

Práctica final: Videojuego con la FPGA

Diseño de Circuitos y Sistemas Electrónicos (DCSE)

Grado en Ingeniería Electrónica Industrial y Automática

Universidad Rey Juan Carlos

Curso Académico 2023 / 2024

Curso 4º - Cuatrimestre 1º

1. Descripción de la práctica

1.1. Introducción

En ese documento se presenta la práctica final de la asignatura de Diseño de Circuitos y Sistemas Digitales. Esta práctica engloba todos los contenidos aprendidos en la asignatura a lo largo del curso, teniendo un peso importante el manejo de memorias, máquinas de estados y contadores.

Para superar la práctica final, será necesario alcanzar los objetivos marcados en la parte básica de la práctica, pudiendo obtener una puntuación máxima de 7 puntos. Cabe destacar que se evalúa la calidad de la entrega y la entrega misma no implica la calificación máxima. Además, se proponen una serie de mejoras opcionales para subir nota, donde se indica la puntuación máxima en función de la calidad de la entrega. Se puede elegir cualquiera de ellas y no se encuentran ordenadas por dificultad. Por último, se abre la posibilidad a que el alumno proponga cualquier otro tipo de mejora, que será evaluada en función de su dificultad en función de los criterios de los profesores de la asignatura. La puntuación máxima de la práctica será de 10 puntos.

1.2. Especificaciones de funcionamiento

Realizar un diseño que muestre por un monitor o pantalla HDMI un fantasma (u otra imagen) de 16x16 rebotando por las paredes. Realizando el movimiento del fantasma se hace de cuadrícula en cuadrícula.

Al mismo tiempo, se mostrará otra imagen de un jugador que se moverá por la pantalla a través de los pulsadores. **El fantasma y el jugador deberán estar en una misma memoria.**

Por otro lado, el campo de juego es de 32x30 cuadrículas (32 columnas y 30 filas) de 16x16 píxeles. En el campo de juego se debe dibujar una imagen de 32x30 ampliada al tamaño de la celda.

Por último, el juego comenzará cuando se pulse el pulsador de inicio, mientras tanto, los Sprites estarán quietos y si se vuelve a pulsar se volverán a detener.

2. Parte básica

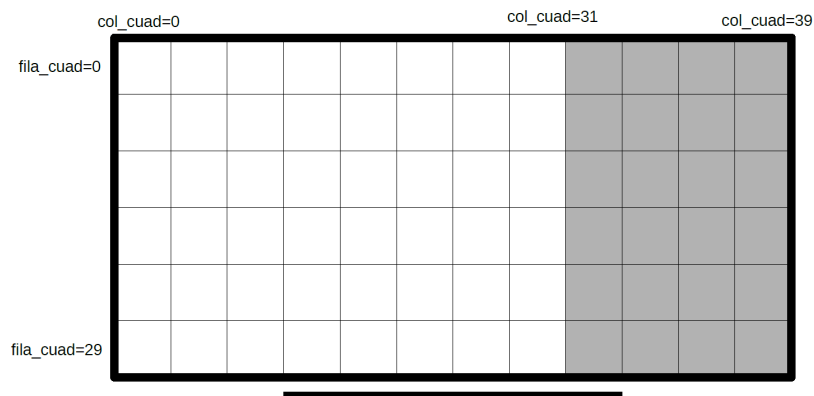
A lo largo de los siguientes puntos se van a describir cada uno de los pasos para abordar la parte básica de la práctica, con dificultad incremental. **Cabe destacar que para empezar esta parte es fundamental que funcione correctamente el módulo de sincronismo para el HDMI.**

2.1. Hito 1: Dibujar la cuadrícula de 16x16

Para dibujar la cuadrícula, se deberán separar de las filas y columnas de nuestro módulo de sincronismo. Por un lado, las filas y columnas interiores (4 bits menos significativos) y, por otro, las columnas y filas de la cuadrícula mediante los bits más significativos.

Se va a dibujar una rejilla, para ello, se debe indicar que el píxel será negro cuando la columna o fila interior sean '0'.

Por último, se va a hacer un campo de juego de 32 celdas (o menor que 512 píxeles). Por tanto, a partir de la celda 32 se pintará su contenido en color negro, o lo que es lo mismo, a partir de la columna 512 se pinta el píxel de negro.

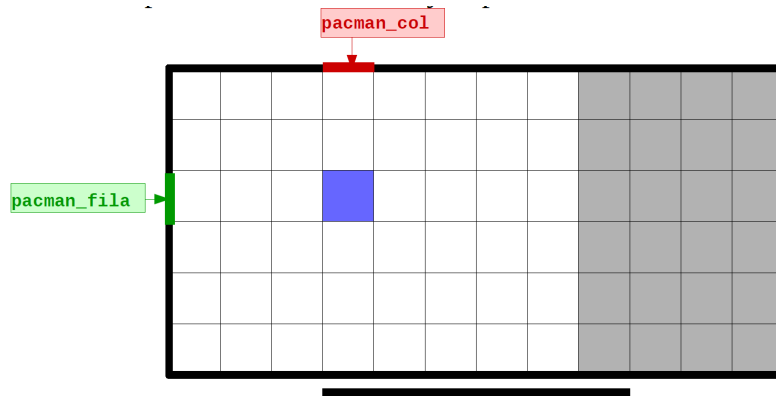


2.2. Hito 2: Dibujar PacMan

El objetivo de este hito es dibujar el PacMan leyendo desde memoria, sin embargo, se van a proponer una serie de hitos intermedios.

2.2.1. Dibujar la cuadrícula de color en la pantalla

En una posición de cuadrícula definida como una constante (pacman_fila, pacman_col), se va a pintar toda la cuadrícula de un color diferente, por ejemplo, azul. La posición es de la cuadrícula, por lo tanto, pacman_fila puede ser cualquier valor de 0 a 29, mientras que el valor de pacman_col puede variar entre 0 y 31.



2.2.2. Mover el cuadrado por la rejilla usando los pulsadores

Mediante los pulsadores, se propone que se mueva el cuadrado por la rejilla. Las posiciones de fila y columna del PacMan, o cuadrícula de color azul, pueden implementarse como dos contadores que van aumentando o disminuyendo en función de los pulsadores que se han presionado.

En lugar de usar los detectores de flanco en los pulsadores para determinar si se había pulsado o no, y evitar tener que pulsar tantas veces como cuadrículas queramos, se van a muestrear los pulsadores. Para ello, se comprueba cada 100 ms el estado de los pulsadores, si están presionados o no, si lo están, se mueve el personaje en la dirección que corresponda. Esto permite tener una experiencia de juego más fluida.

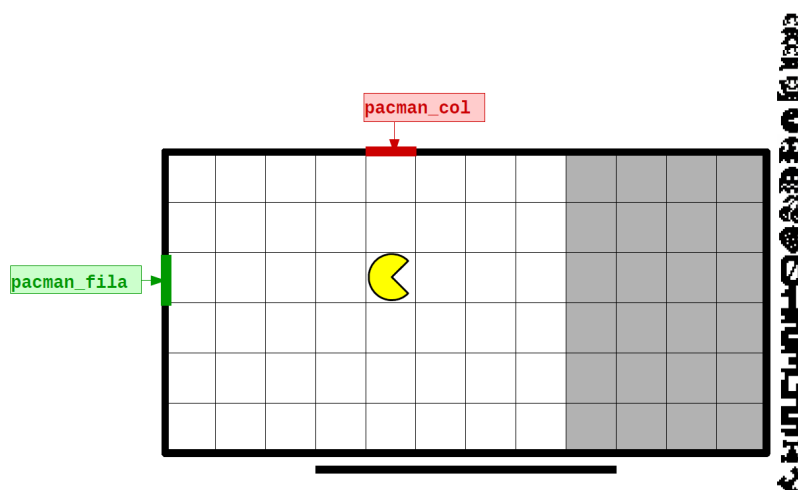
2.2.3. Pintar un personaje desde memoria

Utilizando la memoria de dos colores, se propone pintar uno de los personajes que aparece almacenado en memoria.

La memoria la puedes encontrar en la siguiente dirección:

https://github.com/felipe-m/img_rom/blob/master/examples/vhd/rom1b_1f_imagenes16_16x16_bn_t.vhd

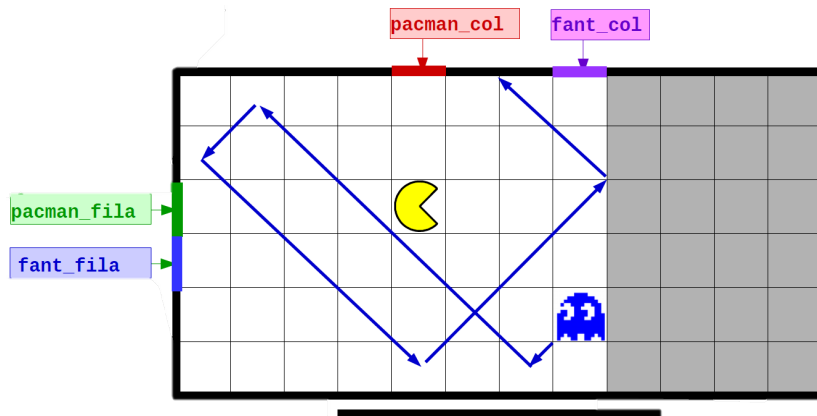
En dicha memoria se encuentran almacenadas 16 imágenes de 16x16 píxeles. Cada una de las filas de la memoria almacena una fila de 16 bits con la información de la imagen a pintar. El conjunto de 16 filas conforma una imagen de 16x16 píxeles, por lo que cada imagen se va a identificar con los bits más significativos de la dirección de memoria. Concretamente, el PacMan se corresponde con los bits más significativos “0011”, mientras que el fantasma “0100”.



2.3. Hito 3: Movimiento del fantasma

En lugar de mostrar el PacMan, ahora se va a usar la imagen del fantasma. Para ello, puedes repetir los pasos anteriores de los puntos 2.2.1 y 2.2.3 para dibujar el fantasma en otra cuadrícula.

Sin embargo, el paso explicado en el punto 2.2.2 es ligeramente diferente en este caso. El fantasma se va a mover en diagonal mediante una máquina de estados rebotando en las paredes del campo de juego. Deberá actualizar su posición cada 100 ms, por lo tanto, el fantasma cambiará de cuadrícula con ese periodo de refresco.



2.4. Hito 4: Dibujar la pista de carreras

Una vez que se controla el PacMan con los pulsadores y el fantasma se mueve por el área de juego, vamos a pintar el laberinto utilizando la memoria de 32x30 píxeles que almacena el mapa.

La imagen de la pista de carreras la puedes encontrar en el siguiente enlace:

https://github.com/felipe-m/img_rom/blob/master/examples/vhd/rom1b_1f_racetrack_1_t_32x30.vhd

Tenga en cuenta, que esta parte sólo se quiere dibujar el mapa, sin tener en cuenta las colisiones. Además, el laberinto se dibuja por cuadrícula, por lo que hay que ampliarlo por 16.



3. Mejoras opcionales para subir nota

3.1. Mejora 1 (0.5 puntos): Hacer que disminuya la velocidad cuando el jugador cruce el borde de la pista

Para tener una experiencia más realista, cuando el jugador se salga de la pista su velocidad será menor. Para ello, en lugar de avanzar una cuadrícula cada 100 ms, lo hará cada 500 ms, sin embargo, se debe conocer la posición del jugador para saber si se encuentra en la pista o fuera de ella.

Por tanto, tenemos que leer la memoria para comprobar si el jugador está en la pista y utilizar la lógica necesaria para realizar el cambio de la velocidad. No obstante, podemos simplificar la práctica, y en lugar de utilizar la memoria, podemos incluir una constante donde está almacenado el mapa de la pista y de esta forma no tenemos que usar memoria. La memoria en la FPGA es limitada y esto nos puede ayudar a resolver el problema de forma sencilla, así que puedes aprovechar esta ventaja de las FPGAs.

Para ello será necesario que descargar el paquete con la constante desde el Aula Virtual (pkg_racetrack.vhd), que contiene el mapa de la pista. Para ellos se puede utilizar la constante **pista**, que es una matriz y realizar la consulta a partir de las filas y columnas de la cuadrícula en la que se encuentre el jugador. A continuación, se muestra un ejemplo de uso, dónde las filas y columnas de la cuadrícula son el primer y segundo parámetro de la matriz, respectivamente.

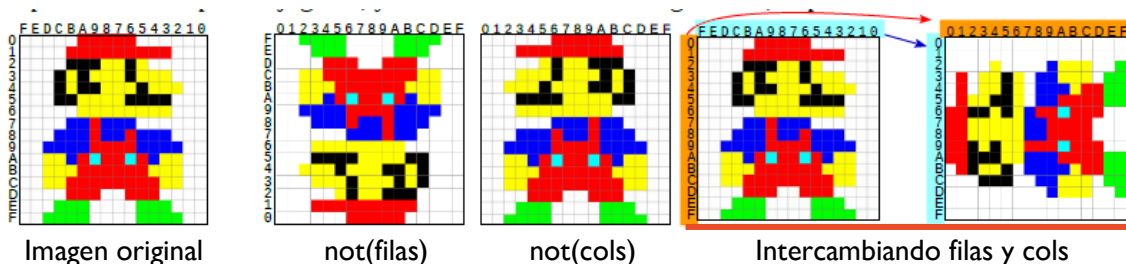
```
jug_en_pista <= pista(to_integer(posjug_fila))(to_integer(posjug_col));
```

En **jug_en_pista** se almacenará el valor obtenido de la constante, si es un '1' el jugador se encontrará en la pista, mientras que si es un '0' estará fuera de pista. El resto de lógica necesaria para reducir la velocidad se debe implementar. Recuerda que, al ser un paquete, se debe incluir y la constante **pista** ya está declarada en el paquete, por lo que no es necesario declararla.

```
use WORK.RACETRACK_PKG.ALL; -- incluye la pista
```

3.2. Mejora 2 (0.25 puntos): Cambio de orientación del Sprite del jugador en función del sentido de dirección

A la hora de cambiar la orientación del Sprite en función del sentido de dirección, es una mejora que es sencilla de implementar y se va a explicar con el siguiente caso de ejemplo. En memoria se tiene almacenado el Sprite de Mario, y se muestra como se vería en el monitor haciendo la lectura a partir de los valores de filas y columnas en la imagen de la izquierda.



Ahora bien, si invertimos los valores de las filas, la lectura comienza por el final, ya que, si **filas** tiene el valor "0000", al invertir el valor pasa a valer "1111", que corresponde con las filas de los

zapatos en la imagen original, de esta forma, se pone boca abajo a Mario. Sin embargo, si invertimos las columnas, tiene el efecto en Mario que mira hacia otro lado. Por otro lado, si se intercambian filas por columnas, lo que hacemos es girar 90° el Sprite. Con la combinación de todos se puede orientar el Sprite en cualquier dirección, basta que añadas la lógica correspondiente para que lo haga en función del sentido de la dirección que indica el usuario.

Esta mejora tiene 0.5 puntos extra si se utiliza el Sprite del coche para el jugador y se añade el sentido de dirección diagonal, si se permite pulsar de forma simultánea dos botones del control del jugador.

3.3. Mejora 3 (0.5 puntos): Pista de carreras y sprites en color

En esta mejora se pretende dibujar la pista de carreras y los sprites en color. Para ello, se va a hacer uso de más de una memoria, donde se tiene almacenado el color.

Para la pista de color, sólo vamos a utilizar dos memorias, una verde y otra azul, siendo el borde de la pista **rojo** "00" (RGB "100"), la línea de meta en color **azul** "01" (RGB "001"), la hierba en color **verde** "10" (RGB "010") y la pista en color **blanco** "11" (RGB "111").

Ten en cuenta que la pista sólo tiene dos memorias, pero son tres colores. Considera la lógica que puedes aplicar para cambiar el color rojo (RGB) en función los colores verde y azul.

Las memorias de la pista la puedes encontrar en los siguientes enlaces:

https://github.com/felipe-m/img_rom/blob/master/examples/vhd/rom1b_green_racetrack_1.vhd

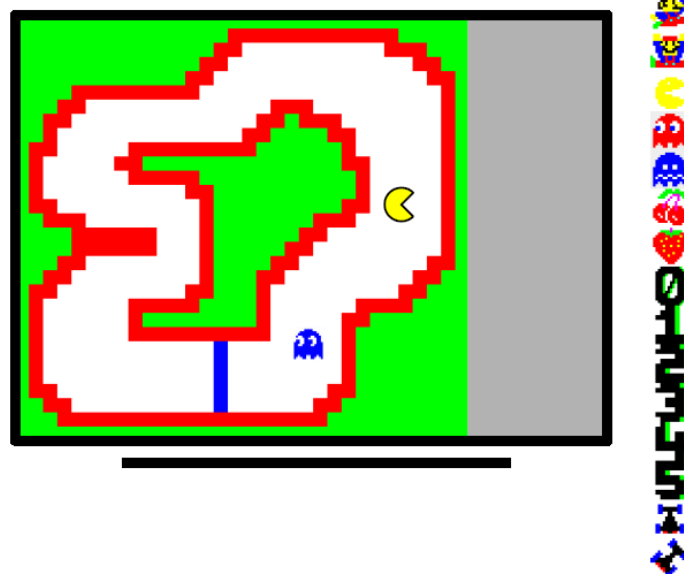
https://github.com/felipe-m/img_rom/blob/master/examples/vhd/rom1b_blue_racetrack_1.vhd

Las memorias de los sprites la puedes encontrar en los siguientes enlaces:

https://github.com/felipe-m/img_rom/blob/master/examples/vhd/rom1b_red_imagenes16_16x16.vhd

https://github.com/felipe-m/img_rom/blob/master/examples/vhd/rom1b_green_imagenes16_16x16.vhd

https://github.com/felipe-m/img_rom/blob/master/examples/vhd/rom1b_blue_imagenes16_16x16.vhd



















Si tienes pensado utilizar varias letras y símbolos, puedes utilizar la memoria de 32 sprites. Las memorias las puedes encontrar en los siguientes enlaces:

https://github.com/felipe-m/img_rom/blob/master/examples/vhd/rom1b_red_32num_play_sprite16x16.vhd

https://github.com/felipe-m/img_rom/blob/master/examples/vhd/rom1b_green_32num_play_sprite16x16.vhd

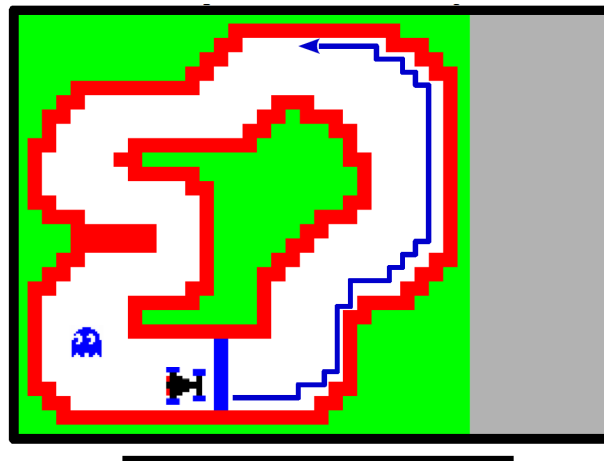
https://github.com/felipe-m/img_rom/blob/master/examples/vhd/rom1b_blue_32num_play_sprite16x16.vhd

0x00 a 0x07	0x08 a 0x0F	0x10 a 0x17	0x18 a 0x1F
	0		8
	1		9
	2		P
	3		L
	4		A
	5		Y
	6		S
	7		T

3.4. Mejora 4 (0.75 puntos): Hacer que el jugador automático recorra la pista bordeándola

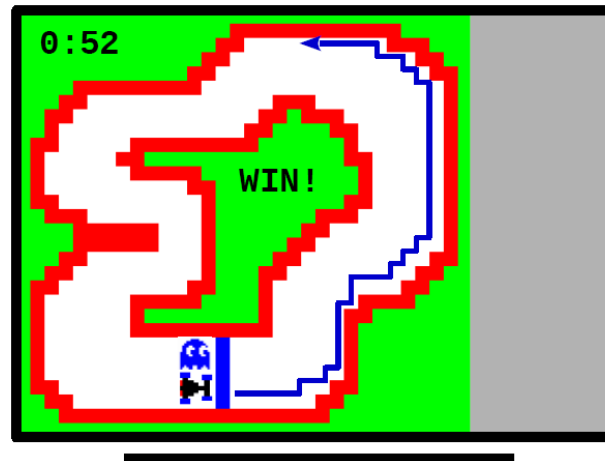
El jugador automático se moverá pegado al borde derecho de la pista. Para ello se hará uso de una máquina de estados en función de los elementos que se encuentren alrededor del jugador automático.

Se propone utilizar hasta cuatro niveles velocidades que se controlan con los switches, a modo de nivel de dificultad. Esto se implementa con contadores y definirá el cambio de cuadrícula del jugador automático.



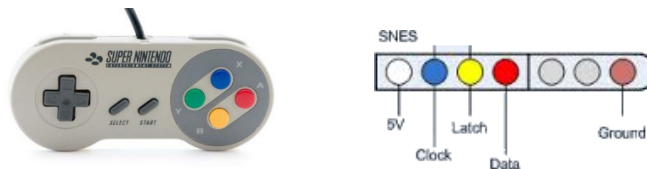
3.5. Mejora 5 (0.75 puntos): Mostrar un cronómetro e indicar quién da la vuelta rápida

Se propone introducir un cronómetro en la pantalla de juego, así como, un contador de vueltas. Se debe incluir también un Play/Stop, que puede usarse un switch para el inicio o pausa del cronómetro.



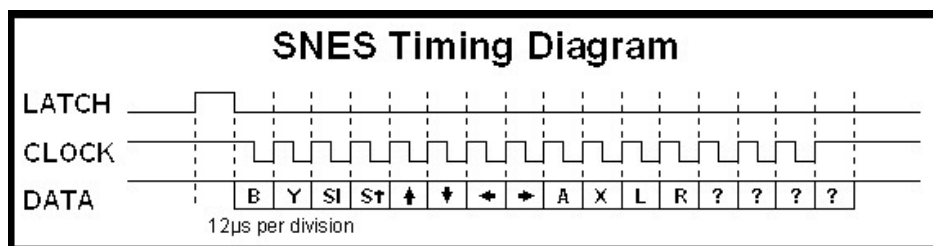
3.6. Mejora HMI 1 (1 punto): Mando de la SNES

Conectar el mando de la SNES a través del protocolo de comunicación implementado en la FPGA. El protocolo de comunicación es fácil de entender y se puede implementar de forma sencilla con una máquina de estados.



El conector tiene cinco pines útiles, dos dedicados a la alimentación y tres para el protocolo de comunicación. Aunque indica que se debe alimentar con 5V, a 3.3V funciona correctamente.

En cuanto al protocolo se debe generar desde la FPGA las señales de Latch y Clock según se indica en el siguiente diagrama. Automáticamente el mando envía la señal de pulsado o no de cada uno de los botones en función de su estado.



Más información: <https://www.fpgalover.com/index.php/boards/17-de0-nano-soc/43-snes-controller-module-de0-nano-soc>

3.7. HMI 2 (1.5 puntos): Nunchuk Wii

Se propone utilizar el nunchuk de la Wii para controlar el personaje. Este dispositivo cuenta con dos botones y un joystick, junto con un acelerómetro. Utiliza el protocolo de comunicación I2C y se propone utilizar el protocolo I2C con el ARM de la Zynq y utilizarlo para el movimiento del personaje.

Más información: <https://bootlin.com/labs/doc/nunchuk.pdf>



4. Entrega

Para la correcta entrega de la práctica final se deben realizar las siguientes entregas.

4.1. Entrega de los ficheros fuente

En la tarea correspondiente habilitada en el Aula Virtual se deberá entregar en un fichero comprimido **zip**:

- Los ficheros **VHDL** del diseño y de los bancos de pruebas realizados.
- Fichero de restricciones de restricciones **.xdc** (Xilinx Design Constraint).
- Fichero **.bit** para programar la FPGA.
- Memoria de la práctica final con una extensión máxima de 20 páginas.
 - Debe incluir capturas de pantalla de la simulación.
 - Tabla con el consumo de recursos.
 - Enlace al repositorio y vídeos realizados.

4.2. Entrega de la memoria para la corrección por pares

En el taller habilitado en el Aula Virtual se deberá entregar la memoria de la práctica final. Esta entrega incluye una revisión por pares, eso quiere decir que será evaluado por vuestros compañeros. La nota obtenida en la memoria no supone la nota global de la práctica, será un porcentaje dentro de la nota correspondiente al proyecto final.

Para evaluar, tendréis ciertas pautas, además, es una forma para los alumnos os podáis involucrar de forma más activa en la actividad y adquirir un mayor grado de comprensión de esta. Además, se pretende que desarrolléis vuestro espíritu crítico para vuestro futuro y lo podáis aplicar en vuestro TFG, en vuestro futuro profesional o académico, si decidís seguir estudiando un Máster.

¿Cómo se evalúa?

La calificación máxima de la actividad es de 10 puntos, de los cuales 8 puntos se corresponden al contenido de la memoria y 2 puntos a la evaluación que has llevado a cabo en la revisión por pares. Esta revisión por pares forma parte de la evaluación y si la haces mal, ya sea de forma injusta o muy favorable, y la nota que otorgas se desvía mucho del resto de compañeros, tu nota se verá penalizada. Sin embargo, si eres crítico con tu trabajo y el de tus compañeros, aplicando los criterios de forma correcta, y la nota que otorgas se desvía poco o nada con respecto a la del resto de tus compañeros, tu nota no se verá penalizada.

Por ejemplo, si mi trabajo no ha sido excelente y recibe una calificación de 4/8 puntos por el resto de mis compañeros, si realizo una correcta evaluación del resto de trabajos obtenido la nota máxima 2/2 puntos, obtendría una nota final será de 6 puntos. Sin embargo, si hago mal la evaluación es probable que no alcance la nota máxima en esa parte 0.5/2 puntos y tenga una nota final de 4.5 puntos.