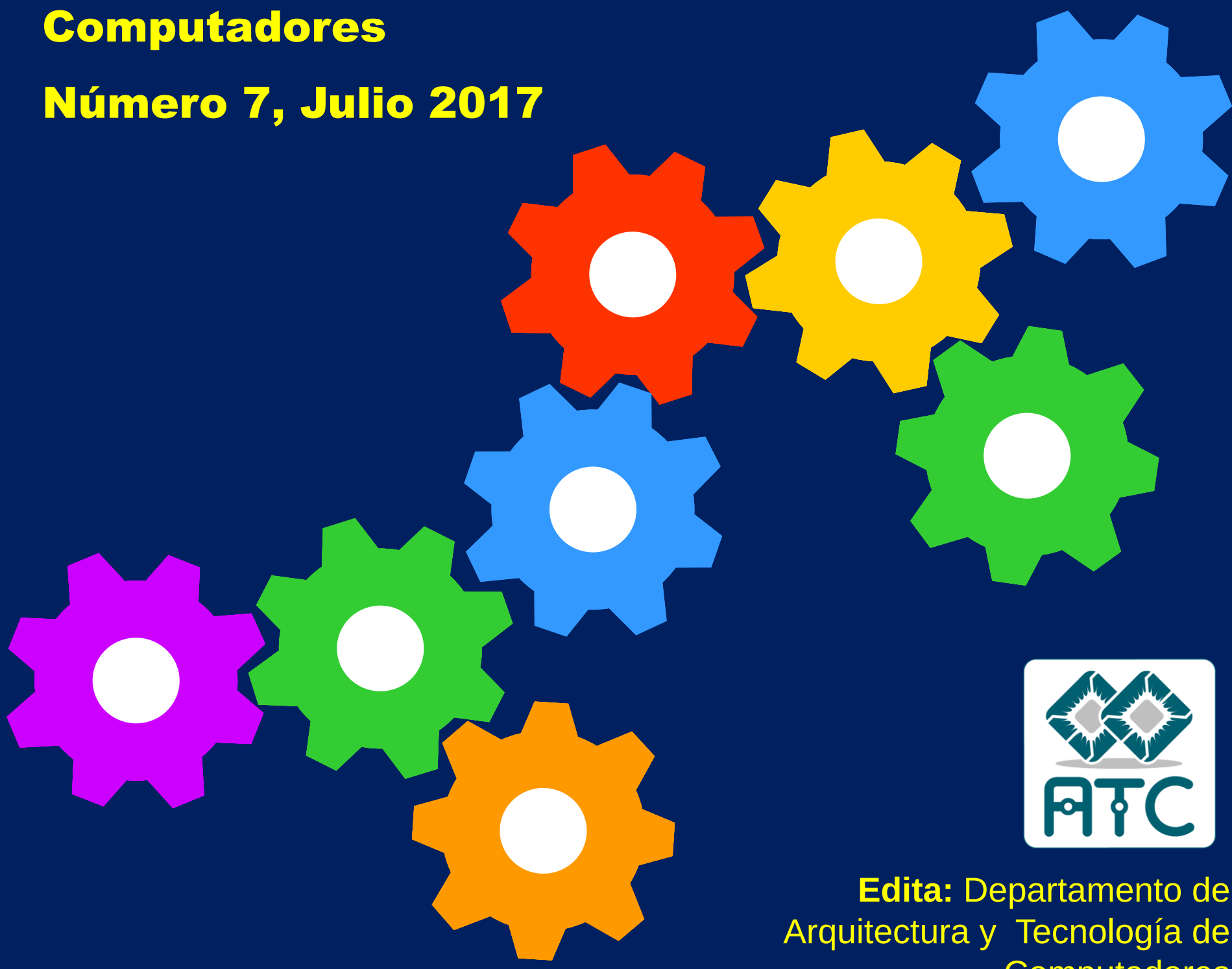
Enseñanza y Aprendizaje de Ingeniería de Computadores

Revista de Experiencias Docentes en Ingeniería de Computadores

Número 7, Julio 2017



Edita: Departamento de Arquitectura y Tecnología de Computadores

Colabora: Vicerrectorado para la Garantía de la Calidad

ENSEÑANZA Y APRENDIZAJE DE INGENIERÍA DE COMPUTADORES
Revista de Experiencias Docentes en Ingeniería de Computadores

TEACHING AND LEARNING COMPUTER ENGINEERING
Journal of Educational Experiences on Computer Engineering

Número 7, Año 2017

Comité Editorial:

ISSN: 2173-8688 **Depósito Legal:** GR-899/2011
Edita: Departamento de Arquitectura y Tecnología de Computadores
Imprime: Copicentro Editorial

Printed in Spain Impresa en España

Índice

Editorial 1

La plataforma Rasberry Pi como base para la coordinación vertical
R. Asenjo. S. González, F. Corbera, Á. Navarro, A. Rodríguez, J. Villalba, E. Hendrix 5

Uso de la infraestructura docente MIPSfpga v2.0 en la asignatura Arquitectura de Sistemas Integrados
D. Chaver, Y. Panchul, E. Sedano, D.H. Harris, R. Owen, Z.L. Kakakhel, B. Ableiginger, S.L. Harris 21

Uso de dispositivos FPGA como apoyo a la enseñanza de asignaturas de arquitectura de computadores
F. Charte, M. Espinilla, A.J. Rivera, F. Pulgar 37

Bomberman modo multijugador
H. Ivanov, A. Lorente, V. Chammorro, A.A. del Barrio, G. Botella 53

Sistema de control de bajo coste de un motor de corriente continua para usos didácticos
G. Olivares, A. Olivares, F. Gómez, M. Damas 69

Consumo de energía y asignaturas de arquitectura y tecnología de Computadores
A.F. Díaz, J. Ortega, J.J. Escobar, M. Anguita, J. González, M. Damas 79

La asignatura “Internet de las Cosas” en el máster DATCOM de la UGR
M. Damas, F. Gómez, S. Moreno, C. Bailón, A. Olivares 93

Evolución Tecnológica del hardware de vídeo y las GPU en los ordenadores personales
F. Charte, A.J. Rueda, M. Espinilla, A.J. Rivera 111

DESDE EL PUPITRE
(Experiencias de estudiantes)

Problema del viajante de comercio con GPU
J.R. Cenit Röleke, M.G. Arenas 131

Honeynet para el análisis del tráfico y muestras de malware
S. de Diego de Diego, G. Romero 145

Instrucciones para autores ……..… 157

Editorial

Es una satisfacción presentar los diez artículos que constituyen este séptimo número de la revista Enseñanza y Aprendizaje de Ingeniería de Computadores. Cinco de esos trabajos provienen de la Universidad de Granada (dos de ellos corresponden a contribuciones de estudiantes), otros dos provienen de la Universidad Complutense de Madrid (aunque en uno de ellos intervienen investigadores y profesores de otras tres instituciones extranjeras más), uno de la Universidad de Málaga, y dos de la Universidad de Jaén.

Tres de los artículos se centran en el uso de distintas plataformas e infraestructuras para la docencia de asignaturas de ingeniería de computadores. En *La plataforma Rasberry Pi como base para la coordinación vertical* de Rafael Asenjo, Sonia González, Francisco Corbera, Ángeles Navarro, Andrés Rodríguez, Julio Villalba y Eligius Hendrix, de la Universidad de Málaga se ilustra una interesante iniciativa, desarrollada a partir de un proyecto de innovación docente, que propone el uso de la Raspberri Pi. Utilizando esta plataforma como base para la docencia de distintas asignaturas de ingeniería de computadores relacionadas, se promueve su coordinación vertical y la motivación de los estudiantes, bastante interesados en la Raspberri Pi, como por otra parte se pone de manifiesto en varios artículos de este número. El artículo *Uso de la infraestructura docente MIPSfpga v2.0 en la asignatura Arquitectura de Sistemas Integrados* de Daniel Chaver, Yuri Panchul, Enrique Sedano, David H. Harris, Robert Owen, Zubair L. Kakakhel, Bruce Ableiginger, y Sarah L. Harris de la Universidad Computense de Madrid, Imagination Technologies Ltd., Harvey Mudd College y la Universidad de Nevada, detalla la aplicación de la infraestructura MIPSfpga v2.0, centrada en el núcleo de código abierto microAptiv de MIPS, a la docencia de la asignatura Arquitectura de Sistemas Integrados de la Universidad Complutense. En *Uso de dispositivos FPGA como apoyo a la enseñanza de asignaturas de arquitectura de computadores* de Francisco Charte, Macarena Espinilla, Antonio J. Rivera y Francisco Pulgar, de la Universidad de Jaén, se propone el diseño de prácticas basadas en el uso de FPGAs para complementar las prácticas de programación en ensamblador usuales en las asignaturas de arquitectura de computadores del grado en ingeniería informática. De esta forma, se introducirían competencias relacionadas con el diseño de procesadores a partir de sus componentes.

La fuerza motivadora de la programación de juegos y de la plataforma Raspberry Pi se pone de manifiesto en el artículo *Bomberman modo multijugador* de Hristo Ivanov, Alberto Lorente, Verónica Chammorro, Alberto A. del Barrio y Guillermo Botella, de la Universidad Complutense de Madrid. Se trata de una adaptación del juego *Bomberman*

en modo multijugador utilizando dos placas de desarrollo S3CEV40 y Raspberri Pi 2 para la asignatura de máster Sistemas Empotrados Distribuidos.

Además de la Raspberry Pi, la plataforma Arduino también ha despertado gran interés entre los estudiantes y, ofrece muchas posibilidades, entre otras cosas por su bajo coste, para la docencia de asignaturas relacionadas con el hardware. En *Sistema de control de bajo coste de un motor de corriente continua para usos didácticos* de Gonzalo Olivares, Alberto Olivares, Francisco Gómez y Miguel Damas, de la Universidad de Granada, se describen experiencias prácticas en asignaturas de control digital de un motor de corriente continua utilizando una tarjeta Arduino Mega 2560 R3 como controlador.

En *Consumo de energía y asignaturas de arquitectura y tecnología de computadores* de Antonio F. Díaz, Julio Ortega, Juan José Escobar, Mancia Anguita, Jesús González y Miguel Damas, de la Universidad de Granada también se describe un sistema basado en Arduino Mega, en este caso un sistema para medir la energía consumida por un computador. Además, en el artículo se discute la inclusión de contenidos relacionados con el consumo de energía en asignaturas que abordan la optimización de código y el desarrollo de programas eficientes, además de en las que abordan el diseño hardware o la gestión de centros de procesamiento de datos.

La relevancia del denominado Internet de las Cosas (IoT, por sus siglas en inglés) y su relación con la ciencia de datos y el Big Data se aborda en el artículo *La asignatura "Internet de las Cosas" en el máster DATCOM de la UGR*, de Miguel Damas, Francisco Gómez, Salvador Moreno, Carlos Bailón, y Alberto Olivares, de la Universidad de Granada. Además de motivar la inclusión de la asignatura en el máster oficial de "Ciencia de Datos e Ingeniería de Computadores" de la Universidad de Granada, y describir la metodología seguida para impartirla, el artículo proporciona un breve pero muy completo recorrido histórico del IoT y una revisión de sus aplicaciones, de las tecnologías habilitadoras, y de las plataformas existentes.

El interés que despiertan las GPUs también es patente en este séptimo número de la revista, que incluye dos artículos centrados en este tópico. Por un lado, en *Evolución Tecnológica del hardware de vídeo y las GPU en los ordenadores personales* de Francisco Charte, Antonio J. Rueda, Macarena Espinilla y Antonio J. Rivera, de la Universidad de Jaén, se revisan los logros más relevantes en la evolución del hardware para el procesamiento de gráficos hasta la incorporación de GPUs en los computadores personales y del hardware para gráficos en el propio microprocesadores. Además, en el artículo *Problema del viajante de comercio con GPU* de José Rafael Cenit Röleke, y la profesora M. G. Arenas, de la Universidad de Granada, se ilustra la utilidad de la programación paralela con CUDA en el problema de optimización del Viajante de Comercio, a través del trabajo realizado por un estudiante dentro de la asignatura Arquitectura y Computación de Altas Prestaciones de la especialidad en ingeniería de computadores.

También en este número se incluyen varios artículos dentro de la sección "Desde el pupitre", que iniciamos en el número de 2014 con artículos elaborados por estudiantes (con o sin participación de profesores). Unos de los objetivos de esta revista es promover el estudio de asignaturas de ingeniería de computadores entre los estudiantes universitarios de ingeniería informática. Por ello, es importante involucrar a los estudiantes en los proyectos y en la elaboración de artículos que, con la correspondiente

supervisión, sus propuestas y sus inquietudes en los tópicos relacionados con la ingeniería de computadores. Junto con el artículo relacionado con las GPUs al que nos hemos referido antes, se incluye en esta sección el trabajo *Honeynet para el análisis del tráfico y muestras de malware*, de Santiago de Diego y el profesor Gustavo Romero López, de la Universidad de Granada, En este trabajo, que describe el despliegue de *honeypots* en dispositivos Raspberry Pi para analizar los ataques dirigidos a la red de la UGR, se pone de manifiesto el interés de los estudiantes en la plataforma Raspberri Pi y en la seguridad informática.

Como en los seis números anteriores, algunos de los artículos de este séptimo número están relacionados con temas discutidos en las Jornadas de Coordinación Docente y Empresas, JCDE (http://atccongresos.ugr.es/jcde/), organizadas por el Departamento de Arquitectura y Tecnología de Computadores, cuya séptima edición tuvo lugar en Diciembre de 2016. Nuestro agradecimiento a todos los que han contribuido a la celebración de esas VII JDCE, y también al Comité Editorial de la revista por su inestimable ayuda en la difusión de la revista y en la revisión de los artículos.

El Comité Editorial

La plataforma Raspberry Pi como base para la coordinación vertical

Rafael Asenjo, Sonia González, Francisco Corbera, Ángeles Navarro, Andrés Rodríguez, Julio Villalba and Eligius Hendrix

Depto. de Arquitectura de Computadores. E.T.S.I. Informática, E.T.S.I. Telecomunicación, Universidad de Málaga
{asenjo,sgn,fjcorbera,magonzalez,andres,jvillalba,eligius}@uma.es

Abstract. This article describes the process of making use of the interest of students for the Raspberry Pi platform to facilitate study of concepts and techniques in several Engineering courses. Moreover, we show how this platform was used to improve the vertical coordination in the Computer Engineering study from 1st to 4th year in the University of Málaga. The stimulus for this teaching experiment was an educational innovation project PIE13-082 at this university. This project helped to renew the theoretical material and laboratory guides for several courses leading to an improvement of the interest and satisfaction of the students.

Keywords: Raspberry Pi, motivation, vertical coordination, renewing laboratory material.

Resumen. En este artículo exponemos cómo hemos aprovechado el interés que los alumnos demuestran por la plataforma Raspberry Pi para facilitar el estudio de conceptos y técnicas impartidas en varias asignaturas de Ingeniería. Además, proponemos usar esta misma plataforma en distintos cursos de forma que se mejore la coordinación vertical en asignaturas de primero a cuarto del Grado en Ingeniería de Computadores de la Universidad de Málaga. El detonante para realizar esta experiencia fue el Proyecto de Innovación Educativa PIE13-082 de la misma universidad. Gracias a este proyecto, se impulsó la renovación de los temarios teóricos y guiones de prácticas de distintas asignaturas lo cual ha resultado en un aumento del interés y en un mayor grado de satisfacción de los alumnos.

Palabras clave: Raspberry Pi, Motivación, Coordinación Vertical, Renovación de Temarios.

1 Introducción

Muchos alumnos de ingeniería perciben que las asignaturas de la carrera no son atractivas y están alejadas de su realidad cotidiana. Sin embargo, a la mayoría de estos alumnos les atrae el minicomputador Raspberry Pi (RPi) [16], muy popular en foros tecnológicos relacionados con domótica y robótica. En este trabajo detallamos cómo

hemos aprovechado el interés que los alumnos ya demuestran por esta plataforma para ponerlo a trabajar en pro de nuestros objetivos docentes: facilitar el estudio de conceptos y técnicas impartidas en varias asignaturas del Departamento de Arquitectura de Computadores de la Universidad de Málaga (UMA). Otro objetivo que se suma al anterior es el de usar la misma plataforma Raspberry Pi para los laboratorios de varias asignaturas de distintos cursos (de 1º a 4º), de forma que se mejore la coordinación vertical. Para la consecución de estos objetivos se han elaborado nuevos guiones de prácticas basados en la RPi para varias asignaturas del departamento.

El uso en la docencia de tecnologías familiares al alumno con objeto de incrementar el atractivo de las asignaturas no es algo nuevo. Por ejemplo, en [2] los autores plantean enseñar Sistemas Operativos, pero no usando Windows o Linux como plataforma para las prácticas, sino Android (el Sistema Operativo, SO, usado en casi el 90% de los Smartphones [3]). Dado que las nuevas generaciones de estudiantes se interesan más por un Smartphone que por un PC, esta estrategia conduce a contenidos más motivadores.

También en [4] se propone potenciar el interés de los alumnos del Grado de Informática en el análisis de los componentes hardware y de su organización para construir un computador moderno mediante el uso de placas basadas en la arquitectura ARM como RPi y Arduino.

La RPi es también la plataforma elegida para motivar la enseñanza de ensamblador en procesadores ARM en [5]. Los autores corroboran en ese trabajo que los alumnos encuentran que el aprendizaje del lenguaje ensamblador es bastante aburrido y tedioso, pero que gracias a la RPi los contenidos se tornan más interesantes y divertidos. Sin embargo, este reciente estudio solo aprovecha la RPi para enseñar una única materia, mientras que nosotros pretendemos que el alumno utilice la RPi como la plataforma de referencia para integrar los distintos conceptos que se van a adquirir en diferentes asignaturas de cursos consecutivos. En esta última línea, en [6] y [7] el objetivo es mejorar la coordinación vertical integrando los distintos conceptos aprendidos en diferentes asignaturas.

En todos estos trabajos mencionados se utilizan distintas tecnologías familiares a los alumnos, pero hasta donde conocemos, ningún trabajo previo considera central el objetivo de la coordinación vertical entre distintas asignaturas al tiempo que se reutiliza la misma plataforma hardware para motivar el aprendizaje de las distintas materias que aparecen en el currículo del Grado de Ingeniería de Computadores.

La coordinación docente vertical es importante para poder desarrollar adecuadamente las competencias de un título y puede contribuir a potenciar el trabajo colaborativo, además de prevenir una formación donde, por ejemplo, se produzcan solapamientos o repeticiones de materia en distintas asignaturas [8], o se fragmente el conocimiento de forma artificial [9]. En [10] se potencia la coordinación vertical para conseguir, entre otros objetivos, que se integren en un solo proyecto, sin solape, los conceptos aprendidos en distintas materias. De forma similar, en [11] se ejercitan las habilidades necesarias para trabajar en grupo mediante proyectos a lo largo de los tres primeros cursos.

Resumiendo, nuestra propuesta destaca sobre los trabajos recientes que acabamos de mencionar en que su finalidad última es triple: i) motivar los contenidos teóricos de

las asignaturas con prácticas atractivas basadas en RPi; ii) renovar los laboratorios y los guiones de prácticas gracias a una plataforma muy asequible y con una arquitectura de referencia en la era Post-PC; y iii) coordinar verticalmente las asignaturas, de 1º a 4º, de forma que el alumno construya cada año un proyecto sobre la base aprendida en el curso anterior.

2 Contexto de la experiencia

El minicomputador Raspberry Pi [1], o RPi, es una placa del tamaño de una tarjeta de crédito y un precio de alrededor de 30€. El objetivo principal de sus creadores, la Fundación Raspberry Pi [12], era promover la enseñanza de conceptos básicos de informática en los colegios e institutos. La plataforma RPi tiene una gran comunidad de usuarios activos, lo que redunda en una gran cantidad de documentación (principalmente en inglés), foros, software libre, aplicaciones, así como ideas de proyectos muy atractivos para los así llamados "makers".

Es patente que, en los alumnos de ingeniería, la plataforma RPi despierta gran interés pues es muy popular en diversos foros tecnológicos. Por ello, en el marco del Proyecto de Innovación Educativa de la UMA PIE13-082, se comenzó una experiencia cuyo objetivo era aprovecharse de ese interés para facilitar el aprendizaje de conceptos y materias impartidas por profesores del Departamento de Arquitectura de Computadores. En el camino, aprovechamos para dar homogeneidad y coordinación vertical entre las prácticas de asignaturas de distintos cursos. Por ejemplo, para el Grado en Ingeniería de Computadores, se hace que un alumno use la RPi para programación en ensamblador en "Tecnología de Computadores" de 1º. A partir de esa base, en 2º se abordan prácticas más avanzadas de Entrada/Salida en la asignatura "Estructura de Computadores". Al año siguiente, en 3º, se vuelve a usar la misma plataforma para ciertas prácticas de las asignaturas de "Diseño de Sistemas Operativos" y "Arquitectura de Computadores". Por último, en 4º, los alumnos implementarán y programarán un sistema paralelo basado en 4 RPi conectadas en red en la asignatura "Arquitecturas Paralelas". Anteriormente, en cada una de estas asignaturas se usaban herramientas y plataformas muy dispares por lo que cada año el alumno se enfrentaba a un reto nuevo y desconectado de los que superó el año anterior.

Por otro lado, aprovechando que la comunidad de desarrolladores y usuarios de RPi es principalmente anglo-parlante, en varias de las asignaturas contempladas en esta experiencia se favoreció el uso del idioma inglés. Actualmente existen infinidad de manuales y tutoriales en Internet para usar la RPi como herramienta docente así que es de sentido común aprovechar ese material y redirigir directamente a los alumnos a esas fuentes cuando sea conveniente. Además, en algunas asignaturas (por ejemplo, en uno de los grupos de "Tecnología de Computadores" y de "Estructura de Computadores") las clases de laboratorio se imparten también en ese mismo idioma.

Por último, sabedores de la importancia del trabajo en grupo en las prácticas de asignaturas de Ingeniería, se propuso el uso de herramientas y aplicaciones web que permiten evaluar la participación de cada alumno dentro del grupo. En casi todas las asignaturas contempladas en esta experiencia, los alumnos deben desarrollar aplica-

ciones software y deben hacerlo de forma colaborativa. Pues bien, herramientas en la web como GitHub o BitBucket simplifican la gestión de versiones y la colaboración durante el desarrollo de aplicaciones. Tanto es así, que muchas empresas que demandan desarrolladores software, en los procesos de selección de personal, solicitan a los candidatos un portafolio con los proyectos alojados en GitHub o BitBucket en los que han participado. Por tanto, usar estas herramientas en la carrera presenta dos ventajas: i) para el alumno, que ganará enteros cuando a la hora de entrar en el mercado laboral se enfrente a un departamento de recursos humanos en un proceso de selección; y ii) para los profesores, ya que estas herramientas, de forma similar a las Wiki's, permiten identificar el grado de colaboración y de esfuerzo que invierte cada alumno al trabajo que se realiza en grupo. Con este último punto, pretendemos que, dentro del grupo de prácticas, la distribución del trabajo esté realmente balanceada, o en caso contrario el profesor lo detectará inmediatamente en la aplicación y tomará medidas para corregirlo.

La Tabla **1** resume las características de las asignaturas implicadas en este trabajo, las cuales pertenecen a las escuelas de Ingeniería Informática, de Ingeniería de Telecomunicación y de Ingeniería Industrial. Para completar este estudio, está previsto incorporar la asignatura de Arquitecturas Paralelas que se imparte en 4ª curso del Grado de Ingeniería de Computadores.

Tabla 1. Asignaturas implicadas con implantación evaluada.

Asignatura	Grado en Ingeniería de	Curso	Num. de alum.
Tecnología de Computadores	Computadores, Software, Informática	1	350
Estructura de Computadores	Computadores, Software, Informática	2	200
Diseño de Sistemas Operativos	Computadores	3	25
Arquitecturas Emergentes	Sistemas de Telecomunicación	4	10
Informática Industrial	Tecnologías Industriales	4	10

3 Descripción de la experiencia

Aunque en las ramas de ingeniería no es extraño encontrar profesores convencidos de que "el alumno debe venir motivado de casa", nosotros creemos por el contrario que es clave que el profesor haga todo lo posible por hacer atractiva su asignatura. Incluso para alumnos de nivel universitario, impregnar las asignaturas de cierto contenido lúdico es muy beneficioso a la hora de conseguir mejores tasas de éxito académico. Pues bien, la RPi es para un estudiante de ingeniería algo parecido a un juguete, aspecto que los profesores no podemos dejar de aprovechar. También el hecho de que la RPi esté basada en el mismo tipo de procesador ARM que incorporan todos los móviles del mercado, hace que esta plataforma sea atractiva a las nuevas generaciones que han crecido con un Smartphone desde edades muy tempranas.

Sin embargo, los contenidos teóricos y prácticos de gran parte de las asignaturas impartidas en el Departamento de Arquitectura de Computadores de la UMA, han girado en los últimos años en torno a otra arquitectura más tradicional: los procesado-

res RISC de la familia MIPS. Esto no sólo es así en nuestro departamento, sino también en la mayoría de departamentos de "Computer Science" de Universidades líderes en el mundo. La razón principal es que el libro de referencia [13] usa un MIPS como caso de estudio. Nuestro argumento es que el procesador MIPS no es familiar ni atractivo para los alumnos. Los móviles no llevan MIPS, ni la RPi, ni la mayoría de dispositivos móviles y de consumo que los alumnos ven como tecnología cercana y de uso cotidiano (televisores, electrónica de consumo, etc). Por el contrario, todos estos dispositivos se basan en procesadores ARM y la plataforma RPi es, desde nuestro punto de vista, ideal para enseñar cómo están diseñados estos procesadores y cómo sacarles el máximo provecho. De hecho, los autores de libros de Arquitectura de Computadores, conscientes de esta convergencia hacia los procesadores ARM están cambiando el foco de los textos educativos, reorientándolos hacia estos nuevos procesadores [14], [15].

El Proyecto de Innovación Educativa PIE13-082 concedido por la UMA, sirvió como detonante para renovar gran parte de las asignaturas y laboratorios de nuestro departamento. Primero permitió la compra de 20 RPi a cargo del presupuesto concedido, y segundo sirvió de acicate para sacar a los profesores implicados, de la inercia de asignaturas asentadas y de larga trayectoria. Hay que reconocer que renovar los temarios teóricos y guiones de prácticas de tantas asignaturas no es fácil, pero, como veremos, la satisfacción de los alumnos que ahora usan la RPi como plataforma común para el desarrollo de las prácticas ha merecido la pena.

Los recursos utilizados por más de 15 profesores fueron principalmente la Sala de Coordinación del PIE en el Campus Virtual de la UMA, así como una carpeta compartida en Google Drive. También contamos con la ayuda de dos alumnos que realizaron un Proyecto Fin de Carrera (PFC) y un Trabajo Fin de Grado (TFG). El PFC tuvo como objetivo portar un antiguo manual de prácticas en ensamblador para PC (basado en procesador 80x86 y con Sistema Operativo MS-DOS) a la arquitectura ARM de la RPi. El antiguo manual abordaba contenidos para las asignaturas de "Tecnología de Computadores" y "Estructura de Computadores" del plan a extinguir de Ingeniería Informática. Sin embargo, estas prácticas quedaron en desuso debido a las dificultades técnicas para mantener MS-DOS en los PCs de los laboratorios y a que no son factibles en un SO con permisos de usuario, como Windows XP o Windows 7. Portar las prácticas a la RPi eliminaba los problemas de seguridad y administración de los PCs de laboratorio, al tiempo que actualizaba los contenidos para enseñar la arquitectura ARM. Como parte de este PFC se desarrolló una placa de expansión que se puede conectar a la RPi para visualizar el estado de los pines del GPIO (General Purpose Input/Output) mediante LEDs, provocar interrupciones mediante pulsadores o generar sonido en un zumbador (ver Fig. 1). La memoria de este PFC fue posteriormente completada y extendida ligeramente para convertirse en un manual [16]. Este manual se puso a disposición de la comunidad gracias a la plataforma RIUMA de la UMA, manual que algunos profesores de la Universidad Pablo de Olavide están usando en una asignatura de "Fundamentos de Computadores".

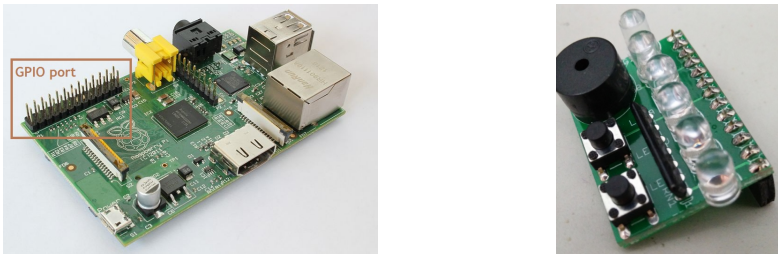


Fig. 1. Placa RPi con puerto GPIO (izq.). Placa de expansión a conectar en GPIO (der.).

La primera mitad del manual está orientada a facilitar el aprendizaje del lenguaje ensamblador para ARM y encaja con las prácticas de la asignatura de "Tecnología de Computadores". La segunda mitad, aborda aspectos de control de la Entrada/Salida (E/S) de la RPi, como son el GPIO y las interrupciones y está más orientada a los contenidos que se imparten en "Estructura de Computadores".

En cuanto al TFG, se desarrolló un planificador de procesos con política Round Robin para RPi, lo que deja abonado el proceso de elaboración de prácticas de laboratorio adicionales para las asignatura de "Diseño de Sistemas Operativos".

Una vez comprobado con este trabajo previo la viabilidad de la RPi como plataforma común de prácticas para un gran número de asignaturas, a lo largo de 2015 y 2016 se compraron más RPi. En la actualidad el departamento cuenta con 172 unidades en total, en parte gracias al reducido precio de este computador.

Con estos precedentes y el material necesario para implantar la RPi como plataforma de prácticas en la mayoría de las asignaturas que imparte el departamento de Arquitectura de Computadores, se han desarrollado un gran número de prácticas y de material docente para las asignaturas recogidas en la Tabla 1, y que describimos en la siguiente sección.

4 Asignaturas implicadas

4.1 Tecnología de Computadores, TC (1º)

Tecnología de Computadores es una asignatura obligatoria que se imparte en primer curso de las titulaciones de Grado en Ingeniería Informática, Grado en Ingeniería del Software y Grado en Ingeniería de Computadores. Existen 6 grupos en total de esta asignatura, uno de ellos impartido en inglés. En esta asignatura, partiendo de los conocimientos básicos de electrónica digital adquiridos por el alumno en asignaturas del primer semestre del primer curso, se sientan las bases de la formación en el área de Arquitectura y Tecnología de Computadores, impartiendo los conceptos fundamentales para la comprensión del funcionamiento y diseño de un procesador básico, así como su programación al más bajo nivel. De entre las competencias generales contempladas en esta asignatura, encontramos una única competencia específica que establece como objetivo el "conocimiento de la estructura, organización, funcionamiento e interconexión de los sistemas informáticos y los fundamentos de su programación". De entre los resultados de aprendizaje enfocados a conseguir esta compe-

tencia, el relacionado con el uso de la plataforma RPi en esta asignatura es: "el alumno deberá ser capaz de realizar programas utilizando el lenguaje ensamblador de una máquina específica".

Metodología.

Para que el alumno alcance los resultados de aprendizaje especificados, éste debe desarrollar una serie de ejercicios prácticos de programación en ensamblador.

Históricamente la programación en ensamblador se realizaba utilizando el repertorio de instrucciones de un procesador tipo MIPS, que es el procesador utilizado en el libro de referencia por excelencia de estos cursos [13]. Para ello se utilizaba un software que simulaba dicho procesador sobre un PC con Windows. Esto provocaba que el alumno viera de una manera mucho más abstracta y alejada de la realidad este tipo de prácticas.

Con el objetivo de intentar aumentar la motivación del alumno a la hora de realizar las prácticas de programación en ensamblador, en el curso 2014-2015 se impartió un taller de programación en ensamblador sobre la plataforma hardware RPi. Las prácticas se desarrollaron sobre un hardware real y no sobre un simulador, de manera que los alumnos interactuaron directamente con un dispositivo con el que se puede generar sonidos, encender LEDs o accionar pulsadores. Esta interactividad con un sistema real resultó ciertamente incentivadora.

Una vez terminado el taller se sondeó la opinión del alumnado sobre la realización de las prácticas en un dispositivo real (en vez de sobre un simulador). La respuesta del alumnado fue bastante buena, como se puede apreciar en los resultados que mostramos en la sección 5.

En el curso académico 2015-2016, el equipo docente de la asignatura decidió incorporar de forma obligatoria, un par de prácticas de programación en ensamblador sobre la plataforma RPi, sin abandonar completamente las prácticas de programación basada en un simulador de procesador MIPS. Se les proporcionó nueve vídeos donde se mostraba de forma visual y sonora el resultado que debían conseguir para cada práctica. Además, se le dejaba libertad al alumno para que eligiera al menos dos prácticas de las nueve presentadas. La dificultad de las distintas prácticas no era la misma, por lo que la puntuación asignada a cada una era relativa a dicha dificultad.

Muchos de los alumnos realizaron bastantes más prácticas de las exigidas, e incluso hicieron variaciones sobre las que se les proponían. Entendemos que esto es indicativo del alto grado de motivación que se alcanzó en buena parte del alumnado.

Tras la finalización de las prácticas con las RPi, de nuevo se les pasó a los alumnos una encuesta para que opinaran sobre las mismas, teniendo en cuenta que también habían realizado prácticas en la plataforma antigua. De nuevo el grado de satisfacción de los alumnos con estas prácticas frente a las de la plataforma antigua es muy alto.

Una vez analizadas las ventajas de la nueva plataforma para realizar las prácticas de programación en ensamblador de la asignatura, el equipo docente de la asignatura de Tecnología de Computadores decidió cambiar las prácticas de programación en ensamblador del curso académico 2016-2017 y pasar a realizarlas completamente con la RPi. El tipo de prácticas es muy similar al realizado con la plataforma MIPS, pero cambiando el enfoque al uso de elementos hardware disponibles en la nueva plata-

forma (LEDs, altavoz y pulsadores), teniendo cuidado para que la complejidad de las prácticas sea básicamente la misma. Aunque entendemos que una de las desventajas del uso de un dispositivo físico frente a un simulador a la hora de realizar las prácticas es la posibilidad de realizarlas fuera del horario de laboratorio asignado a cada grupo, también pensamos que este problema no es muy grave al ser una plataforma bastante asequible para el alumno, además de que nos aseguramos que el tiempo de laboratorio asignado para la realización de las prácticas es suficiente para que el alumno las termine.

4.2 Estructura de Computadores, EC (2º)

Estructura de Computadores es una asignatura obligatoria que se imparte en segundo curso de las titulaciones de Grado en Ingeniería Informática, Grado en Ingeniería del Software y Grado en Ingeniería de Computadores. Existen 5 grupos en total de esta asignatura, uno de ellos impartido en inglés
Esta asignatura parte de los conocimientos básicos impartidos en la asignatura de Tecnología de Computadores, de primer curso, que se ha mencionado anteriormente. En Estructura de Computadores se profundiza en el estudio del subsistema de Entrada/Salida, cuyo diseño y programación se ilustra a través de las prácticas con RPi que se proponen. Tras la realización de las prácticas los alumnos serán capaces de entender el proceso de arranque básico de un procesador, entender los distintos métodos de comunicación de los periféricos con un procesador, así como programar rutinas de tratamiento de interrupción o manejadores en "bare-metal" (sin intervención del SO). Los conocimientos y competencias adquiridos en esta asignatura serán fundamentales para las asignaturas de "Sistemas Operativos" y "Arquitectura de Computadores" que forman parte de la titulación.

Metodología.

Las prácticas realizadas en EC se organizan en dos tipos de sesiones: 6 sesiones de prácticas guiadas y 6 sesiones de práctica libre. En la primera sesión de prácticas guiadas se les explica el entorno de desarrollo, el sistema de arranque del procesador ARM de la RPi y cómo se compila y carga un programa en la placa para trabajar en modo bare-metal. En las siguientes sesiones, a través de pequeños ejercicios prácticos guiados, se les ilustra cómo funcionan y se programan los distintos métodos de Entrada/Salida en la RPi y cómo interactuar con los LEDs, el altavoz y los pulsadores de la placa de expansión conectada a la RPi en bare-metal.

Tras las sesiones de prácticas guiadas, se propone al estudiante la realización de una práctica final que consiste en la reproducción de una melodía a través del altavoz, acompañada de una función de ecualización (visible a través de los LEDs de la placa de extensión) así como la manipulación de la melodía y ecualizador a través de los pulsadores. Para el desarrollo de esta práctica final los estudiantes disponen de 6 sesiones de práctica libre, durante las cuáles pueden trabajar en parejas. Los estudiantes afrontan esta fase con bastante entusiasmo, porque la práctica es cómo un juego y pueden colaborar entre ellos para ayudarse en el desarrollo y depuración de cada una

de las funcionalidades de la práctica final. Al finalizar estas sesiones de práctica libre, cada estudiante entrega su práctica final y concierta una entrevista con el profesor/a, que evalúa el nivel de adquisición de los resultados de aprendizaje previstos con el desarrollo de estas prácticas.

4.3 Diseño de Sistemas Operativos, DSO (3º)

Esta es una asignatura obligatoria de tercer curso del Grado de Ingeniería de Computadores (un solo grupo). Los alumnos de 3º vuelven a encontrarse con la RPi pero ahora como plataforma para trabajar a bajo nivel dentro del SO Linux (Raspbian). En esta asignatura se pretende profundizar en el diseño de los sistemas operativos, para los que ya se han introducido los conceptos fundamentales en la asignatura de "Estructura de Computadores". Desde un enfoque práctico se estudia cómo se diseñan e implementan los mecanismos y políticas que gestionan, administran y facilitan el uso de los recursos hardware de un computador (procesador, memoria, Entrada/Salida, sistemas de fichero, etc.), función principal del SO.

De entre las competencias específicas que debe adquirir el alumno se encuentran las de "capacidad de diseñar y construir sistemas digitales, incluyendo computadores, sistemas basados en microprocesador y sistemas de comunicaciones" y la "capacidad de diseñar e implementar software de sistema y de comunicaciones". En ambos casos, la plataforma RPi se ha mostrado como una herramienta valiosa para ejercitar y adquirir estas capacidades. Al usarla a lo largo de toda la asignatura, se convierte en clave para obtener los resultados de aprendizaje planteados en la asignatura.

Metodología.

Se realizaron varias prácticas y proyectos que supusieron la mayor parte de la asignatura. Cada grupo de alumnos fue responsable de su placa de desarrollo (RPi) y pudo, de forma autónoma, recomponerla después de un resultado catastrófico, realizar la depuración o análisis post-mortem necesarios y reconducir su diseño para resolver las contingencias. La facilidad de reinstalación del sistema, simplemente volviendo a escribir el SO en una tarjeta de memoria y restaurando su trabajo en curso desde un servidor de control de versiones Git, permitió que en unos pocos minutos el grupo estuviera de nuevo probando las mejoras y soluciones diseñadas. De esta forma, gracias a la RPi, el alumno no tiene miedo a equivocarse en el diseño, o errar en la implementación, hasta el punto de dejar el sistema inoperativo (irónico, si recordamos el título de la asignatura), ya que la recuperación es indolora. En un PC del laboratorio los alumnos no pueden hacer este tipo de procedimientos delicados que requieren acceso total al sistema (privilegios de administración), puesto que hay muchas posibilidades de dejar el sistema inutilizable con los consiguientes costes asociados a volver a configurar el PC.

Los alumnos más aventajados también tuvieron tiempo de desarrollar un driver para la lectura del sensor de temperatura y humedad DHT11, un reto motivador, puesto que este dispositivo no estaba soportado por el sistema Linux en la RPi. Así que se convirtió en una contribución real de nuestros estudiantes a la comunidad de software

libre. Este es un desafío algo más importante, porque, aunque el protocolo de comunicación serie con el sensor no es demasiado complejo, debe realizarse de forma eficiente y segura para conseguir lecturas válidas de forma estable.

También destacamos que en esta asignatura es donde por ahora ha tenido mayor sentido usar herramientas de control de versiones.. En particular, se creó una cuenta en BitBucket gestionada por el profesor, que usaron los alumnos durante las prácticas para descargar los materiales, ejemplos y proyectos de partida. A su vez, los alumnos crearon sus propias cuentas en este servidor donde fueron alojando sus proyectos. Cada uno de los proyectos desarrollados tenía como contribuidores a los alumnos que formaban el grupo de trabajo, pero también incluían al profesor, de forma que se podía seguir el desarrollo de los proyectos y comprobar el nivel de implicación y actividad de cada componente del grupo de trabajo.

4.4 Arquitecturas Emergentes, AE (4º)

Esta es una asignatura optativa de cuarto curso del Grado de Sistemas de Telecomunicación. En esta asignatura se imparten los conceptos relacionados con arquitecturas avanzadas y emergentes, principalmente relacionadas con las arquitecturas usadas en dispositivos móviles (teléfonos, tablets), sistemas multicore, multiprocesador, procesadores gráficos y arquitecturas heterogéneas. De entre las competencias que debe adquirir el alumno se encuentra la de "entender y explotar el hardware de los computadores y dispositivos móviles".

Metodología.

La RPi se usó en 2 de las 5 prácticas de la asignatura. Estas dos prácticas se preceden de una explicación de tipo magistral sobre los modos de funcionamiento de la RPi con SO y en bare-metal.

La primera consiste en implementar en ensamblador de ARM algunos códigos en C de procesado de arrays, así como saber enlazar código C y código ensamblador, donde este último contenga una versión optimizada a bajo nivel. En dos horas de laboratorio, los alumnos aprenden a conectarse a una RPi con SO Raspbian y a editar, compilar y ejecutar programas ensamblador sobre esa plataforma. Se continúa con un tutorial guiado en el que se enseña cómo hacer que se enciendan/apaguen los leds de la placa de expansión, leer el estado de los botones o generar un sonido, todo ello con soporte de la librería WiringPi. Por último, se les asigna un problema para resolver en el laboratorio (escribir un código que encuentra el valor máximo de un vector) y los ejercicios de procesado de vectores y matrices para resolver de forma no presencial.

La segunda práctica también aborda la programación en ensamblador, pero ahora en modo bare-metal. Esta práctica consume 3 horas de laboratorio y comienza con un tutorial guiado en el que se muestra cómo generar un ejecutable bare-metal mediante compilación cruzada, cuál es el proceso de arranque de la Raspberry Pi y cómo se controlan los leds, el temporizador y el zumbador de la RPi cuando no disponemos del soporte del SO ni de la librería WiringPi. En tiempo de laboratorio los alumnos tienen que resolver un ejercicio en el que deben conseguir que la RPi genere un soni-

do a 440Hz (nota "La" de la escala musical). Como problema a resolver de forma autónoma, no presencial, a los alumnos se les pide implementar un programa ensamblador que reproduce por el altavoz de la placa de expansión el tema "Marcha Imperial" de la BSO de "Star Wars", al tiempo que los LEDs siguen el ritmo de la música.

4.5 Informática Industrial, II (4º)

Esta asignatura es optativa en cuarto curso del Grado de Ingeniería de Organización Industrial. La asignatura aborda la Informática Industrial y las Comunicaciones Industriales desde un punto de vista aplicado.

La competencia específica fundamental de esta asignatura es la de "Adquirir conocimientos aplicados de informática industrial y comunicaciones" y para ello se usa como elemento central la RPi. Esta plataforma nos permite una interconexión fácil con sensores y actuadores que imitarán a un entorno industrial.

Metodología.

Toda la parte práctica de esta asignatura (50% de las horas) se ha impartido utilizando la RPi. Se usó la RPi como plataforma de programación en ejercicios de control y automatización, en los que los alumnos jugaron conectando sensores de temperatura, módulos de radio y de control domótico.

Mediante los GPIOs (entradas/salidas de propósito general) que equipa la RPi y que soportan también diferentes buses de comunicación, se conectaron desde sensores de temperatura hasta módulos de radio de bajo consumo. El uso de la placa RPi permitió la programación e interacción de los alumnos con un sistema completo de control, realizando prácticas de conexión en red y control distribuido.

5 Evaluación y resultados

En esta sección presentamos los métodos de evaluación y sus resultados para cada una de las asignaturas presentadas en la sección anterior.

5.1 Tecnología de Computadores

La evaluación de la experiencia en TC se realizó a base de encuestas que contestaron los alumnos que realizaron el taller en 2014-2015 y los alumnos que tuvieron que hacer las dos prácticas obligatorias en 2015-2016. De los alumnos que realizaron el taller y contestaron la encuesta, tan solo un 5% poseían ya una Raspberry, pero el 81% contestó que se plantearía la compra de una para realizar las prácticas en casa. Esto se ve reflejado en la encuesta del curso 2015-2016 donde el porcentaje de alumnos que poseían una Raspberry ascendió al 28%, y de nuevo el 82% considerarían la posibilidad de adquirir una. Consideramos que este aumento puede ser debido, en parte, a que el alumno ya está percibiendo que varias de las asignaturas de su currículum van a utilizar esta plataforma como base para realizar las prácticas (cerca del 90% de los alumnos consideraban muy interesante que se utilice la misma plataforma en

distintas asignaturas). Además, se le suma su bajo precio y su versatilidad lo que anima a la compra por parte del alumno. Otro dato a destacar es que a más del 94% de los alumnos les parecieron muy interesantes estas prácticas, cuestión que ha sido una de las que hemos tenido en cuenta a la hora de decidir cambiar todas las prácticas en el curso 2016-2017 y basarlas en la RPi. Muchos de los comentarios en un campo de texto libre de las encuestas van en el sentido de que a los alumnos les hubiera gustado tener más prácticas con la RPi.

5.2 Estructura de Computadores

Se realizó una encuesta de satisfacción de las prácticas basadas en RPi entre los estudiantes de todos los grupos de esta asignatura, EC, al final del primer semestre del curso 2015-2016. Uno de los grupos actuó como grupo de control: en este caso, estos alumnos realizaron las prácticas de programación utilizando una plataforma de desarrollo basada en una FPGA en la que se implementa el MIPS, aunque vieron algunos ejercicios adicionales usando RPi o asistieron a sesiones de laboratorio de otros grupos que utilizaban RPi y pudieron programar en ensamblador de ARM. En total contestaron la encuesta 71 estudiantes, mientras que en el grupo de control recogimos 21 respuestas.

De los 71 estudiantes que realizaron las prácticas en la plataforma RPi y respondieron a la encuesta, el 86% consideró las prácticas como interesantes o muy interesantes. De hecho, un 75% cree que con estas prácticas se ilustra bien o muy bien los conceptos de Entrada/Salida que se estudian en esta parte de la asignatura y que son uno de los objetivos fundamentales de estas prácticas. Un 47% encontró fácil o muy fácil adaptarse al nuevo lenguaje de programación basado en la arquitectura ARM, que es el que se requiere en la RPi. Un 80% también valora muy positivamente que esta plataforma se utilice en las prácticas de otras asignaturas del Grado. Un 33% ya tienen una RPi, y de los que no la tienen un 76% estaría dispuestos a comprársela si se utiliza en otros laboratorios del Grado. En general los alumnos valoraron que se tratara de un sistema real, lo que les permite una inmediata transferencia de conocimiento al mundo profesional. También comentaron que les pareció poco el tiempo dedicado a las prácticas, especialmente el número de horas dedicadas a las sesiones de práctica libre, aspecto que se corrigió en el curso 2016-2017.

En cuanto a los 21 alumnos que participaron en el grupo de control, se obtuvieron porcentajes similares respecto a la percepción del interés de las prácticas y de cómo se ilustran los conceptos que son objetivo de la asignatura. Curiosamente en este grupo, los alumnos parecen más dispuestos a aprender otros lenguajes de programación, no sólo el que necesitan para programar la RPi, sino también otro tipo de plataformas. Recordemos que estos alumnos estuvieron expuestos a varios lenguajes de programación (MIPS y ARM) por lo el esfuerzo fue algo mayor, pero a cambio fueron conscientes del valor añadido que supuso realizar prácticas en varias plataformas y lo valoraron muy positivamente.

5.3 Diseño de Sistemas Operativos

En la encuesta realizada a los nueve alumnos matriculados en DSO el primer año en que se usó la RPi (curso 2014-2015), el 100% valoró el uso de la plataforma como muy interesante y adecuada. Además, todos ellos opinaron que se debería usar de forma continuada en el resto de asignaturas del área. Muchos de ellos ya tenían o han adquirido una RPi, al final del curso, todos menos uno, poseían una tarjeta propia y la habían usado para poder profundizar en sus desarrollos prácticos fuera del horario de laboratorio. La información sobre la RPi en Internet (principalmente en inglés) permite a los alumnos documentarse y resolver los problemas de diseño y desarrollo planteados.

5.4 Arquitecturas Emergentes

La evaluación de la experiencia en AE, se realizó en base a una encuesta a la que contestaron los 5 alumnos matriculados en el curso 2014-2015 (primer año de impartición de la asignatura). Solo uno de los alumnos tenía una RPi en propiedad, pero todos consideraron las prácticas interesantes y formativas. En el campo de texto libre, un alumno comentó: “creo que se le podría dar mucho juego a la Raspberry si ésta se usara desde otras asignaturas, pues se podría ir añadiendo ladrillos en cada asignatura y finalmente conseguir algo más "tecnológico" que hacer sonar una musiquita.”. Este comentario nos lleva a reflexionar si los alumnos de 4º realmente necesitan la motivación/juego con luces y sonidos o ya prefieren actividades más técnicas y profesionales, aunque sean más áridas.

5.5 Informática Industrial

Se valoró la introducción de la plataforma RPi mediante una encuesta realizada a los 5 alumnos matriculados en el primer año de impartición de la asignatura (curso 2014-2015). El 100% valoró el uso de la plataforma como muy interesante y adecuada. En este caso, tan sólo uno de ellos ya tenía una RPi propia. El perfil de alumno en este Grado, está un poco más alejado del área de las tecnologías de la información y comunicaciones que son los pertenecientes a los Grados de Informática o Telecomunicación. Sin embargo, aceptaron muy positivamente el uso de esta plataforma, que vieron como más atractiva que el clásico PC con el que hasta ahora estaban acostumbrados a trabajar. Los ejercicios de programación y desarrollo en lenguaje C sobre el entorno Linux (Raspbian) no les resultaron triviales y requirieron de un tiempo de aprendizaje y esfuerzo extra. Sin embargo, el hecho de “tocar” e interactuar con la tecnología, casi jugando, les motivó a seguir intentándolo, mejorando cada vez más sus destrezas y capacidad de manejarse con la programación.

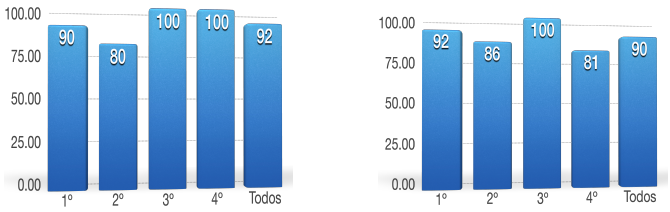
El hecho de trabajar e interactuar con los sensores físicos y realizar el conexionado/cableado de los sensores al interfaz GPIO de la RPi da un valor añadido a la experiencia que la hace muy realista. En los ejercicios prácticos parece que el proceso de creación es más completo cuando además de programar la parte software o inteligencia del sistema, también pueden intervenir físicamente en el diseño y montaje del

hardware o aparataje necesario. Desde luego, la placa de desarrollo RPi se ha revelado como una opción muy asequible para equipar un laboratorio de programación, control y automatización. Hasta ahora se utilizaban placas de desarrollo equipadas con microcontrolador y diferentes actuadores/sensores en el laboratorio de esta asignatura, pero ahora se combinan con la RPi que ofrece un SO completo y mayor flexibilidad y potencia, de forma que la RPi se convierte en el supervisor de la placa con microcontrolador, dándole además conectividad de red y así poder realizar ejercicios aplicados de IoT (Internet of Things).

5.6 Resultados Globales

Aunque se ha hecho una evaluación pormenorizada de cada una de las asignaturas evaluadas en la experiencia, dedicamos este apartado a dar una visión global sobre los resultados de las encuestas de todas ellas. La satisfacción del alumnado que asistió a los laboratorios de estas asignaturas mencionadas se evaluó mediante 9 encuestas anónimas implementadas mediante formularios Google. En total, respondieron 212 alumnos que realizaron las prácticas con RPi. De los resultados de dichas encuestas se desprende claramente que los alumnos disfrutaron con las prácticas propuestas ya que al 89% de los encuestados las prácticas les parecieron muy interesantes.

La Fig. 2 muestra las estadísticas de dos de las preguntas de dichas encuestas, distinguiendo a los alumnos por curso (de 1º a 4º), e incluyendo la última columna con la media para todos los alumnos de todos los cursos. Las gráficas muestran el porcentaje de alumnos que ha respondido estar "De acuerdo" o "Totalmente de acuerdo" a las preguntas: (a) ¿Te gustaría usar la RPi en otras asignaturas de 1º a 4º?; y (b) ¿Te han parecido motivadoras las prácticas con RPi? El rango de las respuestas contempla 5 opciones ("Total desacuerdo", "En desacuerdo", "Neutral", "De acuerdo" y "Totalmente de acuerdo").



(a) (b)

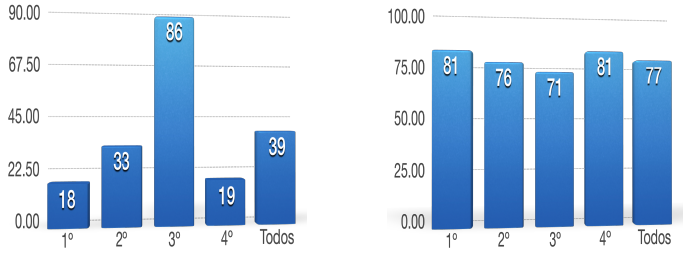
Fig. 2. Valoración de los alumnos a las preguntas: (a) ¿Te gustaría usar la RPi en asignaturas de todos los cursos?; y (b) ¿Han sido las prácticas motivadoras?

De estas gráficas se desprende que siempre para más del 80% de los alumnos (sobre el 90% de media) las prácticas resultaron motivadoras y se considera que la RPi se debería usar en otras asignaturas de la carrera, dando una continuidad a la plataforma y cierto grado de coordinación vertical.

En la Fig. 3 se muestra el resultado de preguntar por el grado de penetración de la RPi entre los alumnos: qué porcentaje de alumnos ya tienen una RPi en propiedad y,

de los que no, cuántos estarían "De acuerdo" o "Totalmente de acuerdo" en comprar una.

En todos los cursos más del 15% de los alumnos ya tiene una RPi, incluso entre los alumnos de primer curso. Destaca el grupo de DSO (3º), donde el 86% de los alumnos cuentan con una RPi, lo que eleva la media de alumnos que pueden continuar con las prácticas fuera del horario de laboratorio al 39%. No sorprende que dado el bajo coste de la RPi y la gran cantidad de aplicaciones que se le puede dar (fuera del ámbito académico) que el 77% de los alumnos se planteara comprar una RPi si se usara en las asignaturas de la carrera.



(a) (b)

Fig. 3. Aceptación de la RPi: (a) % alumnos que ya tienen una RPi en propiedad y (b) % alumnos que en caso contrario se la comprarían.

Para la asignatura EC se contó con 21 alumnos de control que cursaron las prácticas según el temario de años anteriores usando MIPS en lugar de ARM. De este grupo destacamos que, aunque las prácticas con MIPS eran las obligatorias, el 70% de los alumnos decidieron voluntariamente hacer también las prácticas con RPi (en algunos casos asistiendo a los laboratorios de otros grupos). El 84% de este grupo de control también estaba "De acuerdo" o "Totalmente de acuerdo" con que la RPi se usara en el mayor número posible de asignaturas de la carrera.

6 Conclusiones

Queremos alumnos motivados en clase y los resultados de este trabajo confirman que la plataforma RPi es un elemento que mejora dicha motivación. Con este objetivo, hemos abordado la renovación de una buena cantidad de prácticas de laboratorio que habían quedado obsoletas y faltas de atractivo para el alumnado. Además, con esta experiencia se ha conseguido actualizar varias de las asignaturas impartidas por el departamento de Arquitectura de Computadores y convencer a los profesores de las bondades de la plataforma RPi a la hora de poner en práctica los contenidos teóricos impartidos en las asignaturas de este departamento. Aunque en las ramas de Ingeniería de Telecomunicación y de Ingeniería Industrial, tenemos menos presencia, esperamos que nuestra experiencia con la RPi sea contagiosa a otros departamentos y áreas.

Agradecimientos

El presente trabajo ha sido financiado mediante el proyecto de innovación educativa PIE13-082 de la Universidad de Málaga. Agradecemos a Mª Antonia Trenas ("Estructura de Computadores"), José Mª González ("Arquitectura de Computadores") y a Javier Hormigo ("Informática Industrial") su trabajo para incorporar, en mayor o menor medida, la RPi como plataforma de prácticas en sus asignaturas. También agradecemos la colaboración de los alumnos que han participado en la experiencia contestando a las encuestas y comentando constructivamente sobre la misma.

References

1. E. Upton and G. Halfacree, Raspberry Pi user guide, 3rd edition. Wiley, 2014.
2. J. Andrus and J. Nieh, "Teaching Operating Systems Using Android," in Proceedings of the 43rd ACM Technical Symposium on Computer Science Education, 2012, pp. 613–618.
3. "IDC: Smartphone OS Market Share 2016, 2015." [Online]. Available: http://www.idc.com/promo/smartphone-market-share/os.
4. G. Ortega et al., "Procesadores de bajo coste y su aplicación en la docencia de Ingeniería de Computadores," in Actas de las XXII Jenui, 2016, no. 2, pp. 343–349.
5. J. Kawash, A. Kuipers, L. Manzara, and R. Collier, "Undergraduate assembly language instruction sweetened with the raspberry pi," in SIGCSE 2016 - Proceedings of the 47th ACM Technical Symposium on Computing Science Education, 2016.
6. Y.-H. Lu, G. Zhu, and C.-K. Koh, "Using the Tetris game to teach computing," in ASEE Annual Conference and Exposition, Conference Proceedings, 2010.
7. A. Esakia, S. Niu, and D. S. McCrickard, "Augmenting Undergraduate Computer Science Education With Programmable Smartwatches," in Proceedings of the 46th ACM Technical Symposium on Computer Science Education, 2015, pp. 66–71.
8. A. García Martín, Coordinación docente horizontal y vertical. Universidad Politécnica de Cartagena, Servicio de Documentación, 2015.
9. L. Torrego and C. Ruiz, "La coordinación docente en la implantación de los títulos de Grado," Rev. Electrónica Interuniv. Form. Del Profr., vol. 14, no. 4, pp. 31–40, 2011.
10. E. Manchado Pérez and I. López Forniés, "Coordinación por módulos de asignaturas en el Grado de Ingeniería de Diseño Industrial y Desarrollo de Producto de la Universidad de Zaragoza," REDU Rev. Docencia Univ. ISSN 1887-4592, Vol. 10, No. 3, 2012 (Ejemplar Dedic. a Innovaciones en el diseño Curric. los Planes Estud., vol. 10, no. 3, p. 195, 2012.
11. A. Hurst et al., "Towards a Multidisciplinary Teamwork Training Series for Undergraduate Engineering Students: Development and Assessment of Two First-year Workshops," in 2016 ASEE Annual Conference & Exposition Proceedings, 2016.
12. "Raspberry Pi - Teach, Learn, and Make with Raspberry Pi." [Online]. Available: https://www.raspberrypi.org/.
13. D. A. Patterson and J. L. Hennessy, Computer Organization and Design, 5th ed. Elsevier, 2014.
14. D. A. Patterson and J. L. Hennessy, Computer Organization and Design: The Hardware Software Interface: ARM Edition, 1st ed. Elsevier, 2016.
15. S. L. Harris and D. M. Harris, Digital design and computer architecture.
16. A. J. Villena Godoy, R. Asenjo and F. Corbera, "Prácticas de Ensamblador Basadas en Raspberry Pi.", Repositorio Institucional de la Universidad de Málaga, RiUMA, 2016. http://hdl.handle.net/10630/10214

Uso de la infraestructura docente MIPSfpga v2.0 en la asignatura *Arquitectura de Sistemas Integrados*⋆

Daniel Chaver[1], Yuri Panchul[2], Enrique Sedano[2], David M. Harris[3], Robert Owen[2], Zubair L. Kakakhel[2], Bruce Ableidinger[2], Sarah L. Harris[4]

(1) Grupo ArTeCS, Universidad Complutense de Madrid
(2) Imagination Technologies Ltd., Kings Langley, United Kingdom
(3) Harvey Mudd College, Claremont, CA, USA
(4) Electrical and Computer Engineering, University of Nevada, USA

Resumen: En este artículo se describe el uso de la infraestructura docente MIPSfpga v2.0 en las prácticas de la asignatura *Arquitectura de Sistemas Integrados*, una asignatura obligatoria en el *Grado en Ingeniería Electrónica de Comunicaciones* que se imparte en la Universidad Complutense de Madrid.

Palabras Clave: MIPS, FPGA, Arquitectura de Computadores, Infraestructura Docente

Abstract: In this paper we describe the use of the MIPSfpga v2.0 teaching infrastructure for the labs included in the course *Integrated Systems Architecture*, a compulsory subject in the fourth year of the *Electronic Engineering of Communications* degree offered at University Complutense of Madrid.

Keywords: MIPS, FPGA, Computer Architecture, Teaching Infrastructure

1 Introducción

En Junio de 2015, Imagination Technologies publicó la primera versión (v1.0) del proyecto MIPSfpga [1, 2], que consta de 3 paquetes (MIPSfpga GSG, MIPSfpga Fundamentals y MIPSfpga SoC), y que incluye como componente central el procesador soft-core microAptiv de MIPS en código abierto. El primero de los paquetes, denominado MIPSfpga Getting Started Guide (GSG), incluye, además del propio procesador soft-core, otros componentes, como una extensa guía de

⋆ Este paper se basa en un artículo anterior titulado "Practical experiences based on MIPSfpga", publicado en el Workshop on Computer Architecture Education (celebrado en la conferencia ISCA-2017). Incluye algunas modificaciones: (1) Hemos ampliado la Sección II-D (que en este paper corresponde a la Sección 2.4) y la Sección III-A (que corresponde a la Sección 3); (2) Hemos eliminado las Secciones III-B y III-C; (3) Hemos traducido el paper al español.

uso, multitud de documentación, y todas las herramientas sotware necesarias para utilizar la infraestructura. En cuanto al paquete MIPSfpga Fundamentals, incluye 9 prácticas muy completas con sus correspondientes soluciones, en las que se explica cómo configurar el hardware, cómo desarrollar y depurar programas en C y ensamblador de MIPS, o cómo extender el hardware para interactuar con una serie de periféricos. Por último, en el paquete MIPSfpga SoC, se implementa un System on Chip en torno al procesador microAptiv y se carga y ejecuta una versión reducida de Linux.

En Julio de 2017, Imagination Technologies publicó la segunda versión (v2.0) de esta infraestructura, ampliando notablemente MIPSfpga v1.0. La nueva versión se encuentra disponible a través de [3] y se describe en gran profundidad en [4] y [5]. En concreto, el paquete MIPSfpga GSG ha sido extendido con nuevas funcionalidades, entre las que destacan las siguientes: posibilidad de utilizar la placa FPGA prescindiendo del depurador BusBlaster, soporte para el Sistema Operativo Linux, o disponibilidad de los módulos de alto nivel en lenguaje VHDL. En cuanto al segundo de los paquetes, que es el que ha experimentado mayores modificaciones y que en la v2.0 se pasa a denominar MIPSfpga Labs, incluye 17 nuevas prácticas, en las que se examina la ruta de datos del procesador, la jerarquía de memoria, la modificación del core en diversas formas, o el uso de Entrada/Salida basada en interrupciones.

En este artículo, describimos en primer lugar (Sección 2) cada uno de los paquetes que componen MIPSfpga v2.0, y analizamos a continuación un caso de uso de esta infraestructura (Sección 3). Por último, comparamos MIPSfpga con otras alternativas docentes (Sección 4) y extraemos las principales conclusiones de este trabajo (Sección 5).

2 Análisis de MIPSfpga v2.0

Como hemos mencinado en la Introducción, MIPSfpga v2.0 consta de tres paquetes: MIPSfpga GSG, MIPSfpga Labs y MIPSfpga SoC. En esta sección explicaremos brevemente el contenido de cada uno de ellos (en [4] se incluye una descripción mucho más detallada) y analizaremos cómo de bien se adapta esta infraestructura docente a las *IEEE/ACM Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering* [6].

2.1 MIPSfpga GSG

El paquete MIPSfpga Getting Started Guide (GSG) nos proporciona acceso a un soft-core comercial de MIPS para su uso en una FPGA. Además, el paquete incluye los instaladores para las herramientas de programación y depuración, una guía detallada sobre el uso de la infraestructura MIPSfpga, y un conjunto de scripts y ejemplos prácticos.

El hardware necesario para utilizar MIPSfpga es una placa FPGA y un depurador Bus Blaster. Las placas que se utilizan como ejemplo en la guía de usuario son la Nexys4 DDR de Digilent y la DE2-115 de Terasic, pero se proporciona

también información detallada sobre cómo portar el sistema a otras placas más pequeñas y asequibles (como una Basys3 de Digilent o una DE0 de Altera). En cuanto al software, todo él gratuito, se necesita instalar una herramienta CAD (Vivado si utilizamos una FPGA de Xilinx o Quartus II si utilizamos una FPGA de Altera), y herramientas de programación y depuración (Imagination Codescape MIPS SDK Essential y OpenOCD respectivamente). Estas herramientas se pueden ejecutar tanto sobre sistemas operativos Windows como Linux.

El soft-core incluido en MIPSfpga es una versión ligeramente reducida del procesador microAptiv-UP, utilizado en el conocido microcontrolador PIC32MZ de Microchip, e implementa la Instruction Set Architecture (ISA) MIPS32r3 en un pipeline de 5 etapas. El core (Figura 1) incluye una Memory Management Unit (MMU) con un Translation Lookaside Buffer (TLB), caches separadas de instrucciones y datos, y diversos interfaces. En el Datasheet del core [7] se pueden ecntrar el resto de especificaciones del mismo.

La Figura 2 muestra el sistema MIPSfpga completo, que incluye el propio soft-core, los periféricos, y el bus AHB-Lite de comunicación entre ambos. Entre los periféricos se incluye la memoria principal, implementada en la block-RAM de la FPGA, y el General-Purpose I/O (GPIO), con el que se gestiona la comunicación con los LEDs, los interruptores o los botones de la placa FPGA. El sistema utiliza una señal de reloj (SI_ClkIn) y una de reset (SI_Reset_N), activa en baja. Además, aunque su uso no es imprescindible, existe un interfaz EJTAG que facilita la carga y permite la depuración de programas en el sistema MIPSfpga.

La memoria incluye dos bloques de memoria, uno a partir de la dirección física 0x00000000 (Code/Data RAM) de 256KB, y otro a partir de la dirección física 0x1fc00000 (Reset RAM) de 128KB, como ilustra la Figura 3. Tras el reset, el procesador comienza buscando instrucciones a partir de la dirección 0x1fc00000. Así pues, como mínimo, esa dirección debe contener instrucciones. Normalmente, dichas instrucciones corresponden al código de arranque que se encarga de inicializar el sistema. Tras ello, se salta al código de usuario, ubicado en el otro bloque de memoria. Aunque este es el procedimiento habitual, en códigos simples que no utilizan ciertos componentes del sistema (como las caches o la TLB), es posible prescindir del código de arranque y ubicar el código de usuario a partir de la dirección física 0x1fc00000 para comenzar la ejecución inmediatamente tras el reset.

2.2 MIPSfpga Labs

Este segundo paquete incluye una serie de prácticas de laboratorio orientadas a la enseñanza de arquitectura de computadores, diseño de SoC, o codiseño HW/SW. En la segunda versión de MIPSfpga se ha ampliado significativamente el conjunto de prácticas y los temas tratados, pasando de las 9 prácticas iniciales a 25 prácticas que cubren todos los aspectos del computador, desde la programación en alto nivel y ensamblador (Parte 1) hasta el sistema de memoria (Parte 4), pasando por la Entrada/Salida (Parte 2) y el Core (Parte 3). La Tabla 1 pro-

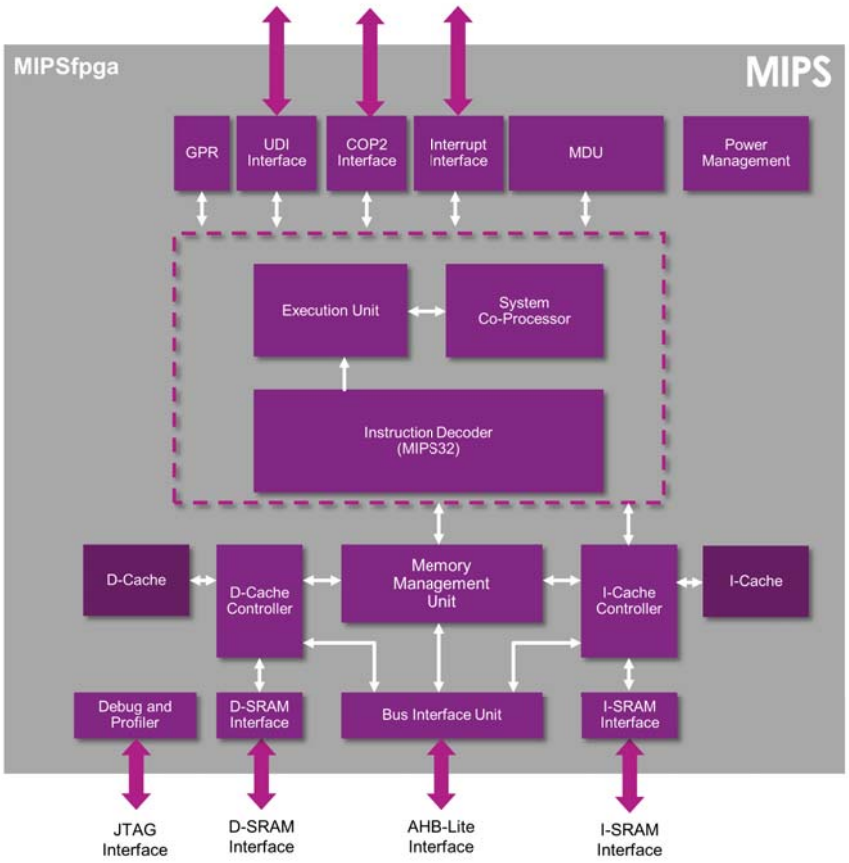


Fig. 1. Core de MIPSfpga

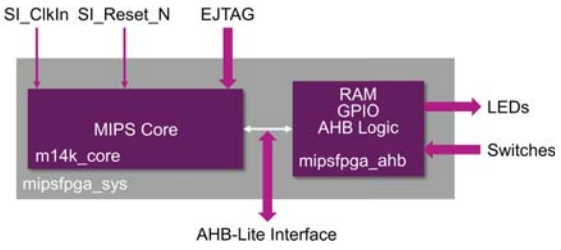


Fig. 2. Sistema MIPSfpga

porciona una breve descripción de estas 25 prácticas, cuya explicación se amplía notablemente en los siguientes párrafos.

La primera parte incluye cuatro prácticas. En la primera, se explica paso a paso cómo crear un proyecto en Vivado (para FPGAs de Xilinx) o en Quartus-

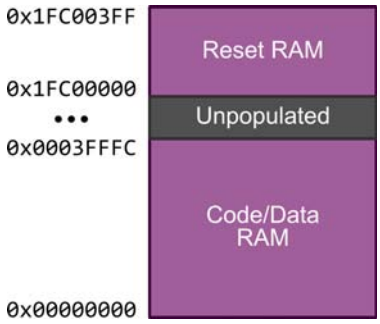


Fig. 3. Memoria física en MIPSfpga

Table 1. Prácticas incluidas en el paquete MIPSfpga Labs

#	Descripción
1	Creación de un proyecto en Vivado o Quartus-II
2	Crear, compilar, depurar y ejecutar programas en lenguaje C
3	Crear, compilar, depurar y ejecutar programas en ensamblador MIPS
4	Más ejercicios de programación en lenguaje C (opcional)
5	Ampliar el sistema con los displays de 7-segmentos de la placa
6	Ampliar el sistema con un contador
7	Ampliar el sistema con un timbre
8	Ampliar el sistema con un sensor de luz SPI
9	Ampliar el sistema con un LCD SPI
10	Comunicación por medio de interrupciones
11	Implementar un motor DMA para la comunicación entre periféricos
12	Implementar un motor Data Encryption Standard (DES)
13	Uso de los Performance Counters
14	Ejecución de una instrucción ADD y otras instrucciones aritméticas
15	Ejecución de una instrucción AND y otras instrucciones lógicas
16	Ejecución de una instrucción LW y otras instrucciones relacionadas
17	Ejecución de una instrucción BEQ y otras instrucciones relacionadas
18	Análisis de la Unidad de Gestión de Riesgos
19	Uso del Interfaz CorExtend
20	Introducción a las caches disponibles en MIPSfpga
21	Análisis de la D$ e implementación de nuevas configuraciones
22	Controlador de Cache: Análisis del acierto y el fallo
23	Controlador de Cache: Análisis de las políticas de gestión de contenido
24	Controlador de Cache: Análsis del Store Buffer y del Fill Buffer
25	Implementación de una Scratchpad

II (para FPGAs de Altera), y cómo portar MIPSfpga a otras placas. En las prácticas 2 y 3 se explica cómo crear, compilar, descargar, ejecutar y depurar programas en C y ensambaldor de MIPS en el sistema MIPSfpga. Por último, la práctica 4 propone más ejercicios de programación en C.

La segunda parte comienza con cinco prácticas sobre Entrada/Salida mapeada en memoria, en las que se da soporte a nuevos periféricos y se interactúa con ellos. Para las prácticas 7, 8 y 9 se requiere de ciertos componentes adicionales especificados en [4]. A continuación, las prácticas 10 a 12 analizan Entrada/Salida basada en interrupciones y DMA. Por último, en la práctica 13 se explica el uso de los Performance Counters disponibles en microAptiv y se proponen algunos programas sencillos en los que se evalúa analítica y experimentalmente el CPI.

Las prácticas de la tercera parte nos sumergen en el core. En las primeras cuatro prácticas (14-17) se añaliza el hardware utilizado en la ruta de datos y en la unidad de control del core para ejecutar los diferentes tipos básicos de instrucciones: Aritmética (ADD), Lógica (AND), Transferencia con Memoria (LW) y Salto Condicional (BEQ). Las cuatro prácticas siguen una estructura similar, comenzando con una extensa explicación teórica, realizando a continuación una simulación detallada de la instrucción correspondiente, y proponiendo por último una serie de ejercicios en los que se llega a implementar nuevas instrucciones (ADDIUPC, SEQ, NAND, SELEQZ/SELNEZ, LWI, LWPC o BC). En la práctica 18 se analiza la Unidad de Gestión de Riesgos. Para ello, se implementa un reloj de baja frecuencia, que nos permite observar a través de los LEDs las señales relacionadas con dicha unidad por medio de la ejecución de varios programas de ejemplo. El análisis en la placa se complementa con una simulación en Vivado de los mismos programas. Por último, en esta tercera parte se incluye una práctica sobre el uso del Interfaz CorExtend de MIPS (práctica 19). Esta funcionalidad permite al diseñador especificar e implementar sus propias instrucciones (User Defined Instructions, o UDIs), acelerando la ejecución de ciertos algoritmos o regiones críticas. La práctica describe en primer lugar el Interfaz, sus características y limitaciones, y su ubicación e interacción con el resto de estructuras del core. A continuación, se proponen una serie de ejercicios, desde los más básicos, en los que se añaden varias instrucciones (SELEQZ, NAND y SEQ), hasta los más avanzados, en los que se incluyen instrucciones de DSP o de punto flotante y se comparan diversos algoritmos por medio de los Performance Counters.

Por último, en la cuarta parte, se explora el sistema de memoria de MIPSfpga. En la práctica 20, al igual que en la 18, se implementa un reloj de baja frecuencia, que nos permite observar a través de los LEDs las señales relacionadas con los aciertos y fallos en la cache de datos, por medio de diversos códigos de ejemplo. En la práctica 21 se analiza el interfaz y organización interna de los arrays que constituyen la cache (i.e. Array de Datos, Array de Tags y Array Way-Select), se implementan y comparan nuevas configuraciones de la cache, en las que se varía el tamaño o la asociatividad, y se prueban diversas técnicas software de optimización del rendimiento. Las prácticas 22 a 24 analizan el controlador de cache. En la 22 se estudia la gestión del acierto y del fallo, primero de forma teórica, luego por medio de una simulación en la que se afianzan los conceptos teóricos, y por último a través de una serie de ejercicios. En la práctica 23 se describen las distintas políticas de gestión de contenido disponibles en las caches de microAptiv (política de reemplazo LRU, diversas políticas de escritura), se

implementan nuevas políticas (por ejemplo, una política de reemplazo FIFO) y se evalúan a través de varios códigos de ejemplo y haciendo uso de los Performance Counters. La práctica 24 explica el funcionamiento del Store Buffer (que almacena temporalmente el dato que escribe un STORE en la cache de datos) y del Fill Buffer (que almacena temporalmente el bloque a escribir en la cache de datos, proveniente de memoria, a consecuencia de un fallo), y propone una serie de ejercicios en los que se deben analizar varias secuencias de acceso a memoria. Por último, en la práctica 25 se añade una Scratchpad de instrucciones, y se compara el rendimiento de un mismo algoritmo ejecutado desde la cache de instrucciones y desde la Scratchpad.

Tras completar las prácticas del paquete MIPSfpga Labs, los estudiantes estarán preparados para afrontar proyectos más avanzados, como añadir periféricos I^2C o UART, o añadir nuevas funcionalidades al core (un prefetcher hardware, un predictor de saltos) y al sistema de memoria (un segundo nivel de cache, un predictor de vías).

2.3 MIPSfpga SoC

El último paquete de MIPSfpga 2.0 se llama MIPSfpga-SoC, y en él se explica cómo implementar un SoC centrado en el soft-core de microAptiv, y posteriormente cargar y ejecutar Linux en dicho SoC. El core de MIPS actúa como maestro de diversos periféricos (esclavos), a través del bus AHB-Lite (Figura 4). Para implementar los periféricos, se utilizan bloques IP de Xilinx (excepto el controlador GPIO, que proporciona Imagination), lo que reduce significativamente el tiempo de desarrollo. Sin embargo, dado que dichos periféricos se comunican a través de un bus Advanced eXtensible Interface (AXI), es necesario incluir un bridge AHB-Lite - AXI.

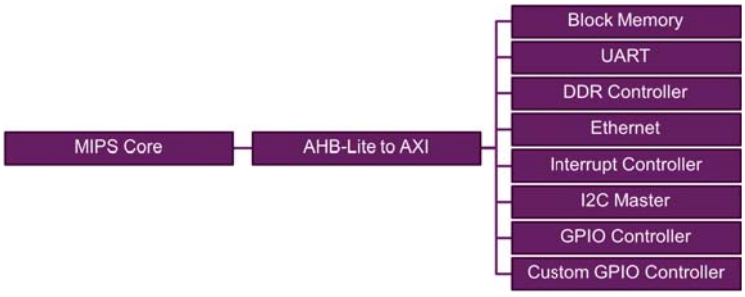


Fig. 4. Diagrama del Linux SoC

El sistema operativo Linux puede dividirse en dos partes: el Userspace y el Kernel. El primero, interactúa con el hardware a través de las llamadas al sistema que implementa el kernel. En este caso, utilizamos Buildroot. Por su parte, el kernel interactúa directamente con el hardware, proporcionando una

capa de abstracción. Se puede implementar el kernel con poco soporte hardware; en este SoC (Figura 4), incluimos una Unidad de Gestión de Memoria (MMU), un controlador de interrupciones, una serie de timers, un UART, una memoria, y un interfaz e-JTAG.

2.4 Adaptación a las *IEEE/ACM Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering*

Las asociaciones IEEE y ACM establecen en [6] once unidades fundamentales para la enseñanza de asignaturas del área de Ingeniería de Computadores, resumidas en la Tabla 2. Como tratamos de justificar en esta sección, creemos que las prácticas incluidas en MIPSfpga v2.0 cubren a la perfección estas unidades.

Table 2. Unidades fundamentales para la docencia en Ingeniería de Computadores establecidas por IEEE y ACM

Unit	**Name**
CE-CAO-1	History and overview
CE-CAO-2	Tools, standards and/or constraints
CE-CAO-3	Instruction set architecture
CE-CAO-4	Measuring performance
CE-CAO-5	Computer arithmetic
CE-CAO-6	Processor organization
CE-CAO-7	Memory system organization and architectures
CE-CAO-8	Input/Output interfacing and communication
CE-CAO-9	Peripheral subsystems
CE-CAO-10	Multi/Many-core architectures
CE-CAO-11	Distributed system architectures

El ISA de MIPS existe desde comienzos de la década de los 80 y sirvió de base a muchas otras arquitecturas posteriores, siendo por tanto indiscutible su papel en la historia de los computadores cubierta en la unidad CE-CAO-1. Las prácticas 2 y 3 del paquete MIPSfpga Labs describen cómo crear proyectos en Vivado, compilar programas en C o ensamblador de MIPS, y ejecutar y depurar estos programas en MIPSfpga, alineandose así perfectamente con las unidades CE-CAO-2 y CE-CAO-3.

En la práctica 13 se explica en detalle el uso de los Performance Counters disponibles en el core microAptiv. A continuación, se proponen una serie de ejercicios en los que se evalúa el rendimiento de distintos programas, tanto analítica como experimentalmente. Por otra parte, este recurso se utiliza ampliamente en las prácticas pertenecientes a las partes 3 y 4 de MIPSfpga Labs (Tabla 1), con el objetivo de medir eventos tales como el número de ciclos, número de instrucciones finalizadas, número de accesos y fallos a la cache de instrucciones/datos, etc. Podemos por tanto afirmar que la unidad CE-CAO-4 queda cubierta en esta infraestructura docente.

En cuanto a la unidad 5 (CE-CAO-5) de las recomendaciones del IEEE/ACM, ésta se cubre parcialmente en las prácticas 14 y 19. En la primera, se analiza en detalle la ejecución de una instrucción ADD, se analizan las señales de control relacinadas con las instrucciones aritméticas implementadas en microAptiv, se estudia la unidad aritmética incluida en este procesador, y se implementan nuevas instrucciones aritméticas. En la práctica 19, que analiza el Interfaz CorExtend, se propone un extenso ejercicio sobre aritmética en punto flotante, en el que se incluyen a través de este interfaz nuevas instrucciones de suma, multiplicación y división en punto flotante, se utilizan estas nuevas instrucciones para implementar el algoritmo de la bisección (que permite calcular las raices de una función), y se compara, por medio de los Perfromance Counters, el rendimiento de este algoritmo con el de uno en el que las operaciones en punto flotante se emulan por software.

La unidad CE-CAO-6 se cubre completamente con las prácticas 14 a 18. En estas prácticas, se analiza en gran detalle la organización del procesador microAptiv, y se compara con el procesador segmentado utilizado en [8]. Así, las prácticas comienzan con una introducción teórica, en la que se describe la unidad de control y la ruta de datos desde el punto de vista de los distintos tipos de instrucciones básicas: instrucciones aritméticas (práctica 14); instrucciones lógicas (práctica 15); instrucciones de transferencia con memoria (práctica 16); e insrucciones de salto condicional (práctica 17). Por su parte, en la práctica 18, se analiza teóricamente la Unidad de Riesgos. Después de esta profunda descripción, se realiza una simulación de la instrucción estudiada, y se propone un amplio conjunto de ejercicios.

Las prácticas 20 a 25 analizan exhaustivamente el sistema de memoria de MIPSfpga, cubriendo así la unidad CE-CAO-7. En la Sección 2.2 describimos en detalle cada una de estas prácticas. Entre muchas otras cosas, se comparan distintas configuraciones de cache, se prueban diferentes políticas de gestión de contenido, se analiza la gestión de aciertos y fallos, o se implementa una Scratchpad de instrucciones.

Las unidades CE-CAO-8 y CE-CAO-9 se cubren en las prácticas 5 a 12. En un primer grupo de prácticas (5-9), se añaden distintos periféricos, algunos de los cuales se comunican por medio del extendido bus SPI. Un segundo grupo de prácticas (10-12) introduce el uso de interrupciones y DMA para la Entrada/Salida.

Las unidades CE-CAO-10 y CE-CAO-11 no se cubren directamente en MIPSfpga, pues se centran en sistemas multi-core y en sistemas distribuidos (MIPSfpga es un sistema single-core). Sin embargo, la disponibilidad de un sistema completamente abierto y no ofuscado, hace que se puedan afrontar ejercicios y prácticas relacionados con estas unidades a modo de proyectos avanzados. Así, por ejemplo, en [11], los autores realizan profundas modificaciones al core microAptiv para implementar un multi-procesador de memoria distribuida de 120 cores.

3 Caso de uso de MIPSfpga v2.0

En esta sección describimos la utilización de MIPSfpga en la asignatura *Arquitectura de Sistemas Integrados* durante el curso 2016-17.

3.1 Resumen de la asignatura

La analizada en este artículo es una asignatura obligatoria del segundo cuatrimestre del cuarto curso de la titulación *Ingeniería Electrónica de Comunicaciones*, grado que se imparte desde hace cinco años en la Universidad Compulutense de Madrid (UCM). El objetivo fundamental de la misma es que los estudiantes adquieran conocimientos avanzados sobre la implementación y gestión del procesador, el sistema de memoria y la Entrada/Salida en los sistemas empotrados, microprocesadores y microcontroladores actuales.

En el curso 2016-17, se realizaron un total de 12 sesiones prácticas, cada una de dos horas, y 26 clases de teoría, de una hora cada una (nótese que estos números pueden variar ligeramente cada curso, dependiendo de diversos factores, por lo que pueden ser necesarias pequeñas modificaciones a lo que aquí exponemos). La planficación establecía que cada semana se realizara una sesión de laboratorio y se impartieran dos clases teóricas. Al tratarse de una asignatura con alta carga práctica (no en vano las práctias van a suponer un 50% de la calificación final), tratamos de coordinar y sincronizar de forma muy precisa ambas actividades, de forma que cada sesión de laboratorio reforzase los conceptos que se estaban tratando en la parte teórica.

3.2 Programa

Los alumnos que se matriculan en esta asignatura tienen conocimientos avanzados de diseño digital, programación en VHDL, y estructura de computadores (ISA de MIPS, procesadores mono- y multi-ciclo, sistema de Entrada/Salida), así como conocimientos básicos de programación en C++ y Sistemas Operativos. Por tanto, dada la formación inicial de los estudiantes, se puede plantear un programa ambicioso, tanto desde el punto de vista teórico como en lo que se refiere a las prácticas.

El programa se divide en cuatro módulos. En el Módulo 1 se revisan contenidos que los estudiantes ya han estudiado y deben conocer, como el ISA de MIPS, implementaciones mono- y multi-ciclo del procesador, y los conceptos básicos sobre jerarquía de cache y Entrada/Salida. Es por tanto un módulo breve, al que no se deben destinar más de 3 o 4 clases teóricas. En el Módulo 2 se describe detalladamente el procesador segmentado de [8], desde el punto de vista teórico primero, y con ejemplos y ejercicios después, y se analizan algunas técnicas microarquitectónicas de alto rendimiento, como la ejecución fuera de orden y superescalar, la predicción de saltos, el renombrado de registros, y algunas otras. Se deben destinar aproximadamente 10 clases teóricas y de problemas a este módulo. El Módulo 3 explora la jerarquía de la cache y la implementación de la memoria virtual. Los estudiantes ya deben conocer la gestión básica de

la cache (emplazamiento directo, política de post-escritura, etc.), por lo que se pueden analizar políticas y técnicas de gestión de la cache avanzadas, como diversas políticas de reemplazo de bloques, caches asociativas y caches multi-nivel, predicción de vías, cache de víctimas, cache no bloqueante, búsqueda de la palabra crítica en primer lugar, optimizaciones del compilador, etc. Al igual que en el Módulo 2, creemos que se deben dedicar alrededor de 10 clases a esta parte. Por último, el Módulo 4 introduce brevemente los Sistemas en Chip (SoC) y los Sistemas Empotrados, por lo que 3 o 4 clases teóricas serán suficientes.

3.3 Bibliografía

Indudablemente, el texto óptimo para esta asignatura es [8], tanto por el contenido teórico y práctico del mismo, como por el hecho de que las prácticas de MIPSfpga v2.0 están completamente alineadas con este libro. Concretamente, nos centramos en los capítulos 4 (*Hardware Description Languages*), 6 (*Architecture*), 7 (*Microarchitecture*) y 8 (*Memory and I/O Systems*). Además, con la compra de este libro se proporcionan conjuntos de trasparencias de cada módulo, que el profesor puede utilizar como apoyo para las clases teóricas. En nuestro caso, hemos extendido notablemente estas trasparencias, intercalando la descripción detallada del procesador y la jerarquía de cache utilizados por el sistema MIPSfpga, y comparándolos con los propuestos en el libro.

Para los problemas, hemos diseñado varias hojas de ejercicios, en las que se combinan problemas extraídos de diversos libros con otros de elaboración propia. Por último, en cuanto al material bibliográfico para la parte práctica de la asignatura, el incluido en MIPSfpga es más que suficiente, pues contiene extensos guiones de prácticas, una completa guía de uso de la infraestructura, y múltiples documentos propios de MIPS en los que se describe el procesador, la memoria, los Performance Counters, el Interfaz CorExtend, etc.

Como textos adicionales, se proponen los dos libros de John L. Hennessy y David A. Patterson [9] y [10], referencias obligadas en cualquier asignatura relacionada con la arquitectura de computadores.

3.4 Organización de las prácticas

Teniendo en cuenta los conocimientos iniciales de los alumnos, el programa de la asignatura y su ubicación en el cuarto curso de grado, creemos que la elección de la infraestructura MIPSfpga para la parte práctica de esta asignatura es totalmente acertada.

Como dijimos en la Sección 3.1, la asignatura incluye una sesión de laboratorio por semana. Durante el curso 2016-17, realizamos 12 sesiones, aunque estos números pueden variar ligeramente en otros cursos. En cualquier caso, no hay tiempo suficiente para completar las 25 prácticas de MIPSfpga Labs, por lo que tendremos que realizar una selección. La Tabla 3 recoge las prácticas elegidas para cada módulo del temario, a saber:

– La *Práctica 1* ilustra cómo crear un proyecto en Vivado, y cómo sintetizar y descargar el sistema MIPSfpga en la placa

- Para complementar la revisión del ISA de MIPS, se realizan las *Prácticas 2, 3 y 4*, en las que se ilustra cómo ejecutar y depurar un programa en MIPSfpga y se proponen varios ejercicios de programación en C y en ensamblador MIPS.
- Para complementar la revisión de la Entrada/Salida, se realiza la *Práctica 5*, en la que se añade el soporte necesario para comunicarse con los displays de 7-segmentos. Un aspecto muy interesante de esta práctica, a diferencia de las que los estudiantes han realizado en cursos anteriores, es que el dispositivo no solo se maneja a nivel software, sino que se implementa también su controlador hardware.
- El Módulo 2 de la asignatura analiza el procesador segmentado:
  - Para complementar la parte teórica, se realiza en primer lugar la *Práctica 13*, en la que se analiza el CPI de varios programas, comparando el rendimiento en el procesador segmentado de [8] con el de microAptiv, y realizando el análisis de forma analítica y, para el caso de MIPSfpga, también de modo experimental (por medio de los Performance Counters).
  - Para reforzar los conceptos teóricos también se realizan las *Prácticas 14-18*, en las que se analiza la ejecución de distintos tipos de instrucciónes, se añaden otras nuevas, y se estudia la unidad de gestión de riesgos del procesador.
- El Módulo 3 se complementa realizando la *Práctica 22*, en la que se analiza en profundidad la gestión de un acierto y un fallo de cache.
- Por último, asociado al Módulo 4, se realiza el Starter Tutorial del paquete MIPSfpga SoC, en el que se guía al estudiante en la creación de un SoC en el que se ejecuta Linux.

Table 3. Asociación entre módulos y prácticas

Módulo	**Contenido**	**Prácticas asociadas**
1	ISA de MIPS	Prácticas 2, 3 y 4
	Procesador mono/multi-ciclo	
	Gestión de la entrada/salida	Práctica 5
2	Procesador segmentado	Prácticas 13-18
3	Jerarquía de memoria	Prácticas 22-A y 22-B
4	Sistemas empotrados y SoC	MIPSfpga SoC - Starter Tutorial

Aunque, desgraciadamente, por falta de tiempo, muchas prácticas interesantes de MIPSfpga para complementar el temario se quedan en el tintero (como las Prácticas 10 y 11, en las que se estudia la gestión de la Entrada/Salida con interrupciones y DMA respectivamente; la Práctica 19, en la que se analiza el Interfaz CorExtend; la Práctica 21, en la que se experimenta con distintas configuraciones de cache; o la Práctica 23, en la que se analizan distintas políticas

de gestión de contenido), creemos que el subconjunto seleccionado cubre con bastante eficacia los conceptos más importantes de la asignatura.

La Tabla 4 muestra la planifiación específica empleada durante el curso 2016-17. En la primera clase, se reparten a los alumnos las FPGA y los BusBlaster, que conservarán durante todo el cuatrimestre para poder trabajar por su cuenta, y les pedimos que descarguen los tres paquetes de la web de Imagination Technologies [3], instalen las herramientas software en su portátil (siguiendo las indicaciones de MIPSfpga GSG), y comprueben que todo funciona correctamente. Además, se indica a los estudiantes que deben comenzar a estudiar por su cuenta el Capítulo 4 de [8], pues MIPSfpga utiliza lenguaje Verilog (salvo para los módulos de alto nivel, que están tanto en Verilog como en VHDL) y sus conocimientos se limitan a VHDL.

Table 4. Planificación específica del curso 2016-17

Sesión	**Descripción**
-	Casa: Instalar MIPSfpga antes de la primera sesión
1	Terminar instalación + **Práctica 1**
2	**Práctica 2** y **3** (C y ensamblador) + **Práctica 4**
-	Casa: Terminar Práctica 4
3	Ejercicio extra Práctica 4 + Test
4	**Práctica 5** (Displays de 7-segmentos)
-	Casa: Completar Práctica 5
5	Ejercicio extra Práctica 5 + Test
6	**Práctica 13** (Performance Counters)
-	Casa: Completar Práctica 13
7	Ejercicio extra Práctica 13 + Test
8	Trabajo en grupo (**Prácticas 14-18**)
-	Casa: Completar Prácticas 14-18
9	**Práctica 22-A** (Gestión del acierto cache)
-	Casa: Completar Práctica 22-A
10	**Práctica 22-B** (Gestión del fallo cache)
-	Casa: Completar Práctica 22-B
11	Ejercicio extra Práctica 22-B + Test
12	**MIPSfpga SoC** - Advanced Tutorial

Como se observa en la Tabla 4, cada práctica se realiza en 1 o 2 sesiones. Por ejemplo, la Práctica 13 se realiza en las sesiones 6 y 7: en la sesión 6 los alumnos comienzan a trabajar la práctica; luego, en el intervalo hasta la siguiente sesión, trabajan la práctica por su cuenta; y, para terminar, en la sesión 7, completan la práctica y responden a un examen individual sobre la misma. Como excepción, las Prácticas 14-18 se realizan en grupos de 3-4 miembros. Cada uno de los grupos realiza una práctica, prepara un trabajo en su casa, y lo expone al resto de estudiantes en una clase teórica.

3.5 Evaluación

La calificación final se calcula del siguiente modo: 0.5*EF + 0.3*PL + 0.2*TG; donde EF corresponde a la calificación del examen final, PL resulta de la calificación del laboratorio, y TG corresponde a la calificación del trabajo en grupo. Es importante evaluar adecuadamente el trabajo de laboratorio, pues la mitad de la calificación de la asignatura resulta de esta actividad, por lo que al finalizar cada práctica se realiza un examen individual sobre la misma.

3.6 Opinión de los alumnos

Para finalizar, incluimos a continuación algunas opiniones de alumnos que realizaron esta asignatura durante el curso 2016-17: "El curso, y en especial las prácticas, permiten conocer a fondo la arquitectura y la microarquitectura de un computador" (G. Diaz-Tejeiro); "Las sesiones de laboratorio nos han permitido entender muy bien el funcionamiento de un procesador comercial" (M. Sanchez); "El texto de las prácticas facilita el aprendizaje gracias a su amplio contenido" (J. Martin); "En un principio, resultaba muy complicado adaptarse a esta forma de aprendizaje. Sin embargo, una vez que nos acostumbramos, empezamos a disfrutar la asignatura, logrando entender a fondo cómo trabajan el core, la cache, la entrada/salida, etc. En mi opinión, es la mejor manera de estudiar una asignatura como esta." (A. Menendez); "Las prácticas nos permitieron afianzar los conceptos teóricos" (P. Fernandez); "Las prácticas nos mostraron cómo aplicar los conceptos de arquitectura de computadores al mundo real" (A. Villarin).

4 Trabajo relacionado

En la actualidad, podemos encontrar una gran cantidad de procesadores soft-core de muy diversas características. En esta sección, los analizamos brevemente y los comparamos con MIPSfpga. Las principales compañías de desarrollo y comercialización de FPGAs (Altera y Xilinx), ofrecen sus propios soft-cores (Nios/NiosII [12] y MicroBlaze [13] respectivamente), configurados específicamente para sus propias FPGAs. Estas alternativas presentan diversas desventajas: no son de código abierto, lo cual limita su uso significativamente; no están basados en soft-cores industriales/comerciales ni en un ISA comercial; y carecen de material docente extenso y de calidad. Por su parte, ARM también proporciona una opción de código no abierto, el Cortex M0 Design Start [14], un soft-core muy básico (compuesto únicamente de 8K puertas) y de bajo rendimiento. Se trata de un soft-core con el código ofuscado y con un soporte para depuración muy limitado (no incluye EJTAG). Además, no proporciona la posibilidad a la comunidad académica de integrarlo en silicio y, al igual que los dos anteriores, carece de buen material docente.

Existe también disponibilidad de soft-cores de código abierto. Dos opciones muy conocidas, ambas basadas en el ISA SPARC, son la familia OpenSPARC [15] y la familia LEON [16], desarrollados actualmente por Oracle (originalmente por

Sun Microsystems) y por Aeroflex Gaisler (originalmente por la European Space Agency), respectivamente. Si bien son alternativas interesantes, al igual que las descritas anteriormente no incluyen buen material docente, y están basadas en un ISA menos extendido en el mundo académico que el ISA de MIPS o el de ARM. Por último, otras dos alternativas que debemos mencionar son RISC-V [17], desarrollada originalmente por la Universidad de California, Berkeley, y openRISC, desarrollada por opencores.org [18]. Estos soft-cores están basados en ISAs no comerciales y, al igual que los anteriores, proporcionan escaso material docente.

MIPSfpga, por su parte, soluciona todas las limitaciones anteriores. Incluye un procesador soft-core industrial, de código no ofuscado, utilizado en diversos dispositivos comerciales, como el conocido PIC32MZ de Microchip. Dicho soft-core utiliza el release 3 del ISA de MIPS, ampliamente empleado en el mundo académico y con disponibilidad de multitud de documentación y soporte docente. MIPSfpga también proporciona gran cantidad de documentación, entre la que se incluye mucho material docente teórico y práctico. Gran parte de esta documentación está disponible en 5 idiomas (castellano entre ellos). Además, MIPSfpga incluye soporte para diversas FPGAs, tanto de Xilinx como de Altera, y es fácilmente extensible a otras FPGAs. Por último, debemos mencionar que Imagination Technologies ha cerrado un acuerdo con Europractice y MOSIS para ofrecer, al mundo académico, la posibilidad de integrar en silicio dos cores muy similares a microAptiv (los cores Warrior M-class 5100 y 5150).

5 Conclusiones

En este paper hemos descrito en gran profundidad la última versión de MIPSfpga (v2.0), publicada en julio de 2017, así como su aplicación a la enseñanza de la asignatura *Arquitectura de Sistemas Integrados* impartida en la UCM. Esta completa infraestructura ofrece gran cantidad de material docente, que permite cubrir perfectamente los temarios de asignaturas de Arquitectura de Computadores o Diseño SoC. Además, permite a los estudiantes enfrentarse a problemas similares a los que debe solucionar un ingeniero de computadores en la industria actual, por lo consideramos que puede ser especialmente adecuada para asignaturas avanzadas de grado (como la analizada en este paper) o asignaturas de máster.

6 Agradecimientos

Los autores agradecen la contribución de Imagination University Program, University of Nevada, Las Vegas, Imperial College London (UK), Grupo ArTeCS de la Universidad Complutense de Madrid y contratos TIN2015-65277-R, TIN2015-65316-P y Artículo-83 (nº 411-2016), Munir Hasan (IMG UK), Prashant Deokar (IMG India), Mahesh Firke (IMG India) Parimal Patel (Xilinx), Kent Brinkley (IMG USA), Rick Leatherman (IMG USA), Chuck Swartley (IMG USA), Sean Raby (IMG UK), Michio Abe (IMG Japan), Bingli Wang (IMG China),

Sachin Sundar (IMG USA), Alex Wong (Digilent Inc.), Matthew Fortune (IMG UK), Jeffrey Deans (IMG UK), Laurence Keung (IMG UK), Roy Kravitz (Portland State University), Dennis Pinto (UCM), Tejaswini Angel (Portland State University), Christian White, Gibson Fahnestock, Jason Wong, Cathal McCabe (Xilinx), Larissa Swanland (Digilent).

References

1. Kakakhel, Z., Harris, S., Harris, D.,: 'MIPSfpga: An unobfuscated commercial MIPS core and SoC that runs Linux". Embedded World 2016, Nuremberg, Germany, February 2016.
2. Harris, S., Owen, R., Sedano, E., Chaver, D.,: 'MIPSfpga: Hands-On Learning on a Commercial Soft-Core". European Workshop on Microelectronics Education, Southampton, UK, May 2016.
3. 'Imagination University Program - Teaching Resources'. https://community.imgtec.com/university/resources.
4. Harris, S., Harris, D., Chaver, D., et al.: 'MIPSfpga: Using a Commercial MIPS Soft-Core in Computer Architecture Education.'. IET Circuits, Devices and Systems, March 2017.
5. Chaver, D., et al.: 'Practical experiences based on MIPSfpga'. Workshop on Computer Architecture Education 2017 (held in conjunction with ISCA-17), June 2017.
6. 'CE2016 - Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering' IEEE and ACM, 2016.
7. Imagination Technologies Ltd., 'MIPS32® microAptiv™ UP Processor Core Family Datasheet', July 31, 2013
8. Harris, D., and Harris, S., 'Digital Design and Computer Architecture', Elsevier Science and Technology, 2a edición, 2012
9. Patterson, David A., and Hennessy, John L., 'Computer Organization and Design', Morgan Kaufmann, 5a edición, 2013
10. Hennessy, John L., and Patterson, David A., 'Computer Architecture: A Quantitative Approach', Morgan Kaufmann, 5a edición, 2011
11. Kumar H B, C., Ravi, P., Modi, G., Kapre, N..: '120-core microAptiv MIPS Overlay for the Terasic DE5-NET FPGA board', Int. Symp. on Field-Programmable Gate Arrays, Monterey, USA, February 2017
12. 'Altera - NIOS-II Processor', https://www.altera.com/products/processors/overview.html
13. 'Xilinx - MicroBlaze Soft Processor Core', http://www.xilinx.com/products/design-tools/microblaze.html
14. 'ARM - Cortex M0 Design Start', http://www.arm.com/products/designstart/index.php
15. 'Oracle - OpenSPARC', http://www.oracle.com/technetwork/systems/opensparc/index.html
16. 'Aeroflex Gaisler - LEON series Softcores', http://www.gaisler.com/
17. Waterman, A., Lee, Y., Patterson, D.A., et al., 'The RISC-V Instruction Set Manual, Volume I: User-Level ISA', 2014
18. 'OpenCores - OpenRISC', http://opencores.org/or1k/Main_Page

Uso de dispositivos FPGA como apoyo a la enseñanza de asignaturas de arquitectura de computadores

Francisco Charte, Macarena Espinilla, Antonio J. Rivera, and Francisco Pulgar

Departamento de Informática, Universidad de Jaén.
{fcharte,mestevez,arivera,fpulgar}@ujaen.es

Resumen Los estudiantes del actual Grado en Ingeniería Informática han de cursar obligatoriamente asignaturas en las que se aborda el estudio teórico de la arquitectura de un computador, dedicándose la parte práctica fundamentalmente a la programación en ensamblador usando un determinado conjunto de instrucciones y un software de emulación. En este artículo se propone complementar esa parte práctica, introduciendo el uso de dispositivos FPGA, de forma que el estudiante aprenda a diseñar un microprocesador a partir de sus componentes básicos.

Palabras clave: FPGA, microprocesador, arquitectura de computadores.

Abstract Computer Engineering students in Spanish universities have to take one or more courses devoted to theur learning of computer architecture. The theoretical part of these subjects are usually focused on describing the architecture itself, while practical sessions are used to introduce assembly programming by means of a certain instruction set which runs into a software emulator. This paper proposes to supplement practical sessions, so that students learn to design a microprocessor by themselves from its basic components, by introducing the use of FPGA devices.

Keywords: FPGA, microprocessor, computer architecture.

1. Introducción

El conocimiento detallado de las herramientas con las que se trabaja es un requisito imprescindible para cualquier profesional. En el campo de la Ingeniería informática esta necesidad se traduce en el aprendizaje de lenguajes de programación, sistemas operativos, protocolos de red y un sinfín de capas de software que es necesario configurar, administrar y gestionar. Por debajo de todas esas capas se encuentra el hardware, la máquina que, con el microprocesador actuando como CPU (*Central Processing Unit*) al frente, se encarga de ejecutar el software, traduciendo conjuntos de órdenes en operaciones concretas de procesamiento de datos.

El aprendizaje de la arquitectura del computador, por tanto, ha de ser también parte esencial de la formación en Ingeniería Informática. El diseño de computadores, el desarrollo de compiladores, la programación a bajo nivel de controladores de dispositivos o el núcleo del sistema operativo, con la optimización que permite aprovechar al

máximo las características de cada máquina, se apoyan en el saber profundo del funcionamiento del computador. Por estas razones en los estudios oficiales de Grado en Ingeniería en Informática se incluyen asignaturas como *Electrónica Digital*, *Fundamentos de Arquitectura de Computadores*, *Arquitectura de Computadores* o *Microprocesadores y Microcontroladores*, entre otras.

La metodología empleada para el estudio de la arquitectura de computadores ha ido, como no podía ser de otra manera, evolucionando con el tiempo, dependiendo de los medios disponibles en cada momento. Son muchas las universidades en las que desde hace años se recurre al uso de emuladores/simuladores [1] de una determinada arquitectura, tomada como referencia, de tal forma que los estudiantes se centran fundamentalmente en escribir código en ensamblador, ejecutarlo en dichos emuladores y analizar los resultados. De esta forma el emulador, una aplicación software, se ha convertido en la herramienta de enseñanza básica, al permitir operar con una arquitectura bien conocida y en un entorno controlado.

El progresivo abaratamiento de los circuitos FPGA (*Field Programmable Gate Array*), conjuntamente con la disponibilidad de herramientas software gratuitas y en algunos casos de acceso libre para trabajar con este tipo de hardware, abre un abanico de nuevas posibilidades para la enseñanza de arquitectura de computadores que merece la pena considerar. El objetivo de este artículo es plantear cuál podría ser la adaptación de la metodología usada para impartir asignaturas como las antes citadas, evaluando las ventajas y problemas que potencialmente aportaría el uso de hardware reconfigurable, concretamente FPGA, en sustitución o complementariamente a los medios software empleados actualmente.

La estructura del presente artículo es la siguiente: en la Sección 2 se hace una somera descripción de la metodología empleada actualmente para la enseñanza de asignaturas relacionadas con la arquitectura de computadoras, centrada en la Universidad de Jaén donde los autores desarrollan su actividad. La Sección 3 describe potenciales alternativas a la metodología existente. El objetivo de la Sección 4 es la de introducir conceptos fundamentales sobre hardware reconfigurable y, más concretamente, sobre el trabajo con FPGAs, como paso previo para, en la Sección 5, realizar propuestas relativas al uso de ese tipo de hardware como herramienta de enseñanza.

2. Enseñanza de arquitectura de computadores

En el libro blanco de la titulación Grado en Ingeniería Informática publicado por la Agencia Nacional de Evaluación de la Calidad y Acreditación [2] (ANECA), concretamente en la categoría *Contenidos específicos de la Ingeniería Informática*, a la que se asigna entre un 35 % y un 40 % de la carga de contenidos de la titulación, se define la subcategoría 2.4: *Ingeniería de computadores*, formando parte de ella los descriptores *Fundamentos, Estructura y Arquitectura de computadores. Tecnología de Computadores.*

Por otra parte, en el Computer Engineering Curricula 2016 [3] (CE2016) se detallan cuáles deberían ser los contenidos centrados en el estudio de la arquitectura de computadores, concretamente en el módulo *CE-CAO Computer Architecture and Orga-*

nization, al que se recomienda asignar un total de 60 *core hours*[1]. Entre otros aspectos se destacan conocimientos como la estructura básica de un computador, los diferentes niveles de memoria, conjuntos de instrucciones con representación a nivel máquina y ensamblador, etc.

El libro blanco de la ANECA y el CE2016 están entre los recursos más importantes a la hora de diseñar y planificar los contenidos del Grado en Ingeniería Informática en el sistema universitario español. En el caso concreto de la Universidad de Jaén[2], que es el tomado aquí como referencia, los contenidos correspondientes a las recomendaciones de ANECA y CE2016 son recogidos fundamentalmente en las asignaturas detalladas en las siguientes subsecciones[3].

2.1. Fundamentos de Arquitectura de Computadores

Esta asignatura representa el primer contacto del estudiante de Grado en Ingeniería Informática con la arquitectura de un ordenador, al ser impartida durante el primer cuatrimestre del primer curso. Cuenta con una carga docente de 6 ECTS y es impartida por el profesorado del área Arquitectura y Tecnología de Computadores.

Los contenidos teóricos de la asignatura abordan los diferentes niveles de organización estructural de un computador: componente físico, electrónico, digital, transferencia de registros, etc., la evolución de los ordenadores (generaciones), su arquitectura básica (unidad aritmético-lógica, unidad de control, registros, etc.), representación de datos a bajo nivel, gestión de entrada/salida, etc.

El programa de prácticas se desarrolla esencialmente a través de la implementación en ensamblador de soluciones a problemas cuya complejidad va creciendo paulatinamente a lo largo del cuatrimestre, usando para ello el conjunto de instrucciones Intel 8085. La ejecución de estos programas se lleva a cabo con el emulador j8085sim, cuya interfaz de usuario puede apreciarse en la Figura 1.

2.2. Electrónica Digital

Es una asignatura impartida en el segundo cuatrimestre del primer curso de la titulación, estando a cargo del área de Tecnología Electrónica y con una carga docente de 6 ECTS. Pertenece al módulo de formación básica, junto a materias como física y matemáticas.

El objetivo de esta asignatura es transmitir al estudiante los conocimientos necesarios sobre el funcionamiento de componentes básicos de la arquitectura de un computador, incluyendo biestables, contadores, multiplexores, sumadores y otros elementos esenciales, creándolos a partir de puertas lógicas simples. Son los componentes electrónicos a partir de los cuales se diseñan la unidad aritmético lógica, unidad de control, registros y demás partes de un microprocesador, según la estructura introducida en la anterior asignatura.

[1] En dicho documento se denomina así a las horas dedicadas exclusivamente a introducir nuevos conocimientos, con independencia de los medios y actividades empleadas para ello.

[2] http://www.ujaen.es.

[3] Las guías docentes correspondientes a estas asignaturas están disponibles públicamente en la web de la Escuela Politécnica Superior de Jaén (http://eps.ujaen.es).

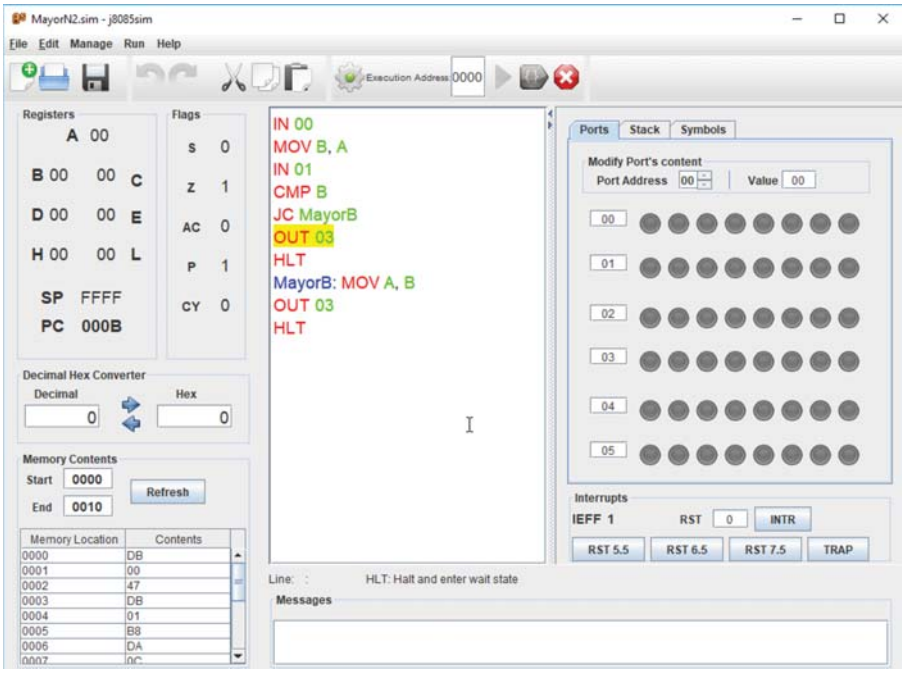


Figura 1. Simulador 8085 empleado en las sesiones prácticas.

La realización de prácticas se lleva a cabo mediante un *entrenador digital*[4], un software que permite al estudiante disponer estos componentes básicos en una placa virtual y conectarlos entre sí para crear otros de mayor complejidad.

2.3. Arquitectura de Computadores

Sobre la base del conocimiento adquirido durante el primer curso de la titulación, en el primer cuatrimestre del segundo curso los estudiantes cursan esta asignatura que, como las dos anteriores, cuenta con una carga docente de 6 ECTS. Al igual que Fundamentos de Arquitectura de Computadores, esta también es impartida por el área de Arquitectura y Tecnología de Computadores.

El núcleo del temario teórico de esta asignatura es el paralelismo en la arquitectura de microprocesadores, desde la segmentación de cauce y los procesadores superescalares a los de tipo VLIW (*Very Large Instruction Word*) y vectoriales [4]. Los estudiantes aprenden a detectar los diferentes tipos de conflictos que aparecen durante la ejecución de código a nivel de máquina: riesgos de datos, de control y estructurales, así como los mecanismos diseñados a lo largo del tiempo para superar dichos problemas: caminos de *bypass*, predictores de saltos, emisión desordenada de instrucciones, etc.

Al igual que ocurre con las dos asignaturas previas, en esta las prácticas también se realizan mediante software de emulación. Concretamente se emplea un simulador de

[4] http://www4.ujaen.es/ aabarca/descargas.htm

procesador con arquitectura DLX y segmentación de cauce y otro de tipo superescalar (véase la Figura 2) con el mismo conjunto de instrucciones. Los estudiantes aprenden a usar un conjunto de instrucciones ensamblador distinto al empleado en Fundamentos de arquitectura de computadores, más avanzado, así como a identificar las técnicas que hacen posible optimizar su ejecución: reordenación de código, desenrollado de bucles, etc.

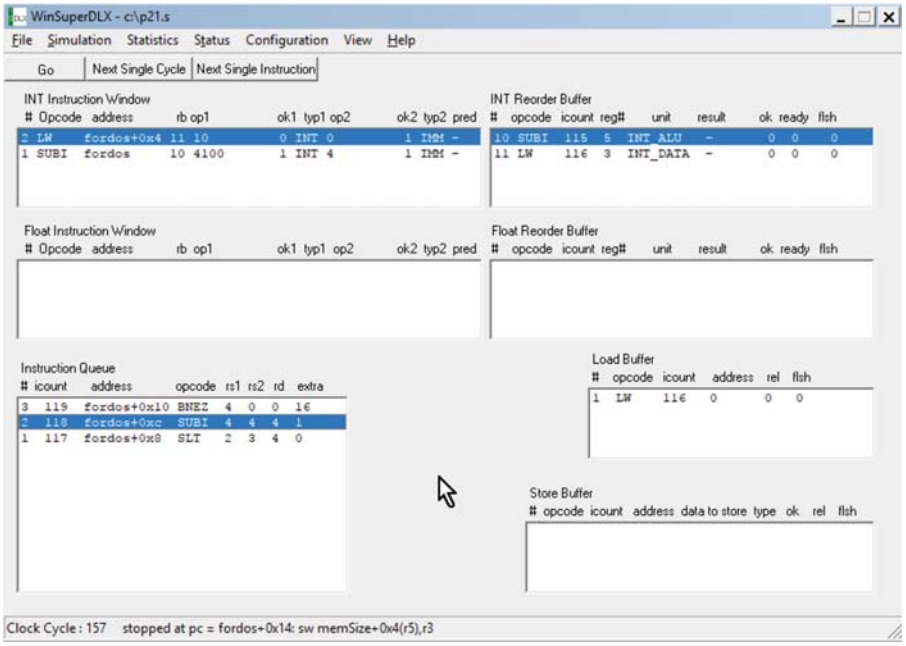


Figura 2. Simulador de procesador superescalar WinSuperDLX.

2.4. Microprocesadores y Microcontroladores

A diferencia de las previas, asignaturas todas ellas obligatorias, esta es de carácter optativo y se imparte en el primer cuatrimestre del cuarto curso de la titulación, estando a cargo del área Tecnología electrónica. Como el resto, tiene asignada una carga docente de 6 ECTS.

La mayor parte del programa teórico de esta asignatura se centra en el estudio de la arquitectura de una familia de microcontroladores, incluyendo la estructura de su memoria, puertos de E/S y funcionamiento de las interrupciones. Se compara esta arquitectura con la de otros circuitos de control, como los microprocesadores, ASIC (*Application Specific Integrated Circuit*) y PLD (*Programmable Logic Device*). Tal y como se apunta en la propia guía docente, el objetivo principal es complementar los estudios abordados en las tres asignaturas citadas en los apartados anteriores.

En cuanto al trabajo práctico, este se centra en el desarrollo de soluciones simples con un microcontrolador, concretamente unas placas de tipo Arduino, e incluyen la programación de sus puertos de E/S y el trabajo con distintos tipos de periféricos, todo ello orientado a aplicaciones de adquisición de datos y de control.

3. Metodología actual y potenciales acciones complementarias

A partir de la descripción dada en la sección previa, que aunque corresponde a la Universidad de Jaén nos consta que es similar en cuanto a estructura y metodología a la seguida en otras universidades, pueden derivarse algunos hechos de interés:

- Los contenidos teóricos de Fundamentos de Arquitectura de Computadores y Arquitectura de Computadores se centran en el estudio de las diferentes partes de un computador, esencialmente las partes de la CPU, memoria y E/S. En contrapartida las prácticas consisten básicamente en aprender a programar en ensamblador, primero, y a optimizar dicho código usando algunas técnicas comunes, después.
- La realización de las prácticas se desarrolla casi en todos los casos mediante aplicaciones que emulan/simulan una determinada arquitectura, como la del procesador 8085 o DLX[5]. En general son aplicaciones considerablemente antiguas, con muchas limitaciones y que plantean a los estudiantes problemas para su uso.
- El salto desde la asignatura Electrónica Digital, en la que se conocen los componentes electrónicos más básicos, al diseño de las partes de una CPU (registros, ALU, etc.) usando dichos elementos no se encuentra definido en los contenidos posteriores, especialmente desde una perspectiva práctica que permita a los estudiantes establecer una conexión clara entre las técnicas empleadas en ambas partes.

Con el objetivo de mejorar tanto la perspectiva global sobre la materia como la experiencia adquirida por los estudiantes, planteamos una serie de potenciales cambios en la actual metodología, como por ejemplo:

- Las dificultades que plantea el uso de los actuales emuladores/simuladores representan un obstáculo que requiere un esfuerzo adicional por parte del estudiante. La sustitución de los mismos por herramientas de uso más accesible reduciría dicho esfuerzo, que podría en su lugar dedicarse a los objetivos siguientes.
- Aprender a usar el conjunto de instrucciones de una determinada arquitectura, programando en ensamblador, es un paso fundamental en el aprendizaje. Estas prácticas, no obstante, podrían complementarse incluyendo el propio diseño del computador a partir de componentes básicos, dejando en manos del estudiante el diseño una ALU básica, la correspondiente unidad de control, el conjunto de registros, buses de comunicaciones, etc.
- La configuración de procesadores avanzados como los estudiados en Arquitectura de Computadores, con cauces segmentados o arquitectura superescalar, es prácticamente la única intervención del estudiante en su funcionamiento, estableciendo por ejemplo el número de unidades funcionales, tamaño de la ventana de instrucciones, etc., todo ello mediante una interfaz gráfica de usuario. Una alternativa a este

[5] Una arquitectura definida con fines didácticos basada en la del procesador MIPS.

método de aprendizaje podría ser la introducción de un lenguaje de descripción de hardware, como VHDL o Verilog [5], que ofreciese mucha más flexibilidad a la hora de diseñar cada aspecto del microprocesador sobre el que se quiere trabajar.

La introducción de todos o parte de estos cambios, así como de otros en el mismo sentido, precisan de la selección de una herramienta de trabajo adecuada para los estudiantes. Podría plantearse el desarrollo de nuevo software de emulación/simulación, sin las limitaciones y problemas que plantea el actual. Otra alternativa, más cercana a la realidad en cuanto que se emplearía hardware real en lugar de software, y previsiblemente de mayor utilidad para los estudiantes, se basaría en el uso de hardware reonfigurable.

4. Hardware reconfigurable y FPGAs

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

Paralelamente a los anteriores, durante las últimas tres décadas también se han desarrollado CI configurables por el usuario. Inicialmente estas soluciones, plasmadas en circuitos como las PAL (*Programmable Array Logic*) y PLA (*Programmable Logic Array*) [6], se basaban en tecnologías de fusibles, por lo que únicamente podían configurarse una vez. La disponibilidad de nuevos tipos de memorias, incluyendo las estáticas y las no volátiles, han hecho posible fabricar CI reconfigurables.

4.1. Introducción al hardware reconfigurable

Los orígenes del hardware reconfigurable se remontan más de medio siglo atrás, concretamente a 1959, y tienen lugar en la UCLA [7] a iniciativa del matemático John Pasta. Este plantea un desafío: crear un ordenador en el que se combine una parte de hardware fijo con otra variable configurable por el operador, naciendo así el concepto de arquitectura F+V. La *configurabilidad* se conseguía merced a unos módulos que actuaban como componentes básicos y que el operario podía instalar e interconectar en una placa base que, posteriormente, se instalaba en el ordenador en cuestión.

El diseño de circuitos digitales empleando integrados discretos, con unas pocas puertas lógicas, dispuestos sobre una placa de conexionado es algo del pasado, aunque sigue siendo una herramienta didáctica muy útil (véase Figura 3) para la comprensión de los principios más básicos del funcionamiento de un computador. Actualmente es habitual el uso de estos recursos sustituyendo los CI discretos por microcontroladores como la

Raspberry y Arduino, combinándolos con diversos sensores, leds, etc., o bien recurriendo a entrenadores digitales: aplicaciones software que simulan el funcionamiento del hardware.

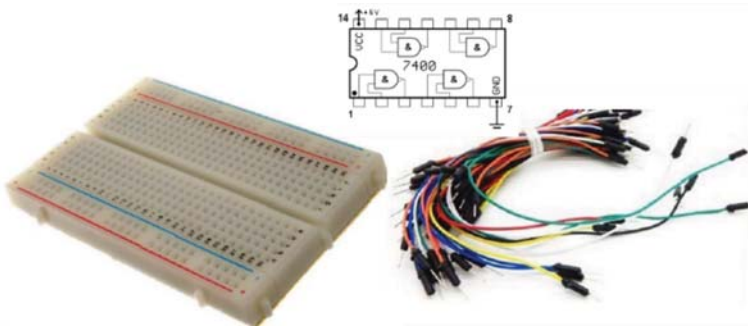


Figura 3. Placa de prototipado, circuito integrado de la familia 7400 con cuatro puertas AND y cables para establecer conexiones.

Durante la década de los 70 aparecen los primeros circuitos integrados con arquitectura configurable, las anteriormente citadas PAL y PLA. Estos ofrecen matrices de puertas lógicas con conexiones sin establecer, de forma que el CI puede realizar distintas funciones según el proyecto en el que vayan a utilizarse. Las conexiones no se cablean, como ocurría con los integrados discretos, sino que se fijan mediante un proceso que consiste en *quemar* unos fusibles internos [8] usando para ello un hardware a medida, como podía ser una placa conectable a un PC.

En la década de los 80 el fabricante Lattice introduce los GAL (*Generic Array Logic*), análogos a los PAL pero con una tecnología de establecimiento de las conexiones basada en memoria EEPROM [9]. A diferencia de PAL y PLA, que eran CI que únicamente podían configurarse una vez, los GAL podían ser reconfigurados tantas veces como se necesitase, aportando mucha más flexibilidad.

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

4.2. Estado actual de la tecnología FPGA

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

En 2016 Intel presentó sus procesadores para servidores Xeon con circuitería FPGA integrada[6], destinados al diseño de centros de procesamiento de datos que, como los creados por Microsoft para sus servicios Azure y Bing a partir de su proyecto Catapult [10], combinan un microprocesador clásico con la posibilidad de implementar en hardware las partes más críticas para el rendimiento y consumo del sistema. En el extremo opuesto, en cuanto a nivel de prestaciones se refiere, también encontramos la familia de procesadores Atom E600C[7] con FPGA integrada, producto dirigido a sistemas de tipo IoT (*Internet of Things*) y empotrados.

Por otra parte Xilinx, el primer fabricante mundial de dispositivos FPGA, también cuenta con productos que aúnan en un mismo encapsulado hardware reconfigurable con núcleos de procesamiento de tipo ARM[8]. Ambas gamas de producto, la de Intel y la Xilinx, hacen posible la implementación de soluciones híbridas hardware-software. Esta es una tendencia que está llegando incluso a los servicios de computación en la nube, como los ofrecidos por AWS (*Amazon Web Services*)[9], en los que el desarrollador cuenta con instancias de ejecución que combinan los servidores tradicionales con hardware reconfigurable de tipo FPGA. Todo ello apunta a un importante crecimiento en los próximos años de la demanda de profesionales formados en técnicas de hardware reconfigurable, por lo que su incorporación en la enseñanza de las asignaturas relacionadas con arquitectura de computadores conllevaría una mejora en la formación del estudiante de cara a su futuro.

Esta evolución del uso de hardware reconfigurable, cada vez más habitual para distintos tipos de soluciones, queda recogida en la actualización del Computer Engineering Curricula de la ACM/IEEE [11] a través de un incremento en las horas de dedicación a su estudio respecto a versiones previas de este documento, en detrimento de otras técnicas heredadas como VLSI.

4.3. Metodología de trabajo con hardware reconfigurable

El procedimiento de trabajo con hardware reconfigurable implica múltiples pasos específicos, estando más cerca del proceso de diseño de circuitos tipo ASIC que del

[6] https://www.nextplatform.com/2016/03/14/intel-marrying-fpga-beefy-broadwell-open-compute-future.
[7] http://www.embeddedintel.com/commentary.php?article=2143.
[8] https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html
[9] https://aws.amazon.com/es/blogs/aws/developer-preview-ec2-instances-f1-with-programmable-hardware/.

desarrollo de software. Los pasos fundamentales son los presentados esquemáticamente en la Figura 4.

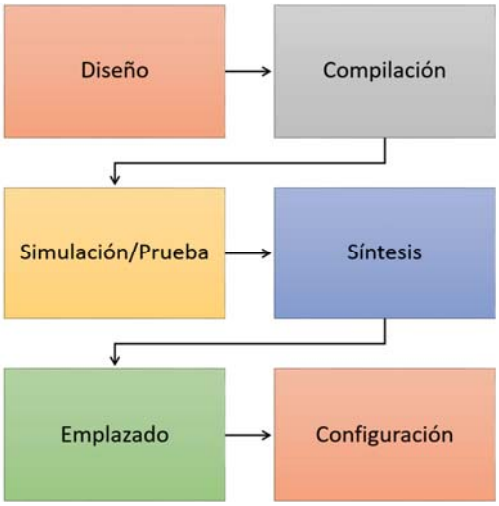


Figura 4. Pasos habituales en el flujo de trabajo con FPGA.

Se parte con el diseño de la solución que quiere implementarse en el hardware reconfigurable, tarea para la cual se puede, dependiendo de la complejidad del caso, elaborar un esquemático (véase la Figura 5) o bien describir el hardware mediante un lenguaje apropiado[10].

Tras compilar el diseño se procede normalmente a probarlo, usando para ello una herramienta de simulación. Esta puede ser de bajo nivel, construyendo *waveforms* que representan los cambios en las señales a lo largo del tiempo (véase la Figura 6), o bien estar descrita en forma de *test bench* usando el propio lenguaje de descripción de hardware.

Una vez se ha confirmado el correcto funcionamiento del circuito en el entorno de simulación llega el momento de sintetizar dicho circuito, determinando el hardware de la FPGA que se empleará para implementarlo. Asimismo es necesario emplazar las señales de entrada/salida (véase la Figura 7), estableciendo los pines del integrado al que se conectarán. Los elementos conectados a dichos pines pueden estar predeterminados o no, dependiendo del hardware concreto que se emplee.

El paso final consiste en transferir la configuración generada por la herramienta de diseño/desarrollo al hardware, configurando la FPGA. Las placas de aprendizaje suelen contar con conexión USB a fin de facilitar esta tarea, pero al trabajar con FPGA integrada en el encapsulado de un microprocesador, como las mencionadas en la sección previa, el conexionado sería interno.

[10] Los dos estándares en cuando a lenguajes de descripción de hardware son VHDL y Verilog, estando el uso del primero más extendido en Europa y el del segundo en Estados Unidos.

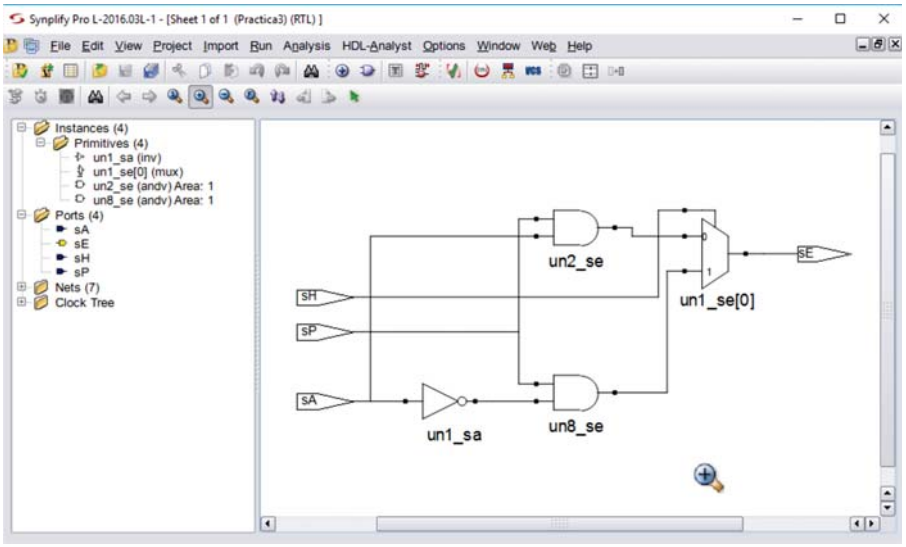


Figura 5. Diseño de un circuito simple a partir de un esquemático.

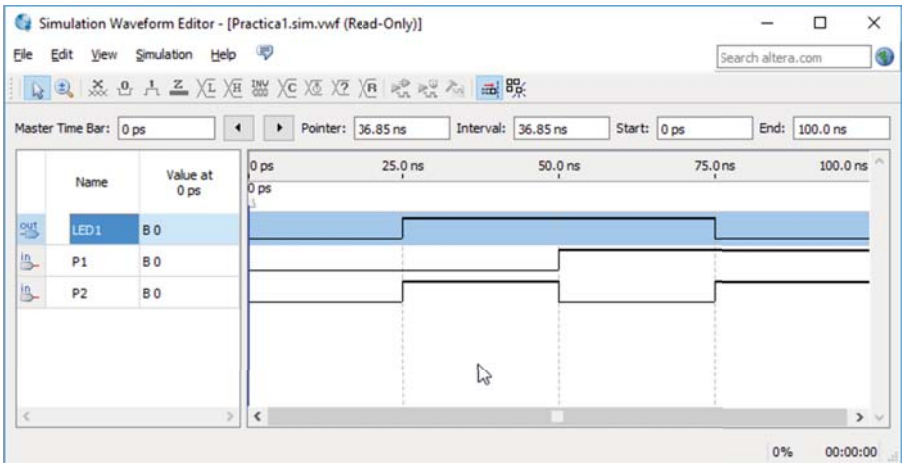


Figura 6. Se prueba el diseño mediante una herramienta de simulación.

5. Uso de FPGA para la enseñanza de arquitectura de computadores

En la Sección 3 se planteaban potenciales cambios en la metodología seguida hasta ahora para la enseñanza de arquitectura de computadores, apuntando al final que estas actividades requerían el uso de las herramientas adecuadas. En la presente sección se describe cómo podrían desarrollarse dichas actividades empleando como herramientas hardware reconfigurable, concretamente tarjetas de aprendizaje con circuitería FPGA.

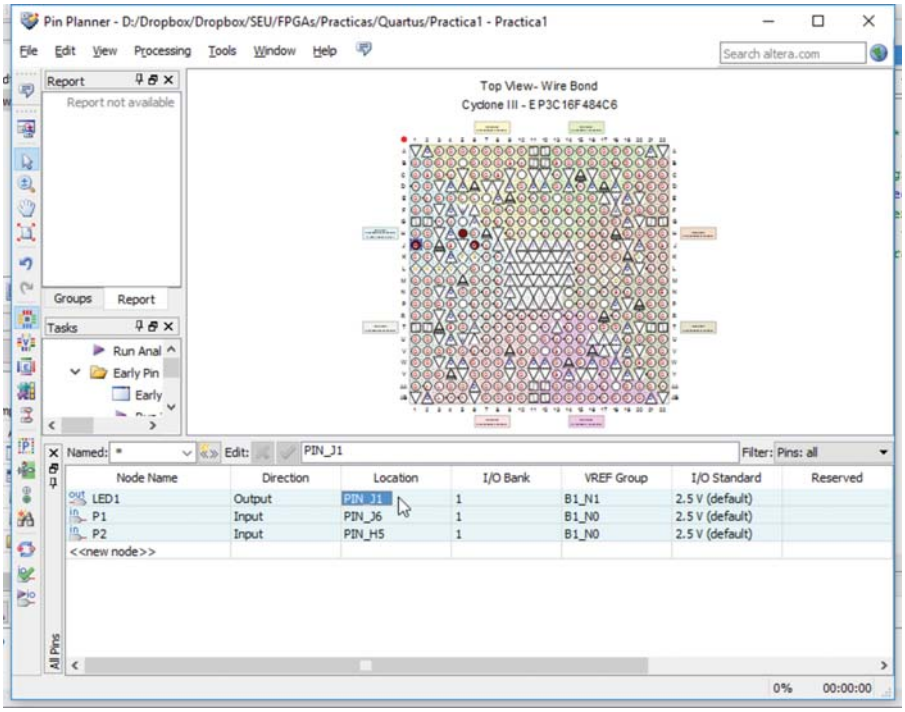


Figura 7. El emplazamiento conecta las señales de E/S del diseño a pines físicos de la FPGA.

Las FPGA actualmente se usan en asignaturas como *Implementación de algoritmos Hardware* [12], perteneciente al Grado en Ingeniería Informática (especialidad en Ingeniería de Computadores) de la Universidad de Granada, o *Programación Hardware*, perteneciente al Grado en Ingeniería Informática de la Universidad de Jaén. Se trata, por tanto, de una tecnología presente en algunas universidades españolas y con la que parte del profesorado ya está familiarizado.

5.1. Selección del hardware a emplear

Trabajar con hardware real, en el que la implementación de un cierto microprocesador simple, diseñado para el aprendizaje, opere como lo haría cualquier otro circuito integrado, ofrece al estudiante una visión alternativa, posiblemente complementaria, a la de los emuladores software. No obstante, la adquisición de dicho hardware implica un determinado coste que será necesario asumir. Este será proporcional al número de unidades que se precise adquirir que, a su vez, dependerá del número de estudiantes matriculados en la asignatura.

Actualmente los tres principales fabricantes de FPGA, Xilinx, Altera/Intel y Lattice, cuentan con productos dedicados al aprendizaje. Son tarjetas con una FPGA básica y algunos puertos de E/S, incluyendo leds, pulsadores, microinterruptores, etc., dependiendo de cada caso. Los precios oscilan desde los 20 euros del producto más básico de

Lattice (véase la Figura 8) hasta más de 100 euros para las placas de entrenamiento avanzadas que cuentan con salida de vídeo, pantallas de siete segmentos y LCD, etc. Las primeras son incluso accesibles para los estudiantes, que podrían adquirirlas como material propio de prácticas a título individual o en pequeños grupos.

Figura 8. Tarjeta FPGA para aprendizaje Lattice ICEstick iCE40HX1K.

Como se apuntaba anteriormente, en universidades como la Universidad de Jaén y la Universidad de Granada ya existen asignaturas en las que se usa hardware FPGA, por lo que esos mismos dispositivos podrían también emplearse ocasionalmente en las asignaturas correspondientes a arquitectura de computadores.

5.2. Diseño basado en componentes simples

Las herramientas de diseño facilitadas por los diferentes fabricantes de productos FPGA, así como las disponibles a través de licencias de código libre[11], ofrecen bibliotecas de puertas lógicas y algunos componentes simples, como los codificadores, descodificadores y multiplexores, que pueden ser empleados durante las prácticas de asignaturas como Electrónica Digital.

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

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

[11] http://fpgawars.github.io/

5.3. Diseño basado en módulos reutilizables

Tanto VHDL como Verilog, los dos lenguajes de descripción de hardware de uso más habitual, contemplan la posibilidad de crear bibliotecas de módulos con componentes prediseñados. Se pueden tener módulos con elementos básicos, como registros, sumadores, unidad de control, etc., de tal forma que el estudiante únicamente tiene que aprender a diseñar un microprocesador/computador "instanciando" esos componentes con la configuración que se demande. Pueden ajustar el tamaño de los registros, ancho de los buses de datos y direcciones, etc. Asimismo sería posible incluir elementos más complejos: unidades funcionales de distintos tipos, predictor de saltos, etc. De esta forma el aprendizaje teórico sobre arquitectura de computadores se reforzaría con una componente práctica, en la que es el propio estudiante el que puede diseñar su microprocesador.

Una alternativa al enfoque anterior, basado en el aprendizaje a través de la construcción de soluciones desde abajo (componentes básicos) hacia arriba (sistema completo), consistiría en facilitar al estudiante el diseño completo de un sistema, permitiéndole realizar ajustes en la configuración de este: cambiar el tamaño de la ventana de instrucciones, número de unidades funcionales de cada tipo, memoria del predictor de saltos, etc. Existen diseños libres de multitud de microprocesadores escritos en VHDL/Verilog, preparados para su uso o adaptación a las necesidades de cada asignatura, desde micros clásicos de 8 bits [13] hasta microprocesadores multinúcleo [14], pasando por micros teóricos como DLX [15]. Incluso cabe la posibilidad de sintetizar en FPGA emuladores de procesadores didácticos como WepSIM [16].

Al igual que para el aprendizaje de electrónica digital, el procedimiento de trabajo podría estar basado en el diseño esquemático, colocando y conectando bloques para formar el computador, o bien en una descripción formal en un lenguaje como VHDL. Ambas son herramientas que servirían al estudiante no solo para cursar posteriormente otras asignaturas, como las antes citadas Implementación de algoritmos hardware y Programación hardware, sino también como experiencia útil para un hipotético futuro laboral en el diseño de soluciones basadas en hardware reconfigurable.

6. Conclusiones

La formación de los estudiantes del Grado en Ingeniería Informática en el área de arquitectura de computadores es fundamental. En la actualidad, tal y como se ha descrito, las clases teóricas suelen dedicarse al estudio de la arquitectura y las prácticas a la programación en ensamblador. La estructura del computador descrito en teoría ya se encuentra implementada, habitualmente en un emulador software, y el estudiante se limita a utilizarla.

Nuestra propuesta tiene el objetivo de complementar la metodología seguida actualmente, ampliando los conocimientos adquiridos por los estudiantes. Para ello se recomienda la introducción del hardware reconfigurable, ya en uso en otras asignaturas que el estudiante puede cursar posteriormente de forma optativa. Parte de los recursos necesarios, así como la experiencia por parte del profesorado en este campo, ya están disponibles.

Desde nuestra perspectiva los beneficios para los estudiantes son claros. Por una parte, el diseño de la arquitectura de un computador simple, a través de esquemáticos y/o lenguajes de descripción de hardware, reforzará la comprensión que de esta materia adquieren en las clases teóricas. Por otra, adquirirán experiencia en un campo y con unas herramientas, la circuitería FPGA y el desarrollo de soluciones con VHDL/Verilog, cuya demanda de profesionales es creciente.

Referencias

1. M. Rivas Pérez, M. Domínguez Morales, F. Gómez Rodríguez, A. Linares Barranco, G. Jiménez Moreno, A. Civit Balcells, Diseño e implementación de un simulador software basado en el procesador MIPS32, Enseñanza y Aprendizaje de Ingeniería de Computadores (5) (2015) 79–104.
   URL http://hdl.handle.net/10481/36571
2. A. N. de Evaluación de la Calidad y Acreditación, Libro blanco del título Grado en Ingeniería Informática, 2004.
   URL http://www.aneca.es/var/media/150388/libroblanco_jun05_informatica.pdf
3. ACM/IEEE, Computer Engineering Curricula, 2016.
   URL http://www.acm.org/binaries/content/assets/education/ce2016-final-report.pdf
4. J. Ortega Lopera, M. Anguita López, A. Prieto Espinosa, Arquitectura de computadores, Thomson, 2005.
5. D. J. Smith, VHDL & Verilog Compared & Contrasted - Plus Modeled Example Written in VHDL, Verilog and C, in: DAC, 1996.
6. T. L. Floyd, Digital Fundamentals, 10/e, Pearson Education, 2011.
7. G. Estrin, Reconfigurable computer origins: the ucla fixed-plus-variable (f+v) structure computer, IEEE Annals of the History of Computing 24 (4) (2002) 3–9. doi:10.1109/MAHC.2002.1114865.
8. J. M. Birkner, PAL, programmable array logic, handbook, Monolithic Memories, 1983.
9. D. Gembris, Generic Array Logic (GAL), Elektor Electronics (4) (1992) 24–25.
10. A. Putnam, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. M. Caulfield, A. Smith, J. Thong, P. Y. Xiao, D. Burger, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, A reconfigurable fabric for accelerating large-scale datacenter services, Communications of the ACM 59 (11) (2016) 114–122. doi:10.1145/2996868.
11. E. Durant, J. Impagliazzo, S. Conry, R. Reese, H. Lam, V. Nelson, J. Hughes, W. Liu, J. Lu, A. McGettrick, CE2016: Updated computer engineering curriculum guidelines, in: 2015 IEEE Frontiers in Education Conference (FIE), IEEE, 2015, pp. 1–2. doi:10.1109/FIE.2015.7344157.
12. J. Díaz Alonso, Diseño e implementación de un simulador software basado en el procesador MIPS32, Implementación de algoritmos en hardware: de la puerta NAND al bucle FOR (3) (2013) 95–105.
    URL http://hdl.handle.net/10481/26401
13. E. Ayeh, K. Agbedanu, Y. Morita, O. Adamo, P. Guturu, FPGA Implementation of an 8-bit Simple Processor, in: 2008 IEEE Region 5 Conference, IEEE, 2008, pp. 1–5. doi:10.1109/TPSD.2008.4562743.
14. W. Wójcik, J. Długopolski, FPGA-based multi-code processor, Computer Science @BULLET 14 (3). doi:10.7494/csci.2013.14.3.459.
15. P. J. Ashenden, P. J. Ashenden, 15 – Case Study: The DLX Computer System, in: The Designer's Guide to VHDL, 2002, pp. 373–458. doi:10.1016/B978-155860674-6/50017-4.

16. A. Calderón Mateos, F. García Carballeira, J. Prieto Cepeda, WepSIM: Simulador modular e interactivo de un procesador elemental para facilitar una visión integrada de la microprogramación y la programación en ensamblador, Enseñanza y Aprendizaje de Ingeniería de Computadores (6) (2016) 35–53.
URL http://hdl.handle.net/10481/41909

Bomberman modo multijugador

Hristo Ivanov[*] , Alberto Lorente[+], Verónica Chamorro[+], Alberto A. Del Barrio[1+], Guillermo Botella[1+]

[1] Departamento de Arquitectura de Computadores y Automática, Facultad de Informática de la Universidad Complutense de Madrid
Madrid, España

* hristo.idgh@gmail.com,+ {albertol, verocham, abarriog, gbotella}@ucm.es

Resumen. Este trabajo presenta el proyecto Bomberman, realizado en la asignatura Sistemas Empotrados Distribuidos, perteneciente a la titulación del Máster en Ingeniería Informática de la Universidad Complutense de Madrid. En este trabajo se describe e implementa una adaptación del conocido juego Bomberman en modo multijugador (dos jugadores). En esta versión los dos jugadores tratarán de salir de un laberinto o derrotar a su contrincante para ganar. Este proyecto utiliza dos placas de desarrollo S3CEV40 representado a cada jugador, una Raspberry Pi 2, dos cables hembra-hembra de 9 pines y dos adaptadores a 9 pines-USB para conectar cada cable desde cada placa S3CEV40 a la Raspberry.

Palabras Clave: Sistemas empotrados distribuidos, raspberry. ARM, Sprites.

Abstract. This paper presents the Bomberman project, carried out in the Distributed Embedded Systems subject, which belongs to the Computer Science Master that is taught at the Complutense University of Madrid. This work describes and implements an adaptation of the well-known Bomberman game in multiplayer mode (for two players). In this version, the two players will try to escape from a labyrinth or to destroy his opponent to win. This project use two S3CEV40 boards to represent the players, a Raspberry Pi 2, two female-to-female 9 pin cables and two 9 pins-to-USB adapters to connect each board to the Raspberry.

Keywords: Distributed embedded systems, Raspberry, S3CEV40, ARM, Sprites.

1 Introducción

La asignatura Sistemas Empotrados Distribuidos (SED) es una asignatura que aparece en el plan del Máster de Ingeniería Informática de la Universidad Complutense de Madrid (UCM) [1], implantado durante el curso 2013/2014. La idea fundamental del máster es proporcionar conocimientos adicionales a los estudiantes que les haga más competitivos en el mundo laboral. Con el objetivo de conseguir estas competencias las asignaturas del máster, y en concreto SED, contienen una alta carga práctica, de tal forma que los alumnos puedan enfrentarse a problemas desde una perspectiva más allá de la teórica. Por ello, además de las sesiones de laboratorio, la asignatura cuenta con un proyecto final [9-11] en el que los estudiantes demuestran la adquisición de conocimientos en el área de los Sistemas Empotrados Distribuidos, también conocida como *Computación Ubícua* o *Pervasive Computing* [6-8,17].

La importancia de los sistemas distribuidos puede observarse en distintos ámbitos de la Informática actualmente: desde el diseño de circuitos [13-15] hasta el *Wearable Computing* [12,16] o el *Internet of Things* (IoT) [18-19]. Es por ello fundamental que los alumnos aprendan y experimenten con la problemática que tales sistemas presentan.

El proyecto descrito en este artículo consiste en una adaptación del conocido juego Bomberman en modo multijugador sobre la placa de desarrollo S3CEV40 [3-5], la cual se utiliza también en las prácticas de SED.

El resto del artículo se organiza de la siguiente manera: la Sección 2 especifica el proyecto; en las Secciones 3 y 4 se presentan los aspectos generales del hardware y el diseño e implementación del proyecto, respectivamente; mientras que en las Secciones 5 y 6 se detallan aspectos más concretos sobre el envío de mensajes y la lógica interna del juego; y en la Sección 7 se presentan algunas de las pruebas realizadas. Finalmente, en la Sección 8 se resumen las conclusiones y describen las posibles líneas de trabajo futuro.

2 Especificación del juego

La versión de Bomberman que ha sido implementada, trata de dos personajes intentado salir de un laberinto, o bien derrotando a su contrincante haciendo que un elemento llamado *bomba* explote en las proximidades del jugador. Dicho laberinto está compuesto de diferentes elementos; *paredes* (elementos indestructibles) y por *rocas fracturadas* (elementos destructibles). Cada jugador es capaz de eliminar estas rocas con las mencionadas bombas. Cuando el jugador así lo desee podrá colocar una bomba en la posición en la que se encuentre. Tras un cierto tiempo la bomba detonará rompiendo aquellas rocas fracturadas que tenga a su alrededor. Además, si cualquiera

de los jugadores se encontrase en el rango de explosión de la bomba, este jugador será derrotado.

Para ganar el juego, el jugador deberá romper las rocas necesarias hasta encontrar la salida oculta por las mismas. Una vez visible la salida tan solo deberá caminar hasta salir por ella para ganar el juego. Idénticamente un jugador ganará la partida automáticamente si su contrincante muere a consecuencia de una bomba.

Para añadir más emoción al juego el mapa se generará de forma aleatoria en cada partida. De esta forma cada partida contará con una nueva distribución de las rocas fracturadas, la salida y la colocación de los dos jugadores. Esta colocación de los jugadores siempre deberá permitir a los mismos comenzar la partida sin estar bloqueados entre rocas destructibles, es decir, permitirá al jugador colocar bombas sin verse afectadas por ellas.

Un comienzo correcto de la partida sería pues cuando un jugador tiene al menos 2 casillas libres a su alrededor. Esto le permitiría colocar en una la bomba y huir a la otra para no verse afectada por la misma.

Dado que se trata de un juego, el objetivo es que ambos jugadores tengan una experiencia de juego en "tiempo real" por parte del sistema. Esta experiencia deberá compaginarse con el hecho de que no habrá un componente central con toda la información del juego.

3 Hardware empleado

El proyecto se desarrolla utilizando dos placas de desarrollo S3CEV40 para cada jugador y una Raspberry Pi 2. Una representación general del sistema empleado sería la que muestra la Figura 1.

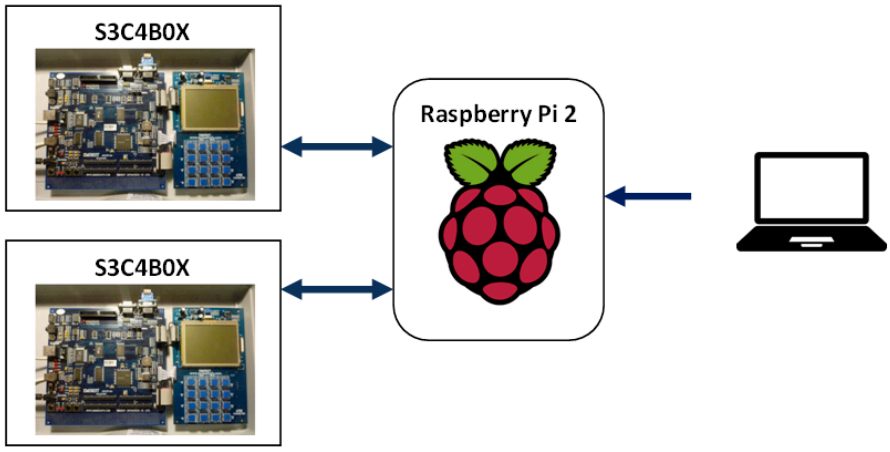


Figura 1. Esquema general del sistema

A su vez, cada elemento sería como los mostrados en las Figuras 2 y 3. Además, los conectores mostrados en la Figura 4 son necesarios para que las placas de desarrollo puedan comunicarse entre sí.

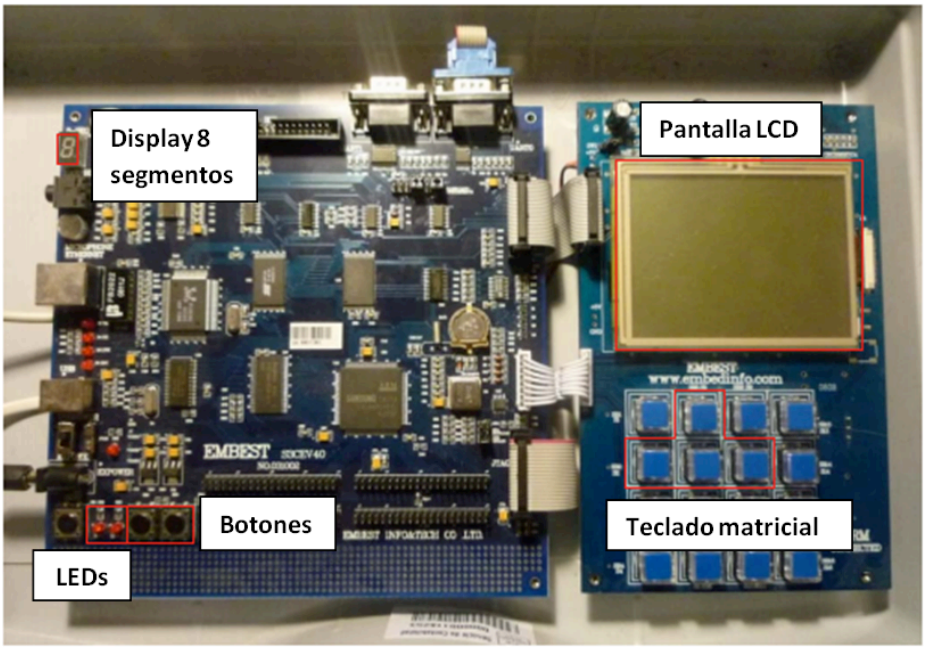


Figura 2. Placa de desarrollo S3CEV40 (*maletín*)

Figura 3. Raspberry Pi 2 que se utiliza como puente entre los dos maletines

Figura 4a. Dos cables hembra-hembra, de 9 pines.

Figura4b. Dos adaptadores a 9pin-USB para conectar los cables de la figura 4a a través de la UART de los maletines a la Raspberry.

La placa de desarrollo S3CV40 es un System on Chip (SoC) de Samsung basada en un procesador ARMTDMI. Dicha placa contiene varios periféricos, como puede observarse en la Figura 2, y contiene el microcontrolador S3C44B0X, que será el encargado de gestionar las interrupciones producidas por los periféricos.

Además, será necesario un equipo que funcione como host y desde el cual descargar el juego a los maletines y para conectarse a la Raspberry a través de SSH. Esto es así para poder ejecutar el código en python de la Raspberry.

4 Diseño e implementación

Dado que el componente central de la asignatura es la creación de sistemas distribuidos, el diseño del videojuego se centró principalmente en varias placas de desarrollo S3CEV40. Cada una tendrá una *copia* del juego de su jugador correspondiente. No habrá, por tanto, un sistema central en donde se almacene todo el estado del juego y sobre el cual los jugadores realicen acciones. Es decir, cada maletín es independiente y tan solo tiene un canal de entrada/salida de información a través de la UART.

Además, para mejorar la experiencia de juego se realizarán unos *sprites* en 2D que representarán a los jugadores, paredes, rocas fracturadas y salida. Para dar la sensación de movimiento, los jugadores tendrán varios *sprites* distintos.

Con el objetivo de comprender el funcionamiento del juego, se explican a continuación los elementos utilizados así como su función:

4.1 Botones

Están configurados mediante interrupciones y tienen 2 funcionalidades:

1. Inicializar el juego. El juego no comenzará hasta que los dos maletines hayan pulsado un botón para indicar que están preparados.

2. Colocar bombas. Cada jugador dispondrá de una bomba que se colocará cuando éste pulse un botón. Esta bomba se ubicará en la posición donde esté en ese momento el jugador. Concretamente, la bomba se depositará en la casilla donde esté situada la mayor parte del personaje.

4.2 LEDs

Los LEDs están configurados para informar visualmente al jugador del estado de la bomba.

- Si la bomba no está colocada los LEDs permanecen encendidos.
- Si la bomba está colocada los leds parpadean. Una vez explota la bomba vuelven a parpadear.

4.3 Teclado matricial

El teclado matricial no está configurado ni por interrupciones ni por espera activa. La forma de conocer qué tecla ha pulsado el jugador es mediante el uso de un temporizador. Cuando este temporizador llega a 0 realiza la lectura del registro correspondiente para conocer así cual ha sido la última pulsación del jugador. Este método es utilizado para mantener una sincronización de juego entre los dos maletines y para que un jugador no pueda realizar más acciones que otro, ya que ambos tienen la misma configuración de su temporizador.

Así el teclado matricial indica la dirección hacia la que se quiere mover el jugador. Siguiendo el estilo de las flechas de dirección de un teclado, las teclas del juego serían las que se muestran en la Figura 5.

Tecla 1 → Arriba
Tecla 4 → Izquierda
Tecla 5 → Abajo
Tecla 6 → Derecha

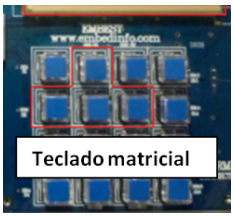


Figura 5. Teclado matricial para los movimientos

Como se puede ver en la Figura 5 las únicas teclas tienen efecto sobre el juego son las teclas 1, 4, 5 y 6. Se pueden entender como las flechas de dirección de un teclado convencional.

4.4 Display de 8 segmentos

Dicho display se utiliza a modo informativo para representar visualmente el último movimiento realizado por el jugador a través del teclado matricial.

4.5 Temporizadores o *timers*

En la implementación del juego se han utilizado dos timers distintos, cada uno configurado de forma idéntica en cada maletín.

Timer 0

Es el timer de juego principal. Configurado en modo *autoreload*, se utiliza para mantener la sincronización del juego. Este *timer* impide al jugador realizar más movimientos de los permitidos en un espacio de tiempo definido. Cada vez que este *timer* se active se encargará de leer la tecla pulsada por el jugador en el teclado matricial. Según sea la pulsación del jugador con la tecla 1, 4, 5 o 6 moverá el jugador en su sentido correspondiente. En otras palabras, cada vez que se active el *timer* 0 el jugador podrá llevar a cabo una única acción de movimiento. Además, mostrará en el display de 8 segmentos la tecla pulsada.

Timer 2

Este timer se encargará de gestionar las bombas. Cada vez que un jugador coloque una bomba pulsando un botón, éste activará el timer 2 en modo *manual update*. Cuando el timer llegue a 0 la bomba explotará, liberando así el camino de rocas fragmentadas o incluso pudiendo matar a los distintos jugadores.

4.6 Pantalla LCD

Se utiliza para mostrar todo el desarrollo del juego. Se encargará de dibujar el escenario con todas sus paredes y rocas fragmentadas, así como los dos jugadores y las bombas.

La pantalla LCD funciona mostrando el contenido de un array de enteros sin signo de 8 bits (*uint8*). Este array (llamado buffer en el código) tiene un tamaño de 320 píxeles de ancho por 240 de alto. Además, dentro de cada posición del array (8 bits) se representan dos píxeles distintos. Los 4 bits de las posiciones superiores representan el color del píxel izquierdo y los 4 bits de posiciones inferiores representan el color del píxel derecho.

1111,1110,1101,1100,

1011,1010,1001,1000,

0111,0110,0101,0100,

0011,0010,0001,0000

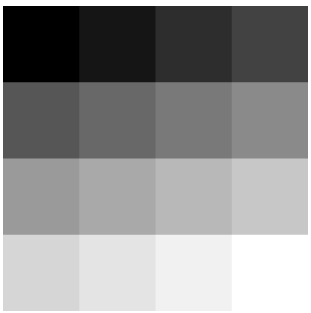


Figura 6. Codificación de escala de grises en la pantalla y su valor en binario.

Esto nos deja una profundidad de color en cada píxel de 4 bits. Con esta codificación podemos representar una escala de colores de gris (gray-scale) como la que puede verse en la Figura 6.

Para trabajar más fácilmente con el LCD, se ha dividido la pantalla de 320x240 píxeles en casillas de 16x16 dando como resultado una pantalla de 20x15 casillas lógicas de juego. Aprovechando estas casillas se pintarán los distintos componentes del juego: personajes, bombas, rocas, etc. Además, se han creado distintos sprites para pintar en estas casillas con el fin de mejorar la experiencia de juego.

La función visual de un juego de estas características es muy importante, por lo que para pintar el LCD no se recorre todo el array cada vez que se desea cambiar la visualización de la pantalla. Por el contrario, se utilizan refrescos locales, es decir, no se modifica el contenido al completo del array. Tan solo se refrescan casillas lógicas de 16x16 mencionadas en el apartado anterior. Para ello se utilizan las funciones *lcd_draw16x16* y *lcd_clear16x16* capaces de pintar los sprites y borrarlos más rápidamente, evitando aquellos píxeles que ya estén en blanco.

4.7 UART

La UART, o Universal Asynchronous Reciver-Transmitter, es el elemento utilizado para la comunicación entre las dos placas de desarrollo S3CEV40.

En nuestro proyecto se han configurado las dos UARTs presentes en la placa (UART0 y UART1). Sin embargo, para la comunicación específica entre los 2 maletines únicamente se utiliza la UART1. Ambas UARTs han sido configuradas mediante interrupciones en modo lectura (*Rx*). Además, se ha activado la cola FIFO (*Rx*) presente en la UART para evitar perder mensajes enviados de una placa a otra. El tratamiento de mensajes se hace a nivel de byte.

Por último, se ha definido un protocolo propio de transmisión en el cual se define cómo son los mensajes que se envían los jugadores para actualizar sus escenarios, es decir, la pantalla LCD. La Sección 5, especificará en profundidad dicho protocolo.

4.8 Raspberry Pi 2

Entre las dos placas S3CEV40, conectada por los cables hembra-hembra y los adaptadores, se coloca la Raspberry. Su función es únicamente reenviar los mensajes que le llegan a través de un puerto serie por el otro puerto serie.

El único código utilizado en la Raspberry tiene por finalidad la creación de dos puertos serie (*io1* e *io2*) y la creación de dos threads (*tr1* y *tr2*), los cuales serán los encargados de leer de un puerto serie el mensaje recibido, mostrarlo y reenviarlo por el otro puerto serie.

La Raspberry se ha utilizado fundamentalmente para mitigar los problemas de potencia de las UARTs de las placas S3CEV40. Sin la Raspberry se perdían o malinterpretaban varios mensajes tras cierto tiempo de juego, con lo que concluimos que la potencia de señal transmitida que transmiten las placas a través de sus UART no es lo suficientemente elevada como para que la otra placa detecte el mensaje transmitido con claridad. Además, el uso de esta placa se pensó como posible solución para escalar el juego a N jugadores, ya que con 2 jugadores los maletines consumían casi todos sus recursos con la lógica del juego y el refresco.

4.9 Sprites

Los sprites son conocidos en el mundo de los videojuegos como mapas de bits dibujados generalmente por hardware y utilizados para dotar al videojuego de un atractivo adicional.

Para nuestro proyecto se han generado distintos sprites para los distintos objetos presentes en la escena. Estos sprites tienen un tamaño de 16x16 píxeles y cada píxel o posición de esa matriz tiene un valor en [0,15] o en binario [0000,1111].

Figura 7. Sprites correspondientes al estado *Idle* (0), y el *movimiento* (2 y 3) del jugador.

Para representar el movimiento del jugador alternamos la presentación de los sprites en secuencia 0-1-0-2-0 según están representados en la Figura 7. Esto genera una sensación de movimiento al avanzar el personaje. Para distinguir un segundo jugador se han retirado los cuernos al sprite de la Figura 7. De esta forma obtenemos dos jugadores *demon* y *human*.

Una vez seleccionada la imagen generamos la matriz gracias a un programa externo creado por nosotros mismos. Este programa leerá el color establecido en cada píxel haciendo una conversión a la tonalidad de gris correspondiente. Además creará una matriz como la de la Figura 8 que copiaremos dentro de nuestro código fuente.

```
0b1110,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b1110,
0b1110,0b1110,0b0000,0b0000,0b0000,0b1101,0b1101,0b1101,0b1101,0b1101,0b1101,0b0000,0b0000,0b0000,0b1110,0b1110,
0b1110,0b1110,0b1110,0b1110,0b1101,0b0011,0b0011,0b0011,0b0011,0b0011,0b0011,0b1101,0b1110,0b1110,0b1110,0b1110,
0b0000,0b1110,0b1110,0b1111,0b0011,0b0010,0b0001,0b0010,0b0010,0b0001,0b0010,0b0011,0b1111,0b1110,0b1110,0b0000,
0b0000,0b0000,0b1111,0b1111,0b0011,0b0001,0b1110,0b0001,0b0001,0b1110,0b0001,0b0011,0b1111,0b1111,0b0000,0b0000,
0b0000,0b0000,0b0000,0b1101,0b0100,0b0001,0b1110,0b0001,0b0001,0b1110,0b0001,0b0100,0b1101,0b0000,0b0000,0b0000,
0b0000,0b0000,0b0000,0b1101,0b1011,0b0010,0b0001,0b0010,0b0010,0b0001,0b0010,0b1011,0b1101,0b0000,0b0000,0b0000,
0b0000,0b0000,0b0000,0b0000,0b1101,0b1011,0b1011,0b1011,0b1011,0b1011,0b1011,0b1101,0b0000,0b0000,0b0000,0b0000,
0b0000,0b0000,0b0000,0b1101,0b0011,0b1101,0b1101,0b1101,0b1101,0b1101,0b0011,0b1101,0b0000,0b0000,0b0000,
0b0000,0b0000,0b1101,0b0011,0b1011,0b0100,0b0011,0b1101,0b1101,0b0011,0b0100,0b1011,0b0011,0b1101,0b0000,0b0000,
0b0000,0b1101,0b0011,0b1011,0b1101,0b1011,0b0100,0b0100,0b0100,0b0100,0b1011,0b1101,0b1011,0b0011,0b1101,0b0000,
0b0000,0b1101,0b0100,0b0100,0b1101,0b1011,0b1011,0b1011,0b1011,0b1011,0b1011,0b1101,0b0100,0b0100,0b1101,0b0000,
0b0000,0b0000,0b1101,0b1101,0b1101,0b1100,0b1100,0b1100,0b1100,0b1100,0b1100,0b1101,0b1101,0b1101,0b0000,0b0000,
0b0000,0b0000,0b0000,0b0000,0b0000,0b1101,0b0101,0b1101,0b1101,0b0101,0b1101,0b0000,0b0000,0b0000,0b0000,0b0000,
0b0000,0b0000,0b0000,0b0000,0b1101,0b1110,0b1110,0b1101,0b1101,0b1110,0b1110,0b1101,0b0000,0b0000,0b0000,0b0000,
0b0000,0b0000,0b0000,0b0000,0b1101,0b1111,0b1111,0b1101,0b1101,0b1111,0b1111,0b1101,0b0000,0b0000,0b0000,0b0000
```

Figura 8. Matriz de 16x16 elementos enteros en donde cada uno representa el color de un píxel.

0b indica que los siguientes dígitos especifican un número en binario. Para el segundo jugador el sprite es el mismo, pero sin cuernos (Human).

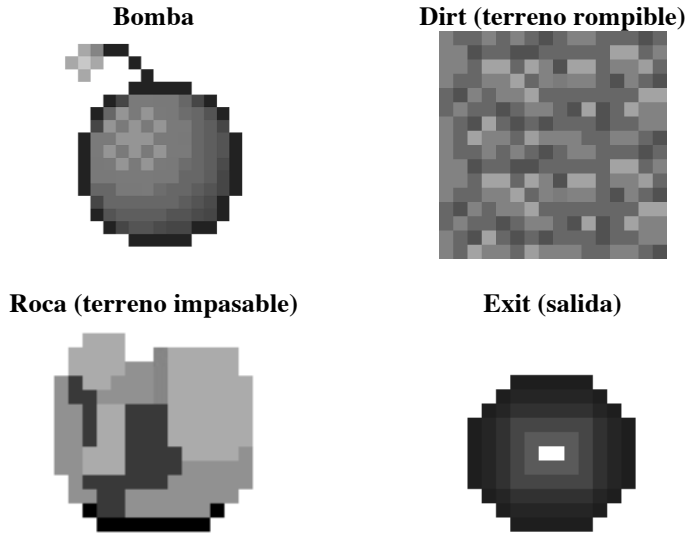


Figura 9. Otros sprites generados para la pantalla LCD.

Las matrices generadas como la de la Figura 8 serán añadidas como arrays en el código, representando todos los sprites de las Figura 7 y 9.

5 Protocolo

Las placas S3CEV40 se comunican por la UART1 enviando bytes de uno a otro para que actualicen el escenario presentado en la pantalla LCD. A continuación, se describen los diferentes mensajes que los jugadores pueden intercambiar.

Tabla 1. Mensaje correspondiente a la posición del jugador.

Bit	7	6	5	4	3	2	1	0
Byte 1	1	0	0	PosX(8)	PosX(7)	PosY(7)	Sprite(1)	Sprite(0)
Byte 2	0	PosX(6)	PosX(5)	PosX(4)	PosX(3)	PosX(2)	PosX(1)	PosX(0)
Byte 3	0	PosY(6)	PosY(5)	PosY(4)	PosY(3)	PosY(2)	PosY(1)	PosY(0)

Dado que la pantalla tiene un tamaño de 320 x 240 píxeles, y el jugador puede estar en cualquiera de estos píxeles, necesitamos 9 bits para la *posición X* y 8 bits para la *posición Y*. Con los dos bits de *Sprite* indicamos el sprite que debe ser utilizado

para pintar el movimiento del jugador [0,2]. Toda esta información aparece resumida en la Tabla 1.

Tabla 2. Mensaje correspondiente a la posición de la bomba.

Bit	7	6	5	4	3	2	1	0
Byte 1	1	0	1	PosX(4)	PosX(3)	PosX(2)	PosX(1)	PosX(0)
Byte 2	0	_	_	_	PosY(3)	PosY(2)	PosY(1)	PosY(0)

Dado que la bomba únicamente puede estar en las casillas de 16x16, tan solo son necesarios 5 bits (320/16 = 20) para la *posición X* y 4 bits (240/16 = 15) para la *posición Y*. Cuando el jugador pulsa un botón, este mensaje es enviado por una placa a la otra para que la segunda pinte en su pantalla LCD la bomba en su correspondiente posición. El contenido de este mensaje se detalla en la Tabla 2.

Tabla 3. Mensaje que contiene la semilla del juego.

Bit	7	6	5	4	3	2	1	0
Byte 1	1	1	0	0	0	0	0	0
Byte 2	0	seed(6)	seed(5)	seed(4)	seed(3)	seed(2)	seed(1)	seed(0)

Para generar el mapa de juego de forma aleatoria se utiliza una semilla de juego. Cuando los dos jugadores pulsan el botón de inicio de juego ambos intercambian la semilla. Como se utilizan 7 bits para la semilla, tenemos 2^7 juegos posibles. La especificación de este mensaje se muestra en la Tabla 3.

Tabla 4. Mensaje de GameOver.

Bit	7	6	5	4	3	2	1	0
Byte 1	1	1	0	0	0	0	0	1

El mensaje especificado en la Tabla 4 indica el fin del juego. Este mensaje se genera cuando el jugador ha muerto al explotar una bomba a su lado, por lo que este jugador ha perdido la partida. Por tanto, el jugador que recibe este mensaje ha ganado.

Tabla 5. Mensaje de GameWin.

Bit	7	6	5	4	3	2	1	0
Byte 1	1	1	0	0	0	0	1	0

El mensaje mostrado en la Tabla 5 indica el fin del juego. Este mensaje se genera cuando el jugador alcanza la salida, por lo que este jugador gana la partida. Por el contrario, el jugador que recibe este mensaje ha perdido.

Tabla 6. Mensaje correspondiente a la explosión de la bomba.

Bit	7	6	5	4	3	2	1	0
Byte 1	1	1	1	PosX(4)	PosX(3)	PosX(2)	PosX(1)	PosX(0)
Byte 2	0	_	_	_	PosY(3)	PosY(2)	PosY(1)	PosY(0)

El mensaje “Explosión Bomba” se muestra en la Tabla 6. Dado que la bomba únicamente puede estar en las casillas de 16x16, tan solo son necesarios 5 bits (320/16 = 20) para la *posición X* y 4 bits (240/16 = 15) para la *posición Y*. Análogamente al mensaje "Posición Bomba" de la placa que controla la bomba será responsable de retransmitir a la otra placa que la bomba ha explotado. Este mensaje se transmitirá cuando el timer 1 llegue a 0 y será el responsable de eliminar la bomba de la pantalla LCD y de procesar si la bomba a afectado al jugador.

6 La lógica del juego

Para procesar los mensajes se utiliza una máquina de estados. El primer bit de cada byte está reservado. Cuando este bit toma el valor ‘1’, indica el principio de un nuevo mensaje (comando), en otro caso este bit vale ‘0’ (dato). Este primer bit nos permite sobrevenir la posible pérdida de bytes que pueda darse. Al recibir un byte con un ‘1’ inicial empezamos la recepción de un nuevo mensaje. Por el contenido de este byte podemos identificar el mensaje (bits 6 y 5). Dependiendo del mensaje, esperamos un número variable de bytes que empiezan por un ‘0’. Si, por contrario, recibimos un byte con un ‘1’ inicial, la recepción del mensaje actual se descarta y se procede al procesado del nuevo mensaje.
La máquina de estados quedaría entonces representada por la Figura 11.

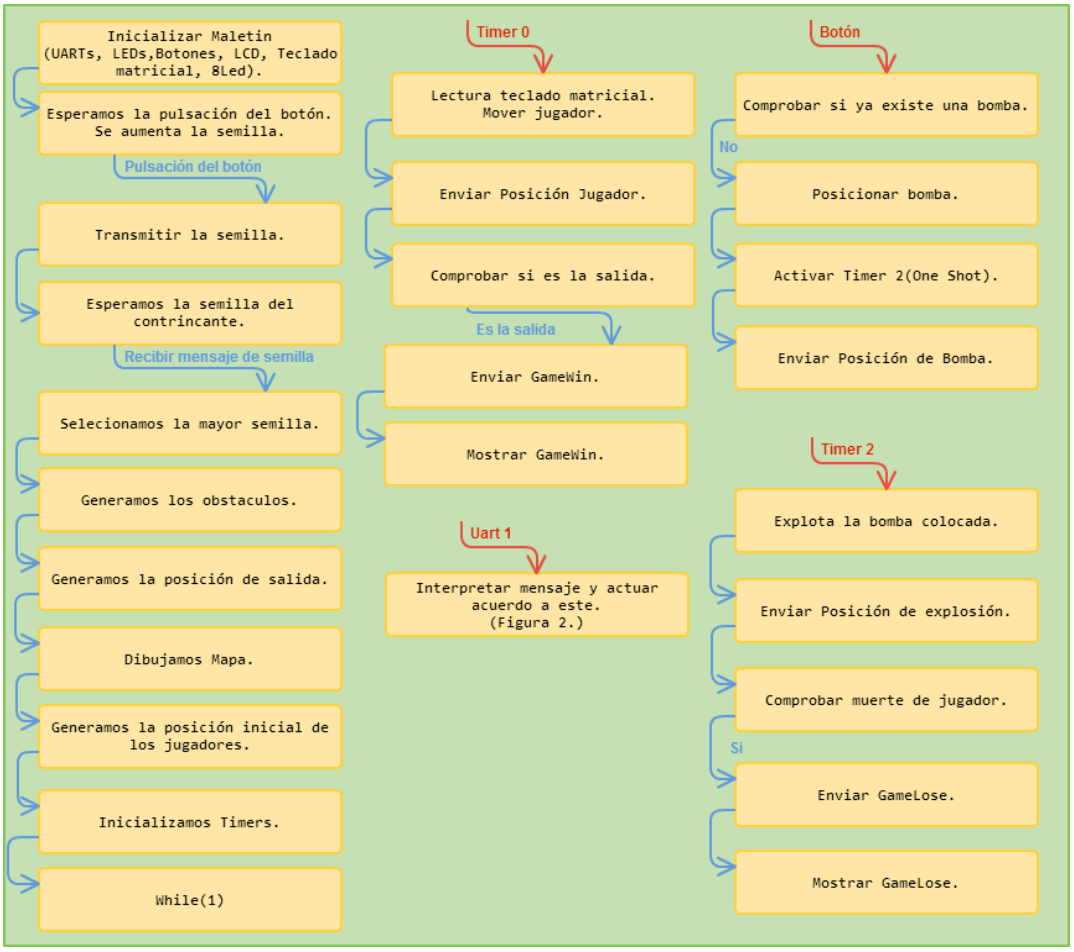


Figura 11. Lógica del juego.

Paralelamente se utiliza la UART para enviar mensajes a la otra placa. Estos mensajes recibirán un tratamiento distinto dependiendo de cuáles sean:

- En caso de recibir un mensaje de semilla la placa entenderá que el otro jugador está listo para empezar el juego. En caso de que ambos lo estén se generará el mapa en base a las semillas transmitidas entre ambos.
- En el caso de recibir un mensaje de "posición jugador" la placa se encargará de mover al jugador contrario en la pantalla LCD.
- En el caso de recibir un mensaje de "colocar bomba" la placa tan solo la pintará en la pantalla LCD.
- En el caso de recibir un mensaje de "explosión bomba" la placa se encargará de eliminar visualmente en la pantalla la bomba y las paredes rompibles adyacentes a la misma. También se encargará de procesar si la explosión de la bomba ha podido afectar al jugador. En el caso de que así sea la placa enviará un mensaje de

"GameOver" indicando que el jugador en su placa ha perdido y el contrario ha sido victorioso.

7 Experimentos

El desarrollo del videojuego y la experimentación han sido dos estados cíclicos en el desarrollo del proyecto. Además esta experimentación nos permitió tomar decisiones importantes de cara al desarrollo como la creación de una maquina de estados en cada placa o la sincronización a través de un timer en el teclado matricial y no por interrupciones.

Sin embargo, los experimentos más importantes y más satisfactorios se produjeron al final del proyecto. Al ser un videojuego el papel que más resalta es la visualización del mismo.

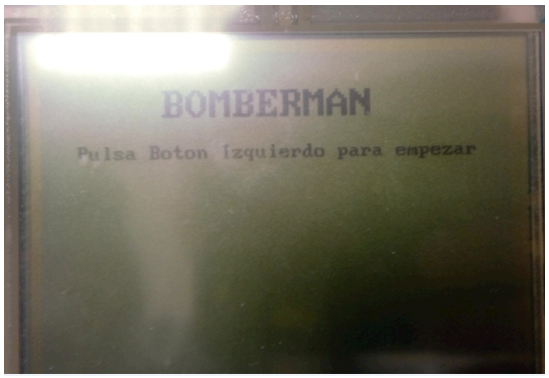


Figura 12. Inicio del videojuego.

En la Figura 12 puede verse cómo se escribe el mensaje inicial en la pantalla LCD, trabajando además con la intensidad del negro en las letras así como con su tamaño.

Figura 13. Juego en proceso.

En la Figura 13 puede observarse cómo se muestra el laberinto del juego durante una partida.

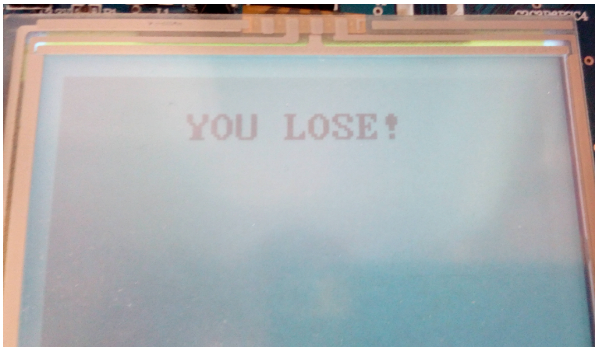


Figura 14. Fin del videojuego con derrota

Finalmente, en la Figura 14 podemos ver el mensaje que mostraría el LCD al jugador que hubiera perdido el juego.

8 Conclusiones

En este artículo se ha presentado una plataforma distribuida y heterogénea para implementar el juego del Bomberman para dos jugadores. Las placas de desarrollo S3CEV40 se han empleado para implementar a cada jugador, mientras que la Raspberry Pi 2 ha funcionado como puente para comunicar ambas placas.

Además de la lógica de juego, se ha desarrollado un protocolo de comunicación para informar a cada jugador sobre el estado del otro y así poder avanzar en la ejecución del juego.

Aunque la versión presentada es plenamente funcional, en el futuro podrían añadírsele las siguientes características:

- Creación de un menú de juego: una vez el juego se finaliza no hay ninguna forma de volver a relanzarlo. Todas las interrupciones y timers se desactivan una vez ha finalizado el juego.
- Sistema de puntuaciones: asignar un valor por cada bloque de tierra eliminado o por el tiempo en salir del laberinto. Para ello se pensó inicialmente en la utilización de la EEPROM como almacén de puntuaciones que pudiera mostrarse también desde el menú de juego.
- Escalar el juego a N jugadores. Habría que evaluar la carga computacional asociada para procesar los mensajes de todos los jugadores, y muy posiblemente descargar parte del control sobre la Raspberry Pi.

Referencias

1. Máster en Ingeniería Informática de la Universidad Complutense de Madrid, http://informatica.ucm.es/estudios/2016-17/master-ingenieriainformatica
2. https://github.com/AlbertoLorente92/master-ucm-SED. Repositorio de la asignatura y del Proyecto final donde está el juego.
3. Embest S3CEV40 EVB User Guide, http://www.vas.co.kr/products/support/S3CEV40_UserGuide.pdf
4. http://datasheets.chipdb.org/Samsung/S3C44B0X.pdf. Manual de la placa S3C44B0X de Samsung.
5. http://www.fdi.ucm.es/profesor/mendias/PSyD/PSyD.html. Manuales para el uso de la placa S3C44B0X y la pantalla LCD.
6. Chtourou, S. et al., "Evolution of robot programming, towards the Ubiquitous Computing era," *Individual and Collective Behaviors in Robotics (ICBR), 2013 International Conference on* , vol., no., pp.44,48, 15-17 Dec. 2013
7. Daoqing Sun, "Researches to the trusted ubiquitous computing," *Electronics, Communications and Control (ICECC), 2011 International Conference on* , vol., no., pp.433,437, 9-11 Sept. 2011
8. Kortuem, G. et al., "Smart objects as building blocks for the Internet of things," *Internet Computing, IEEE* , vol.14, no.1, pp.44,51, Jan.-Feb. 2010
9. D. Lora et al., "Sistema de Seguridad Basado en una Plataforma Heterogénea Distribuida", Enseñanza y Aprendizaje de Ingeniería de Computadores, 5: 29-38 (2015).
10. F. Parrales et al. "Una Orquesta Sinfónica como Ejemplo de Aplicación de un Sistema Empotrado Distribuido", Enseñanza y Aprendizaje de Ingeniería de Computadores, 5: 115-124 (2015).
11. I.M. Laclaustra et al. "Sistema Domótico Distribuido para Controlar el Riego y el Aire Acondicionado en el Hogar", Enseñanza y Aprendizaje de Ingeniería de Computadores, 6: 87-102 (2016).
12. R. Braojos et al., "Ultra-low power design of wearable cardiac monitoring systems," *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, San Francisco, CA, 2014, pp. 1-6.
13. A. A. Del Barrio et al., "A Distributed Clustered Architecture to Tackle Delay Variations in Datapath Synthesis," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 3, pp. 419-432, March 2016.
14. A. A. Del Barrio et al., "A Distributed Controller for Managing Speculative Functional Units in High Level Synthesis," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 3, pp. 350-363, March 2011.
15. Jesús Martín Alonso et al., 2016. A distributed HW-SW platform for fireworks. In Proceedings of the *Summer Computer Simulation Conference* (SCSC '16). Society for Computer Simulation International, Montreal, Article 17 , 7 pages.
16. A. Dias Junior et al., "Estimation of Blood Pressure and Pulse Transit Time Using Your Smartphone," *Digital System Design (DSD), 2015 Euromicro Conference on*, Funchal, 2015, pp. 173-180.
17. Marwedel, P. *Embedded system design*. Kluwer, 2003.
18. C. Perera et al., "Context Aware Computing for The Internet of Things: A Survey," in *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 414-454, First Quarter 2014.
19. A. Zanella et al., "Internet of Things for Smart Cities," in *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22-32, Feb. 2014.

Sistema de control digital de bajo coste de un motor de corriente continua para usos didácticos

G. Olivares[1], A. Olivares[2], F. Gómez[1], M. Damas[1].

1) Departamento de Arquitectura y Tecnología de Computadores. ETSI Informática y de Telecomunicación. Universidad de Granada
{gonzalo, aolivares, frgomez, mdamas}@ugr.es
2) Nazaríes Information Technologies S.L. (http://www.nazaries.com)
alberto.olivares@nazaries.com

Resumen. Este trabajo describe el desarrollo de un conjunto de experiencias prácticas de diseño e implementación de algunas técnicas de simulación y control digital de un motor de corriente continua llevadas a cabo con elementos físicos de muy bajo coste. El sistema de control desarrollado se está empleando en las prácticas de las asignaturas impartidas por el Área de Ingeniería de Sistemas y Automática y el Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada.

Palabras Claves: Control digital, Motor de corriente continua, Mecatrónica, Bajo Coste, Docencia Experimental.

Abstract. This work describes the development of a set of practical experiences in the design and implementation of various simulation and digital control techniques of a DC motor carried out with very low cost elements. The control system developed is being used in the practices of the subjects taught by the Systems Engineering and Automation Area and the Department of Architecture and Computer Technology of the University of Granada.

Keywords: Digital-Control, DC motor, Mechatronics, Low-cost, Experimental Learning.

1 Introducción

El motor de corriente continua es uno de los principales componentes que se utilizan tanto en Robótica como en Mecatrónica. Conocer bien su comportamiento, modelado y métodos de control es muy importante para avanzar y abordar más adelante el control de sistemas mecánicos más complejos. Además, controlar la posición angular del eje de un motor (servo-motor) o la velocidad angular del mismo, permite de una forma sencilla, experimentar y asimilar correctamente un conjunto amplio de conceptos relacionados con el desarrollo de sistemas discretos de control, tales como: funciones de transferencia continuas y discretas, estabilidad, controlabilidad, observabilidad, controladores PID, control LQR, representación en el espacio de

estados, control por ubicación de polos, controladores-observadores, o bien simplemente experimentar con filtros digitales, análisis de ruido, etc. Existen muchos equipos de prácticas con motores de corriente continua, que se incluyen en los catálogos de diversos fabricantes [1][2]. En la mayoría de los casos, su coste es elevado, y con bastante frecuencia, el alumno tiene que aprender herramientas específicas de software propietario del fabricante. A veces se utiliza una nomenclatura distinta a la que el alumno ha aprendido en las clases teóricas, o bien se hace hincapié en conceptos que no siempre son esenciales. Por tanto, lo que la mayoría de las veces se dispone en el laboratorio son sistemas didácticos cerrados, que no permiten que el alumno experimente fácilmente, de forma autónoma y a bajo coste, lo que en las clases teóricas ha aprendido. Por supuesto, ello dificulta que el alumno se pueda llevar el material incluso a casa para experimentar con más tranquilidad y ensayar por si mismo las soluciones propuestas en clase de teoría. Por tanto, pretendemos diseñar un pequeño equipo, muy barato, fácil de transportar y de reparar, con un conjunto básico de guiones de prácticas, y que a partir de ahí el alumno experimente después por sí sólo, y use posteriormente estas técnicas en maquetas electromecánicas más complejas que integren motores DC. En este trabajo, a continuación se describe el hardware que se propone usar,

2 Descripción del hardware

Dado que lo que pretendemos es realizar un sistema de control digital para la regulación de la posición y también la velocidad angular de un motor de corriente continua, proponemos utilizar un motor de baja potencia y bajo coste que incorpore además un codificador incremental. Hemos seleccionado el motor Namiki 22CL-3501PG de 12 VDC, con una relación de engranajes de 80:1 [3]. El codificador dispone de dos opto-acopladores, tal y como puede apreciarse en la figura 1, en la que además se muestra una fotografía del motor. Permite medir la posición angular (en los dos sentidos), con 640 pulsos por vuelta, lo que equivale a una resolución de 0.0098175 radianes (0.56 grados).

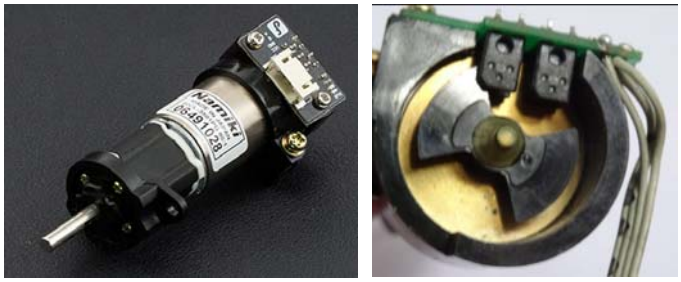


Fig 1. Motor y detalle del codificador incremental

Como controlador se utiliza una tarjeta Arduino Mega 2560 R3 [4]. Podríamos haber utilizado alguna otra tarjeta Arduino de menor coste, sin embargo esta es más apropiada para poder utilizarse en tiempo real con el modo *External* de tiempo real de Simulink [4]. Para suministrar la potencia adecuada, utilizaremos la tarjeta Arduino

Motor Shield [5], que incorpora el circuito integrado L293 (Puente H dual), mediante el cual se pueden controlar la velocidad angular de hasta dos motores de corriente continua, mediante dos salidas PWM. Admite un consumo de corriente de hasta 2 amperios. En la figura 2 se muestra una fotografía del sistema completo.

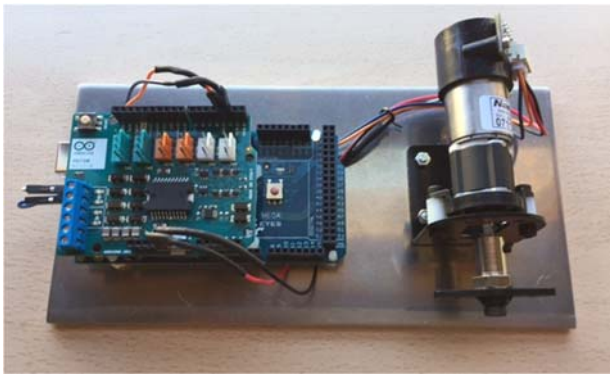


Fig 2. Fotografía del sistema completo

3. Objetivos de aprendizaje

Presentamos a continuación una lista de tareas experimentales desarrolladas con el equipo anteriormente descrito:

1. Obtención experimental de parámetros del motor.
2. Funciones de transferencia continua y discreta: cálculo teórico y experimental.
3. Representación en el espacio de estados continuo y discreto del modelo del motor DC.
4. Diseño y simulación de Control PID discreto de posición y de velocidad angular.
5. Estudio del codificador incremental del motor y desarrollo de software en C++ y Simulink para la medida de ángulo y velocidad angular.
6. Control de la velocidad del motor en lazo abierto.
7. Implementación en C++ y Simulink de controlador PID discreto.
8. Diseño y programación de control de posición y velocidad angular por realimentación en el espacio de estados y ubicación de polos.
9. Control por realimentación en el espacio de estados, con integrador.
10. Control LQR.
11. Diseño, simulación e implementación en tiempo real de controlador-observador
12. Modelado y simulación del sistema con SolidWorks y Sinmechanics.

Todas estas unidades didácticas han sido planificadas y ensayadas con éxito.

3 Resultados obtenidos

A continuación se describen algunos de los módulos experimentales y los resultados obtenidos.

3.1 Control del motor en lazo abierto.

En la figura 3 se presenta un módulo realizado con Simulink para el control de velocidad del motor. Se puede seleccionar el sentido de giro mediante la activación de un pin digital; asimismo, la velocidad se regula con una salida PWM.

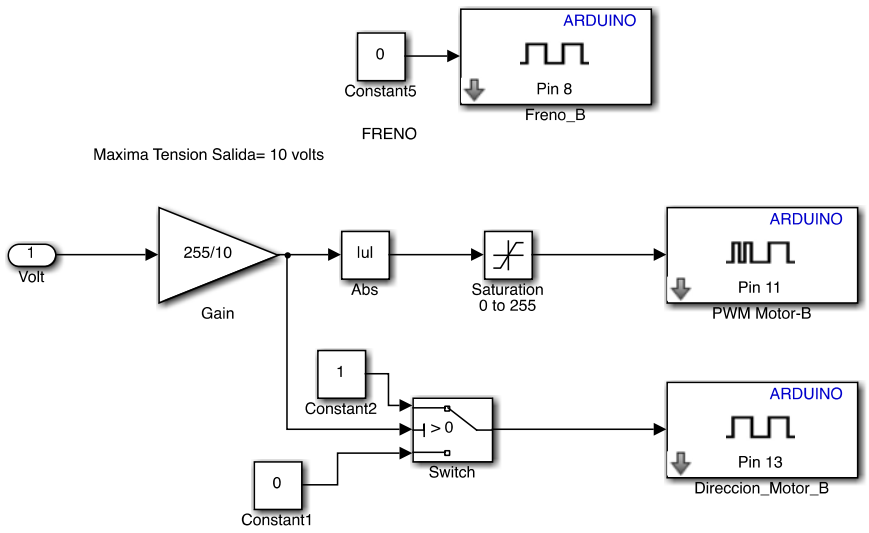


Fig 3. Control de la velocidad y sentido de giro del motor en lazo abierto

3.2 Control PID

Los parámetros del controlador PID (constantes proporcional, diferencial e integral) se diseñan a partir de la función de transferencia del motor y en función de la respuesta deseada (sobrepasamiento máximo y tiempo de asentamiento). Se han utilizado los siguientes procedimientos:

- Diseño por ubicación de los polos de la respuesta final deseada.
- Método de Ziegler Nichols.
- Procedimiento de auto-tuning de Matlab y de Simulink con simulación previa de resultados.

Una vez diseñado el controlador PID, este se implementó de nuevo con Simulink, visualizando en tiempo real la respuesta ante un cambio de consigna periódico (ver figura 4).

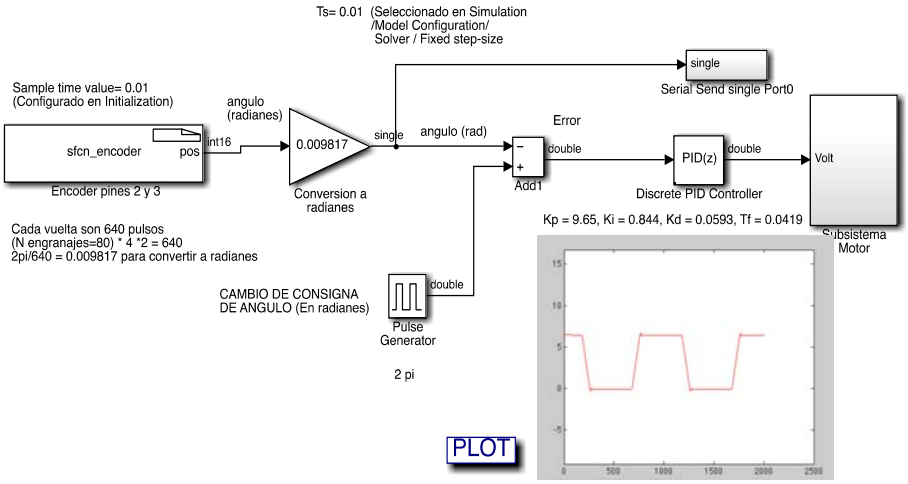


Fig 4. Control PID de posición angular realizado con Simulink y respuesta obtenida

3.3 Control LQR

A partir de las ecuaciones diferenciales que definen el comportamiento físico de un motor de corriente continua:

$$Ki = J\ddot{\theta} + b\dot{\theta} \quad \Rightarrow \quad \ddot{\theta} = -\frac{b}{J}\dot{\theta} + \frac{K}{J}\,\mathrm{i}$$

$$V - K\dot{\theta} = Ri + L\dot{i} \quad \Rightarrow \quad \dot{i} = -\frac{K}{L}\dot{\theta} - \frac{R}{L}i + \frac{V}{L}$$

donde i es la corriente del motor, J es el momento de inercia del eje, K la constante de fuerza electromotriz, con el mismo valor que la constante de par motor (K=Kt=Ke), R la resistencia, L la autoinducción, b el coeficiente de fricción, y V la entrada de control (voltaje aplicado al motor), se obtiene el modelo continuo del motor en el espacio de estados:

$$\begin{pmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{i} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & -b/J & K/J \\ 0 & -K/L & -R/L \end{pmatrix} \begin{pmatrix} \theta \\ \dot{\theta} \\ i \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1/L \end{pmatrix} V$$

y también la funciones de transferencia con respecto a la posición angular:

$$\frac{\theta(s)}{V(s)} = \frac{K}{s[JLs^2 + (RJ + Lb)s + (Rb + K^2)]}$$

y la función de transferencia con respecto a la velocidad angular:

$$\frac{\omega(s)}{V(s)} = \frac{K}{JLs^2 + (RJ + Lb)s + (Rb + K^2)}$$

Con la representación en el espacio de estados, hemos implementado un sistema de control de posición angular por realimentación en el espacio de estados (ver figura 5), donde las constantes de realimentación se calculan mediante un Regulador Cuadrático Lineal (LQR) discreto que optimiza la respuesta de control, minimizando además el coste energético [7]. Se trata de obtener las constantes K de realimentación en el espacio de estados discreto, que minimizan la función de coste:

$$J = \sum_{k=0}^{\infty} [x_k^T Q x_k + u_k^T R u_k]$$

Para ello se realiza la elección apropiada de los parámetros de peso Q, R, utilizando la regla de Bryson. Se ha discretizado el modelo, con un periodo de muestreo de 0.01 segundos. En la figura 6, se puede apreciar que las respuestas simulada y real se aproximan muy bien; sin embargo, hemos podido comprobar que la medida de la corriente eléctrica del motor, no se realiza con precisión con la tarjeta *Arduino Shield Motor*, a pesar de que dispone de una salida analógica para ello.

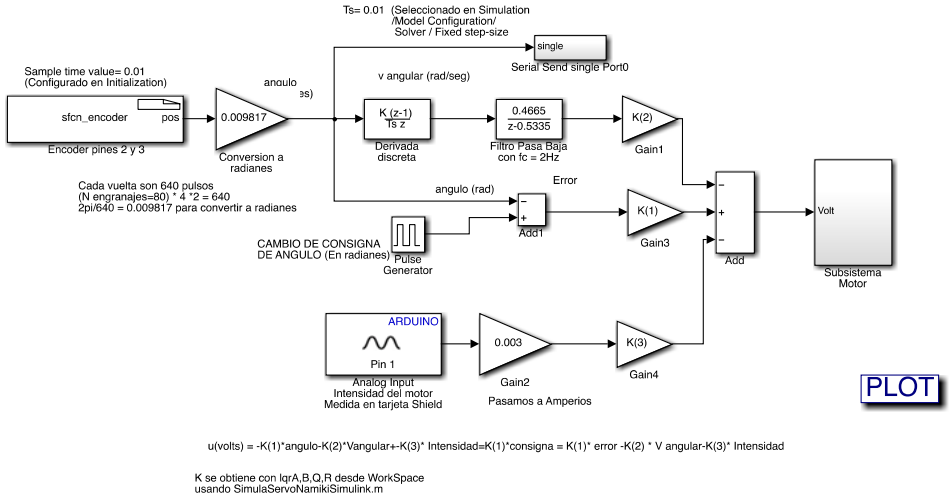


Fig 5. Diseño del control LQR del servo motor, utilizando la biblioteca de Arduino para Simulink.

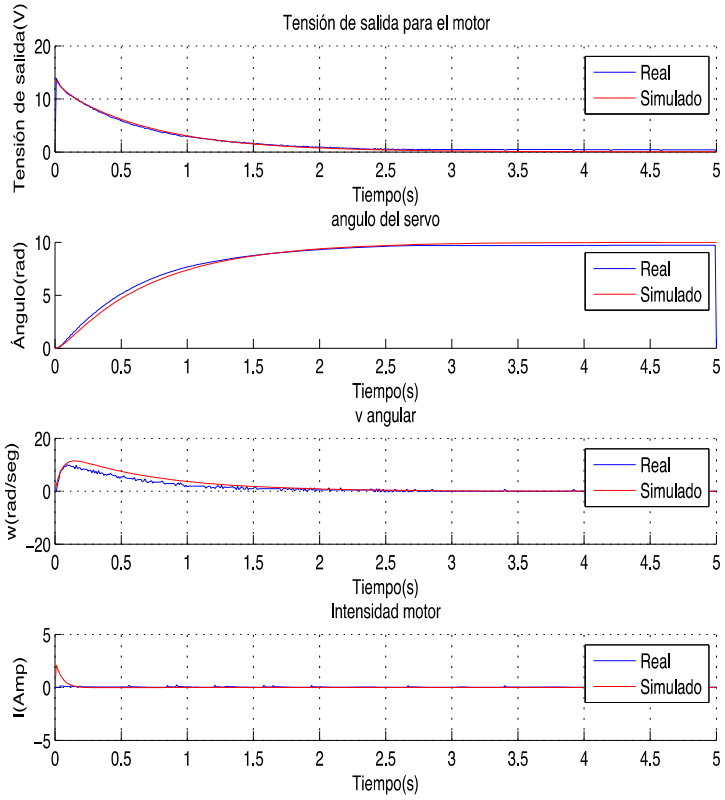


Fig 6. Comparación de resultados del sistema de control simulado con el real.

3.4 Controlador-Observador

Cuando no puede medirse con precisión alguna de las variables de estado, se puede utilizar un observador de estados discreto (siempre que el sistema sea observable) [6] a partir del modelo de estados y de la salida, que en nuestro caso es la posición angular del eje del motor. El diseño de las constantes L de realimentación del observador y las constantes K de realimentación del controlador, se realiza previamente mediante Matlab. Los resultados obtenidos en tiempo real con la maqueta del motor son también satisfactorios y nos permiten mostrar fácilmente al alumno la utilidad del diseño de sistemas de control con observadores.

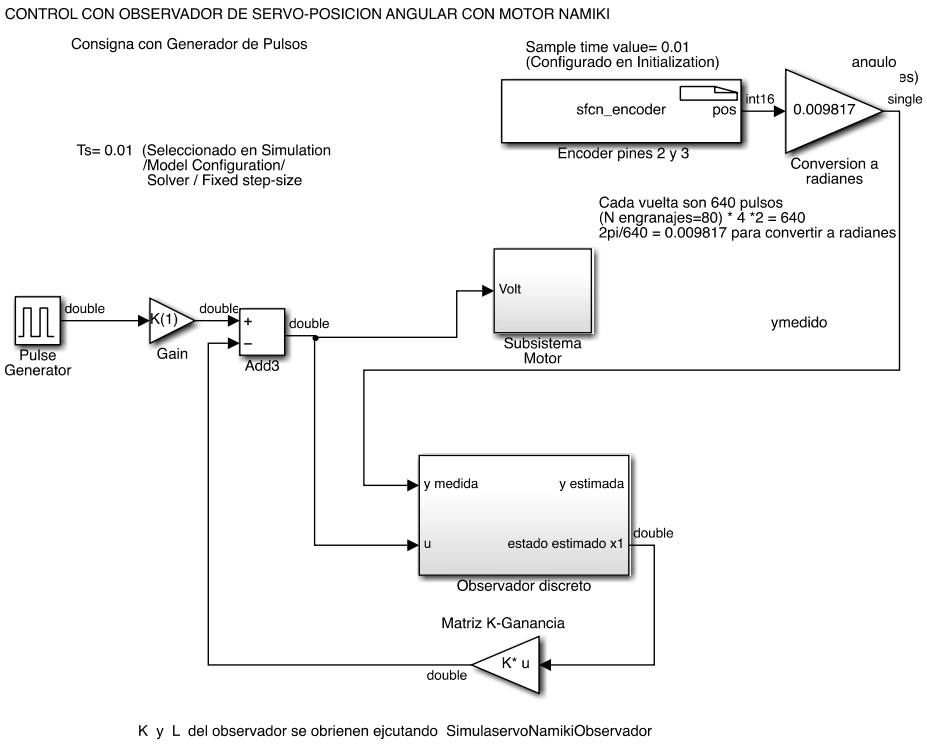


Fig 7. Controlador-Observador

4 Conclusiones

Se ha diseñado un sistema y un conjunto de experiencias prácticas para el estudio de las principales técnicas de control digital de la posición y velocidad de un motor de corriente continua. El sistema es de bajo coste, fácil de programar mediante Simulink o en lenguaje C de Arduino), y sencillo de mantener en un laboratorio de prácticas. Hemos podido ensayar (simular y controlar físicamente en tiempo real), varias configuraciones de control de posición y velocidad angular, tales como:

- Diseño control discreto PID a partir de la f.d.t o bien con el método de sintonización de Ziegler-Nichols.
- A partir del modelo en espacio de estados, se diseña: a) un sistema de control por realimentación de estados y ubicación de polos, b) Control LQR (Regulador Cuadrático Lineal) y c) un controlador-observador.

La programación gráfica de Simulink en tiempo real es sencilla y muy intuitiva. Dado que disponemos del modelo matemático del motor (función de transferencia discreta y su representación en el espacio de estados), se puede comparar la respuesta del sistema simulado con la respuesta real del sistema de control digital de manera muy rápida. El bajo coste de los materiales utilizados permite la fabricación a mayor escala de las maquetas de prácticas implementadas, por lo que, sin duda, a corto plazo los alumnos podrán disponer del material adecuado para simultanear las prácticas con la teoría o incluso para trabajar en casa.

Agradecimientos. Al Secretariado de Innovación Docente de la Universidad de Granada por los Proyectos de Innovación Docente concedidos que nos han permitido financiar la adquisición de algunas de las maquetas de laboratorio utilizadas en prácticas.

Referencias

1 Alecoop: http://www.alecop.com/equipamiento-didactico/areas/regulacion-electronica-de-motores/. Accedido el 12/07/2017.
2. Feedback-instruments. http://www.feedback-instruments.com/. Accedido el 12/07/2017
3. Dfrobot. https://www.dfrobot.com/product-1460.html. Accedido el 12/07/2017
4. Arduino support from Simulink. http://es.mathworks.com/hardware-support/arduino-simulink.html. Accedido el 12/07/2017.
5. Arduino. https://www.arduino.cc. Accedido el 12/07/2017.
6. Ogata, N. “Ingeniería de Control Moderna”. ISBN: 9788483226605
7. Bishop. R.H: “Sistemas de Control Moderno”. ISBN-10: 8420544019. 2005.

Consumo de energía y asignaturas de arquitectura y tecnología de computadores

Antonio F. Díaz, Julio Ortega, Juan José Escobar, Mancia Anguita, Jesús González, Miguel Damas
Departamento de Arquitectura y Tecnología de Computadores
E.T.S.I.I.T., Universidad de Granada
{afdiaz, jortega, jjescobar, manguita, jesusgonzalez, mdamas}@ugr.es

Resumen. El consumo energético de los programas ha pasado a ser una medida de prestaciones tan importante como el tiempo de procesamiento, a pesar de que no suele incluirse en las medidas de rendimiento de los programas. Por esta razón es conveniente incluir en las asignaturas del área de arquitectura y tecnología de computadores contenidos relacionados con las prestaciones energéticas de los programas y las arquitecturas, y disponer de herramientas que permitan caracterizar la potencia y la energía consumida según las características del código a ejecutar. La necesidad de evaluar los programas según su eficiencia energética y su tiempo de ejecución constituye una aproximación multiobjetivo a la evaluación de prestaciones que debería introducirse en las asignaturas de Ingeniería Informática. En este artículo también describimos un sistema basado en Arduino que permite obtener medidas de potencia y energía consumida en las prácticas y proyectos que abordan la generación de códigos óptimos para una determinada plataforma.

Palabras clave: Arduino, Computación energéticamente eficiente, Phyton, Planificación sensible al consumo energético.

Abstract. Nowadays, energy consumption of applications has become a performance measure as relevant as runtime although does not frequently appear in the program performance measures. This way, issues related with the energy consumption of applications and systems should be included in the subjects of computer architecture. Moreover, the availability of tools and strategies to characterize the instant power and consumed energy according to the code profile should be also considered. This could make possible the development of approaches to distribute the workload among the hardware to reach a tradeoff among time and energy efficiency. The searching for these tradeoffs clearly sets a multi-objective approach for performance evaluation that should be taken into account in the Computer Engineering and Computer Science courses. This paper also describes an Arduino-based system to measure the instant power and consumed energy in projects and practical exercises related with the generation of optimal codes.

Keywords: Arduino, Power-aware computing, Phyton, Energy-aware scheduling.

1 Introducción

La energía consumida por los computadores siempre ha constituido una medida importante desde el punto de vista de su propia viabilidad. El consumo energético de los primeros computadores electrónicos era de tal envergadura, que circulan anécdotas del efecto de su funcionamiento en la red eléctrica de la ciudad o el campus donde estaban ubicados. Así, el ENIAC I, capaz de realizar 5000 sumas por segundo, 357 multiplicaciones por segundo, o 35 divisiones por segundo, consumía una potencia de alrededor de 175 KW. Desde el ENIAC I, capaz de realizar 0.03 operaciones/s por watio en 1946, hasta los computadores actuales que pueden alcanzar las 10^{10} operaciones/s por watio, la eficiencia energética de los computadores ha crecido en un factor de 10^{12}, casi igual al factor de incremento en la velocidad de procesamiento. A pesar de esta mejora en el rendimiento energético, la generalización del uso de los dispositivos personales de cómputo y el desarrollo de supercomputadores cada vez más veloces ha hecho que el consumo energético de las diversas aplicaciones TIC sea una parte importante del consumo energético mundial y constituya una barrera esencial para el desarrollo de los supercomputadores y de los centros de procesamiento de datos. Además, el consumo energético de los procesadores ha sido un factor a considerar en la propuesta de nuevas microarquitecturas, y plantea retos esenciales en el desarrollo de supercomputadores cada vez más veloces: actualmente el desarrollo de supercomputadores que proporcionen Exaflops.

Así, la eficiencia energética de los programas se ha convertido en una medida de prestaciones tan importante como la velocidad de procesamiento, si bien no suele considerarse en las publicaciones que incluyen información sobre la eficiencia de los programas. Por un lado está la necesidad de mejorar la autonomía de los dispositivos que utilizan baterías, y por otro las razones económicas y medioambientales que motivan la reducción en el consumo energético de las plataformas de cómputo. Estas plataformas pueden incluir un número considerable de núcleos de cómputo, y otros elementos, que dan lugar a valores de consumo de potencia instantánea elevados. Además del aumento de la temperatura ocasionado por la mayor potencia instantánea, con la consiguiente mayor probabilidad de fallo en los componentes y la necesidad de sistemas de ventilación más complejos, el costo de la energía consumida puede llegar a ser mayor que el de la propia plataforma [1]. De hecho, ya hace algún tiempo que se ha observado la nada despreciable contribución a las emisiones de CO_2 de las tecnologías de la información y las comunicaciones. Por ejemplo, el impacto medioambiental de las emisiones de CO_2 de los centros de procesamiento de datos de Estados Unidos fue similar, en 2010, al de economías de países como Argentina [2]. El interés en la eficiencia energética de los computadores de altas prestaciones es patente en la aparición de la lista Green500 [3] que ha puesto de manifiesto mejoras de un factor de 5 en los valores de MFLOPS/W entre 2009 y 2013, y la viabilidad de la computación en la escala de los Exaflops (Exascala) para 2020 incluso precisa ritmos de mejora mayores si se quiere disponer de un sistema con una velocidad del orden de los 10^{18} flops que consuma unos 20 MW como máximo [4].

Por tanto, es conveniente incorporar e intensificar los contenidos relacionados con el consumo energético en las asignaturas de arquitectura y tecnología de computadores de

grado, especialmente en los estudios de Ingeniería Informática, poniendo de manifiesto la perspectiva multiobjetivo desde la que se debe abordar la optimización de código y la evaluación de las prestaciones. Para la docencia de estos temas, entre otros aspectos, es necesario disponer de herramientas y estrategias para medir los consumos de energía de los computadores e instrumentar convenientemente los códigos.

En la Sección 2 de este artículo se describen algunos de los modelos de consumo energético de los circuitos junto con algunas técnicas para mejorar la eficiencia energética de los computadores y los programas. En la Sección 3 describiremos el sistema de medida basado en *Arduino* que hemos desarrollado, y en la Sección 4 propondremos contenidos relacionados con el consumo energético que consideramos podrían incluirse en algunas de las asignaturas relacionadas con la arquitectura y tecnología de computadores de los grados en ingeniería informática, y más concretamente en el grado de Ingeniería Informática y en la especialidad de Ingeniería de Computadores de la Universidad de Granada. Finalmente, la Sección 5 proporciona las conclusiones del artículo.

2 Estrategias para mejorar el consumo de energía

El consumo de energía ha sido un factor importante en el diseño de las microarquitecturas. En el campo de los procesadores embebidos, dadas las características de los sistemas en los que se incluyen, las restricciones relacionadas con el consumo de energía y la disipación de calor siempre han sido esenciales. Por otro lado, la evolución hacia arquitecturas de propósito general multi-núcleo ha sido una consecuencia de la necesidad de limitar la potencia por unidad de superficie disipada por el microprocesador, que podría haber llegado a valores inaceptablemente altos si se hubiera mantenido asociada la mejora de prestaciones a microarquitecturas superescalares cada vez más complejas, para poder terminar más instrucciones por ciclo. Así, a principios de siglo se plantearon distintas alternativas para mantener el ritmo marcado por la ley de Moore. Entre ellas las microarquitecturas VLIW de propósito general, en las que el compilador era el principal responsable para extraer el paralelismo y evitar estructuras que consumen energía y recursos hardware como los buffers de renombramiento o reordenamiento. También se propusieron microarquitecturas con varios núcleos de procesamiento, que han sido las que a la postre se han convertido en la tendencia dominante en la evolución de los microprocesadores.

Para entender la influencia de la tecnología en la potencia disipada de un circuito integrado CMOS se puede utilizar la expresión (1) [4], en la que el primer término se debe al consumo dinámico de carga y descarga de la capacidad de salida, *C*, de una puerta lógica, en un circuito integrado con un coeficiente de actividad, *A*, que representa la fracción de puertas del circuito que conmutan en cada ciclo, a una tensión, *V*, y una frecuencia de reloj, *f*. El segundo término de (1) se debe a la corriente entre fuente de alimentación y tierra, I_{cortoc}, durante un el instante de tiempo, t, en el que conmuta la puerta. El tercer y último término es la potencia consumida debido a la corriente de

pérdidas, I_{leak}, independiente del estado de la puerta lógica. Es posible reducir la potencia consumida bajando la frecuencia del circuito o la tensión de alimentación (con el consiguiente efecto en la velocidad del mismo).

$$Potencia = ACV^2f + tAVI_{cortoc} + VI_{leak} \quad (1)$$

Sin embargo, hay que tener en cuenta que la frecuencia máxima a la que puede funcionar el circuito está relacionada con la tensión de la alimentación, tal y como muestra la expresión (2):

$$f_{max} \sim \frac{(V - V_{umbral})^2}{V} \quad (2)$$

Así, para incrementar la velocidad de procesamiento aumentando la frecuencia del procesador, habría que reducir la tensión umbral, V_{umbral}, en (2). No obstante, esto supondría aumentar la corriente de pérdidas, que crece exponencialmente al reducirse la tensión umbral, como muestra (3):

$$I_{leak} \sim e^{\frac{-qV_{umbral}}{kT}} \quad (3)$$

Por tanto, el aumento de la frecuencia máxima de los microprocesadores reduciendo la tensión de alimentación conjuntamente con la tensión umbral se ve limitado por el aumento que se origina en la potencia asociada a la corriente de pérdidas, y así surgen alternativas como las microarquitecturas VLIW de propósito general o las microarquitecturas multinúcleo, a las que nos hemos referido.

Junto con la potencia, existen otras medidas de prestaciones relacionadas con el consumo de energía de los circuitos integrados. Entre ellas están la *densidad de potencia*, o potencia consumida por unidad de superficie; la *potencia pico*, o potencia máxima que puede consumir un circuito sin sufrir daños; y la *potencia dinámica*, que hace referencia al cambio brusco en el consumo de potencia que puede producirse en el circuito sin que el ruido asociado a las variaciones temporales de corriente originen comportamientos anómalos. La *energía necesaria para realizar un cálculo* es una magnitud importante, por ejemplo, para evaluar la eficacia con la que se aprovecha la energía que almacena una batería. Así, un procesador puede consumir menos potencia que otro en ejecutar un programa, pero si tarda más tiempo puede ocurrir que al final la energía consumida sea mayor. Relacionada con la energía necesaria para realizar un cálculo está la figura de mérito de los MIPS/W (millones de instrucciones por segundo por watio), que conecta las instrucciones ejecutadas y la energía consumida en dicha ejecución. Cuanto mayor sean los MIPS/W de un procesador, mejor será desde el punto de vista del consumo de energía. En los circuitos integrados CMOS, para los que se pueden aplicar las ecuaciones (1-3) puesto que el consumo de potencia depende del cuadrado de la tensión, es posible reducir la potencia por un factor dado, de manera que la frecuencia (y por tanto la velocidad de los circuitos) se vea afectada por un factor de reducción menor. Esto permite establecer un compromiso en el que se reduzca algo la velocidad del circuito para conseguir un aumento en los MIPS/W.

Además de la energía consumida por el procesador o procesadores, existen otros factores que afectan al consumo energético en el computador que están relacionados con la operación de la memoria, los buses y los sistemas de interconexión, los dispositivos de entrada/salida, etc. Por tanto, la computación eficiente desde el punto de vista energético implica técnicas basadas no sólo en el diseño de la microarquitectura, sino también en los niveles de sistema computador y sistema operativo [5].

Por ejemplo, la memoria consume una parte significativa de la energía. Este consumo depende de dos causas fundamentales: la frecuencia de acceso y la corriente de pérdidas (I_{leak} en la expresión (1)). La organización de la memoria de forma que se active sólo la parte de la misma a la que se realiza el acceso (afectando al coeficiente A en la expresión (1)) contribuye a reducir el consumo. Así, si la memoria se distribuye entre bancos que pueden activarse independientemente, se puede reducir el consumo activando los bancos necesarios en cada momento. La localidad de los accesos a la cache de datos y de instrucciones determina la eficiencia con la que se puede aplicar esta técnica. Otra estrategia basada en la localidad de los accesos consiste en situar una cache pequeña delante de la cache L1 para reducir el número de accesos a la cache L1. Si se producen aciertos a esta cache sólo en un 50% de los casos, se ahorraría la mitad de la diferencia entre el consumo de la cache L1 y el de la cache pequeña.

Los buses y demás elementos de interconexión también son responsables de una parte importante del consumo de energía, sobre todo los buses entre circuitos integrados, con un número considerable de líneas, cuyos drivers pueden representar el 15%-20% de la potencia del circuito integrado. Existen técnicas que implementan los controladores de los buses junto con la circuitería de codificación/decodificación adecuada. Así, codificando las direcciones del bus mediante el código Gray se pueden reducir las transiciones de nivel en las líneas del bus cuando las direcciones cambian secuencialmente, por ejemplo en las transferencias de líneas de cache. También mejora el consumo de energía en los buses la transmisión de la diferencia entre direcciones que se solicitan sucesivamente y la compresión de la información de las líneas de dirección para minimizar el número de líneas del bus.

El aprovechamiento eficiente del paralelismo constituye una forma importante de reducir el consumo de energía de las aplicaciones. Según la expresión (2) se puede reducir la frecuencia de reloj reduciendo la tensión y, además, a partir de (3), una reducción de la tensión tiene ventajas importantes en cuanto al consumo de energía. En la Figura 1 se pone de manifiesto el efecto de distribuir una carga de trabajo W entre p procesadores que funcionan a una frecuencia p veces menor que la del procesador de partida. Cada uno de los procesadores consumiría una energía p^2 veces menor. Dado que se utilizan p procesadores, se necesitaría una energía total p veces menor para completar la carga de trabajo en el mismo tiempo.

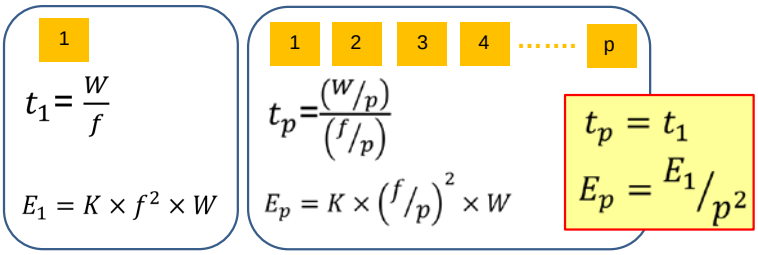


Figura 1. Procesamiento paralelo y consumo energético

Además, el procesador no necesita funcionar siempre a su frecuencia máxima. Así, como se muestra en la Figura 2, según sean las características de la sincronización temporal entre tareas que se procesan en paralelo, se puede reducir la frecuencia a la que funciona alguno de los procesadores y reducir la energía consumida sin aumentar el tiempo de procesamiento. El sistema de *runtime* puede controlar la tensión a la que trabaja el procesador, y también se puede conseguir que las aplicaciones utilicen las funciones del sistema operativo para controlar los valores de tensión que precisan.

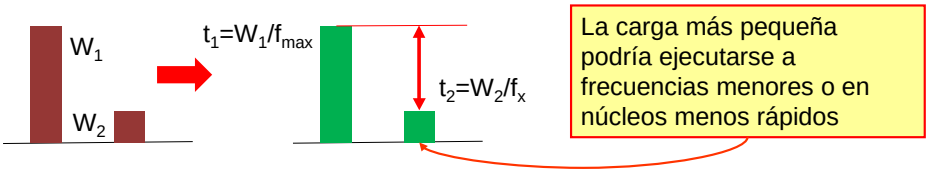


Figura 2. Ajuste de frecuencias para reducir consumo de energía en tareas de distinta carga de trabajo

Según esto, para disminuir el consumo energético de la aplicación, se podría ajustar la frecuencia a la que trabajan los núcleos (DVFS, de *Dynamic Voltaje and Frequency Scaling*). En cualquiera de los casos hay que tener en cuenta el efecto que la reducción en el consumo energético puede tener en la velocidad de la aplicación. Sobre todo en aquellos casos en los que precisamente se está desarrollando un programa paralelo para ganar velocidad, o existen restricciones de tiempo-real que satisfacer. Por tanto, al aparecer la reducción del consumo energético como un objetivo más entre las prestaciones de los códigos, se plantea la necesidad de alcanzar un compromiso entre consumo energético y velocidad, y la optimización de prestaciones pasa a convertirse en un problema multiobjetivo y surgen dos preguntas relacionadas con este hecho: (1) cuál es la configuración de núcleos y/o frecuencias de funcionamiento que permite el mínimo consumo energético y que la aplicación termine dentro de los requisitos de tiempo establecido, y (2) cuál es la configuración que proporciona la máxima velocidad con un consumo de energía inferior a un límite máximo de consumo.

El diseño de algoritmos de planificación de la carga de las aplicaciones que tengan en cuenta requisitos de consumo además de los de velocidad necesita disponer de una caracterización del consumo energético de la aplicación en la plataforma en que se va a ejecutar. Esta caracterización no es sencilla. Los microprocesadores multinúcleo

disponen de una unidad de control, PCU (*Package Control Unit*), que implementa ciertas heurísticas para determinar las frecuencias de uso de los núcleos del microprocesador a partir de factores como el número de llamadas del SO, el uso esperado de los núcleos, la temperatura, la tasa de fallos de las caches, etc. Teniendo en cuenta todos esos factores, el microprocesador gestiona los distintos modos de funcionamiento (estados C y P en los microprocesadores de AMD e Intel, por ejemplo), y hay que relacionar los cambios en dichos modos con el tipo de procesamiento que realiza la aplicación y el ahorro energético que se consigue. Existen aproximaciones de caja-negra (*black-box*) que buscan caracterizar el consumo energético de una aplicación específica correlacionando las características de los códigos correspondientes con los consumos observados [1,6,7]. Por ejemplo, en [4] se ilustra el aprovechamiento de los modos de ahorro de potencia de la especificación ACPI (*Advanced Configuration and Power Interface*) [8] implementada en la resolución paralela de sistemas lineales dispersos. Se trata de establecer la relación entre las características del código y las condiciones en que el sistema operativo, a través de ACPI en este caso, promueve los cambios a los estados de ahorro de energía en los núcleos de los microprocesadores, si no se puede ocasionar la transición desde la aplicación. En cualquier caso, cuando se promueve el paso a modos en los que la potencia instantánea consumida se reduce, hay que tener en cuenta si se produce una penalización en el tiempo de ejecución que afecte al consumo energético final de manera que no se produzca ningún ahorro de energía.

Otra alternativa se basa en el uso modelos de consumo energético teóricos más o menos aproximados para diseñar heurísticas de distribución de carga bien en plataformas que implementan DVFS (*Dynamic Voltage and Frequency Scaling*) para las que se supone que se puede actuar sobre los niveles de frecuencia o tensión, bien en plataformas heterogéneas incluyendo procesadores con distintas frecuencias y capacidades de procesamiento [9-12].

3 Un sistema de medida de consumo de energía

En esta sección se describe el sistema que hemos desarrollado para medir el consumo de energía en tiempo real en un computador (Figura 3). Concretamente, el sistema permite medir de forma individual, en cada uno de los nodos de un cluster, tanto la potencia instantánea (en W) como el consumo de energía acumulado (Wxh).

La medida se lleva a cabo a través de sensores que proporcionan la intensidad de corriente que circula en el cable de corriente que conecta el equipo a la red eléctrica. Al llevarse a cabo la medida en este punto es posible obtener el consumo real de todo el nodo, incluyendo tanto el de los componentes activos como el debido a las pérdidas asociadas a la conversión en la fuente de alimentación. Las conclusiones respecto a la eficiencia energética que se puedan alcanzar tras el estudio del comportamiento de estas medidas de consumo del nodo completo son relevantes, ya que ponen de manifiesto si las estrategias que se han implementado para mejorar la eficiencia energética son observables y por tanto, realmente significativas y útiles.

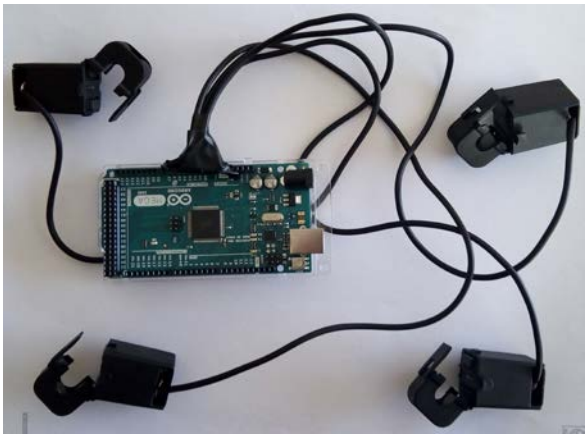


Figura 3. Placa basada en Arduino Mega y sensores de corriente

El sistema consta de una placa de *Arduino Mega* y un conjunto de sensores conectados directamente a las tomas de corriente. Una vez conectado, el cable de corriente de cada equipo pasa por cada sensor, que detecta la intensidad de corriente que circula. El sensor utilizado es el YHDC-SCTD010T-5A, cuyo esquema se muestra en la Figura 4. Puede medir hasta 5A y tiene una salida proporcional entre 0 y 5V, con una precisión de ±2%. El *Arduino* dispone de un conversor A/D interno de 10 bits y, para aprovechar mejor su rango dinámico, se utiliza su referencia interna de 2,56V que sólo está disponible en los *Arduino Mega*. Aunque esto limita la medida a 2,56 V de entrada máxima, en la práctica se podrían medir hasta 588 W de potencia. Dado que los nodos consumen una potencia menor, esta cota superior no representa ninguna limitación.

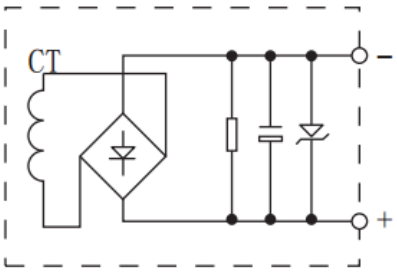


Figura 4. Diagrama interno del sensor de corriente

El *Arduino Mega* se conecta por USB a uno de los nodos del cluster, lo que permite, además de suministrarle la correspondiente alimentación, transmitir los datos mediante el puerto serie. En el caso de un ordenador Linux, se crea el interfaz /dev/ttyACM0, que permite transmitir y recibir datos en serie. En la primera versión implementada el *Arduino* realiza 4 capturas por segundo de cada sensor. Cada segundo, transmite por el

puerto los datos de potencia (en W) de cada sensor, así como la energía consumida (en watios por hora, Wxh). Los valores de energía consumida se pueden poner a 0 en cualquier momento para volver a estimar fácilmente el consumo en un periodo de tiempo. El programa de captura que ejecuta el *Arduino Mega* está escrito en *Python*.

4 Contenidos sobre consumo energético en ingeniería informática

En los curricula de IEEE/ACM para los estudios de grado de *Computer Science* [13] y *Computer Engineering* [14] se hace referencia a contenidos relacionados con el consumo energético. Así en los curricula de *Computer Science*, en el área de conocimiento de *Arquitectura y Organización* se incluye la energía dentro de las restricciones físicas para los cursos de *Lógica Digital y Sistemas Digitales*, y dentro del tópico de mejora de prestaciones en los cursos de *Componentes Digitales y Diseño*. También dentro del área de *Cuestiones Sociales y Práctica Profesional* los curricula de *Computer Science* incluyen contenidos relacionados con el consumo energético en asignaturas de *Sostenibilidad* que describen los impactos medioambientales de las decisiones de diseño de sistemas de cómputo relacionadas con los algoritmos, los sistemas operativos, las redes, las bases de datos etc. En cuanto a los estudios de grado de *Computer Engineering* [14], los conceptos relacionados con el consumo energético se incluyen en el área de *Sistemas Embebidos* en el curso de *Techniques for low-power operation* (CE-ESY-10), en el área de *Systems and Project Engineering* dentro del curso de *Maintainability, sustainability, manufacturability* (CS-SPE-12), y en el área de *Electrónica*, en el curso de *Foundations of Electronics* (ELE_E101). Aparte de la inclusión de criterios de consumo energético en asignaturas de diseño hardware, como reflejan los curricula de *Computer Engineering*, los conceptos relacionados con la eficiencia energética deberían tenerse en cuenta de forma similar a los que tienen que ver con el tiempo de ejecución y la complejidad hardware, por ejemplo. De hecho, en la introducción del área de conocimiento de *Arquitectura y Organización* en [13] se indica que los profesionales informáticos no pueden contemplar el computador como una caja negra que ejecuta programas por arte de magia, sino que deben tener un conocimiento profundo del hardware en el que se basa la computación y de la interfaz que proporciona la arquitectura a las capas software superiores, y deben conocer los componentes funcionales del computador, sus características, interacción y prestaciones, para desarrollar programas de altas prestaciones. Las prestaciones elevadas o la viabilidad de que una aplicación pueda ejecutarse en un computador no solo está relacionada con restricciones en el tiempo de ejecución, sino también con el consumo de energía.

Por tanto, consideramos interesante la inclusión de contenidos relacionados con el consumo energético en las asignaturas del grado en Ingeniería Informática (y en la especialidad en Ingeniería de Computadores) de la Universidad de Granada. Dado que los contenidos de las asignaturas que se imparten actualmente están definidos en las correspondientes guías [15] de acuerdo con los créditos y el periodo de tiempo asignado para ser impartidos, y con los objetivos y contenidos de la memoria de la titulación, integrar los nuevos tópicos no es trivial, sobre todo teniendo en cuenta las limitaciones temporales y la carga lectiva de los estudiantes. En las asignaturas que no incluyan ya los correspondientes tópicos relacionados con la energía, estos se pueden introducir a través de seminarios y demostraciones en algunas de las sesiones de prácticas. Por ejemplo, en alguno de los ejercicios prácticos que impliquen realizar medidas de rendimiento de programas, junto con las medidas usuales que utilizan el número de procesadores, también se pueden incluir las que consideran la energía consumida.

Los contenidos de consumo energético a introducir en asignaturas del grado se pueden clasificar teniendo en cuenta su relación con: (1) la medida del consumo energético; (2) el control de los modos de funcionamiento; y (3) el desarrollo de programas energéticamente eficientes. En cuanto a la medida del consumo de energía, aparte de usar sistemas como el descrito en la Sección 3 de este artículo, se puede hacer uso de los contadores de prestaciones (*Performance Monitoring Counters*) que proporcionan los procesadores. Por ejemplo, en [16] se describe una extensión de la biblioteca PAPI (Performance API) para medir el consumo de potencia y energía basada en los contadores de prestaciones. Respecto al control de los modos de funcionamiento, la interfaz estándar ACPI (*Advanced Configuration and Power Interface*) [8] incluye mecanismos para la gestión y ahorro de la energía, controla convenientemente el funcionamiento del BIOS, y proporciona información acerca de la configuración y el control de los estados de ahorro de energía (C0, C1, C2, C3,.., Cn) y de prestaciones (P0, P1,…, Pn) del procesador. En la misma línea, por ejemplo, el núcleo de Linux implementa la infraestructura *cpufreq* [17], que permite que el sistema operativo, bien automáticamente a partir de eventos que genera la ACPI, bien a partir de llamadas de programas de usuario, pueda aumentar o disminuir la frecuencia del procesador para ahorrar energía. Dentro de las *cpufreq* están los denominados *governors* [18], que implementan políticas específicas de control de la velocidad del reloj del procesador. La interfaz para usar estos servicios a nivel de usuario se puede encontrar en `cpufreq.h` [19].

En primer lugar, asignaturas como *Fundamentos Físicos y Tecnológicos* podrían incluir modelos de consumo energético en los circuitos integrados y principales elementos de la electrónica del computador (buses, memorias, etc.), abordando la relación entre conceptos como frecuencia, voltaje, potencia y consumo energético. En *Tecnología y Organización de Computadores* podrían incluirse, junto a los criterios que se utilizan para comparar diseños alternativos de los elementos del computador basados en tiempos de ejecución y complejidad, otros criterios relacionados con el consumo energético. La introducción que estas asignaturas fundamentales pueden proporcionar sobre los conceptos relacionados con el consumo energético sería muy útil, no solo para asignaturas del grado posteriores, como *Estructura de Computadores*, *Arquitectura de Computadores*, e *Ingeniería de Servidores*, sino también para *Sistemas Operativos* y las asignaturas de la especialidad en *Ingeniería de Computadores*. Precisamente, en asignaturas relacionadas con el diseño hardware, como *Desarrollo de Hardware Digital* o *Sistemas Empotrados*, los criterios de consumo energético son importantes desde el punto de vista de la autonomía de los dispositivos alimentados por baterías.

En *Estructura de Computadores*, donde entre otros contenidos se aborda el estudio del lenguaje ensamblador, seguramente se podrían incluir en las prácticas que se realizan al respecto, ejercicios relacionados con la consulta de los contadores de prestaciones que proporcionan información del consumo energético. Como se ha indicado, los fabricantes de microprocesadores incluyen recursos que permiten cambiar la frecuencia de reloj a la que funciona el procesador para ahorrar energía. El conocimiento de estos estados y su utilidad, incluyendo referencias a interfaces como ACPI, constituyen tópicos importantes a tener en cuenta en esta asignatura.

En asignaturas que tratan temas relacionados con el paralelismo, la optimización de código o la evaluación de prestaciones como *Arquitectura de Computadores*, *Ingeniería de Servidores* y *Arquitecturas y Computación de Altas prestaciones* (esta última en la especialidad de Ingeniería de Computadores) se pueden incluir criterios de optimización de los códigos y medidas de eficiencia que tengan en cuenta el consumo energético. De hecho el procesamiento paralelo constituye una estrategia eficaz para este propósito. Por ejemplo, en la Figura 5 se muestran las potencias instantáneas medidas mediante el sistema descrito en la Sección 3 en uno de los nodos de un clúster para un programa secuencial (un algoritmo evolutivo multiobjetivo de selección de características en un problema de clasificación de electroencefalogramas) y dos versiones paralelas distintas (una basada en el paradigma de matestro-trabajador, MW, y otra en el paradigma de islas). Como se puede observar, la potencia instantánea de los programas paralelos, que utilizan ocho núcleos de procesamiento, alcanza valores más altos que el programa secuencial. No obstante, teniendo en cuenta que los programas paralelos consiguen reducir el tiempo de procesamiento, al final se consiguen factores de ganancia de velocidad de alrededor de siete (con ocho núcleos), con consumos energéticos de alrededor de un tercio del que arroja el secuencial. A partir de las curvas de consumo de potencia medidas, se pone de manifiesto incluso la situación en la que está el programa paralelo: bajadas de potencia en el programa paralelo maestro-trabajador cuando solo está trabajando un procesador, o aumentos en el programa paralelo de islas cuando hay comunicación entre dichas islas.

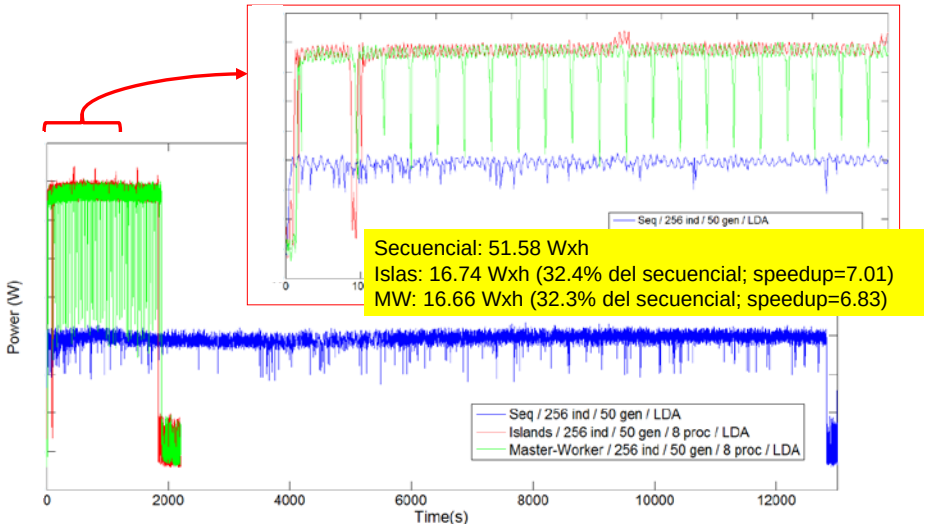


Figura 5. Ganancias de velocidad y consumos de energía de varias alternativas paralelas de un programa

El aprovechamiento adecuado de los recursos que están consumiendo energía puede mejorar la eficiencia energética además de reducir el tiempo de cómputo. Una técnica usual para mejorar las prestaciones en los procesadores actuales que se estudia en la asignatura de *Arquitectura de Computadores* es el procesamiento especulativo, que permite iniciar cálculos antes de que hayan terminado las instrucciones de las que depende que el resultado de dichos cálculos sea útil o correcto. Si la predicción que se ha realizado es correcta, el tiempo de procesamiento se reduce y la relación MIPS/W se puede mejorar considerablemente. No obstante, dado que para implementar eficientemente la especulación se utilizan una serie de recursos que consumen energía, si la especulación no es correcta se malgasta energía. En esta misma línea, la optimización en el uso de los recursos de cómputo como pueden ser los buses o las redes de comunicación, la memoria o el uso del nivel de precisión adecuado en los cálculos no sólo tiene efecto en los tiempos de cómputo sino también en el consumo energético. Es importante introducir estos criterios energéticos en la optimización de prestaciones. Por otra parte, el uso de criterios de rendimiento energético, permite comparar la eficacia con la que se usan distintas arquitecturas paralelas heterogéneas actuales donde una medida de rendimiento basada en la división de la ganancia entre el número de núcleos de procesamiento puede resultar discutible dado que puede haber núcleos con capacidades de cómputo bastante diferentes.

La importancia en la gestión del consumo energético de las grandes infraestructuras de cómputo y los requisitos de consumo que se plantean actualmente en el diseño de

supercomputadores (listas TOP500 y GREEN500) deben abordarse en asignaturas de la especialidad de Ingeniería de Computadores como *Centros de Procesamiento de Datos*, y *Arquitecturas y Computación de Altas Prestaciones*. De hecho, en la asignatura de *Centros de Procesamiento de Datos* ya se incluye el estudio de los requisitos técnicos y la normativa de gestión energética de estas infraestructuras

5 Conclusiones

En este artículo hemos puesto de manifiesto que los mecanismos y estrategias implementados en las arquitecturas actuales para mejorar su consumo de energía, junto con las herramientas disponibles para hacer uso de ellos, permiten elaborar propuestas de trabajos prácticos y proyectos que facilitan la incorporación de contenidos relacionados con el consumo energético en los estudios de Ingeniería Informática e Ingeniería de Computadores.

Por otra parte, el estudio que se ha realizado acerca de los efectos en el consumo energético de ciertas estrategias de programación está contribuyendo a la identificación de técnicas de programación eficientes desde el punto de vista energético que se pueden estudiar conjuntamente con las que buscan la optimización de los tiempos de ejecución. Esta situación plantea una perspectiva multiobjetivo en el estudio del rendimiento de los computadores en términos del tiempo de ejecución, del consumo energético, y de la calidad de la solución alcanzada muy adecuada para la formación como ingeniero del profesional de la informática.

Agradecimientos. Este artículo ha sido financiado por el proyecto TIN2015-67020-P (Ministerio de Economía y Competitividad" y fondos FEDER).

6 Referencias

1. De Sensi, D.:"Predicting performance and power consumption of parallel applications". In 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), 2016. DOI: 10.1109/PDP.2016.41.
2. Lucente, E.J.:"The comming 'c' change in data centers". http://www.hpcwire.com/2010/06/15/the_coming_c_change_in_datacenters/, 2010.
3. Lista Green500 de Noviembre de 2016 https://www.top500.org/green500/lists/2016/11/
4. Mudge, T.:"Power: A first-class architectural design constraint". IEEE Computer, Vol. 34, No, 4, pp.52-58. Abril, 2001.
5. IEEE Computer, Vol. 36, No.12. Diciembre, 2003 (Número dedicado a la optimización de consumo de energía en los computadores: *power-aware computing*).
6. Aliaga, J.I.; Barreda, M.; Dolz, M.F.; Martín, A.F.; Mayo, R.; Quintana-Ortí, E.S.:"Assessing the impact of the CPU power-saving modes on the task-parallel solution of sparse linear systems". Cluster Computing, 17, pp. 1335-1348, 2014.

7. Barik, R.; Farooqui, N.; Lewis, B.T.; Hu, C.; Shpeisman, T.:"A black-box approach to energy-aware scheduling on integrated CPU-GPU systems". CGO'15, March 12-14, Barcelona, Spain, pp.70-81, 2016.
8. Advanced configuration and power interface specification (ACPI): http://www.acpi.info/
9. Lee, Y.C.; Zomaya, A.Y.:"Energy conious scheduling for distributed computing systems under different operationg conditions". IEEE Trans. On Parallel and Distributed Systems, Vol.22, No.8, pp.1374-1381. August, 2011.
10. Dorronsoro, B.; Nesmachnow, S.; Taheri, J.; Zomaya, A.Y.; Talbi, E-G; Bouvry, P.:"A hierarchical approach for energy-efficient scheduling of large workloads in multicore distributed systems". Sustainable Computing: Informatics and Systems, 4, pp.252-261, 2014.
11. Rotem, E.; Weiser, U.C.; Mendelson, A.; Ginosar, R.; Weissmann, E.; Aizik, Y.:"H-EARtH: Heterogeneous multicore platform energy management". IEEE Computer magazine, pp.47-55. October, 2016.
12. Ortega, J.; Escobar, J.J.; Díaz, A.F.; González, J.; Damas, M.:"Energy-aware scheduling for parallel evolutionary algorithms in heterogeneous architectures". 2nd Workshop on Power-Aware Computing (PACO 2017). Magdeburgo, Alemania, 2017.
13. IEEE/ACM Computer Science Curricula 2013: http://www.acm.org/education/curricula-recommendations.
14. IEEE/ACM Computer Engineering Curricula 2016: http://www.acm.org/education/curricula-recommendations.
15. Guías docentes del Grado en Ingeniería Informática de la Universidad de Granada: http://grados.ugr.es/informatica/pages/infoacademica/guias_docentes/guiasdocentes_curso_actual.
16. Weaver, V.N.; Johnson, M.; Kasichayanula, K.; Ralph, J.; Luszczek, P.; Terpstra, D.; Moore, S.:"Measuring energy and power with PAPI". 41st Intl. Conference on Parallel Processing Workshops (ICPPW), pp. 262-268, 2012
17. CPU frequency scaling: https://wiki.archlinux.org/index.php/CPU_frequency_scaling
18. CPUFreq Governors: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt
19. `cpufreq.h`: https://code.woboq.org/linux/linux/include/linux/cpufreq.h.html

La asignatura “Internet de las Cosas” en el master DATCOM de la UGR

M. Damas[1], F. Gómez[1], S. Moreno[1], C. Bailón[1], A. Olivares[2]

1) Departamento de Arquitectura y Tecnología de Computadores. ETSI Informática y de Telecomunicación. Universidad de Granada.
{mdamas, frgomez}@ugr.es, {smoreno94, cbailon37}@correo.ugr.es
2) Nazaríes Information Technologies S.L. (http://www.nazaries.com)
alberto.olivares@nazaries.com

Resumen. En este artículo se presenta tanto la motivación como la metodología docente de la asignatura Internet de las Cosas, una optativa que se imparte en la ETSIIT de Granada dentro del módulo de “Sistemas de Aplicaciones Específicas” de la especialidad en Ingeniería de Computadores y Redes del Master de Ciencia de Datos e Ingeniería de Computadores (DATCOM) de la Universidad de Granada. Se pretende con ello justificar la inclusión de esta asignatura en el master DATCOM y su vinculación con el resto de asignaturas impartidas en dichos estudios de posgrado. Concretamente, en este trabajo se muestra que hay detrás del concepto de Internet de las Cosas y por qué precisamente ahora se ha convertido en una verdadera revolución, también cómo se ha organizado la asignatura para lograr los objetivos propuestos en la guía docente, así como la metodología y herramientas que se van a utilizar, y finalmente se indican las principales conclusiones del trabajo.

Palabras Clave: Internet de las Cosa, Computación física, Ciencia de Datos, Cloud Computing, Ingeniería de Computadores.

Abstract. This paper shows the motivation and teaching methodology of the elective course 'Internet of Things', which is imparted in the ETSIIT of Granada within the Specific Application Systems module of the Computer and Network Engineering specialty of the Master in Data Science and Computer Engineering (DATCOM) of the University of Granada. It is intended to justify the inclusion of this subject in the master DATCOM and its linkage with the rest of subjects taught in these postgraduate studies. Specifically, this paper shows what is actually behind the Internet of Things concept and why it has now become a real revolution. It also shows how the course has been organized to achieve the objectives proposed in the teaching guide, as well as the methodology and tools to be used. Finally, the main conclusions of this work are presented.

Keywords: Internet of Things, Physical computing, Data Science, Cloud Computing, Computer Engineering.

1 Introducción

Desde sus inicios, Internet ha ido incorporando nuevos actores conforme los avances tecnológicos lo han permitido. Primero fueron los dispositivos móviles, luego las personas a través de las redes sociales y en estos últimos años también los objetos cotidianos [1], a lo que se ha denominado Internet de las Cosas (o IoT por sus siglas en inglés, *Internet of Things*). Este concepto, atribuido al investigador del MIT Kevin Ashton en 1999, se podría definir por tanto como la convergencia en la evolución de distintos tipos de tecnologías hardware y software que están permitido que cada vez más objetos heterogéneos se puedan interconectar entre sí, dotándolos de mayor inteligencia, y permitiendo además crear nuevos servicios y oportunidades de negocio. Precisamente, desde hace 4 años la consultora Gartner considera el Internet de las Cosas como una de las tecnologías con mayores expectativas, y al igual que otras compañías como CISCO y ERICSSON, predice que para el año 2020 habrá más de 25.000 millones de objetos conectados en el mundo, con el consiguiente impacto que esto supondrá tanto a nivel económico como social (Figura 1). En pocos años tendremos tantas cosas conectadas a nuestro alrededor que no nos daremos ni cuenta que Internet estará ahí, tal como decía Mark Weiser ya en el año 1991.

En este artículo se hará en primer lugar un breve recorrido histórico de hitos importantes de este nuevo paradigma y se analizará su repercusión actual. A continuación se mostrarán algunos productos comerciales de objetos conectados representativos en distintos ámbitos (salud, deporte, industria, hogar y ciudades inteligentes, etc.), se introducirán las tecnologías habilitadoras (desarrollo rápido de prototipos, estándares y protocolos de comunicaciones, ciencia de datos, cloud computing, plataformas IoT, etc.), y se mencionarán brevemente los retos a los que se enfrenta. Seguidamente, y una vez argumentados los motivos por los que consideramos apropiado impartir dicha asignatura en el master DATCOM, también se muestra la metodología docente que se va a seguir y los equipos y herramientas que se van a utilizar en la parte práctica de dicha asignatura. Finalmente se presentan las conclusiones más importantes de este trabajo.

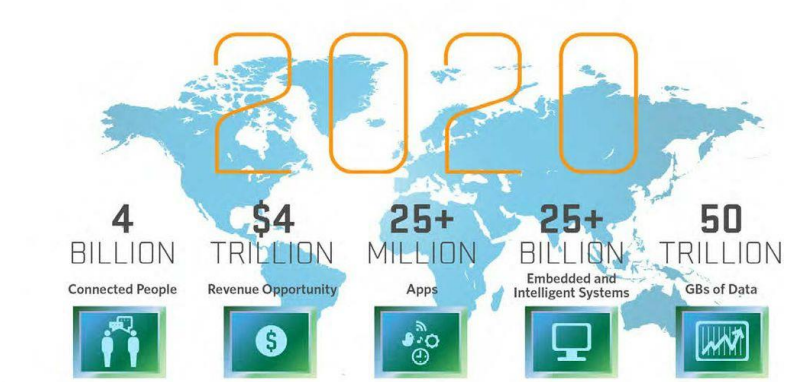


Figura 1. Predicciones sobre IoT en 2020 (fuente: IDC)

2 ¿Por qué esta asignatura en el Master DATCOM?

2.1 Breve recorrido histórico

A continuación se indican algunas pinceladas sobre hitos representativos en relación a la breve historia del IoT, que no pretende ser exhaustiva, pero que dan idea de sus inicios y de cómo ha ido evolucionando este concepto:

2.2 Situación actual

Para entender en qué situación nos encontramos actualmente, Gartner, una consultora americana que representa gráficamente mediante una curva la evolución de las tecnologías (https://es.wikipedia.org/wiki/Ciclo_de_sobreexpectación), ha situado

durante varios años el Internet de las Cosas en el pico de gran expectativa, como se puede apreciar en la figura 2, y de ahí que se escuche tanto hablar de esta tecnología. No obstante, en la última curva publicada en agosto de 2016 (http://www.gartner.com/newsroom/id/3412017), ya no aparece explícitamente el IoT, lo cual se puede interpretar de varias formas: que se ha considerado que otros conceptos como el hogar conectado o las plataformas IoT la representan, que ya está consolidada y no es una expectativa sino una realidad, o sencillamente que no han sabido situarla en ninguna parte de la curva.

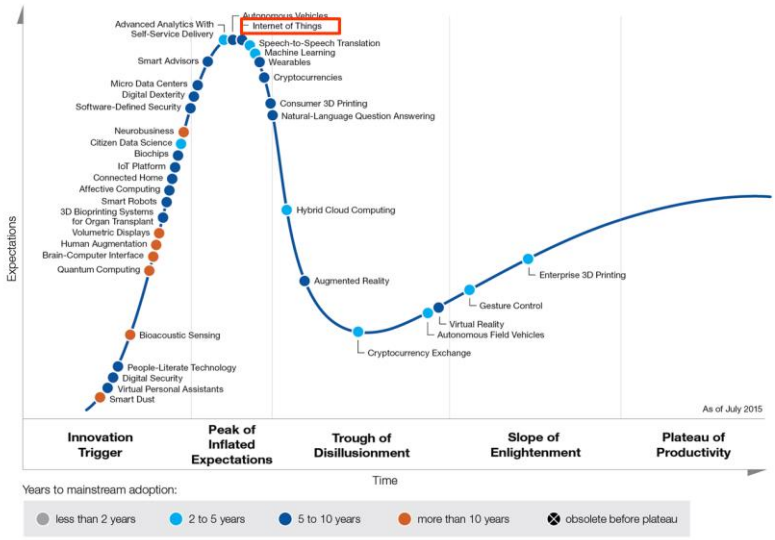


Figura 2. Curva de Gartner para tecnologías emergentes (http://www.gartner.com/newsroom/id/3114217)

Pero la realidad es que este concepto sigue siendo muy popular, como se puede ver también según las tendencias de búsqueda en Google (ver figura 3), donde se observa que desde hace 4 años dicha popularidad empieza a crecer muy rápidamente, frente a otros conceptos relacionados.

Desde el punto de vista económico, los ingresos globales del Internet de las Cosas también está creciendo de forma significativa y todas las predicciones apuntan a que este incremento se va a mantener e incluso crecer a lo largo de los próximos años. No obstante, lo que sí es una realidad es que sólo en el año pasado (2016) se produjeron una serie de adquisiciones muy significativas relacionadas con el IoT, como por ejemplo:

- **Qualcomm** compra NXP por 47.000 millones de dólares
- **SofBank** compra ARM por 31.000 millones de dólares
- **Cisco** compra Jasper (plataforma IoT) por 1.400 millones de dólares
- **TDK** compra InvenSense Inc. (fabricante sensores IoT) por 1.300 millones de dólares
- **Cypress Semiconductor** adquiere Broadcom por 550 millones de dólares, …

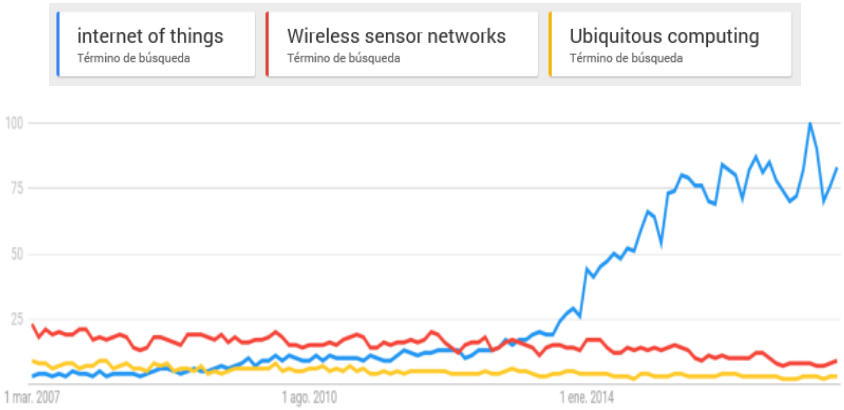


Figura 3. Popularidad de IoT según las tendencias de Google

En cuanto al impacto en investigación, si se busca en las bases de datos donde se encuentran las publicaciones científicas con prestigio a nivel internacional, también se puede observar (ver figura 4) un incremento exponencial en el número de publicaciones donde aparece el tópico "Internet of Things". Comentar además que en los documentos de trabajo que se acaban de publicar del HORIZONTE 2020, donde se marcan las líneas de investigación que se van a incentivar a nivel Europeo, se puede comprobar también que el IoT aparece en un lugar muy destacado. Y concretamente en España, en abril de este año, el Gobierno, y más específicamente el ministro de Fomento Íñigo de la Serna, ha aprobado un plan para potenciar la digitalización, el Internet de las cosas y la transformación energética.

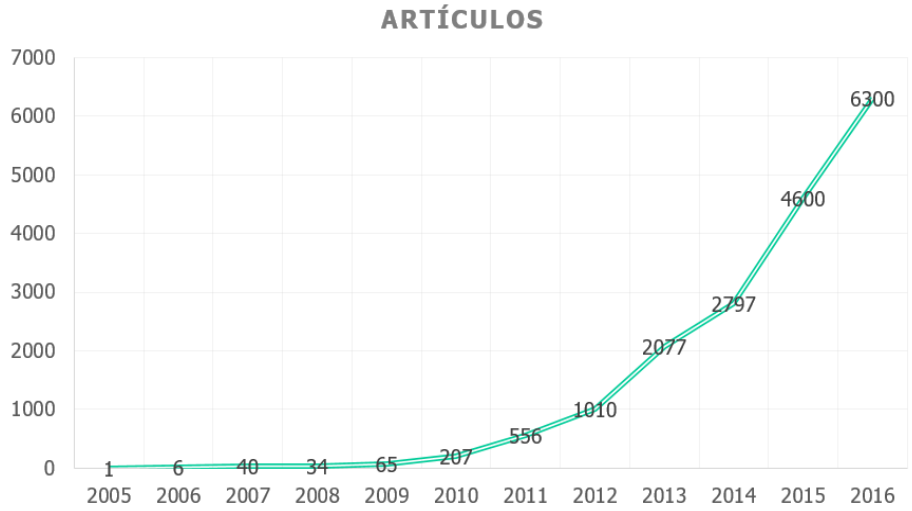


Figura 4. Impacto del IoT en investigación

2.3 Ejemplos de objetos IoT representativos

Los ámbitos de aplicación relacionados con el IoT son diversos y heterogéneos, de tal forma que existen dispositivos para la casa inteligente, objetos vestibles, para ciudades inteligentes, para la industria, para el coche conectado, para salud, ganadería, comercio, agricultura, etc. [2, 3, 4]. Precisamente, en la siguiente tabla se muestran como ejemplo algunos dispositivos IoT representativos o curiosos.

Nombre	Descripción	Enlace
Báscula Withings 	Báscula que te reconoce, y envía tu peso y % de grasa corporal a tu teléfono móvil. Uno de los primeros productos IoT comerciales.	http://www.withings.com/
Trackdot 	Objeto conectado que rastrea tus maletas cuando viajas.	http://www.trakdot.com/es
Sen.se Mother 	Dispositivo que te cuida como una madre, ya que permite conectar muchos sensores que se harán cargo de su salud, seguridad y bienestar	https://sen.se/store/mother/
Jawbone Up 	Pulsera de actividad, que junto con los relojes inteligentes son de los dispositivos IoT de mayor éxito.	https://jawbone.com/up
SSE-TN1W 	Dispositivo de Sony para ayudar a mejorar su nivel de tenis. Existen otros dispositivos similares para el golf, ciclismo, gimnasios, etc.	https://www.sony.es/electronics/dispositivos-inteligentes/sse-tn1w
Parrot Pot 	Macetero que integra distintos sensores para monitorizar el estado de las plantas mediante una aplicación móvil.	https://www.parrot.com/es/jardin-conectado/parrot-pot
HapiFork 	Tenedor inteligente que ayuda a comer mejor.	https://www.hapi.com/product/hapifork
Smart electric bike 	Bicicleta inteligente que ayuda a circular mejor por las ciudades	https://www.smart.com/id/en/index/smart-electric-bike.html

SkyBell 	Timbre Wifi, para responder de forma remota y visualizar en el móvil quién está llamando a nuestra puerta.	http://www.skybell.com/
Bombilla LIFX 	Bombilla inteligente controlada por Wifi desde el teléfono móvil.	http://lifx.co/
SmartThings 	Plataforma que permite añadir sensores de todo tipo a nuestro entorno doméstico.	http://www.smartthings.com/
Termostato Nest 	Termostato inteligente que se ajusta automáticamente para ahorrar energía en función de las preferencias del usuario.	https://nest.com/
Amazon Dash Button 	Dispositivo Wifi que te permite pedir un producto pulsando un botón.	http://www.amazon.es/dashbutton
Coche Tesla 	Vehículo autónomo que actualiza su software por internet.	http://www.tesla.com

De estos productos comerciales destacar tanto la plataforma SmartThings, una empresa pequeña comprada por Samsung por 200 millones de dólares, como el termostato inteligente de Nest, en este caso una empresa comprada por Google por nada menos que 3.200 millones de dólares, con el objetivo en ambos casos de tratar de posicionarse estratégicamente con dispositivos de gran éxito comercial en un sector, el del hogar inteligente, donde se prevé que en el futuro habrá miles de aplicaciones [5].

Para mostrar además el potencial de esta revolución del IoT, incluso para generar nuevos modelos de negocio, señalar los dos últimos ejemplo, el botón de Amazon y el coche de Tesla. En el caso de Amazon, con un dispositivo extremadamente sencillo, que incluso se regala una vez hecho el primer pedido, ha conseguido crear un nuevo producto claramente IoT, que le permite incrementar sus ventas. Y en el caso de

Tesla, Elon Musk (CEO de la compañía), está proponiendo también un modelo de negocio disruptivo, consistente en ofrecer un precio único a la hora de comprar un Tesla que englobe al propio coche, el coste de asegurarlo y sus hipotéticos arreglos, ya que su creador está convencido de que tendrá menos averías y menos accidentes.

2.4 Tecnologías habilitadoras

¿Pero cuales son las tecnologías habilitadoras, facilitadoras o potenciadoras de este concepto que hemos visto que está tan de moda, y que está revolucionando el mercado?. Para entenderlo mejor hay una frase de Peter Senge que dice "La innovación depende de la integración de tecnologías", frase que explica muy bien lo que está sucediendo, ya que lo que se ha producido es una convergencia de tecnologías y circunstancias que han hecho posible el IoT, como se puede observar en la siguiente figura:

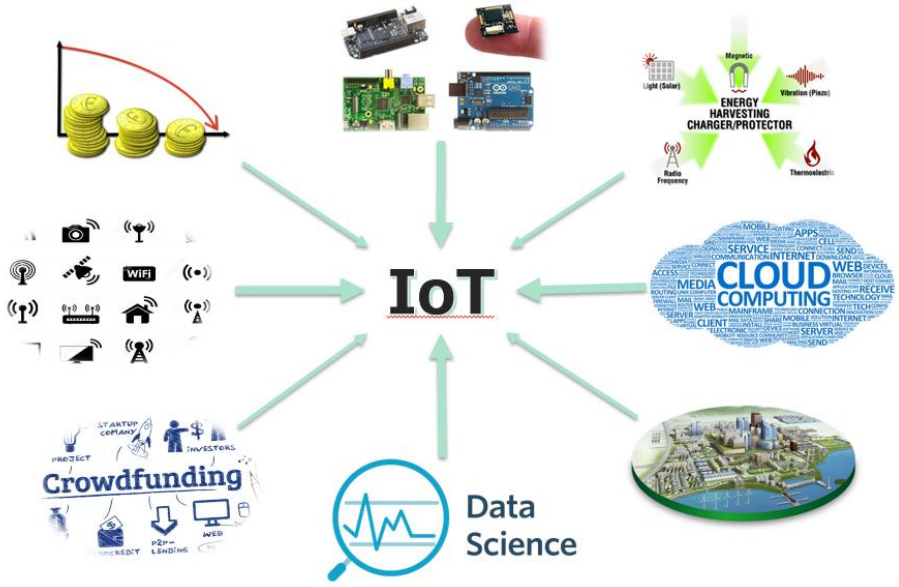


Figura 5. Tecnología y circunstancias que han hecho posible el IoT

En primer lugar, el abaratamiento de la tecnología ha sido clave para conseguir que este concepto sea una realidad, ya que ha permitido que el coste necesario para poder poner un dato en internet sea lo suficientemente bajo como para que no sea un impedimento a la hora de desarrollar un producto IoT. Otros avances que están impulsando el IoT es la posibilidad de usar fuentes de energía alternativas para estos dispositivos, utilizando para ello los logros que se están consiguiendo en los últimos años relacionados con el concepto de *Energy Harvesting* [6], y que pueden hacer realidad el que ciertos productos IoT sean totalmente autónomos, incluso energéticamente.

Además y aunque no sean tecnologías, tampoco hay que olvidar que están ayudando los nuevos modelos de negocio que están apareciendo, impulsados por las estrategias de mecenazgo y financiación colectiva como las aceleradoras y el *crowdfunding*, que están facilitando crear muchas startups y empresas; y ciertas políticas Gubernamentales (en determinados países y ciudades), que han financiado sobre todo proyectos relacionados con las Smart City, para mejorar temas como el tráfico, la iluminación, la polución, etc.

Y por supuesto, también están siendo claves en el impulso de este nuevo paradigma la facilidad para el desarrollo rápido de prototipos, la evolución en estándares y protocolos de comunicación, la computación en la nube y en gran medida la ciencia de datos, tecnologías que se van a tratar con algo más de detalle a continuación.

2.4.1 Desarrollo rápido de prototipos

A la hora de crear un dispositivo IoT lo más razonable es empezar por un prototipo [7], donde tendremos:

- El diseño del objeto físico (para lo cual la fabricación aditiva es decir las impresoras 3D están ayudado mucho).
- La electrónica (es decir los periféricos IoT y las plataformas de desarrollo hardware). Plataformas de desarrollo existen muchas, las más conocidas son Arduino y Raspberry PI (que recientemente han comercializado productos muy enfocados al IoT), pero existen otras muy interesantes como las de Libelium (nacional), BeagleBone, Intel Curie, NanoPi, ESP8266, ESP32, etc.
- El servicio de Internet con el que se conectará, y que le va a proporcionar un valor añadido al objeto. Para la implementación de estos servicios las API juegan un papel muy importante, ya que son como el lenguaje necesario para que los programas se puedan comunicar entre sí a través de Internet.

Destacar aquí que la comunidad de código abierto en los últimos años ha impulsado mucho todo este concepto del IoT, ya que han facilitado el acceso al hardware y al software necesario para desarrollar dispositivos de este tipo. Remarcar además, que a la hora de escribir código para estos dispositivos IoT, que suelen tener recursos limitados, habría que tener en cuenta técnicas para optimizar la gestión de la memoria, la vida de la batería e incluso el rendimiento.

2.4.2 Estándares y protocolos de comunicación

Otra de las tecnologías que están influyendo en el IoT son las comunicaciones. Se pueden usar todos los estándares de infraestructuras de redes existentes, tanto por cables como por radio, aunque las preferidas para el IoT son la comunicaciones inalámbricas. Existen muchos estándares de comunicaciones inalámbricas dependiendo por ejemplo del ámbito de aplicación. Concretamente tenemos redes como NFC, ZigBee, Bluetooth, WiFi, etc., para redes de área personal o local, y redes para mayor cobertura como por ejemplo WiMAX y las redes celulares.

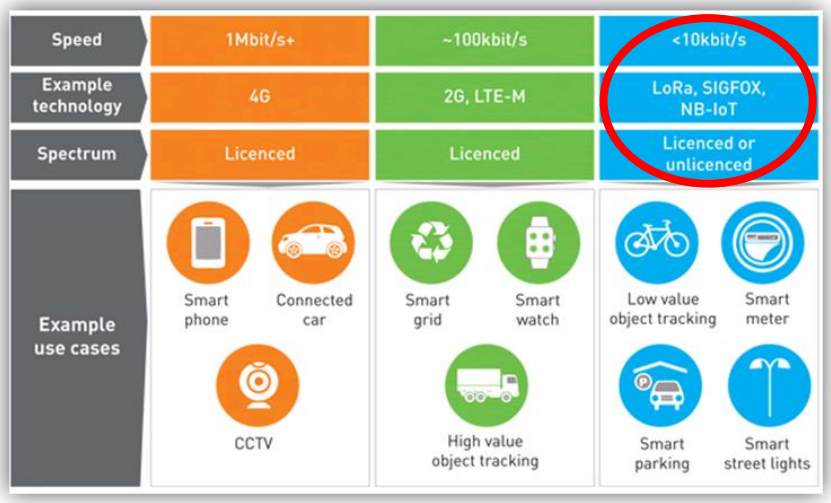


Tabla 1. Redes inalámbricas celulares para IoT (Fuente: Analysys Mason, 2015)

Concretamente en la tabla 1 se muestran las redes inalámbricas celulares que más se están usando actualmente según el ámbito de aplicación. El 4G para los dispositivos que necesitan mayor velocidad y transferencia de información, como los teléfonos móviles, las cámaras de vigilancia o los coches conectados. Para los dispositivos con menores requerimientos en cuanto a velocidad y tamaño de los mensajes se puede usar el 2G o el LTE-M (o 4G para IoT), y lo destacable es que precisamente para las aplicaciones donde se necesite cubrir también zonas extensas, pero con menores consumo y a un precio mucho más bajo (contadores y farolas inteligentes, o gestión de aparcamiento en grandes ciudades), han surgido recientemente nuevas soluciones de redes, como por ejemplo Sigfox, Lora o NB-IoT entre otras. Es decir, que el IoT también está impulsando y generando nuevas oportunidades de negocio en el sector de las comunicaciones. No obstante, para 2020 parece ser que el 5G será la solución para el IoT en cuanto a las comunicaciones inalámbricas se refiere para la mayoría de las aplicaciones, ya que según sus especificaciones permitirá un consumo eficiente, bajo coste y una gran cobertura, y habrá que ver cómo afecta esto a las nuevas soluciones comentadas.

Añadir que además de los estándares a nivel de infraestructuras de red, también existen protocolos como los que se muestran en la tabla 2, que facilitan y simplifican el trabajo a los programadores de aplicaciones y proveedores de servicios para IoT, y que dependiendo de los tiempos de respuestas que se requieran y para qué se vayan a utilizar será mejor usar unos u otros. Concretamente, para la comunicación en tiempo real entre dispositivos en la industria se suele utilizar DDS o OPC UA, para enviar datos desde los dispositivos a los servidores en la nube mejor usar MQTT o CoAP, para la comunicación entre los servidores un protocolo adecuado es AMQP, y para que los usuarios accedan a los datos en la nube lo recomendable es usar XMPP o REST [8].

Application Protocol		DDS	CoAP	AMQP	MQTT	MQTT-SN	XMPP	HTTP REST
Service Discovery		mDNS				DNS-SD		
Infrastructure Protocols	Routing Protocol	RPL						
	Network Layer	6LoWPAN					IPv4/IPv6	
	Link Layer	IEEE 802.15.4						
	Physical/ Device Layer	LTE-A	EPCglobal	IEEE 802.15.4		Z-Wave		
Influential Protocols		IEEE 1888.3, IPSec					IEEE 1905.1	

Tabla 2. Protocolos de aplicación para IoT [9]

2.4.3 Computación en la nube

Por supuesto otra tecnología que está potenciando el IoT es el Cloud Computing o computación en la nube, ya que a medida que se van recibiendo más y más datos de dispositivos IoT se necesitará más capacidad de procesamiento y memoria. Es decir, se requiere que las infraestructuras puedan crecer, y precisamente esta escalabilidad la proporciona las soluciones cloud, como por ejemplo las de Amazon Web Services, Microsoft Azure, Heroku, OpenStack, OpenShift, etc., que van a permitir tener servicios que se ajustan a la demanda, que es un requerimiento clave en muchas de las aplicaciones IoT.

2.4.3 Ciencia de datos

Pero sin duda alguna, una de las tecnologías que más han influido en esta revolución del IoT es la Ciencia de Datos. Para entenderlo hay que saber que para el 2020 se prevé que haya alrededor de 40 Zettabytes, y gran parte de estos datos van a venir de las cosas o las máquinas conectadas a internet, que de forma autónoma generan información. Luego, el IoT es ante todo una enorme fuente de datos que pueden convertirse en conocimiento mediante las técnicas y procedimientos adecuados que va a proporcionar la Ciencia de Datos, apoyándose en sus paradigmas de Big Data, minería de datos, aprendizaje automático, etc. Concretamente, se utiliza el Big Data para poder afrontar de forme eficiente ese gran volumen de datos heterogéneos, y los algoritmos de minería de datos y de aprendizaje automático para poder realizar predicciones, clasificaciones (por ejemplo para resolver los problemas de atascos de tráfico o gestión del aparcamiento en grandes ciudades, o para la detección de

enfermedades cardiovasculares), dar recomendaciones, reconocer patrones o coincidencias (por ejemplo para ayudar a los clientes de un supermercado a encontrar los productos que están buscando), también para descubrir comportamientos o detectar anomalías en el funcionamiento de máquinas industriales, entre otras cosas, y todo ello analizando los distintos datos que llegan de los miles e incluso millones de sensores que se pueden tener en determinadas aplicaciones [10,11].

2.5 Plataformas IoT

Precisamente para abstraernos de la complejidad de todas estas tecnologías que se acaban de presentar, están proliferando en estos últimos años unas herramientas en la nube, denominadas Plataformas IoT, que facilitan el desarrollo integral de soluciones para el Internet de las Cosas. Concretamente, como se puede ver en la figura 6, las funcionalidades de estas herramientas son: simplificar la conectividad y la normalización ofreciendo el código o las interfaces necesarias para acceder a las plataformas hardware de desarrollo; gestionar los dispositivos de forma centralizada para poder por ejemplo hacerles a todos a la vez una actualización del firmware; añadir reglas para disparar eventos y construir interfaces gráficas para visualizar los datos; almacenar los datos y realizar el tratamiento analítico sobre dichos datos, es decir, facilitar la aplicación de la ciencia de datos; ofrecer las interfaces externas necesarias para poder integrarse con otras aplicaciones de terceros a través fundamentalmente de las API; y por supuesto también dotar de la seguridad y privacidad necesarias a las aplicaciones IoT.

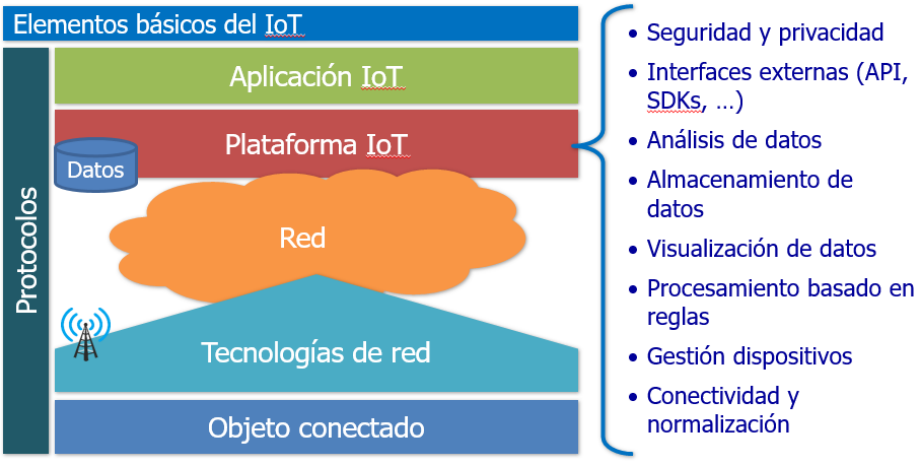


Figura 6. Ubicación y funcionalidad de las Plataformas IoT

Precisamente en la última curva de Gartner, estas plataformas están ascendiendo hacia el pico de máximas expectativas, y se están convirtiendo en una de las herramientas en las que se está trabajando e invirtiendo más actualmente. Ejemplos de este tipo de plataformas que ya están disponibles son: Xively, Thingworx, Comulocity, Thing+, Axeda, MyDevies, Relayr, ThingSpeak, Temboo, Thethings.io,

Thinger.io, Carriots, Sofia2, etc. Incluso las grandes compañías, como Microsoft, Google, Amazon, Oracle, Intel, IBM, etc., también se están posicionando desarrollando nuevas plataformas o adaptando las ya existentes para el IoT, como por ejemplo, IBM con su Plataforma Watson IoT. De hecho, según algunos estudios se estima que existen ya más de 300 plataformas de este tipo [12].

2.6 Obstáculos

Pero también existen problemas u obstáculos que están frenando el despliegue masivo de productos IoT, como son por ejemplo:

- La seguridad en las redes de comunicaciones. Según algunos estudios, el 70% dispositivos IoT tienen vulnerabilidad en contraseñas, cifrado o permisos de acceso, y el 50% de las aplicaciones no encriptan las comunicaciones (Hewlett Packard, 2015). Esto lo están aprovechando los delincuentes informáticos que están focalizando su atención en estos dispositivos vulnerables para lanzar sus ciberataques.
- La privacidad de los datos personales de los usuarios del IoT, que desconfían de dejar sus datos en manos de compañías que pueden utilizarlos para otros fines para los que fueron solicitados.
- Los desafíos en el plano ético o jurídico, es decir, ¿quién es el responsable en el caso de un atropello de un vehículo autónomo o en el caso de que un enfermo con un marcapaso IoT sufra un ciberataque?
- La inversión de las empresas, es decir, aunque se han realizado importantes adquisiciones de empresas pequeñas por grandes compañías (como hemos visto en la sección 2.2), y también se está invirtiendo mucho en el desarrollo de plataformas IoT, falta que las empresas dediquen más recursos a la hora de proponer nuevos productos finales para los usuarios.
- La falta de estandarización, es decir, como se ha comentado en las secciones anteriores existen multitud de plataformas de desarrollo hardware, estándares y protocolos de red, plataformas IoT, Sistemas Operativos, etc., que están complicando la interoperabilidad de las soluciones propuestas hasta ahora.

2.7 Adecuación de la asignatura IoT en el Master DATCOM

Luego teniendo en cuenta todo lo comentado en los apartados anteriores, está claro que esta asignatura es fundamental en el Master de Ciencia de Datos e Ingeniería de Computadores, ya que introduce y muestra una visión global y unificada de todas las tecnologías necesarias de este nuevo y popular paradigma del Intenten de las Cosas, y enlaza perfectamente con otras asignaturas del master tal como: Sistemas Empotrados y Co-Diseño Hw/Sw, Minería de Datos, Big Data y Cloud Computing, Servidores Seguros, Modelado de Sistemas y Predicción de Series Temporales, Computación de Altas Prestaciones para Clasificación y optimización, etc.

3 Temario Teórico y Práctico

En la guía docente elaborada para la asignatura Internet de las Cosas aparecen los siguientes objetivos expresados como resultados de aprendizaje:

- Comprender los conceptos básicos correspondientes a IoT
- Adquirir conocimientos sobre herramientas, lenguajes, y plataformas de desarrollo de IoT.
- Conocer los protocolos normalizados de comunicación definidos para IoT
- Conocer los escenarios de aplicación de IoT
- Adquirir la capacidad de concepción, diseño y caracterización de proyectos IoT

En base a estos objetivos se ha planteado el siguiente temario teórico y práctico para esta asignatura:

TEMARIO TEÓRICO:

Tema 1. Introducción al Internet de las Cosas (IoT)
1.1. ¿Qué es el Internet de las Cosas?
1.2. Un poco de historia y situación actual
1.3. Aplicaciones y ejemplos
1.4. Tecnologías habilitadoras
1.5. Plataformas IoT
1.6. Obstáculos

Tema 2. Tecnologías de comunicaciones para IoT
2.1. Identificación de las cosas y direccionamiento. RFID. NFC. IPv6 y 6LoWPAN.
2.2. Comunicaciones de corto alcance. WSN. IEEE 802.15.4, ZigBee, Bluetooth LE.
2.3. Redes de área amplia y baja potencia (LPWAN). LoRaWAN, Sigfox.
2.4. Tecnologías LTE para IoT y conectividad M2M. LTE_M, NB_IoT.
2.5. Comunicaciones IoT por cable. Tecnologías PLC.
2.6. Comunicaciones para la domótica y la inmótica. Z-Wave, EnOcean, KNX.
2.7. Otras tecnologías de comunicación.

TEMARIO PRÁCTICO:
Seminarios sobre Smart Health, Smart City, Smart Grid, Smart Home, etc.
Prácticas sobre redes inalámbricas de sensores y plataformas IoT.

Como se puede observar en el temario teórico, el capítulo 1 se corresponde prácticamente con los contenidos resumido en la sección 2 de este artículo, aunque en la asignatura se profundiza y se dan más detalles de los conceptos que se tratan. En cuanto al capítulo 2, al ser las comunicaciones uno de los aspectos claves del Internet de las Cosas, se ha decidido incluir este tema específico para tratar con mayor

detenimiento y a nivel teórico todo lo relacionado con cómo conectar cualquier tipo de objeto al mundo digital, independientemente de donde se encuentre.

En cuanto al temario práctico, en primer lugar se realizan una serie de seminarios en grupos sobre los distintos escenarios de aplicaciones IoT actuales, para a continuación, ya de forma individual, realizar un proyecto concreto donde se diseñe por completo un prototipo de producto en algunos de los escenarios IoT introducidos previamente.

4 Equipos, herramientas y metodología a utilizar

Utilizar un material adecuado de prácticas que motive al alumno en su aprendizaje es un factor muy importante a tener en cuenta. En nuestro caso, hemos optado por utilizar el módulo NodeMCU con el chip ESP8266 (ver figura 7), una de las placas de desarrollo hardware más interesante teniendo en cuenta sus prestaciones, precio y soporte. El ESP8266 es un SOC que integra una CPU de 32 bits (similar en prestaciones a un Pentium de mediados de los noventa), pero además con conectividad Wifi, 10 puertos GPIO y buses de comunicaciones SPI, I2C y UART. Este chip se ha popularizado de tal forma que se puede comprar integrado dentro de distintos diseños de PCB. Uno de los módulo de este tipo con mayor éxito es el NodeMCU, una iniciativa *Open Source* de Espressif systems, con un precio que ronda los 5€, y que a diferencia de otros módulos, viene con todo lo necesario para empezar a trabajar de forma autónoma, ya que incluye un adaptador serie/USB, se alimenta a través del microusb, y dispone de firmware que permite programarlo en lenguajes como LUA, Python, Basic o JavaScript.

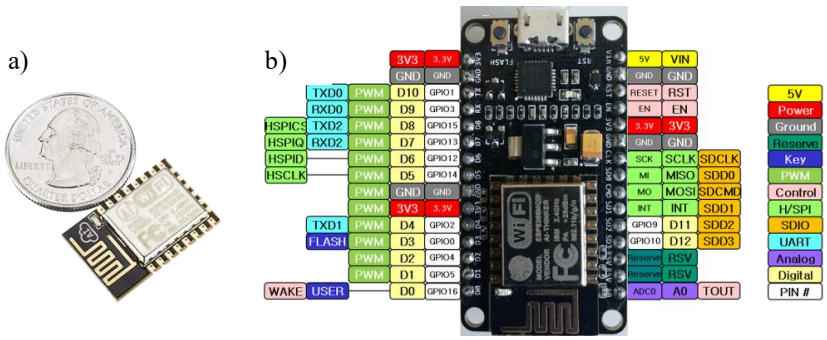


Figura 7. Plataforma de desarrollo hardware elegida para las prácticas: a) ESP8266, b) NodeMCU

En cuanto a la metodología que se propone, como se ha indicado en la guía docente de la sección 3, se basa en una serie de seminarios y prácticas, acordes con el temario teórico, para que el alumno pueda abordar de forma práctica todas las fases necesarias a la hora de realizar un primer prototipo de producto IoT. Concretamente, a lo largo de la teoría, seminarios y prácticas al alumno se le describe un sistema

completo para la medida, comunicación y visualización del nivel de un depósito de líquidos (gasoil, agua, etc.), a partir de un sensor de ultrasonidos, la placa de desarrollo NodeMCU descrita anteriormente y una Plataforma IoT de las introducidas en la teoría. Para ello se dispone en el laboratorio del material necesario para que cada alumno pueda realizar dicha práctica guidada, tal como se muestra en el esquema de la figura 8. Los alumnos deben realizar el circuito eléctrico necesario para interconectar adecuadamente el sensor con el módulo NodeMCU, programar dicho módulo con el IDE de Arduino [13], elegir los protocolos de comunicación a utilizar para el intercambio de información (MQTT, REST, etc.), configurar y programar varias Plataformas IoT para conectar y gestionar la placa NodeMCU, y realizar una comparativa de las funcionalidades de las Plataformas IoT probadas. Otra de las fases a realizar en el desarrollo de un prototipo IoT está relacionada con el diseño del producto, y para ello también se le propone al alumno imprimir en 3D una carcasa que permita albergar el circuito implementado, y que además se pueda instalar en algún depósito estándar del mercado.

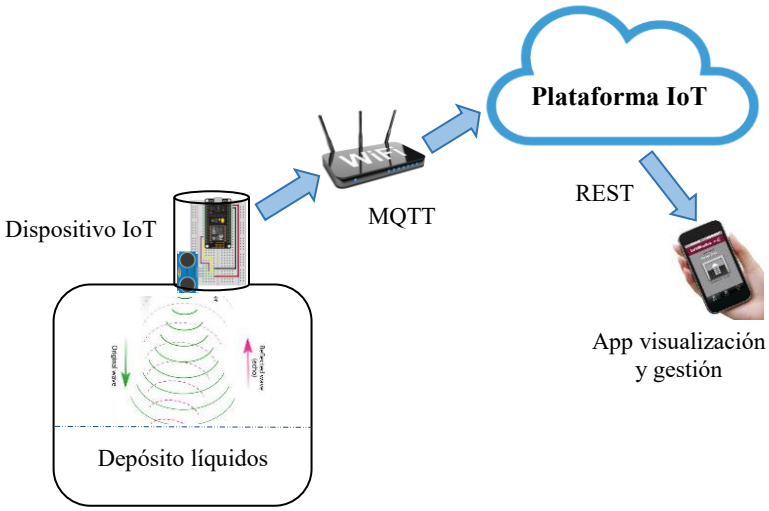


Figura 8. Esquema general de la práctica a realizar en el laboratorio

5 Conclusiones

En este trabajo se ha mostrado la popularidad del paradigma del Internet de las Cosas, así como las tecnologías y circunstancias que están haciendo posible que este concepto se haya convertido en una verdadera revolución tanto a nivel económivo como social. Dicha exposición inicial ha permitido esclarecer la necesidad de incorporar una asignatura sobre este concepto en el Master de Ciencia de Datos e Ingeniería de Computadores que se imparte en la Universidad de Granada, ya que en dicho master se cursan asignaturas que complementan perfectamente al Internet de las Cosas. Remarcar que esta asignatura dentro de la especialidad en Ingeniería de

Computadores y Redes del Master, es además un pilar básico para la especialidad en Ciencia de Datos y Tecnologías Inteligentes, ya que como se ha indicado previamente, se puede considerar como uno de los afluentes principales a la ingente cantidad de información que requieren los algoritmos relacionados con los paradigmas de Big Data, minería de datos, aprendizaje automático, etc.

Se ha presentado además la metodología, así como los equipos y herramientas que se van a utilizar en la asignatura, y todo ello en torno a una práctica guiada en el laboratorio que permite abordar todas las fases necesarias en el desarrollo de un prototipo de producto IoT, fundamental para poder evaluar adecuadamente que los alumnos hayan adquirido las competencias necesarias.

Por lo tanto, se considera que el esquema de trabajo presentado para la asignatura Internet de las Cosas tiene, a nuestro parecer, un gran potencial didáctico, ya que permite que el alumno pueda trabajar con herramientas de programación y equipos reales que se emplean habitualmente en la implementación de soluciones IoT, sin que su disponibilidad, coste, tamaño, o ubicación sean un problema.

Referencias

1. C. Perera, A. Zaslavsky, P. Christen, D. Georgakopoulos: “Context Aware Computing for The Internet of Things: A Survey”, IEEE Communications Surveys & Tutorials, 2014.
2. O. Vermesan, P. Friess: “Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems”. River Publishers, 2013.
3. M.A. Razzaque, et al.: ”Middleware for Internet of Things: A Survey”, IEEE Internet of Things Journal, Vol. 3, no. 1, February 2016.
4. Fundación Telefónica. “Internet Industrial: Máquinas inteligentes en un mundo de sensores”, Grupo Planeta, 2016.
5. A. Vazhnov: “La Red de Todo: Internet de las Cosas y el Futuro de la Economía Conectada”, http://castellano.andreivazhnov.net/cuando-internet-desaparezca-libro-iot/, Accedido el 25/07/2017.
6. M. Ku, W. Li, Y. Chen, K. J. Ray Liu: “Advances in Energy Harvesting Communications: Past, Present, and Future Challenges”, IEEE Communications Surveys & Tutorials, vol. 18, no. 2, 2016.
7. A. McEwen, H. Cassimally: “Internet de las Cosas. La tecnología revolucionaria que todo lo conecta”, Anaya Multimedia, 2014.
8. Stan Schneider: “Understanding the Protocols Behind The Internet Of Things”, Electronic Design, Oct 09, 2013.
9. A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash: “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications”, IEEE Communications Surveys & Tutorials, Vol. 17, No. 4, 2015.
10. R. Ciobanu et al.: “Big Data Platforms for the Internet of Things”, in Big Data and Internet of Things: A Roadmap for Smart Environments, Springer, 2014.
11. T. Chun-Wei et al.: “Data Mining for Internet of Things: A Survey”, IEEE Communications Surveys & Tutorials, Vol. 16, No. 1, 2014.
12. IoT Platforms: Market Report 2015-2021, IoT Analytics 2016.
13. M. Banzi, M. Shiloh: “Introducción a Arduino. La plataforma de código abierto para la creación de prototipos electrónicos”, Anaya, 2015.

Evolución tecnológica del hardware de vídeo y las GPU en los ordenadores personales

Francisco Charte, Antonio J. Rueda, Macarena Espinilla, Antonio J. Rivera

Departamento de Informática. E.P.S. Universidad de Jaén.
{fcharte,ajrueda,mestevez,arivera}@ujaen.es

Resumen. En este artículo se ofrece una revisión de los hitos más importantes en la evolución del hardware gráfico. La comunicación entre los ordenadores y las personas ha ido avanzando a lo largo del tiempo, alcanzando la interactividad con la aparición de los sistemas de tiempo compartido a principios de la década de los 60 del siglo pasado. Los ordenadores personales, cuya expansión se inició casi dos décadas después, adoptaron desde un inicio la visualización de información en una pantalla como medio principal de comunicación con el usuario. El hardware a cargo de esa tarea ha ido evolucionando paulatinamente hasta, en la actualidad, convertirse en parte indispensable de la arquitectura del computador, hasta tal punto que una gran parte de los ordenadores portátiles y de sobremesa incorporan el hardware gráfico en el mismo circuito integrado que aloja al microprocesador.

Palabras Clave: Vídeo, hardware gráfico, GPU, shaders.

Abstract. This article provides a review of the most important milestones in the evolution of graphics hardware. Communication between computers and people has been advancing over time, reaching interactivity with the emergence of time-sharing systems in the early 1960s. Personal computers, whose expansion began almost two decades later, used the visualization of information on a screen as the main means of communication with the user from the very beginning. The hardware in charge of this task has gradually evolved to become an indispensable part of the computer architecture, to such an extent that a large part of laptops and desktop computers incorporate the graphic hardware into the same integrated circuit that houses the microprocessor.

Keywords: Video, graphics hardware, GPU, shaders.

1 Introducción

Los primeros ordenadores disponibles comercialmente, y que comenzaron a emplearse en empresas como las aerolíneas norteamericanas [1], grandes bancos y algunas universidades, los conocidos como *mainframes*, eran sistemas extremadamente caros, por lo que aprovechar al máximo su tiempo de procesamiento era un objetivo esencial. Para

conseguirlo, las tareas ejecutadas por estas máquinas se llevaban codificadas en tarjetas perforadas a un centro de procesamiento de datos, donde se ponían en cola a fin de que el ordenador fuese ejecutándolas por lotes. No se contemplaba la interacción directa entre el ordenador y el usuario final, ya que esto conllevaba dedicar el valioso tiempo de uso de la máquina a una sola persona.

En 1961 el MIT (*Massachusetts Institute of Technology*) llevó a cabo una demostración de uno de los primeros sistemas de tiempo compartido [2]. Este tipo de sistemas permiten que múltiples usuarios interaccionen con el ordenador a través de una terminal que, originalmente, era un dispositivo similar a un teletipo: una impresora con un teclado. El usuario introducía los comandos a ejecutar a través del teclado, empleado como una máquina de escribir, y recibía la respuesta del sistema directamente en papel. El ordenador atendía simultáneamente a múltiples usuarios, dividiendo su tiempo entre todos ellos, pero debido a su velocidad cada usuario tenía la noción de estar trabajando con el sistema a su completa disposición. El principal obstáculo para esta vía de comunicación, aparte del coste de este tipo de terminales y el gasto de papel y tinta, era la velocidad de comunicación, limitada por la propia velocidad de impresión del dispositivo.

La aparición del primer terminal basado en un sistema de vídeo, una pantalla con un teclado, se retrasó casi una década, hasta que en 1969 DataPoint Corp. lanzó el DataPoint 3300 [3]. Este era un terminal que, conectándose al ordenador por la misma vía que se había usado hasta entonces para el teletipo, emulaba el funcionamiento de este, pero prescindiendo del papel. La mayor dificultad para el desarrollo de un terminal así, basado en un sistema de vídeo, estribaba en la cantidad de memoria que se necesitaba para almacenar una página (una pantalla completa) de información a fin de visualizarla en la pantalla. La capacidad podía, en algunos casos, exceder la memoria con la que contaban los ordenadores de la época. El abaratamiento de la memoria RAM (*Random Access Memory*), a partir de la introducción del Intel 1103 en 1970 [4], permitió superar ese obstáculo y popularizar el uso de los terminales basados en vídeo.

Cuando los ordenadores personales, también conocidos como microordenadores [5], comenzaron a estar al alcance del público en general, a partir de mediados de la década de los 70 y, sobre todo, en los 80, el uso de un sistema de vídeo como vía principal de comunicación con el usuario era algo estándar. Estos equipos incorporaban como parte del propio ordenador la memoria necesaria para almacenar la información a mostrar en la pantalla, así como el hardware necesario para convertir el contenido de esa memoria en una señal visible en pantalla. En este momento nace el hardware de vídeo integrado en el ordenador, cuya evolución es el objeto del presente artículo.

Al tratar el progreso en los adaptadores de vídeo, o adaptadores gráficos, destinados a ordenadores personales suele considerarse un agrupamiento en generaciones, atendiendo a la funcionalidad ofrecida por el hardware en cada momento. En la sección 2 de este artículo se ofrece una visión general sobre cómo el hardware de vídeo ha cambiado en las últimas décadas. La sección 3, centrada ya en los últimos 20 años, describe las características fundamentales de las cinco primeras generaciones de adaptadores de vídeo para ordenadores personales. En la sección 4 se detalla cómo ha cambiado el cauce interno o *pipeline* gráfico a lo largo de esas generaciones.

2 Visión general del hardware de vídeo en ordenadores personales

La evolución del hardware de vídeo utilizado en los ordenadores personales ha sido vertiginosa en las últimas décadas, llegando en la actualidad a ofrecer una potencia de cálculo muy superior a la de las costosas estaciones de trabajo[1] (*workstations*) que hasta hace veinte años copaban el segmento profesional, dedicado principalmente a tareas de CAD (*Computer Aided Design*), CAM (*Computer Aided Manufacturing*), CAE (*Computer Aided Engineering*), investigación y simulación científica, etc.

Del clásico adaptador de vídeo, dotado únicamente de la circuitería necesaria para convertir la información digital, almacenada en memoria[2] relativa a los atributos de los píxeles, en una señal analógica adecuada para ser enviada a un monitor, se ha pasado al campo de las GPU (*Graphics Processing Unit*) [6], procesadores especializados con un alto nivel de paralelismo y que eximen al microprocesador de todas las tareas gráficas. Entre ambos extremos han quedado las tarjetas de vídeo que, aparte del adaptador, ya contaban con una cierta cantidad de memoria propia, así como aquellas capaces de llevar a cabo una parte del trabajo que hasta ese momento recaía en la CPU (*Central Processing Unit*), como la aplicación de texturas, primero, y la ejecución de transformaciones, después, a través de operaciones implementadas en hardware y sobre las que el programador tenía muy poco control, más allá de establecer algunos parámetros limitados.

Las actuales GPU también cuentan con el hardware necesario para llevar a cabo todas esas tareas sin depender de la CPU pero, además, el tratamiento aplicado a cada vértice o fragmento es programable en mayor o menor medida, dependiendo del hardware concreto y la API (*Application Programming Interface*) que se utilice. Todo ello permite un control mucho mayor por parte del desarrollador a la hora de establecer transformaciones, definir algoritmos de sombreado e iluminación, aplicar texturas, etc.

2.1. Hardware de vídeo en microordenadores

Los microordenadores[3], sistemas surgidos desde mediados de la década de los 70 hasta principios de los 90 [5] con microprocesadores de 8 bits y en algunos casos de 16 bits, eran productos dirigidos al usuario doméstico, razón por la que el precio era un factor determinante. El hardware de vídeo de estas máquinas era, en la mayoría de los casos, muy simple.

---

[1] A diferencia de los *mainframes*, las *workstations* eran ordenadores diseñados su uso por una sola persona, aunque solían estar conectadas en red, contando con capacidad de procesamiento y para mostrar gráficos en alta resolución, a diferencia de los terminales conectados a los *mainframes* y los ordenadores personales de la época, que carecían en general de capacidades gráficas-

[2] Los primeros adaptadores de vídeo usados en los ordenadores personales ni siquiera contaban con una memoria propia, convirtiendo en señal de vídeo la información que se almacenaba en una cierta porción de la propia memoria principal del sistema. El único aspecto configurable era la dirección de memoria a partir de la que se leían los píxeles.

[3] En http://museopc.ujaen.es/ puede encontrarse información detallada sobre muchos de los microordenadores mencionados en esta sección.

En el caso del Commodore PET (véase la Figura 1), el primer ordenador comercializado por Commodore en el año 1977, el hardware de vídeo se diseñó a medida usando circuitos TTL discretos a fin de generar la imagen enviada al monitor integrado. Muchos otros ordenadores de la época, incluyendo los Apple I y Apple II, Jupiter ACE, Osborne 1, TRS-80 y Sinclair ZX-80, usaban este mismo enfoque para la producción de gráficos. El hardware de vídeo a medida continuó utilizándose en equipos de principios de los 80, aunque integrando toda la circuitería TTL (*Transistor-Transistor Logic*) en un único chip tipo ASIC (*Application Specific Integrated Circuit*). Este fue el caso de varios modelos de Sinclair, como los ZX-81, ZX-Spectrum y QL, con su ULA (*Uncommited Logic Array*), y de otros producidos por Thomson, Oric y Acorn.

Figura 1. Commodore PET 2001 (1977).

La circuitería de vídeo a medida tenía su contrapartida en el hardware de vídeo genérico basado en un CRTC (*Cathode Ray Tube Controller*), un circuito estándar encargado de generar las señales de sincronización y control para poder mostrar una imagen en pantalla. Uno de los CRTC más usados fue el Motorola 6845. El ordenador aportaba la memoria en la que se almacenaban los datos a partir de los cuales se generaba dicha imagen, encargándose además de todas las tareas gráficas. Algunos de los modelos

cuyo hardware de vídeo se basaba en un CRTC fueron los Commodore PET 4000/8000, Kaypro y microordenadores japoneses como el Sharp MZ-700 y Sharp X1.

El incremento en el número de fabricantes de microordenadores y de unidades vendidas provocó la aparición de circuitos integrados específicos que operaban como controladores de vídeo. Estos se encargaban no solamente de generar la imagen, como los CRTC, sino también de gestionar la configuración de los distintos modos de texto y gráficos, generar los juegos de caracteres, configurar las paletas de color, etc. Descargan, por tanto, de una cierta cantidad de trabajo al microprocesador, pero este seguía siendo el responsable de generar las imágenes a mostrar finalmente en pantalla. Uno de los controladores de vídeo más conocidos de la época era el VIC-II, al ser el que integraba el Commodore 64 que, durante mucho tiempo, fue el ordenador más vendido de la historia.

Desde mediados de los 80 en adelante la mayor parte de los microordenadores incorporan procesadores de vídeo más sofisticados, ya fuesen genéricos (fabricados por un tercero y disponibles en el mercado) o diseñados a medida. En el primer grupo estarían los VDP (*Video Display Processor*) de Yamaha, como el V9938/V9958 usado en los MSX2 y MSX2+. Del segundo forman parte integrados míticos como los "Agnus" y "Denise", diseñados para el Commodore Amiga, o el "Nick" de los Enterprise 64/128. Estos circuitos contaban normalmente con su memoria dedicada, conocida como VRAM (*Video RAM*), y operaban de forma paralela e independiente a la CPU, siendo capaces de resoluciones mucho más altas y usando miles de colores. Fueron, en su momento, lo más cercano a una GPU actual.

2.2. Hardware de vídeo en los orígenes del PC

La mayoría de los ordenadores usados actualmente tienen sus orígenes en la arquitectura del IBM PC, presentado en agosto de 1981. Incluso los Apple Mac actuales, desde el cambio introducido en el año 2006 en su fabricación, tienen la arquitectura de un PC. Aquel PC original era un sistema pensado para la empresa, razón por la que el sistema de vídeo con el que se vendía, conocido como MDA (*Monochrome Display Adapter*), fuese monocromo, al igual que el monitor. Se ponía el énfasis en la calidad del texto. No obstante, IBM ofrecía como opción cambiar el vídeo MDA por un adaptador CGA (*Color Graphics Adapter*).

CGA contemplaba dos modos gráficos: uno de baja resolución con 4 colores y otros de alta resolución con solo 2. En modo texto se disponía de una paleta de 16 colores. El adaptador de vídeo CGA podía conectarse a un monitor específico, como el MDA, pero también directamente a un TV, abriendo la puerta a usos más domésticos. IBM desarrolló CGA a partir del Motorola 6845 anteriormente mencionado, siendo por tanto un CRTC con algunas características más avanzadas, entre ellas la inclusión de 16Kbytes de memoria para almacenar los datos de la imagen.

Un año después del lanzamiento del PC, con su configuración inicial MDA/CGA, apareció el primer competidor de IBM en el campo del hardware gráfico: la empresa Hercules. Esta diseñó un adaptar de vídeo (véase la Figura 2) cuya principal característica era la de ofrecer un modo de texto compatible con MDA y un modo gráfico con mayor resolución que CGA. Dicho modo, no obstante, era monocromo, pensado más para la empresa que para su uso doméstico.

Hercules fue la primera, pero en los años siguientes muchas otras empresas se lanzaron al negocio de la fabricación de hardware de vídeo para PC. La competencia entre dichas empresas se concentró inicialmente en ofrecer cada vez mayor resolución, tanto en número de píxeles como en número de colores en pantalla. Estos avances se reflejaron en la comercialización de los conocidos adaptadores EGA (*Enhanced Graphics Adapter*), VGA (*Video Graphics Array*), SVGA (*Super VGA*), etc. La principal diferencia entre el hardware ofrecido por cada empresa estribaba en la cantidad de memoria que incorporaba el adaptador, un factor que restringía los modos gráficos que podía representar.

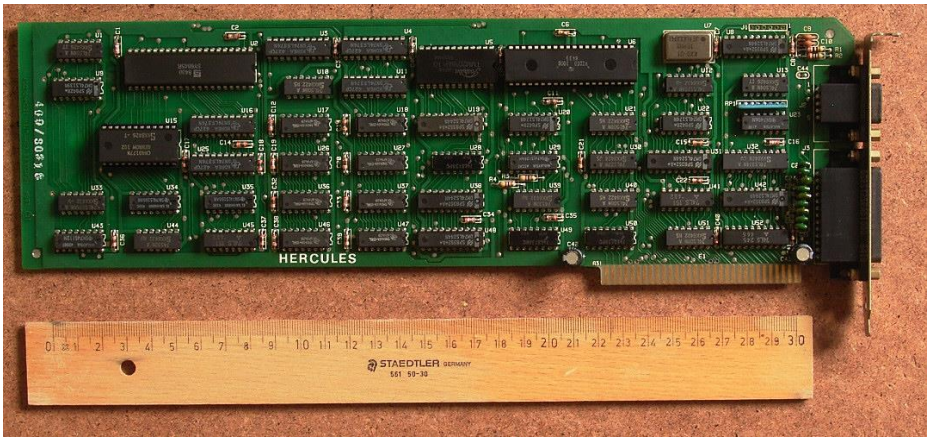


Figura 2. Adaptador de vídeo Hercules comercializado en 1984.
Fotografía de Rainer Knäpper reproducida bajo Licence Art Libre.

El concepto de adaptador de vídeo basado en el CRTC Motorola 6845, con una cantidad mayor de memoria, un generador de caracteres y gestión de paletas de color, era prácticamente todo lo que aportaba este hardware de vídeo, manteniendo la compatibilidad hacia atrás con CGA. Desde el momento en que los fabricantes incorporaron al adaptador de vídeo capacidades adicionales, en principio un procesador capaz de aplicar ciertas funciones automáticamente a los píxeles de una imagen, podemos comenzar a hablar de GPU. A partir de entonces el hardware gráfico no se limita a tomar el contenido de la memoria de vídeo y enviarlo a una pantalla, sino que interviene en el proceso de generación de la propia imagen.

3 Generaciones de los adaptadores de vídeo

Hasta hace pocos años todos los procesadores de tarjetas gráficas, lo que se conoce como GPU, contaban con un conjunto de funciones fijas para la generación de imágenes. Si imaginamos que la GPU es como una tubería en la que introducimos por un extremo, el de entrada, un flujo de datos, normalmente coordenadas de vértices, y obtenemos por el otro, el de salida, una imagen, esas funciones fijas permitían ajustar,

siempre dentro de unos límites preestablecidos, aspectos como la iluminación, las transformaciones y aplicación de texturas. Sería, por tanto, como tener a lo largo de la tubería una serie de interruptores que pueden tomar una de un conjunto limitado de posiciones.

El nacimiento de las GPU programables, a partir de 2001, supone un salto cualitativo crucial, ya que el programador puede cambiar porciones de esa hipotética tubería sustituyendo las funciones fijas por funciones a medida, eliminando cualquier límite en cuanto a los efectos gráficos que pueden generarse en tiempo real. Esas funciones se denominan coloquialmente sombreadores o *shaders* [7], aunque realmente su función no es únicamente la de aplicar sombreados. Una función de este tipo se envía como código ensamblador al controlador de la tarjeta gráfica, encargándose esta de generar el código ejecutable que, alojado en la propia GPU, procesará cada uno de los vértices o fragmentos de una escena.

Para comprender la importancia que tiene la existencia de GPU programables, y situar este hecho en su contexto actual, lo mejor es echar la vista atrás y analizar qué ofrecían las GPU del pasado. En realidad, se trata de un pasado muy reciente, ya que la historia de las GPU tiene apenas dos décadas. Hasta la aparición de los primeros adaptadores de vídeo con GPU, en 1998, todo el trabajo de generación de gráficos recaía en la CPU del ordenador, limitándose el mencionado adaptador [8] a generar la señal de vídeo, tal y como se apuntaba anteriormente. Todo el cálculo lo efectuaba la CPU y el hardware de vídeo era poco más que un DAC (*Digital Analogic Converter*).

Es a partir de la presentación de las Voodoo3 y TNT, denominaciones del hardware gráfico de las empresas 3dfx y nVidia[4], cuando se comienza a hablar de las GPU. A medida que estas fueron incorporando capacidades adicionales se empezaron a agrupar en familias y, posteriormente, en generaciones. Hoy en día las generaciones reconocidas históricamente son las mencionadas en las subsecciones siguientes.

3.1. La primera generación

Se trataba de GPU con capacidad para aplicar sobre los píxeles una serie de operaciones matemáticas simples, con el objetivo de calcular el color final facilitando así la aplicación de texturas. Además, *rasterizaban* [9] directamente los triángulos, de manera que se eximía a la CPU de la manipulación de píxeles individuales.

La aplicación de transformaciones a los vértices, y cualquier otra tarea que no fuese la rasterización y aplicación de una o dos texturas (dependiendo de los modelos) seguía quedando completamente en manos de la CPU. No obstante, el trabajo que realizaba la GPU mejoraba de manera notable el rendimiento, respecto a sistema sin GPU, llegando a alcanzarse tasas de entre 6 y 9 millones de polígonos por segundo.

Esta primera generación abarca temporalmente desde 1998 hasta mediados de 1999, formando parte de ella productos como las ATI Rage, nVidia RIVA TNT y la Voodoo3 (Figura 3) de la hoy desaparecida empresa 3dfx Interactive [10]. Hay que tener en cuenta que estas GPU no eran programables, sino "configurables", es decir, el programador podía establecer una serie de parámetros que determinaban cómo se efectuaría el rasterizado o la manera en que se aplicarían las texturas, pero nada más. El *pipeline*

[4] En http://www.tomshardware.com/reviews/nvidia,87.html podemos encontrar una comparativa de la época de estos dos productos.

de estas GPU, por tanto, tenía una estructura fija que llevaba a cabo siempre las mismas operaciones, con unos parámetros ligeramente variables.

Como curiosidad, en su época se denominaba a estos adaptadores "aceleradoras 3D", para distinguirlas de los adaptadores de vídeo corrientes. Algunos de ellos, como era el caso de la Voodoo3, incluían funciones de procesamiento en 3D pero, por el contrario, carecían de la capacidad para operar en 2D, por lo que este tipo de gráficos seguían siendo generados al completo por la CPU.

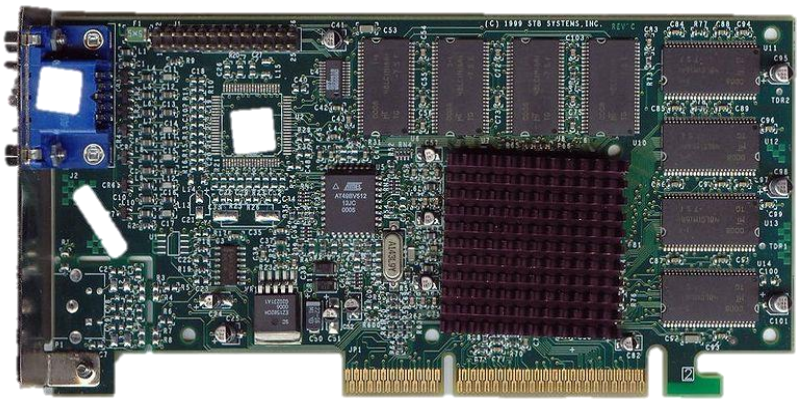


Figura 3. Voodoo3 con conector AGP[5].

3.2. La segunda generación

Esta generación se inicia con la aparición de las primeras GPU capaces de llevar a cabo operaciones de transformación e iluminación 3D, funciones que se agregan a la rasterización y aplicación de texturas. Abarca desde finales de 1999 hasta la aparición de la siguiente generación, ya en 2001.

Los representantes más destacables de esta generación son las ATI Radeon 7500, S3 Savage3D y nVidia GeForce 256 (anunciada por nVidia como la primera GPU del mundo, según se aprecia en la Figura 4) y, posteriormente, la GeForce2. Era un hardware capaz de procesar entre 15 y 25 millones de polígonos por segundo

El cauce de procesamiento de estos dispositivos seguía siendo configurable, no programable, si bien integraba funciones matemáticas más complejas y, además, tenía la

[5] Imagen de Wikimedia Commons.

posibilidad de utilizar aritmética con signo y de mayor precisión. Además de los cambios en el cauce gráfico, cada generación introdujo mayor integración y velocidad de la GPU, nuevas interfaces de conexión (PCI, AGP, PCIe, etc.) y avances en la velocidad de la memoria. Son aspectos, no obstante, que quedan fuera del ámbito de este trabajo.

Figura 4. Anuncio de la GeForce 256 por parte de nVidia.

3.3. La tercera generación

Es a principios de 2001, con la presentación del circuito integrado GeForce3, cuando aparece el primer dispositivo que podría calificarse realmente como GPU, al incorporar por primera vez la posibilidad de programar una parte del *pipeline*. También forman parte de esta generación, que se extiende hasta 2003, productos como la ATI Radeon 8500 (Figura 5), la GeForce4 y el hardware gráfico fabricado específicamente para la consola Xbox de Microsoft.

Los procesos de rasterización y aplicación de texturas siguen ejecutándose en unidades configurables de función fija, pero las transformaciones aplicadas a los vértices de la geometría pasan a ser programables. Aparecen los denominados *vertex shaders*, que abren las puertas a que, por primera vez, los programadores definan funciones a medida que serán ejecutadas no por la CPU, sino por la GPU.

A pesar de la posibilidad de definir *vertex shaders*, estos primeros diseños de cauce programable tenían muchas limitaciones y restringían, por ejemplo, el número de parámetros de entrada o el número de instrucciones que podía tener como máximo un *shader*. Además, estos programas se ejecutaban siempre de manera secuencial, lo que implicaba que no podían definirse bucles.

Figura 5. Circuito integrado ATI Radeon 8500.

Los productos de esta generación eran capaces de procesar entre 30 y 60 millones de polígonos por segundo, volviendo a multiplicar la potencia de la generación anterior que, a su vez, había multiplicado la de la primera generación.

La aparición del cauce parcialmente programable en el hardware gráfico marca también la aparición de los primeros lenguajes de alto nivel para la programación de *shaders*, incorporados en DirectX 8.0 y en OpenGL a través de una extensión. Previamente era preciso recurrir al lenguaje ensamblador propio de cada GPU.

3.4. La cuarta generación

Si en la tercera generación se añadió la capacidad para programar la unidad encargada de procesar los vértices de la geometría, en la cuarta se extiende esa capacidad a otro elemento del *pipeline*: la unidad encargada del *rastering*, incluyendo la texturización, sombreado y resolución de visibilidad. Nacen así los denominados *fragment shaders* o *pixel shaders*.

Las GPU de esta generación son casi completamente programables, pero tienen una estructura heterogénea en la que se dedican unos núcleos de procesamiento concretos a trabajar con vértices y otros a tratar fragmentos (*pre-píxeles*). En ambos casos el número de unidades es reducido, por ejemplo 4 núcleos de procesamiento de fragmentos en los primeros diseños, y así lo son también los programas que es posible escribir, con una longitud máxima de 12 instrucciones y no pudiendo operar con más de 4 texturas de manera simultánea.

Son miembros de esta generación las familias GeForce FX de nVidia y Radeon 9x00 de ATI, con capacidad para procesar hasta 200 millones de polígonos por segundo. Temporalmente hablando, esta generación se solapa parcialmente con la anterior, abarcando los años 2002 y 2003.

DirectX 9 [11] contemplaba la programación de los nuevos *pixel shaders*, generando un código que, con posterioridad, era compilado y optimizado por parte del controlador específico del fabricante, según el esquema de la Figura 6. Ese código, ya ejecutable, era el que se enviaba al hardware. OpenGL, por su parte, contaba con una extensión adicional, denominada `ARB_fragment_program`, que ofrecía una interfaz similar.

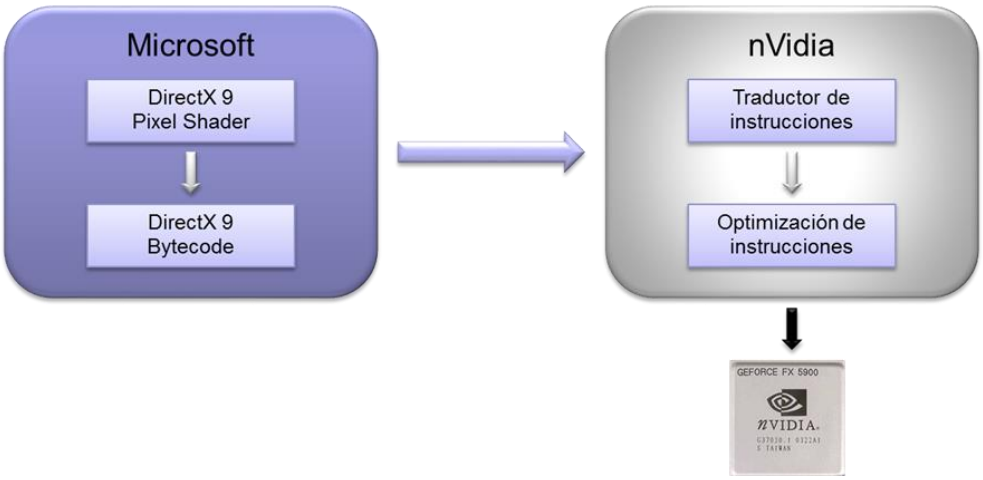


Figura 6. Pasos desde la escritura de un *pixel shader* en DirectX 9 hasta la ejecución.

3.5. Quinta generación y posteriores

Algunos autores sitúan la quinta generación de GPU aproximadamente a partir de 2009, mientras que otros hablan de una quinta generación entre los años 2004 y 2005, denominando sexta generación a la posterior, que abarca desde entonces hasta el presente. En cualquier caso, lo importante es conocer las características básicas de una generación y otra.

En los productos aparecidos entre 2004 y 2005 se introducen algunos avances en la tecnología ya existente, aunque sin que se produzca un salto sustancial en la misma. En estos productos los *vertex shaders* tienen, por primera vez, acceso a la memoria de la tarjeta de vídeo, no solamente a la información de los vértices que están procesándose. Por su parte, los *fragment shaders* pueden tener una mayor longitud y, también por primera vez, pueden ejecutar saltos y, en consecuencia, abren la puerta al uso de bucles. A esta generación corresponden la familia 6800 de nVidia y la ATI X1800.

Con 2006 llegan al mercado las GPU que componen la quinta generación, cuyos máximos representantes son los núcleos G80 de nVidia y R600 de ATI, usados en varias gamas de producto de ambos fabricantes. Las características más destacables de esta generación, que vienen acompañadas de una nueva versión de la especificación de

shaders y también una actualización de DirectX, son las siguientes:

- Unificación de los núcleos de la GPU [12], de forma que no están especializados en tratamiento de vértices o píxeles, como en las generaciones previas, sino que tienen una función más genérica y pueden de esta forma distribuir mejor el trabajo que requiera cada programa. Es el modelo *Unified Shading Architecture*.
- Aparición de un nuevo tipo de programa para GPU, denominado *geometry shader* [13], incrementando así las funciones que pueden ejecutarse en la GPU y que, hasta el momento, se llevaban a cabo en la CPU. Tras estos aparecieron los denominados *tessellation shaders* y más recientemente los *compute shaders*. Estos últimos permiten implementar computación de propósito general mediante *shaders*.
- Aunque el término GPGPU (*General Purpose-computation on Graphics Processing Units*) [14], que hace referencia a la ejecución en la GPU de algoritmos de propósito general, realmente nació con la tercera generación de GPU, para poder ejecutar algoritmos de propósito general mediante los *shaders* disponibles en ese momento había que recurrir a determinadas técnicas y trucos que no resultaban sencillos. La aparición de lenguajes como CUDA y OpenCL, conjuntamente con la quinta generación de GPU, simplificó enormemente esta tarea. La flexibilidad a la hora de programar la GPU, con la aparición de dichos lenguajes de más alto nivel, da lugar al nacimiento del GPGPU moderno o *GPU Computing*.

Desde la aparición de los citados núcleos G80 y R600 en adelante, la propuesta de los fabricantes ha ido básicamente encaminada a incrementar el número de núcleos de procesamiento con que cuentan las GPU, aumentar la cantidad de memoria integrada y acelerar todo el funcionamiento incrementando las frecuencias de reloj, gracias a la mayor escala de integración en la fabricación de circuitos integrados. Un exponente clásico de la potencia de esta quinta generación es la nVidia GTX 285. Esta dispone de 240 procesadores de *shaders* operando a 1.5 Ghz y una memoria GDDR3 de 1 GB a la que se accede a través de un bus de 512 bits. En la actualidad, el hardware de consumo más potente cuenta con varios miles de núcleos y 4 o más GB de memoria GDDR5.

4 Evolución del *pipeline* gráfico

A la vista de la historia esbozada en la sección previa, se deduce que las GPU, y más concretamente el cauce interno o *pipeline* [15] por el que va transcurriendo la información gráfica hasta terminar su procesamiento, ha ido cambiando de una manera paulatina pero constante.

4.1. Pipeline gráfico: etapas configurables

Partiendo de los datos sobre geometría de la escena que facilita la aplicación, para obtener una imagen es necesario efectuar una serie de transformaciones a los vértices, aplicar parámetros de iluminación, reconstruir las primitivas en 2D, rasterizar esas primitivas y asociar información sobre texturas, llevar a cabo distintas operaciones sobre estos fragmentos para, finalmente, obtener los píxeles que se almacenarán en el buffer y que formarán la imagen a enviar a la pantalla.

Antes de que apareciesen las primeras GPU los adaptadores de vídeo, tal y como se apuntaba anteriormente, se limitaban a convertir la información digital de los píxeles para enviarla a la pantalla, actuando en este sentido como un *frame buffer*. El esquema de la Figura 7 muestra el cauce gráfico con este tipo de hardware. Como puede apreciarse, todo el trabajo recaía en la CPU.

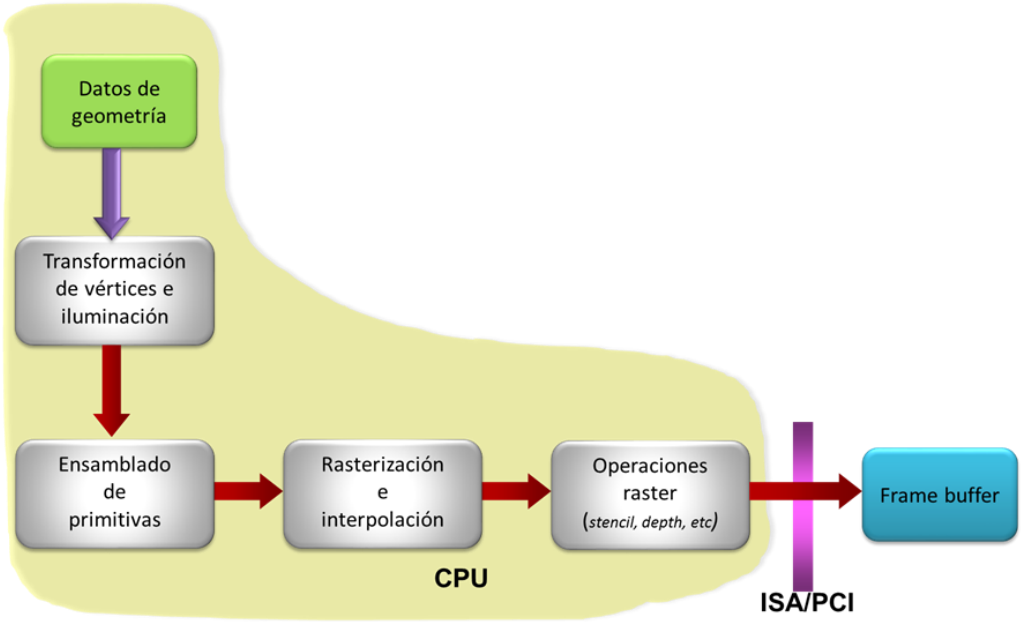


Figura 7. Cauce gráfico de los adaptadores de vídeo primitivos en ordenadores personales.

La primera generación de GPU eximió a la CPU de las operaciones relacionadas con el *rastering* y la aplicación de texturas. El cauce gráfico correspondería al representado en la Figura 8. Los bloques de la GPU encargados de la rasterización, interpolación y ejecución de operaciones *raster* tienen codificada una función fija parametrizable, de forma que la CPU configura hasta cierto punto esas etapas del *pipeline*.

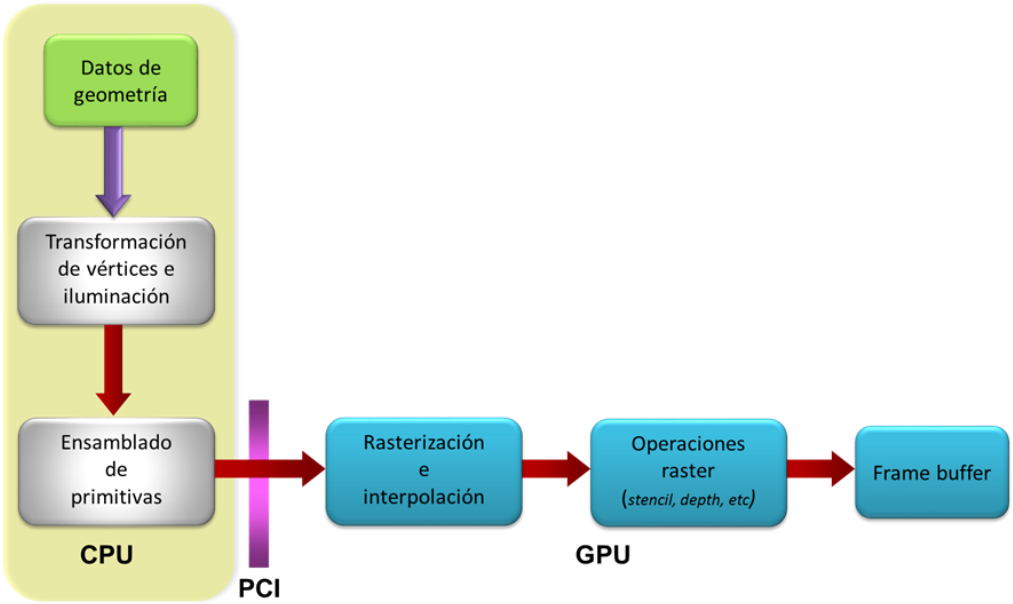


Figura 8. Cauce gráfico típico en la primera generación de GPU.

Es con la segunda generación de GPU cuando se pasa a tener un cauce que, como puede verse en el diagrama de la Figura 9, se ejecuta al completo en el adaptador de vídeo. La aplicación se limita a facilitar los datos de geometría. Cada una de las etapas cuenta con una serie de parámetros configurables, tal y como se indicó anteriormente.

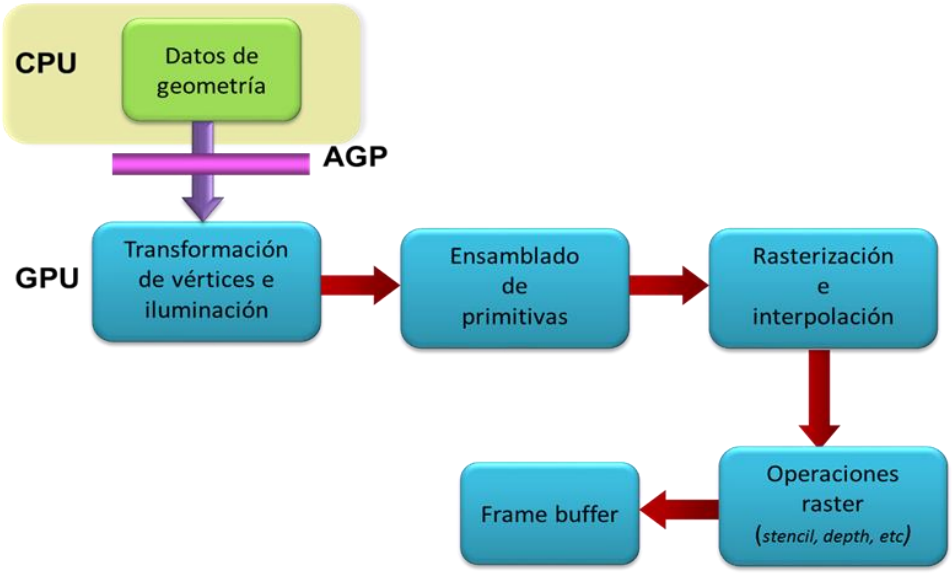


Figura 9. Estructura del cauce gráfico correspondiente a la segunda generación.

4.2. Pipeline gráfico: etapas programables

La tercera generación de GPU introduce en el cauce gráfico el primer elemento programable: los *vertex shaders*. Estos pequeños programas, que se ejecutan en el interior de la GPU y no en la CPU, pueden llevar a cabo operaciones sobre los vértices que no puedan efectuarse en la antigua etapa configurable. Un ejemplo sería la utilización de un algoritmo de iluminación que no esté implementado directamente en el hardware.

Como se aprecia en la Figura 10, la etapa del cauce en la que se ejecutan los *vertex shaders* está en una posición intermedia entre la etapa configurable de transformación y la etapa de ensamblado de primitivas. Al activar un *vertex shader* se está anulando por completo la etapa fija del *pipeline*, por lo que es necesario escribir código para sustituir todas las funciones que dicha etapa efectuaba: transformación de vértices y normales, normalización de normales, generación y transformación de coordenadas de texturas, cálculos de iluminación, material y color para cada vértice.

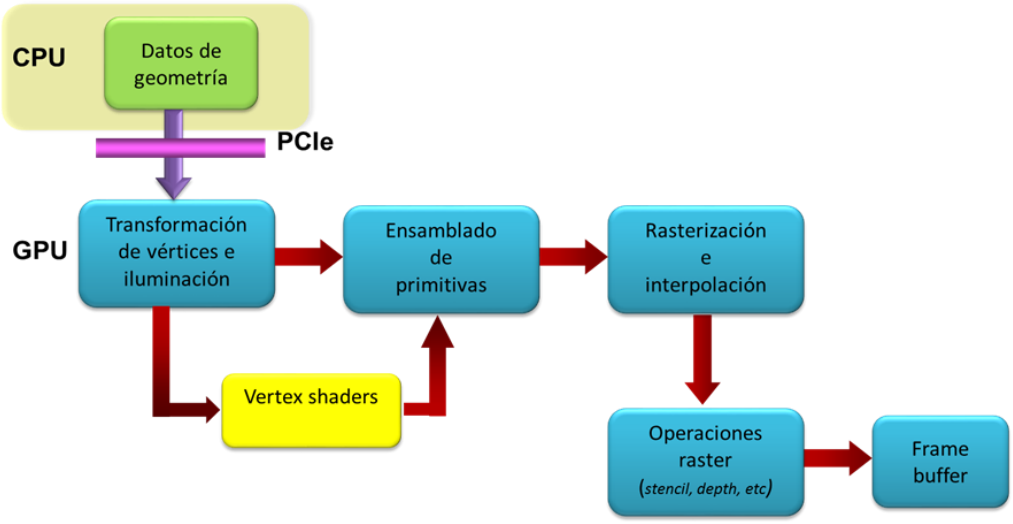


Figura 10. En la tercera generación apareció la primera etapa programable.

Aunque técnicamente el número de vértices que serán procesados en paralelo por un *vertex shader* dependerá del número de unidades de cálculo con que cuente la GPU, desde una perspectiva puramente lógica al implementar un programa de este tipo debe asumirse que va a ejecutarse de manera simultánea sobre todos los vértices de la escena. Esto implica que los cálculos de un vértice no pueden, en ningún caso, afectar a otros vértices. La ejecución es masivamente paralela en el hardware actual.

La relación de entrada-salida en un *vertex shader* es siempre de uno a uno, es decir, son programas que toman como entrada un vértice y generarán como resultado otro, nunca se elimina ni se genera nueva geometría.

La introducción de los *fragment shaders*, ya en la cuarta generación de GPU, extendió una vez más el cauce gráfico como queda reflejado en la Figura 11. Existe una nueva etapa programable, en este caso interpuesta entre las etapas fijas de rasterización

e interpolación y la encargada de ejecutar operaciones sobre el *raster* justo antes de llegar al *frame buffer*.

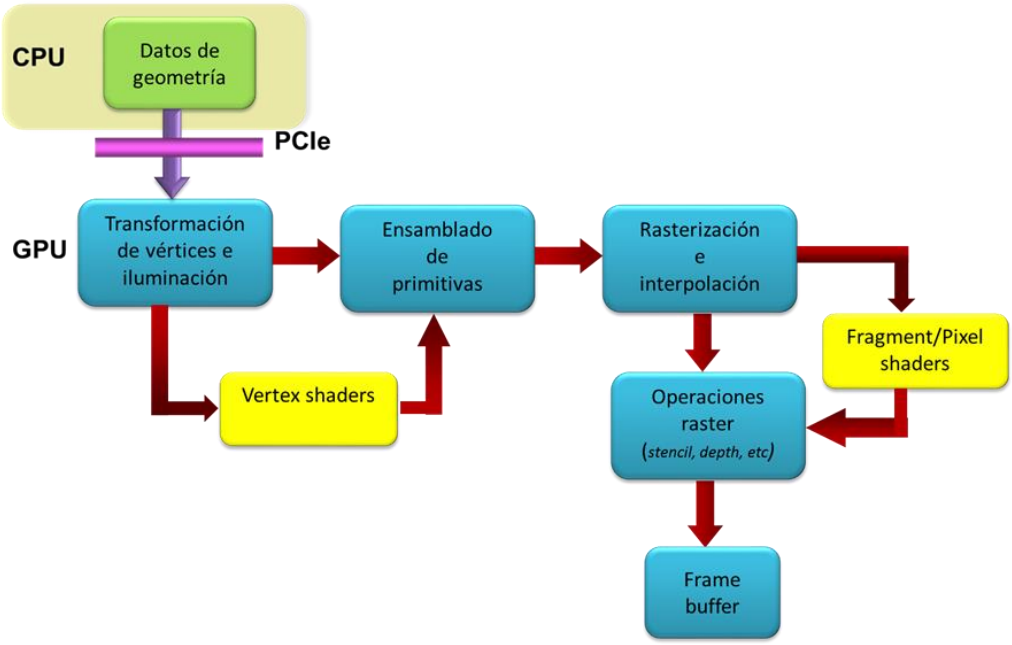


Figura 11. Las etapas programables de la cuarta generación de GPU.

Al igual que ocurre con los *vertex shaders*, al programar un *fragment shader* hay que asumir que este procesará en paralelo todos los pre-píxeles de la imagen, por lo que no pueden producirse efectos colaterales entre fragmentos. Es decir, el resultado de aplicar un cierto tratamiento a un fragmento no puede usarse como base para modificar otro fragmento distinto.

El *fragment shader* recibe los fragmentos una vez que se han llevado a cabo tanto la rasterización como la interpolación de los vértices, siendo tarea suya la aplicación de texturas y generación de otros efectos visuales, decidiendo qué color se asignará finalmente a cada uno de los píxeles que se enviarán al *frame buffer*.

En el *pipeline* de la Figura 11, que corresponde al de la cuarta generación, hay dos etapas programables pero también muchas limitaciones. Estas afectan tanto a la extensión máxima de los *shaders* como a las operaciones que pueden efectuar, sin estructuras de control ni acceso a la memoria del sistema de vídeo. No obstante, de forma global este cauce representa un avance muy significativo respecto al de las GPU de generaciones previas.

Dichas limitaciones desaparecen, en gran parte, en el cauce gráfico de las últimas generaciones de GPU, hasta llegar a las actuales, gracias a la aparición de una arquitectura unificada de núcleos de procesamiento. Dicha arquitectura, además, ha ido acompañada de nuevas especificaciones del modelo de programación de *shaders*, en las que se abren las puertas a la creación de programas más grandes y complejos, con capacidad de compartir información a través de la memoria integrada en la GPU.

En la Figura 12 se ha representado este nuevo cauce, en el que aparecen por primera vez los *geometry shaders*. Se trata de programas que interactúan con los *vertex shaders* y tienen la capacidad para generar nueva geometría a partir de los vértices originales.

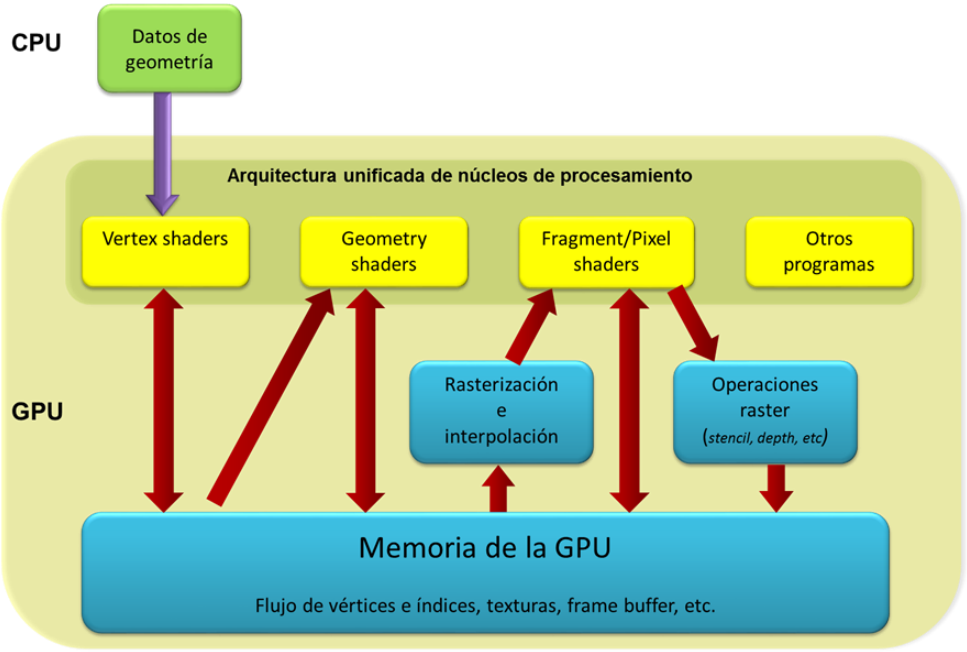


Figura 12. El *pipeline* de las actuales GPU es mucho más flexible.

A pesar de que el rendimiento gráfico se ha incrementado de manera constante, en parte debido a la evolución del *pipeline* de las GPU, para el programador esta nueva arquitectura representa casi una vuelta al pasado, cuando tenía que programar por sí mismo los algoritmos de transformación de vértices, sombreado, aplicación de texturas, etc. La diferencia estriba en que ahora esos algoritmos se ejecutan en la GPU y no en la CPU, aprovechando el gran paralelismo que otorga la disponibilidad de cientos de núcleos de procesamiento.

Obviamente el uso de *shaders* en el desarrollo de una aplicación gráfica es totalmente opcional. El programador puede seguir concentrándose únicamente en la generación de la geometría de la escena y dejar el resto del trabajo en las etapas configurables del *pipeline* clásico.

5 Conclusiones

El uso de terminales de vídeo para facilitar la comunicación interactiva con los sistemas de cómputo surgió hace medio siglo, siendo el medio preferente desde la aparición del ordenador personal, cuya expansión se inició a principios de la década de los ochenta. Desde entonces, el hardware a cargo de producir la imagen que los usuarios ven en sus

pantallas, tanto texto como gráficos, ha ido evolucionando y haciéndose cada vez más sofisticado. La consecuencia más destacable de este progreso, aparte de la patente mejora en la calidad de los gráficos y la velocidad con que pueden ser generados, es el hecho de que se descarga a la CPU de una tarea que consumía mucho tiempo de procesamiento. De hecho, en muchos microordenadores la generación de gráficos detenía por completo cualquier otra tarea, absorbiendo por completo al microprocesador.

En el presente artículo se ha hecho un recorrido por algunos de los hitos más destacables de la evolución del hardware gráfico, desde sus inicios, cuando ni siquiera incorporan memoria propia, hasta los adaptadores de vídeo modernos que no ofrece únicamente funciones gráficas, sino también posibilidades de cómputo de altas prestaciones mediante técnicas GPGPU. Incluso el hardware de vídeo integrado en los microprocesadores de última generación, como la familia Intel Core iX, ofrece un nivel de prestaciones que era prácticamente impensable hace apenas dos décadas.

Referencias

1. Plugge, W. R., & Perry, M. N. (1961, May). American Airlines' Sabre electronic reservations system. In Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference (pp. 593-602). ACM.
2. R. M., Fano and P. J. Corbato, “Time-Sharing on Computers” in Scientific American, Information (San Francisco, 1966), 76-95.
3. Walther, G. H. (1974, May). On-line user-computer interface: The effects of interface flexibility, terminal type, and experience on performance. In Proceedings of the May 6-10, 1974, national computer conference and exposition (pp. 379-384). ACM.
4. Sideris, G. (1973). Intel 1103-MOS memory that defied cores. Electronics, 46(9), 108-113.
5. Charte, F. (2011). El pasado de la computación personal: historia de la microinformática. ISBN: 978-84-8439-575-1.
6. Chen, J. Y. (2009, December). GPU technology trends and future requirements. In Electron Devices Meeting (IEDM), 2009 IEEE International (pp. 1-6). IEEE.
7. Guenter, B., Knoblock, T. B., & Ruf, E. (1995, September). Specializing shaders. In Proceedings of the 22nd annual conference on Computer graphics and interactive techniques (pp. 343-350). ACM.
8. Wilton, R. (1987). The programmer's guide to PC and PS/2 video systems: maximum performance from the EGA, CGA, HGC, and other graphics adapters. Microsoft Press.
9. Foley, J. D., & Van Dam, A. (1982). Fundamentals of interactive computer graphics (Vol. 2). Reading, MA: Addison-Wesley.
10. Eccles, A. The Diamond Monster 3Dfx Voodoo 1, Gamespy Hall of Fame, 2000.
11. Pipeline stages in Direct3D. https://msdn.microsoft.com/en-us/library/bb205123.aspx.
12. Lindholm, E., Nickolls, J., Oberman, S., & Montrym, J. (2008). NVIDIA Tesla: A unified graphics and computing architecture. IEEE micro, 28(2).
13. Kazakov, M. (2007, October). Catmull-Clark subdivision for geometry shaders. In Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa (pp. 77-84). ACM.
14. Charte, F., Rivera, A.J., Pulgar, F. J., del Jesus, M.J. Explotación de la potencia de procesamiento mediante paralelismo: un recorrido histórico hasta la GPGPU. Enseñanza y Aprendizaje de Ingeniería de Computadores, 6: 19-33 (2016). http://hdl.handle.net/10481/41910.
15. Blinn, J. (1996). Jim Blinn's corner: a trip down the graphics pipeline. Morgan Kaufmann.

DESDE EL PUPITRE
(Experiencias de estudiantes)

Problema del Viajante de Comercio con GPU

José Rafael Cenit Röleke, M. G. Arenas

Departamento de Arquitectura y Tecnología de Computadores.
ETSI Informática y de Telecomunicación. Universidad de Granada
Granada, España

ticjoserafael@gmail.com, mgarenas@ugr.es

Abstract. Este documento presenta el trabajo realizado en la ETSIIT de la Universidad de Granada. Se pone de manifiesto la ventaja y potencial de la programación paralela utilizando la tecnología CUDA para resolver un problema de optimización como es el Viajante de Comercio. Se analizará el rendimiento con diferentes entradas para el problema. A su vez este documento pretende ser un ejemplo de cómo resolver un problema mediante la tecnología que nos brinda la GPU. Finalmente este documento muestra un estudio comparativo de distintas versiones del algoritmo paralelo que muestran las ventajas en prestaciones de la GPU frente a la CPU para la asignatura Arquitectura y Computación de Altas Prestaciones.

Keywords: gpu, cuda, viajante comercio, tsp

Abstract. This document presents the work undertaken in the ETSIIT of the University of Granada. It manifest the advantage and potential of the parallel programming using CUDA technology for solving an optimization problem like the Travelling Salesman Problem. The performance will be analyzed with different inputs for the problem. At the same time this document pretends to be an example for how to solve a problem using the technology that the GPU has to offer. Finally this document shows a comparative studio for the different versions of the parallel algorithm showing the advantages of the GPU versus the CPU for the subject “Arquitectura y Computación de Altas Prestaciones”.

Keywords: gpu, cuda, travelling salesman, tsp

1 Introducción

Este documento presenta una aplicación distinta de la programación paralela utilizando una GPU y la tecnología CUDA [1] para resolver el problema del Viajante de Comercio, además de ser una práctica para la asignatura Arquitectura y Computación de Altas Prestaciones de la Universidad de Granada.

Se trata de un proyecto práctico de carácter investigador que puede ser perfectamente utilizado en el ámbito docente para la mejor comprensión de esta nueva forma de computación paralela.

En las diversas asignaturas que se cursan en los primeros años de la carrera como pueden ser Arquitectura de Computadores o Estructura de Computadores se estudian los distintos grados de paralelismo como pueden ser el paralelismo a nivel de instrucción ILP (Instruction Level Paralelism) y el paralelismo a nivel de datos DLP (Data Level Paralelism). Se utilizan herramientas de programación de memoria compartida como OpenMP.

Otras asignaturas como Sistemas Concurrentes y Distribuidos impartidas por el departamento de Lenguajes y Sistemas Informáticos muestran la programación paralela mediante el uso de sistemas de paso de mensajes como MPI, OpenMPI y variables compartidas.

El presente proyecto se enfoca en el estudio de la explotación de la tecnología CUDA impartida en la asignatura Arquitectura y Computación de Altas Prestaciones de la especialidad Ingeniería de Computadores para afianzar conceptos importantes como es el *mapeo* de las hebras en CUDA aplicado al problema del Viajante de Comercio. Los alumnos podrán proponer modificaciones del código y analizar el rendimiento.

2 Estado del Arte

En el año 2006 la empresa de procesadores gráficos NVIDIA lanza al mercado la primera GPU capaz de renderizar gráficos en 3D y que además incluye la posibilidad de ejecutar programas escritos en el lenguaje C utilizando el modelo de programación CUDA.

Desde entonces NVIDIA ofrece la capacidad de sus tarjetas gráficas para grandes centros de datos y otras instituciones tanto científicas como gubernamentales, siendo energéticamente eficientes para el procesamiento que son capaces de realizar. Además se utilizan versiones reducidas para potenciar las aplicaciones tanto de teléfonos móviles, ordenadores portátiles y tablets entre otros.

Los numerosos núcleos de los que dispone una tarjeta gráfica no solo se pueden utilizar para dibujar gráficos sino que además permiten ejecutar programas diversos, dando lugar al auge en nuestros días del denominado GPGPU computing.

3 Arquitectura y organización de la GPU

A continuación explicaremos la arquitectura y organización lógica de una GPU con CUDA tal y como se puede apreciar en la Figura 1.

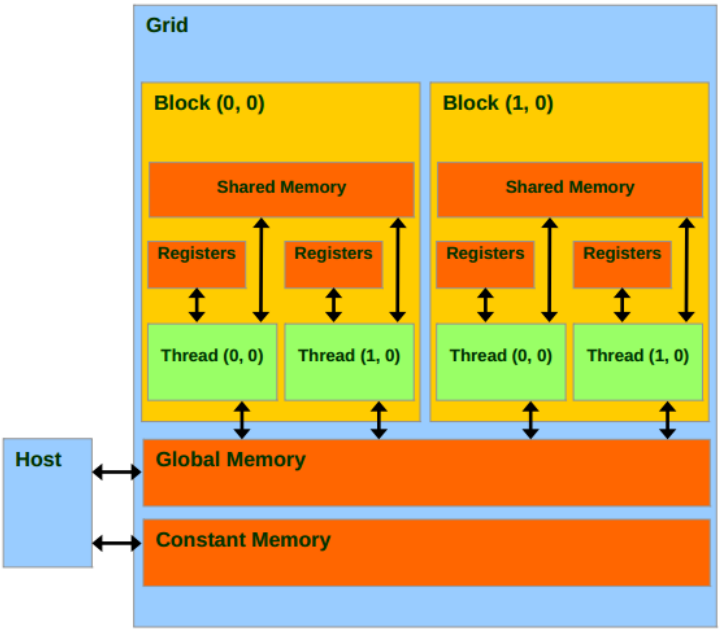


Fig. 1. Ejemplo de la arquitectura de memoria CUDA [2].

El primer elemento lógico que nos encontramos en CUDA es el hilo de ejecución o *thread*. En segundo lugar nos encontramos con el concepto de bloque o *block* y el tercer elemento lógico es rejilla o *grid*. Esta forma de organización está relacionada en cómo se gestionan los recursos dentro del dispositivo o *device*.

Tanto el número de bloques por *grid* como el número de hebras por bloque está limitado según las características físicas del dispositivo y/o de la generación del mismo. La denominada *compute capability* nos indica los límites del *device*.

Como podemos observar en la Figura 2 el número máximo de *threads* por bloque es de 1024 para todas las versiones de *compute capability*. Tanto los bloques como los *threads* pueden organizarse de forma unidimensional, bidimensional y/o tridimensional. Los *grids* pueden organizarse de forma unidimensional y/o bidimensional.

Esta plasticidad de la tecnología CUDA nos permite adaptar la estructura lógica del *grid* a la estructura de datos del problema que nos ocupe en ese momento. Por ejemplo si vamos a realizar un filtro sobre imágenes 2D lo adecuado sería organizar los bloques y las hebras de forma bidimensional puesto que la correspondencia seria 1:1. A este proceso se le denomina *mapping* o mapeo.

	FERMI GF100	FERMI GF104	KEPLER GK104	KEPLER GK110
Compute Capability	2.0	2.1	3.0	3.5
Threads / Warp	32	32	32	32
Max Warps / Multiprocessor	48	48	64	64
Max Threads / Multiprocessor	1536	1536	2048	2048
Max Thread Blocks / Multiprocessor	8	8	16	16
32-bit Registers / Multiprocessor	32768	32768	65536	65536
Max Registers / Thread	63	63	63	255
Max Threads / Thread Block	1024	1024	1024	1024
Shared Memory Size Configurations (bytes)	16K 48K	16K 48K	16K 32K 48K	16K 32K 48K
Max X Grid Dimension	2^16-1	2^16-1	2^32-1	2^32-1
Hyper-Q	No	No	No	Yes
Dynamic Parallelism	No	No	No	Yes

Compute Capability of Fermi and Kepler GPUs

Fig. 2. Tabla de especificaciones de cada versión de la capacidad de cómputo (*Compute Capability)* [3].

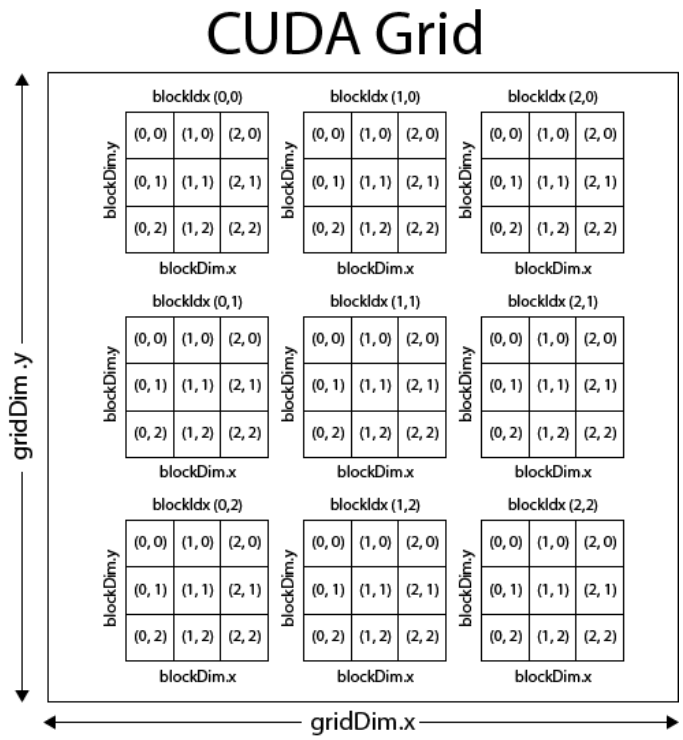


Fig. 3. Ejemplo de organización bidimensional en CUDA [4].

Cuando lanzamos un conjunto de hebras en CUDA la unidad mínima de ejecución física denominada *warp* ejecuta 32 hebras. Es decir aunque nuestro programa solo lanzase una sola hebra realmente se lanzan 32 hebras simultaneas donde la ejecución es la misma para todas ellas porque comparten el mismo contador de programa.

El programa que ejecuta cada hebra se denomina *kernel*. Este programa es ejecutado de la misma forma por cada hebra. Antes de lanzar un *kernel* necesitamos haber asignado previamente la memoria en el *device* copiando en dicha zona de memoria los datos que necesitemos. Esto es, se copian los datos del *host* al *device*.

Una vez finalizada la ejecución de nuestro *kernel* se realiza la operación inversa, se copian los resultados del *device* al *host*.

4 Proyecto: optimización del TSP mediante la GPU

Este proyecto aborda la computación en GPU para resolver el problema del viajante de comercio nombrado de aquí en adelante por sus siglas en inglés (TSP) [5].
El TSP es un problema que se encuentra en la categoría NP-Completo puesto que no se conocen algoritmos que sean capaces de generar la solución óptima en tiempo polinomial [6].

Es un problema combinatorio que se basa en: dadas varias ciudades, encontrar el camino más corto que las recorra todas una sola vez volviendo a la ciudad origen. El amplio estudio del TSP dentro de las ciencias de la computación lo convierten en un problema emblemático que tiene repercusiones y aplicaciones para la resolución de problemas de la vida real.

Las aplicaciones del TSP son múltiples: la planificación, la logística, la fabricación de circuitos integrados, etc. Por ejemplo uno puede pensar en un brazo robótico industrial que realiza puntos de soldadura. Las ciudades en este caso serían cada uno de los puntos de soldadura y la resolución del problema nos daría la secuencia de puntos que tendría que soldar el brazo robótico para minimizar el tiempo total de este trabajo.

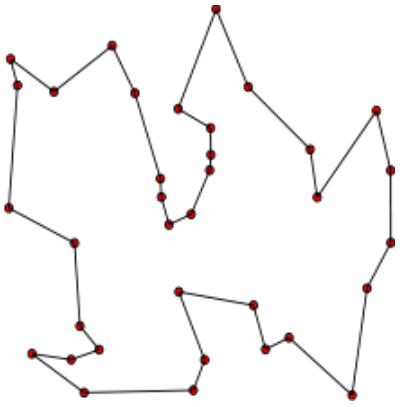


Fig. 4. Ejemplo de circuito obtenido para el TSP [7].

Dada la naturaleza del problema del TSP se ha desarrollado un algoritmo que haciendo uso de una heurística sencilla, logra generar soluciones sub-óptimas, cercanas al óptimo real, en un tiempo de ejecución aceptable.
El algoritmo que se describe a continuación es el siguiente:

```
for()//8192 soluciones
for()//n iteraciones
for()//Calcular distancia de la solución
{
   //Matriz de distancias
   intercambiar elementos
   if(mejora)
   {
    distancia=nueva_distancia
   }
   else
   {
    restaurar elementos
   }
}
actualizar solución
```

La estructura de datos del problema es un vector unidimensional que presenta la siguiente forma:

$$[e0, e1,\ldots, eN-1, c0, eN+N+2,\ldots, eN+N+N+1, cN-1]$$

Donde *e* representa el elemento de la solución y *c* el coste asociado a dicha solución.

5 Paralelización del problema

La paralelización del problema se consigue haciendo que cada hebra aplique el algoritmo y genere una solución sub-optima. Por lo tanto como generamos 8192 soluciones se lanzarán 8192 hebras, cada una de ellas generando 1 solución sub-óptima.

Este número de hebras (8192) no es trivial. Se ha escogido este número puesto que si lanzásemos más hebras dependiendo del tamaño del problema tendríamos también que ocupar más memoria VRAM de la tarjeta gráfica y los tiempos de reserva, copiado y acceso a la memoria global de los resultados serían mayores. A su vez evitamos el control de divergencia al ser potencia de dos.

Se procede entonces paralelizando el primer bucle, puesto que si intentásemos una supuesta paralelización del segundo bucle el costo de sincronizar el resultado de las hebras en cada iteración mediante memoria compartida perjudicaría gravemente el rendimiento.

La organización propuesta es la siguiente, unidimensional para los bloques y unidimensional para las hebras. No obtendríamos ventaja alguna haciendo otro tipo de distribución puesto que esta distribución se adapta perfectamente a la naturaleza del problema.

Por lo tanto el esquema del mapeo para las 8192 hebras es el siguiente:

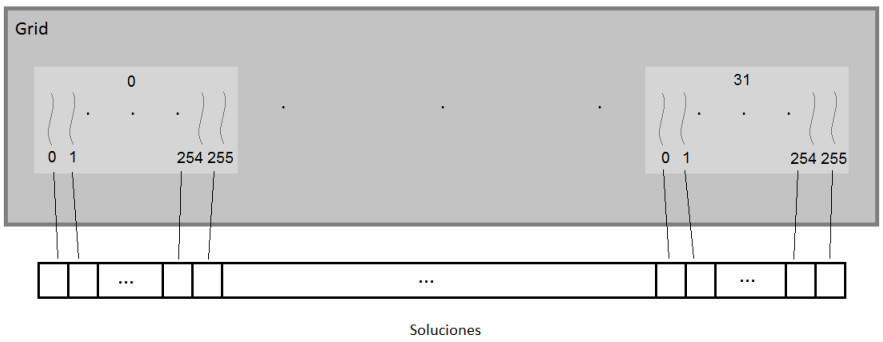


Fig. 5. Mapeo de trabajo por hebra.

Se lanzarán hebras organizadas en bloques, es decir, 32 bloques de 256 hebras cada uno dando un total de 8192 hebras.

Cada hebra escribirá su resultado en su parte correspondiente del vector de soluciones. Esto se realiza calculando el offset de cada hebra de la siguiente forma:

```
offset = ((blockIdx.x * blockDim.x) + threadId.x)*(dimension + 1)
```

Donde *dimensión* es el número de ciudades del problema a resolver. A esta variable se le suma una unidad puesto que cada solución de dimensión tiene su coste.

6 Soluciones propuestas

Una vez decidido como paralelizar, se han implementado tres versiones de un mismo algoritmo cuyas diferencias son:

- La primera versión utiliza memoria *shared* para guardar la matriz de distancias y *registros* para él vector solución. Esta versión del algoritmo ha resultado ser la más rápida.
- La segunda versión algoritmo utiliza *registros* para él vector solución.
- La tercera versión utiliza memoria *shared* para el vector de números aleatorios y *registros* para él vector solución. Esta versión del algoritmo ha resultado ser la más lenta. Se lanza n/45 veces siendo *n* el número de ciudades.

La utilización de la primera, la segunda o la tercera versión del algoritmo depende del tamaño del problema, puesto que hay problemas, que por su tamaño no se pueden abordar con la versión primera, puesto que la matriz de distancias ocupa demasiado y no es posible albergarla en la memoria tipo *shared* del dispositivo.

- Número de ciudades menor o igual a 45: Primer algoritmo
- Número de ciudades entre 45 y 50: Segundo algoritmo
- Número de ciudades mayor que 50: Tercer algoritmo

Como disponemos de un tamaño de 16KB para la memoria *shared* nos caben 16384 elementos de 8 bits. Con 45 ciudades llegamos a ese límite para almacenar la matriz de distancias puesto que 45x45x8 = 16200 elementos. Por este motivo el primer algoritmo es el más eficiente en tiempo de ejecución al tener la matriz de distancias en memoria *shared* y además el vector solución en *registros*. A partir de un tamaño mayor de 45 ciudades y menor de 50 ciudades ya no podemos introducir en memoria *shared* la matriz de distancias por lo que hemos utilizado el segundo algoritmo.

Cuando el número de ciudades es mayor a 50 utilizamos el tercer algoritmo lanzado *n*/45 veces siendo *n* el número de ciudades. En cada ejecución el tamaño máximo del vector de solución almacenado en registros es de 45. En la memoria *shared* se almacenan 4000 números aleatorios de 32 bits. La matriz de distancias permanece en memoria *global* por la imposibilidad de almacenarla en memoria *shared*. Este algoritmo produce soluciones peores puesto que las permutaciones se realizan sobre partes del vector solución y no sobre el vector solución completo.

El pseudocódigo de cada versión se expone a continuación:

```
for()//8192 soluciones
for()//n iteraciones
for()//Calcular distancia de la solución
{
     //Matriz de distancias en memoria shared
     //Vector solución en registros

     intercambiar elementos
     if(mejora)
     {
       distancia=nueva_distancia
     }
     else
     {
       restaurar elementos
     }
}
actualizar solución
```

Fig. 6. Primera versión del algoritmo, ejecutada cuando número de ciudades [1, 45].

```
for()//8192 soluciones
for()//n iteraciones
for()//Calcular distancia de la solución
{
    //Vector solución en registros

    intercambiar elementos
    if(mejora)
    {
        distancia=nueva_distancia
    }
    else
    {
        restaurar elementos
    }
}
actualizar solución
```

Fig. 7. Segunda versión del algoritmo, ejecutada cuando número de ciudades (45, 50].

```
for()//8192 soluciones
for()//n iteraciones
for()//Calcular distancia de la solución
{
   //Vector solución en registros
   //Números aleatorios en memoria shared

   intercambiar elementos
   if(mejora)
   {
       distancia=nueva_distancia
   }
   else
   {
       restaurar elementos
   }
}
sumar distancia de última ciudad de la solución parcial
  actual con primera ciudad del siguiente trozo
actualizar solución
```

Fig. 8. Tercera versión del algoritmo, ejecutada n/45 veces cuando número de ciudades (50, +∞].

7 Resultados paralelización I

Los tiempos obtenidos quedan reflejados en la Tabla 1, la primera columna indica el número de ciudades y el resto de columnas el tiempo empleado tanto en CPU como en CUDA con distinto número de iteraciones. Todas ejecuciones se realizan sobre 8192 hebras.

Table 1. Tiempo (seg.) distintas ejecuciones del algoritmo con distintos tamaños de entrada.

Ciudades	CPU 10000	CUDA 10000	CPU 100000	CUDA 100000
8	1,11	0,01	10,84	0,16
10	1,42	0,02	14,03	0,21
15	1,97	0,03	19,70	0,29
20	2,58	0,05	25,53	0,47
29	3,54	0,08	35,35	0,69
45	5,27	0,14	52,53	1,20
46	5,40	1,01	53,95	9,83
48	5,66	1,04	56,41	10,41
52	6,07	7,21	60,33	72,27
120	14,2	21,08	141,64	207,77

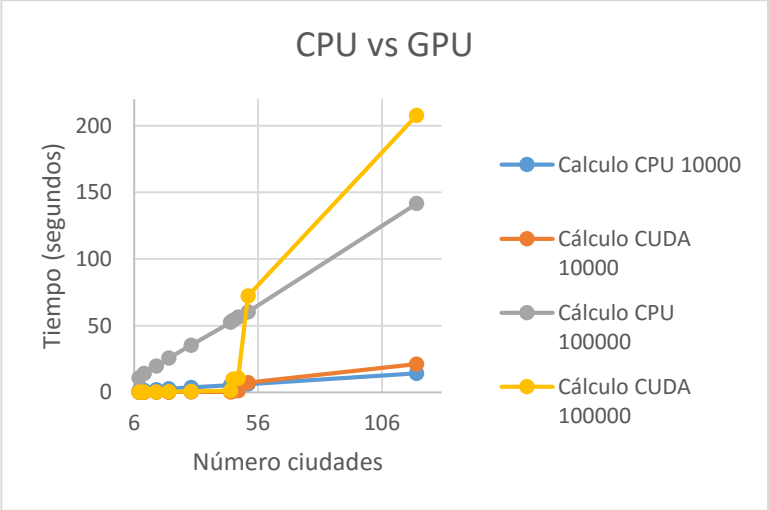


Fig. 9. Tiempo de cálculo de CPU frente a GPU. Como podemos apreciar cuando el número de ciudades es inferior a 45 el tiempo de cálculo en GPU es extremadamente pequeño obteniendo una ganancia significativa. A partir de 52 ciudades la CPU gana en tiempo de cálculo a la GPU.

En la Figura 9 podemos apreciar la diferencia entre los tiempos obtenidos con la CPU y con GPU. El problema se puede apreciar con claridad. Cuando el número de ciudades es inferior a 45 la diferencia de tiempos es bien clara no llegando al segundo en la GPU. Esto es debido a que tanto el vector local de soluciones de cada hebra cabe en sus *registros* y la matriz de distancias está en memoria *shared* que es aproximadamente 10 veces más rápida que la memoria *global*. Ahora bien cuando el número de ciudades es mayor que 50 tanto el vector soluciones, como el vector de números aleatorios, como el vector de distancias están en memoria global. Esto produce un cuello de botella considerable haciendo que prácticamente los accesos a memoria y la ejecución del programa sean secuenciales, eliminando así toda ganancia. En este punto la CPU supera en tiempo de cálculo a la GPU.

8 Resultados paralelización II

Los tiempos obtenidos quedan reflejados en la Tabla 2, la primera columna indica el número de ciudades y la segunda columna el tiempo. Todas ejecuciones se realizan sobre 8192 hebras.

Table 2. Tiempo (segundos) de las distintas ejecuciones del algoritmo con distintos tamaños de entrada.

Ciudades	CPU 10000	CUDA 10000	CPU 100000	CUDA 100000
52	9,18	1,73	94,57	16,45
120	20,8	3,82	219,21	37,74
150	26,61	4,74	280,38	47,07
202	35,17	6,50	368,77	64,11
561	99,89	40,02	1053,34	399,39

En la Figura 10 y Figura 11 podemos apreciar cómo se ha mejorado el tiempo de ejecución en la GPU cuando el número de ciudades es mayor a 50. Esto se ha conseguido lanzando varias veces el *kernel* en CUDA donde en cada ejecución se almacenan en registros 45 ciudades, y en memoria *shared* 4000 números aleatorios. Se consigue así minimizar el acceso a la memoria *global* y por lo tanto disminuir el tiempo de cálculo a costa de obtener soluciones menos óptimas puesto que las permutaciones se realizan en trozos de 45 elementos.

9 Repercusión del número de iteraciones

El número de iteraciones con el que lanzamos el programa determina la calidad de la solución obtenida para una entrada de ciudades concreta. A continuación se muestra una comparativa de la ejecución.

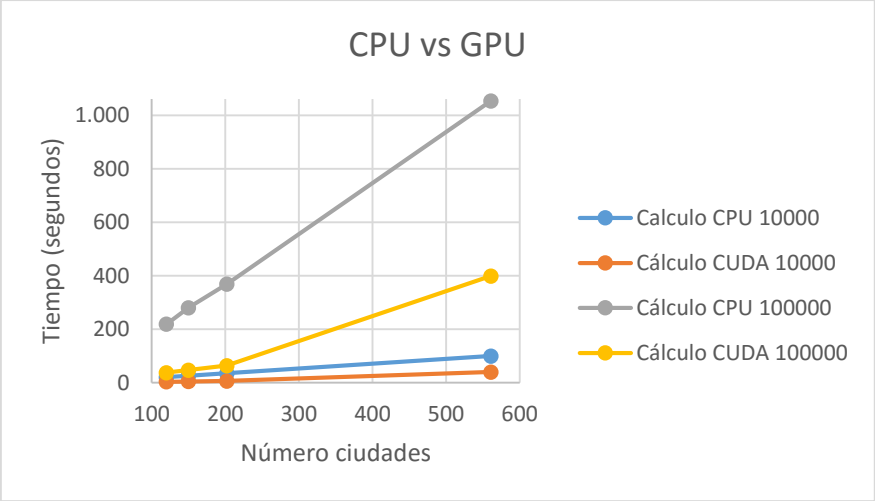


Fig. 10. Tiempo de cálculo de CPU frente a GPU. Como podemos apreciar ahora se obtienen ganancias significativas con más de 50 ciudades.

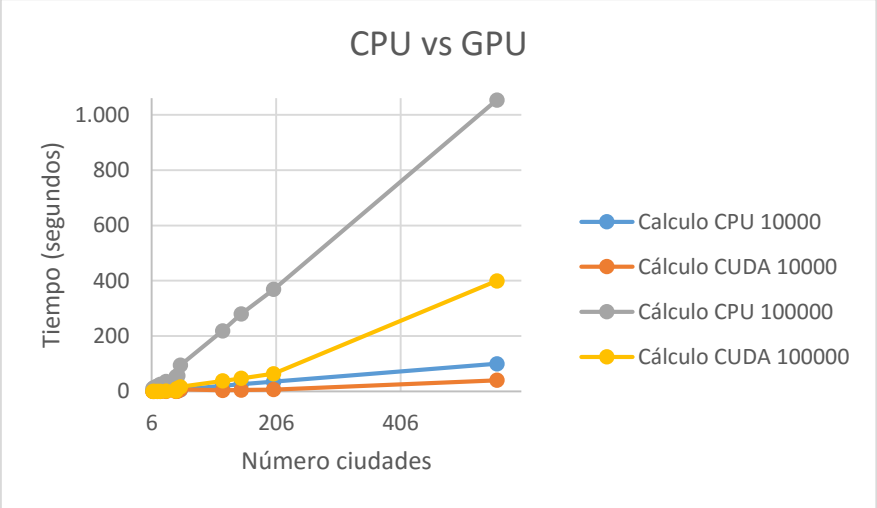


Fig. 11. Tiempo de cálculo de CPU frente a GPU. Unión de ambas gráficas anteriores.

Table 3. Distancia y tiempo de la ejecución del algoritmo para un mismo tamaño de entrada y distinto número de iteraciones.

Ciudades	Iteraciones	Tiempo (segundos)	Distancia
22	10	0,02	104
22	1000	0,19	72

Como se puede observar en la Tabla 3 el tiempo de ejecución es menor con 10 iteraciones. Por contrapartida se obtiene una mejor solución en la con 1000 iteraciones puesto que la distancia es menor (72 frente a 104).

10 Conclusiones

En esta práctica se ha comprobado como la GPU es capaz de obtener ganancias significativas en problemas que en apariencia no son aptos de paralelizarse como los clásicos algoritmos de procesamiento de imágenes que encajan perfectamente con la filosofía de programación CUDA. Algunos de los resultados presentados presentan ganancias de 58x cuando tenemos un número inferior a 45 ciudades.

Lo más destacable de esta práctica es la importancia de optimizar los algoritmos paralelos para que minimicen el acceso a la memoria global poniéndose de manifiesto en el apartado 7 y 8.

Finalmente esta práctica podrá ser reutilizada en la asignatura "Arquitectura y Computación de Altas Prestaciones" impartida en la especialidad de Ingeniería de Computadores para que los alumnos observen y deduzcan a partir de los distintos tamaños de entrada, cuando se empieza a utilizar la memoria global de forma más intensiva, observándose la degradación de la ganancia. También podrán examinar el código y proponer mejoras o sencillamente aprender de él.

Referencias

[1] NVIDIA Corporation. (2016, May) nvidia.com. *URL* http://www.nvidia.com/object/what-is-gpu-computing.html

[2] Delbosc. Nicolas. 2016. Overview of the CUDA device memory model. URL https://sites.google.com/site/computationvisualization/_/rsrc/1321741184041/programming/cuda/article1/memory.png?height=359&width=400

[3] StreamComputing BV. 2016. Compute Capability of Fermi and Kepler GPUs. URL http://streamcomputing.eu/wp-content/uploads/2012/10/Compute-Capability-of-Fermi-and-Kepler-GPUs.png

[4] Microway Inc. 2016. CUDA Grid Block Thread Structure. URL http://www.microway.com/wp-content/uploads/CUDA-GridBlockThread-Structure.png

[5] Wikipedia Inc. (2016, May) wikipedia.org. URL https://en.wikipedia.org/wiki/Travelling_salesman_problem

[6] Ellis Horowitz, Sartaj Sahni, "Fundamentals of Computer Algorithms", Rockville, Maryland, U.S.A., 1978.

[7] Wikipedia Inc. 2016. Solution of a travelling salesman problem. URL https://upload.wikimedia.org/wikipedia/commons/thumb/1/11/GLPK_solution_of_a_travelling_salesman_problem.svg/220px-GLPK_solution_of_a_travelling_salesman_problem.svg.png

Honeynet para el análisis del tráfico y muestras de malware

Santiago de Diego de Diego
Estudiante de doble grado de matemáticas e ingeniería informática.
Gustavo Romero López
Departamento de Arquitectura y Tecnología de los Computadores
Universidad de Granada
santidediego@gmail.com
gustavo@ugr.es

Resumen. En este proyecto nos hemos propuesto desplegar varios honeypots en dos dispositivos del tipo Raspberry Pi a fin de analizar ataques dirigidos a la red de la UGR. Presentamos a continuación un breve resumen del experimento. Por un lado hemos obtenido resultados de un honeypot de tipo Kippo, relacionados con ataques del tipo fuerza bruta, procedentes de varias direcciones IP, la mayoría de ellas de la zona de Asia. Además mostraremos los resultados del análisis de las muestras de malware obtenidas mediante Kippo. Por otro lado tenemos analizaremos los resultados obtenidos sobre ataques de tipo web recibidos por otro honeypot de baja interacción, Glastopf. No obstante, el principal objetivo del proyecto es identificar y clasificar diferentes muestras de malware así como proporcionar al lector una receta para este fin.

Palabras Clave: seguridad, honeypots, honeynet, malware, análisis del tráfico

Abstract. In this project we are about to deploy several honeypots in two Raspberry PI devices in order to analyze attacks directed to the UGR network. We present here a brief resume of the results of the experiment. On the one hand, we have results from a Kippo honeypot related to brute force attacks from several IP directions, most of them coming from Asia. In addition we show the results of a malware analysis of samples obtained from Kippo. On the other hand, we will analyse results related to web attacks with another low/medium interaction honeypot, Glastopf. In this particular project, the main purpose is to identify and classify several samples of malware as well as to show to the reader a recipe to achieve this goal.

Keywords: Security, honeypots, honeynet, malware, traffic analysis

1 Introducción

Antes del surgimiento de los honeypots, la seguridad era principalmente defensiva y se basaba en técnicas para parar a los atacantes. Existía mucha información sobre los elementos de un ataque tales como exploits, metodología del ataque… pero poca sobre los atacantes en sí. Por aquel entonces los administradores de sistemas no tenían ni tiempo ni los recursos necesarios para analizar todos los ataques.

Alrededor de 1999 la situación empezó a cambiar y se comenzó a estudiar también a los atacantes. Se desarrollaron los primeros dispositivos que se centraban la monitorización de sistemas, lo que dio lugar al *Honeypot Project* [1]. Algunos trabajos, [2] y [3], utilizan análisis de componentes principales para intentar caracterizar a los atacantes, mientras que otros, [4], se centran en elaborar una clasificación de atacantes según su comportamiento. Al principio los honeypots eran sistemas reales pero gracias a la virtualización, a lo largo de los años se fueron implementando soluciones más flexibles.

Actualmente, constituyen una solución ampliamente utilizada para el análisis de los diversos factores que componen un ataque. En [5] y [6] se centran en el análisis del malware, el primero de ellos proporciona un procedimiento genérico empleando análisis estático y dinámico para lidiar con esta problemática, del cual se han aplicado algunas nociones para realizar el presente trabajo. Otros como [7] se centran en explicar las técnicas más comunes empleadas para la ofuscación del malware, a fin de dificultar su desensamblado.

Según su interacción con el usuario podemos clasificarlos en honeypots de **baja, media** o **alta interacción.** Los más interesantes de cara a la obtención de información son los últimos, pero también son los más complicados de implementar. Además las posiciones donde podemos colocar un honeypot son:

- **Detrás del firewall:** en esta posición se encuentra protegido por las reglas de filtrado del firewall y por tanto deberemos configurar este para que no bloquee ataques del exterior. Tiene la ventaja de que permite detectar ataques internos, además de la posibilidad de comprometer la red interna.
- **Delante del firewall:** en esta posición se encuentra expuesto directamente a internet por lo que no es necesario configurar el firewall. Por contra, tiene la desventaja de que no permite detectar ataques internos.
- **En una zona desmilitarizada:** el honeypot se encuentra en una zona donde se encuentran los servidores pero separada de la red interna. De esta forma permite recibir ataques tanto internos como externos sin comprometer la red interna. Tiene la desventaja de que será necesario también configurar el firewall.

2 Escenario

Para dicha investigación se han desplegado dos honepots de baja/media interacción (Kippo y Glastopf) en una posición delante del firewall a fin de poder recibir ataques procedentes del exterior. Para este propósito se han empleado dos dispositivos del tipo Raspberry Pi 3, uno de ellos con sistema operativo Raspbian y el otro con sistema operativo Honeeepi, una distribución especial que viene con varios honeypots preinstalados, lo cual facilita enormemente la labor de configuración. Se ha prestado especial atención a las medidas de seguridad de la honeynet a fin de no comprometer la red de la UGR, ya que al ser dispositivos vulnerables por definición y además el objetivo es que sean atacados, este punto es crucial.

Para el análisis de las muestras de malware se han empleado máquinas virtuales de las mismas características que el objetivo de las muestras, las cuales veremos en la sección correspondiente. Posteriormente, se ha procedido a la eliminación de dichas máquinas virtuales.

3 Resultados generales

En este momento se han recibido **29342** ataques dirigidos al puerto 22, todos ellos con patrones muy similares (ver Fig. 1). La mayoría de ellos proceden de China, y en menor medida de Polonia, Vietnam y Holanda. Sin embargo, a pesar de la enorme tasa de ataques, resulta sorprendente la poca tasa de éxito de los mismos, ya que solamente el **12.62%** de los mismos ha resultado existoso. Este dato resulta aún más sorprendente si tenemos en cuenta que se han empleado credenciales de acceso realmente sencillas (admin-admin, root-root…).

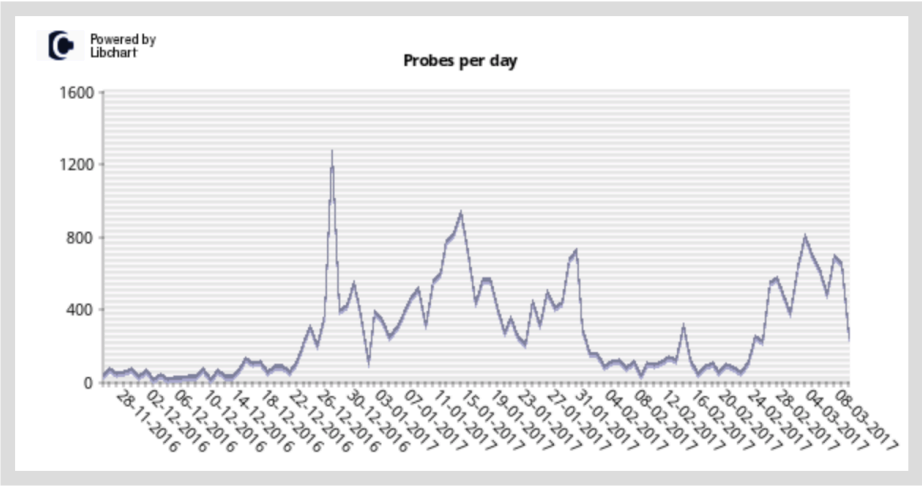


Figura 1. Pruebas por día

En la imagen puede verse un incremento repentino cerca del 22 de Diciembre, que seguramente haya sido debido a que en esa fecha es cuando se terminó de ocultar el honeypot como tal, de forma que ya simulaba ser un servidor real. Además, al principio el servidor no era muy conocido pero en unas pocas semanas pudimos comprobar como apareció en el buscador *Shodan* y por tanto es lógico esperar que el número de ataques recibidos se incremente enormemente desde entonces.

Por si fuera poco, se ha grabado las sesiones de los atacantes una vez entraban en el honeypot, a fin de poder descubrir patrones en su comportamiento. Por ejemplo, uno de los patrones observado es que todos ellos escribían a gran velocidad, así como que no cometían errores de escritura, por lo que es seguro que emplearon scripts automatizados para realizar los ataques. Podemos ver una de estas sesiones en la siguiente imagen:

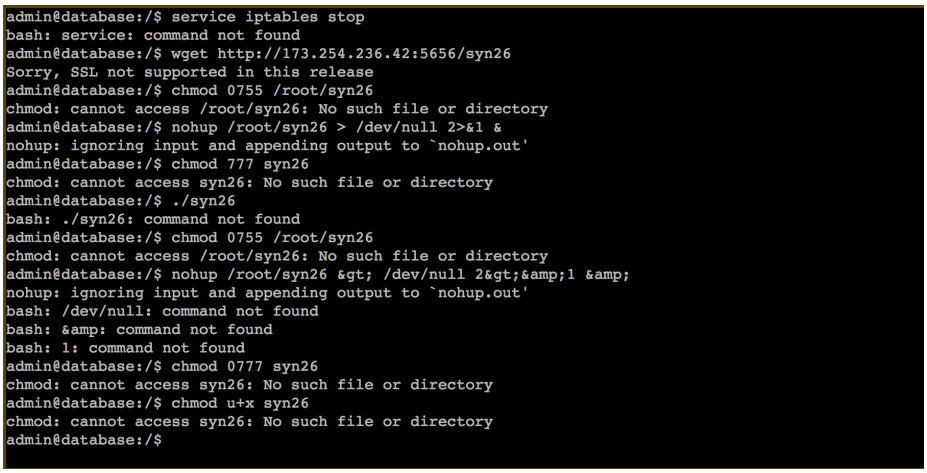


Figura 2. Sesión grabada de un atacante

Además hemos encontrado un patrón común a todos ellos:

1. El atacante accede al sistema por el puerto 22
2. Desactiva las reglas de filtrado del *firewall*
3. Descarga un fichero de una dirección IP remota
4. Ejecuta el fichero mediante el comando *nohup,* de forma que la ejecución continue cuando salga de la sesión

A raíz del análisis de estos *logs*, se ha descubierto además que los atacantes siempre realizan acciones similares sin tener en cuenta la situación, por ejemplo, es común ver como un atacante intenta ejecutar un fichero que no ha sido descargado correctamente, o incongruencias similares (ver imagen anterior) y por tanto tenemos otro indicio de que los ataques son realmente automatizados.

Además se ha procedido a clasificar a los atacantes por países, a fin de poder realizar estadísticas. Por ejemplo, podemos ver en la siguiente imagen una lista de los 10 países más beligerantes, de forma que se puede obtener una clasificación por países de los ataques recibidos.

ID	IP Address	Probes	City	Region	Country Name	Code	Latitude	Longitude	Hostname
1	116.229.239.244	1511	Shanghai	Shanghai Shi	China	CN	31.0456	121.3997	116.229.239.244
2	202.109.143.116	708	Nanchang	Jiangxi Sheng	China	CN	28.55	115.9333	202.109.143.116
3	109.236.91.85	423			Netherlands	NL	52.3667	4.9	customer.worldstream.nl
4	122.227.189.222	348	Ningbo	Zhejiang Sheng	China	CN	29.8782	121.5495	122.227.189.222
5	217.23.10.181	305			Netherlands	NL	52.3667	4.9	customer.worldstream.nl
6	93.190.143.155	229			Netherlands	NL	52.3667	4.9	customer.worldstream.nl
7	202.109.143.111	185	Nanchang	Jiangxi Sheng	China	CN	28.55	115.9333	202.109.143.111
8	123.31.34.215	183	Hanoi	Thanh Pho Ha Noi	Vietnam	VN	21.0333	105.85	localhost
9	121.18.238.99	179	Hebei	Hebei	China	CN	39.8897	115.275	121.18.238.99
10	183.91.14.188	133	Hanoi	Thanh Pho Ha Noi	Vietnam	VN	21.0333	105.85	static.cmcti.vn

Figura 3. Evaluación por países

Otra información útil puede ser ver qué clientes ssh han sido los más utilizados en los ataques. Podemos ver también una clasificación de los 10 clientes más utilizados en la figura 4:

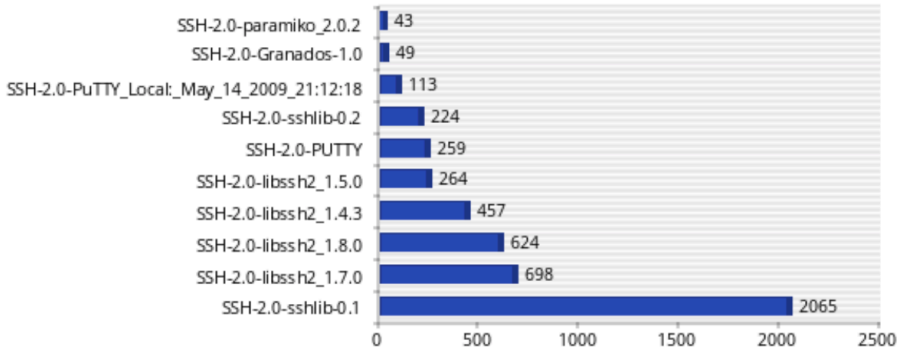


Figura 4. Top 10 de clientes ssh

Centrándonos ahora en Glastopf, hemos recibido numerosos ataques del tipo inyección SQL, así como búsqueda de los ficheros *robots.txt* o *sitemap.xml*, ambos de ellos conocidos por contener información valiosa sobre la estructura de un sitio web. Además hemos encontrado varias pruebas hacia la ruta */wp-login.php*.

En otro ataque muy común, el atacante trata de acceder a la carpeta */shell*, la cual no existe, lo que nos hace suponer que los ataques dirigidos al nodo Glastopf también son automatizados. El atacante escribe en dicha ubicación una cadena muy larga de caracteres, por ejemplo una como la siguiente (la más común):

/ s h e l l ? % 6 3 % 6 4 % 2 0 % 2 F % 7 4 % 6 D % 7 0 % 3 B %77%67%65%74%20%68%74%74%70%3A%2F%2F %36 %31 %2E %31 %36 %30 %2E%32%31%33 %2E %32 %38 %3A %35 %34 %33 %32 %31 %2 F %64 %6C %72 %2E %61 %72 %6D %3B %63 %68 %6D %6F%64%20%37%37 %37 %20 %2A %3 B %2E %2F %64 %6C %72 %2E %61 %72 %6D

La cual puede traducirse como:

cd /tmp && wget http ://61.160.213.28:54321/ dlr .arm; chmod 777 ∗;./ dlr.arm

Se han encontrado varias cadenas de este tipo y todas ellas presentan un patrón similar:

1. El atacante se mueve a la carpeta */tmp* ya que es el único lugar donde tiene permiso de escritura con el usuario *www*.
2. Descarga un binario malicioso de una dirección IP
3. Le asigna los permisos necesarios y lo ejecuta

Además hemos observado que el origen de los ataques es, a diferencia de en Kippo, bastante diverso. Podemos encontrar ataques de países muy diferentes, como podemos observar en la figura 5.

Se han empleado para el experimento **865** muestras, en lugar del enorme número de ataques registrados en Kippo. Esto es debido a que el ratio de ataques es mucho más bajo en el protocolo HTTP que en el SSH, debido a el tipo de explotación, ya que la segunda se presta más a su automatización por *script kiddies*[1] que la primera.

[1] Término despectivo aplicado a atacantes con poca experiencia

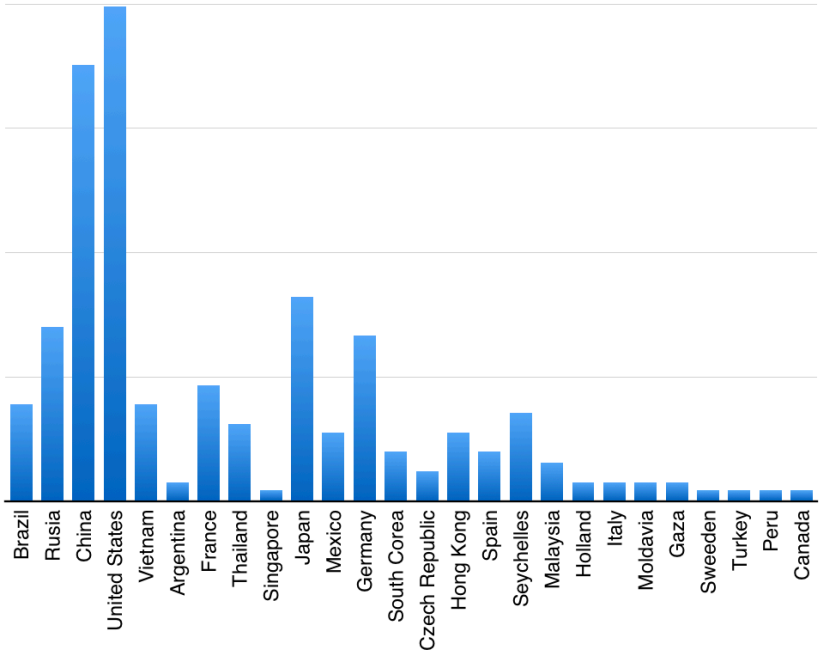


Figura 5. Pruebas por día en Glastopf

4 Análisis de las muestras de malware obtenidas

Para tal fin se han empleado herramientas tanto manuales como automatizadas (*objdump, gdb, radare, API de VirusTotal*…).

Las muestras recogidas son muy similares aunque con ligeras variaciones, como por ejemplo la IP a la que se conectan. En la siguiente tabla podemos ver un resumen con algunas de las muestras encontradas y analizadas, siendo el resto similares a estas. Todas ellas están escritas con objetivo sistemas Linux con arquitectura de 32 bits siendo el resto de características las que vemos en la tabla:

Table 1. Muestras de malware analizadas

Muestra	**Procesador**	**Tipo de ejecutable**	**Lenguaje**	**Información de depuración**	**Enlazado**
Z	Intel 80386	ELF	C++	Sí	Estático
Pomo	Intel 80387	ELF	C	No	Estático
Udp26	Intel 80388	ELF	C++	No	Estático
Syn26	Intel 80389	ELF	C++	Sí	Estático
Wrt	Intel 80390	ELF	C++	Sí	Estático
Xudp	Intel 80391	ELF	C	Sí	Estático
Mips	Intel 80392	ELF	C++	Sí	Estático

Hemos descubierto que las muestras pertenecen a un tipo muy particular de malware chino, muy común y potencialmente dañino. Las muestras tienen funcionalidad de *backdoor* y abren un puerto elegido aleatoriamente para conectarse a una IP remota. Además, se ha descubierto que son capaces de realizar ataques de denegación de servicio a otras máquinas una vez que han comprometido el host objetivo. Algo que nos ha llamado mucho la atención es que algunas muestras presentan información de depuración, por lo que nos hemos centrado en estas para poder obtener el máximo posible de información.

Sospechamos que dichas muestras pueden pertenecer a la botnet *BillGates,* calificada como de alto riesgo, a causa de las similitudes encontradas con otras muestras de malware de dicha botnet. Por ejemplo, un patrón en común es que todas ellas almacenan información en la librería */usr/libamplify.so* de sistemas Linux. Para realizar el análisis hemos empleado técnicas de ingeniería inversa, tales como el análisis estático y dinámico, todo ello en un entorno aislado y controlado. Si el lector quiere profundizar en estas tácticas de análisis, puede consultar [9], donde se proporcionan ejemplos y procedimientos para este fin.

Analizando la entropía de los binarios con *radare* hemos descubierto que no han sido encriptados, ya que dicha entropía es menor del **60%.** Las muestras son bastante indetectables, ya que todas ellas fueron subidas a VirusTotal utilizando un script en Python y solamente cerca del **15%** de los antivirus las detectaban como malware.

Hemos descubierto varias funciones y variables que nos han permitido afianzar nuestras sospechas sobre el comportamiento del malware. Podemos ver una captura de las funciones de una de las muestras, extraídas con ddd en la figura 6:

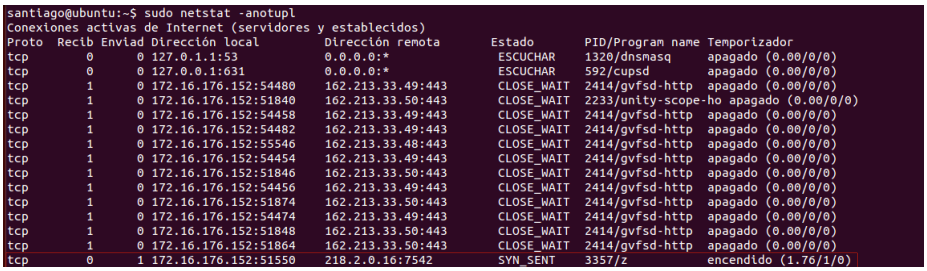
Figura 6. Funciones usadas por una de las muestras

Algunas herramientas de línea de comandos permiten recuperar este tipo de información por separado, como pueden ser *file, strings* o *strace* y pueden ser muy útiles para complementar la información obtenida examinando el código ensamblador. Un examen más exhaustivo del código, como mencionábamos anteriormente, revela

funcionalidad de DOS. Podemos encontrar varias cadenas relacionadas con este tipo de ataque como son:

- *AttackBase*
- *PacketAttack*
- *AttackUDP*
- *AttackSyn*
- *AttackICMP*
- *AttackDNS*
- *Tcpattack*

Un ejemplo de análisis dinámico muestra que las muestras de malware son completamente funcionales y funcionan como se espera de ellas. En la última línea de la imagen podemos ver la conexión con una ip desconocida:



```
santiago@ubuntu:~$ sudo netstat -anotupl
Conexiones activas de Internet (servidores y establecidos)
Proto  Recib Enviad Dirección local         Dirección remota       Estado       PID/Program name Temporizador
tcp        0      0 127.0.1.1:53            0.0.0.0:*              ESCUCHAR     1320/dnsmasq     apagado (0.00/0/0)
tcp        0      0 127.0.0.1:631           0.0.0.0:*              ESCUCHAR     592/cupsd        apagado (0.00/0/0)
tcp        1      0 172.16.176.152:54480    162.213.33.49:443      CLOSE_WAIT   2414/gvfsd-http  apagado (0.00/0/0)
tcp        1      0 172.16.176.152:51840    162.213.33.50:443      CLOSE_WAIT   2233/unity-scope-ho apagado (0.00/0/0)
tcp        1      0 172.16.176.152:54458    162.213.33.49:443      CLOSE_WAIT   2414/gvfsd-http  apagado (0.00/0/0)
tcp        1      0 172.16.176.152:54482    162.213.33.49:443      CLOSE_WAIT   2414/gvfsd-http  apagado (0.00/0/0)
tcp        1      0 172.16.176.152:55546    162.213.33.48:443      CLOSE_WAIT   2414/gvfsd-http  apagado (0.00/0/0)
tcp        1      0 172.16.176.152:54454    162.213.33.49:443      CLOSE_WAIT   2414/gvfsd-http  apagado (0.00/0/0)
tcp        1      0 172.16.176.152:51846    162.213.33.50:443      CLOSE_WAIT   2414/gvfsd-http  apagado (0.00/0/0)
tcp        1      0 172.16.176.152:54456    162.213.33.49:443      CLOSE_WAIT   2414/gvfsd-http  apagado (0.00/0/0)
tcp        1      0 172.16.176.152:51874    162.213.33.50:443      CLOSE_WAIT   2414/gvfsd-http  apagado (0.00/0/0)
tcp        1      0 172.16.176.152:54474    162.213.33.49:443      CLOSE_WAIT   2414/gvfsd-http  apagado (0.00/0/0)
tcp        1      0 172.16.176.152:51848    162.213.33.50:443      CLOSE_WAIT   2414/gvfsd-http  apagado (0.00/0/0)
tcp        1      0 172.16.176.152:51864    162.213.33.50:443      CLOSE_WAIT   2414/gvfsd-http  apagado (0.00/0/0)
tcp        0      1 172.16.176.152:51550    218.2.0.16:7542        SYN_SENT     3357/z           encendido (1.76/1/0)
```

Figura 7. Malware **z** conectándose a una IP remota

5 Conclusiones y trabajos futuros

La principal conclusión que podemos extraer es que es esencial asegurar nuestros sistemas, ya que mientras no nos damos cuenta estamos recibiendo numerosos ataques. Además los atacantes disponen de mucho más tiempo y mejores recursos económicos que nosotros por lo que no debemos subestimarlos. Por tanto es imprescindible el sentido común para evitar ataques de ingeniería social, además de una sólida política de seguridad ya que, como hemos visto, la mayoría de los ataques son automatizados y pueden ser evitados. Enfatizamos en el empleo de contraseñas seguras y en evitar el uso de configuraciones “por defecto” para prevenir este tipo de ataques.

Las futuras ampliaciones del trabajo, ordenadas por prioridad son:

1. Creación de una receta de despliegue automático
2. Creación de una interfaz web que simplifique la configuración y el uso de la honeynet

3. Creación de nuevos nodos, con diferentes honeypots, algunos de ellos de alta interacción, a fin de obtener más información
4. Empleo de técnicas de clasificación automática usando técnicas de inteligencia artificial

Referencias

[1] "The honeynet project,"http://www.honeynet.org/project,accessed:2017-05-24.

[2] S.Almotairi, A.Clark, G.Mohay, and J.Zimmermann,"Characterization of attackers' activities in honeypot traffic using principal component analysis," in *Network and Parallel Computing, 2008. NPC 2008. IFIP International Conference on*. IEEE, 2008, pp. 147–154.

[3] ——, "A technique for detecting new attacks in low-interaction honeypot traffic," in Internet Monitoring and Protection, 2009. ICIMP'09. Fourth International Conference on. IEEE, 2009, pp. 7–13.

[4] G. Salles-Loustau, R. Berthier, E. Collange, B. Sobesto, and M. Cukier, "Characterizing attackers and attacks: An empirical study," in *Dependable Computing (PRDC), 2011 IEEE 17th Pacific Rim International Symposium on*. IEEE, 2011, pp. 174–183.

[5] K. Kendall and C. McMillan, "Practical malware analysis," in *Black Hat Conference, USA*, 2007, p. 10.

[6] D. A. Quist and L. M. Liebrock, "Visualizing compiled executables for malware analysis," in *Visualization for Cyber Security, 2009. VizSec 2009. 6th International Workshop on*. IEEE, 2009, pp. 27–32.

[7] R. Harwood and M. Serrano, "Lecture 26: Obfuscation," 2013, car- negie Mellon University, https://www.cs.cmu.edu/~fp/courses/15411-f13/ lectures/26-obfuscation.pdf.

[8] "Shodan is the world's first search engine for internet-connected devices," https://www.shodan.io, accessed: 2017-05-25.

[9] H.project,KnowYourEnemy:LearningAboutSecurityThreats. Addison Wesley, 2004.

Instrucciones para Autores

Enseñanza y Aprendizaje de Ingeniería de Computadores (Teaching and Learning Computer Engineering) es una revista de Experiencias Docentes en Ingeniería de Computadores que edita el Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada, se publica anualmente, y se difunde tanto en papel como electrónicamente, a través del repositorio institucional de la Universidad de Granada (http://digibug.ugr.es/).

Los artículos remitidos para su evaluación pueden estar escritos en castellano o inglés, incluyendo un resumen y palabras clave en inglés en caso de que estén escritos en castellano, y deben seguir el formato descrito en la dirección web:

http://atc.ugr.es/pages/actividades_extension/

El correspondiente fichero .pdf debe enviarse a la dirección de correo electrónico jortega@ugr.es o mdamas@ugr.es

Los artículos deben abordar, tanto contenidos relacionados con la docencia universitaria en general, como con la docencia de asignaturas específicas impartidas por las áreas de conocimiento involucradas en estudios relacionados con la Ingeniería de Computadores, y también pueden aspectos relativos a las competencias profesionales y la incidencia de estos estudios en el tejido socio-económico de nuestro entorno.

En particular, se anima a antiguos alumnos de los estudios de Informática y a estudiantes de grado y posgrado a que envíen colaboraciones relacionadas con sus experiencias al cursar asignaturas relacionadas con la Ingeniería de Computadores, sugerencias, propuestas de mejora, etc.

Teaching and Learning Computer Engineering

Journal of Educational Experiences on Computer Engineering

July 2017, Number 7