Departamento de Ingeniería Electrónica
E.T.S.I. de Telecomunicación
Universidad Politécnica de Madrid

Enunciado del proyecto estándar de Sistemas Digitales II (SDG2)

ArkanoPi: una versión para la Raspberry Pi del popular videojuego arcade desarrollado por Atari

Fernando Fernández Martínez (coordinador docente)
fernando.fernandezm@upm.es
Manuel Gil Martín (coordinador administrativo)
manuel.gilmartin@upm.es
José Manuel Pardo Muñoz, Luis Fernando D'Haro Enríquez,
Alberto Boscá Mojena, Juan José Gómez Valverde,
Josué Pagán, Lucilio Cordero Grande,
Amadeo De Gracia, Cristina Luna Jiménez, Felicia Alfano

Curso 2020-2021

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Última revisión: 5, Febrero, 2021

Contenido

Índice de figuras

Índice de tablas

1. Aviso inicial

Cuando aborde la lectura de este documento, hágalo con tranquilidad y detenimiento. Frases o comentarios que no se entiendan en una primera lectura pueden encerrar avisos y recomendaciones que le serán útiles a lo largo del desarrollo.

No se preocupe si no alcanza a comprender todos los términos, conceptos y detalles que se discuten. Todos ellos se irán aclarando a medida que avance en la lectura de éste y otros documentos (i.e. tutoriales y demás recursos de apoyo, que permitirán profundizar en determinados aspectos del diseño). Por supuesto, asuma que **necesitará varias lecturas y una reflexión a fondo sobre todo esto**.

Finalmente, preste especial atención a todas las referencias explícitas a aspectos que se indican como obligatorios para incluir en el sistema y en la memoria correspondiente: no quiere decir que algo no referenciado explícitamente no tenga que ser tratado, sino que nuestra experiencia demuestra que algunos de esos aspectos no son considerados por un cierto número de alumnos, lo que da lugar a desagradables sorpresas en los exámenes.

2. Introducción

SDGII tiene por objetivos docentes los siguientes: en primer lugar, la aplicación y consolidación de los conocimientos sobre sistemas basados en microprocesadores o microcontroladores adquiridos en SDGI; y en segundo lugar, y como ampliación del primero, la adquisición y aplicación de conocimientos sobre programación de sistemas autónomos o empotrados (embedded) basados en un microprocesador (incluyendo hardware y software).

Con esos propósitos, la asignatura plantea el **desarrollo de un sistema electrónico complejo basado en un microcontrolador** partiendo de una descripción y unas especificaciones básicas **conforme a un caso real de diseño** que el alumno podrá consultar en este mismo documento.

El curso está organizado en sesiones de laboratorio orientadas a la implementación de un nuevo módulo o una nueva versión del sistema final a implementar. En particular, cada sesión planteará al alumno la consecución de un **hito** que corresponderá a un cierto nivel de desarrollo o madurez (funcionalidad) alcanzado por el prototipo.

Las primeras tres sesiones, en las que se presentarán los conceptos y las herramientas básicas necesarios para el desarrollo del proyecto propuesto, permitirán al alumno conseguir una **primera versión simplificada pero completamente funcional del sistema** (versión 1.0). Posteriormente, la consecución de cada nuevo hito, aplicando las herramientas y fundamentos adquiridos, significará la implementación de una nueva versión del prototipo (versión 2.0, 3.0, ...) al que se le irán añadiendo **nuevos elementos hardware** (pulsadores, displays, ...) **y software** (nuevos eventos y estados, temporización, ...) **que completarán y mejorarán su funcionalidad**.

Para ello, deberá seguir las instrucciones aquí incluidas, que implicarán diversas fases de diseño, análisis, implementación y medida de los circuitos y programas propuestos. Igualmente, se contará con todos los medios disponibles en el laboratorio B-043 así como también con la ayuda de los profesores y colaboradores docentes.

2.1. Sobre la evaluación del proyecto

El alumno deberá entregar electrónicamente, y haciendo uso de los medios habilitados a tal efecto en la plataforma Moodle de la asignatura, **una copia del código desarrollado correspondiente a cada versión haciendo notar la consecución o no del hito en cuestión.**

Salvo que se indique lo contrario, el proyecto propuesto contiene las **especificaciones mínimas obligatorias** que deben cumplir los sistemas y serán **valoradas con un máximo de 7 puntos**. **Esta nota máxima sólo se conseguirá si se cumplen todas las especificaciones y los resultados de las diferentes pruebas de evaluación continua son perfectos.** Adicionalmente a las especificaciones mínimas, se presentarán sugerencias de mejora opcionales, dejando a los alumnos la libertad para que añadan nuevas mejoras o esquemas alternativos. Con estas mejoras o montajes alternativos añadidos al prototipo básico, y dependiendo de su dificultad y realización, se podrán sumar puntos hasta alcanzar la máxima nota, 10 puntos. **Nótese que las mejoras añadidas sólo serán consideradas en caso de haber superado[1] la evaluación final sobre la versión definitiva del sistema (revisión de requisitos final y prueba individual en laboratorio, ambos aprobados).**

El trabajo realizado para la implementación de las mejoras consideradas debe quedar debidamente reflejado en una **memoria final** que contenga todos los detalles del proceso, incidiendo particularmente en los aspectos de **diseño HW y SW**, así como también en los **resultados** obtenidos y en todas aquellas cuestiones específicas que se indiquen en el presente enunciado o que sean notificadas a través de la plataforma Moodle de la asignatura. Tanto las instrucciones de entrega como la **plantilla** para la elaboración de esta memoria serán convenientemente publicadas y anunciadas a través de dicha plataforma.

2.2. Proyectos innovadores

Aquellos alumnos que deseen realizar un **proyecto innovador** basada en un **problema o un diseño propios** (o propuesto por un profesor), deberán hablar con alguno de los profesores de la asignatura y presentarle un listado de notas y una propuesta de proyecto donde describan, en 2 o 3 páginas:

- Objetivos del sistema propuesto.
- Recursos necesarios para llevarlo a cabo.
- Arquitecturas HW y SW propuestas para la resolución del problema.

Para poder abordar el proyecto será necesario contar con la aprobación de dicho profesor. No será admitido ningún proyecto (por muy complejo o perfecto que sea) que no se ajuste a estas normas. En el canal de YouTube[2] de la asignatura pueden visualizarse vídeos de demostración de los diferentes proyectos emprendidos por los alumnos de las últimas ediciones de SDG2.

[1] Para aspectos propios de la evaluación de la asignatura se remite al alumno a la guía docente: https://www.upm.es/comun_gauss/publico/guias/2020-21/2S/GA_09TT_95000033_2S_2020-21.pdf

[2] https://www.youtube.com/channel/UCYIw_gl745WMJ1n0MamDzQw/

2.3. Sobre la organización del documento

Los apartados siguientes del presente documento están organizados de la siguiente manera: en primer lugar, se facilitará una descripción detallada de la **especificación de requisitos** que debe cumplir el prototipo final a implementar. A continuación, se detallarán las **arquitecturas HW y SW** del mismo, haciendo especial énfasis en la descomposición modular del sistema, en la interacción HW-SW y en los requisitos de tiempo real, todos ellos aspectos clave para abordar con éxito el proyecto. Finalmente, se describirá la **planificación recomendada** para su desarrollo, **organizada por sesiones** de laboratorio para las cuales se indicarán tanto los **objetivos a conseguir** durante las mismas como los **recursos disponibles** para ello.

3. Descripción general

3.1. Antecedentes

Hoy en día los **microcontroladores** (μC) suponen más del 50% de los Circuitos Integrados existentes. Basta analizar un hogar cualquiera para encontrar al menos una o dos docenas de microcontroladores distribuidos entre los diferentes dispositivos presentes en el mismo: lavadoras, frigoríficos, hornos, microondas, teléfonos, mandos inalámbricos, teclados, automóviles, etc. Casi cualquier dispositivo electrónico cuenta con un μC como elemento esencial para la toma de decisiones o para la supervisión del sistema.

Uno de los sectores de mayor auge de la industria electrónica, tanto a nivel nacional como internacional, es el de los videojuegos. A modo de ejemplo, en España, los datos del informe de la AEVI, Asociación Española de Videojuegos, correspondiente al año 2015 [8] indicaban que los videojuegos facturaron durante ese año 1083 millones de euros en España (lo que suponía un crecimiento del 8.7% con respecto al año anterior), dato que consolidaba a la industria del videojuego como líder del ocio audiovisual en España (se estiman 15 millones de jugadores en todo el país), superando al cine, con 571 millones de euros en facturación. Este auge es también evidente en los datos de empleo que arroja el sector, donde las contrataciones crecen anualmente por encima del 20%. Estos datos subrayan la importancia de un sector cuyo crecimiento se estima garantizado a corto plazo (gracias, entre otras, a nuevas tendencias como la realidad virtual y los dispositivos inmersivos) y que constituye un auténtico motor de las industrias tecnológicas.

Para definir el proyecto estándar de este curso académico hemos realizado un cierto ejercicio de retrospectiva que nos ha llevado hasta casi el mismo comienzo de la historia de los videojuegos: el lanzamiento al mercado el 13 de mayo de 1976 de ***Breakout***, un videojuego arcade desarrollado por Atari (fundada en junio de 1972 por Nolan Bushnell y Ted Dabney). *Breakout* fue creado por el propio Bushnell junto con Steve Bristow, ambos influenciados por el videojuego de 1972 *Pong*, también de Atari y considerado por muchos el primer videojuego de la historia.

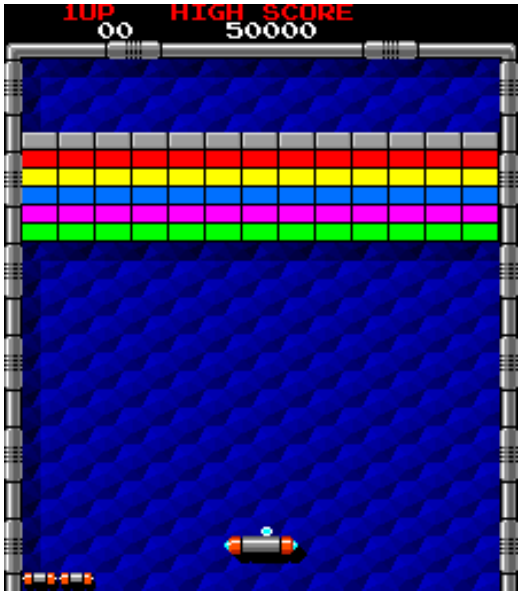


Figura 1. Captura de pantalla de Arkanoid, nueva y mejorada versión de Breakout lanzada por Taito en los 80.

La mecánica del *Breakout* era bien sencilla. Básicamente en la parte inferior de la pantalla una rayita simulaba una raqueta de front-tenis que el jugador podía desplazar de izquierda a derecha. En la parte superior se situaba una banda conformada por rectángulos que simulaban ser ladrillos. Una pelotita descendía de la nada y el jugador debía golpearla con la raqueta, entonces la pelota ascendía hasta pegar en el muro y los ladrillos tocados por la pelota desaparecían. La pelota volvía a descender y así sucesivamente. El objetivo del juego era terminar con la pared de ladrillos.

En el presente proyecto supondremos que un cliente (el Departamento de Ingeniería Electrónica) nos ha contratado para desarrollar un prototipo de videojuego de bajo coste. Este prototipo estará inspirado en el modelo o la mecánica de juego anteriormente detallada pero, a diferencia del de 1976 para el que los creadores simplemente emplearon dispositivos discretos y lógica discreta, contaremos con la ventaja de disponer de una Raspberry Pi, un microcontrolador de referencia y con altas prestaciones. Dado que el propósito de esta asignatura es eminentemente docente (con especial énfasis en la interacción HW-SW, así como en los requisitos de tiempo real), el objetivo será implementar un **prototipo funcional completo**. Este enunciado constituye una guía para su diseño e implementación. Adicionalmente, el alumno dispone de un **vídeo demostrativo** sobre este proyecto en YouTube: https://youtu.be/XII_JPf5SR8[4].

3.2. Objetivo general

El objetivo del proyecto es mostrar la viabilidad de la idea de diseño básica desarrollando un prototipo de videojuego plenamente funcional. El programa que ejecutará el micro se realizará en **lenguaje C**. El sistema digital basado en un microcontrolador será la plataforma ENT2004CF disponible en el laboratorio B-043 (construida en torno a una Raspberry Pi). Para más detalles relacionados con el sistema de desarrollo (utilización, pines de conexión, elementos disponibles), consulte la publicación [1].

[4] La demo mostrada corresponde a la versión del sistema desarrollada sobre las herramientas de virtualización y emulación desarrolladas para el trabajo desde casa. En la siguiente URL dispone igualmente de una versión del sistema completo con un joystick como elemento de control en lugar del teclado matricial: https://youtu.be/BhlKW1HB3Oc

En los apartados siguientes se detallarán las arquitecturas HW y SW, haciendo énfasis en la descomposición modular del sistema, tarea clave para abordar con éxito el diseño de cualquier sistema HW o SW medianamente complejo.

El objetivo general se resume en desarrollar un "sistema digital basado en la plataforma ENT2004CF" con las siguientes características:

- El juego se mostrará en una **matriz de leds de 7x8** , que actuará como display de visualización. En dicha matriz se deberá mostrar la **raqueta** (de dimensiones 1x3), **la pelota, y los ladrillos** (de 1x1 cada elemento). Para mostrar todos los elementos se llevará a cabo una adecuada excitación de los leds del display, empleándose 11 terminales digitales de salida de la plataforma ENT2004CF (7 terminales para las filas y 4 terminales para las columnas). De esta manera, el jugador observará cómo la pelota se desplaza por el display a una determinada velocidad constante.
- **La raqueta se posicionará en la fila inferior del área de juego**. Con objeto de poder eliminar todos los ladrillos, el jugador podrá modificar la posición de la raqueta horizontalmente, sin que esta rebase completamente los límites del área de juego en ningún momento y sin retardo en su movimiento apreciable. Para ello **el jugador contará con dos** de las 16 teclas de un **teclado matricial** (izquierda y derecha, respectivamente) cuya acción permitirá desplazar la raqueta en una posición (i.e. un led) hacia izquierda o derecha.
- **La raqueta no podrá detener la pelota pero sí que podrá modificar su trayectoria dentro del área de juego**. Para ello se aprovechará la detección del impacto de la pelota con la raqueta para definir una nueva trayectoria (i.e. nuevo ángulo de salida que adquiere la pelota tras su impacto con la raqueta) que dependerá de:
    - el ángulo de incidencia o entrada de la pelota: ángulo correspondiente a la trayectoria inicial o previa al choque de la pelota con la raqueta,
    - el lugar de impacto o posición de la raqueta en la que se produce el impacto con la pelota. Existirán tres posibles lugares de impacto, correspondientes a los tres "leds" de los que estará compuesta una raqueta.
- El juego comenzará con la pelota siempre en el centro del área de juego, pero aleatorizando para cada partida la trayectoria inicial asignada a la misma. **Durante el juego la pelota se desplazará a una velocidad constante** (e.g. la posición de la pelota se actualizará una vez cada segundo).
- **La detección de cada impacto de la pelota con alguno de los ladrillos será aprovechada para el borrado o la eliminación de dicho ladrillo.**
- Igualmente, **los límites superior, izquierdo y derecho** del área de juego constituirán superficies o **paredes contra las que la pelota deberá poder impactar**, saliendo esta rebotada con una nueva trayectoria en caso de que esto suceda. Esta nueva trayectoria dependerá, en este caso, única y exclusivamente de su ángulo de incidencia.
- Cuando el jugador no consiga golpear la pelota, rebasando esta la posición ocupada por su raqueta, y por tanto el límite inferior del área de juego, se **dará por finalizado el juego. Igualmente, el juego se dará también por concluido**

cuando el jugador consiga eliminar todos los ladrillos. En ambos casos se informará al usuario del número de ladrillos eliminado mediante un mensaje por la terminal y, tras la acción por parte del usuario de cualquiera de los pulsadores, se devolverá al sistema a su estado inicial a la espera de que dé comienzo una nueva partida.

- El sistema deberá mostrar diferentes **mensajes** en la pantalla del ordenador, a través de la **ventana de terminal** del entorno de desarrollo Eclipse, acerca del **estado** en el que el sistema se encuentra y de la ocurrencia de los diferentes **eventos de relevancia** para el funcionamiento del mismo (e.g. comienzo y final de juego).
- En la versión final, los retardos debidos a, por ejemplo, escribir por la terminal o gestionar las pulsaciones, no deben afectar a la correcta visualización de los elementos del juego. En particular, **no deben apreciarse parpadeos significativos en los leds del display** (requisitos de concurrencia y tiempo real).
- El sistema SW generará y procesará las señales necesarias para gobernar el HW externo.
- Los posibles rebotes mecánicos debidos al accionar una tecla se suprimirán por SW. **Si se realiza una única pulsación el sistema debe interpretar una única pulsación** sin que le afecten los posibles rebotes en la señal que se reciban de la tecla. La duración de una pulsación (sea breve o sea larga) no debe dar lugar a que se detecten varias pulsaciones de la misma tecla. El sistema debe detectar pulsaciones lo suficientemente breves como para desplazar la raqueta de la manera deseada.

Este enunciado constituye una **guía** para el diseño e implementación de nuestro prototipo funcional completo conforme al funcionamiento anteriormente detallado. Para ello, contaremos además con la ventaja de disponer de una Raspberry Pi, un microcontrolador de referencia y con altas prestaciones, así como también de otros elementos HW de relevancia para el diseño como es el caso de la matriz de leds, empleada como display de visualización, y el teclado matricial empleado como dispositivo o elemento de control, ambos disponibles en cada uno de los puestos del laboratorio (ver Figura 2 y Figura 3).

Figura 2. Teclado disponible en placa TL04.

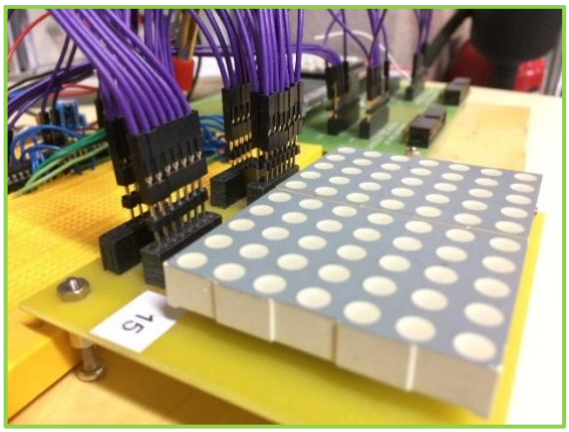


Figura 3. Display de leds 7x10 en placa accesoria.

En los apartados siguientes se detallarán las arquitecturas HW y SW, haciendo énfasis en la descomposición modular del sistema, tarea clave para abordar con éxito el diseño de cualquier sistema HW o SW medianamente complejo.

3.3. Especificaciones mínimas OBLIGATORIAS

Todas las especificaciones de funcionamiento recogidas en el apartado anterior tendrán el carácter de **obligatorias**. El cumplimiento de las **especificaciones obligatorias** de funcionamiento permitirá obtener **hasta un máximo de 7 puntos sobre 10 en la nota final de la asignatura**. Adicionalmente, y con **carácter opcional**, los alumnos podrán **mejorar** su diseño, y en consecuencia su calificación, con cualesquiera otras modificaciones complementarias a las presentes especificaciones hasta alcanzar la máxima nota.

Figura 4. Diagrama de bloques del sistema.

4. Subsistema Hardware

En la Figura 4 se muestra el diagrama de los bloques o módulos principales que componen el sistema. En dicho diagrama se han destacado las relaciones existentes entre los diferentes módulos tanto desde el punto de vista de diseño SW, destacadas mediante flechas en color azul, como desde el punto de vista de diseño HW, destacadas mediante flechas en color naranja. En los siguientes apartados se irán describiendo las partes más importantes que forman el subsistema HW.

4.1. Matriz de leds

Conceptualmente necesitamos un display o subsistema HW de visualización constituido por **una matriz de 7 filas y 8 columnas de leds**, subsistema que podremos implementar siguiendo un esquema similar al presentado en la Figura 5 (nótese que el esquema propuesto es para un display de dimensiones 8x8).

Siguiendo dicho esquema, podemos comprobar cómo para encender el led ubicado en una fila y columna determinadas basta con poner a nivel bajo la fila (i.e. 0V) y a nivel alto la columna (i.e. 5V) correspondientes. Igualmente, excitada una cierta columna, para mantener apagado alguno de sus leds bastará con poner a nivel alto la fila correspondiente al mismo.

La ubicación de los ladrillos, la pelota y la raqueta dentro del área de juego permitirá determinar qué leds de la matriz deben permanecer encendidos o apagados en cada momento, consiguiéndose así la visualización de los elementos del juego en el display.

Como los leds estarán organizados en forma de matriz 7x8, se usarán 7 terminales del puerto de salida para iluminar o no cada uno de los 7 leds (o filas) que hay en cada columna, y otros 3 para seleccionar, por medio de un decodificador, qué columna iluminar en cada momento.

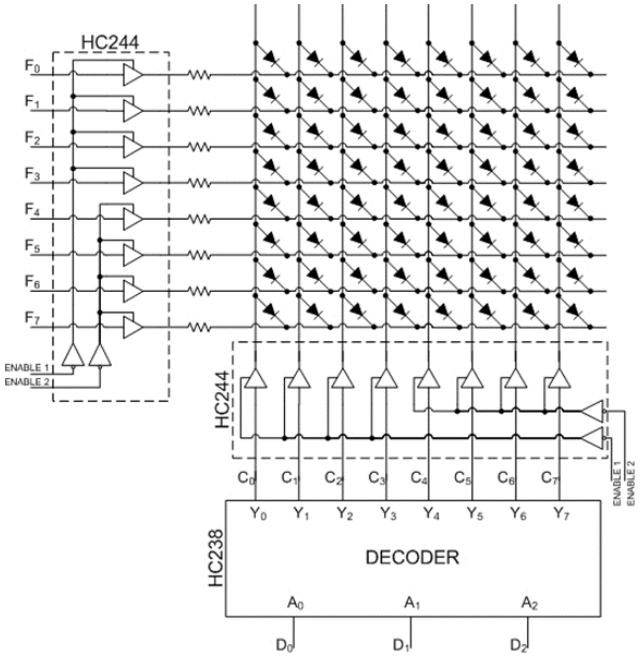


Figura 5. Detalle del subsistema HW de visualización propuesto (ejemplo para 8 filas y 8 columnas).

Puerto de salida	13	12	11	10	9	8	7	6	5	4	3	2	1
Conexión	-	-	ADC/DAC SPI_Dout	ADC SSTRB	-	GPIO-10 DISPLAY FILA4	GPIO-08 DISPLAY FILA3	GPIO-07 DISPLAY FILA2	GPIO-04 DISPLAY FILA1	GPIO-03 TECLADO COL4	GPIO-02 TECLADO COL3	GPIO-01 TECLADO COL2	GPIO-00 TECLADO COL1
Puerto de salida	26	25	24	23	22	21	20	19	18	17	16	15	14
Conexión	-	GND	GND	-	-	GPIO-25 LIBRE	GPIO-24 DISPLAY FILA7	GPIO-23 DISPLAY FILA6	GPIO-22 DISPLAY FILA5	GPIO-18 LIBRE	GPIO-17 DISPLAY COL3	GPIO-14 DISPLAY COL2	GPIO-11 DISPLAY COL1

Tabla 1. Propuesta de uso de pines GPIO de salida.

Puerto de entrada	13	12	11	10	9	8	7	6	5	4	3	2	1
Conexión	SPI_CS_DAC	SPI_CS_ADC	ADC/DAC SPI_Din	ADC/DAC SPI_CLK	-	GPIO-19 LIBRE	GPIO-16 LIBRE	GPIO-15 LIBRE	GPIO-09 LIBRE	GPIO-13 TECLADO FILA4	GPIO-12 TECLADO FILA3	GPIO-06 TECLADO FILA2	GPIO-05 TECLADO FILA1
Puerto de entrada	26	2	24	23	22	21	20	19	18	17	16	15	14
Conexión	-	GND	GND	-	-	-	-	-	-	GPIO-27 LIBRE	GPIO-26 LIBRE	GPIO-21 LIBRE	GPIO-20 LIBRE

Tabla 2. Propuesta de uso de pines GPIO de entrada.

En la Tabla 1 se puede observar el detalle de la propuesta de pines del puerto de salida a emplear para las filas en naranja y para las columnas en amarillo respectivamente.

Para que el refresco columna a columna del display se produzca sin parpadeo apreciable, el subsistema SW:

- iluminará los leds apropiados de la primera columna durante 4 ms,
- posteriormente, iluminará sucesivamente las siguientes columnas durante el mismo periodo de tiempo,
- tras alcanzar e iluminar la última columna, volverá a la 1ª cíclicamente.

De esta manera, cada led se puede encender y apagar aproximadamente unas 30 veces por segundo (i.e. 1 excitación cada 32 ms, 31.25 veces/seg), produciéndose la ilusión de que los leds de varias columnas están encendidos a la vez, cuando realmente en cada instante de tiempo sólo están siendo excitados los leds de una columna en particular.

Nótese que las conexiones al puerto de salidas digitales del entrenador, tanto las de las columnas como las de las filas, requieren un buffer 74HC244. Estos buffers tienen por objetivo que no se exija mucha corriente a los puertos del entrenador, sino a los propios 74HC244. Las conexiones entre el buffer y las filas requieren además de un conjunto de resistencias de un valor adecuado (será responsabilidad del alumno determinarlo) que permita que los leds no consuman mucha corriente pero al mismo tiempo se iluminen bien. En el anexo 8.2 dispone de información detallada con el pinout de la matriz de leds que le permitirá realizar las conexiones requeridas.

4.2.Teclado matricial

Para la implementación del subsistema HW de control de la dirección de la raqueta se propone emplear el **teclado matricial de la placa TL04** disponible en cada puesto de laboratorio (ver Figura 6).

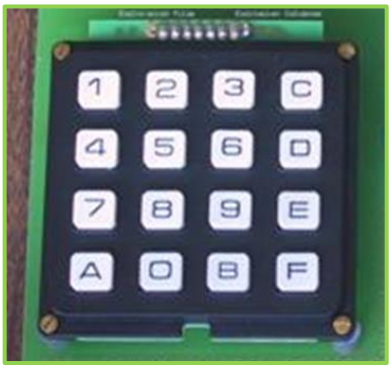


Figura 6. Detalle del teclado matricial disponible en la placa TL04.

Se trata de un teclado matricial de 16 teclas cuyo funcionamiento, representado en la Figura 7, se basa en la excitación secuencial de las columnas y la consiguiente exploración secuencial de las filas. Para ello el teclado cuenta específicamente con conexiones a los siguientes pines GPIO de la Raspberry Pi: 4 salidas para excitar las columnas y 4 entradas para explorar las filas, cuyos detalles aparecen recogidos en la Tabla 2 y en la Tabla 1 anteriormente presentadas.

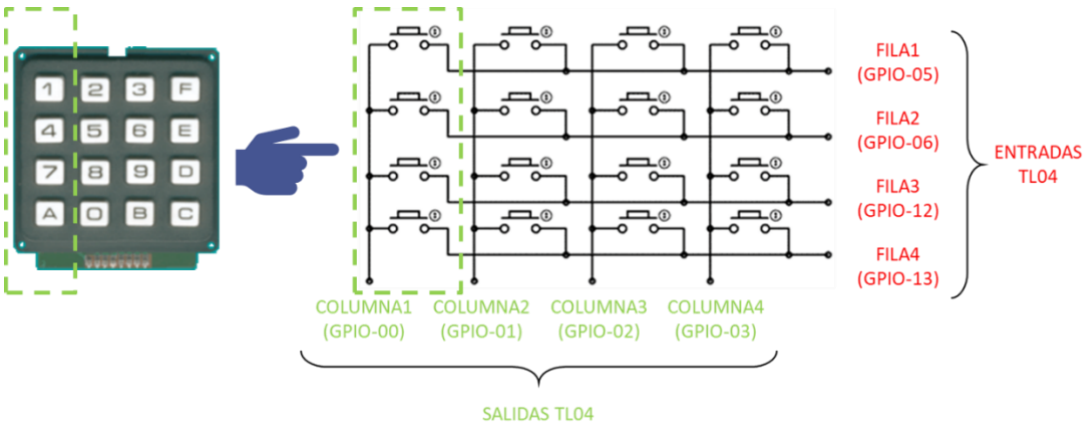


Figura 7. Detalle del funcionamiento del teclado matricial.

En cada momento sólo una de las columnas permanece excitada lo que permite la detección de cualquier pulsación que pueda afectar a cualquiera de las teclas dispuestas en dicha columna (en la figura la activación o excitación de la primera columna sólo permitiría la detección de pulsaciones en las teclas 1, 4, 7 ó A).

Para permitir la detección de pulsaciones en el resto de teclas, la excitación debe actualizarse periódicamente trasladándose de forma cíclica al resto de columnas. En ese sentido, el tiempo transcurrido entre diferentes excitaciones de una misma columna deberá ser lo suficientemente breve como para evitar la no detección de una pulsación rápida.

Cada tecla hace de pulsador de manera que el esquemático equivalente para cualquiera de las filas de entrada es el representado en la Figura 8, en la cual se pone de manifiesto que no es necesario incorporar la habitual resistencia de pull-down (o pull-up) propia de estos montajes, al contar ya la entrenadora con dichas resistencias (i.e. de valor 10K para cada una de las entradas). Recuerde que estas resistencias permiten establecer un determinado estado lógico (i.e. un 0 en nuestro caso) en la entrada de nuestro circuito cuando el pulsador se encuentra en estado de reposo.

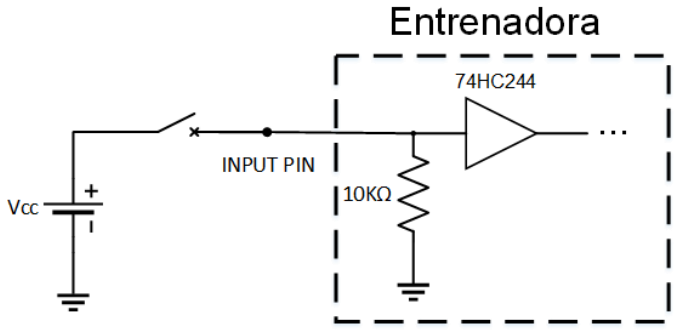


Figura 8. Esquemático equivalente para cada tecla.

4.3. Observaciones adicionales sobre el HW

Nótese que, en el laboratorio B-043, las conexiones entre el HW externo y la Raspberry Pi **NO pueden ser establecidas de forma directa,** ya que la Raspberry está integrada dentro de la plataforma ENT2004CF y sólo es posible acceder a sus pines GPIO indirectamente, por medio de unos conectores ubicados en la placa externa llamada TL04 convenientemente conectada a la ENT2004CF. Las entradas y salidas de la TL04 están oportunamente conectadas a los pines GPIO de la propia Raspi a través de optoacopladores, situados en el interior de la plataforma e incorporados para la protección de dichos pines (dispone de más información al respecto en [3]). Aunque necesarias, estas protecciones funcionan unidireccionalmente, lo que se traduce en que **unos pines GPIO sólo pueden utilizarse como entradas y otros sólo como salidas**. En el anexo 8.1 se incluye una tabla con la descripción completa del conjunto de entradas y salidas finalmente disponibles en el laboratorio.

De igual modo, y también en relación con dichas protecciones, es necesario recordar que los puertos de entrada disponibles en la plataforma ENT2004CF están formados por buffers digitales **con resistencias de pull-down de 10 kΩ**[5]. Si no se tiene en cuenta este hecho, se podrían producir efectos de carga no deseados al conectar el HW a la plataforma.

[5] Los puertos de salida están disponibles directamente a través de buffers digitales.

Igualmente importante, desde el punto de vista de la alimentación de los diferentes subsistemas, es el hecho de desacoplar las alimentaciones y alimentar en estrella.

Conviene también recordar que **el diseño y la compra** de componentes, **así como la implementación de los distintos montajes** circuitales, **se debe realizar con anterioridad a las sesiones de laboratorio**. Durante dichas sesiones, el objetivo fundamental del alumno debería ser la verificación y ajuste de los montajes requeridos. En ese sentido, resulta esencial la visualización de las diferentes señales implicadas en el osciloscopio.

5. Subsistema Software

El software estará basado en una máquina de estados que gestionará la entrada/salida, las interrupciones y la temporización.

5.1. Esquema de procesos

Frecuentemente, los sistemas electrónicos digitales precisan realizar diversas acciones de una manera paralela (concurrente en varios procesadores) o aparentemente paralela (todas las acciones se ejecutan entrelazadas, produciéndose la apariencia de paralelismo si la frecuencia de conmutación entre ambas es elevada). Son los denominados **procesos**. La creación, ejecución, sincronización y destrucción entre procesos suele ser facilitada por el sistema operativo a través de diversas primitivas. Pero incluso en ese caso, el diseñador debe definir qué tareas o rutinas serán concurrentes, elegir los mecanismos de comunicación entre ellas, los tiempos de vida, etc.

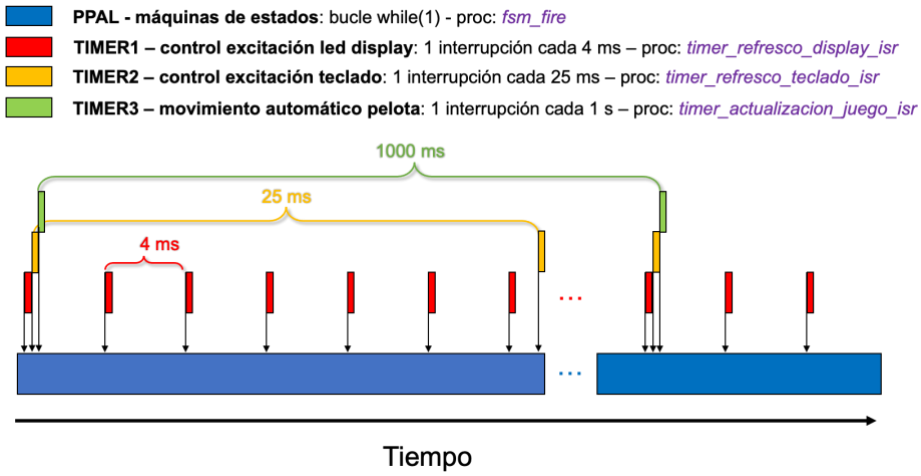


Figura 9. Detalle del esquema de procesos propuesto para el proyecto.

En la Figura 9 se muestra a modo de ejemplo uno de los posibles esquemas de procesos que podrían emplearse para resolver el presente proyecto. Dicha figura pone de manifiesto la existencia de 4 procesos fundamentales:

- **el programa principal (PPAL)**: siempre existe y debe inicializar los objetos del sistema, inicializar el HW, e inicializar y habilitar tanto los procesos de interrupción periódica como los vinculados a las interrupciones externas. Será igualmente el **responsable de inicializar y ejecutar las máquinas de estados** con las que controlaremos el sistema. Dichas máquinas de estados se ejecutarán cíclica y secuencialmente dentro de un **bucle infinito** mediante la **invocación**

permanente de las funciones *fsm_fire* ligadas a las mismas. Tales funciones *son* las encargadas de ejecutar las funciones de comprobación (si se ha producido algún evento especial para el sistema, e.g. que haya ocurrido una interrupción) y actualización del estado del sistema.

- **una interrupción periódica** (**TIMER1**): interrupción responsable de controlar la parte de temporización del sistema relacionada con el **refresco de la matriz de leds**, refresco del cual depende directamente su correcta visualización. Su ejecución es cíclica, discontinua y sucede a intervalos regulares de tiempo (1 vez cada 4 milisegundos); es concurrente con el proceso principal y las otras interrupciones periódicas. Se encargará de la ejecución de todas las funciones críticas en el tiempo relacionadas con la necesaria excitación de las diferentes columnas de leds del display. Este temporizador estará activo de forma permanente, desde el mismo momento en que finalice la inicialización del sistema e independientemente del estado en el que éste se encuentre de manera que, en cualquiera de dichos estados, resulte posible (aunque no obligatoria) la visualización de cualquier tipo de información en el display.

- **una interrupción periódica** (**TIMER2**): responsable de controlar la parte de **temporización del sistema relacionada con el refresco del teclado matricial empleado como dispositivo de control.** En particular, este temporizador se configurará para que interrumpa una vez cada 25 ms. Su ejecución será, por tanto, cíclica, discontinua y sucederá a intervalos regulares de tiempo; es concurrente con el proceso principal y el resto de interrupciones periódicas. Se encargará de la ejecución de todas las funciones críticas en el tiempo relacionadas con la necesaria excitación de las diferentes columnas del teclado. Este temporizador estará activo de forma permanente, desde el mismo momento en que finalice la inicialización del sistema e independientemente del estado en el que éste pueda encontrarse de manera que, en cualquiera de dichos estados, resulte posible la detección de la pulsación de cualquier tecla.

- **una interrupción periódica** (**TIMER3**): interrupción responsable de controlar la parte de **temporización del sistema relacionada con el movimiento automático de la pelota**, movimiento del cual depende también el correcto funcionamiento del juego. Su ejecución también es cíclica, discontinua y sucede a intervalos regulares de tiempo (1 vez cada segundo); es concurrente con el proceso principal y las otras interrupciones periódicas pero, a diferencia de aquellas, ésta **sólo estará activa durante el transcurso del juego**.

Al margen del programa principal y de cualquier proceso que requiera de algún control del tiempo como los mencionados, **serán igualmente necesarias 4 interrupciones externas** ligadas, respectivamente, a la exploración de cada una de las 4 filas del teclado de la TL04 mediante la cual podremos detectar la pulsación de las distintas teclas (en particular, de las teclas ligadas al movimiento hacia la izquierda y la derecha, respectivamente, de la raqueta).

Este esquema de procesos es obligatorio, en particular en cuanto concierne al uso de máquinas de estados.

Para la implementación de sistemas basados en máquinas de estados será de obligada lectura [5]. En ese documento encontrará la información necesaria para la definición y uso de máquinas de estados identificando para ello: los distintos estados del sistema, las posibles transiciones entre ellos, las funciones de entrada (o de

comprobación de eventos), y las funciones de salida (o de actualización del sistema) que gobiernan dichas transiciones.

Para el desarrollo de software de tiempo real sobre la plataforma disponible en el laboratorio es indispensable la lectura de [4]. En ese documento encontrará la información necesaria para la definición de diferentes procesos por medio de interrupciones periódicas, interrupciones externas y el programa principal, así como para la provisión de los requeridos mecanismos de sincronización entre todos ellos.

5.2. Proceso principal y máquinas de estados

El sistema completo propuesto estará basado en **cuatro máquinas de estados:**

- FSM1: máquina específica **para el control de la excitación de los leds del display**, representada en la Figura 10.
- FSM2: máquina específica para el **control de la excitación del teclado**, representada en la Figura 10.
- FSM3: máquina específica para **la detección y gestión de las diferentes pulsaciones de teclas**, representada en la Figura 10.
- FSM4: máquina específica **para el control de la raqueta y demás aspectos propios del juego**, representada en la Figura 11.

Las dos primeras máquinas de estados tienen un comportamiento muy similar: los procedimientos de excitación o refresco del display de leds y del teclado matricial son conceptualmente idénticos salvo por los periodos de refresco adoptados en cada caso. Por ese motivo, simplemente haremos particular hincapié en el diseño especialmente simple de las mismas. Dicho diseño propone una solución basada en un único estado (y no 4, como podría sugerir un primer análisis del problema) común a todas las columnas del dispositivo y que aprovecha una solución única, común y válida para la actualización de los 4 esquemas de excitación requeridos (i.e. la función de actualización correspondiente a la auto-transición recibe como argumento de entrada la columna en cuestión a procesar en cada caso). El funcionamiento de las dos máquinas de estados ligadas al manejo del teclado matricial está detallado en [3]. La última máquina de estados será objeto de análisis a continuación.

Nótese que en todos los casos se han destacado oportunamente los siguientes elementos: los posibles **estados** de cada subsistema, las **transiciones** contempladas entre todos ellos, las **funciones de comprobación** (en azul) de cuyo resultado depende el que se hagan efectivas dichas transiciones, los **flags** específicos (en rojo) que son objeto de comprobación por parte de dichas funciones y, finalmente, las **funciones de actualización** (en verde) responsables de llevar a cabo las acciones necesarias para instaurar el nuevo estado cuando proceda efectuar la transición en cuestión.

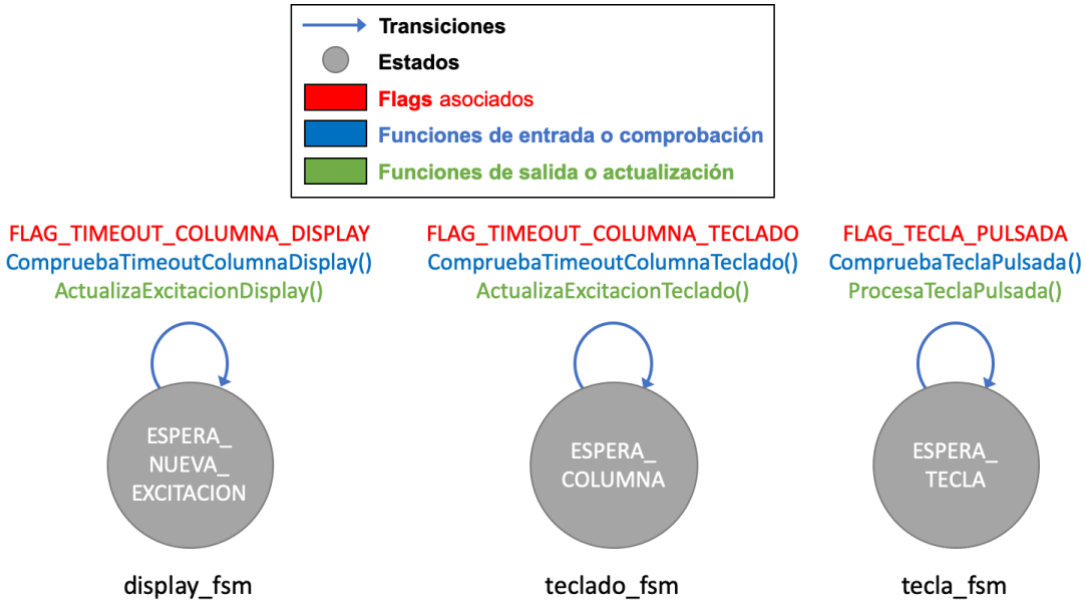


Figura 10. Máquinas de estados FSM1, FSM2 y FSM3 propuestas respectivamente para el control de la excitación del display, el control de la excitación del teclado y el procesado de las teclas pulsadas.

5.2.1. Control de la raqueta y demás aspectos del juego

Se trata, sin duda, del subsistema más importante desde el punto de vista del comportamiento del sistema final toda vez que, al margen de orquestar los procedimientos necesarios para gestionar el control de la raqueta, será además quien gobierne, mediante la activación y desactivación de los flags adecuados, el resto de aspectos propios de la mecánica y reglas del juego, incluyendo entre otros: el movimiento automático de la pelota y/o la inicialización/finalización del juego.

Como se ha comentado previamente, la interfaz de control para el usuario consiste fundamentalmente en el teclado matricial disponible en la TL04 con el que podemos dirigir la raqueta en las 2 direcciones permitidas dentro del área de juego (izquierda y derecha). El comportamiento descrito a continuación se repetirá indefinidamente hasta que se apague o reinicie el sistema.

El sistema parte de un estado inicial (i.e. S1 o WAIT_START) en el que se debe llevar a cabo la oportuna inicialización del sistema y presentar el correspondiente mensaje informativo indicando al usuario que el sistema está listo para ser utilizado.

Una vez completada dicha inicialización, el sistema permanecerá en dicho estado a la espera a que cualquiera de las teclas de control sea accionada por el usuario para dar así comienzo al juego.

Con el comienzo del juego el sistema alcanzará el estado *WAIT_NEXT* en el que permanecerá a la espera de que se produzca alguno de los siguientes eventos:

- **que el usuario accione el dispositivo de control**, eventos cuya ocurrencia estará ligada a la activación del cualquiera de los flags *FLAG_MOV_XXX,* activados por la FSM3, máquina específica para la detección y gestión de las diferentes pulsaciones de teclas. La ocurrencia de cualquiera de los citados eventos provocará que el sistema lleve a cabo todas las acciones oportunas para tratar de desplazar la raqueta en una posición en el display en la dirección pulsada (en caso de que sea posible, esto es, siempre y cuando no se excedan los límites del área de juego).

- **o que se agote el tiempo máximo para que la pelota permanezca en una misma posición**, evento cuya ocurrencia estará ligada a la activación del flag *FLAG_TIMER_JUEGO*, activado por la rutina o procedimiento de atención a la interrupción asociada al temporizador empleado para resolver la actualización de la ubicación ocupada por la pelota. Con motivo de la ocurrencia de este evento el sistema deberá:
  - Desplazar la pelota en el display conforme a su actual trayectoria.
  - Gestionar su eventual rebote contra algún elemento del juego mediante la actualización de su trayectoria. La pelota puede rebotar contra la raqueta, contra las paredes (o límites del área de juego) o contra un ladrillo. En este último caso el sistema deberá actualizar el área de juego ocupada por dicho ladrillo para su eliminación y el número de ladrillos restante.
  - Realizar la comprobación de las posibles condiciones para la terminación del juego: 1) por haber eliminado todos los ladrillos ó 2) porque la nueva posición de la pelota excede el límite inferior del área de juego. En ambos casos se debe llevar a cabo la activación del flag *FLAG_FIN_JUEGO*.

Finalmente, y una vez satisfecha una cualquiera de las posibles condiciones para la terminación del juego (i.e. *FLAG_FIN_JUEGO*), pasaremos al estado de finalización del juego (i.e. *WAIT_END*), estado en el que podremos aprovechar para informar al jugador del resultado del juego (e.g. número de ladrillos eliminados) y en el que podremos permanecer hasta que cualquier nueva acción por parte del jugador signifique el reinicio de una nueva partida retornándose al estado inicial.

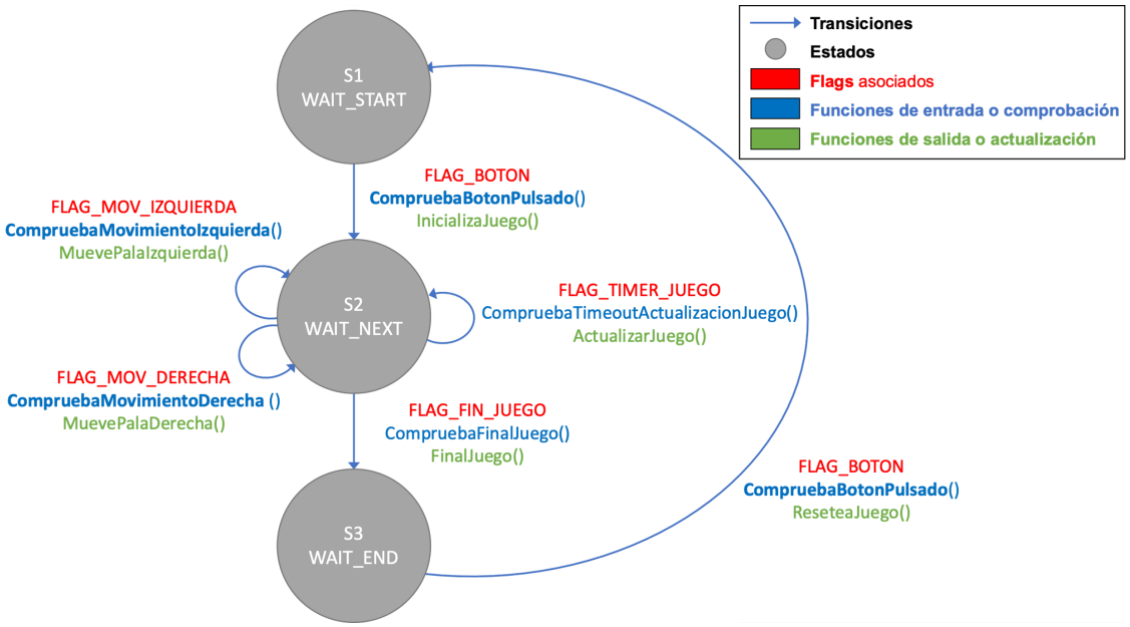


Figura 11. Máquina de estados propuesta para el control del juego.

5.2. Interrupciones

Las máquinas de estados pueden hacer uso de las interrupciones, independientemente de que estas sean periódicas (i.e. vinculada a un temporizador) o externas (i.e. vinculada a una entrada digital), por medio de la misma aproximación presentada en [5]: utilizaremos una **variable global para indicar si se ha producido o no** la interrupción en cuestión (a modo de flag). De este modo, la ocurrencia de cada interrupción provocará la llamada y ejecución de una **función de atención a la**

interrupción (procedimientos cuyo nombre termina por _isr conforme a la nomenclatura empleada en [5], correspondiente a Interrupt Service Routine), **que pondrá a 1 la variable activando el correspondiente flag**.

Cuando el sistema, a través de la máquina de estados, haya tenido en cuenta el evento reflejado por dicha activación, volverá a poner la variable, y con ello el flag, a 0. Con objeto de simplificar la gestión de las diferentes interrupciones y sus correspondientes flags asociados, en vez de tener una variable para cada interrupción, como solo hace falta un bit, **utilizaremos una única variable global *flags* para todas** (al utilizar un entero de 32 bits hay espacio para tener 32 interrupciones).

El uso de un conjunto de máscaras con las funciones binarias OR y AND nos permitirán referirnos a cada uno de los flags por separado para su oportuna lectura o escritura (el alumno debe leer [5] para comprender en detalle estos mecanismos; también puede consultar el anexo 8.3 incluido al final del presente documento).

Como consecuencia inmediata de simplificar, según el modo sugerido, la atención a las interrupciones, el tiempo necesario para completar dicha atención será despreciable en comparación con el requerido por *fsm_fire* para actualizar (si procede) el estado del sistema. El alumno **deberá diseñar un valor adecuado para la constante CLK_FSM** que permita una frecuencia de actualización del estado del sistema lo suficientemente alta como para que los tiempos de respuesta sean bajos (algo necesario para una buena experiencia de juego), y sin que por ello se vean comprometidas ninguna de las tareas a realizar en el peor escenario posible (i.e. co-ocurrencia múltiple de eventos entre sucesivas llamadas a *fsm_fire*).

Nótese que las interrupciones, ya sean de naturaleza externa o ligadas a un temporizador, pueden interrumpir la ejecución del programa principal en instantes de tiempo que resultan imprevisibles en el momento de escribir el código. Por tanto, será **fundamental contar con los mecanismos de sincronización oportunos que nos permitan actualizar correctamente todos los objetos compartidos** (conjunto de variables globales) entre los distintos procesos con el fin de intercambiar información. Preste para ello especial atención al capítulo 6 de [4] y en particular a la sección 6.5. Sincronización de threads.

6. Calendario de la asignatura: fechas importantes

A continuación, se facilita un **calendario** en el que se han destacado algunas **fechas importantes** para el desarrollo de la asignatura. **Revíselas concienzudamente** pues algunas de ellas afectan específicamente a los procedimientos de evaluación.

En la Figura 12 se ha añadido a la derecha del calendario una **planificación por sesiones orientativa**[7] en relación al posible desarrollo del proyecto y su correspondencia con las diferentes semanas del curso. Los detalles específicos de dicha planificación se han organizado en diferentes documentos, **uno por cada bloque o versión funcional del proyecto**, disponibles en Moodle y cuya lectura se recomienda igualmente al alumno como complemento al presente enunciado.

[7] Nótese que, debido a la acumulación de festivos a lo largo del curso, algunos grupos pueden alcanzar ciertas sesiones 1 semana antes o después que el resto.

Tal y como el alumno podrá observar, el diseño propuesto ha sido dimensionado en coherencia con las 12 sesiones de laboratorio disponibles[8]. En este sentido, es importante destacar que **la parte estrictamente obligatoria** (y suficiente para aprobar la asignatura[9]), **desde el punto de vista de los requisitos funcionales que debe cumplir el sistema final, ocupa aproximadamente la mitad de las sesiones,** quedando la otra mitad a libre disposición del alumno para la realización de cuantas mejoras del sistema adicionales estime oportunas.

Por tanto, y con objeto de, no sólo poder disponer del mayor número de sesiones posible para la puesta a punto y mejora del sistema, sino también para garantizar una distribución del esfuerzo requerido a lo largo del semestre lo más racional y equilibrada posible, **se recomienda al alumno un seguimiento de dicha planificación lo más estricto posible.**

Para cualquier duda en relación a los procedimientos y fechas indicadas, no dude en ponerse en contacto con el coordinador docente o administrativo de la asignatura (Manuel Gil Martín, manuel.gilmartin@upm.es).

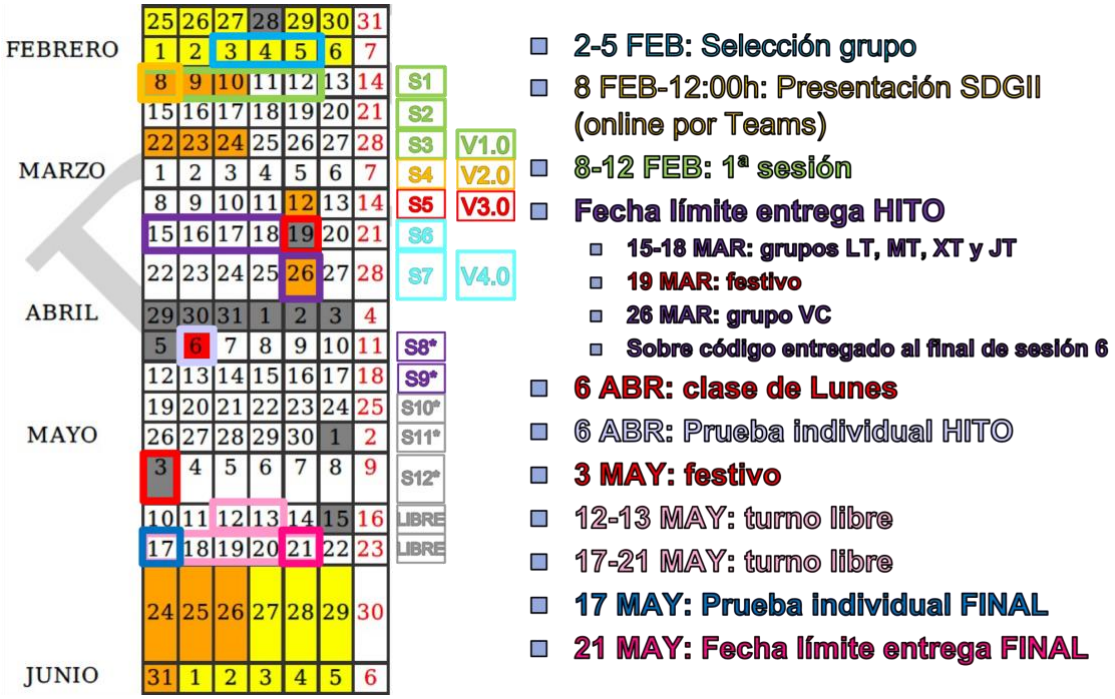


Figura 12. Calendario de fechas importantes para la asignatura.

6.1. Entregas electrónicas del código previstas

Las diferentes versiones a implementar del sistema exigirán la entrega vía Moodle del SW desarrollado. Concretamente, los alumnos deberán realizar una entrega del mismo al final de la última sesión dedicada a cada versión (e.g. primera entrega al finalizar la sesión 3).

En el calendario reflejado en la Figura 12 se ha añadido una columna adicional a la izquierda (e.g. "V1.0") que hace referencia a la versión del sistema prevista a la

[8] Alguna más si tenemos en cuenta los turnos libres que contempla el calendario de la asignatura.
[9] Como es obvio, siempre y cuando se superen las diferentes pruebas de evaluación previstas conforme a lo especificado en la guía docente de la asignatura.

finalización de cada semana y que indica, por tanto, la obligatoriedad de realizar una entrega del código desarrollado hasta ese momento independientemente de que cumpla o no los requisitos previstos.

En resumen, los alumnos **deberán realizar un total de 5 entregas**: 4 correspondientes a las diferentes versiones 1.0-4.0 del sistema, conforme a las especificaciones básicas y obligatorias, y una última entrega final con las posibles mejoras que se hayan podido añadir al mismo.

7. Material complementario

[1] Introducción al entorno de desarrollo en C para Raspberry Pi.
https://moodle.upm.es/titulaciones/oficiales/mod/folder/view.php?id=851543

[2] Prácticas de lenguaje C para Raspberry Pi.
https://moodle.upm.es/titulaciones/oficiales/mod/folder/view.php?id=846845

[3] Iniciación al Manejo de las Entradas/Salidas del BCM 2835.
https://moodle.upm.es/titulaciones/oficiales/mod/folder/view.php?id=851738

[4] Manejo de temporizadores, interrupciones y procesos con la Raspberry Pi.
https://moodle.upm.es/titulaciones/oficiales/mod/folder/view.php?id=851744

[5] Introducción a las máquinas de estados en C para Raspberry Pi
https://moodle.upm.es/titulaciones/oficiales/mod/folder/view.php?id=851735

[6] Manejo de DAC/ADC vía SPI.
https://moodle.upm.es/titulaciones/oficiales/mod/folder/view.php?id=851746

[7] Manejo de Display LCD con Raspberry Pi B+.
https://moodle.upm.es/titulaciones/oficiales/mod/folder/view.php?id=865517

[8] Informe 2015 de la Asociación Española de Videojuegos
http://www.aevi.org.es/web/wp-content/uploads/2016/06/MEMORIA-ANUAL_2015_AEVI_-definitivo.pdf

[9] Circuitos Electrónicos (CELT). Descripción del proyecto. Curso 2018-2019.
https://moodle.upm.es/titulaciones/oficiales/pluginfile.php/882276/mod_resource/content/22/Enunciado_2018_19_vMAS_v11.pdf

[10] Hojas de características del LM386
http://www.ti.com/lit/ds/symlink/lm386.pdf

[11] Tutorial de C online e interactivo
http://www.learn-C.org

[12] Tutorial de C online e interactivo
http://www.sololearn.org

[13] Tutorial de C online
https://www.cprogramming.com/tutorial/c-tutorial.html

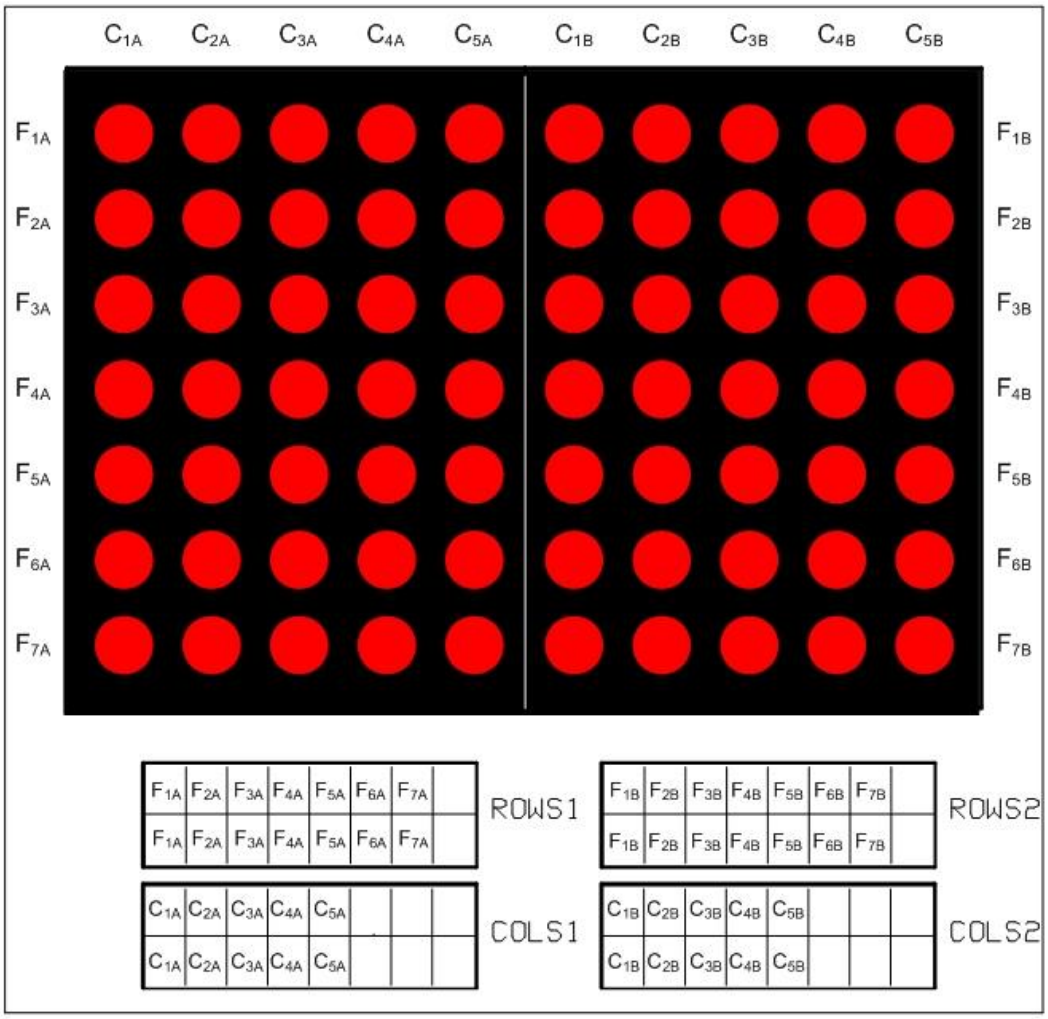
[14] 2017 Embedded Markets Study, Integrating IoT and Advanced Technology Designs, Application Development & Processing Environments. April 2017. EE|Times.
https://m.eet.com/media/1246048/2017-embedded-market-study.pdf

8. ANEXOS

8.1. Tabla de pines de la placa auxiliar TL04

ENTRADAS				SALIDAS			
Placa TL-04 conexión entradas entrenador		Número del puerto BCM sólo utilizable como entrada	ENT2004CF LEDS ROJOS	ENT2004CF LEDS VERDES	Número del puerto BCM sólo utilizable como salida	Placa TL-04 conexión salidas entrenador	
Pin							Pin
1	Teclado Fila 1	**5**	LRE0	LVS0	**0**	Teclado Columna 1	1
2	Teclado Fila 2	**6**	LRE1	LVS1	**1**	Teclado Columna 2	2
3	Teclado Fila 3	**12**	LRE2	LVS2	**2**	Teclado Columna 3	3
4	Teclado Fila 4	**13**	LRE3	LVS3	**3**	Teclado Columna 4	4
5	Libre	**09** (SPI_MISO)	LRE4	LVS4	**04** (GPCLK0)	Libre	5
6	Libre	**15** (UART_RXD)	LRE5	LVS5	**07** (SPI_CS1)	Libre	6
7	Libre	**16**	LRE6	LVS6	**08** (SPI_CS0)	LCD_RS	7
8	Libre	**19**	LRE7	LVS7	**10** (SPI_MOSI)	LCD_Enable	8
9	NC	NC	-	-	NC	NC	9
10	ADC/DAC SPI_CLK	ADC/DAC SPI_CLK	-	-	ADC SSTRB	ADC SSTRB	10
11	ADC/DAC SPI_Din	ADC/DAC SPI_Din	-	-	ADC/DAC SPI_Dout	ADC/DAC SPI_Dout	11
12	SPI_CS_ADC	SPI_CS_ADC	-	-	NC	NC	12
13	SPI_CS_DAC	SPI_CS_DAC	-	-	NC	NC	13
14	Libre	**20**	LRE8	LVS8	**11** (SPI_CLK)	LCD_Bit 0	14
15	Libre	**21**	LRE9	LVS9	**14** (UART_TXD)	LCD_Bit1	15
16	Libre	**26**	LRE10	LVS10	**17**	LCD_Bit 2	16
17	Libre	**27**	LRE11	LVS11	**18**	LCD_Bit 3	17
18	NC	NC	LRE12	LVS12	**22**	LCD_Bit 4	18
19	NC	NC	LRE13	LVS13	**23**	LCD_Bit 5	19
20	NC	NC	LRE14	LVS14	**24**	LCD_Bit 6	20
21	NC	NC	LRE15	LVS15	**25**	LCD_Bit 7	21
22	NC	NC	-	-	NC	NC	22
23	NC	NC	-	-	NC	NC	23
24	GND_Exterior	GND_Exterior	-	-	GND-Exterior	GND_Exterior	24
25	GND_Exterior	GND_Exterior	-	-	GND-Exterior	GND_Exterior	25
26	NC		-	-		NC	26

8.2. Tabla de pines de la placa auxiliar con la matriz de leds



ATENCIÓN: no olvide **emplear unas resistencias para limitar la corriente** que llega a los displays con objeto de evitar que éstos se deterioren (véase esquemático sugerido en el enunciado). Puede emplear resistencias individuales o bien un array de resistencias.

Recuerde igualmente que los **diodos están configurados con él ánodo en común para cada columna** por lo que si desea encender el led que ocupa la posición (C_i,F_j) deberá "escribir" un '1' (i.e. 5v) en la columna C_i y un '0' (i.e. 0v) en la fila F_j (en el resto de filas debe escribirse un '1' si sólo se desea encender ese led).

8.3. Sobre el uso de flags y máscaras

En nuestro proyecto utilizaremos una única variable global, la variable flags, para indicar la ocurrencia de todos los eventos que afectan al funcionamiento del sistema (e.g. si se ha producido la inserción de una tarjeta o, como veremos en los próximos ejemplos, si se ha producido la pulsación de una determinada tecla o no).

Si tenemos en cuenta que basta un bit para indicar la ocurrencia de un determinado evento, dado que la variable flags es de tipo int, disponemos de 32 bits para gestionar la ocurrencia de hasta un total de 32 eventos distintos (más que suficiente para cubrir nuestras necesidades en el proyecto).

En este sentido, las máscaras juegan un papel fundamental toda vez que nos permiten referirnos a cada uno de los flags (i.e. bits) por separado por medio de las funciones binarias OR y AND (| y &).

En particular, un flag determinado puede activarse con la función binaria OR de nuestra variable flags con una constante (i.e. máscara) que tenga todos los valores a 0 salvo aquél que se desea activar. Por ejemplo:

```
#define FLAG_TECLA_OFF 0x02

flags|=FLAG_TECLA_OFF; // Pone a 1 únicamente el segundo bit de los 32 disponibles en la variable
```

Igualmente, un flag puede limpiarse o desactivarse con la función binaria AND de nuestra variable flags con una máscara que tenga todos los valores a 1 salvo aquél que se desea desactivar. Para ello podemos hacer uso de la función binaria NOT (~) para obtener la versión negada de la máscara del flag. Por ejemplo:

```
flags&= ~FLAG_TECLA_OFF; // Pone a 0 únicamente el segundo bit de los 32 disponibles en la variable
```

Finalmente, también podemos consultar el valor de un flag con la función binaria AND de nuestra variable flags con la misma máscara que empleamos para su activación (i.e. constante que tenga todos los valores a 0 salvo aquél que se desea consultar). Por ejemplo:

```
result= (flags& FLAG_TECLA_OFF); // Devuelve el valor, 1 ó 0, correspondiente al flag en cuestión
```

Para que estos 3 procedimientos funcionen correctamente es fundamental que las máscaras sólo afecten a un único bit. A continuación se muestra a modo de ejemplo la correcta definición de máscaras para los 6 primeros bits de la variable flags:

```
#define MASCARA_FLAG_BIT1 0x01
#define MASCARA_FLAG_BIT2 0x02
#define MASCARA_FLAG_BIT3 0x04
#define MASCARA_FLAG_BIT4 0x08
#define MASCARA_FLAG_BIT5 0x10
#define MASCARA_FLAG_BIT6 0x20
...
```

Ejemplos de definición incorrecta de tales máscaras, fundamentalmente por no cumplir con el requisito de afectar a un único bit, podrían ser los siguientes:

```
#define MASCARA_FLAG_BIT3 0x03
#define MASCARA_FLAG_BIT4 0x04
#define MASCARA_FLAG_BIT5 0x05
#define MASCARA_FLAG_BIT6 0x06
...
```