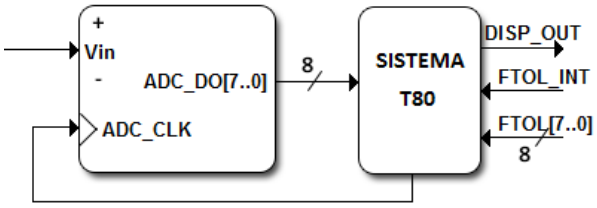
• Nombre y CI en cada hoja • Numere las hojas, indique el total en la primera • Utilice sólo un lado de las hojas	• Incluya un solo problema por hoja • **Sea prolijo** • **Aprobación:** mínimo UN problema

PROBLEMA 1

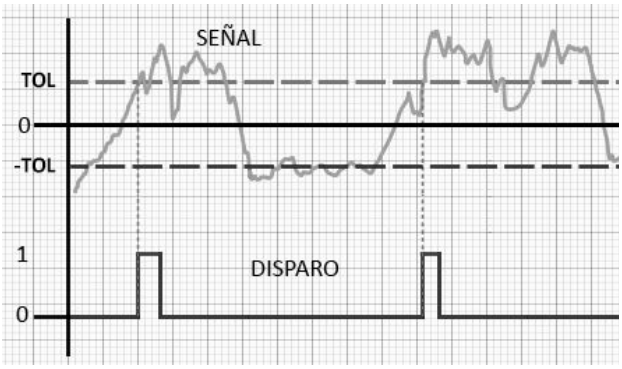


Se quiere agregar un **ADC** (conversor analógico-digital) a un sistema **T80** existente, para detectar cruces por cero de una señal, trabajando por interrupciones en **modo 2**. Cuando se genera un flanco de subida en la entrada **ADC_CLK** del **ADC**, éste muestrea la señal presente en **Vin** e inmediatamente coloca el dato digitalizado (en complemento a 2) en su salida **ADC_DO[7..0],** el cual se mantiene estable hasta el siguiente flanco de subida de la señal **ADC_CLK**. Este dato debe ser leído por el **T80**.

La señal **ADC_CLK** deberá estar conectada a la salida ZC de un bloque **TIMER**, el cual debe generar un flanco de subida cada **1 ms** o **10 ms** (según se especifica más adelante). Además el TIMER debe interrumpir al **T80** para comunicar que hay un nuevo dato disponible (con la prioridad más alta). La frecuencia de reloj del sistema es **fclk = 40,96 MHz (40960 kHz)**. Notar que 1 ms = 4096 x 10 x 1/fclk.

Por otro lado, una señal llamada **FTOL_INT** deberá interrumpir al **T80** (con prioridad siguiente a la del **TIMER**) para notificar que en el bus **FTOL[7..0]** se encuentra disponible un nuevo byte. De este byte, los 4 bits menos significativos representan un entero sin signo, cuyo valor define una tolerancia al ruido, la cual llamaremos **TOL**; y el bit más significativo especifica si el período de muestreo es **1 ms** (cuando **FTOL[7] = 0**) o **10 ms** (cuando **FTOL[7] = 1**) .

Se considera **DISPARO** el instante en que la señal con pendiente positiva atraviesa el valor **+TOL**, *habiendo atravesado previamente* el valor **-TOL**, como se muestra en la siguiente figura:



Observar que luego del primer disparo, un nuevo cruce de pendiente positiva por **+TOL** no genera otro disparo, ya que la señal todavía no cruzó **-TOL** *luego* de haberse detectado el primer disparo. Se garantiza que el valor de la señal digitalizada se mantiene dentro del intervalo **[- 100, + 100]**, lo cual asegura que **no** puede producirse overflow al comparar la señal con la tolerancia.

Cuando es detectada una condición de **DISPARO**, se debe dar un pulso corto a **1** en la señal de salida **DISP_OUT** y se debe setear en **0xFF** una bandera en la dirección de memoria reservada **DISP_FLAG.**

El programa principal (ya implementado) está realizando otras funciones y utiliza los primeros **4kB** de un total de **16kB** de **ROM** (0x0000 – 0x3FFF); los primeros **4kB** de un total de **16kB** de **RAM** (0x4000 – 0x7FFF), y los cuatro primeros lugares de la tabla de interrupciones. Además se encarga de inicializar el SP, y de configurar y habilitar el sistema de interrupciones con **I = 0x70**.

El rango de direcciones disponible para los puertos de entrada/salida es de **0x20** hasta **0x3F**.

Se pide:

a) Diseñar todo el hardware necesario para la detección de disparos: **TIMER**, controladores de interrupciones, salida **DISP_OUT**, manejo del **ADC**, puertos relacionados con **FTOL** y la decodificación correspondiente. No se pide implementar la **ROM** ni la **RAM**.

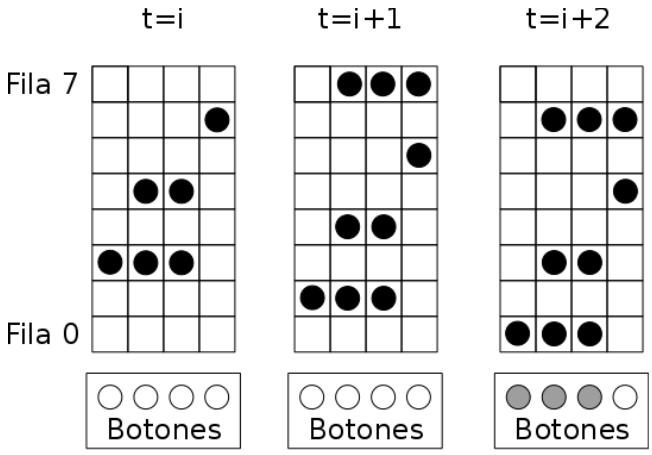
b) Escribir las rutinas de atención a las interrupciones de **FTOL_INT** y de **TIMER** (en esta última es donde debe hacerse la detección de disparos).

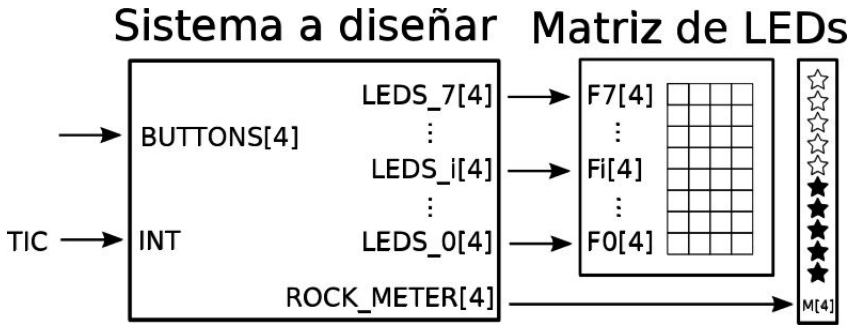
c) Escribir una rutina **INIT_DET_DISP**, que se invocará desde el programa principal, que inicialice todo lo necesario (periféricos, variables, entradas en tabla interrupciones, etc.) para que la detección de disparos funcione correctamente. Inicialmente **TOL**=0xF y muestreo cada 1 mseg.

Utilizar directivas para indicar la ubicación en memoria de todo el código, constantes y variables usadas.

PROBLEMA 2

Se busca diseñar un sistema basado en Z80 que implemente una versión simplificada del juego "*Guitar Hero*". En este juego el jugador debe presionar los botones de su controlador (que representan las cuerdas de una guitarra) en el instante exacto de forma de sincronizar las cuerdas tocadas con los acordes que se muestran en pantalla. Los acordes se van moviendo en la pantalla con una cadencia fija, y el jugador tiene como objetivo acertar la mayor cantidad de acordes.





En nuestra versión tendremos solo 4 botones (4 cuerdas) y una matriz (4 x 8) de LEDs como pantalla. Un acorde se representa en una fila de 4 LEDs. Los acordes se desplazan hacia abajo con el avance del tiempo. El jugador debe presionar los botones que se correspondan con el acorde desplegado en la fila de más abajo (**fila 0**) y mantenerlos presionados hasta que el acorde se "caiga" de la matriz. Al tocar el acorde correcto, el jugador sumará puntos en su **ROCK_METER**. Si equivoca el acorde, o pierde de tocarlo a tiempo, restará puntos. Inicialmente la matriz se despliega con todos los LEDs apagados, y los próximos acordes a tocar se irán insertando por la fila de más arriba (**fila 7**).

Los acordes se irán generando a partir de una subrutina **RANDOM ya existente** que devuelve una palabra aleatoria de 8 bits en el acumulador. Se dejará **siempre un silencio (acorde 0000) entre acordes aleatorios sucesivos**.

La matriz de LEDs tiene 8 entradas **F7[4], …, Fi[4], …, F0[4]** que encienden o apagan los LEDs de cada fila (1=ON, 0=OFF).

El **ROCK_METER** se mostrará al jugador a través de un display tipo “termómetro” que acepta un entero sin signo de 4 bits en **M[4]** y enciende ese mismo número de estrellas.

El avance del juego está dado por una señal periódica **TIC** que interrumpe al sistema. En cada interrupción se debe leer el estado de los botones y actualizar el estado del juego:

- Si el jugador tocó el acorde correcto sumará 1 punto a su **ROCK_METER**
- SI el jugador tocó el acorde incorrecto restará 1 punto a su **ROCK_METER**
- Si **ROCK_METER = 10d** el jugador habrá ganado la partida y el juego termina.
- Si **ROCK_METER = 0d**, el jugador habrá perdido la partida y el juego termina.
- Si el juego termina, todos los LEDs **deben permanecer como estaban hasta un nuevo reset del sistema**.

El programa principal es un loop infinito que realiza otras tareas no descritas aquí y utiliza la mitad más baja del espacio de E/S. Cuando finaliza el juego el programa principal debe poder continuar hasta la llegada de reset. **TIC** es la única interrupción del sistema.

Se pide:

a) Hardware completo del sistema basado en un Z80 tradicional (buses triestado) con interrupciones en modo 1. Se dispone de chips de 16KB de ROM y RAM.
b) Rutina de atención a la interrupción **TIC** que hace avanzar el juego.
c) Reserva de memoria para variables y código de inicialización luego de un reset. El valor inicial de **ROCK_METER** debe ser 5.