The embedded questions (agosto 2021)

Alejandro Garnung Menéndez

I. Introducción

ESTE documento es una curiosa y somera formalización de una serie de cuestiones originalmente planteadas para ser "lanzadas" expresamente en entrevistas de trabajo a potenciales desarrolladores y programadores de sistemas embebidos o empotrados ("sistemas computarizados que están construidos deliberadamente para su aplicación").
Estas cuestiones y los conceptos que sugieren pertenecen al imaginario colectivo de todas las personas que se dedican a este campo de estudio e investigación, pero los méritos particulares (en el caso que se va a tratar en este informe) van para ciertos contribuidores y desarrolladores en el dominio de internet www.eevblog.com (*Electronics Engineering Video Blog*) (fundado por el ingeniero electrónico australiano David L. Jones). Con un total de 12 interesantes y variadas "preguntas", el lector abordará la interiorización de una gran diversidad de conceptos, relacionados mayormente con campos de conocimiento como la electrónica analógica y digital (dispositivos, microcontroladores, microprocesadores, etc.), programación de alto nivel (lenguaje C y C++, …), comunicaciones (estándares para la transferencia de información entre distintos CI) o la ingeniería de control. Tanto durante el proceso de comprensión de cada cuestión como durante la asimilación de sus posibles respuestas y observaciones, se puede ampliar y comprender mejor el "espectro visible" de este tema tan interesante que conforman los sistemas embebidos, en general.

II. Reentrancia, hilos y direcciones

What's the difference between re-entrant, thread-safe, and position-independent code?

En castellano: ¿cuál es la diferencia entre reentrancia, seguridad en hilos y código independiente de la posición?
Si bien el conjunto de diferencias que puedan presentar estos tres conceptos no está muy bien definido simplemente describiendo qué es cada uno de ellos, se comentará de forma básica en qué consisten para luego remarcar en qué difieren. También es importante recalcar que las explicaciones pueden ser asimilables y aplicables a cualquier lenguaje (usual o computacional), aunque para una serie de ejemplos en este informe se utilice "vocabulario" o conceptos asociados al lenguaje C (desarrollado entre los años 1969 y 1972 por Dennis MacAlistair Ritchie en los Laboratorios Bell) y C++ (diseñado en 1979 por Bjarne Stroustrup, un lenguaje híbrido que combina la versatilidad a bajo nivel de C y los paradigmas de la programación orientada a objetos).
Como primera definición se puede plantear lo siguiente: una función (o un programa, una subrutina, …) *reentrante* es aquella que puede ser interrumpida mientras se está ejecutando y, además, ser llamada (volver a entrar a ella) nuevamente, antes (o justo antes) de que la interrupción termine, de manera que se asegure la mantenibilidad y fiabilidad de los datos durante todo el proceso, sin corrupción alguna. De manera más directa, una función es reentrante si se la puede llamar de forma segura múltiples veces, mientras se está ejecutando.
Las causas de la interrupción son varias, pueden ser desencadenadas por *hardware* (evento externo, señales, sensores, …) o por *software* (llamadas, desbordamientos, saltos, …).
Se ve que, por ahora, un hipotético código que pudiera contener una o varias funciones reentrantes, se puede acotar (de la forma más básica) en un solo hilo y flujo de ejecución, lo que sería equivalente (generalmente) a un solo núcleo de procesamiento.
Por otra parte, la transferencia de información es "segura" si, por ejemplo, un dato perteneciente a una función que se presupone igual a una cantidad fija no varía su valor final tras la interrupción y la vuelta a la función original, aun si se ha usado ese dato durante la ejecución de la rutina de interrupción. Cabe decir que una función no será reentrante si ella misma llama a otras funciones no reentrantes, así como si utiliza variables globales (que no pueden ser almacenadas en registros, sino en RAM, por lo que tras la llamada de interrupción pueden quedar corruptas) (de hecho, se suele hacer uso de variables locales, en su lugar). Para proporcionar al código una capa de seguridad adicional se ha de verificar que la función reentrante solo haga uso de sus argumentos de entrada; si fuese necesario recordar algo (de antes de la llamada) tras la llamada, en el propio tratamiento de la interrupción tendría que ocuparse del almacenamiento y de preservar el contexto si fuera necesario.
Si en vez de un solo hilo tenemos la posibilidad de implementar más (es decir, varios núcleos de procesamiento), recae la labor de usar técnicas de programación multihilo (varios flujos de ejecución simultáneos) (realmente se pueden pseudoparalelizar [sub]procesos, creando así hilos "extra" que aprovechan los tiempos de espera que se producen cuando trabajan los propios núcleos o procesadores). Al haber varios hilos ejecutándose simultáneamente, en cada uno de ellos se puede consultar o modificar cualquier variable a la que tengan acceso de forma cruzada, dando lugar fácilmente a errores y corrupción de la información.
De este problema surge el segundo concepto presentado anteriormente: una función es *segura en hilos* si se la puede llamar desde distintos hilos y los datos que se manejan en ella (compartidos o no) no se ven comprometidos frente a interacciones no deliberadas. Se dice que el acceso a los datos ha de estar "serializado"; es decir, el proceso ejecutado por un hilo ha de seguir todos los pasos concatenados (hasta su finalización) para que otro hilo distinto pueda comenzar a

ejecutar de nuevo ese proceso. Una manera típica de otorgar esta característica es mediante un *mutex* (exclusión mutua), que funciona, cualitativamente hablando, como un candado convencional para controlar el acceso de los hilos a las funciones (realmente es una variable especial que toma dos estados, *locked* o *unlocked*, en función de la dirección del flujo de información que sale o que proviene de la función sensible a datos compartidos). Además, si posteriormente se diese algún intento de volver a acceder a datos compartidos mientras un hilo ya está manejándolos (estado locked del mutex), saltaría una alarma y se bloquearía este último hilo solicitante, hasta que el primero haya terminado de acceder a los datos (esto asegura que solo trabaje un hilo por proceso).
Ahora bien, pueden existir funciones reentrantes que a su vez sean seguras en hilos y viceversa (así como cualquier combinación entre estas características).

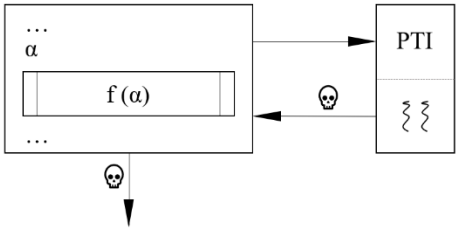

Fig. 1. Función *no reentrante* y *no segura en hilos*. Una función que use variables globales (no constantes) (α) sin capas de seguridad, puede causar información corrupta, ya sea por medio de interrupciones (PTI, programa de tratamiento de la interrupción) o por datos compartidos con otros hilos (procesamiento multihilo, simbolizado debajo de PTI).

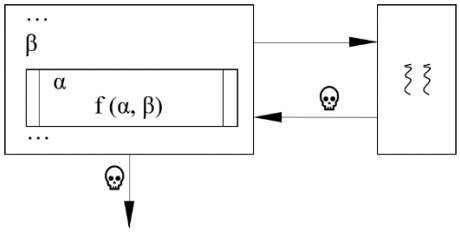

Fig. 2. Función *reentrante* y *no segura en hilos*. Suponiendo que β es una variable global potencialmente compartida con otros hilos (aunque la función usa variables locales como α, lo que puede servir para ser reentrante), los datos no se manejan de forma segura (se puede salvaguardar el valor de β en α al inicio de la función para devolvérselo a la propia β al final de esta [reentrable], pero no se está controlando que otros hilos no manejen β).

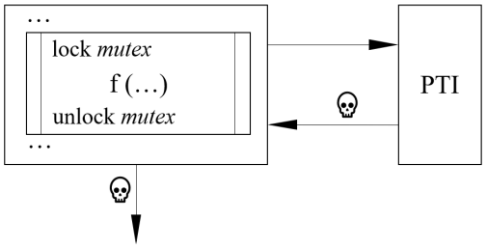

Fig. 3. Función *no reentrante* y *segura en hilos*. Mediante la técnica de mutex comentada anteriormente, se consigue que los hilos accedan unívocamente a los datos (compartidos). Otra opción sería definir una variable de clase (un tipo de tipo) "thread local" (como puede ocurrir en lenguajes como Java), que restrinja su lectura y escritura (modificación) a un solo hilo.

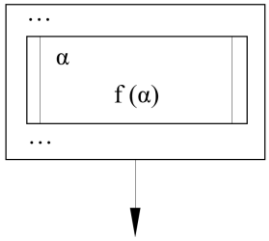

Fig. 4. Función *reentrante* y *segura en hilos*. Esta función solo usa variables locales (cada hilo no proporciona ni maneja más datos que los suyos propios) y no llama a ninguna otra función que sea no reentrante ni no segura en hilos, por lo que cumple los mínimos requisitos para satisfacer ambas cualidades.

El tercer y último concepto es el de código (o ejecutable) *independiente de la posición* (PIC o PIE). Esta cualidad la presentan datos (objetos) compartidos que no están vinculados a una dirección de memoria fija, sino que pueden ser ejecutados correctamente e independientes del proceso y de la dirección de memoria; por ejemplo, bibliotecas compartidas que se pueden cargar en una ubicación en el espacio de direcciones de cada programa que las llame (con esta técnica se añade una capa extra de indirección a los datos compartidos y las referencias). Si bien los dos primeros conceptos se referían, en cierta forma, a la interacción entre varios procesos e hilos interdependientes, los PIC pueden estar dirigidos más al direccionamiento en memoria (más aún, es importante en sus características el hecho de que usan direccionamiento relativo y no absoluto o directo; es decir, se manejan por desplazamientos [*offsets*] entre secciones de la pila del programa).

III. Bootloader

How would you write a bootloader for a microcontroller with only a single FLASH region?

En castellano: ¿cómo escribirías un gestor de arranque para un microcontrolador con una sola región *flash*?
El término *bootloader* hace referencia a un programa básico que es capaz de cargar y ejecutar un código (de inicialización) en el proceso de arranque (en este caso del microcontrolador, pero se refiere indistintamente a un computador, móvil, etc.). Este ha ido mejorando enormemente a lo largo de los años; desde los programas cableados y con interruptores (como la ENIAC [Electronic Numerical Integrator And Computer]), desencadenantes (primeros bootloaders) de otros más grandes almacenados en tarjetas perforadas, pasando por las primeras ROM "físicas" (de la PDP-11, p. ej.) incrustadas (mediante diodos, entre otros) en la propia computadora, hasta los actuales basados en memoria EEPROM y flash.
No debe confundir que la pregunta se refiera particularmente a un microcontrolador con una sola región flash (como podría haber en algún PIC16 o el ATMega328p presente en Arduino Mega), ya que cualquier otro microcontrolador dado puede carecer de esta y depender de un dispositivo externo que almacene el *firmware* inalterable (que sirve de "barrera" entre el hardware y el software, de ahí su nombre [software firme]) (controla elementalmente los circuitos electrónicos a más bajo nivel), de manera que este último copie el código del exterior a una región de memoria del μC desde la cual sí se pueda ejecutar (como la RAM). En resumen, el bootloader es esencial para la seguridad, privacidad y fiabilidad del arranque de una arquitectura informática.
Si se quisiera construir un bootloader, se partiría de la idea de que ha de realizar dos tareas principales: ejecutarse con el encendido del microcontrolador (en este caso) y dar paso a la aplicación que este último vaya a realizar (código o programa). Como un bootloader ocupa espacio (a fin de cuentas, es código almacenable) es preferible minimizar todo lo posible su tamaño (ya que la aplicación o programa principal frecuentemente requiere espacio adicional en algún momento "inoportuno"). El primer paso sería hacer un mapa de memoria donde se establezca el inicio y el tamaño del bootloader (almacenado en un sector [el primero del disco duro, …] generalmente de 512 bytes para los PC [Master Boot Record]) y de la aplicación. Ha de asegurarse que estos

dos se encuentren en sectores diferentes para evitar que el bootloader sea borrado (en definitiva, para que el sistema pueda arrancar) y el firmware pueda ser actualizado en situaciones futuras.

Una característica de C es que no es capaz de producir archivos de texto planos (que solo contienen información binaria textual, desligada a referencias sobre el formato del propio programa); por tanto, el "punto de entrada" (*entry point*) no se situaría en el primer byte del archivo de programa, sino típicamente en el comienzo del *main()*. Años antes, cuando se usaban sistemas operativos como el DOS (*Disk Operating System*¸ lanzado en 1981), los programas sí eran binarios planos: al encender la alimentación, la BIOS del sistema (*Basic Input Output System*¸ dentro de la ROM) "toma el control" y desencadena la ejecución del bootloader, que es el encargado de arrancar el sistema operativo (SO). La BIOS "coloca" el puntero de entrada en el primer byte del archivo del gestor de arranque (o más bien "salta" hacia él), que tiene que estar escrito en lenguaje ensamblador para, como se ha comentado antes, proporcionar información y órdenes "crudas" sin un formato en específico (al contrario de lo que otorgaría un compilador de C) (si bien es cierto que existen alternativas para no utilizar íntegramente lenguaje *asm* o ensamblador).

Algunas de las placas de desarrollo embebido comerciales ya disponen de un bootloader propio (en ROM) pregrabado (como en el ESP32, la familia LPC11xx de *NXP Semiconductors* o múltiples modelos de Arduino), que permite al usuario cargar fácilmente programas disponibles en el PC vía USB (en su uso más general) y desocuparse de usar otras herramientas que permitirían la grabación de forma directa en la memoria flash del microcontrolador (como serían JTAG o ATMEL-ICE en conjunto con USBTinyISP o J-link PLUS). El motivo por el que no todos los microcontroladores disponen de un bootloader (ni mucho menos) es que una inmensa cantidad de ellos se destinan a aplicaciones (como el automovilismo) en las que, a lo largo de su vida útil, solo van a estar usando un programa y, por tanto, solo necesitan ser programados (potencialmente) una única vez.

Por ejemplificar, si se quisiera escribir un bootloader en un µC (microcontrolador) PIC, habría que usar un depurador (y programador) externo, como puede ser el recurrente PICkit3; el archivo hexadecimal se transfiere del PC a la memoria flash del µC por medio de una interfaz específica (en el propio PC, por ejemplo, aunque pueden servir otros buses de comunicación).

El bootloader puede decidir si tiene que cargar una nueva aplicación del usuario o ejecutar la que estuviera ya precargada en el µC. La decisión tomada será función (tras la alimentación o una pulsación de "reset") de eventos provocados por hardware (cambio de estado de un PIN específico, …) o por software (recepción de una trama de bits concreta por un canal de comunicación [puerto serie o paralelo, …]).

En conclusión, a la hora de programar un microcontrolador para su aplicación, se ha de reflexionar sobre si merece o no la pena el espacio y el tiempo que "le quita" el bootloader al propio procesador, teniendo en cuenta que una elección que cumpla los mínimos requisitos de memoria se puede corresponder a la más acertada (en términos de beneficios económicos) para grandes volúmenes de producción (es importante revisar los requerimientos mínimos del sistema).

IV. KEYWORDS

What does the C keyword "volatile" do and when/how is it used? Same for "const" and "static".

En castellano: ¿qué hace la palabra clave de C "volatile" y cómo/cuándo se usa? Lo mismo para "const" y "static".

Los compiladores son software informático, funcionan como intermediarios entre el código máquina (que realmente "comprende" y procesa la CPU [unidad central de procesamiento]) y cualquier lenguaje de programación (código fuente) que use o con el que sea ducho el programador, ya sea en alto nivel (C++, Fortan, Java, …), medio (C, RTL, Forth) o bajo (como el ensamblador). Generalmente, se da un paso intermedio que consiste en traducir primero a lenguaje ensamblador (ensamblar), que posteriormente es usado para generar el código máquina pertinente (conjunto de instrucciones en binario o típicamente, hexadecimal, para ser mostradas por pantalla).

Del desglose de un programa durante el proceso de compilación se pueden extraer ciertos grupos de caracteres, o *tokens*, que constituyen la mínima unidad individual del mismo. Los tokens se distinguen gracias a unos elementos que los disocian denominados "separadores"; como pueden ser los espacios, tabulaciones verticales y horizontales, nuevas líneas y retornos de carro. Entre los distintos tipos de tokens existentes en los lenguajes C y C++, se encuentran: las palabras clave (*keywords*), los identificadores (caracteres alfanuméricos que sirven para designar entidades del programa [funciones, variables, clases, …]), constantes (en el sentido más natural del lenguaje [datos invariables] y en el sentido de referirse a variables de "tipo" constante), operadores (matemáticos, lógicos, …) y signos de puntuación (o "puntuadores", habiendo algunos que también pueden ser usados como operadores).

Por tanto, las keyword (o palabras reservadas) son un tipo de tokens y, particularizando para los lenguajes C y C++, si en el primero hay un total de 32 palabras clave (en el estándar actual, ya que en el "primigenio" C de sus creadores había 27), en el segundo hay 63.

Para empezar a describir qué hace cada keyword que aparece en la pregunta, se puede empezar por imaginar una situación: se necesita inicializar una variable (por el motivo que sea) antes de una rutina en particular que se va a estar repitiendo indefinidamente. Ahora bien, da la "casualidad" de que esta variable (que se ha definido como un guarismo concreto antes de la rutina) está asociada a un parámetro mensurable real (hardware, proceso físico externo, …) y, además, si sobrepasa un determinado valor (por efecto de una interrupción, un puerto de E/S, …), ha de salirse de la rutina y pasar "a otra cosa". Pues bien, para este propósito existe la palabra clave "volatile" (volátil). Es preferible declarar con este tipo cualquier variable que sea potencialmente no constante; se informa al compilador que no debe hacerse ninguna suposición de que esta variable no pueda ser modificada por una subrutina paralela o en segundo plano, fuera del hilo esperado del programa (mapeado de registros de memoria con periféricos, modificaciones por servicios de interrupción, acceso a variables dentro de una aplicación multihilo, …). Alternativamente, también se está indicando que esta variable no ha de ser interpretada como una "variable de registro" (de rápido acceso en memoria), en cuyo caso se aceleraría, en

cierto modo, la ejecución del código del programa.

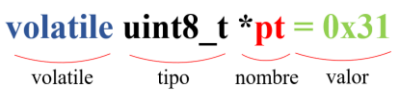

Fig. 5. Sintaxis de la palabra reservada volatile. En este caso se declara un puntero a la variable sin signo (*unsigned*) entera (*integer*) de 8 bits (equivalente al tipo *char*) de nombre "pt", asignado a la dirección que se encuentra indicada como valor.

Ahora se puede suponer otra situación: alguien está desarrollando una librería (matemática, p. ej.) o cualquier apartado o sección de datos fundamentales y quiere asegurarse de que, al ser constantes con muchos decimales (de los que depende la importante precisión del código), no puedan ser modificadas a lo largo del programa (menos aún durante su ejecución). Así pues, estos datos han de ser declarados de tipo "const" (constante). Consecuentemente, un intento de redefinir su valor (tras su declaración) causará un error de compilación. Lo mismo sucede si se designa así a una variable parámetro de una función (solo están disponibles para su lectura). Tiene una característica especial, relacionada con que C++ sea un lenguaje tipado o de tipado fuerte (son necesarias conversiones para tratar el valor de una variable como si fuera de otro tipo distinto al que es); es aplicable a cualquier objeto o dato de cualquier tipo, resultando así uno nuevo que mantiene las características del anterior pero que no puede ser modificado tras su inicialización. A no ser que sea declarado como "extern" (para definir que una variable global existe en otro fichero del programa; redundante escribirlo para declaraciones de funciones), a un identificador de variable constante ha de asignársele el valor de inicialización en el mismo momento de la declaración (misma línea de instrucción). La sintaxis en estos casos es análoga a la del especificador o calificador de tipo volatile.
De las distintas variaciones resultantes de intercalar esta keyword en diferentes posiciones, surge una importante diversidad de tipos de variables; en cierto modo unos son leves deformaciones de los otros. En la siguiente figura se muestran algunos de estos posibles tipos.

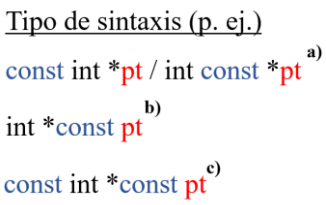

Fig. 6. Diferentes usos de la palabra reservada const. a) Puntero a constante. (ambas opciones son válidas). Está permitido cambiar la dirección de memoria a la que apunta el puntero, pero no cambiar el contenido (valor) de la dirección a la que está apuntando. b) Puntero constante a variable. Se puede cambiar el valor del entero al que se está apuntando, pero no se puede cambiar la dirección a la que apunta el puntero "pt". c) Puntero constante a constante. No se puede modificar ni la variable apuntada ni la dirección a la que se apunta. Por ejemplo, si se quisiera cambiar la variable: pt = &valor; si quisiera cambiarse la dirección: *pt = dir.

Como clarificación, es importante recordar que (refiriéndose a variables o funciones) mientras una "declaración" simplemente establece inespecíficamente qué existe y cuál es su tipo (o en caso de las funciones cuáles son sus argumentos, el tipo de sus datos, del retorno, …), con una "definición" hay, además, una asignación de memoria específica (la definición completa, en cierto modo, a la declaración). Indistintamente de las veces que se declare una variable o función, solo se puede definir una vez (es un error que existan varias asignaciones simultáneas de posiciones de memoria para una misma entidad).
Por último, se puede suponer una situación en la que existen muchas rutinas "independientes", las cuales se ejecutan impredeciblemente en ciertos momentos y se dan en forma de varias funciones que realizan determinadas tareas (irrelevante especificar más). Si llegado a un punto, el programador (o más bien, la persona que ejecuta el código) quiere acceder a una variable en concreto, cuyo valor se encuentra constantemente actualizado dentro de cada una de estas funciones de subrutina mencionadas, primero ha de asegurarse de que, efectivamente, estas variables son capaces de mantener su último valor (inmediato anterior a la salida de su bloque o ámbito en el código) y no lo pierden al terminar de ejecutarse la función (por ejemplo). Para este fin se diseñó la palabra reservada "static" (estática). Es, en cierto modo, la contrapartida de "auto" (automática); mediante esta última se definen variables locales (solo son conocidas por el bloque en el que son definidas, hasta que termine el programa) (de hecho, este es el tipo por defecto, a menos que se indique lo contrario; es decir, que se indique como static o extern).
Las entidades de tipo static pueden ser locales (solo son "conocidas" por los bloques o las funciones en las que son declaradas) o globales (solo son conocidas en el archivo en el que se declaran [enlazado interno], posibilitando la existencia de entidades homónimas pero independientes en otros ficheros) (cabe destacar que en C++ las funciones han de tener mismo nombre, argumentos y tipo para ser consideradas iguales [polimorfismo]).
Como las variables estáticas se deben recordar (por la labor de la entidad que sea) durante toda la ejecución del programa, no se almacenan en la pila (*stack*), sino en zonas de memoria especiales: en el segmento de datos (.data) (si ya han sido inicializadas) o en el símbolo de inicio de bloque (.bss) (si no se han inicializado aún o están a cero). Para concluir, cuando se declaran los atributos o datos de una clase como static, se está indicando que estos han de ser compartidos por todos los objetos de la misma (una clase es una abstracción común de sus objetos [que son instancia de la clase], que a su vez encierran y comparten toda la información sobre sus atributos [variables o propiedades] y métodos [funciones]).

V. Máquina de estados

Draw the state diagram for a simple FSM on the whiteboard (a 4-way traffic signal is a good starting place).

En castellano: dibuja en la pizarra el diagrama de estados para una simple máquina de estados finita (una señal de tráfico de 4 vías es un buen punto de partida).
Se supone que la idea de la cuestión ronda sobre una especie de semáforo que regula el tráfico en una localización con 4 vías (intersección) para automóviles. Un sistema como este es fácilmente describible mediante una máquina de estados finitos (aparte de que es el único tipo que una computadora convencional [no cuántica] puede simular) porque abarca modelos en los que sus salidas dependen no solo de las entradas actuales, sino también del estado actual (derivado de las entradas anteriores) (es un sistema secuencial); puede haber muchas combinaciones, pero quedan empíricamente suplidas mediante la simplificadora interconexión y compartición de eventos entre todos los posibles estados.
Partiendo de que la respuesta a esta pregunta es bastante directa (que no necesariamente sencilla), puede resultar interesante secundar la comprensión del significado de una

máquina de estados (o autómata) y varios conceptos relacionados con ella.

Estos sistemas son útiles para realizar procesos claramente definidos dentro de una discretización de tiempos y acciones. Gracias a la computación digital es posible resolver (bajo un marco de restricciones) problemas de forma algorítmica, siempre y cuando estos vengan planteados mediante un conjunto de pasos (o estados) finitos.

Los autómatas pueden tener un estado inicial, dependiendo del tipo o la aplicación. Con cada entrada incipiente la máquina puede permanecer en el estado o cambiar de él (transición a estados finales). Según el ámbito de interés para su uso, las máquinas de estados se pueden clasificar en dos clases: aceptoras o transductoras (siendo estas últimas en realidad una variación de las primeras). Mientras que las aceptoras (generalmente prácticas en teoría de la computación [de autómatas] en la matemática) tratan de indicar si un conjunto de símbolos (representaciones de elementos) pertenecen o no a un lenguaje específico (conjunto arbitrario de cadenas de símbolos de un alfabeto [conjunto finito de símbolos]), las transductoras (de uso más general en la electrónica y computación digital) entregan como resultado (salida) un conjunto de símbolos que pertenecen al lenguaje que sirve como entrada.

Observando todas las posibilidades que nos proporciona la electrónica actual, una máquina de estados finitos (transductora) puede ser llevada a cabo mediante circuitos de función fija (con bloques digitales combinacionales y/o secuenciales MSI [integración a media escala]) (proceso sistemático para baja complejidad), mediante ASIC (circuitos integrados de aplicación específica), PLD (dispositivos lógicos programables, de organización matricial) o FPGA (matrices de puertas lógicas, con conjuntos de puertas configurables), con microcontroladores o microprocesadores (empleando lenguajes de programación concretos), con PLC (controladores lógicos programables) o computadoras. Es recomendable acudir a lenguajes de descripción hardware (HDL) como VHDL, AHDL o Verilog, para compilar diseños complejos partiendo de descripciones algorítmicas del proceso del autómata (como diagramas de estado). Otra alternativa a los diagramas son las tablas de fases, que resultan de mayor practicidad si se pretende realizar el diseño con bloques funcionales combinacionales y secuenciales.

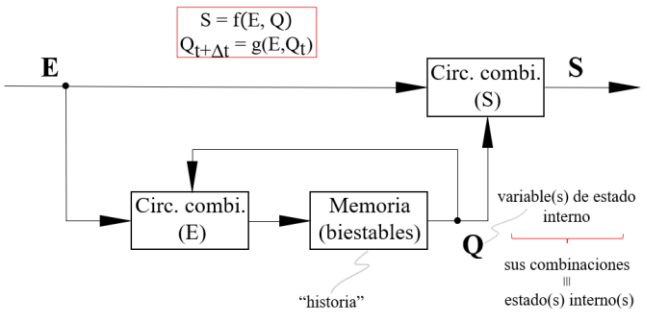

Fig. 7. Diagrama de bloques ejemplificador de las partes de una máquina de estados secuencial. Las salidas son función de las entradas y de las variables de estado; las variables de estados se "actualizan" en función de las entradas y de las propias variables de estado anteriores. El bloque "Memoria" representa la funcionalidad del autómata secuencial con biestables (ya sean síncronos o asíncronos), mientras que los otros bloques (circuitos) representan la lógica combinacional necesaria para la traducción algebraica (se divide en una entrada y una salida por practicidad general).

Una máquina de estados (en el marco de la electrónica digital) puede ser síncrona o asíncrona, según exista o no una señal de reloj concreta (CLK) que indique con alguno de sus flancos los cambios de estado interno. Los estados internos están formados por conjuntos de sus variables (que sirven para distinguir respuestas o estados distintos ante las mismas entradas), almacenadas en memoria para contener la "historia" del proceso; si la memoria la forman biestables asíncronos (entonces el estado interno puede variar si lo hacen las entradas) el autómata es un circuito secuencial asíncrono. La solución asíncrona es discutiblemente beneficiosa si el grado de complejidad del sistema aumenta demasiado (por acumulación y propagación de retardos físicos de respuesta de los componentes, retrasos en las señales...).

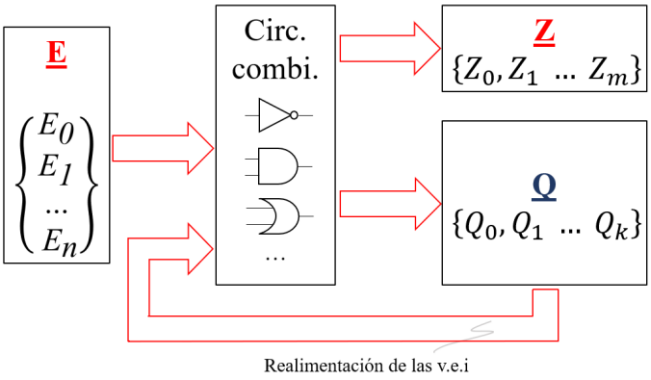

Fig. 8. Máquina de estados secuencial asíncrona. La etapa combinacional "combina" lógica digital usando las variables de entrada (E) y/o las de estado interno (Q). La realimentación asíncrona es causante de propagaciones de tiempos transitorios indeseados que pueden provocar respuestas fortuitas e inesperadas en la salida (Z).

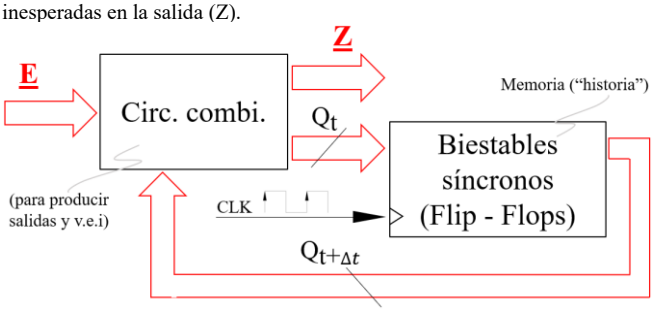

Fig. 9. Máquina de estados secuencial síncrona. La memoria del sistema (biestables capaces de almacenar 1 bit [0 ó 1 lógico] de información) se ve afectada a cambios internos solo con la llegada de flancos de reloj (tren de pulsos periódicos); visto de otro modo, los estados internos solo pueden cambiar ante flancos activos de la señal CLK en los biestables síncronos (JK, D...). Los flancos (el período) de CLK deben "esperar" a que el tiempo de propagación de todas las señales cese, para un correcto funcionamiento del autómata.

Por otra parte, la máquina es secuencial síncrona si los biestables (en este caso) utilizados son síncronos; los cambios del estado interno pasan a estar "gobernados" por la señal de reloj. Aunque surgen dos subdivisiones (modelos) más dentro de este tipo de autómatas: Máquinas de Moore (entregan una salida al llegar a cada estado [incluso desde que se entra al estado inicial]) y Máquinas de Mealy (entregan una salida en cada transición). En ambos modelos, estos estados internos están sincronizados por la señal CLK, pero solo en el segundo caso, las salidas no están ligadas solamente al flanco del reloj, sino que pueden cambiar directamente por modificaciones en la(s) entrada(s).

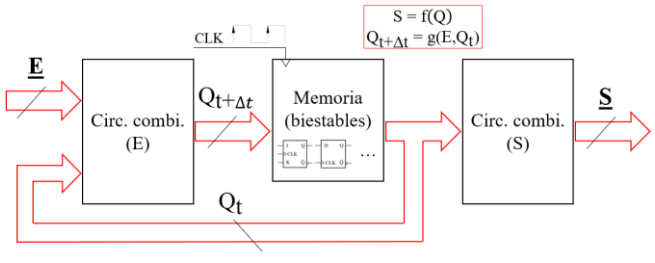

Fig. 10. Máquina de Moore. Las salidas (S) son función de las variables de estado interno (Q) y su variación está sincronizada con los flancos de la señal de reloj, que "actualiza" la memoria del autómata (biestables). El estado

interno evoluciona en consonancia a su historia anterior y al valor de las entradas. En el diagrama de estados de una máquina de Moore, mientras no cambie el estado (por tanto, no haya un flanco de reloj), no cambiará la salida.

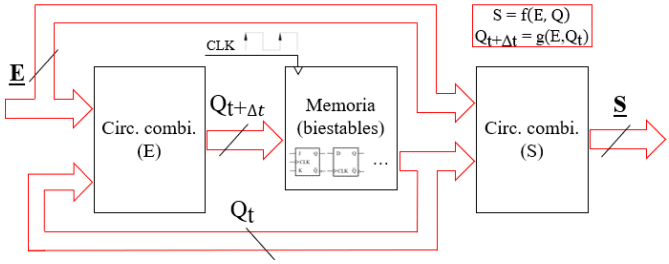

Fig. 11. Máquina de Mealy. Las salidas (S) son función de las variables de estado interno (Q) y de las entradas (E), su variación no es puramente síncrona con la señal de reloj, puede cambiar antes del flanco en esta. El estado interno evoluciona en consonancia a su historia anterior y al valor de las entradas (estas últimas pueden provocar cambios "instantáneos" [siempre hay retardos en un circuito] en las salidas). En el diagrama de estados de una máquina de Mealy, la salida dependerá del estado en el que se encuentre y del valor instantáneo de las entradas.

Retomando la raíz de la cuestión, se puede ilustrar el proceso sistemático para realizar un diagrama de estados (o una tabla de fases) para el diseño del sistema digital (secuencial) del autómata.
Primeramente, se puede empezar por definir el sistema, con la información necesaria y suficiente; esto es, estados, entradas y salidas. En la siguiente imagen se muestran las características a describir del sistema:

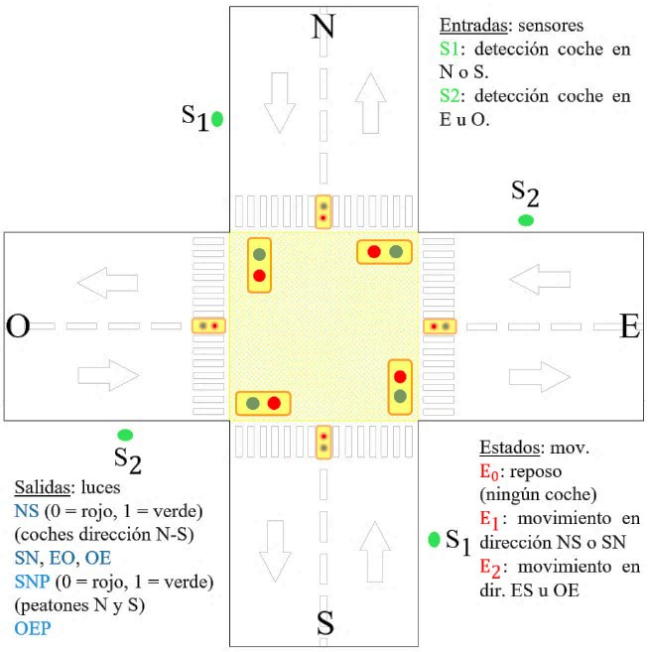

Fig. 12. Propuesta para el funcionamiento del semáforo de 4 vías imaginario. Se supone mov. solo en direcciones N-S (Norte-Sur), S-N, E-O y O-E. También hay varios pasos de peatones (salidas), que se encenderán (luz verde) cuando los pertinentes semáforos para coches estén apagados (rojo); "SNP" significa los semáforos para peatones del lado Norte y lado Sur. Los sensores (detector de presencia, sensor de movimiento) S1 y S2 captan si hay algún coche en las vías verticales (ambas 2) u horizontales.

La frecuencia de reloj del sistema se supone suficientemente alta para que no se detecte ningún cambio simultáneo en los sensores entre dos flancos activos. Además, la naturaleza pulsatoria escogida del sensor causará que tampoco sea posible el estado alto simultáneo en ambas señales (pero sí el bajo) (de no ser así, habría que implementar una ampliación del sistema con, por ejemplo, temporizadores e indicadores de preferencia). Este reloj es el encargado de sincronizar los cambios de estado interno de la máquina secuencial, por medio de la "actualización" de los biestables, como se ha explicado antes.
A continuación, se puede elaborar, directa y finalmente, la tabla de fases y el diagrama de estados (posibles soluciones) del autómata, teniendo en cuenta el funcionamiento secuencial del sistema y todas sus combinaciones viables.

Combinación de entradas: (S1S2)				Salidas:						Descripción de los estados:
00	01	10	11	NS	SN	EO	OE	SNP	OEP	
⓪	2	1	-	0	0	0	0	1	1	E_0: reposo (ningún coche)
0	-	①	-	1	1	0	0	0	1	E_1: mov. en dir. NS o SN
0	②	-	-	0	0	1	1	1	0	E_2: mov. en dir. ES u OE

ambas pueden agruparse en 2 pares: NS y EO — simplificación: $\overline{NS}$ — simplificación: $\overline{EO}$

Fig. 12. Tabla de fases del autómata. Representa todas las posibles combinaciones y cambios de estado que se dan en el proceso. Los estados rodeados con un círculo en la columna de las combinaciones de entrada representan "estados estables" (vuelven a sí mismos ante la combinación que se indica en su columna). Se ve que surgen, en el desarrollo de la tabla, una serie de posibles simplificaciones, indicadas abajo en la imagen. Las salidas pueden reducirse en número a solamente dos: NS y EO.

La tabla de fases sirve como apoyo para establecer los biestables y/o estados internos necesarios en un posible diseño o una plasmación del circuito que modela el comportamiento de la máquina. Con 3 estados (filas) en la tabla de fases, se escogerían 2 variables de estado interno y, por tanto, dos biestables (JK o D). Las v.e.i. pueden elegirse coincidentes con las propias salidas, para simplificar el diseño (las señales para los peatones son, simplemente, las salidas negadas).

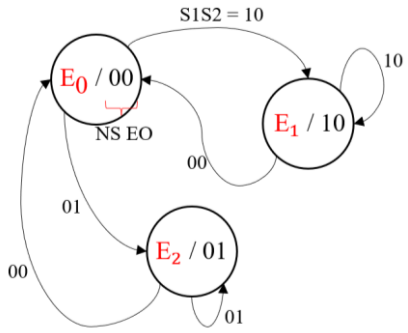

Fig. 13. Diagrama de estados del autómata propuesto. A los estados (círculos) los acompañan las salidas (NS y EO, en este caso) y en las transiciones se indican los valores de entrada que, con cada flanco activo de reloj, provocan el cambio de estado (diagrama de estados de Moore).

Cabe decir que el paso de Moore a Mealy y viceversa es sistemático, partiendo de la tabla de fases y de la manera en la que se simplifiquen sus líneas (tabla fusionada). En el primer caso, para que dos filas puedan ser fusionadas (cada línea de la tabla fusionada ha de tener distintas combinaciones de v.e.i.), es necesario que ambas presenten el mismo valor en las salidas. Esta condición de igualdad no es necesaria en el segundo caso ya que, en los autómatas de Mealy, las salidas dependen también de las entradas (estas pueden servir para diferenciar las salidas, sin falta de las v.e.i.).

VI. Latencia

What is interrupt latency and why is it important?

En castellano: ¿qué es la latencia de interrupción y por qué es importante?
Una medida deliberada y segura que llevan a cabo los desarrolladores de sistemas operativos es establecer una capa

separadora entre dos zonas divisorias y fundamentales del SO (de su memoria física), que son el espacio del usuario (procesos o programas de aplicación) y el espacio del núcleo o kernel. Con esta diferenciación se consigue bloquear la entrada directa al sistema a procesos que no pertenecen o no provienen del mismo; se ha de recurrir a las "llamadas del sistema" (*system calls*) para la comunicación entre espacios, que generalmente se ocupan del acceso a recursos de información del hardware o del propio sistema (CPU, memorias internas, disco duro [HDD]...). Entre otras, las principales "motivaciones" de un sistema operativo (y por tanto los propósitos de las llamadas del sistema) son: la gestión de dispositivos (administrar, solicitar y manejar recursos hardware, como se comentó antes), gestión de archivos (para el control del almacenamiento secundario compartido por los usuarios; operaciones *create*, *delete*, *open*, *close*, *write* y *read*), control de procesos (inicio, ejecución, finalización..., de procesos informáticos y supervisión de la yuxtaposición de estos), servicio de comunicaciones (coordinación de los procesos del modo privilegiado [núcleo] y el usuario) y gestión de memoria (intercambio y solicitud de información de forma organizada y prioritaria, mediante llamadas).

Se puede decir que el principal cometido de un sistema operativo (acotando en especificaciones) de tiempo real (pensado para interactuar y dar un correcto funcionamiento en un entorno dinámico con actividad conocida respecto a entradas, salidas y restricciones temporales) es proporcionar al usuario un servicio plausible y adecuado a las condiciones, ajustándose, de suerte, a un intervalo de tiempo mínimo u óptimo requerido por las necesidades del proceso. Para poder cuantificar o valorar este aspecto, se pueden usar dos medidas temporales distintas: el "tiempo de latencia" (o latencia) y la "fluctuación de retardo" (o *jitter*).

El jitter representa la causa de una deformación de alguna señal digital; es la variación temporal en el envío cuasi-periódico de información (mide la estabilidad de su frecuencia) (paquetes de bytes por internet...) como efecto de una desincronización acumulativa con una señal de reloj.

Ya se ha comentado en párrafos anteriores aspectos generales, características y aplicaciones de las interrupciones (servicios de interrupción hardware o software y excepciones [causadas por la propia CPU]). De este modo, se puede apreciar que, para convertir un SO de propósito general en uno de tiempo real, es importante un buen manejo de las interrupciones. En este sentido, la latencia está fuertemente ligada a tales eventos. Esta se define como el tiempo que transcurre desde que se "dispara" un evento externo hasta la inminente ejecución del código correspondiente a la interrupción.

Paralelamente, resulta práctico mencionar que un SORT (Sistema Operativo en Tiempo Real) debe reconocer y atender rápida y debidamente cualquier evento, por tanto, proporcionar una correcta respuesta funcional y temporal. Además, el SO debe ser, obviamente, reentrante (concepto explicado en entradas anteriores) (sistemas como el antiguo DOS [Disk Operating System] no son reentrantes y, por tanto, no son de tiempo real).

La latencia caracteriza verosímilmente a las prestaciones del sistema; si se habla de "latencia de interrupción", esta tiene exactamente la definición citada con anterioridad. Se puede derivar de retrasos por acceso a buses de comunicación (con celdas de memoria..., externas al procesador), reactivación de permisos de interrupciones temporalmente inoperativas, o mismamente, acceso a algún *buffer* (espacios de memoria temporales, generalmente usando sistemas de cola FIFO [First In First Out] para leerlos secuencialmente) o registro del programa.

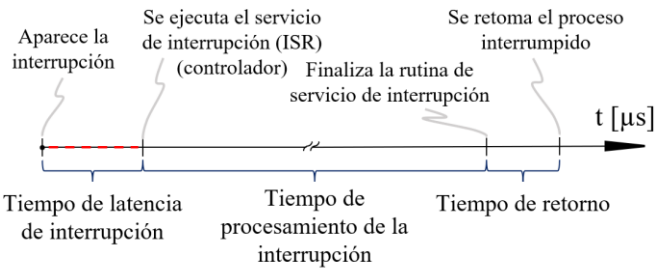

Fig. 14. Esquema cronográfico que sitúa el tiempo de latencia de interrupción y algunos de los pasos que se dan (de forma general) cuando aparece un proceso de este tipo.

Se han implementado a lo largo del tiempo varias modificaciones a estructuras kernel estándar para optimizar la escalabilidad y, sobre todo, el manejo temporal de la interrupción. Para el kernel de Linux, se introdujo a partir de la versión 2.6, en el año 2004, una mejora que consistía en la atención prioritaria (*preemption*) de tareas que se podían considerar de mayor preferencia. Así, los procesos del núcleo pasaban a ejecutarse antes que otros menos "importantes" y se evitaban confrontaciones dentro del manejador (o controlador) de interrupciones, en caso de que un evento prevaleciera respecto a otro.

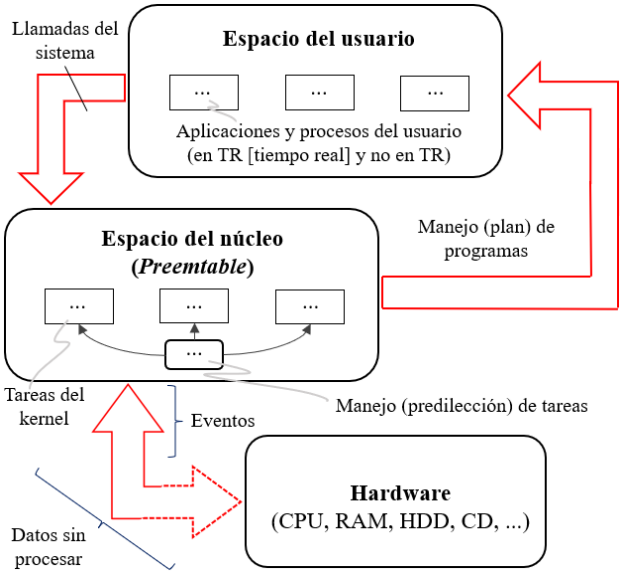

Fig. 15. Diagrama que esquematiza el comportamiento y la comunicación entre algunas de las partes de un sistema operativo (en tiempo real) y la zona hardware. En este caso se particulariza para una topología (kernel) de Linux.

Cabe destacar que existe otra posibilidad interesante que se puede utilizar con el mismo propósito que la anterior; es el "microkernel", que sirve como interfaz entre el hardware y el kernel original para ser el que se ocupe de las tareas en tiempo real cuando no haya de otro tipo. Es, realmente, una especie de HAL (capa de abstracción de hardware) cuyo acceso (no al del núcleo estándar) es otorgado a las interrupciones, que se anteponen y restan prioridad a otras tareas.

Se puede concluir en que ambas variables mesurables (latencia y jitter) deben ser minimizadas, a fin de mejorar las prestaciones del sistema y la interfaz del usuario.

VII. Secciones críticas I

What's the difference between a semaphore and a mutex and when would you use them?

En castellano: ¿cuál es la diferencia entre un "semáforo" y un *mutex* y cuándo los usarías?

Puede pasarse a responder de una forma bastante directa esta cuestión, sin dar lugar a claras confusiones; el semáforo y el "mutex" (exclusión mutua) son soluciones (por software) viables cuando se quiere tratar una "sección crítica" dentro de un programa. Por definición, una sección o región crítica es toda parte del código en la que se accede a algún recurso compartido (estructuras de datos, contenido de funciones, clases de objetos...), pero no por más de un proceso a la vez (por tanto, no por más de un hilo en ejecución). Un tema similar se ha tratado anteriormente durante la primera cuestión de este documento (se explicó someramente el mutex), aunque en este caso se particulariza para diferenciar dos mecanismos del mismo tipo, cuya función es hacer concordar, de algún modo, la entrada y salida de los procesos a los datos compartidos cuando sea requerido.

El primer "mecanismo" (el más común) es el mutex; se trata de un objeto (puede constar de un estado, un comportamiento, de datos almacenados y tareas ejecutables) cuyo propósito es otorgar el acceso a un recurso a un solo [sub]proceso al mismo tiempo. En cierto modo, del nombre "exclusión mutua" se puede inferir el funcionamiento del mecanismo. La técnica con la que se consigue esto es un mecanismo de bloqueo. Cuando un proceso solicita el acceso a un recurso (que puede ser compartido [no usado simultáneamente] por otros procesos), se crea un objeto asociado a ambos (un mutex) que, mediante operaciones de bloqueo y desbloqueo del acceso al recurso, logra la sincronización de procesos en la sección crítica correspondiente. Si aparece un solicitante cuando un recurso compartido está bloqueado, se añade dicho proceso a la cola y se mantiene en espera hasta que este recurso quede liberado.

El segundo tipo se trata del semáforo; se puede decir que se asemeja más a una variable que a un objeto, es una variable (positiva entera) que es compartida por diferentes hilos o procesos. El valor al que se inicializa (generalmente) esta variable se puede llamar "S" y se corresponde al número total de recursos compartidos (disponibles) en el sistema. Cada proceso puede requerir del acceso a varios recursos en su ejecución, pero el semáforo ha de sincronizar todos ellos para que solo un proceso acceda a un recurso de cada vez; para esto, se hace uso de dos funciones "atómicas" (una función es atómica si su resultado puede ser visto por otro hilo solo al final de su ejecución, convirtiéndose en un conjunto de instrucciones indivisibles): *espera()* y *señal()*. Haciendo que las funciones sean de esta forma se consigue que el valor del semáforo no pueda ser modificado por más de un proceso simultáneamente. Por otra parte, se puede decir que existen dos tipos de semáforo: de "conteo" y "binarios".

La variable "S" en los semáforos de conteo se inicializa, como se ha comentado antes, al número total de recursos disponibles del sistema. Si un proceso quiere acceder a un recurso, se realiza una operación espera() y el contador de S disminuye su valor en uno, quedando ese recurso no disponible para otros procesos. En caso de que un recurso quede liberado por su solicitante, se realiza una operación señal() y el contador aumenta en uno. Si este llega a alcanzar el valor de su inicialización entonces significa que todos los recursos vuelven a estar disponibles; por el contrario, si llega a cero y un proceso solicita un acceso, se realiza una operación espera() y queda bloqueado hasta que el contador sea mayor que cero (hasta que vuelva a aparecer un recurso disponible; funcionamiento similar al de un semáforo de tráfico convencional).

En los semáforos binarios, S puede tomar los valores 0 ó 1 e inicialmente es igual a 1 (no necesariamente ha de haber un solo recurso compartido). Si algún proceso quiere acceder a algún recurso, se realiza en el semáforo una operación espera() y el valor de S disminuye a 0. En ese momento, hasta que el recurso quede liberado, ningún otro proceso (distinto al solicitante) puede tener acceso al mismo (tampoco a otros). Cuando un recurso queda liberado, se realiza una operación señal() y S pasa a valer 1.

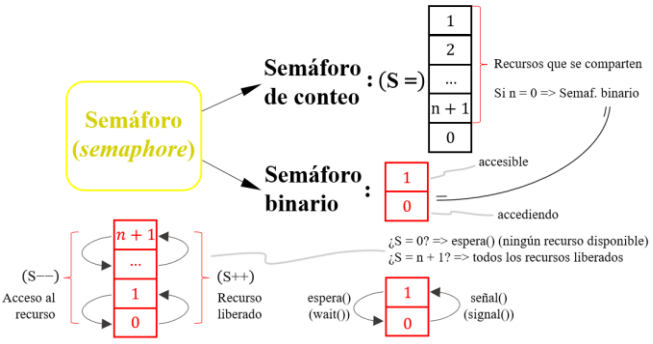

Fig. 16. Esquema gráfico que clarifica los conceptos explicados en el párrafo anterior sobre el semáforo y sus dos tipos. Cabe destacar que no hace falta que únicamente haya un solo recurso compartido en el caso del semáforo binario, pero se muestra la relación existente.

Hasta ahora puede concluirse, de manera superficial, en que la principal diferencia entre estos es la manera de sincronizar el acceso a los recursos compartidos; mientras el semáforo emite dos señales indicadoras "espera" y "señal" para encaminar al proceso, el mutex trata de bloquear directamente el flujo de información, consiguiendo que el proceso deba solicitar el acceso solamente cuando el "candado" se encuentre desbloqueado. Únicamente de la primera forma puede haber múltiples hilos de programa simultáneos accediendo a la sección crítica. Mientras que usando un semáforo se pueden dar situaciones inoportunas como tiempos de espera demasiado largos, usando un mutex pueden generarse colas de procesos "de espera" excesivamente grandes, así como solo puede ser desbloqueado por el mismo proceso que generó su contexto. También puede darse el caso (conocido como *deadlock* [punto muerto]) en que dos procesos se queden ambos a la espera de que el otro libere el semáforo.

Dos ejemplos de aplicación típica de ambos son los siguientes: si se quiere dividir un buffer en varias partes (separar tantos MB en otros tantos KB...) para acceder a cada una de esas subdivisiones separadamente, se puede hacer uso de un semáforo, asociando ciertos procesos o hilos a cada buffer, individual pero simultáneamente. Si se quiere, por otra parte, que unos datos consistentes se vayan almacenando y actualizando en un único buffer (información monitorizada y compartida a varios usuarios...), es preferible usar un mutex; a los recursos accede un solo hilo de programa a la vez.

VIII. Secciones críticas II

What is a critical section and how would you implement one?

En castellano: ¿qué es una sección crítica y cómo implementarías una?
De forma muy similar a lo comentado al inicio de la anterior cuestión, ya se han explicado bastantes conceptos relevantes que, de no haberlo hecho allí, se tendrían que tratar en este apartado.
Para recordar definiciones, una sección (o región) crítica es aquella parte o zona de un código (conjunto de instrucciones) en la que se da acceso a algún recurso compartido (ya sean estructuras de datos, contenido de funciones, dispositivos, etc.), pero no por más de un proceso o hilo en ejecución al mismo tiempo (exclusión mutua). Como las secciones críticas están definidas espacial y temporalmente, pueden desarrollarse mecanismos que sincronicen el paso por ellas, como el semáforo y el mutex, anteriormente explicados. También existen otros métodos, llevados a cabo mediante hardware (candados o cerrojos, deshabilitar interrupciones...) o software (mensajes, monitores tipo Hoare [descritos por Charles Antony Richard Hoare en 1974] o tipo Mesa [por Butler W. Lampson y David D. Redell en 1980], *rendevous* en el lenguaje orientado a objetos ADA...). Los monitores de Hoare tienen la ventaja frente a los semáforos (por ejemplo) de privar al código de contener dentro suyo las instrucciones literales para su implementación; por el contrario, se conforman de una estructura de datos abstracta compuesta por una inicialización (instrucciones ejecutables cuando se crea el monitor), unos datos privados (procedimientos conocidos solo por ellos), unos métodos públicos (procedimientos que pueden ser llamados desde fuera de ellos) y una cola de entrada (hilos cuya solicitud de manejo de los métodos públicos está aún pendiente de ser aceptada).
Pueden implementarse deliberadamente secciones críticas en aplicaciones en las que concurran varias tareas (en programas multihilo...) que se ocupen, por ejemplo, de modificar de forma “cruda” o directa cierto conjunto de variables correspondientes y pertenecientes a los recursos compartidos. Las regiones críticas también pueden ser útiles en impresoras, procesos de lectura/escritura concurrentes, bases de datos, periféricos o buses para comunicaciones bidireccionales...; se quiere asegurar que estos recursos o aplicaciones puedan ser accedidos únicamente por un proceso al mismo tiempo.

IX. Análisis y visualización de señales

Have you used an oscilloscope? A logic analyser? Give me examples of how you've used them.

En castellano: ¿has usado un osciloscopio? ¿Un analizador lógico? Dame ejemplos de cómo los has usado.
Para abordar esta interesante cuestión se hará un recorrido en el cual estarán camufladas sus respuestas concretas y comunes, pero, también se completará con escenas históricas, funcionalidades, aplicaciones y otros aspectos que resultarán de valiosa calidad para cultivar múltiples ideas en torno a estos dos conceptos apuntados en la pregunta.
En el marco actual, cuando se habla de un "osciloscopio" se está refiriendo a un instrumento electrónico de prueba (que somete a pruebas a otros dispositivos [a los DTU]) que logra capturar señales en forma de tensión o voltaje y mostrarlas, por lo general, gráficamente en una escala bidimensional temporal. Es un equipo de pruebas porque gracias a él se pueden llevar a cabo importantes análisis, calibraciones o el control de ondas y señales, mediante sus características intrínsecas y extrínsecas como la amplitud, frecuencia, fase, distorsión armónica, ruido, etc. Es una herramienta muy potente cuando se quiere extraer información de alguna señal impredecible, repetitiva o rápida, ya que permite "parar" la imagen continuamente para poder visualizarla como si tratase de una imagen en detención con estaticidad virtual.
La evolución de este dispositivo es extensa; partiendo de los primeros prototipos con galvanómetros o tubos de rayos catódicos íntegramente para la visualización de las señales y aparte amplificadores lineales para su procesamiento, se ha pasado por otros tipos, de almacenamiento digital (indudable protagonismo e inclusión de la tecnología del silicio) usando convertidores A/D, D/A (“analógico-digital” y viceversa) y procesamiento de señales e, incluso, algunos sin pantalla incorporada para aprovechar la del propio computador de propósito general como monitor de visualización.
Como idea general, es importante recalcar que existen y se diferencian dos grandes familias de osciloscopios: analógicos y digitales. A grandes rasgos: mientras los analógicos trabajan directamente con la señal analizada, que se amplifica (o atenúa) y se usa para desviar verticalmente un haz electrónico de manera proporcional al valor de la señal, los digitales utilizan conversores A/D para almacenar en memoria el valor de la señal entrante y plasmar luego la información gráficamente en la pantalla.
Los osciloscopios se basan en medidas. El "arte" de la medición se remonta a siglos atrás; alrededor de 1780 Luigi Galvani usaba galvanómetros de rana (ataba terminales eléctricos a nervios de la pata del animal) con los que lograba mensurar (o más bien detectar) tensiones eléctricas. Un descubrimiento esencial (el del electromagnetismo) vino de mano del profesor danés Hans Christian Ørsted sobre 1820; en una de sus clases se sorprendió al ver cómo giraba la aguja de una brújula al acercarla a dos cables que unían los polos de una pila Volta, que es la precursora de las actuales baterías, hecha con conductores (plata y zinc, cobre y estaño...), dispuestos alternadamente y separados por capas de cartón, cuero..., empapadas en algún fluido preferiblemente mejor conductor que el agua pura (como el agua salada, salmuera, lejía...). Las descripciones matemáticas de este fenómeno por André-Marie Ampère, iniciadas en la época, también son principales, mismamente para la determinación indirecta de la corriente circulante por una bobina.
A su vez, el concepto de aplicación de un osciloscopio (visualizar una forma de onda para su análisis "a posteriori") se remonta a la segunda mitad del siglo XIX con el uso (en contraposición) del galvanómetro y el dibujo manual.
El galvanómetro es un aparato electromecánico de medición de corriente eléctrica mediante un indicador deformable calibrado, basado en fuerzas magnéticas inducidas (Ampère le dio este nombre en honor al galvanoscopio de Luigi Galvani). La idea era ir anotando manualmente mediciones de corriente (o tensión) realizadas con el galvanómetro alrededor del eje de un rotor en rotación; con esto se obtenía, puntualmente, una representación estacionaria de una onda.
Posteriormente, el científico francés Jules François Joubert modificó ligeramente el proceso y también dio uno de los primeros pasos en la plasmación de formas de onda; lo automatizó parcialmente (con un mecanismo de dibujo acoplado, conformado por un conmutador y un contacto movibles), convirtiendo aquel método arcaico y tedioso de registro en uno más práctico y que lograba acercarse más a

las filosofías actuales del osciloscopio (seguía siendo, aun así, impreciso).

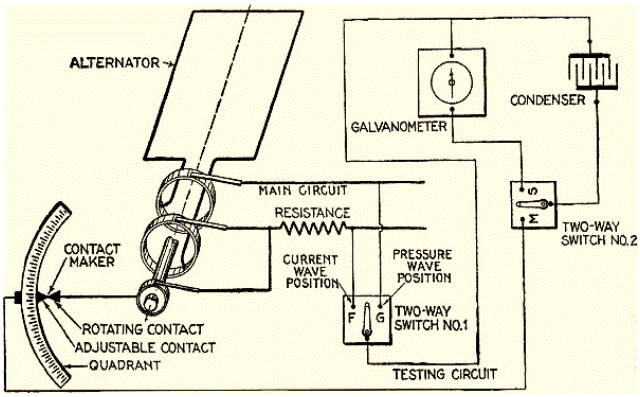

Fig. 17. Esquema adaptado de *Hawkins Electrical Guide* nº 8, 1915. El método "paso a paso" de Joubert consiste en un galvanómetro, un condensador, dos interruptores de dos vías, una resistencia y dos contactos. Uno de estos últimos va unido al eje de un alternador, con el que rota sincronizadamente; el otro se ajusta en una escala angular graduada; al juntarse ambos se cierra el circuito. Para medir la caída de tensión en la resistencia, los interruptores 1 y 2 van en posiciones F y M (respectivamente) (la posición G para la medida de tensión ["presión", antiguamente]) para que el condensador se cargue; luego se descarga hacia el galvanómetro cambiando el segundo interruptor a la posición S. Repitiendo el proceso varias veces y graficando el ángulo del contacto ajustable frente a la medida del galvanómetro, puede representarse la forma de onda.

Aparte de este dispositivo, otros muchos aparatos e inventos son causa directa del desarrollo progresivo del osciloscopio (de almacenamiento) digital moderno, tales como el "tubo de rayos catódicos", el "oscilógrafo" y los "osciloscopios analógicos".

Es conceptualmente interesante nombrar e introducir en la historia a los rayos catódicos (totalmente equivalentes a los haces de electrones). Estos son, por definición, corrientes electrónicas que se observan en tubos de vacío (originalmente se notaron en un tubo de descarga [vacío parcial] ideado por William Crooke y otros [Julius Plücker, Johann Wilhelm Hittorf...] en 1869-1875). Con este dispositivo se controla el flujo de corriente eléctrica (en un medio de alto vacío) al aplicar una diferencia de potencial a dos electrodos. Dentro de esta clase, los tubos "termoiónicos" se basan en el principio de la emisión termoiónica (flujo de iones incitado por la agitación térmica en la superficie de un metal) con electrones. Por otra parte, los tubos no termoiónicos (fototubos, algunos transductores...) utilizan el efecto fotoeléctrico para emitir electrones. En ambos casos, esta emisión se da desde el cátodo (puede estar caliente, en el primer caso) hacia el ánodo (de ahí el nombre de rayos "catódicos"), como consecuencia del campo eléctrico aplicado al tubo. Si bien los primeros pueden usarse, típicamente, para la amplificación y rectificación, los segundos son útiles para la detección de intensidades luminosas.

Para relatarlo de otra forma; con los tubos de Crooke se usaba un método de "cátodo frío" para ionizar las partículas de gas (residual) con un alto voltaje de entrada (de cientos a varios miles de voltios). En 1897, Karl Ferdinand Braun inventó el tubo de rayos catódicos (TRC) (más bien su primera versión, el "tubo de Braun"), que consistía en una modificación del tubo de Crooke, al cual le añadía una pantalla de vidrio recubierta de plomo (para proteger al usuario de la radiación electromagnética en forma de rayos X) y fósforo (que se ilumina al entrar en contacto con los electrones, proporcionando información visual instantánea); de esta forma, logró crear una (primera) concepción de tal tecnología como aparatos de visualización de señales eléctricas y electrónicas.

Se puede comentar cómo funcionaba la mayor aproximación de este tipo al osciloscopio: un cañón de electrones emitía un haz de rayos catódicos colimados (paralelos entre sí) desviados mediante campos eléctricos transversales a ellos a través de placas horizontales (deflexión horizontal, asimilable a la escala temporal del visor de un osciloscopio) y verticales (deflexión vertical, asimilable a la amplitud o escala vertical) alrededor del haz.

Años más tarde, se descubrió precisamente la emisión termoiónica: en 1904, John Ambrose Fleming reemplazó aquellos tubos por otros con los que se hacía uso de un método de cátodo caliente (calentar al "rojo vivo" el electrodo mediante una corriente separada para motivar la emisión de electrones). En 1906, Lee De Forest añadió a este invento una rejilla entre los terminales a la que se aplicaba un pequeño voltaje para regular la corriente de rayos catódicos, inventando así el triodo (aunque posteriormente el tubo de vacío y sus variantes quedarían "sepultados" por el transistor).

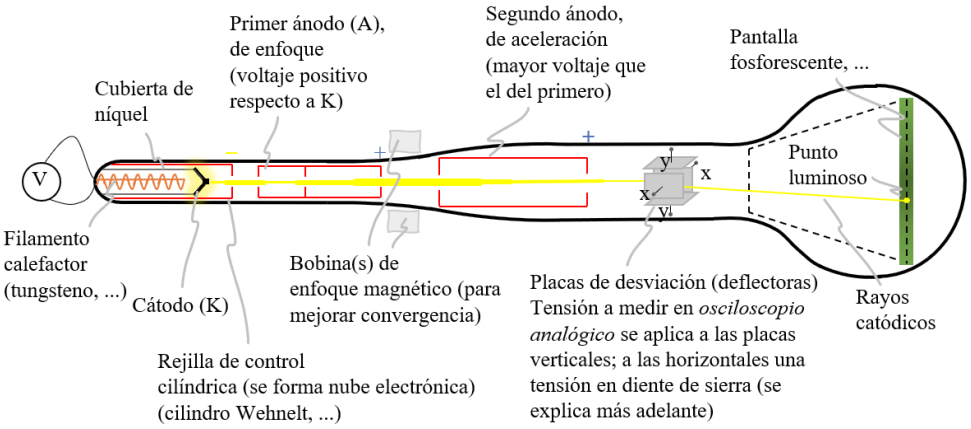

Fig. 18. Esquema que muestra una configuración típica del tubo de rayos catódicos. La tensión variable aplicada en las placas paralelas (tanto en las verticales como en las horizontales) permite obtener en la pantalla de fósforo una variación de puntos de luz sincronizada (de suerte) con ese cambio de voltaje. La rejilla de control ayuda a focalizar el haz cuando se dispara por el cañón de electrones. En el tubo de Braun original no había filamento, ya que era una modificación del de Crooke, en el que todavía se ionizaba el gas residual contenido, aplicando un alto voltaje entre A (un ánodo) (que además tenía forma cilíndrica y hueca) y K (un cátodo).

Sobre 1902, Édouard Hospitalier inventó lo que puede considerarse el primer aparato generador de formas de onda completamente automatizado, el oscilógrafo (u ondógrafo) de Hospitalier. Mediante la reacción a mediciones eventuales de corriente con un galvanómetro, se dibujaba (con un bolígrafo...) características de la forma de onda, que quedaban grabadas en un rollo de papel giratorio. Un capacitor podía "recoger" la energía en forma de carga eléctrica cada cierto número de ciclos de onda (aunque las frecuencias típicas de la señal son ciertamente incompatibles con la capacidad mecánica de reacción de los actuadores) y, tras repetidas (bastantes) descargas en el galvanómetro, se podía extraer un promediado de la onda formada.

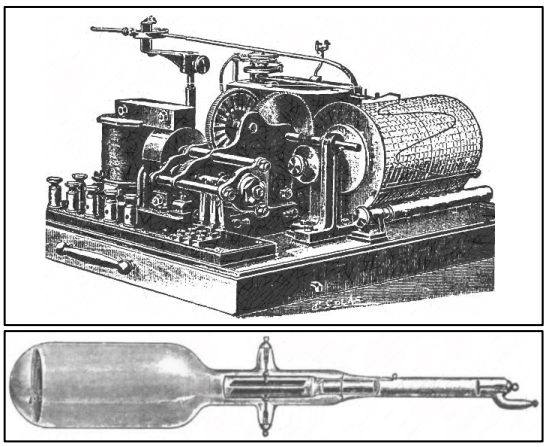

Fig. 19. Arriba, el ondógrafo de Hospitalier. Se ve cómo esta modificación del método de Joubert consta de un pivote actuado por electroimanes, unido a un bolígrafo que va dibujando la forma de onda en un tambor de revolución. Más abajo, el tubo (de cátodo frío) de Braun original, datado en 1897. De derecha a izquierda, se diferencia el cátodo (de platino), el ánodo, dos platos deflectantes y la pantalla fosforescente. Recogidas de *Hawkins Electrical Guide* nº 8, 1915 y *Handbuch der drahtlosen Telegraphie und Telephonie,* vol. 1, 1921, respectivamente.

Poco más tarde, William Du Bois Duddell solventó (en cierta medida) el problema de la imprecisión en tiempo real que ocasionaba el anterior aparato, haciendo uso de un espejo rotatorio, sensible (realmente no él directamente, sino su entorno) a campos magnéticos inducidos por corrientes (las que se querían medir); el espejo giraba en función de las características de esa corriente ("fuerza" o intensidad y dirección) y así, proyectando un haz de luz contra una pantalla y haciéndola pasar a través del espejo o grabando la forma de onda, se podía entrever la misma. Esta solución mejoró la problemática comentada de las altas frecuencias, pero aún requería de un posprocesado aparatoso.

En 1931, Vladimir K. Zworykin (inventor del iconoscopio [predecesor de las cámaras de televisión] y del quinetoscopio [predecesor de los proyectores de películas]) ideó un tubo de rayos catódicos con emisor termoiónico que serviría como componente básico para que, posteriormente, la compañía estadounidense "General Radio" introdujera el osciloscopio como un equipo de instrumentación con cierta independencia del entorno de aplicación.

Desde el año 1932, la empresa británica *A.C. Cossor* (más tarde adquirida por *Raytheon*) desarrolló este concepto y potenció el vínculo del TRC (o CRT en inglés) con las aplicaciones militares, por ejemplo, con el equipamiento radar. Otra compañía, *DuMont Laboratories*, comercializó y diseñó, ya en 1939, osciloscopios de "disparo y barrido" (*trigger and sweep*); esta característica permite visualizar señales en la pantalla estáticamente, pudiendo seleccionar un punto concreto (nivel de tensión) de la onda, desde el que se referencia constantemente la base de tiempos.

Aquellos países que habían propuesto una sólida investigación en la electrónica militar, tenían, al acabar la Segunda Guerra Mundial, cierta ventaja en este campo, que pudieron canalizar en buscar otras soluciones más sanas y comerciales; por ejemplo, se fundó en 1946 la empresa *Tektronix*, que se convertiría en el primer fabricante de osciloscopios calibrados (que disponían de una retícula y ejes en pantalla para la medición y producían gráficos con escalas calibradas), así como desarrolló otros de trazas múltiples y variaciones de los fabricados con tubos de vacío. Uno de los primero modelos más relevantes fue el "Tektronix Type 511".

Los años subsiguientes fueron prolíficos para las compañías que investigaban en este entorno; *LeCroy Corporation* (*LeCroy*) (fundada en 1964), *Hewlett-Packard* (HP) (fundada en 1935) o *Nicolet Instrument Corporation* (*Nicolet Test Instrument*, 1966) son solo algunas, las cuales consiguieron alcanzar la comercialización y el desarrollo de osciloscopios de almacenamiento digital (o de sus predecesores) como los que se usan hoy en día (y que prevalecieron en el mercado a partir de entonces). Aunque su invención puede ser atribuida a Hiro Moriyasu, de Tektronix, su manufacturación se corresponde a LeCroy. HP lanzaría en 1982 el primer osciloscopio digital basado en microprocesadores (MPU, µP), el HP 1980A/B. Nicolet Test Instrument también fue pionera con su DSO de 1 MHz (frecuencia de muestreo del ADC o conversor A/D).

De aquí en adelante, la evolución de los osciloscopios fue impresionante y continuamente creciente, gracias a las mejoras en los anchos de banda (rango de frecuencias que pueden mostrar "de manera útil") (desde los 500 kHz del HP 1200 en 1969 a los 1.5 GHz de los HP 54800 *Series Infinium* en 1998...), los bits de resolución del conversor A/D (osciloscopio digital) (de los 8 bits del *LeCroy Model* 9400 en 1985 a los 16 bits del *PicoScope* 5000 en 2013...), los canales de entrada (hasta 8 canales analógicos [DLM4000 de *Yokogawa* en 2012...] y 16 digitales [MSO70000 de Tektronix en 2009...]), la eficiencia energética, el tamaño, la calidad en pantalla, etc.

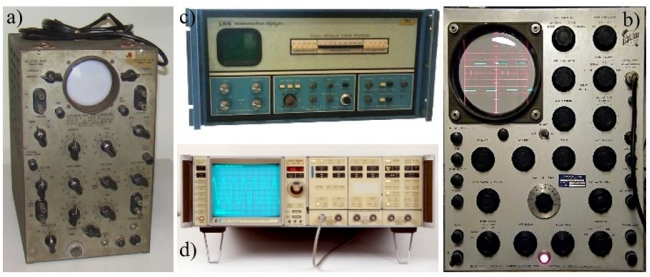

Fig. 20. Varios modelos de osciloscopios relevantes para su historia: a) Model 224-A de DuMont (con un TRC de 3 pulgadas de diámetro, un canal y ancho de banda de 2 MHz). b) Tektronix Type 511 de Tektronix (que introduce el disparo automático [*triggering*] según la especificación del usuario). c) WD 2000 de LeCroy, primer osciloscopio digital de tiempo real, de 20 muestras de profundidad de memoria (cuántos puntos como máximo se registran o se memorizan, digitalmente, en un tiempo de adquisición) y período de muestreo de 1 ns. d) HP 1980A/B de Hewlett-Packard (basado en MPU, con dos canales analógicos de 100 MHz).

Resulta vital la comprensión de las diferencias entre los osciloscopios digitales y analógicos y el funcionamiento de ambos (con todas las ideas históricas comentadas anteriormente se logra tomar una buena consciencia de lo que los dos abarcan). Dado que el funcionamiento de uno digital se puede comprender más sencillamente si se tiene una visión general de cómo funciona uno analógico, se procederá a caracterizar este último. La Fig. 21, más adelante, muestra de forma muy clara, mediante un diagrama de bloques, la estructura mínima y típica de uno. Aunque los conocimientos recomendados antes de pasar al digital puedan cubrirse con tal figura, hay una serie de datos que merece la pena mencionar primero.

En la etapa inicial del osciloscopio hay un amplificador (o atenuador) de la señal proveniente de la sonda, cuya ganancia varía a petición del usuario, dependiendo de la naturaleza de la propia señal y sus posibles dimensiones. En función del signo de la señal de entrada, las placas deflectoras verticales antes mencionadas pueden mover el haz de electrones hacia arriba (si la señal es positiva) o hacia abajo (si la señal es negativa); en ambos casos el movimiento es efecto del campo eléctrico causado por la tensión de entrada, que se dirige "directamente" a las placas.

La frecuencia de una señal con forma de diente de sierra (base de tiempos) sintetizada con osciladores (generalmente) es ajustable y, precisamente, es lo que tiene que regular el usuario (además de otras cosas) para adaptar la escala de tiempos a la frecuencia particular de la señal a medir; es decir, para una visualización correcta de la onda.
Durante la subida "más lenta" de la base de tiempos, si no hubiese ninguna desviación en las placas verticales, se dibujaría un línea horizontal (sea, de izquierda a derecha) en la pantalla del tubo de rayos catódicos, tras lo cual, durante un tiempo mucho menor (bajada del diente de sierra, casi instantánea) e imperceptible para el ojo humano, el punto luminoso retornaría al inicio, donde se encontraba justo antes de dibujarse la línea (es decir, volvería a la izquierda). El nivel de tensión por el que comienza a muestrearse la señal con cada barrido (por la izquierda, en este ejemplo) es el que se ajusta (o se puede ajustar) con la opción de disparo.
Las divisiones en pantalla del osciloscopio permiten deducir, de forma directa y visual, mediciones temporales y de amplitud (voltaje), relacionándolas con el período del diente de sierra y con la ganancia de amplificación vertical.

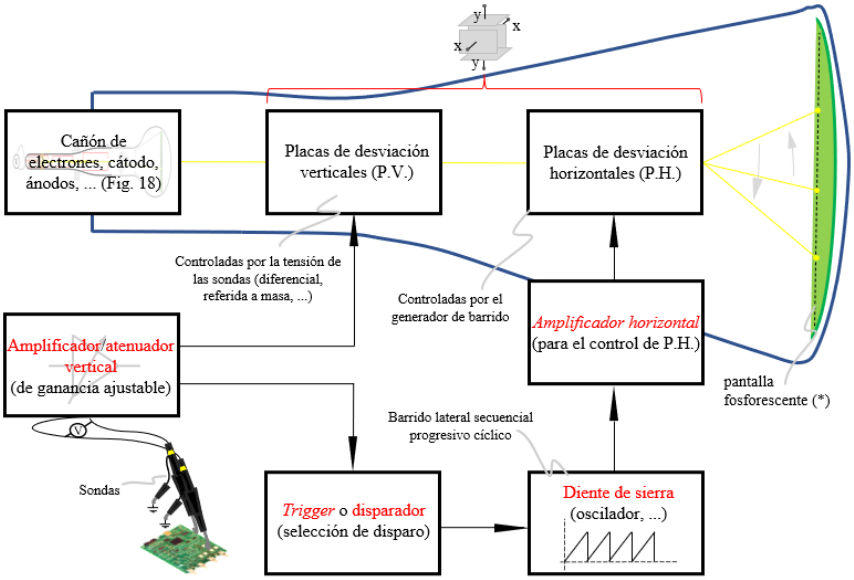

Fig. 21. Diagrama explicativo del funcionamiento y la estructura de un osciloscopio analógico. El *trigger* permite capturar estáticamente la forma de onda; se puede seleccionar un punto concreto (nivel de tensión) de la señal para que comience "ahí" cada disparo (barrido de la base de tiempos).
La señal en diente de sierra (generador de barrido) controla indirectamente (hace falta un acondicionamiento variable de tensión) las placas deflectoras horizontales (recordar que el campo eléctrico en estas [y en las verticales] desvía el haz de rayos catódicos). (*): la siguiente figura (Fig. 22) es una posible vista frontal de esta pantalla fosforescente o fluorescente.

Los mecanismos de barrido vertical y horizontal (aunque realmente "barrido" es solo el horizontal) trabajando conjuntamente permiten la recreación de formas de onda (periódicas) en la etapa final del tubo de rayos catódicos perteneciente al osciloscopio.

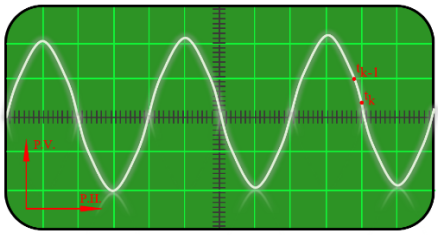

Fig. 22. Representación de lo que puede ser la vista frontal de la pantalla (relativamente antigua, fosforescente) de un osciloscopio analógico. En rojo, aclaraciones externas para la figura; los puntos correspondientes a los instantes t_k y t_{k-1} corresponden al mismo barrido horizontal, sea el segundo punto el que se haya representado primero. La escala vertical (tensión) varía según la tensión de las sondas, que van a las placas de desviación verticales (P.V.), mientras que, en el caso de la escala horizontal (tiempo), se puede decir que el barrido "tira" de la onda a lo largo de la dimensión temporal, haciéndola variar con esta.

Adicionalmente, es importante añadir que la primera etapa de amplificación o atenuación de la señal que captan las sondas sirve de acondicionamiento para no dañar los componentes internos del osciloscopio, así como para visualizar de manera adecuada la forma de onda. También suele haber otro amplificador extra, que conforma un tercer eje "z", el cual maneja externamente las funciones de brillo de la pantalla. Todo ello, obviamente, alimentado desde su respectiva fuente de alimentación.
Se puede observar también en Fig. 21 que las sondas por las que ingresan las señales muestreadas "parecen" especiales. En efecto, se caracterizan por no alterar (o hacerlo insignificantemente) la onda a analizar. Están formadas por conectores BNC (*Bayonet Neill-Concelman*, nombrados así por su forma y sus inventores), una punta, un conductor coaxial y un atenuador capacitivo (se puede escoger reducir la señal [por 10, 100 veces...] externamente al osciloscopio), además de haber otro conductor acoplado (típicamente pinzas de tipo cocodrilo) para conectar a la masa del circuito en el que se está midiendo. Si la sonda o el DSO atenúa por diez veces se escribiría como 10X y, si amplificase, X10.
Entre varios inconvenientes de la opción analógica, está el hecho de que solo se logra una visualización correcta de señales que sean periódicas, ya que la periodicidad es la responsable de la estabilidad visual en pantalla de la forma de onda. También hay problemática si la frecuencia de la señal es demasiado baja; la pantalla no logra retener la representación completa (el barrido es muy lento) y la imagen se ve a trazas inconexas o a puntos. Aun así, este tipo de osciloscopios se suelen preferir para visualizar señales en tiempo real (buscando variaciones rápidas prolongadas o imprevisibles...). Por el contrario, se prefiere usar los digitales cuando no se quiere "monitorizar" la repetitividad periódica, sino analizar eventos (y tramas, transitorios...) de forma estática; sin mencionar los tratamientos matemáticos que se pueden realizar con estos sobre la onda una vez capturada.
Los osciloscopios digitales se han convertido, con el paso de los años, en equipos funcionalmente asimilables, de forma cualitativa, a computadoras (procesadores de información, en el sentido de la palabra), gracias, esencialmente, a las técnicas de conversión analógico-digital, microprocesadores y microcontroladores y a la tecnología del silicio. Aun así, aún conservan la esencia e interfaz de los analógicos (exceptuando los de conexión USB portables basados en PC [PicoScope] y los novísimos software de almacenamiento en nube para osciloscopios [*TekDrive*], entre otros).
El ADC puede decirse que es el elemento más relevante en este aparato, que marca la diferencia entre una buena resolución de la forma de onda y una mediocre. Cumple una función básica: discretizar señales continuas de tensión; se toman valores a intervalos de tiempo regulares, siguiendo una frecuencia de muestreo. Para hacerlo correctamente, ha de cumplirse el *criterio de Nyquist*: para recuperar sin pérdidas una señal que ha de ser muestreada, la frecuencia a la que se haga esto ha de ser mayor que el doble del ancho de banda

(BW) de la señal original (como mínimo, dos muestras cada período) (definiendo BW como el valor que va desde el nivel de continua [0 Hz] hasta la frecuencia en que la señal [senoidal] se atenúe 3 dB; es decir, alcance el 70.7% de su amplitud máxima). También se tratará este tema más en profundidad en siguientes apartados.
Estos osciloscopios pueden estar basados en µP (como ya se ha comentado), dispositivos DSP (D*igital Signal Processor*) de alta velocidad o en tarjetas FPGA (*Field Programmable Gate Array*) (por ejemplo), siendo el propósito de todos estos controlar, tanto al convertidor como al resto de componentes internos (memoria, transistores, periféricos de entrada/salida, buses de comunicación, buffers...).

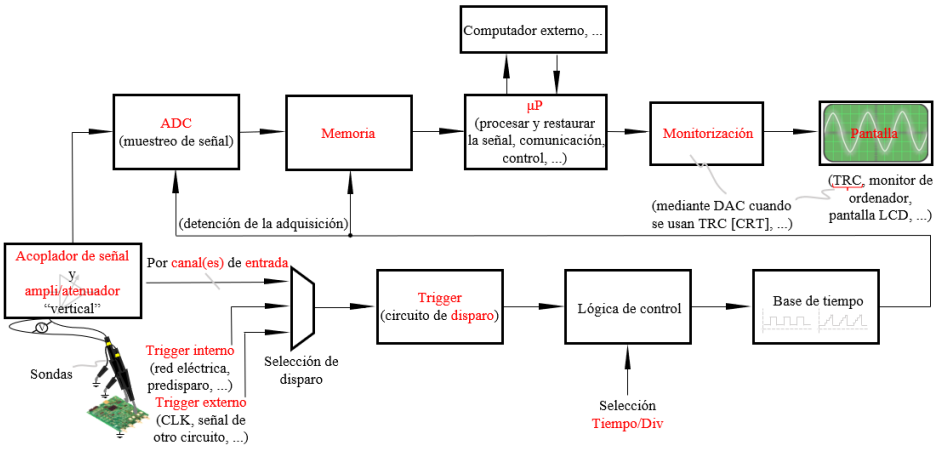

Fig. 23. Diagrama explicativo del funcionamiento y la estructura de un osciloscopio digital. La figura es ejemplificadora; se ha añadido un microprocesador y no una FPGA, no se ha indicado el método concreto de monitorización... El trigger puede ser causado por tres fuentes diferentes (típicamente), como se puede ver en la imagen. Además, entre otras muchas opciones, están las de elegir el tipo de disparo por flanco: se dispara al llegar a un determinado nivel de voltaje en la dirección (flanco de subida o de bajada) especificada. Puede haber una etapa congruente con la de "Base de tiempo" con un comparador de voltaje para facilitar el método de las "aproximaciones sucesivas," si es el que utiliza el ADC integrado.

Tras el muestreo de la señal, se cuantifica su amplitud (o sus amplitudes, hablando discretamente) en código binario (bits), teniendo en cuenta la resolución escogida, que se relaciona con el número de divisiones en que se haya segmentado el eje de tensiones (muestras del conversor). Es ampliamente sabido, pero por poner un ejemplo, un conversor de 8 bits proporcionará una resolución de $2^8 = 256$ divisiones (desde el nivel mínimo hasta el máximo) para la mensurabilidad de la amplitud de la señal de entrada.
La diferencia es clara en la anterior figura; en vez de actuar en placas de deflexión verticales, se usa un ADC que muestrea la señal, guarda la información en memoria (incluso de señales no periódicas) y luego se reconstruye en pantalla. Por otra parte, el trigger o disparo (que también era característico de algunos osciloscopios analógicos) se puede establecer de tres formas distintas, según cómo quiera sincronizarse la señal: puede dispararse según uno de los canales de entrada (lo más habitual), según la frecuencia de la red eléctrica (útil en aplicaciones de potencia e iluminación) o según una fuente externa (por ejemplo, cuando ya se están usando los dos canales para la adquisición de datos o es conveniente sincronizar el disparo con un reloj extrínseco). Es fácil de entender sabiendo que, si la señal no pasa por el nivel de trigger seleccionado, no se obtiene ninguna señal en pantalla (no llega a darse el momento del disparo). También es sumamente práctica la opción de disparo por flanco, que lo produce cuando se alcanza un determinado nivel de tensión en la dirección especificada (flanco de subida o de bajada).

Estas son solo unas pocos de las muchísimas posibilidades que puede llegar a ofrecer un osciloscopio digital moderno. Son capaces de proporcionar un análisis exhaustivo de las señales, con aplicaciones matemáticas (aplicadas en forma discreta, "aprovechándose" del ADC) como el cálculo de valores eficaces, medios, extracción de los armónicos y transformada de Fourier, espectro en frecuencia y diagramas de Bode... Pueden observarse de forma ampliada transitorios de señales y relacionar parámetros típicos (tiempo de establecimiento, de pico, sobreoscilación...) de forma directa, incluso puede establecerse comunicación con una computadora (como se ve en Fig. 23) o transmitir el contenido en memoria a una impresora, memoria flash externa, etc. Aun así, también pueden presentar ciertos inconvenientes. Entre otras cosas, hay que estar "sumamente atento" a satisfacer el teorema de Nyquist; si se amplía demasiado la escala del tiempo (por consecuencia, varía [indebidamente] la frecuencia de muestreo) o se quiere observar una señal muy lenta, puede aparecer el fenómeno del *aliasing*, lo que causaría una señal falsa en pantalla. Son aparatos, a lo sumo, de mayor costo y que requieren bastante más mantenimiento y dedicación que los analógicos.
Punto y aparte, la otra gran familia de equipos de instrumentación que se menciona en la pregunta principal es la de los "analizadores lógicos" (AL). Merece la pena dedicar un tiempo a citar las características comunes que tienen estos y los osciloscopios (se referirá mayormente a los digitales), ya que se puede aprovechar gran parte del conocimiento adquirido para los últimos con fin de comprender el funcionamiento y las aplicaciones de estos.
La diferencia más notable entre ambos es el número de canales de entrada; mientras algunos modelos de osciloscopios disponen de la capacidad de análisis lógico de los otros aparatos, presentan muchos menos canales digitales (16 frente a los 136 [típico máximo comercial actual] que pueden ofrecer los analizadores lógicos). Consecuentemente, los DSO suelen tener hasta 4 ADC (de 8, 10 bits...); por el contrario, un AL puede llegar a estar compuesto por más de cien convertidores analógico-digital de 1 bit. Los AL ofrecen una característica de la forma de onda menos detallada: dos niveles lógicos diferenciables, por encima y por debajo de una tensión umbral; si bien es cierto que para el fin de su aplicación es suficiente (aunque con el avance tecnológico, la diferencia en exactitud converge cada vez más rápido a cero).
La historia de los AL viene ligada al nombre de Gary Gordon, que trabajaba en el campo de los osciloscopios digitales un laboratorio de HP. En 1967 inventó el analizador lógico como propuesta de mejora, en varios aspectos, de los DSO. Inicialmente estaban fuertemente emparentados al análisis y "testeo" hardware, ya que se desarrollaron bajo el contexto de los osciloscopios, pero con el paso de los años sus funciones se encaminaron hacia la depuración software y la monitorización de señales de microprocesadores, IC, etc. (visualización de mnemotécnicos relativos a instrucciones, obtención de cronogramas...).
Concretando, los analizadores lógicos son un tipo perteneciente a un grupo de aparatos y dispositivos destinados a analizar la "lógica" de un circuito (de electrónica digital, típicamente; lógica biestado). Los acompañan las sondas lógicas (que detectan estáticamente estados de

tensión, al margen de la evolución temporal) y analizadores de protocolo (que se usan para analizar métodos o procesos de comunicación o transmisión de información en la interconexión de dispositivos).
Así, como primera conclusión, se puede decir que los analizadores lógicos permiten un análisis dinámico (teniendo en cuenta la evolución temporal) de distintos puntos (nodos) de un circuito digital y, además, se prefieren usar en circuitos digitales complejos, cuando no se requiere una gran exactitud en la medidas de amplitud y tiempo, sino en el sincronismo de las señales entre sí.

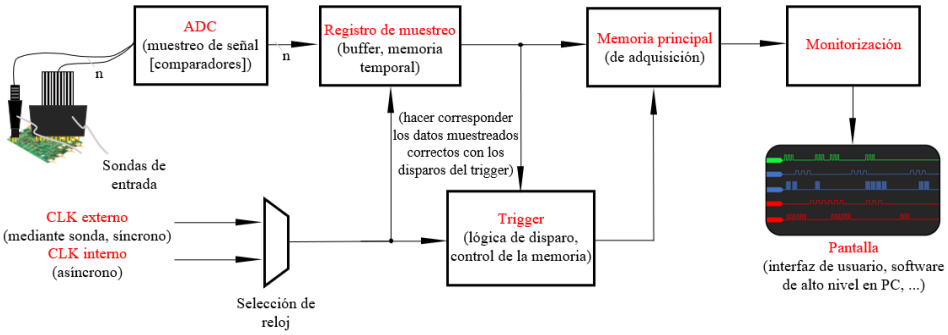

Fig. 24. Diagrama explicativo del funcionamiento y la estructura de un analizador lógico. La forma comienza siendo similar a la de los DSO, pero claramente el propósito en este caso no es la visualización fidedigna de la forma de onda. Se ven dos métodos de muestreo: síncrono (una de las señales entrantes se toma como reloj de muestreo) o asíncrono (se escoge un reloj interno de frecuencia variable; la incertidumbre máxima que aporta coincide con su período). Los datos se almacenan en memoria al efectuarse un disparo.

La herramienta de disparo tiene una importancia enorme para estos equipos; de entre otras opciones, generalmente puede haber varios tipos de trigger: por flanco (trigger dispara en la transición a estado alto o bajo de una señal digital), por transición (dispara cuando una señal lleva en transición más de un tiempo determinado), por transitorio (para detectar *glitches* [no deseados], o pulsos de menor duración que un tiempo determinado), disparo lógico (por combinación [funciones lógicas...] de varias señales de entrada) o disparo por exceso de duración (cuando se mantiene en un estado un tiempo mayor que el especificado).
Aparte, las puntas de las sondas que se utilizan para muestrear las señales (ya sea individualmente o en "pods" que agrupan conjuntos de ellas) pueden aportar varios efectos parásitos indeseado a la tensión, como puede ser disminuirla en amplitud (efecto resistivo) o retrasar su cambio de estado (capacitivo). Aunque para facilitar el acceso a dispositivos de montaje superficial (SMD) se han desarrollado puntas que poseen dos zonas de contacto con el "pin" unidas entre sí, lo que reduce estos efectos adversos. Más aún, existen adaptadores montables sobre el circuito y el propio integrado que facilitan la depuración de dispositivos con encapsulado cuadrado plano de perfil bajo (TQFP), por ejemplo.

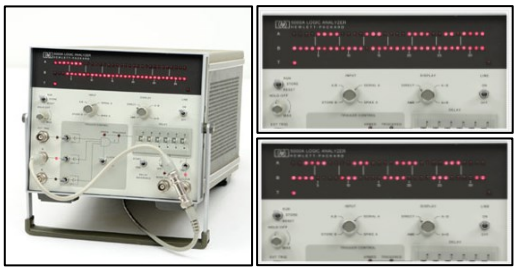

Fig. 25. El primer prototipo de analizador lógico, el modelo HP 5000A, lanzado por la empresa que lleva en su nombre en 1973. Con dos canales de entrada y 32 diodos LED dispuestos horizontalmente para visualizar temporalmente las señales digitales que captan las sondas. A la derecha se pueden ver dos instantes de funcionamiento del aparato. Se siguió y mejoró este canon de trabajo con otros modelos como los HP 1601L o 1600A, que mostraban matrices de ceros y unos en pantalla para visualizar la información.

Finalmente, como reflexión, se puede comentar que, con el desarrollo de software específico (simuladores, depuradores...) para dispositivos programables como microprocesadores, el uso de los analizadores lógicos ha ido decayendo a lo largo del tiempo (exceptuando las aplicaciones particulares), si bien es cierto que las simulaciones no llegan fácilmente a mimetizar el comportamiento que tendría un sistema supervisado por equipos y aparatos reales.
Todos los instrumentos referidos en este apartado pueden usarse con gran versatilidad, dependiendo en gran medida de la imaginación y capacidad de adaptación del usuario. Por ejemplo, en la industria electrónica para el desarrollo, mantenimiento de aparamenta y trabajos de laboratorio, en la medicina para la medición de parámetros biológicos (ritmo cardíaco, medición de la presión arterial...), en navegación para sistemas de radar y sonar, en automoción (control de calidad de motores, refinamiento de dispositivos...) y un amplio etcétera.
Son, en definitiva, un claro ejemplo del apoyo intelectual concatenado durante generaciones, vertido comúnmente en un invento que aúna fascinantes fenómenos y teorías de una forma práctica y totalmente funcional.

X. PULL-UP Y PULL-DOWN

What is a pull-up resistor used for? How about a pull-down resistor?

En castellano: ¿para qué se usa una resistencia de *pull-up*? ¿Y una resistencia de *pull-down*?
Se puede comenzar por comentar alguna cosa sobre el concepto de "familia lógica". Las familias lógicas son un conjunto de conjuntos. Cada una puede contener varios dispositivos electrónicos, como circuitos integrados, que tienen en común ciertos aspectos: estructura interna, niveles lógicos, tensión de alimentación y tecnología de fabricación; consecuentemente, presentan similitudes físicas tanto en entradas, salidas, como en sus componentes electrónicos intrínsecos. El concepto primario de "familia lógica" surgió como un método de interconexión de distintos CI mediante puertas lógicas individuales o pequeños grupos de ellas, en vista de abarcar configuraciones cada vez más complejas, que no serían sino inoperables usando CI únicos. La gran escalabilidad de los dispositivos a mayores escalas de integración (gracias a la tecnología del silicio, al VLSI estructurado de Carver Mead...) apartó al desuso a antiguos componentes de estado sólido o válvulas de vacío y provocó la estandarización de los métodos con familias lógicas, con los cuales se busca, entre otros propósitos, disminuir el consumo de energía, reducir el tamaño del integrado y engrosar la densidad de transistores.
El uso de circuitos que pertenecen a una misma familia permite facilitar el acceso a capas más complejas de desarrollo, ya que los componentes tienen una electrónica común. A principios de 1960, se empezó a desarrollar en IBM la primera familia lógica, denominada ECL (*Emitter Coupled Logic*) o CML (*Current Mode Logic*), que utiliza transistores

bipolares conduciendo en zona activa (lo que aumenta la velocidad respecto a las conmutaciones entre corte y saturación). Fueron apareciendo otras familias lógicas como RTL (*Resistor Transistor Logic*) que trataban de simular, en un integrado, el comportamiento de ciertos componentes discretos usualmente usados hasta esa fecha; se implementa mediante transistores bipolares conmutando entre zona de corte y saturación, junto con resistencias que minimizan el número de compuertas o "pines" del propio circuito integrado. También surgió la familia DTL (Diode Transistor Logic); la utiliza, por ejemplo, la primera computadora totalmente transistorizada, la IBM 608 *Transistor Calculator*, lanzada en 1957. Consta de ciertos diodos cuyo propósito es optimizar el área de integración y realizar (junto a los transistores bipolares inversores) la función lógica. Un poco después (en 1963 y 1964), empresas como *Osram Sylvania* y *Texas Instruments* introdujeron al mercado una nueva familia lógica, que se corresponde a la última (original) que usaría transistores bipolares, denominada TTL (Transistor-Transistor Logic). Esta topología tiene, en realidad, transistores multiemisor en la etapa de entrada y varias resistencias entre estos y los bipolares, que hacen las funciones lógicas. Si bien la velocidad de transmisión entre estados lógicos es plausible, su consumo de potencia (al tratar con ese tipo de transistores) no lo es; por eso, se han ido desarrollado y mejorando varias modificaciones de esta familia (como pueden ser S [*Schottky*], FAST [*Advanced Schottky*], ABT [*Advanced BiCMOS,* un híbrido]...) que, en cierto modo, siguen siendo compatibles con la primera. Estos circuitos se alimentan a una tensión (V_{CC}) de, generalmente, +5 V (con histéresis de 0.25 V), su nivel alto (de entrada) se corresponde a tensiones mayores de 2 V (el 40% de V_{CC}) y el nivel bajo a tensiones menores de 0.8 V (16% de V_{CC}). En la salida, los valores son de 2.7 y 0.4 V, respectivamente (los valores intermedios pertenecen a un estado indeterminado). Su gran desventaja ocasiona que no sea una familia idónea para altas escalas de integración, ni circuitos sumamente complejos. Por esto mismo, surgió otra familia más (MOS), basada en los aclamados transistores de efecto campo de metal-óxido-semiconductor (de puerta aislada), los MOSFET (de enriquecimiento). Como en la implementación de funciones lógicas con esta familia puede evitarse el uso de resistencias, la escala de integración se dispara (por lo alto) y más aún si la tecnología de fabricación incipiente (la del silicio) es beneficiosa para la misma. Cabe mencionar que con la familia MOS las funciones lógicas quedan controladas por tensión (tensión de la puerta de los transistores de entrada). Como la movilidad de los portadores de carga mayoritarios (responsables de la conducción) es menor cuando estos son huecos que cuando son electrones, se generalizó el uso de transistores NMOS (de canal n) para la familia. Pero, sobre 1968, la empresa RCA lanzó la serie de circuitos lógicos CD4000, junto a la que emergía una familia más (la última que se va a comentar, mas aún existen múltiples subfamilias), derivada de la MOS: la familia CMOS, con la cual se podían implementar transistores de canal n y p en una misma función lógica (realmente, su invención puede atribuirse a Frank Wanlass y Chih-Tang Sah, de Fairchild Semiconductor, unos 5 años antes). La efectividad tecnológica de esta familia superó con creces a la TTL y también a su antecesora, la MOS, en términos, mayoritariamente, de consumo de potencia estática, vida útil y facilidad de diseño. Potenció también el desarrollo de CI personalizados para cada aplicación (ASIC [*Application Specific Integrated Circuit*]). Los circuitos de esta familia se alimentan a tensiones desde +2 V a +18V (aunque puede variar: de 3 a 18 V para la serie CMOS 4000 [subfamilia CD4K], 2 a 6 V para la subfamilia HC...); su nivel alto (de entrada) se corresponde a tensiones mayores del 70% de V_{CC} y el bajo a tensiones menores del 30% de la misma. Realmente los niveles de tensión dependen de la serie y subfamilia que desarrolle cada fabricante y de los datos que se proporcionen en sus hojas de características, por lo que es infructuoso reunir más datos de los básicos ya presentados sobre este aspecto.

Como los transistores que forman parte de la estructura de los circuitos CMOS presentan una fina capa de dióxido de silicio (típicamente) que aísla la puerta del substrato, este queda expuesto a ser dañado por electricidad estática de alguna fuente externa, si no se encuentra debidamente alimentado. Su gran impedancia de entrada (corriente por la puerta idealmente nula) hace de ellos unos dispositivos que requieren muy poca potencia para ser controlados (en este caso por tensión), pero su delgada capa de óxido puede no ser capaz de soportar la carga acumulada (funciona como una especie de condensador [MOS]) y, por tanto, la "ruptura" de continuidad eléctrica es capaz de dejarla dañada y con ella, el transistor y el resto del circuito. Cabe destacar que existen ciertos componentes que proporcionan protección interna contra descargas electrostáticas que provengan de este tipo de causas (como los *bus-holder* o *bus-keeper* en los procesadores i.MX RT *Crossover* de NXP Semiconductors). No es recomendable dejar puertas "flotantes" (esto es, pines sin conectar a nada) en muchos integrados (se indicaría el caso contrario); particularmente, los CMOS pueden llegar a disipar mucho calor ya sea por corrientes de fuga, inducción de conductores cercanos o electricidad estática. La causa es que, al dejarlos "al aire", el voltaje no queda definido, así que puede llegar a tomar algún valor intermedio tal que ambos transistores (p y n) conduzcan al mismo tiempo, generando calor, causando respuestas espurias e impredecibles y deteriorando (presumiblemente) el circuito. Para evitar esto, hay que especificarles un nivel de tensión concreto, conectándolos a masa (GND) o a V_{CC}, con resistencias entre estos y los pines en cuestión; son las resistencias de pull-up y de pull-down. Las primeras tratan de asegurar un nivel lógico alto (un "1") cuando una entrada (o salida) está en reposo, mientras que las segundas establecen un "0" lógico.

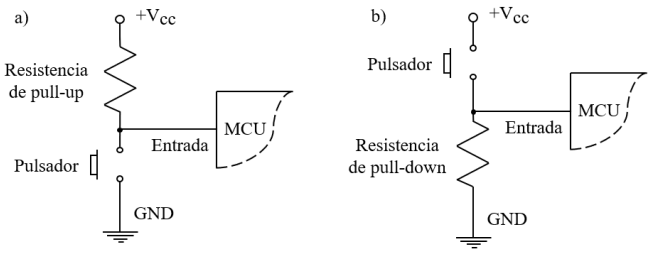

Fig. 26. a) Resistencia de pull-up (1k, 10k...), conectada a alimentación y a un pin de entrada digital de un µC. Con el interruptor (o pulsador) abierto, la corriente fluye a la entrada dando un 1 lógico; cuando está cerrado, se dirige hacia GND por el "camino de menos resistencia", dando un 0 lógico (niveles de estado bien definidos). b) Resistencia de pull-down, conectada a GND y a un pin de entrada digital. Con el interruptor abierto, la caída de tensión en la resistencia es prácticamente cero y la entrada queda conectada a masa dando un 0 lógico; cuando se cierra, la corriente fluye hacia el pin dando un 1 lógico.

Como se ha comentado, en la salidas de los circuitos digitales también suele haber este tipo de resistencias. Hay ciertas topologías de conexión en las salidas denominas de "colector abierto". El motivo es que las puertas lógicas que convergen en el pin del integrado van unidas a este mediante un transistor BJT (bipolar) de tipo NPN (si fuese un MOSFET,

la salida sería de “drenaje abierto”), quedando su colector abierto.

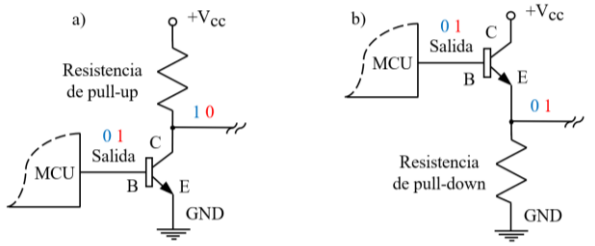

Fig. 27. a) Salida de un CI de colector abierto conectada a una resistencia de pull-up; con un 1 lógico en la base (B), el transistor entra en saturación y se cierra el camino entre el colector (C) y el emisor (E, conectado a GND), llevando la salida (C) a masa (0 lógico). Cuando hay un 0 lógico en la base (o queda desconectado, fuera de saturación), la salida queda conectada a la alimentación y se da un 1 lógico (tanto mayor será la corriente hacia la salida y el riesgo de cortocircuito entre alimentación y masa cuanto menor sea la resistencia). b) Caso con resistencia de pull-down; se seguiría la lógica es inversa al caso anterior, solo un 1 en B da un 1 en la salida (E), mientras que un 0 en B da un 0 lógico.

Como sin este “arreglo” de las resistencias, en algún momento la salida puede tener cualquier valor de tensión entre 0 V y su alimentación (puerta flotante) (aunque generalmente su corriente sea despreciable), puede haber lecturas erróneas en posteriores etapas (bobinados de relés, tiras LED...) y se puede inducir algún fallo. El sistema comentado solventa esto y, además, la alimentación (con las resistencias de pull-up) puede ayudar para accionar o excitar las mencionadas etapas subsiguientes, que de otra manera podrían requerir una corriente notablemente mayor que la que puede proporcionar el propio integrado.

Cabe mencionar que ciertos integrados (generalmente algunos µC) proporcionan resistencias de pull-up internas que permiten facilitar la conexión de otros elementos a sus salidas, evitando el uso de otras resistencias externas, si bien con estas el usuario puede escoger el valor que considere apropiado. Además, son sumamente importantes en protocolos de comunicación como el bus I^2C (Circuito interintegrado, bus serie de datos desarrollado en 1982 por *Philips* [actualmente NXP] *Semiconductors*); de ellas depende su funcionamiento.

XI. Convertidores ADC

What's the difference between a successive approximation ADC and a Sigma-Delta ADC?

En castellano: ¿cuál es la diferencia entre un ADC de aproximaciones sucesivas y un ADC Sigma-Delta?

La antigüedad de las conversiones de cantidades analógicas a digitales puede resultar sorprendente. Realmente, hay una discusión “casi” filosófica sobre si esta denominación es verídica, ya que el mundo analógico (esto es, continuo, compuesto de infinitas partes) debería de darse, como tal, solamente en abstracciones matemáticas, no en el entorno físico real. Es interesante rememorar citas y pensamientos de ciertas personas relevantes; ha de recalcarse que se manejan “cantidades”, en todo momento (“nada mensurable puede medirse si no es por fracciones que expresen el resultado de la medición, a menos que la medida esté contenida un número exacto de veces en lo que se quiere medir”, Joseph Louis Lagrange) y se ha de relacionar todo el contenido del apartado con el concepto de no cuantificar el infinito como algo completo (Carl Friedrich Gauss), sino como una manera de visualizar el redimensionamiento discreto de las cantidades para la práctica pragmática con ellas.

Ya en el siglo XVIII, se había diseñado en Turquía (Imperio Otomano, antes de 1923) una máquina hidráulica con la que se regulaba el flujo de agua pública. A grandes rasgos, el sistema se dividía en tres partes; un almacén al que entraba el agua (variable [presión, potencial eléctrico...] analógica) y el cual se mantenía a nivel constante (puede imaginarse como una referencia de tensión de alimentación en los equipos electrónicos [ADC] actuales), una sección con 8 boquillas divisoras de caudal, los cuales estaban referenciados, decrecientemente, a una métrica base (concretamente 1 *lüle*, equivalente a 36 l/min) y debían ser manualmente monitorizados para controlar si había un flujo razonable a través de ellos en un determinado momento (se pueden relacionar, directamente, con los 8 bits de conversión de un ADC) y, por último, una etapa de salida en la que se recogía el agua medida (o su volumen o masa, indistintamente) (semejable al espacio de memoria, que recoge las conversiones digitales).

Desde los inicios del siglo XIX (primer desarrollo significativo del telégrafo en 1829 por Joseph Henry), la tecnología de las comunicaciones ha sido uno de los mayores propulsores del desarrollo de convertidores AD electrónicos. Del telégrafo (que hace uso de pulsos eléctricos por cable para la transmisión de información discreta en forma de lenguaje codificado [Morse, Gray...]) surge el teléfono (gracias, en parte, a Alexander Graham Bell en 1875), también la necesidad de multiplexar más de una señal en un mismo conductor para aumentar su capacidad (posibilidad de múltiples flujos de información simultánea por un solo medio de transmisión; multiplexación por división de tiempo [TDM], de frecuencia [FDM], modulación por pulsos codificados [PCM]...) y consecuentemente, aparece la influyente *Bell Telephone Company*. Es remarcable que Bell notó, a partir de un sistema de multiplexación con un telégrafo que, de la misma forma que estaba transmitiendo señales “digitales” con el mismo, podía hacer de su voz una variable analógica y reconvertirla mediante cierto mecanismo: una membrana vibratoria provocaba una variación de resistencia de una armadura con virutas de carbón alimentada en tensión, una corriente se conducía luego por un cable que hacía inducir un campo magnético variable y proporcional a la misma en un electroimán, provocando la vibración de otra membrana que reproducía de nuevo una “imagen” del sonido de entrada.

La multiplexación del telégrafo mediante TDM (que consiste en asignar intermitentemente la totalidad del ancho de banda del canal a cada usuario que comparte la señal) llevó en 1903 a Willard M. Miner a experimentar con un sistema (usando un conmutador electromecánico en rotación) que permitía juntar varias señales de teléfono analógicas en un solo par de cables, previamente a la consecuente reconstrucción de las señales; con sus resultados se intuyó el requerimiento de muestrear la señal a una determinada velocidad mínima (o más bien adecuada) para preservar de forma correcta sus características originales.

Sobre la década de 1930, en los Laboratorios Bell se desarrolló la FDM, que buscaba el mismo propósito que la técnica antes mencionada. Hacían uso de tubos de vacío (Lee de Forest, 1906) para transmisiones largas y, por otra parte, de amplificadores de realimentación negativa (Harold Stephen Black, 1927), los cuales estabilizaban mejor la ganancia y eran más adecuados para la aplicación en los repetidores de señales; sin embargo, se quería buscar otro tipo de método de transmisión que no fuera tan sensible al ruido y a la distorsión, pero que permitiera simultáneamente la multiplexación.

La PCM comenzó a investigarse en 1921 dentro de *Western Electric*: Paul M. Rainey describió un método de transmisión y reestructuración de facsímil en paquetes de 5 bits a través de un telégrafo, usando un convertidor optomecánico para la codificación de la información. Se diferenciaban varias etapas en su sistema: una emisión de luz hacia la transparencia del material a ser transmitido, una fotocélula que recogía la luz y proporcionaba una corriente a un galvanómetro que dirigía otro haz de luz hacia una de 2^5 (32) fotocélulas, una serie de relés localizados de tal forma que codificaban la posición de las células que los activaban, un conmutador rotatorio que hacía de distribuidor serial de esta información eléctrica codificada (asimilable a los actuales ADC tipo flash) y, de forma inversa, las etapas de salida, que se conformaban por otro receptor serial en rotación, un convertidor D/A de 5 bits que reconstruía en paralelo el código (digital) en otro banco de relés y una resistencia variable conectada a una lámpara, mediante lo que se alteraba su intensidad (analógica) y, por tanto, la cantidad de luz incipiente en el receptor fotográfico que conformaba la etapa final del sistema, presumiblemente reproduciendo la imagen (en negativo) de partida.

Alec Harley Reeves redescubrió en 1937 la modulación por pulsos codificados, sus patentes presentaban unos de los primeros ADC y DAC electrónicos; hacía uso, entre otros, de contadores (de 5 bits) encadenados, biestables (*flip-flop*, inventados en 1928 por William Henry Eccles y Frank Wilfred Jordan), divisores de frecuencia, un reloj oscilador (de 600 kHz), un modulador por ancho de pulso y filtros paso bajo.

Partiendo de los estudios que desarrollaron estos inventores, en los Laboratorios Bell se empezó a desarrollar durante la IIGM una tecnología propia de PCM. En la línea de trabajo y motivación estaba la encriptación de los mensajes de voz y discursos. Por ejemplo, H. S. Black y J. O. Edson diseñaron en 1947 un sistema que digitalizaba una señal de voz humana (su espectro completo, [250, 3000 Hz]) en 5 bits, muestreando a 8 kS/s (muestras al segundo) y usando un ADC con registro de aproximaciones sucesivas (SAR).

El método de aproximaciones sucesivas usa un comparador entre la tensión de la señal de entrada (V_1) y la que entrega un DAC interno (V_2), que se va ajustando en proporciones recíprocas a potencias de dos a medida que su valor se acerca más o menos al primero (comenzando por el punto medio del rango de tensión de entrada permitida). Al comenzar la orden de conversión, el bit más significativo del registro (MSB) sube a 1; si V_1 es mayor que V_2 el bit se mantiene, si no es así, baja a cero y el siguiente bit más significativo sube a 1, repitiéndose el proceso al inicio del siguiente muestreo (el tiempo total de conversión viene determinado por el producto del período del reloj y el número de bits del sistema). Al final de la conversión, los bits en paralelo del SAR están disponibles para su transmisión y lectura.

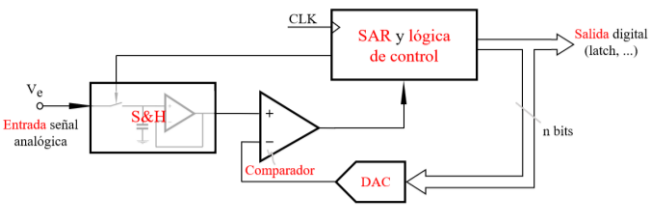

Fig. 28. Diagrama de bloques que muestra el funcionamiento de un convertidor analógico-digital de aproximaciones sucesivas. En el momento en que se da la orden de inicio a la lógica de control, se desencadena el muestreo de la señal y la adquisición de datos de forma sincronizada. A la salida, un biestable (por ejemplo) puede recoger los bits que cuantifican digitalmente la señal analógica de entrada.

Para optimizar el funcionamiento del SAR, hay inicialmente una etapa de muestreo y retención (amplificadores S&H...) con la que se logra mantener la señal analógica de entrada constante durante la toma de la muestra.

La arquitectura sigma-delta (Σ-Δ) (o delta-sigma, según la literatura o la topología del circuito en cuestión) tuvo sus orígenes en las primeras fases de desarrollo de sistemas PCM. Más concretamente, en relación con las técnicas de transmisión denominadas "modulación delta" (*ITT Labs*. en Francia, 1946) y "PCM diferencial" (p.ej., Labs. Bell. en 1950). La idea general en ambos casos es lograr una mayor eficiencia de la transmisión mediante los cambios o las diferencias (Δ) entre muestras consecutivas, no mediante los propios resultados de los muestreos. En la modulación delta, una señal analógica es captada por un comparador (ADC de 1 bit) y reconvertida mediante un DAC de 1 bit, para luego ser substraída de la entrada del comparador, pasando antes por un integrador (amplificador que produce a la salida un voltaje proporcional a la amplitud y duración del de entrada); un 1 a la salida del comparador significa una evolución positiva de la señal entrante desde la última muestra, por el contrario, un 0 significa una excursión negativa. Los PCM diferenciales siguen la misma estructura de funcionamiento, solo que, en vez de un comparador, usan un ADC flash (y el mismo DAC) de n bits. Cabe destacar que los convertidores flash (o directos) están formados por una cadena de divisores de tensión y de comparadores, permitiendo realizar la conversión de manera inmediata y directa; el número de comparadores (2^n-1) (igual que el de los niveles) aumenta con los bits de conversión (n). Por otra parte, empíricamente se ha observado que, para que los PCM diferenciales logren la calidad de los PCM originales, se han de alcanzar velocidades de muestreo de hasta 20 veces la máxima de la señal muestreada (frente a las 2 veces que marca el criterio de Nyquist [se verá posteriormente]).

Los convertidores Σ-Δ actuales sobremuestrean la señal (esto es, muestrear a una frecuencia mayor que la dictada por Nyquist) para que disminuya la densidad espectral de potencia (PSD, muestra la distribución energética para cada frecuencia que forma la señal), ya que el ruido de cuantificación causado por la discretización de la señal analógica se reparte sobre un ancho de banda mayor, por tanto, en la banda de interés recaería un ruido menor. Adicionalmente, un lazo de realimentación negativa puede proporcionar una compensación del error generado y mejorar aún más la resolución del sistema. Una primera etapa consiste en un modulador Σ-Δ, el cual convierte la señal analógica de entrada para proporcionar una corriente lógica digital cuantificada en 1 bit (normalmente). La variable analógica pasa por un amplificador diferencial (Δ) y, la diferencia entre esta y la tensión del DAC de realimentación, se dirige a un integrador (Σ); posteriormente, se compara (con el ADC de 1 bit, a una frecuencia muy alta) con un voltaje de referencia, dando la salida lógica antes mencionada. Se puede decir entonces que este modulador mide la diferencia entre la señal de entrada analógica y la salida del DAC de realimentación. Una segunda etapa está formada por un filtro digital (*noise-shaping*, elimina el ruido de cuantificación que queda fuera de la banda frecuencial de interés) (por ejemplo, mediante un DSP) para generar una representación digital (de bajo ruido y de alta resolución) de la señal de entrada y un diezmador o *decimator* que reduce la tasa de datos de salida a la frecuencia establecida por Nyquist sobre la señal original (ajustando el factor denominado "proporción del diezmador" [DR], para reconstruir y asegurar la señal correctamente).

Puede haber una etapa anterior a la primera que consiste en un filtro "antialiasing", para eliminar de la señal de entrada aquellas componentes espectrales que se encuentren por encima de la mitad de la frecuencia de muestreo; es una opción razonable ya que la frecuencia de corte de este filtro ha de caer después del ancho de banda de entrada (para no acortarlo) y ser menor que la frecuencia de muestreo (para cumplir con Nyquist).

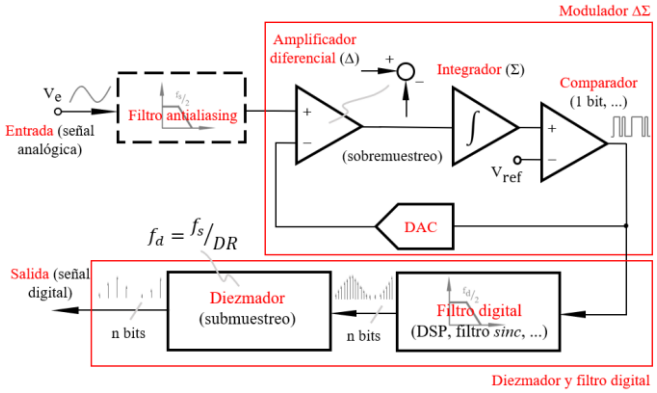

Fig. 29. Diagrama de bloques que muestra el funcionamiento de un convertidor analógico-digital sigma-delta (Σ-Δ). Se pueden diferenciar varios bloques funcionales básicos. Es intuitivo notar que el desarrollo de la electrónica del estado sólido y circuitos integrados facilita la implementación de este tipo de topologías (circuitos con amplificadores diferenciales e integradores, combinacionales y procesadores de señales...). A la salida del diezmador se "reconstruye" la señal a la frecuencia de muestreo de los datos (teóricamente la de Nyquist, relativa a la señal analógica de entrada), manteniendo la resolución (típicamente 12 a 32 bits de conversión) y reduciendo el ruido (alto) de cuantificación generado por el modulador.

En 1954, Cassius Chapin Cutler (de Bell Labs.) ya había introducido el sobremuestreo con el propósito de transmitir la señal ruidosa y conseguir mejores resoluciones sin disminuir la velocidad de muestreo. Sin embargo, la inexistencia de electrónica de estado sólido suficiente por aquel entonces dificultó el refinamiento de sus métodos, excluyendo de etapas como el diezmado o el filtro digital a sus sistemas.

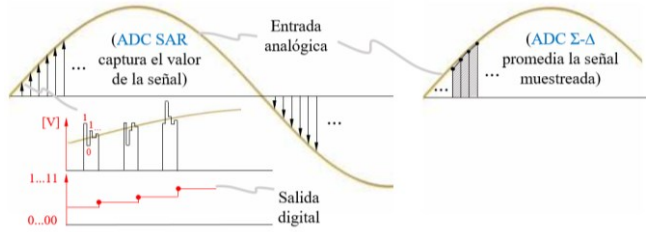

Fig. 30. En esta figura se puede observar la diferencia principal entre ambas topologías de convertidores AD, respecto al método de muestrear la señal analógica de entrada.

Otra topología (lenta pero precisa) de convertidores AD son los de rampa; se usa un integrador para aumentar una tensión (rampa) hasta alcanzar la variable de entrada, mientras que un contador registra los pulsos que aparecen hasta que eso sucede. También hay convertidores de dos rampas, con los que se compensa el error generado con el circuito de integración.

Bien es cierto que las estructuras comentadas son básicas o genéricas, ya que una parte importante de la labor de los fabricantes de convertidores consiste en optimizar cualquiera de sus topologías. Aun así, se pueden sintetizar las diferencias de los tres tipos de convertidores más importantes según sus posibles usos o aplicaciones: los ADC de aproximaciones sucesivas ofrecen una buena resolución con una tasa de muestreo moderada, los convertidores flash presentan la tasa de muestreo más rápida pero la resolución más baja y, finalmente, los ADC sigma-delta proporcionan una resolución muy alta, así como mejores reducciones de ruido, pero también velocidades más bajas que los demás.

XII. Transmisión en serie de datos

What are the differences between SPI and I²C?

En castellano: ¿cuáles son las diferencias entre SPI e I²C?

Manteniendo la temática del apartado anterior, puede imaginarse una situación en la que los datos salientes (ya digitalizados) de un conversor AD necesiten ser volcados en algún otro sistema (MPU, memoria...). Primeramente, la forma razonable de establecer esta comunicación puede parecer ser en paralelo (como hace el HP *Instrument Bus* desde finales de 1960, pensado para facilitar la interconexión de instrumentos y controladores), ya que la trama de datos se transmitiría rápida y directamente; pero si el número de bits es alto y, además, se añaden líneas adicionales para gestión de lectura, acceso de los dispositivos..., la cantidad de conexiones puede ser demasiado alta para resultar fiable y manejable. Por eso, suele ser habitual que estos sistemas se conecten mediante salidas comunicadas en serie síncronamente (un solo bit por ciclo de reloj), gracias, concretamente, a los protocolos I²C y SPI.

El protocolo I²C (o I2C, Circuito *Inter-Integrado*) fue desarrollado en 1982 por Philips (actualmente NXP) Semiconductors. En líneas actuales, se define como un bus de comunicación serie síncrono (cambios de estado sincronizados con una señal de reloj), multimaestro y multiesclavo (se selecciona a un maestro de un grupo de dispositivos concreto, que controla la comunicación con uno o más [dispositivos o procesos] esclavos), conmutado por paquetes (método por el que se dirigen los datos útiles, transmitidos hacia una ruta, mediante unos paquetes de control), *half-duplex* (admite comunicación bidireccional, pero no simultánea, entre los dispositivos conectados) y de un solo extremo (método más usual de señalización, en el que un cable maneja un voltaje variable que representa la señal [estado alto y bajo...] y otro cable está conectado a un voltaje de referencia o masa).

Se suele implementar para conectar circuitos integrados y periféricos de baja velocidad a procesadores, controladores, etc., a corta distancia (puede ser bastante sensible al ruido y, por tanto, a reducciones de la velocidad efectiva). No debe confundirse con I²S (*Inter-IC Sound*, introducido también por Philips Semiconductors en 1986), que es un estándar de bus serie para comunicar datos de audio digital PCM entre varios CI dentro de un dispositivo electrónico.

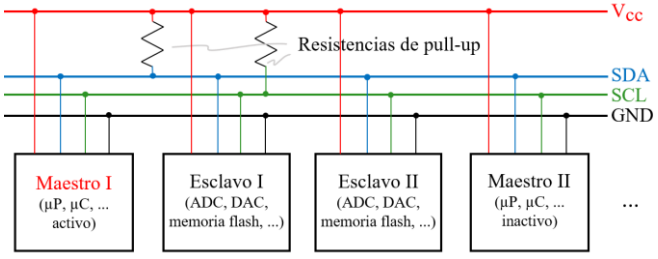

Fig. 31. Esquema que muestra las dos líneas necesarias en el protocolo I²C: SCL [línea serie de reloj] proporciona el reloj, SDA [línea serie de datos] proporciona los datos (ambas conectadas mediante resistencias de pull-up, porque cuando están inactivas han de mantenerse en estado alto [3.3, 5 V...], entre otras razones) y, adicionalmente, masa. El maestro comienza la comunicación mandando 7 bits con la dirección del esclavo más un bit para especificar si quiere leer o escribir. El esclavo con esa dirección responde con un bit de reconocimiento (ACK), tras lo cual el maestro procede con la interacción; cuando acaba, envía un bit de parada (*stop*).

SPI significa Interfaz Periférica Serial, es un tipo de conexión que se ha ido estandarizando a lo largo que su uso se extendía; fue diseñada a mediados de 1980 por *Motorola Inc.* También es una especificación de interfaz de comunicación describible en serie síncrona, que es de tipo *full-duplex* (ambos dispositivos conectados pueden comunicarse bidireccional y simultáneamente entre sí) y de arquitectura maestro y multiesclavo (un sistema digital, p. ej., sería el maestro, mientras que uno de entre varios sistemas ADC o DAC sería esclavo).

Bastantes circuitos de memoria flash se comunican con procesadores a través de este bus, ya que son útiles para almacenar información externamente. Más aún, recordando que los analizadores de protocolo (y algunos lógicos) pueden interpretar buses digitales, es beneficioso implementar sistemas SPI si, posteriormente, se quiere monitorizar la información transmitida.

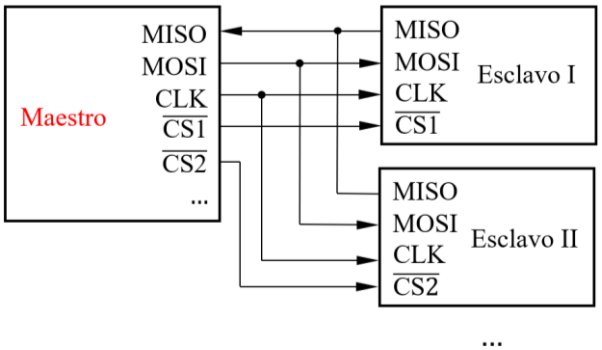

Fig. 32. Esquema que muestra las cuatro líneas necesarias en el protocolo SPI: MISO (*Master-In Slave-Out* o SDO [*Slave Data Out*], recepción del maestro), MOSI (*Master-Out Slave-In* o SDI [*Slave Data In*], transmisión del maestro), CLK o SCLK (*Clock*, reloj sincronizador generado por el maestro) y CS o SS (*Chip Select* o *Slave Select* [una línea de estas por cada periférico subordinado], si la línea está activa, entonces se permite la transmisión de datos hacia el esclavo al que va dirigida [o al revés]) (normalmente de nivel activo bajo).

Como el maestro proporciona el reloj sincronizador, no hace falta que el procesador (por ejemplo) o el periférico tengan un oscilador externo particular y, desde luego, se ahorra que ambas partes tengan que conocer la configuración del protocolo (como ocurre con la tasa de baudios, paridad... del estándar RS-232). La velocidad puede ir de varias decenas de kHz a unos 100 MHz, resultando en 1Mbyte/s (8000 kbps) con un reloj a 8 MHz, en contraste con los 19.2 kbps típicos del RS-232 o los 5000 kbps (máximo actual, originalmente 100 kbps [estándar]) del I²C. Bien es cierto que las señales bajo el estándar SPI no están pensadas para viajar a más distancia que la que aborda la placa del circuito en cuestión (distancias algo más largas permite el I²C, pero nada más lejos que varios metros antes de que el cable degrade, prácticamente por completo, la señal). Cabe apuntar que los controladores SPI no necesitan resistencias de pull-up, ya que sus salidas son de tipo *push-pull* (salidas en contrafase), al contrario de lo que pasa en las salidas de colector abierto I²C. Resulta interesante concluir con una comparación práctica entre estos dos importantes y recurridos protocolos de comunicación (ambos útiles y populares para usar con dispositivos de memoria SRAM, flash o EEPROM). Si la prioridad recae en que la velocidad de transmisión sea alta (debe recordarse que el código puede afectar negativamente) la mejor opción es elegir SPI. Si el número de pines de que dispone el dispositivo a conectar es reducido, se prefiere usar I²C, lo mismo (generalmente) si el tamaño requerido del circuito (y número de pistas) no se ha de exceder. Finalmente, el consumo de energía de dispositivos I²C es, normalmente, mayor que el de los SPI.

XIII. Teorema de muestreo

How fast do you have to sample a 10 kHz signal to faithfully reproduce it?

En castellano: ¿cuán rápido tienes que muestrear una señal de 10 kHz para reproducirla fielmente?

Un parámetro relevante de los conversores AD es el tiempo (o período) de conversión (no confundir con el de muestreo), que se define como el tiempo que tarda el sistema desde que se ejecuta la orden de realizar una muestra hasta obtener un valor cuantificado. La inversa de la frecuencia de muestreo escogida ha de ser mayor que este tiempo, ya que, si no, se estaría ordenando lanzar una nueva conversión antes de que la anterior haya sido finalizada y podrían darse variaciones en la señal de entrada que introducirían ruido y, en definitiva, la conversión no funcionaría. Un período de conversión pequeño es beneficioso porque se traduce en un mayor número posible de muestras digitales. Debe de haber una forma práctica de determinar cuál es la frecuencia de muestreo adecuada que se puede tomar para cualquier convertidor en particular; si es alta, el manejo de la información se complica, pero al mismo tiempo se obtiene una imagen más representativa (a priori) de la señal original. Parece lógico adelantar que, en efecto, esta frecuencia puede variar según la aplicación de la conversión, pero se relatará brevemente la historia detrás de la regla general que rige este aspecto.

A mediados de 1920, Harry Nyquist se encontraba estudiando, con relación a la telegrafía, la máxima tasa de señalización de datos (DSR [bits/s]) que podía transmitirse y recuperarse por un canal con un ancho de banda W (capacidad máxima de transmisión) concreto. En un artículo de 1928 (*Certain Topics in Telegraph Transmission Theory*), definió una señal $s(t) = \sum_k a_k f(t-kT)$ particularizada para el telégrafo: con f(t) como una señal rectangular (pulso digital) de período T y a_k igual a 1 o 0 (amplitud de la señal rectangular). Basándose en la transformada de Fourier de este pulso rectangular ($\mathrm{sinc}(x) = \mathrm{sen}(x)/x$, *sinc* o seno cardinal), observó que, si la función está limitada superiormente por una frecuencia significativa, entonces la condición de muestrear "instantáneamente" la señal a intervalos regulares a una velocidad que sea el doble que esa frecuencia significativa, es suficiente para que las muestras contengan toda la información de la señal original. En otras palabras, señaló que la tasa de pulsos (definida por $1/T$) no podía ser incrementada a más de 2W pulsos por segundo para resolverlos de manera inequívoca (asimilable a la ulterior propuesta del teorema, que se referiría a resolver [o reconstruir] la señal en todas sus frecuencias).

No fue hasta 1949 (en *Communication in the Presence of Noise*) cuando Claude Elwood Shannon formuló una prueba formal de lo conjeturado años atrás por Nyquist (realmente se reconoce que Edmund Taylor Whittaker también descubrió el teorema, en 1915). Shannon estableció: si una función x(t) no contiene frecuencias mayores que "B" hercios, entonces queda completamente determinada tomando sus ordenadas en una serie de puntos espaciados por $1/(2B)$ segundos entre sí. Esto se puede condensar en la ecuación $f_s \geq 2 \cdot f_{máx}$, donde $f_{máx}$ representa la "frecuencia de Nyquist" (más concretamente, esta última es la mitad de f_s) y f_s es la frecuencia de muestreo (mínima en el caso de igualdad) teóricamente necesaria para

realizar el proceso de muestreo (AD) sin pérdida de información.

Una descripción matemática (que recuerda a lo comentado sobre la telegrafía) establece que toda la información de la señal analógica original puede estar contenida en una serie (suma en serie) de muestras que cumpla el teorema de Nyquist-Shannon. Cabe destacar que se puede "engañar" a este criterio, de cierta forma, en el caso del muestreo de algunas señales que no son muy densas (de banda ancha) en el espectro de frecuencias (compuestas por dos sinusoides de varios cientos de Hz...); hay interesantes teorías ("sensado comprimido") que prueban que, muestreando aleatoriamente a una frecuencia menor que la que dicta Nyquist-Shannon, se puede lograr reconstruir el espectro y luego, aplicando transformadas de Fourier inversas, la señal, sin indicios de aliasing.

Para ejemplificar todo esto y concluir el apartado: los humanos pueden detectar (generalmente) frecuencias de entre 20 y 20.000 Hz, por lo que para almacenar música en un CD (disco compacto), la señal de audio debería ser muestreada a una "tasa" de al menos 40.000 Hz, para reproducir (reconstruir) de nuevo la señal a una frecuencia máxima de 20 kHz (de hecho, usualmente, en los CD se hace a 44.1 kHz, cifra muy cercana al mínimo que establece el teorema de muestreo, debido [p. ej.] a los errores de cuantificación generados en el proceso). Por tanto, si se quisiera reproducir fielmente una señal de 10 kHz, una frecuencia de muestreo razonable sería de 20 kHz (con este ancho de banda tendría que contar el convertidor).

XIV. Conclusión

Llegado el final del documento, se puede comentar, sin lugar a errores que, a lo largo del mismo, el lector ha abordado una variedad muy amplia de conceptos y temas de interés científico, tecnológico e histórico, preponderando en todo momento la comprensión de la continuidad explicativa, de una forma ciertamente visual (o así se ha pretendido hacer), tanto por la parte escrita como por las figuras e imágenes (sobre las que "recae" un peso importantísimo).

Si se logra tomar cada apartado como un conjunto conexo de ideas y nociones (todas englobadas), la banda de interés del conocimiento se ensancha gratamente, ramificando de forma profunda las posibilidades por las que el lector puede extender la investigación alrededor de cualquiera de los campos ocupados a lo largo de este documento.

XV. Referencias y lecturas recomendadas

Alvin R. Lebeck, Randy Bryant, Dave O'Halloran, Gershon Kedem, *Computer Organization, Design and Programming*, «*Lecture 11 Finite State Machines & Hardware Control Language*», 2009.

David A. Patterson, John L. Hennessy, *Computer Organization and Design*, 4a ed., 2012, Elsevier Inc., *Morgan Kaufmann*, ISBN 978-0-12-374750-1.

M. M. Lorda, M. M. Hidalgo, Diccionario de Electrónica, Informática y Energía Nuclear, 1999, Ediciones Díaz de Santos, ISBN 9788479784119.

Sitios web: www.oscopes.info (Herbert Hoenle), www.hparchive.com, www.hpmemoryproject.org, www.oscilloscopemuseum.com, www.richardsears.wordpress.com, www.chiark.greenend.org.uk/scopes/tek.html, www.yoefk.com/c++fqa, www.gammon.com.au, www.electronics-tutorial.net, www.eli.thegreenplace.net, www.brokenthorn.com, www.zator.com.

Jonathan W. Valvano, *Embedded Systems: Introduction to ARM Cortex-M Microcontrollers*, 5a ed., 2012, CreateSpace Independent Publishing Platform, ISB 978-1477508992.

The Engineering Staff of Analog Devices Inc., Daniel H. Sheingold, *Analog-Digital Conversion Handbook*, 1986, Analog Devices/Prentice-Hall, ISBN 0-13-032848-0.

Hambley, Allan R., Electrónica, Prentice-Hall, 2000, ISBN 84-205-2999-0.

Elecia White, *Making Embedded Systems: Design Patterns for Great Software*, 2011, O'Reilly Media, ISBN 978-1449302146.

Joseph Louis Lagrange, *Lectures on Elementary Mathematics* (traducido del francés por Thomas J. McCormack), 2a ed., 1901, The Open Court Publishing Company.

Brian W. Kernighan, Dennis M. Ritchie, El Lenguaje de Programación C, 2a ed., 1991, Pearson, Prentice Hall, ISBN 968-880-205-0.

M. A. Pérez García, Instrumentación Electrónica, 2014, Ediciones Paraninfo S.A., ISBN 978-84-283-3702-1.

Wayne Tomasi, Sistemas de Comunicaciones Electrónicas, 4a ed., 2003, Pearson Educación, ISBN 0130221252.

Clyde F. Coombs, Jr., Electronic Instrument Handbook, 3a ed., 1999, McGraw-Hill Education, ISBN 0070126186.

Agilent Technologies, *Feeling Confortable with Logic Analyzers «Application Note 1337»*, 2006.

Tektronix, *The XYZs of Logic Analyzers «Application Note»*, 2001.

Nehemiah Hawkins, *Hawkins Electrical Guide* nº1-10, 1914-1917, Theodore Audel & Company.

René Rateau, "Osciloscopios, Funcionamiento y su Utilización", 1999, Ediciones Paraninfo S.A., ISBN 8428326274.

J. A. Gutiérrez Orozco (ESCOM), Máquinas de Estados Finitos «Breve Introducción», 2008.

DETCP, Universidad Politécnica de Cartagena, Prácticas de Circuitos y Funciones Electrónicas, 2019, Edicions UPC, ISBN 978-84-16325-91-7.

Thomas L. Floyd, Fundamentos de Sistemas Digitales, 9a ed., 2006, Pearson, Prentice Hall, ISBN 84-832-2085-7.

V. P. Nelson, H.T. Nagle, B. D. Carroll, J.D. Irwin, *Digital Logic Circuit Analysis and Design*, 1995, Pearson, Prentice Hall, ISBN 0134638948.

F. Aldana, R. Esparza, P. M. Martinez, Electrónica Industrial: Técnicas Digitales, 1980, Ed. Marcombo, S.A., ISBN 84-267-0487-5.

John G. Proakis , Comunicaciones digitales, 3a ed., McGraw-Hill Book Co., 1995, ISBN 0-07-113814-5.

H. Nyquist, *Certain Topics in Telegraph Transmission Theory*, en *Transactions of the American Institute of Electrical Engineers* Vol. 47, 1928.

C. E. Shannon, *The Bell System Technical Journal* Vol. 27 *«A Mathematical Theory of Communication»,* 1948.

C. E. Shannon, *Communication in the Presence of Noise*, en *Proceedings of the IRE* Vol. 37 (actualmente *Proceedings of the IEEE*), 1949.

Robert J. Marks II, *Introduction to Shannon Sampling and Interpolation Theory*, 1991, Springer-Verlag, New York, ISBN 0-387-97391-5.