Tutorial de hardware

Introducción

A través de esta guía y con ayuda de la herramienta Quartus II, aprenderás a modificar el hardware de un sistema con Z80 implementado en la placa DE0 que se utiliza en el laboratorio del curso.

El sistema que se les entrega cuenta con un Z80, una memoria ROM, una memoria RAM y un periférico *bridged_jtag_uart,* necesario para la comunicación entre el debugger y el Z80.

Para aprender a trabajar con el hardware vamos a ver cómo agregar un puerto de entrada, utilizando las llaves *SW*[7..0] para dar valores a estas entradas, y cómo agregarle un puerto de salida al sistema, utilizando los leds *LEDSG*[7..0] de la placa para visualizar los valores de esta salida.

Para ello vamos a realizar las siguientes tareas:

1. Agregar un puerto de entrada en la dirección 0x80 para las llaves *SW[7..0]*.
2. Agregar un puerto de salida en la dirección 0x84 para los leds *LEDSG[7..0]*.
3. Definir los nuevos bloques y compilar el proyecto.
4. Probar los puertos con un programa de prueba que les entregamos.

Sistema a utilizar

Junto con este tutorial, se les entrega el archivo comprimido demo_hw.zip, que contiene todos los archivos necesarios para abrir con el Quartus II el sistema que incluye el Z80.

Dentro del archivo comprimido *demo-hw.zip* van a encontrar el siguiente árbol de directorios:

- `t80`, carpeta que contiene todos los archivos del diseño del Z80
- `demo-hw`
    - *`hw`,* archivo del proyecto QuartusII para el sistema con Z80.

Para abrir el proyecto, realizar los siguientes pasos:

1. Descomprimir el archivo `demo-hw.zip` en una carpeta de trabajo adecuada, manteniendo la jerarquía de subdirectorios descrita arriba.

2. Abrir en Quartus II el proyecto `carpeta_de_trabajo\demo-hw\hw\demo-hw.qpf`.

El esquemático de mayor jerarquía del proyecto es el archivo `sistema_top.bdf`. Este circuito se compone de 3 bloques los que serán utilizados en todos los laboratorios:

- *Mi_pll_divisor*: Es un bloque divisor de frecuencia utilizado para acondicionar la señal de reloj del sistema. En este caso el cociente de división es uno, por lo que la frecuencia del reloj se mantiene en 50MHz.
- *button_debouncer*: Son eliminadores de rebote para los botones y switches.
- *Sistema*: Este es el sistema con el Z80 donde trabajaremos. Su contenido está definido en *sistema.bdf* y contiene:
    - ***T80s***: Es el microprocesador, la implementación del Z80 en el FPGA.
    - ***ROM***: Memoria de solo lectura. Contiene el programa *monitor.hex* que se comienza a ejecutar luego de un reset del Z80 y queda a la espera de comandos por parte del debugger.
    - ***SSRAM***: Memoria de lectura-escritura. Aquí se cargan nuestros programas.
    - ***Bridged_jtag_uart***: Periférico utilizado para comunicar el debugger con el PC.
    - ***DFF***: Flip-flop empleado para implementar una interrupción no enmascarable (NMI) con los flancos de bajada de SW[9].
    - ***Decodificación***: Lógica utilizando compuertas y el decodificador 74138[1] para decodificar memoria y periféricos.

1 En la página del curso, sección materiales, se encuentra la hoja de datos de este componente.

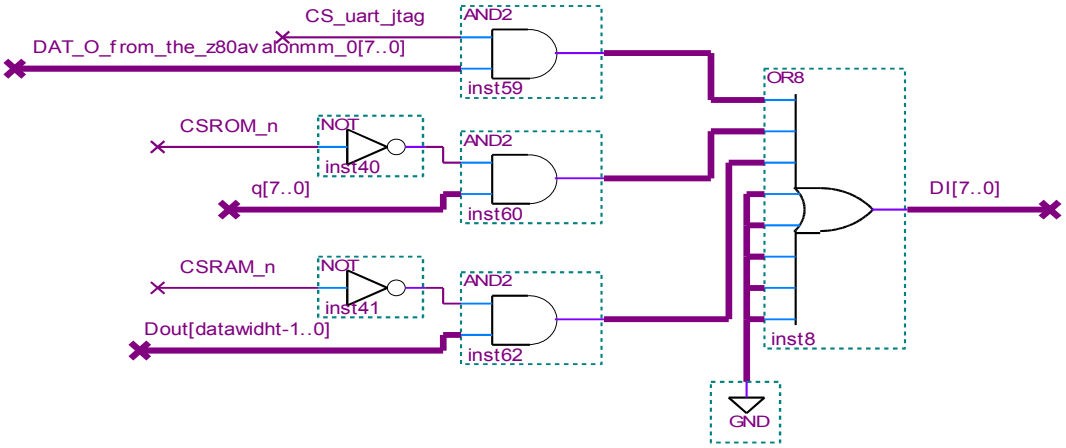


Figura 1: Lógica bus datos de entrada (DI[7..0])

Observar que los buses de datos no son los buses triestado habituales en un Z80 estándar. Debido a que dentro del chip que contiene a todo el sistema (el FPGA de la placa DE0) no pueden definirse buses triestado, el bus de datos debe separarse en dos buses independientes, bus de datos de salida y bus de datos de entrada. Este último debe multiplexarse con una lógica de compuertas AND y OR para combinar los diferentes orígenes de datos. Tanto el microprocesador como los dispositivos conectados a él (memorias y puertos) tienen sus entradas y salidas de datos separadas. Para las transferencias de datos entre el procesador y los dispositivos basta con conectar la salida de datos del microprocesador directamente a las entradas de datos de cada dispositivo. Para las transferencias desde los dispositivos hacia el procesador se hace un AND entre la salida *datos_out[]* de cada dispositivo y su señal *CS* (negada para que sea activa en '"1") generada por la lógica de decodificación. Las salidas de estas compuertas AND se combinan en una compuerta OR (octal en este caso). La salida de esta compuerta OR se conecta a la entrada de datos del procesador, ver figura 1. Como solo hay una de las señales *CS* activa en cada instante, a la salida de la compuerta OR se tendrá la salida *datos_out[]* del dispositivo seleccionado.

El circuito entregado contiene todo lo necesario para cargar un programa y probarlo desde el debugger pero no tiene ningún puerto de propósito general, por lo que si ejecutamos los programas vistos anteriormente en el laboratorio, estos no podrán leer el estado de los switches o botones, ni encender ningún led o display. Las únicas entradas son *Boton[0]*, que activa el reset del microprocesador, y *SW[9]* que genera una interrupción no enmascarable utilizada por el debugger. Para poder realizar esta interacción con el usuario, se deben agregar puertos de entrada y salida al sistema.

1 - Agregar un puerto de entrada

El objetivo es poder leer el estado de las llaves *SW[7..0]*. Para esto vamos a implementar un puerto de entrada en la dirección 0x80 del espacio de entrada. El puerto es muy sencillo y se muestra el esquemático en la figura 2.

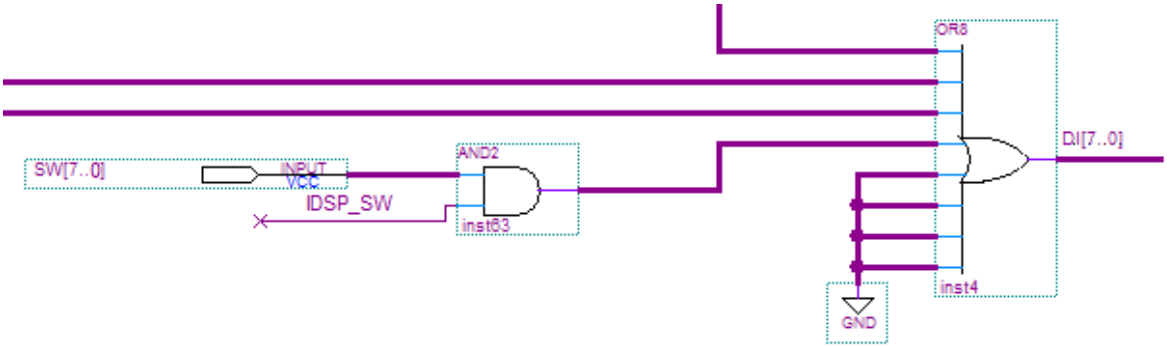


Figura 2: *Puerto de entrada sin memoria.*

Notar que se trata de un puerto de entrada sin memoria, por lo que no se le agregan FFs.

Observar además cómo se multiplexa con las compuertas AND y OR. De esta forma, cuando IDSP_SW = 1, se "habilita" *SW[7..0]* a que pase a la salida de la AND. Como las demás entradas de la OR serán "0" (pues un único IDSP = 1 por vez), su salida, que está conectada al bus de datos de entrada del microprocesador, será igual a ***SW[7..0]***.

Hace falta agregar al circuito de decodificación la salida IDSP_SW que se active para la dirección 0x80 de entrada. Para esto se puede utilizar 2 decodificadores 74138 conectados como se muestra en la figura 3.

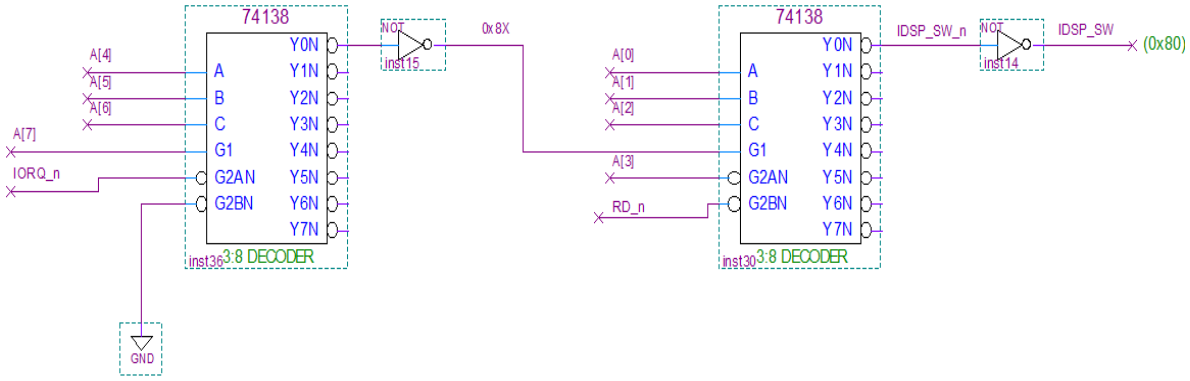


Figura 3: *Decodificación del puerto de entrada en la dirección 0x80*

Tener en cuenta que los pines de entrada *SW[7..0]* ya están incluidos en los

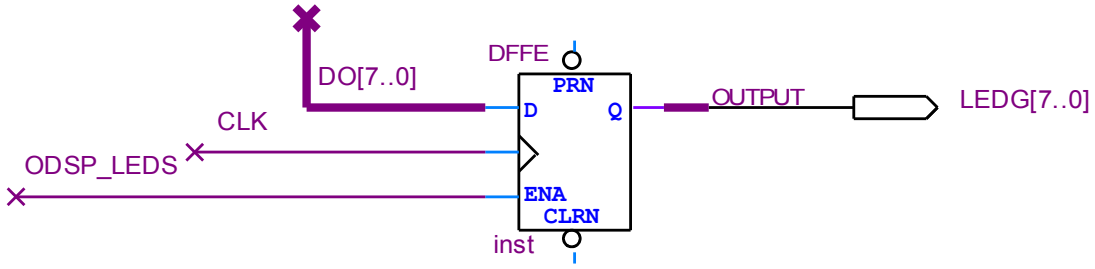
archivos sistema.bdf y sistema_top.bdf, pero no están conectados.

Estudiar la decodificación de la figura 3 y verificar que no tiene decodificación fantasma.

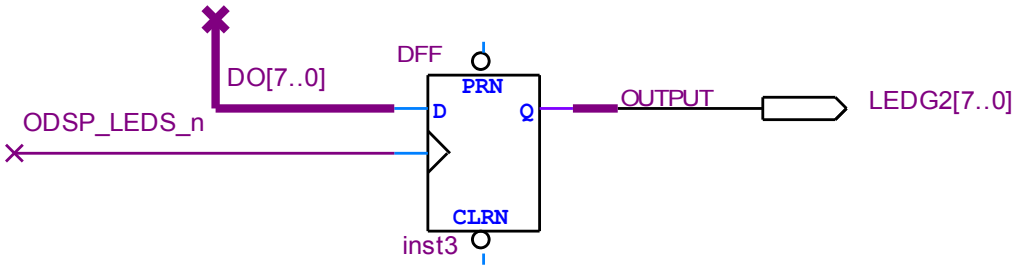
Con este puerto ya se puede leer el estado de *SW[7..0]* utilizando, por ejemplo, la instrucción *IN A,(0x80)* que deja en el acumulador el estado de las llaves.

2 - Agregar un puerto de salida

El objetivo es encender los leds *LEDG[7..0]* con el Z80. Para esto vamos a implementar un puerto de salida en la dirección 0x84 del espacio de entrada/salida. La figura 4 muestra dos alternativas para la implementación del puerto. En ambas se utiliza un registro para memorizar el contenido del bus de datos de salida *DO[7..0]* en el ciclo de escritura en E/S. Se recomienda utilizar la primera variante en la que los datos se memorizan en los flancos de subida de *CLK* mientras *ODSP_LEDS* (activa en nivel alto) habilita la entrada ENA. La segunda es la variante vista en clase en que los datos se memorizan en el flanco de subida de *ODSP_LEDS_n* (activa en nivel bajo).



Alternativa 1 - Registro con habilitación.



Alternativa 2: Registro sin habilitación.

Figura 4: *Puerto de salida.*

La decodificación para el puerto de salida en la dirección *0x84* es similar al caso anterior, pero ahora con la señal *WR_n* en lugar de *RD_n*. Se muestra el esquemático en la figura 5.

Verificar que para este caso tampoco hay decodificación fantasma del puerto.

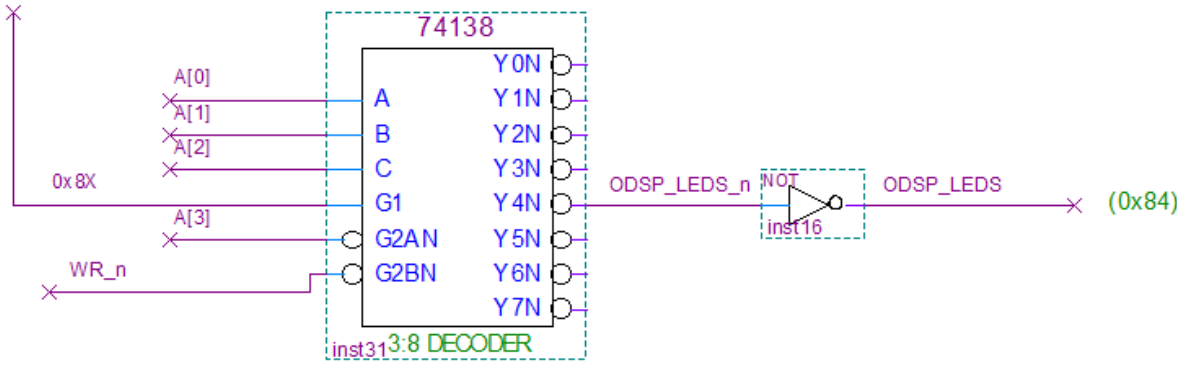


Figura 5: *Decodificación del puerto de salida en la dirección 0x84*

De esta forma, escribiendo en el puerto 0x84, por ejemplo, con la instrucción *OUT (0x84),A* el contenido del registro A se copia al registro del puerto y maneja el estado de los leds *LEDG[7..0]* (1 apagado, 0 encendido).

Tener en cuenta que los pines de entrada *LEDG[7..0]* ya están incluidos en los archivos sistema.bdf y sistema_top.bdf, pero no están conectados.

3 - Definir el nuevo bloque, asignar pines y compilar el proyecto.

Como en el archivo de partida de este tutorial ya estaban incluidas las entradas y salidas agregadas (*SW[7..0]* y *LEDG[7..0]*) el símbolo del bloque *sistema* no se modificará.

Lo que sí se debe verificar es que la asignación de pines está hecha correctamente. Para eso deben abrir la herramienta *Pin Planner* (menú *"Assignments > Pins"*) y ver que aparecen todas las entradas y salidas utilizadas y que los pines asignados coinciden con la información del manual de usuario de la placa DE0.

Una vez hecho esto se puede compilar el proyecto.

Si no hay errores debe obtenerse como resultado el archivo `hw\out\demo-hw.sof` que puede grabarse en la placa utilizando el programador de Quartus (menú *"Tools > Programmer"*) o abriendo el archivo hw\out\demo-hw.cdf con Notepad++ y utilizando el macro *"Grabar Sistema en FPGA"*.

4 - Probar los puertos

Para probar el nuevo hardware, programar la placa DE0 con el archivo `out/demo_hw.sof` y cargar el siguiente programa en el sistema utilizando el Notepad++.

```
;Puerto de entrada:
SW equ 0x80
;Puerto salida:
Leds equ 0x84

.text

Loop: IN A, (SW)
      OUT (Leds), A
      JR Loop
.end
```

Ejecutar paso a paso el programa con el *debugger*. Utilizando el comando "`info registers`", observar el valor del registro A antes y después de ejecutar la instrucción IN A, (SW). Hacer lo mismo con la instrucción OUT (Leds), A

Verificar el funcionamiento moviendo las llaves *SW[7..0]* y verificando que el nuevo valor se copia a los leds.