# Lab 7: Timers

## 1. Objetivos

En experiencias pasadas, descubrió que es posible generar señales PWM mediante el uso de módulos *Timer* en sus microcontroladores. Ahora es momento de conocer en mayor profundidad estos elementos para programar rutinas con ejecuciones precisas, mediante la combinación de *Timers* e *Interrupts*.

Este laboratorio se dividirá en dos actividades fundamentales. Por un lado, conocerá cómo los *timers* permiten llevar a cabo operaciones concisas, mediante la programación de un contador con precisión de un decimal, implementado en un *display* de 7 segmentos. Por otro lado, realizará aplicaciones concretas de este módulo, más específicamente, un *debouncing* preciso de botón y el monitoreo del tiempo que un botón permanece presionado.

## 2. Descripción de la actividad

Este laboratorio se divide en 2 *Tasks* principales, las cuales podrán ser realizadas en códigos distintos. Si decide implementarlas en un mismo código, deberá incorporar algún tipo de menú para escoger la tarea a desplegar, por ejemplo con LEDS y botón, por USART, entre otros. Cabe destacar que esto NO significará un motivo de distinción.

Importante

Ambas tareas pueden realizarse en un mismo microcontrolador. No obstante, si decide implementar cada *Task* en un microcontrolador distinto, **puede optar a distinguido**. Esto como incentivo por volver a utilizar ambos microcontroladores.

### 2.1. *Task* 1: Contador 7 segmentos

Para esta actividad deberá programar un contador en el módulo 7 segmentos que se le ha entregado en su kit de SEP. Haciendo uso de *timers* deberá implementar un contador que inicie en 00.0 y llegue hasta 99.9 segundos (es decir, cambios cada 100ms). Estas cifras deben visualizarse en su *display* 7 segmentos, **diferenciando la parte entera de la decimal mediante el punto**. Una vez que el contador llegue a 100 (condición de *overflow*), la cuenta deberá reiniciarse, volviendo a contar desde 0. Su contador debe tener una precisión de 100 milisegundos, no obstante se permitirá un margen de error de máximo 5 segundos al

llegar a 99.9s.

Dado que en la experiencia pasada se ha familiarizado con el uso de interrupciones, deberá hacer uso de interrupciones de *timer* para esta actividad, por lo cual su bucle de `while(1)` **deberá estar vacío**.

**Consideraciones:**

- Si ya logró hacer funcionar el *display* con más de un segmento en experiencias anteriores, la ejecución de esta actividad se trivializa bastante. Si aún no lo ha hecho, piense cómo poder 'independizar' los segmentos (viendo el *datasheet* podrá percatarse de este problema).
- Las rutinas `switch/case` que probablemente utilizó en los laboratorios de USART pueden resultar de gran utilidad para simplificar el código (y facilitar la corrección) si se usan apropiadamente.

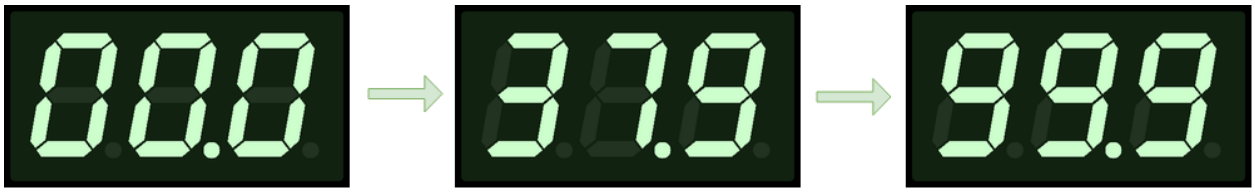


Figura 1: Ejemplo de forma de desplegar la información en Task 1

## 2.2. *Task* 2: Aplicaciones con *timers*

Esta actividad consiste en visualizar algunas de las aplicaciones que tienen los *timers* y se subdivide en dos *sub-tasks*. La división de estas *sub-tasks* es solo para para ayudarle al desarrollo de cada una, sin embargo deben ser realizadas en **un solo código**.

### 2.2.1. *Task* 2.1: Contador y *debouncing* de botón de aluminio

Esta actividad consistirá en recrear parte de lo realizado en el laboratorio 2, en donde se utilizaron *delays* para realizar un debouncer de un botón de aluminio. En esta ocasión, deberá utilizar una interrupción de *timer* configurada a una frecuencia determinada por usted, que permita evitar los *glitches* producidos al tocar ambos trozos del papel. Para detectar que el botón fue presionado o soltado, debe volver a recurrir a las interrupciones de pin utilizadas en el laboratorio anterior y, en conjunto con las interrupciones de *Timers*, realizar el *debouncing* correspondiente.

Para chequear que el *debouncing* fue realizado correctamente, cada vez que presione el botón de aluminio deberá cambiar de color su LED RGB, siguiendo la secuencia

`rojo - verde - azul - rojo - verde - azul - rojo (...)`

Este cambio debe ocurrir **en el instante en que se presiona el botón**, no antes, no después ni tampoco luego de un 'pequeño *delay*'. Además, al soltar el botón o al mantenerlo presionado no debe ocurrir ningún cambio en el RGB.

### 2.2.2. *Task* 2.2: Contador de botón

Esta actividad consistirá en la implementación de un contador, el cual indique por cuánto tiempo se ha mantenido presionado el botón de usuario de su MCU. La cifra en cuestión deberá ser presentada en el *display* de 7 segmentos, teniendo en cuenta las siguientes consideraciones:

- Por un lado, si el botón ha sido presionado por menos de 1 segundo, el formato de tiempo mostrado en el *display* deberá ser de tipo:

Por consideraciones del display, el punto lo ubicaremos al **costado derecho**.

- Por otro lado, si el botón ha sido presionado por más de 1 segundo, el formato de tiempo mostrado en el *display* deberá ser de tipo:


En esta parte, el punto se ubicará al costado derecho de la unidad, perdiendo resolución de milisegundos.

En otras palabras, **su contador deberá tener una precisión de 1ms**, y el tiempo que el botón es presionado debe estar en el rango de 0 a 9.99 segundos. En caso de que el tiempo que el botón es presionado por un tiempo fuera de ese rango, considere señalarlo mediante algún mensaje de error: LED, mensaje USART, escribir `Err` en el *display*, entre otros (esto queda a su elección).

Cabe señalar que para el tiempo a desplegar en el display, puede mostrar cómo aumenta mientras el botón se mantiene presionado o mostrar el valor solamente al soltar el botón. Sin embargo, es importante que este número se mantenga visible para poder observarlo una vez suelto el botón. Para resetearlo, puede ser al volver a presionar el botón, o tener un botón de reset de este valor.

# 3. Lecturas recomendadas

- ATmega328/P Complete Datasheet.
- MSP430x5xx and MSP430x6xx Family User's Guide.
- MSP430F552x, MSP430F551x Mixed-Signal Microcontrollers datasheet.
- Datasheet del *display* 7 segmentos HDSP-431G/433G.

# 4. Pauta de Evaluación

## 4.1. Consideraciones generales

- El laboratorio será evaluado exclusivamente con nota 1.0 (**R**eprobado), 4.0 (**S**uficiente), 5.5 (**A**probado) y 7.0 (**D**istinguido). En ningún caso habrán notas intermedias.
- No se reciben trabajos después del módulo de presentación. Trabajos no entregados son calificados con nota 1.0 y son considerados dentro del criterio de aprobación del curso. La hora límite para inscribir a revisión es a las 10:10 hrs, posterior a esto se asignará una posición aleatoria.
- La nota **S**uficiente se otorgará en el caso de falla de una de las tareas de este laboratorio, quedando a criterio del ayudante. En caso de que un alumno haya decidido solamente hacer un 50 % del trabajo, se evaluará con un 1.0.
- Respecto a los puntos de aprobación acumulados, si un alumno obtiene una nota **S**uficiente, una mitad del puntaje queda asignada a Aprobación, el restante a Reprobación.

  A modo de ejemplo, este Laboratorio es nivel 4, si un alumno tiene Suficiente, 20 pts se acumularán a Aprobación y 20 pts será para Reprobación.
- Cualquier consulta sobre los criterios de evaluación de cada laboratorio debe ser realizada en las issues, donde estará disponible para que sea revisada por todos los alumnos.

## 4.2. Criterios de Aprobación

**Se requiere cumplir con todos los puntos mencionados a continuación para poder aprobar. No existen casos excepcionales.**

1. Funcionamiento de los requerimientos. El alumno realiza una presentación de su trabajo y se responsabiliza de exponer que su trabajo satisfaga todos los requerimientos mínimos solicitados en la *Descripción de la actividad*, los cuales incluyen en este laboratorio:

   - [**Común a ambas task**] Actualiza el display de 7 segmentos a una frecuencia suficiente para dar la sensación de que se encuentran los 3 números activos al mismo tiempo (HINT: una frecuencia de aproximadamente 120 Hz debería ser suficiente.)
   - **Task 1:**
     - Realiza un contador de 0 a 99.9, incluyendo punto, puede mostrar los números del 0 al 9 como 0x.x (no es necesario apagar el primer anodo).
     - El cambio ocurre a aproximadamente 100 ms, se permitirá cierto margen de error (máx 5 seg al llegar a 99.9 seg en su contador).
   - **Task 2:**
     - Las condiciones de debouncing son las mismas para el laboratorio 2, no se acepta el uso de funciones nativas de delays para realizar esto ni soluciones por hadware.
     - El cambio en el led RGB debe ser al momento de tocar el papel de aluminio, no antes, no después ni levemente después (con un delay). Tampoco debe cambiar cuando se mantenga presionado.
     - El contador de tiempo debe mostrarse en el display de 7 segmentos utilizando la estructura mencionada.
     - Debe detectarse el botón mediante interrupciones de pin.

2. Preguntas. Se responde satisfactoriamente a 2 de 3 preguntas aleatorias al momento de la presentación final, las cuales abarcan los siguientes temas:

   - Qué representa cada línea de código y en qué se traducen en el funcionamiento del programa.
   - Funcionamiento del sistema de interrupciones de timers del microcontrolador especificado. Teoría de operación, registros utilizados, etc

- Beneficios de utilizar Timers e interrupciones de timers en el contexto de sistemas embebidos. ¿Por qué son importantes ?
- ¿Qué son las variables de tipo `static` y `volatile` ?

Solo se dispone de una oportunidad para responder estas preguntas. Fallar en este requisito se traduce en la reprobación inmediata de la experiencia de forma inapelable.

## 4.3. Criterios de Distinción

La distinción representa un trabajo adicional que sobresale a los requerimientos mínimos para la aprobación. Agregados adicionales no constituyen por sí mismo una distinción si no representan un verdadero trabajo adicional de comprensión y/o análisis.

Los trabajos distinguidos pueden caer (no exclusivamente) en algunas de las siguientes líneas generales:

- Funcionalidades Creativas :D
- Funcionalidades adicionales, en la línea de I2C, SPI, EEPROM, entre otros.
- Realizar Task 1 en un uC, realizar Task 2 en otro uC. Es hora de volver a ocupar ambos microcontroladores :)
- **Portabilidad de código**: Un código es portable si el código fuente en C del laboratorio puede ser compilado y cargado en cualquiera de los microcontroladores del curso de forma indistinta, **sin hacer ninguna modificación a dicho código**[1].
- Realizar ambas task en un sólo código no es motivo de distinción.

**Las Distinciones son discutidas caso a caso por la totalidad del equipo de ayudantes al finalizar la corrección del laboratorio. Serán notificadas públicamente después del módulo de evaluación.**

[1] Hint