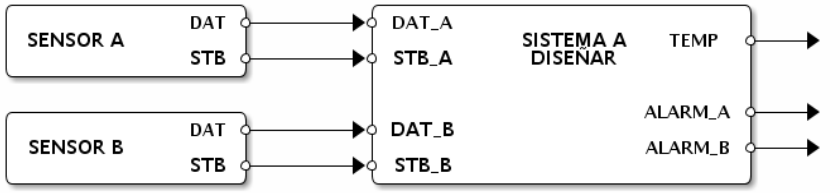
• Nombre y CI en cada hoja • Numere las hojas, indique el total en la primera • Utilice sólo un lado de las hojas	• Incluya un solo problema por hoja • **Sea prolijo** • **Aprobación:** mínimo UN problema

PROBLEMA 1



Se desea diseñar un sistema capaz de monitorear la temperatura de un equipo. Con el objetivo de mitigar posibles fallas, el equipo a monitorear cuenta con dos sensores de temperatura (**SENSOR_A** y **SENSOR_B**) de manera de ofrecer redundancia en la medida.
Cada sensor avisa que tiene una nueva medida dando un pulso a "1" de corta duración en su salida **STB** y poniendo el valor de la medida en su salida **DAT**. **DAT** es un valor binario sin signo de 8 bits, siempre menor a 128, y solamente es válido en el flanco de subida del pulso de **STB**.

En funcionamiento normal, una instancia del proceso de medición consiste en ambos sensores presentando sus medidas en forma más o menos simultánea, con una diferencia no mayor a un tiempo **DELTA_T.** Dos medidas consecutivas siempre están separadas un tiempo mucho mayor a **DELTA_T**.

Cuando llega la medida de uno de los sensores, el sistema debe leer el dato del mismo y esperar por la medida del otro sensor. Si la medida del segundo sensor llega antes de transcurrido el tiempo **DELTA_T**, el sistema leerá el dato del segundo sensor, calculará el promedio de las medidas de ambos sensores para escribirlo en la salida **TEMP** y además desactivará la **alarma** que se describe más adelante.
En cambio, si después de la medida del primer sensor transcurre **DELTA_T** sin que haya llegado la medida del segundo sensor, se deberá escribir en **TEMP** el valor de la primera (y única) medida recibida, y además se activará la salida de alarma correspondiente al sensor que no llegó a tiempo (**ALARM_A = 1** o **ALARM_B = 1**)

Se garantiza que:
- un mismo sensor nunca enviará dos medidas sucesivas en un tiempo menor a **DELTA_T**.
- en una instancia del proceso de medición, la medida del segundo sensor llegará antes de transcurrido **DELTA_T** a partir de la medida del primero o no lo llegará nunca.

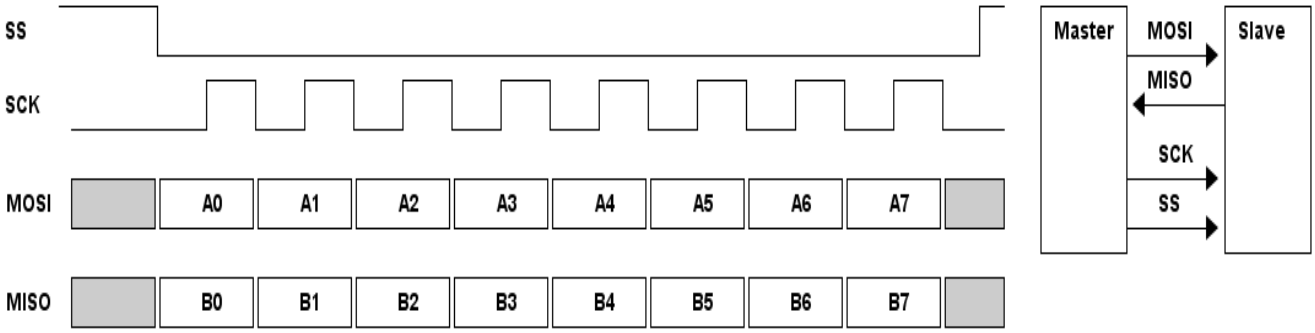
La solución se debe organizar en base a un programa principal que monitorea los sensores por pollling y maneja la salida **TEMP**, y un timer que interrumpe para medir el tiempo **DELTA_T**. Diseñar el sistema de monitoreo con un procesador Z80 compatible trabajando en modo 2 de interrupciones. El sistema contará con otros periféricos no especificados para los cuales se ha reservado la mitad ***0x00-0x7F*** del espacio de E/S y los dos primeros vectores de interrupción.
El sistema funcionará con un reloj de **Fclk= 4MHz**

Se pide:
a) Hardware completo del sistema, incluyendo conexión de periféricos utilizados (timers, controladores de interrupción) y puertos.
b) Loop de programa principal de acuerdo a lo especificado arriba. Debe incluir una llamada a subrutina atiendo_otros que realiza otras tareas.
c) Rutina de atención a la interrupción del timer.
d) Inicialización del sistema luego de un reset incluyendo la invocación a la rutina init_otros responsable de la configuración de otros periféricos no especificados. Directivas de reserva de memoria.

Dato: DELTA_T = 16ms = (1/4MHz) * 256 * 250

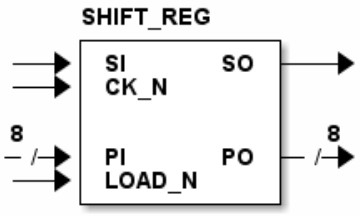
PROBLEMA 2

Se desea dotar a un sistema Z80 de una interfaz *slave* SPI. El protocolo SPI permite intercambiar datos entre un dispositivo *master* y un dispositivo *slave* conectados como se indica en la figura. En cada transferencia el *master* envía en forma serial un byte de datos $A_{7..0}$ por la línea **MOSI** (Master Output Slave Input) y recibe un byte ($B_{7..0}$) por la línea **MISO**.



Como se indica en el diagrama de tiempos, para realizar una transferencia el *master* baja la señal **SS**, da 8 pulsos en el reloj **SCK** y luego sube **SS**. Cuando baja **SS** ambos extremos deben poner el bit menos significativo del dato a transferir sobre la línea correspondiente. Luego en cada flanco de bajada de **SCK** deben ir desplazando el bit presente en su entrada hacia un registro de desplazamiento interno y poniendo en su salida el siguiente bit del dato que transmiten. De esta manera luego de los 8 pulsos se completa la transferencia en ambos sentidos.

Para hacer la conversión de serie a paralelo se utiliza un registro de desplazamiento de 8 bits sensible al flanco de bajada. En el caso de un *slave* se conecta la señal **MOSI** a la entrada serie **SI**, y la salida **MISO** se maneja con la salida serie **SO** del registro que es la salida del último FF de la cadena. El registro puede cargarse en paralelo poniendo el byte a escribir en la entrada paralelo **PI** de 8 bits y dando un pulso a 0 en **LOAD_N**. El contenido del registro está disponible todo el tiempo en la salida paralelo **PO**. Notar que antes de una transmisión **PO** = byte a enviar, y luego **PO** = byte recibido.



La interfaz SPI trabajará por interrupciones, atendiendo dos eventos diferentes: uno al comienzo de la transferencia (flanco de bajada de **SS**) y otro con el flanco de bajada de **SCK**.

- La interrupción originada por **SS** debe prepararse para contar los flancos de **SCK** y cargar el valor a transferir en el registro de desplazamiento. Dicho valor se lee en forma alternada de uno de dos sensores, **SEN0** o **SEN1** de 8 bits, comenzando por **SEN0** en la primera transferencia luego de reset.
- Cuando terminan de enviarse los 8 bits (transcurren 8 flancos de SCK) se debe leer el dato recibido y almacenarlo en la dirección reservada CONFIG para que sea utilizado por el programa principal.

Diseñar el *slave* SPI con un Z80 tradicional (buses triestado) usando modo 1 de interrupciones. Se pide:

a) Hardware. Incluir memoria, conexión del registro de desplazamiento, sensores y todo lo necesario para conectar las dos fuentes de interrupciones trabajando en modo 1. El sistema ya cuenta otros dispositivos que ocupan el rango ***0x00-0x3F*** en espacio de E/S.
b) Rutina de atención a interrupción que atienda las dos fuentes de interrupción (**SS** o **SCK**) trabajando en modo 1.
c) Inicialización del sistema luego de reset y directivas de reserva de memoria. Se debe invocar la subrutina init_otros(), que se supone implementada, para inicializar las partes del sistema no relacionadas con el *slave* spi.