# Práctica 2 - Ratón
## Puerto con handshake y recepción serie

## Objetivos

- Desarrollar programas de complejidad media con E/S controlada por handshake.
- Comunicarse con un periférico utilizando un protocolo estándar.
- Agregar periféricos a un sistema hardware dado.

## Descripción General

Al finalizar la presente práctica se contará con un sistema capaz de recibir datos en forma serial desde un dispositivo PS2. Además el sistema deberá interpretar los datos recibidos y desplegar en los displays 7 segmentos de la placa dos contadores que se actualizarán de acuerdo a los desplazamientos del mouse en Y y en X. Las cuentas volverán a cero presionando los botones derecho o izquierdo.

El programa que se debe escribir estará estructurado en subrutinas que hacen uso unas de otras, esto permitirá partir el problema en partes más sencillas de resolver y contar con subrutinas auxiliares que serán útiles en la siguiente práctica.

## Funcionamiento del ratón

Cuando se conecta un mouse PS2, este queda en un estado de espera hasta que se le envíe cierta secuencia de comandos de inicialización. Las rutinas necesarias para esto les serán dadas y deberán invocarlas luego de un reset, más adelante se ahondará sobre este punto. Luego de inicializado el mouse, cada vez que detecta un cambio (movimientos, cambio de estado de botones) los reporta enviando 3 bytes:

| | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| Byte 1 | Y overflow | X overflow | Y bit de signo | X bit de signo | 1 | Botón central | Botón derecho | Botón izquierdo |
| Byte 2 | Desplazamiento X | | | | | | | |
| Byte 3 | Desplazamiento Y | | | | | | | |

El desplazamiento se expresa en complemento a 2 de nueve bits, por ello se utiliza un byte y uno de los bits del primer byte. En caso de que el desplazamiento exceda lo que puede representarse, se pone en 1 el bit de overflow correspondiente. El estado de los botones se reporta en los 3 bits menos significativos del primer byte: 1 si está apretado, 0 si está suelto.

Más información sobre el protocolo PS2 y los mouse PS2 se puede consultar en la sección *Material* del EVA (*Protocolo PS2* y *Mouse PS2*).

El dispositivo (ratón) envía todos estos datos de 8 bits en forma serial utilizando el protocolo PS2. El protocolo de transmisión serie PS2 utiliza las señales:

- DATA: para enviar los códigos de 8 bits
- CLK_PS2: señal de sincronismo. DATA se debe leer en el flanco de bajada de esta señal

Cuando el dispositivo PS2 está inactivo, ambas señales valen 1. Para enviar un byte, el dispositivo PS2 primero baja DATA y luego baja CLK_PS2, generando un tren de pulsos, los cuales son usados para transmitir en forma sincronizada 11 bits en la señal DATA de la siguiente forma (ver Figura 1):

- 1 bit de arranque en 0 (Start)
- 8 bits de datos comenzando por el bit menos significativo (LSB) (D[7..0])
- 1 bit de paridad impar (Paridad)
- 1 bit de parada (Stop) que siempre es 1

El receptor **debe** leer cada bit en el flanco de bajada de CLK_PS2. El tiempo entre dos flancos de bajada consecutivos (el tiempo de un bit) es del orden de 100 microsegundos, bastante más largo que el período de reloj del sistema (fclk = 50MHz).

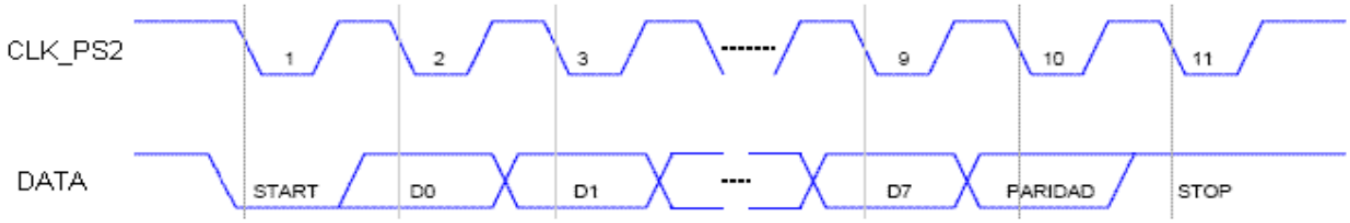


Figura 1: Recepción serie.

**Nota: recordar que luego de inicializar el sistema (stack, puertos de salida, etc) y antes de esperar datos del mouse, se debe invocar la subrutina de inicialización de mouse suministrada.**

# Hardware suministrado

**NOTA: Para comprender esta práctica es IMPRESCINDIBLE haber completado el Tutorial de Hardware.**

Junto con esta letra, recibirán un sistema hardware similar al utilizado en el tutorial. El mismo se encuentra especificado en los esquemáticos *sistema_top.bdf* y *sistema.bdf*.

El siguiente diagrama ilustra los elementos más relevantes del sistema. El esquemático *sistema.bdf* contiene el microprocesador, memoria, periférico *bridged_jtag_uart* para comunicación con el debugger en el PC, puertos de salida para controlar displays y leds y puertos de entrada para leer switches.

El esquemático *sistema_top.bdf* incluye multiplexores para seleccionar si las entradas PSCLK_I y PSDAT_I de *sistema.bdf* están conectadas al dispositivo PS2 o a pulsadores y switches. La llave SW[8] de la placa, permite seleccionar si trabajamos directamente con los pines del conector PS2 (SW[8] = 0 selecciona entradas PS2_KBCLK y PS2_KBDAT) o con señales generadas manualmente mediante un pulsador y una llave (SW[8] = 1 selecciona entradas BUTTON[2] y SW[0]). De esta manera se puede emular el comportamiento del dispositivo PS2 en forma controlada para facilitar la depuración de los programas.

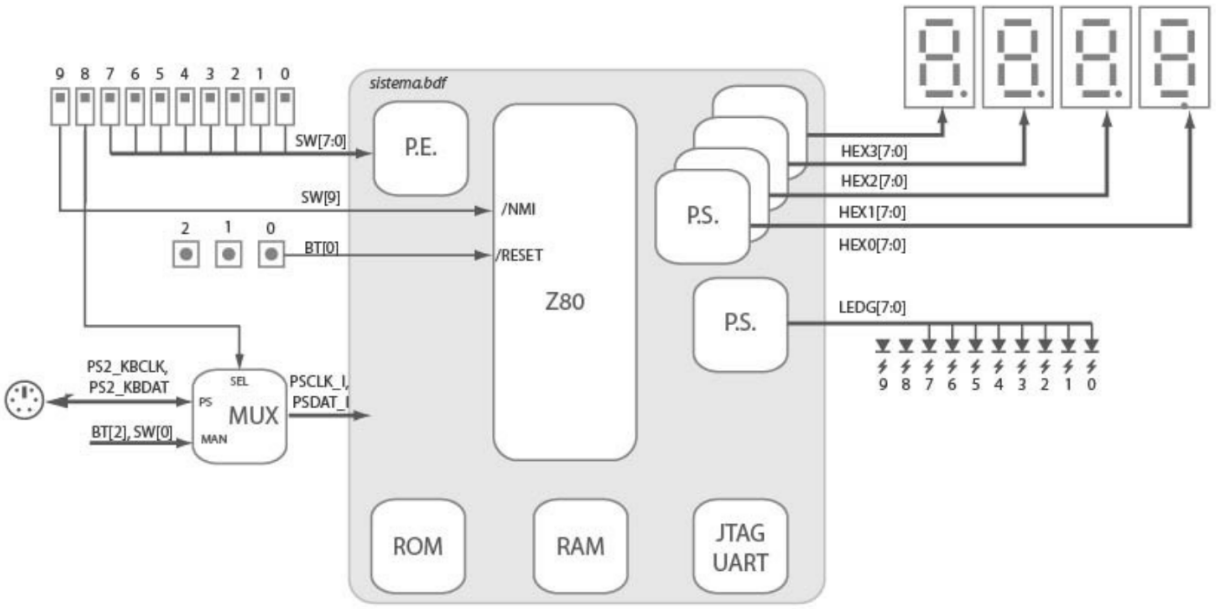


Figura 2: Descripción del hardware.

# Tareas a realizar

## a. Modificar hardware

### a.1. Puertos de entrada/salida

Deberán modificar *sistema.bdf* para agregar puertos de entrada y salida que permitan comunicarse con el dispositivo PS2.

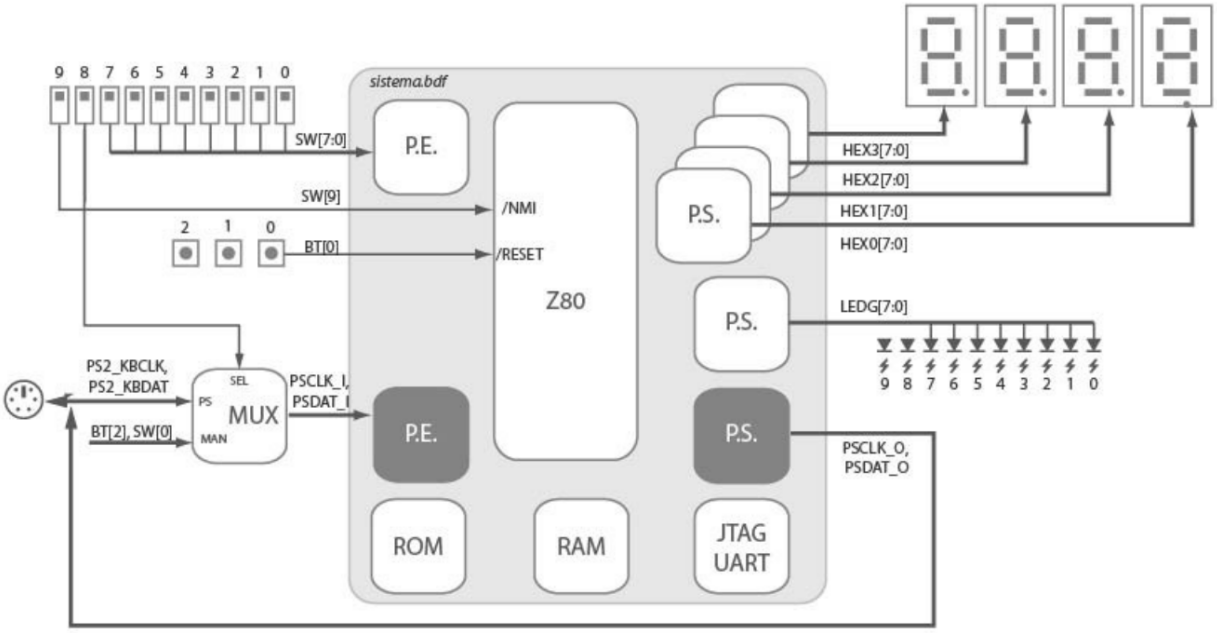


Figura 3: Hardware modificado.

Las entradas PSCLK_I, PSDAT_I y las salidas PSCLK_O y PSDAT_O se encuentran definidas en *sistema.bdf* pero faltan los puertos que manejan estas entradas y salidas.

Para implementar estos puertos, además de agregar los circuitos de decodificación correspondientes que deben diseñar, debe agregarse un circuito similar al que se muestra en la figura 4. Tener en cuenta que el circuito de la figura 4 se conecta a un sistema con bus triestado mientras que nuestro sistema tiene buses mutiplexados.

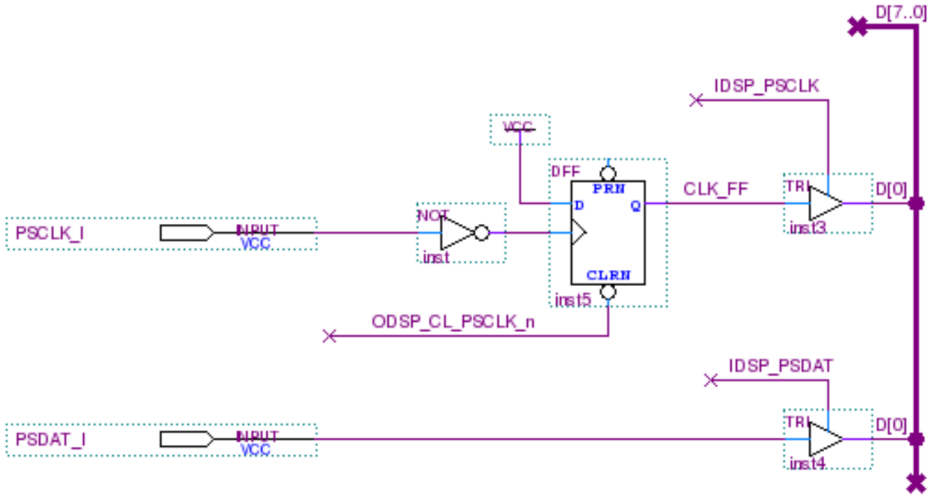


Figura 4: Puertos de entrada con buses triestado.

Los flancos de bajada en la señal de reloj de la interfaz serie PS2 llevan a "1" el FF de la figura 4, que puede ser borrado con el pulso de selección de dispositivo de un puerto de salida. La salida del FF puede leerse en el bit menos significativo de un puerto de entrada. Análogamente la señal DATA puede leerse en el bit menos significativo de otro puerto de entrada.

Observar que para los puertos de salida (ver figura 5) se utiliza un flip-flop con enable; por ese motivo las señales ODSP son activas en nivel alto. Esto se puede ver en el *Tutorial de Hardware* como la alternativa 1 para la implementación de puertos de salida.

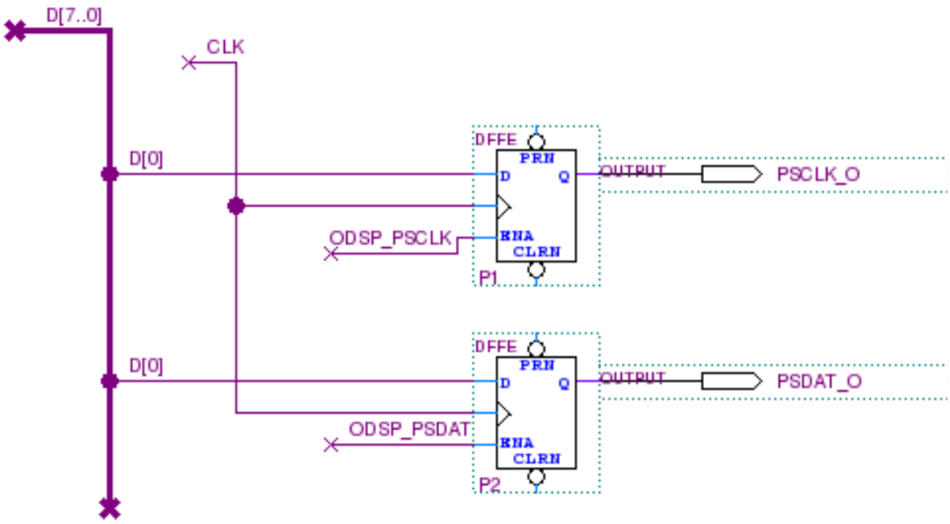


Figura 5: Puertos de salida.

Se dispone de la mitad superior del espacio de entrada/salida para realizar la decodificación de puertos. Algunas direcciones ya se encuentran ocupadas por los puertos de displays 7 segmentos, leds y switches.

| Dirección | Entrada | Salida |
|---|---|---|
| 0x80 | SW[7:0] | HEX0[7:0] |
| 0x81 | | HEX1[7:0] |
| 0x82 | | HEX2[7:0] |
| 0x83 | | HEX3[7:0] |
| 0x84 | | LEDG[7:0] |

Figura 6: Direcciones puertos de entrada y salida.

**Nota: En cada programa de prueba debe inicializarse el stack pointer (SP) y los puertos de salida. Los puertos PSDAT_O y PSCLK_O deben inicializarse a nivel alto.**

### a.2. Personalización de plantilla suministrada.

Junto con la letra les fue suministrada una plantilla (*plantilla_lab2.s*) y un archivo (*ps2-lib.s*) con subrutinas de inicialización del mouse. La plantilla debe tomarse como modelo para los programas de prueba. Incluye la inicialización del stack y llamada a la subrutina ***mouse_init*** encargada de enviar comandos al mouse para su inicialización.

La subrutina ***mouse_init*** debe acceder a los puertos que agregaron en la parte a) de esta práctica, por lo que deben completar las direcciones utilizadas en las constantes:

**.equ PSDAT_I , .....**
**.equ PSCLK_I , .....**
**.equ PSDAT_O , .....**
**.equ PSCLK_O , .....**
**.equ CL_PSCLK , .....**

## b. Esperar un flanco.

**Subrutina *esperoflanco***

**Descripción:** Subrutina que simplemente espera un flanco de bajada en la señal de reloj proveniente del dispositivo PS2 (o BUTTON[2]) y retorna. Los flancos son capturados por el flip-flop, el cual debe ser borrado por la misma subrutina antes de retornar. Se debe preservar todos los registros excepto el acumulador (A) y las banderas (F).

**Prueba:** Realizar un programa que consista en un bucle infinito que invoque a ***esperoflanco***, incremente un contador y lo despliegue en hexadecimal en los display 7 segmentos de la placa (para ello se sugiere utilizar la subrutina ***pbcda7seg*** de la práctica anterior). Probarlo inicialmente seleccionando como entrada BUTTON[2]. Una vez validado seleccionar la entrada desde el dispositivo PS2. ¿Cuánto debe incrementarse el contador al presionar un botón sin mover el mouse?

Considerando el software implementado, se debe completar el diagrama de tiempos de la figura 7. La señal PSCLK_I corresponde al puerto de entrada del sistema con el mismo nombre. La señal Q es la salida del FF de handshake correspondiente a dicho puerto de entrada. ODSP_CL_PSCLK es el pulso de selección utilizado (por el software implementado) para borrar dicho FF. Y la señal IDSP_PSCLK es el pulso de selección del puerto de entrada en cuestión. Adicionalmente se debe utilizar el último renglón para indicar en cada momento si el programa se encuentra ejecutando el programa principal (programa de prueba) o la subrutina ***esperoflanco***. El diagrama resuelto debe ser incluido en el informe. Dadas las diferentes escalas de tiempos involucradas, para las señales de

selección IDSP_PSCLK y ODSP_CL_PSCLK, alcanza con indicar la presencia de los pulsos sin dibujarlos a escala.

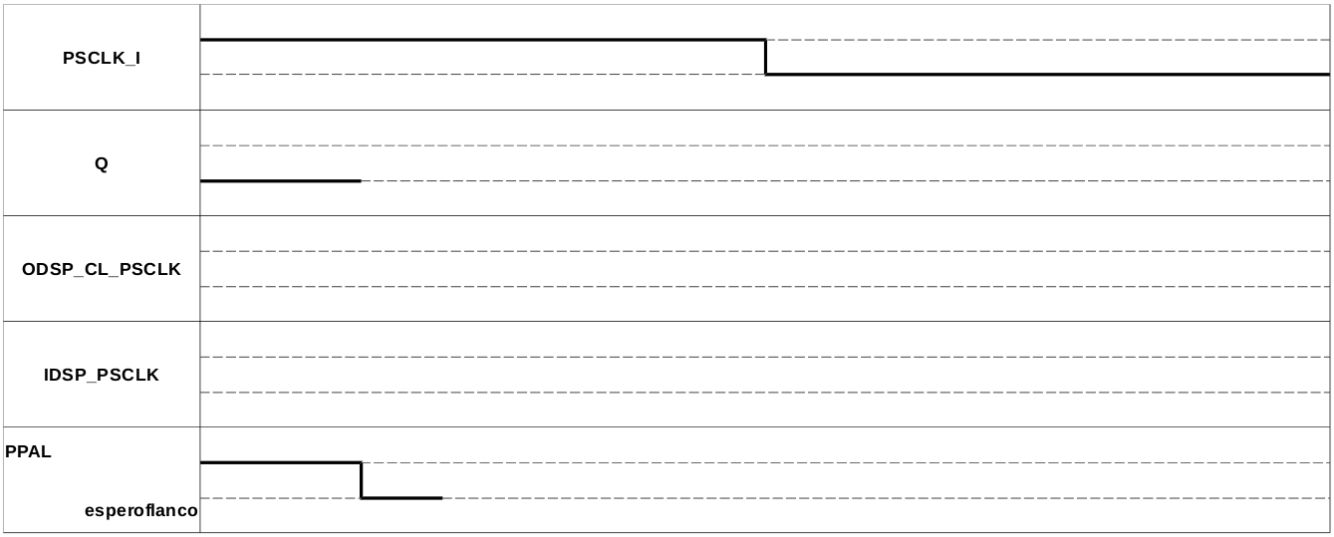


Figura 7: Diagrama de tiempos a completar.

## c. Leer un dato serie.

**Subrutina *get_ps2***

**Descripción:** Subrutina que espera la recepción desde el dispositivo PS2 de un dato de 8 bits en el formato serie descrito anteriormente y lo devuelve en el acumulador. El valor del bit de paridad leído debe devolverse en la bandera Carry. Se deben preservar todos los registros excepto el acumulador (A) y las banderas (F).

**Parámetros:** Devuelve el dato recibido en el registro A y el valor de paridad leído en la bandera Cy.

**Prueba:** El programa de prueba debe ser un bucle infinito que invoque ***get_ps2*** y despliegue en los dos dígitos menos significativos del display 7 segmentos el último dato recibido. El bit de paridad se desplegará en el punto decimal del dígito menos significativo del display. Además, el programa debe mostrar en los dos dígitos más significativos el penúltimo dato y en los 8 leds (LEDG[7:0]) el antepenúltimo dato. Igual que en el caso anterior se sugiere utilizar las subrutinas de conversión de hexadecimal a 7 segmentos de la práctica 1. En los leds el dato se despliega sin convertir. En el caso del penúltimo y antepenúltimo dato no se despliega el bit de paridad. Cada vez que ***get_ps2*** retorna porque se completa la recepción de un byte, el programa de prueba debe actualizar los tres valores de forma que vuelvan a quedar el último, penúltimo y antepenúltimo dato recibido en los displays indicados arriba.

Con el informe se deberá entregar un mapa indicando la ubicación en memoria de cada bloque de código: programa principal, subrutina ***get_ps2***, subrutinas auxiliares, tablas, constantes, variables, stack. Para cada uno de estos bloques indicar si el sistema podría funcionar si estuvieran ubicados en ROM en lugar de en RAM.

Recordar que se puede utilizar una sección adicional de código llamada .data para alojar las variables. Cuando se trabaja con el debugger (Macro >Compilar y Macro >JtagCon + Gdb), la sección .text queda ubicada a partir de la dirección 0xB000 y .data se ubica, a continuación, a partir del siguiente

inicio de una página de 256 bytes (por ejemplo si .text finaliza en la dirección 0xB123, .data se colocará a partir de la 0xB200). En cambio, cuando se carga el programa en ROM (Macro >Compilar para ROM y Macro >Cargar en ROM) .text se ubica en la dirección 0x0000 y .data en 0xB000.

Teniendo en cuenta esto, ¿qué precauciones deben tomarse o qué cambios deben realizarse para que el programa probado trabajando con el debugger funcione correctamente compilándolo y ejecutándolo desde ROM? Incluir la respuesta en el informe. Adicionalmente, se deben realizar todos estos cambios para que el programa de prueba de ésta y todas las subrutinas que siguen, funcionen indistintamente si son compilados para ROM o si son compilados para trabajar con el debugger.

## d. Procesar datos recibidos

**Subrutina *get_packet***

**Descripción:** Subrutina que recibe en IX un puntero a un lugar en memoria y espera los 3 bytes que envía el mouse cada vez que detecta un evento. Una vez recibidos los 3 bytes, realiza una conversión de los desplazamientos representados en complemento a 2 de nueve bits a complemento a 2 de 16 bits.

Devuelve lo siguiente en el lugar de memoria indicado por IX:

| | |
|---|---|
| (IX) | Primer byte recibido conteniendo banderas de overflow, signos y estado de botones |
| (IX+1) | Byte bajo de desplazamiento X representado en complemento a 2 de 16 bits |
| (IX+2) | Byte alto de desplazamiento X representado en complemento a 2 de 16 bits |
| (IX+3) | Byte bajo de desplazamiento Y representado en complemento a 2 de 16 bits |
| (IX+4) | Byte alto de desplazamiento Y representado en complemento a 2 de 16 bits |

**Parámetros:** Recibe en IX puntero a espacio de 5 bytes en memoria.

**Prueba:** El programa de prueba reserva espacio en memoria para los 5 bytes y luego queda en un bucle infinito invocando ***get_packet*** y actualizando el estado de los leds y digitos 7 segmentos. En los leds se muestra el primer byte y en los 4 digitos 7 segmentos el desplazamiento X o Y dependiendo de si el switch $SW[7]$ vale 1 o 0.

Recordar que el programa debe funcionar indistintamente si es compilado para ROM o si es compilado para trabajar con el debugger.

## e. Ajuste de sensibilidad a los desplazamientos

**Subrutina *barrel_shift_right***

**Descripción:** Subrutina que realiza un corrimiento hacia la derecha (división entre 2) entre 0 y 15 lugares de un dato de 16 bits (entero con signo en complemento a 2).

**Parámetros:** Recibe en HL el dato y en el acumulador la cantidad de posiciones a desplazar. Devuelve en HL el dato desplazado. No debe modificar los demás registros.

**Prueba:** El programa de prueba deberá ser un bucle infinito que invoca la subrutina ***get_packet*** y actualiza dos variables contador de 16 bits (para X e Y) donde se lleva la suma acumulada con signo del desplazamiento en X e Y.

A efectos de disminuir la sensibilidad ante los movimientos del ratón, los valores acumulados de desplazamiento en X y en Y deben dividirse por $2^n$ antes de ser desplegados, para lo que se utiliza la subrutina ***barrel_shift_right***. El valor devuelto por la subrutina se utiliza solamente para ser desplegado, no debe modificarse el valor de los contadores.

El valor acumulado de X se muestra, previa división por $2^n$, en los 4 dígitos 7 segmentos cuando switch $SW[7]$ vale 1. Ídem para el valor acumulado de Y cuando $SW[7]$ vale 0. El valor de la cantidad de posiciones a desplazar (n en la división entre $2^n$) debe leerse de los $SW[3.,0]$.

Mediante los switches se determinará el valor apropiado de corrimiento para lograr con comodidad variaciones de +1 o -1 en los valores desplegados (acumulado dividido $2^n$).

Los contadores podrán borrarse al presionar los botones del ratón: izquierdo para contador de X, derecho para contador de Y.

Recordar que el programa debe funcionar indistintamente si es compilado para ROM o si es compilado para trabajar con el debugger.

# Informe

La entrega consistirá de:

(I) Informe: archivo *dd-hh-mm-n-informe.pdf* (donde dd-hh-mm-n son el día, hora y número que identifican al grupo, p. ej.: ma-14-00-3). Este archivo deberá contener:

- Mapa de entrada/salida con puertos originales (ver figura 6) y puertos agregados.
- Esquemático de puertos y decodificación agregada. Para "pegar" en el informe un circuito realizado en el editor gráfico del Quartus se debe seleccionar el circuito, "copiarlo" y realizar un "pegado especial", indicando "mapa de bits".
- El pseudocódigo o el diagrama de flujo de las subrutinas y programas de prueba.
- El código assembler de las subrutinas y programas de prueba.
- El diagrama de tiempos del ejercicio planteado en la parte b.
- El mapa con las ubicaciones en memoria pedido en la parte c y la respuesta a la pregunta planteada para que el programe funcione correctamente cuando es compilado para ROM y cuando es compilado para trabajar con el debugger.
- El valor de corrimiento determinado en la parte e

(II) Programas: archivo comprimido en formato zip (*dd-hh-mm-n-fuentes.zip*) conteniendo los siguientes archivos:

- el código de todas las subrutinas (*subrutinas.s*)
- programas de prueba (*prueba_b.s* a *prueba_e.s*) para cada una de las subrutinas que incluyen el archivo anterior mediante la directiva .include y la plantilla modificada (plantilla_lab2.s).

(III) Hardware modificado: archivo comprimido en formato zip (*dd-hh-mm-n-hardware.zip*) con los siguientes archivos del diseño realizado por el grupo.

- *sistema.bdf*

- *lab_intup.sof*
- *lab_intup.qsf*

Se entregará por dos vías antes de la fecha indicada en EVA:

- versión papel del primer item en secretaría del IIE
- archivo formato .zip con todos los ítems mediante la tarea correspondiente en EVA

Cada estudiante deberá llevar registro de las horas dedicadas a la práctica. Se les solicitará que ingresen esa información a través de la página del curso.

El día de la evaluación **el grupo deberá presentarse 10 minutos antes de la hora establecida** en el laboratorio de software del instituto de Ingeniería Eléctrica.

Además sdeberán traer el KIT DE0-LAB y un "pen drive usb" con todos archivos de los proyectos indicados y las simulaciones realizadas, si se desea usar las pc de la facultad. Si utilizaron una laptop para hacer la tarea se recomienda traer la misma laptop para la defensa. En ambos casos probar que todos los programas funcionan antes de la práctica.