarrollo más individual hasta completar el SoC.
- Otra observación interesante son las fechas de las confirmaciones en el repositorio. La mayoría de ellas se realizan en los días de clase, pero se puede observar como algunos alumnos realizan trabajo fuera de este horario y van subiendo cambios al repositorio.

La iniciativa de los alumnos para la mejora del SoC no es generalizada. De entre los 10-15 alumnos matriculados cada curso académico, sólo entre 2 o 3 continúan con el diseño tras terminar el desarrollo del conjunto completo de instrucciones. De las aportaciones propias realizadas de manera individual por algunos alumnos en los últimos años, cabe destacar las siguientes:

- Nuevas instrucciones: PUSH, POP, SHL, SHR, etc. (curso 2013/14 y 2014/15).
- Controlador de interrupciones junto con las instrucciones necesarias asociadas (curso 2013/14).
- Sistema de arranque (bootloader) desde memoria flash en formato SD, basado en un controlador de tarjetas

SD añadido como periférico (curso 2014/15).

- Módulo de depuración remota (curso 2013/14).
- Ensamblador para el procesador (curso 2013/14 en C, curso 2015/16 en javascript y curso 2015/16 en C++).
- Integración con un controlador VGA añadido como periférico (curso 2015/16).

REFERENCIAS

[1] J. E. Mills, D. F. Treagust *et al.*, "Engineering education—is problem-based or project-based learning the answer," *Australasian Journal of Engineering Education*, vol. 3, no. 2, pp. 2–16, 2003.

[2] M. A. Perales, F. Barrero, L. Sergio, T. Marín, and M. J. Durán, "Experiencia pbl en una asignatura básica de electrónica." *IEEE-RITA*, vol. 7, no. 4, pp. 223–230, 2012.

[3] S. Freeman, S. L. Eddy, M. McDonough, M. K. Smith, N. Okoroafor, H. Jordt, and M. P. Wenderoth, "Active learning increases student performance in science, engineering, and mathematics," *Proceedings of the National Academy of Sciences*, vol. 111, no. 23, pp. 8410–8415, 2014.

[4] T. M. Majós, J. Onrubia, and C. C. Salvador, "Análisis y resolución de casos-problema mediante el aprendizaje colaborativo," *RUSC. Universities and Knowledge Society Journal*, vol. 3, no. 2, p. 8, 2006.

[5] "Arduino," URL: https://www.arduino.cc, (visited on 21/02/2016).

[6] "Raspberry pi foundation," URL: https://www.raspberrypi.org/, (visited on 21/02/2016).

[7] "Openrisc community," URL: http://openrisc.io, 2014, (visited on 21/02/2016).

[8] A. Gohr, "Dokuwiki," URL: http://dokuwiki.org, 2010, (visited on 21/02/2016).

[9] S. Chacon, *Pro Git*, 2nd ed. New York: Apress, 2014.

[10] U. Guide, "Spartan-3 starter kit board," *Digilent Inc*, 2005.

[11] A. Atmel, "Microcontrollers," *STK500 Starter Kit and Development system*, 2007.