Lab 3: Timers e Interrupciones

Laboratorio Nivel 2

1. Introducción

Este laboratorio busca introducir a los estudiantes con nuevas capacidades que tienen los microcontroladores. En primer lugar se tienen los Timers que dentro de sus distintas capacidades, permiten la generación de señales PWM y la ejecución de rutinas precisas que ocurren a intervalos de tiempo regulables. Junto con lo anterior, se incorpora el concepto de interrupciones para la utilización de Timers y para la detección de entradas externas mediante puertos digitales.

Para esta actividad se hará uso del *buzzer* y el botón del *joystick* del Booster Pack MKII, junto con los botones integrados del MSP430F5529LP para tener un sistema que permita la reproducción de melodías cortas.

Para esto, se dividirá en tres tareas que irán agregándole nuevas funcionalidades al programa, por lo que se recomienda realizarlas en orden y siguiendo las sugerencias indicadas.

2. Descripción de la actividad

La actividad se dividará en tres sub actividades en las que solo se deberá entregar la última. La división de sub actividades es para guiar el trabajo. A modo común de todas las distintas Task, se considerá el uso del siguiente *buzzer*:

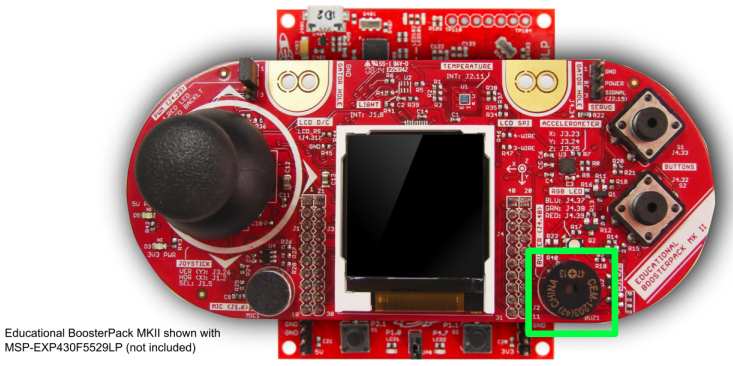

Figura 1: *Buzzer* a utilizar.

Además, se referirá a los botones de usuario de la siguiente forma:

- BTN PLAY/PAUSE: Botón incorporado en Joystick del Booster Pack.
- BTN IZQ: Botón de usuario izquierdo del MSP430F5529LP.
- BTN DER: Botón de usuario derecho del MSP430F5529LP.

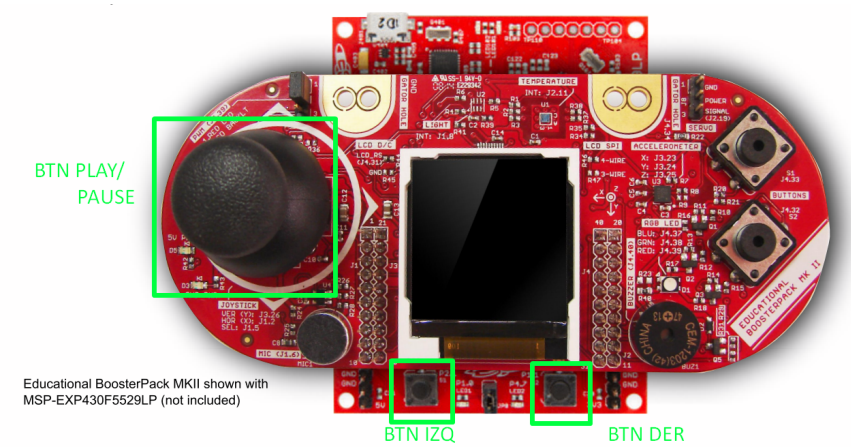

Figura 2: Botones

2.1. *Task 1*: *Buzzer* y botón de Stop (66 %)

Un *buzzer*, es un transductor electroacústico que permite convertir señales eléctricas en sonido, existen de distintos tipos y capacidades que no se entrará en detalle. En el caso del Booster Pack MKII, contiene un *buzzer* modelo CUI CEM-1203[1] que permite generar distintas frecuencias, en función de la frecuencia de una señal PWM.

Esta posibilidad de variar frecuencias permite la creación de pequeñas melodías al seleccionar la duración y tono específico, puede ver un ejemplo de esto aquí y aquí.

El objetivo de este *task* es implementar una biblioteca[2] específica para este *buzzer*, para ser utilizada en el MSP430 mediante el uso de interrupciones de timers.

Esta biblioteca, deberá ser instanciada por un programa principal, que permita la generación de pequeñas melodías y que además, se le agregarán funcionalidades adicionales mediante botones.

[1] https://www.cuidevices.com/product/audio/buzzers/audio-transducers/cem-1203(42)

[2] Que los terrícolas a veces llamamos librerías.

En este *task*, el botón a utilizar es el BTN IZQ, que tendrá la funcionalidad de ser un botón de STOP / RESET. Es decir, que cuando sea presionado, la melodía deberá detenerse, esperar un tiempo y volver a reproducir desde un comienzo.

Debe cumplir los siguientes requisitos:

- Crear y completar los archivos `buzzer.h` y `buzzer.c`.
- Tener una función `beep` que tenga la siguiente estructura: `beep ( frequencyInHertz, timeInMilliseconds, pausetimeInMilliseconds);`. Esta emitirá un tono a una duración `timeInMilliseconds` y esperará `pausetimeInMilliseconds`, antes de la siguiente nota.
- Tener una función `noBeep` que tenga la siguiente estructura `noBeep(time_ms);`. Esta mantendrá un silencio por `time_ms`.
- Reproducir al menos una melodía. Al finalizar, debe esperar un pequeño tiempo y volver a reproducirse.
- El BTN IZQ tiene que estar implementado mediante **interrupciones de pin**, en este se verá enfrentado al efecto de *bouncing* (ya estudiado en el laboratorio anterior), por lo que deberá realizar un *debouncing* por *hardware*, pudiendo ser con delays o *timers*.
- Se debe poder detener la melodía en cualquier minuto y en cualquier instante. Deberá volver a comenzar luego de un pequeño intervalo en silencio.

2.1.1. Consideraciones y recomendaciones *Task* 1

Esta biblioteca debe ser posible utilizarla solo al incorporarla y entregarle las configuraciones pertinentes. Esto debido a que será utilizada en futuros laboratorios. La elección y modo de los *timers* queda a libre elección; además, la estructura entregada es solo una sugerencia, por lo que son libres de realizarla de forma distinta.

2.2. *Task 2*: Mejorar eficiencia de botones y más melodías. (20 %)

A partir de lo ya construido en el *Task* 1, en esta actividad deberá incorporar la capacidad de reproducir 4 melodías. Además, con el BTN DER, que al ser presionado permitirá cambiar entre las distintas melodías. Debe considerar lo siguiente:

- El BTN DER, debe ser implementado mediante interrupciones de pin.
- Tanto el BTN IZQ como el BTN DER, deberá ser mejorado su *debouncing* por *software*, siendo ahora implementado con interrupciones de *timers*. No se permitirá utilizar __delay_cycles() para este propósito.
- Al cambiar entre melodías, esto solo debe ocurrir al momento de presionar el BTN DER, si es que se mantiene presionado, no se debe seguir avanzando indefinidamente.
- Al llegar a la última melodía, si se presiona el botón o termina la melodía, se debe volver a la primera y repetir el ciclo.

2.3. *Task 3*: Incorporar botón Play/Pause (14 %)

A partir de lo construido en el *Task* 2, en esta actividad se deberá incorporar la capacidad de tener un botón que actúe como Play / Pause, para cualquiera de las melodías que se esté reproduciendo. Las consideraciones son los siguientes:

- El BTN PLAY / PAUSE, debe ser implementado mediante interrupciones de pin. En caso de requerir *debouncing* por *software*, este deberá ser realizado con interrupciones de *timers*.
- La utilización de este botón no debe entorpecer el actuar de los demás botones, por lo que se podrían producir alguna de las siguientes situaciones:
  - Una melodía está reproduciéndose y se presiona BTN PAUSE. A continuación se presiona BTN DER (cambio de melodía). Lo que debe ocurrir es que se pasa a la siguiente melodía, pero no se comienza a reproducir, ya que se está en pausa. Si se vuelve a presionar el BTN PAUSE, se comenzará a reproducir la melodía, pero desde el **principio**, ya que se trata de una melodía nueva.
  - Una melodía está reproduciéndose y se presiona BTN PAUSE. A continuación, se presiona BTN IZQ (STOP). Lo que debe ocurrir es que se vuelve al comienzo de la melodía, pero no se comienza a reproducir, ya que se sigue estando en pausa. Para iniciar la reproducción, se debe volver a presionar el BTN PAUSE.

2.4. Consideraciones

Lo siguiente, es una adaptación de lo ya presentado en el LAB 02, pero enfocado a este laboratorio.

2.4.1. Debouncing por software

Para el caso de los BTN IZQ y BTN DER experimentará el efecto de *bounce*, por lo que deberá diseñar mediante *software* una rutina que se encargue del *debouncing* de dicho botón. En el caso del botón del Joystick, este efecto podría ser menor, debido a que contiene un *debouncing* por *hardware*. Sin embargo, de requerirlo, deberá incorporar también *debouncing* en este botón.

Lo que se pide en esta actividad es que primero configure los BTN IZQ y DER del MSP430F5529LP para que, cuando sea presionado ejecuten alguna acción relacionada con la reproducción de melodías.

Su programa debe contemplar que mientras alguno de los tres botones se encuentre presionado, su acción no debe entorpecer a otro. Además, considerar aquellos casos en donde la acción de un botón afecta las decisiones del otro.

IMPORTANTE. La acción solo debe ocurrir en el instante en que el botón es presionado, no antes, no después, no después de un pequeño *delay*, no cuando es liberado ni tampoco cuando se mantiene presionado.

En la figura 3, se muestra lo que sucede cuando se presiona un botón [3] muestra lo que ocurre cuando un botón es presionado.

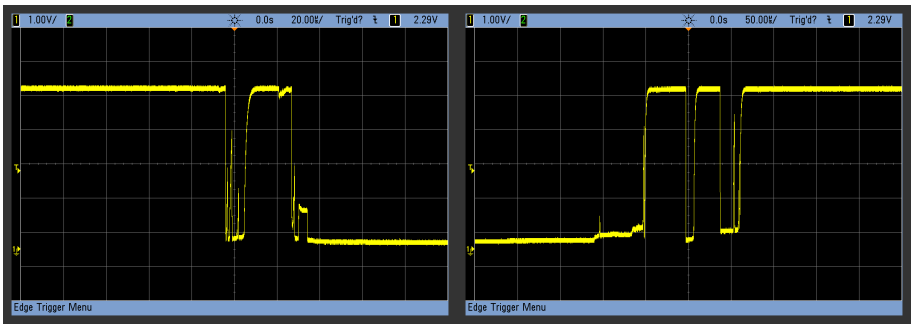

Figura 3: *Bouncing* de un botón

[3] Fuente imagen: Hackaday https://hackaday.com/2015/12/09/embed-with-elliot-debounce-your-noisy-buttons-part-i/

Por otra parte, la figura 4 muestra la lectura que se espera:

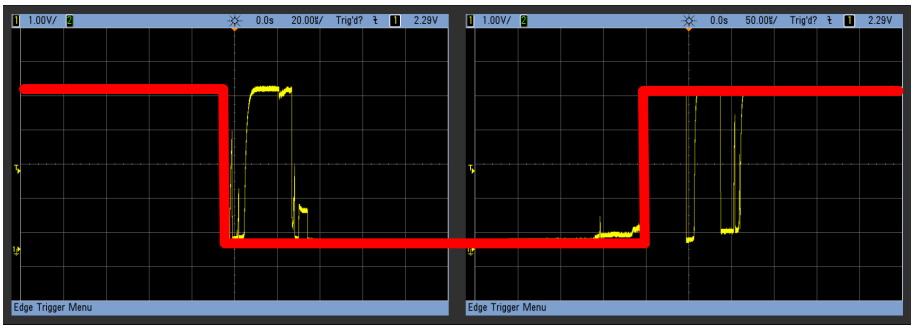

Figura 4: Lectura ideal esperada

2.4.2. Otras consideraciones

- Es importante notar que al estar configurando pines como entradas, es necesario utilizar resistencias de *pull-up*. Estas vienen en general dentro de la tarjeta o se pueden utilizar resistencias externas, pero para este laboratorio es un requerimiento que usen las internas.
- Deben definir si es que utilizarán resistencias de *pull-up* o *pull-down* y si es que es posible realizar esto o no.
- El proceso de *debouncing* tiene que ser realizado mediantes interrupciones de pin e interrupciones de *timers*. A excepción del *Task* 1, en donde el *debouncing*g puede realizarse sin interrupciones de *timers*.

2.5. Recomendaciones

Este *Task*, al tener que realizar múltiples acciones con distintas frecuencias y en distintos momentos, recomendamos lo siguiente:

- Comenzar identificando los puertos a utilizar y dejarlos anotado como comentario en una parte del código. Ayudará a la memoria y recordar cuál es cuál.
- Determinar frecuencias, velocidades, número de bits, modo y que registros de comparación se utilizarán para cada uno de los *timers*.
- Posiblemente, requiera utilizar los 3 *timers* del MSP430.

- Realizar el *debouncing* de cada botón por separado, usando interrupciones de pin y *timers*. Para esto un buen código de inicio sería encender y apagar un led, cada vez que el botón se presiona. Esto ayudará a corroborar que si se mantiene presionado el botón, no hay cambios; y si se suelta, tampoco los habrán. Luego incorporar un código sencillo que tenga los tres botones (o dos) a medida que se van añadiendo a las acciones de *buzzer*. Una buena estructuración puede hacer que el código sea mucho más limpio.
- Investigue acerca de las variables de tipo `volatile` y `static`. Puede que sean útiles a la hora de controlar información entre interrupciones y funciones.
- No sobrecargue de instrucciones a las interrupciones. Las interrupciones deben ser acciones pequeñas que cambien estados del programa, no se espera que todo su programa esté en las interrupciones.
- El modo *debug* de Code Composer Studio es muy útil para monitorear cada línea del código. Se pueden seleccionar breakpoints para monitorear las interrupciones y llegar a esas líneas cuando se ejecuta una acción.
- Revisar y estudiar el Datasheet. Contiene información relevante para la realización del laboratorio.
- Utilizar máquinas de estado. Al igual que el laboratorio anterior, serpa muy necesario.

3. Melodías

Se publicarán algunas melodías a utilizar en una Issue en Github. Esto será considerado como trabajo compartido, por lo que pueden publicar y añadir sus propias melodías y escoger aquellas que quieren incorporar en su código.

4. Lectura recomendada

- Páginas iniciales del Datasheet del MSP430F5529
- Capitulo 12: Digital I/O Module del MSP430x5xx and MSP430x6xx Family User's Guide.
- Capitulo 17 y 18: Timer A/B del MSP430x5xx and MSP430x6xx Family User's Guide.
- *Buzzer* `https://www.cuidevices.com/product/audio/buzzers/audio-transducers/cem-1203(42)`
- MSP430x5xx and MSP430x6xx Family User's Guide.
- MSP430F552x, MSP430F551x Mixed-Signal Microcontrollers datasheet.
- BOOSTXL-EDUMKI User Guide `https://www.ti.com/lit/ug/slau599a/slau599a.pdf`

5. Sesión de preguntas

5.1. Consideraciones generales

- Debe cumplir con el horario asignado y solamente tendrá aproximadamente 10 min para realizar la presentación. Se pide encarecidamente respetar esta regla.
- 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.
- Puede utilizar cualquier recurso que tenga disponible. Ya sea un block de notas, mostrar brevemente alguna parte de su código, tener una presentación de apoyo, el Datasheet, etc. Queda a criterio del ayudante corrector la solicitud de mostrar este apoyo.
- IMPORTANTE: se prohibirá el uso de funciones de Energia.nu para la programación de las tarjetas de desarrollo, esto por la simplicidad que involucra, por lo que deberán mostrar que entienden qué están realizando al momento de programar.

5.2. Preguntas

Responde satisfactoriamente preguntas aleatorias que abordan los siguientes temas:

- Especificaciones generales del MSP430F5529 en lo que respecta a *inputs* y *outpus*. Tales como existencia de resistencias internas, voltajes y corrientes máximas admitidas. Enfocado, al uso de interrupciones.
- Especificaciones generales del MSP430F5529 en lo que respecta a *timers*. Tales como frecuencias admitidas, elección de modos, cantidad de contadores, cantidad de *timers*, etc.
- Uso de variables de tipo *volatile* y *static*.
- Conceptos iniciales del tiempo del procesador y el significado o generación de *delays()*.
- Comprender **cada línea** de su programa.
- Entendimiento del concepto de *debouncing* por *software* y su diferencia por *hardware*.
- Entendimiento de *Headers Files* o archivos de extensión .h. ¿Cuál es el beneficio que tiene la creación de bibliotecas y cómo ayuda a la estructuración de código ?
- ¿Qué significan las directivas #IFNDEF #DEFINE #ENDIF ?

6. Fecha y forma de de entrega

La entrega se debe realizar el día Lunes 19 de octubre las 9:59 hrs. Durante este día se realizará la sesión de preguntas en los horarios por publicar.

Deberá subir su **código completo y ordenado** a su repositorio de GitHub junto con un archivo Readme.md que indique las fuentes utilizadas y una lista de lo que no fue implementado en su código. De forma que el ayudante corrector lo tenga claro. Además debe indicar sistema operativo usado y/o si utilizó un compilador online.

Conectarse en el horario asignado para la ronda de *preguntas y respuestas*

El código entregado debe cumplir con lo siguiente, el no cumplimiento de estos puntos podría incurrir en un descuento:

- Programa compilable y sin errores. Recomendamos subir el proyecto completo generado en Code Composer Studio.
- Código ordenado y debidamente comentado. No se exigirá ningún estándar en particular, pero se debe mantener una estructura que permita una lectura fácil. [4][5]
- Recursos utilizados se deben citar en el README.md de su repositorio.
- Su código debe tener un encabezado como el mostrado en el ejemplo de "Hello World!" del LAB 01.

[4] En https://developer.gnome.org/programming-guidelines/stable/c-coding-style.html.en pueden encontrar ejemplos de cómo mantener un buen formato de código.

[5] En https://codebeautify.org/c-formatter-beautifier pueden reformatear un código.