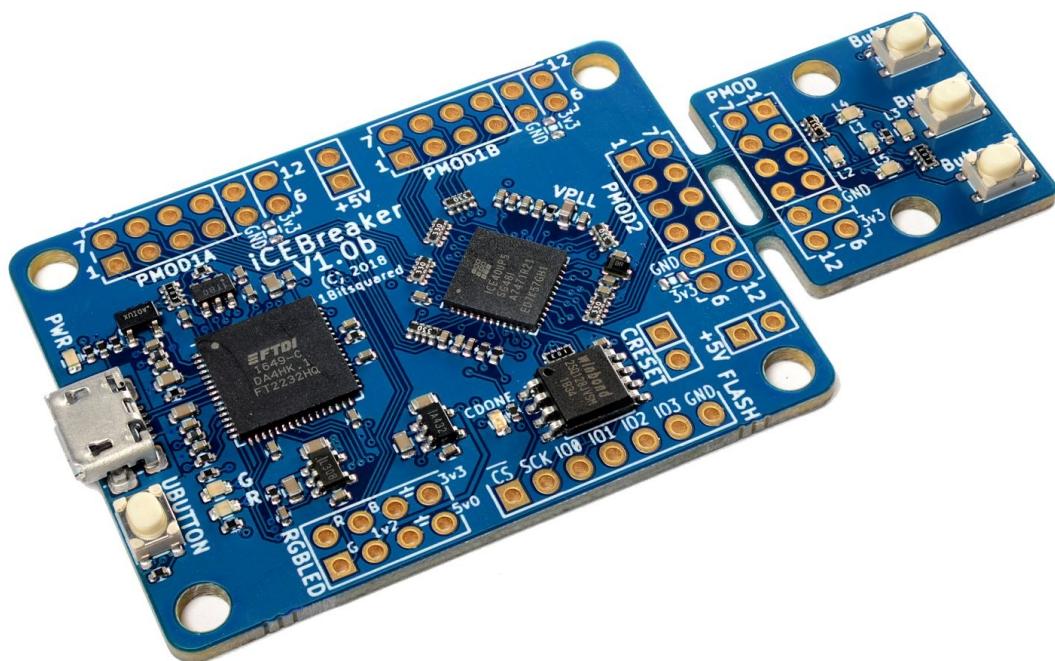


Sistemas Electrónicos

para Automatización

Memoria de trabajo realizado en prácticas



Nombre: Sergio León Doncel, Álvaro García Lora

Curso: 2022/ 2023

Titulación: 4º Curso, GIERM

Memoria de prácticas.

Práctica 1

En primer lugar, una vez que hemos clonado el repositorio del microprocesador neorv32 con el que se trabajará, comentaremos los pasos seguidos para preparar la capas hardware y software, que será nuestra filosofía de trabajo a lo largo de las prácticas.

Como paso previo, necesitamos actualizar las variables de entorno necesarias para los programas que se usarán. Esto se consigue añadiendo el comando `source ~/fosshdl/env.rc`. Para evitar tener que hacerlo continuamente en cada terminal que usemos, optamos por modificar el archivo `.zshrc` el cual contiene la configuración de zsh.

```

.oh-my-zsh.zshrc
01 # ZSH_CUSTOM=/path/to/new-custom-folder
02
03 # Which plugins would you like to load?
04 # Standard plugins can be found in $ZSH/plugins/
05 # Custom plugins may be added to $ZSH_CUSTOM/plugins/
06 # Example format: plugins=(rails git textmate ruby lighthouse)
07 # Add wisely, as too many plugins slow down shell startup.
08 plugins=(git)
09
10 source $ZSH/oh-my-zsh.sh
11
12 # User configuration
13
14 # export MANPATH="/usr/local/man:$MANPATH"
15
16 # You may need to manually set your language environment
17 # export LANG=en_US.UTF-8
18
19 # Preferred editor for local and remote sessions
20 # if [[ -n $SSH_CONNECTION ]]; then
21 #   export EDITOR='vim'
22 # else
23 #   export EDITOR='mvim'
24 # fi
25
26 # Compilation flags
27 # export ARCHFLAGS="-arch x86_64"
28
29 # Set personal aliases, overriding those provided by oh-my-zsh libs,
30 # plugins, and themes. Aliases can be placed here, though oh-my-zsh
31 # users are encouraged to define aliases within the ZSH_CUSTOM folder.
32 # For a full list of active aliases, run `alias`.
33
34 # Example aliases
35
36 alias zshconfig="mate ~/.zshrc"
37 alias ohmyzsh="mate ~/.oh-my-zsh"
38 source /opt/ros/noetic/setup.zsh
39 source ~/fosshdl/env.rc
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105

```

Comenzando por la parte hardware, la configuración e implementación del microprocesador podemos realizarla partiendo del fichero original `neorv32_iCEBreaker_BoardTop_MinimalBoot.vhd` (que en futuras prácticas modificaremos para incluir las conexiones e instancias necesarias para cumplir las funcionalidades requeridas).

```

leon@Leon-PC:~/neorv32/setups/osflow
leon@Leon-PC:~/neorv32/setups/osflow 80x24
~ cd neorv32/setups/osflow
~ osflow make BOARD=iCEBreaker MinimalBoot

```

Gracias a la ejecución del `make` agrupamos en una única orden la siguiente secuencia de pasos: síntesis (por medio de Yosys y GHDL), Place & Route (con `nextpnr`) y generación del bitstream (usando `icestorm`).

De esta forma, se genera el bitstream que cargaremos en la placa de la FPGA. A continuación, se muestra como se ha procedido para hacerlo:

-Lo primero de todo es comprobar que nuestro ordenador haya detectado correctamente la placa física.

```
leon@Leon-PC:~/neorv32/setups/osflow
+ osflow lsusb
Bus 004 Device 001: ID 1deb:0003 Linux Foundation 3.0 root hub
Bus 003 Device 003: ID 0480:c966 Integrated Technology Express, Inc. ITC Device(8176)
Bus 003 Device 002: ID 4e53:5407 USB OPTICAL MOUSE
Bus 003 Device 001: ID 100b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 100b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 0409:e0cd Foxconn / Hon Hua Wireless_Device
Bus 001 Device 002: ID 04f2:b6c2 Chicony Electronics Co., Ltd Integrated Camera
Bus 001 Device 004: ID 0403:6018 Future Technology Devices International Ltd FT2232C/B/R Dual UART/FIFO IC
Bus 001 Device 001: ID 100b:0002 Linux Foundation 2.0 root hub
```

-Posteriormente, con iceprog se configura la FPGA:

```
leon@Leon-PC:~/neorv32/setups/osflow
+ osflow iceprog neorv32_iCEBreaker_MinimalBoot.bit
init...
cdone: high
reset...
cdone: low
flash ID: 0xEF 0x70 0x18 0x00
file size: 104090
erase 64KB sector at 0x000000.. erase 64KB sector at 0x010000..
programming...
reading...
VERIFY OK
cdone: high
Bye...
```

Vamos a compilar un ejemplo software. Para ello entramos en la carpeta del ejemplo `blink_led` e invocamos a `make`:

```
cd ~/neorv32/sw/example/blink_led
make
```

Vemos que, al igual que el resto de makefiles del neorv32, al invocar a `make` sin más argumentos, nos indica los targets disponibles. Como queremos compilar un ejecutable para descargar utilizando el bootloader, generaremos el target `exe`:

```
make exe
```

Alcanzados este punto, tenemos nuestro microprocesador listo para ser programado, finalizando por tanto el trabajo sobre la capa hardware.

Centrándonos en el software, lo primero que debemos hacer es preparar un ejemplo que queramos programar en el microprocesador. Este código será en general desarrollado por nosotros mismos, aunque en este acercamiento inicial se usa uno de los ejemplos básicos ofrecidos, como el `blink_led`. La compilación se realiza de la siguiente forma:

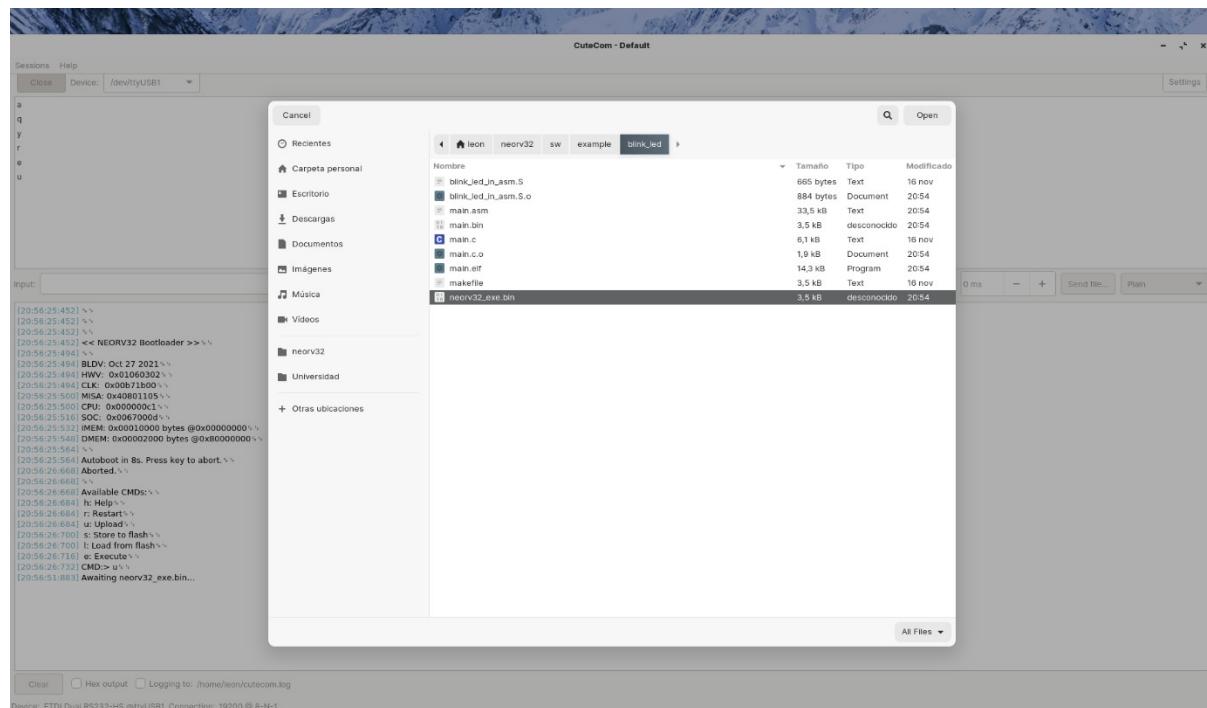
```
leon@Leon-PC:~/neorv32/sw/example/blink_led
+ cd ~/neorv32/sw/example/blink_led
+ blink_led make exe
realmente ficheros top-level descriptos en VHDL.
Memory utilization:
text data bss dec
3468 0 116 3584
Executable (neorv32_exe.bin) size in bytes:
3480
Implementaremos para la tarjeta iCEBreaker el top-level MinimalBoot. Este top-level instancia el micro en una configuración en la que contiene
```

Se puede ver como ha procedido sin errores y el binario generado es de un tamaño de 3kB, mucho menor a la capacidad máxima admitida por el microprocesador.

```
→ blink_led ls
blink_led_in_asm.S      main.asm   main.c      main.elf  neorv32_exe.bin
blink_led_in_asm.S.o    main.bin   main.c.o   makefile
```

Finalmente, nos comunicamos con el microprocesador a través de la UART haciendo uso del bootloader con la aplicación de cutecom.

Tras hacer la configuración necesaria en el mismo, podemos subir el fichero compilado .bin y comprobamos el correcto funcionamiento.



Tras haber reiniciado la placa con el botón N de reset, mandamos cualquier carácter para acceder a la CMD del microprocesador. Enviando una ‘u’ (upload), seleccionando el binario correspondiente y usando ‘e’ (execute) se ejecuta el código en C dentro de nuestro microprocesador, el cual vemos como efectivamente hace parpadear el led cada 200 ms.

Por último, para aunar todos estos pasos en uno y así ser más eficientes trabajando en las siguientes prácticas, hemos decidido implementar el siguiente script:

```

#!/bin/bash
cd ~/neorv32/setups/osflow
make BOARD=iCEBreaker MinimalBoot
iceprog neorv32_iCEBreaker_MinimalBoot.bit
cd ~/neorv32/sw/example/$1
make exe
cutecon &
echo "\n\n"
echo "-----"
echo "open & reset board & send any character via UART"
echo "CMD> u (Upload) + load program + e (execute)"
echo "-----"
echo "\n\n"

```

Éste contiene tanto la capa hardware como software, y su uso sólo requiere pasar como parámetro el nombre de la carpeta (dentro de sw/examples) del proyecto a cargar en el micro.

```

leon@Leon-PC:~/neorv32
leon@Leon-PC:~/neorv32 69x24
$ ~ cd neorv32
$ neorv32 ./basic_comands.sh blink_led

```

Pasamos ahora a realizar una serie de modificaciones tanto en la parte hardware como software con vistas a trabajar con los botones y leds de la tarjeta a través del GPIO. La ampliación que se pretende conseguir es realizar una máquina de estados donde las entradas serán los botones de la placa y las salidas los leds de la misma.

Lo primero de todo es mostrar las modificaciones hardware realizadas sobre el .vhd original. Las entradas y salidas de la entidad del top_level necesarias para usar los leds y botones comentados son las siguientes:

```

entity neorv32_iCEBreaker_BoardTop_MinimalBoot is
    -- Top-level ports. Board pins are defined in setups/osflow/constraints/iCEBreaker.pcf
    port (
        -- 12MHz Clock input
        iCEBreakerv10_CLK : in std_logic;
        -- UART0
        iCEBreakerv10_RX : in std_logic;
        iCEBreakerv10_TX : out std_logic;
        -- Buttons
        iCEBreakerv10_BTN_N : in std_logic;
        iCEBreakerv10_PMOD2_9_Button_1 : in std_logic;
        iCEBreakerv10_PMOD2_4_Button_2 : in std_logic;
        iCEBreakerv10_PMOD2_10_Button_3 : in std_logic;
        -- LEDs
        iCEBreakerv10_LED_R_N : out std_logic;
        iCEBreakerv10_LED_G_N : out std_logic;
        iCEBreakerv10_PMOD2_1_LED_left : out std_logic;
        iCEBreakerv10_PMOD2_2_LED_right : out std_logic;
        iCEBreakerv10_PMOD2_8_LED_up : out std_logic;
        iCEBreakerv10_PMOD2_3_LED_down : out std_logic;
        iCEBreakerv10_PMOD2_7_LED_center : out std_logic
    );
end entity;

```

Vemos como se conectan las señales creadas con los puertos de entrada y salida de la instancia de la CPU (neorv32_top) en el port map.

```

-- GPIO (available if IO_GPIO_EN = true) --
gpio_o => gpio_o, -- parallel output
gpio_i => gpio_i, -- parallel input

```

Por último, se unen las señales con las entradas y salidas del top_level.

```

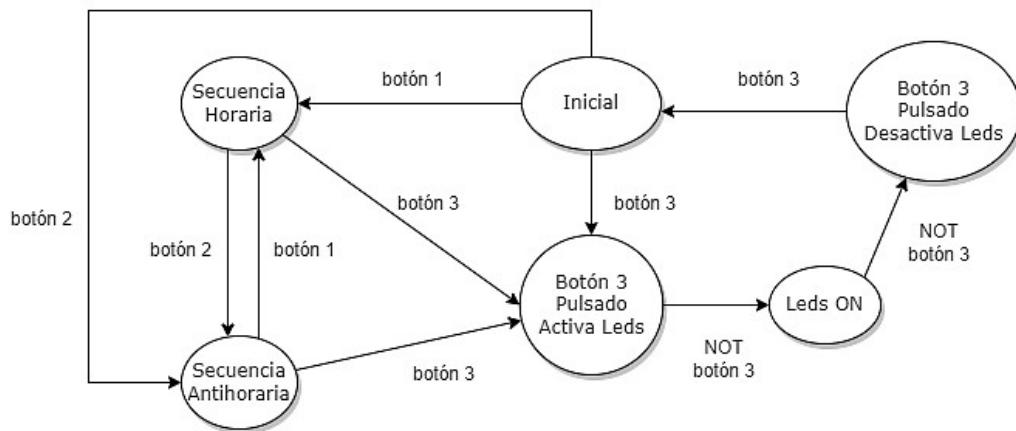
-- -----
-- IO Connections
-- -----

--outputs (leds)--
iCEBreakerv10_PMOD2_8_LED_up <= gpio_o(0);
iCEBreakerv10_PMOD2_2_LED_right <= gpio_o(1);
iCEBreakerv10_PMOD2_3_LED_down <= gpio_o(2);
iCEBreakerv10_PMOD2_1_LED_left <= gpio_o(3);
iCEBreakerv10_PMOD2_7_LED_center <= gpio_o(4);

--inputs (buttons)--
gpio_i(0) <= iCEBreakerv10_PMOD2_9_Button_1;
gpio_i(1) <= iCEBreakerv10_PMOD2_4_Button_2;
gpio_i(2) <= iCEBreakerv10_PMOD2_10_Button_3;
gpio_i(63 downto 3) <= (others => '0');

```

Una vez configurada la parte hardware, pasamos al software. Para demostrar el funcionamiento y entendimiento de uso del GPIO, se ha decidido realizar una máquina de estado como se ha anticipado anteriormente. Veamos el diagrama correspondiente a la misma:



En resumen, al pulsar el botón 1 se dará una secuencia de encendido y apagado de leds que simulan un movimiento circular en sentido horario con una frecuencia de 1 Hz. De igual forma, si se pulsa el botón 2 se da la secuencia en sentido antihorario. Por otro lado, al presionar el botón 3, se entra en un estado de emergencia, donde se iluminan los 5 leds. En el caso de presionar el botón 1 o 2 no tendrá ningún efecto. Para salir de este estado habrá que volver a pulsar nuevamente el botón 3 y los leds se apagarán.

Explicamos las partes más relevantes desarrolladas del código en c.

Usando una enumeración asignamos valor numérico a los nombres de los estados.

```

enum estados
{
    inicial,
    boton3_pulsado_activaLeds,
    boton3_pulsado_desactivaLeds,
    secuencia_horaria,
    secuencia_antihoraria,
    leds_ON
};
  
```

En la función principal se añade la llamada a la función correspondiente a la máquina de estados.

```

int main()
{
    ...
    neorv32_uart0_print("Starting States Machine\n");
    maquina_estados();
}
  
```

Se ha implementado el diagrama anterior de forma síncrona (cada 50 ms) haciendo uso de las funciones `neorv_gpio_port_set()`, `neorv_gpio_pin_set()`, `neorv_gpio_pin_get()` y `neorv_gpio_pin_clr()`.

Puntualizamos que con la variable `time50ms` se comprueba cuando ha pasado 250 ms para realizar las transiciones entre leds consecutivos en la secuencia de giro.

```

void maquina_estados(void)
{
    static enum estados estadosMaq = 0;
    static char led = 0;
    static char time50ms = 0;
    static bool toggleLeds = false;

    while (1)
    {
        neorv32_cpu_delay_ms(50);

        switch (estadosMaq)
        {
        case inicial:
            // Salidas asociadas al estado
            neorv32_gpio_port_set(0b00000);

            // Condiciones de cambio de estado
            if (neorv32_gpio_pin_get(0))
                estadosMaq = secuencia_horaria;
            if (neorv32_gpio_pin_get(1))
                estadosMaq = secuencia_antihoraria;
            if (neorv32_gpio_pin_get(2))
            {
                neorv32_cpu_delay_ms(20); // Filtro antirebote
                estadosMaq = boton3_pulsado_activaLeds;
            }
            break;

        case secuencia_horaria:
            // Salidas asociadas al estado
            if (toggleLeds)
            {
                toggleLeds = false;
                if (led < 3)
                    led++;
                else
                    led = 0;
                if (led > 0)
                    neorv32_gpio_pin_clr(led - 1);
                else
                    neorv32_gpio_pin_clr(3);
                neorv32_gpio_pin_set(led);
            }

            // Condiciones de cambio de estado
            if (neorv32_gpio_pin_get(1))
                estadosMaq = secuencia_antihoraria;
            if (neorv32_gpio_pin_get(2))
                estadosMaq = boton3_pulsado_activaLeds;
            break;

        case secuencia_antihoraria:
            if (toggleLeds)
            {
                toggleLeds = false;
                // Salidas asociadas al estado
                if (led > 0)
                    led--;
                else
                    led = 3;
                if (led < 3)
                    neorv32_gpio_pin_clr(led + 1);
                else
                    neorv32_gpio_pin_clr(0);
                neorv32_gpio_pin_set(led);
            }

            // Condiciones de cambio de estado
            if (neorv32_gpio_pin_get(0))
                estadosMaq = secuencia_horaria;
            if (neorv32_gpio_pin_get(2))
                estadosMaq = boton3_pulsado_activaLeds;
            break;

        case boton3_pulsado_activaLeds:
            // Salidas asociadas al estado
            neorv32_gpio_port_set(0b11111);

            // Condición de cambio de estado
            if (!neorv32_gpio_pin_get(2))
                estadosMaq = leds_ON;
            break;

        case boton3_pulsado_desactivaLeds:
            // Salidas asociadas al estado
            neorv32_gpio_port_set(0b00000);

            // Condición de cambio de estado
            if (!neorv32_gpio_pin_get(2))
                estadosMaq = inicial;
            break;

        case leds_ON:

            // Condición de cambio de estado
            if (neorv32_gpio_pin_get(2))
                estadosMaq = boton3_pulsado_desactivaLeds;
            break;
        }

        if (time50ms < 5)
        {
            time50ms++;
        }
        else
        {
            toggleLeds = true;
            time50ms = 0;
        }
    }
}

```

Práctica 2

En esta práctica utilizaremos el GPIO del sistema para controlar el teclado numérico hexadecimal PMOD Keypad de Digilent que se conecta a los puertos de expansión PMOD de la tarjeta iCEBreaker.

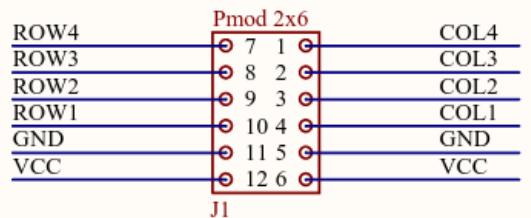
Para comenzar haremos una pequeña explicación sobre la conexión y el uso del mismo a partir de la documentación proporcionada por el fabricante.

La descripción de los pines de entrada y salida del teclado con la que contamos se recoge en esta tabla.

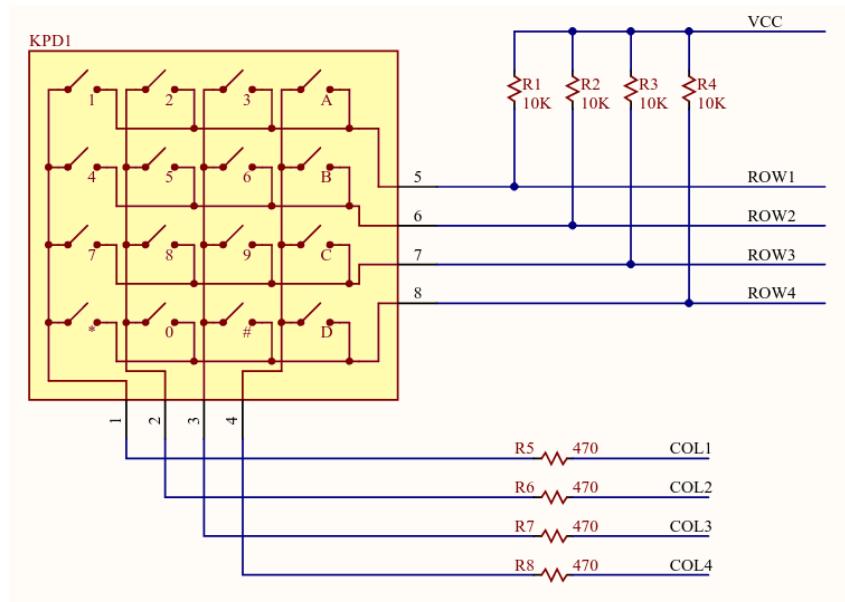
Pinout Description Table

Header J1					
Pin	Signal	Description	Pin	Signal	Description
1	COL4	Column 4	7	ROW4	Row 4
2	COL3	Column 3	8	ROW3	Row 3
3	COL2	Column 2	9	ROW2	Row 2
4	COL1	Column 1	10	ROW1	Row 1
5	GND	Power Supply Ground	11	GND	Power Supply Ground
6	VCC	Power Supply (3.3V/5V)	12	VCC	Power Supply (3.3V/5V)

La conexión con los puertos de expansión de la tarjeta es la que se muestra. En nuestro caso hemos decidido usar el Pmod1A.



Una vez aclarado el conexionado físico, vemos como debemos tratar a los pines correspondientes a columnas y filas de la matriz del teclado para hacerlo funcionar. Para ello nos apoyamos en el siguiente esquemático:



Como se observa, los pines asociados a las filas están conectados por resistencia de pullup a Vcc, por lo que están normalmente a 1. Aprovechando eso, podemos seleccionar una columna forzando su valor a 0, de tal manera que, si hay algún botón de esa columna pulsado, se hará un cortocircuito entre esta y el pin de fila correspondiente. Por ello, si se implementa una especie de registro de desplazamiento que consiga la siguiente secuencia: 0111, 1011, 1101, 1110; comprobando en cada paso si algún valor de fila está a nivel bajo, se puede localizar en la matriz la tecla pulsada.

Concluyendo, las filas y columnas del teclado serán las entradas y salidas de nuestro sistema respectivamente.

Vemos las modificaciones realizadas a nivel hardware sobre el archivo .vhd de la práctica anterior. En la entity, añadimos las entradas y salidas necesarias:

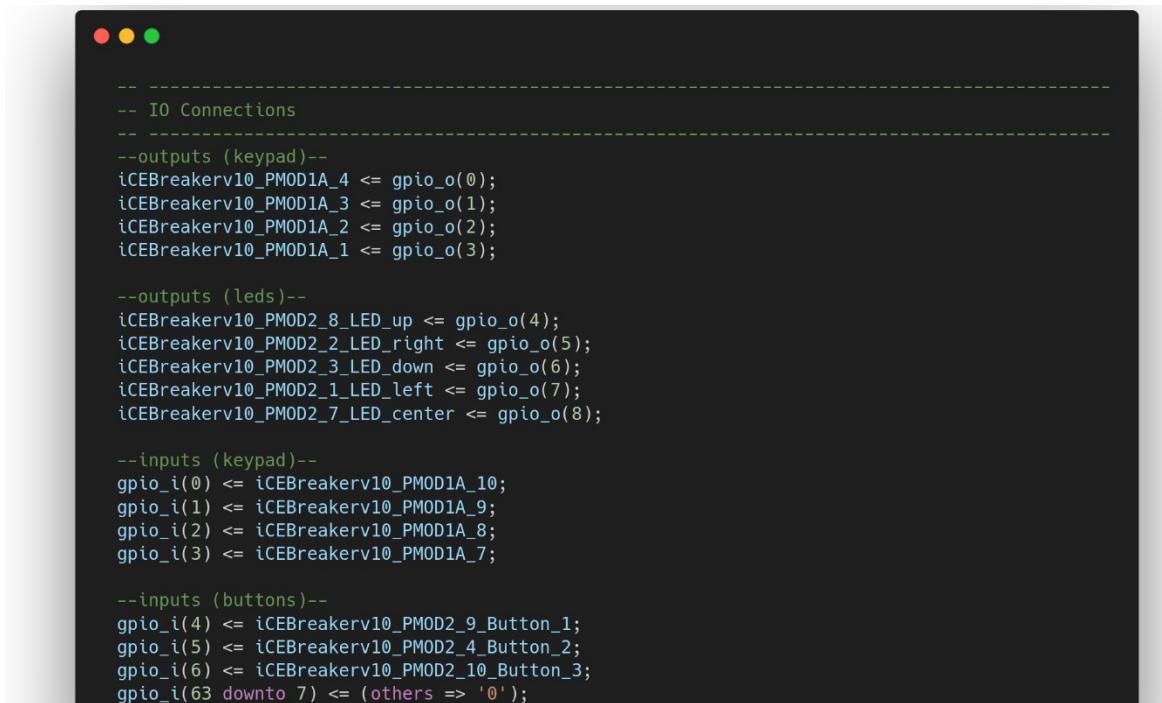
```

-- Keyboard columns
iCEBreakerv10_PMOD1A_1 : out std_logic;
iCEBreakerv10_PMOD1A_2 : out std_logic;
iCEBreakerv10_PMOD1A_3 : out std_logic;
iCEBreakerv10_PMOD1A_4 : out std_logic;

-- Keyboard rows
iCEBreakerv10_PMOD1A_7 : in std_logic;
iCEBreakerv10_PMOD1A_8 : in std_logic;
iCEBreakerv10_PMOD1A_9 : in std_logic;
iCEBreakerv10_PMOD1A_10 : in std_logic

```

Con vistas a la unificación de la práctica 1 con la actual, para evitar solapes en las posiciones del puerto de entrada y salida, se ha decidido cambiar los pines asociados a la máquina de estados que implementamos, además de añadir los necesarios para esta segunda práctica, resultando:



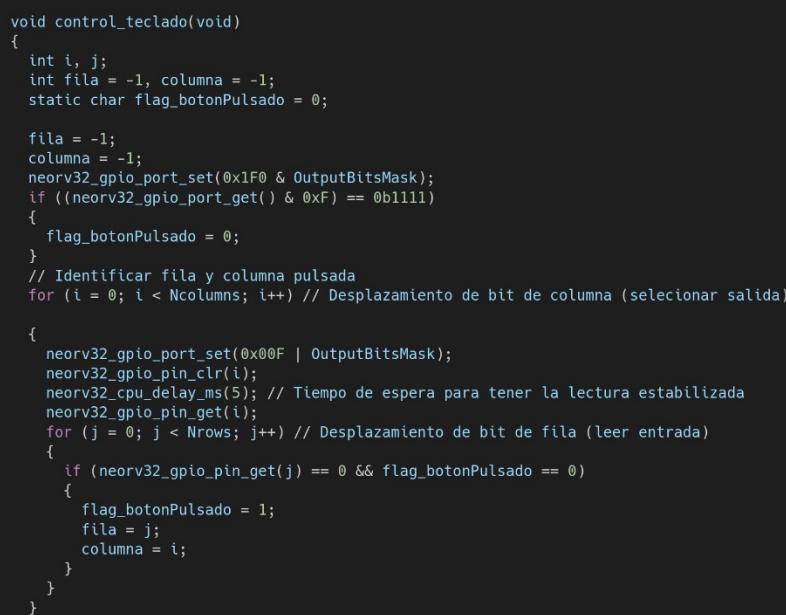
```
-- IO Connections
--outputs (keypad)
iCEBreakerv10_PMOD1A_4 <= gpio_o(0);
iCEBreakerv10_PMOD1A_3 <= gpio_o(1);
iCEBreakerv10_PMOD1A_2 <= gpio_o(2);
iCEBreakerv10_PMOD1A_1 <= gpio_o(3);

--outputs (leds)
iCEBreakerv10_PMOD2_8_LED_up <= gpio_o(4);
iCEBreakerv10_PMOD2_2_LED_right <= gpio_o(5);
iCEBreakerv10_PMOD2_3_LED_down <= gpio_o(6);
iCEBreakerv10_PMOD2_1_LED_left <= gpio_o(7);
iCEBreakerv10_PMOD2_7_LED_center <= gpio_o(8);

--inputs (keypad)
gpio_i(0) <= iCEBreakerv10_PMOD1A_10;
gpio_i(1) <= iCEBreakerv10_PMOD1A_9;
gpio_i(2) <= iCEBreakerv10_PMOD1A_8;
gpio_i(3) <= iCEBreakerv10_PMOD1A_7;

--inputs (buttons)
gpio_i(4) <= iCEBreakerv10_PMOD2_9_Button_1;
gpio_i(5) <= iCEBreakerv10_PMOD2_4_Button_2;
gpio_i(6) <= iCEBreakerv10_PMOD2_10_Button_3;
gpio_i(63 downto 7) <= (others => '0');
```

Centrándonos ahora en la capa software, se consigue detectar la tecla pulsada con la idea presentada antes, simulando un registro de desplazamiento con la secuencia de salidas comentada. Para ello se recurre a un bucle for que selecciona la columna que se debe poner a 0 en cada iteración, estando el resto a 1. Dentro de este bucle, tras un determinado tiempo de estabilización para la correcta lectura de las entradas (filas), se miran sus valores en otro bucle for anidado, comprobándose si se da el caso en el que tanto entrada como salida estén a 0. Dicho caso significará que la tecla en cuestión está pulsada, guardándose la información de la fila y columna correspondiente para ser procesada con vistas a la representación en pantalla.



```
void control_teclado(void)
{
    int i, j;
    int fila = -1, columna = -1;
    static char flag_botonPulsado = 0;

    fila = -1;
    columna = -1;
    neorv32_gpio_port_set(0x1F0 & OutputBitsMask);
    if ((neorv32_gpio_port_get() & 0xF) == 0b1111)
    {
        flag_botonPulsado = 0;
    }
    // Identificar fila y columna pulsada
    for (i = 0; i < Ncolumns; i++) // Desplazamiento de bit de columna (seleccionar salida)

    {
        neorv32_gpio_port_set(0x00F | OutputBitsMask);
        neorv32_gpio_pin_clr(i);
        neorv32_cpu_delay_ms(5); // Tiempo de espera para tener la lectura estabilizada
        neorv32_gpio_pin_get(i);
        for (j = 0; j < Nrows; j++) // Desplazamiento de bit de fila (leer entrada)
        {
            if (neorv32_gpio_pin_get(j) == 0 && flag_botonPulsado == 0)
            {
                flag_botonPulsado = 1;
                fila = j;
                columna = i;
            }
        }
    }
}
```

Una vez detectada la tecla como se ha explicado, se escribirá el carácter recibido por la UART mediante el uso de las sentencias switch-case.

```

// Sentencias condicionales para decidir carácter a enviar por UART
switch (fila)
{
    case 0:
        switch (columna)
        {
            case 0:
                neorv32_uart0_print("1");
                break;

            case 1:
                neorv32_uart0_print("2");
                break;

            case 2:
                neorv32_uart0_print("3");
                break;

            case 3:
                neorv32_uart0_print("A");
                break;
        }
        break;

    case 1:
        switch (columna)
        {
            case 0:
                neorv32_uart0_print("4");
                break;

            case 1:
                neorv32_uart0_print("5");
                break;

            case 2:
                neorv32_uart0_print("6");
                break;

            case 3:
                neorv32_uart0_print("B");
                break;
        }
        break;
}

case 2:
    switch (columna)
    {
        case 0:
            neorv32_uart0_print("7");
            break;

        case 1:
            neorv32_uart0_print("8");
            break;

        case 2:
            neorv32_uart0_print("9");
            break;

        case 3:
            neorv32_uart0_print("C");
            break;
    }
    break;

case 3:
    switch (columna)
    {
        case 0:
            neorv32_uart0_print("0");
            break;

        case 1:
            neorv32_uart0_print("F");
            break;

        case 2:
            neorv32_uart0_print("E");
            break;

        case 3:
            neorv32_uart0_print("D");
            break;
    }
    break;
}

```

Ampliaciones realizadas

Con lo comentado anteriormente no se conseguiría detectar una única pulsación, es decir, si se mantiene una tecla se mostrará continuamente su valor correspondiente en pantalla. Para corregir esto, ponemos todas las salidas a 0 y comprobamos si alguna entrada está a nivel bajo. Si fuera el caso, significaría que hay alguna tecla pulsada. En caso contrario no habría ninguna y por tanto desactivaríamos la bandera `flag_botonPulsado` que nos indica si ya se ha dejado de pulsar.

Al entrar a la sentencia condicional donde se comprueba la fila pulsada (dentro del bucle que selecciona la columna), se revisará a su vez la bandera mencionada anteriormente. De esta forma, sólo se cambiará el valor de fila y columna detectado en la pulsación cuándo la bandera estuviera previamente a 0, es decir, sea la primera vez que se registra dicha pulsación. Por ello, si este fuera el caso, activaríamos la bandera a 1 hasta que nuevamente se dejara de pulsar la tecla.

Por otro lado, queremos unificar las funcionalidades de esta práctica y la anterior. Para aunar las funciones `control_teclado()` y `maquina_estados()` en la misma ejecución, se recurre a una máscara de bits, que se va actualizando cada vez que una salida de la máquina de estados cambia. Esto se implementa así para proteger dichos cambios de ser eliminados por el control del teclado a la hora de usar la función `neorv32_gpio_port_set()`. Más en detalle, se recurre a la codificación en binario en C y el desplazamiento de bits, además de las operaciones lógicas AND y OR para mantener o modificar el valor de los bits de la máscara.

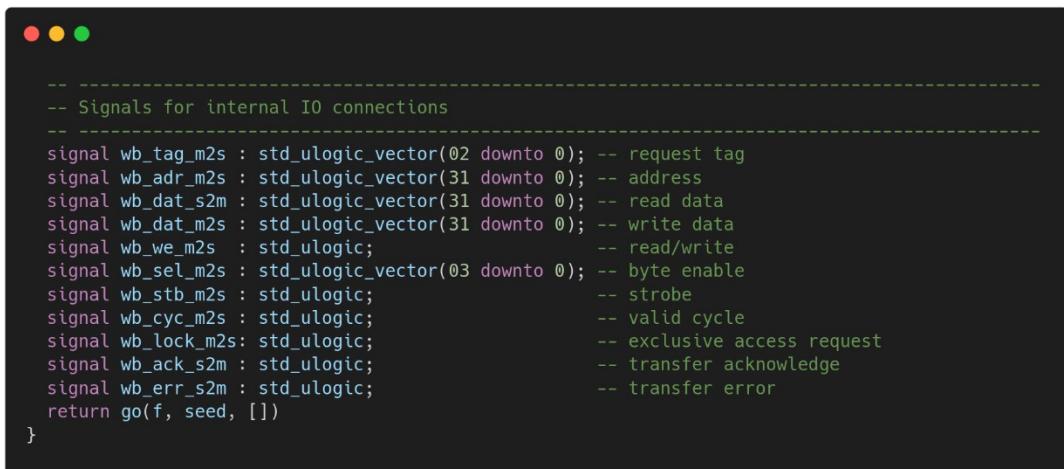
Todo lo comentado puede verse en los códigos adjuntos.

Práctica 3

La temática de esta práctica versa en conectar un periférico Wishbone con el que contamos, descrito en VHDL, al microprocesador neorv32 con vistas a establecer una comunicación entre ambos. Este periférico será un conjunto de registros.

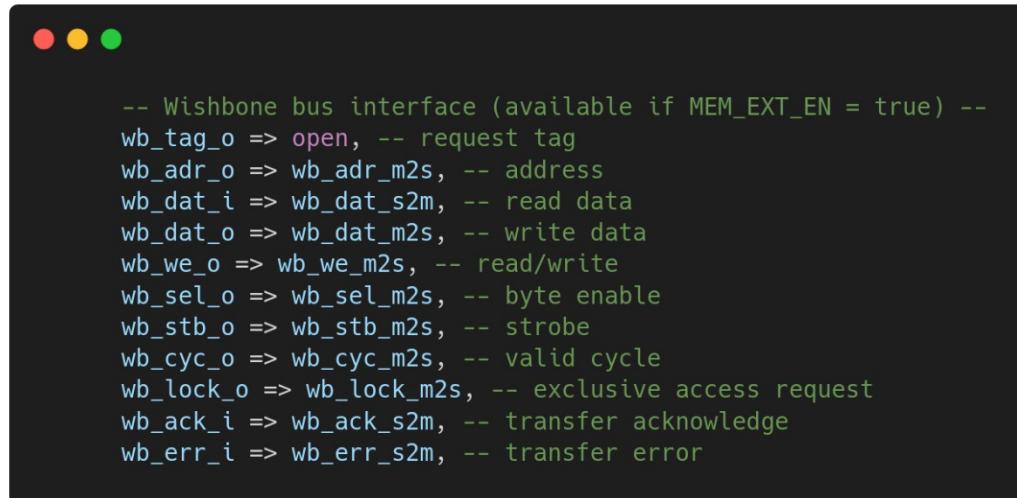
Primeramente, llevaremos a cabo las modificaciones hardware, por medio del archivo .vhd tal y como venimos haciendo de prácticas anteriores.

Creamos las señales necesarias para la conexión entre pines del microprocesador y periférico.



```
-- Signals for internal IO connections
-- 
signal wb_tag_m2s : std_ulogic_vector(02 downto 0); -- request tag
signal wb_dat_m2s : std_ulogic_vector(31 downto 0); -- address
signal wb_dat_s2m : std_ulogic_vector(31 downto 0); -- read data
signal wb_dat_m2s : std_ulogic_vector(31 downto 0); -- write data
signal wb_we_m2s : std_ulogic; -- read/write
signal wb_sel_m2s : std_ulogic_vector(03 downto 0); -- byte enable
signal wb_stb_m2s : std_ulogic; -- strobe
signal wb_cyc_m2s : std_ulogic; -- valid cycle
signal wb_lock_m2s: std_ulogic; -- exclusive access request
signal wb_ack_s2m : std_ulogic; -- transfer acknowledge
signal wb_err_s2m : std_ulogic; -- transfer error
return go(f, seed, [])
}
```

Seguidamente, conectamos en el port map de la instancia del neorv32 las señales a las respectivas entradas/salidas del micro.



```
-- Wishbone bus interface (available if MEM_EXT_EN = true) --
wb_tag_o => open, -- request tag
wb_dat_i => wb_dat_s2m, -- read data
wb_dat_o => wb_dat_m2s, -- write data
wb_we_o => wb_we_m2s, -- read/write
wb_sel_o => wb_sel_m2s, -- byte enable
wb_stb_o => wb_stb_m2s, -- strobe
wb_cyc_o => wb_cyc_m2s, -- valid cycle
wb_lock_o => wb_lock_m2s, -- exclusive access request
wb_ack_i => wb_ack_s2m, -- transfer acknowledge
wb_err_i => wb_err_s2m, -- transfer error
```

Por último, se conecta el otro extremo de las señales con el periférico, instanciándolo con anterioridad a nombre de neorv32.wb_regs.

```

-- Instance the peripheral
--

wb_regs_inst : entity neorv32.wb_regs
generic map(
    WB_ADDR_BASE => x"80002000", --mirando dmem_size=8*1024 bytes -> start 80millones + 8192 -> en
hex x"..." (primer hueco de memoria libre)
    WB_ADDR_SIZE => 16          -- tenemos 4 registros (mirar wb_regs.vhd) disponibles de 32 bits (4
bytes) cada uno -> 4x4=16 bytes de memoria necesarios, acceso de escritura/lectura mínimo de 4 bytes
)
port map(
    wb_clk_i  => std_ulogic(iCEBreakerv10_CLK),                                -- clock
    wb_rstn_i => std_ulogic(iCEBreakerv10_BTN_N),                                -- reset, async, low-active
    wb_adr_i   => wb_adr_m2s, -- address
    wb_dat_i   => wb_dat_m2s, -- read data
    wb_dat_o   => wb_dat_s2m, -- write data
    wb_we_i    => wb_we_m2s, -- read/write
    wb_sel_i   => wb_sel_m2s, -- byte enable
    wb_stb_i   => wb_stb_m2s, -- strobe
    wb_cyc_i   => wb_cyc_m2s, -- valid cycle
    wb_ack_o   => wb_ack_s2m, -- transfer acknowledge
    wb_err_o   => wb_err_s2m -- transfer error
);

```

Aquí cabe destacar varios puntos:

- En el generic map, la dirección base que se usará será la 0x80002000. Vemos como la zona de memoria de datos reservada (DMEM) ocupa 8 Kbytes y la primera zona de direcciones (hasta la 0xFFFFFFFF) corresponde al espacio de instrucciones.

```

constant MEM_INT_DMEM_SIZE : natural := 8 * 1024; -- size of processor-internal data memory in bytes

```

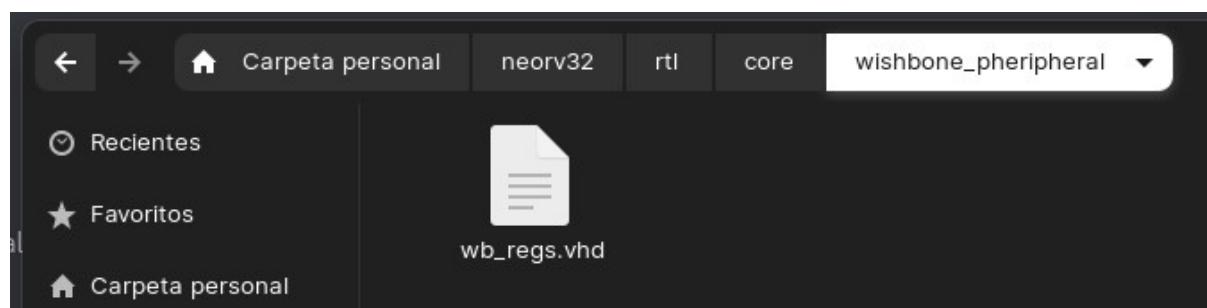
Por tanto, $80M+8*1024=8192$ (bytes). La primera dirección será la que usamos que se corresponde al primer hueco libre de memoria de datos.

- Por otro lado, el tamaño de memoria será de 16 bytes, ya que el tamaño de los registros de nuestro periférico es de 32 bits (4 bytes), como puede verse en wb_regs.vhd, y contamos con 4 de ellos.

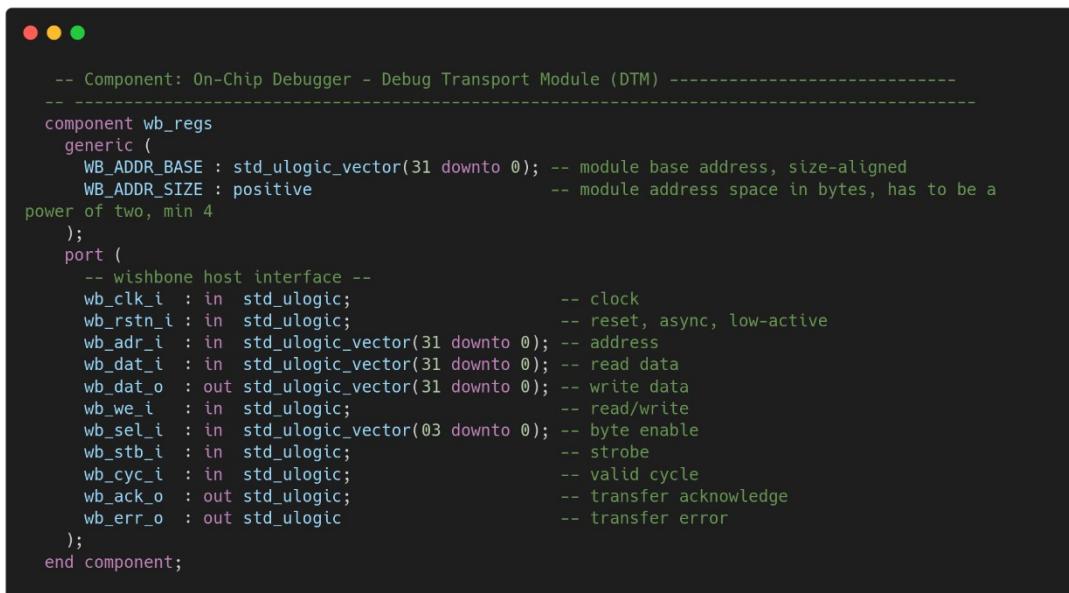
A su vez, para poder generar la instancia correctamente, debemos tener configurado el fichero filesets.mk para disponer del archivo wb_regs.vhd como fuente. Para ello, dentro de este, añadimos la variable RTL_PHERIPH_SRC que contenga la ruta a la carpeta donde está contenido el archivo VHDL y usándola para guardar dentro de la nueva variable NEORV32_PERIPH_SRC, teniendo de esta forma la ruta al fichero en sí.

Finalmente la añadimos a NEORV32_SRC ya existente para tenerla en cuenta.

Es por ello por lo que a su vez debemos localizar el archivo que contiene la descripción hardware del periférico en la carpeta creada dentro de rtl/core para este propósito:



Por último, añadimos el componente correspondiente a nuestro periférico dentro del neorv_package.vhd.

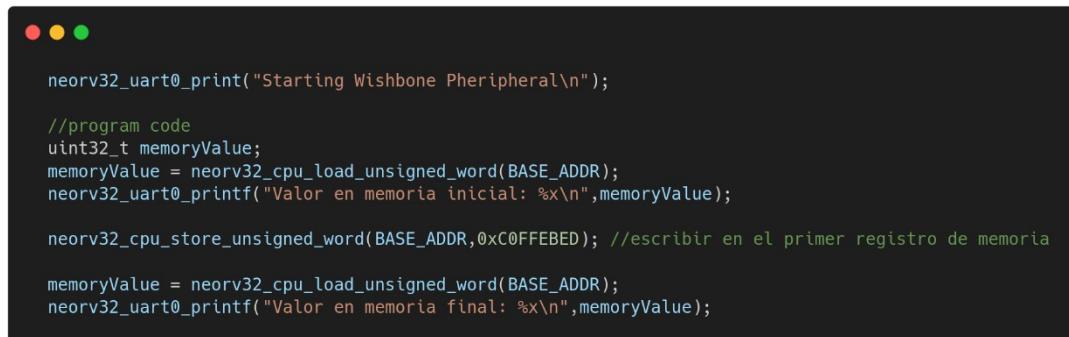


```
-- Component: On-Chip Debugger - Debug Transport Module (DTM) -----
component wb_regs
generic (
    WB_ADDR_BASE : std_ulogic_vector(31 downto 0); -- module base address, size-aligned
    WB_ADDR_SIZE : positive                         -- module address space in bytes, has to be a
power of two, min 4
);
port (
    -- wishbone host interface --
    wb_clk_i : in std_ulogic;                      -- clock
    wb_rstn_i : in std_ulogic;                      -- reset, async, low-active
    wb_adr_i : in std_ulogic_vector(31 downto 0);   -- address
    wb_dat_i : in std_ulogic_vector(31 downto 0);   -- read data
    wb_dat_o : out std_ulogic_vector(31 downto 0);  -- write data
    wb_we_i : in std_ulogic;                        -- read/write
    wb_sel_i : in std_ulogic_vector(03 downto 0);  -- byte enable
    wb_stb_i : in std_ulogic;                      -- strobe
    wb_cyc_i : in std_ulogic;                      -- valid cycle
    wb_ack_o : out std_ulogic;                     -- transfer acknowledge
    wb_err_o : out std_ulogic;                     -- transfer error
);
end component;
```

Con todo ello, ya se tendría completada la configuración hardware. Una vez hecho esto solo quedaría reimplementar.

Pasando ahora a la capa software, sólo habría que escribir un código simple utilizando las funciones desarrolladas para la lectura y escritura en direcciones de memoria, que están dentro de la librería de la CPU.

`neorv32_cpu_load_unsigned_word()` : para lectura del registro
`neorv32_cpu_store_unsigned_word()` : para escritura del registro



```
neorv32_uart0_print("Starting Wishbone Peripheral\n");

//program code
uint32_t memoryValue;
memoryValue = neorv32_cpu_load_unsigned_word(BASE_ADDR);
neorv32_uart0_printf("Valor en memoria inicial: %x\n",memoryValue);

neorv32_cpu_store_unsigned_word(BASE_ADDR,0xC0FFEBED); //escribir en el primer registro de memoria

memoryValue = neorv32_cpu_load_unsigned_word(BASE_ADDR);
neorv32_uart0_printf("Valor en memoria final: %x\n",memoryValue);
```

Por tanto, en nuestro programa, se procede primero leyendo el valor dentro del primer registro (con dirección de memoria `BASE_ADDR=0x80002000`), mostrando el resultado en hexadecimal por pantalla gracias a la UART desde cutecom. Tras esto, escribimos la palabra `0xC0FFEBED` y revisamos si este cambio ha sido efectivo volviendo a realizar una lectura y mostrando el valor leído.

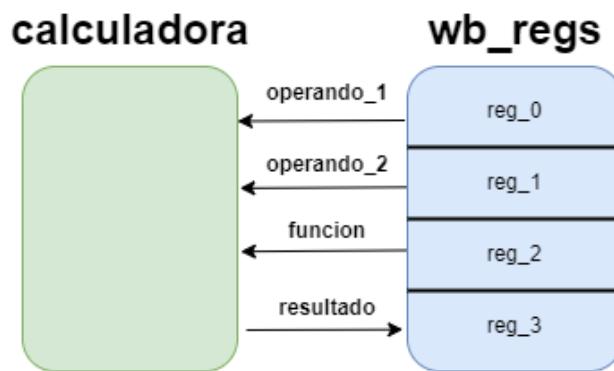
```
[17:17:41:651] Starting Wishbone Peripheral
[17:17:41:666] Valor en memoria inicial: 27637dc3
[17:17:41:698] Valor en memoria final: c0ffebed
```

De la misma forma, se ha realizado la lectura y escritura en el resto de los registros disponibles.

Hasta aquí la parte básica de esta tercera práctica. Hemos corroborado el correcto funcionamiento de nuestro periférico.

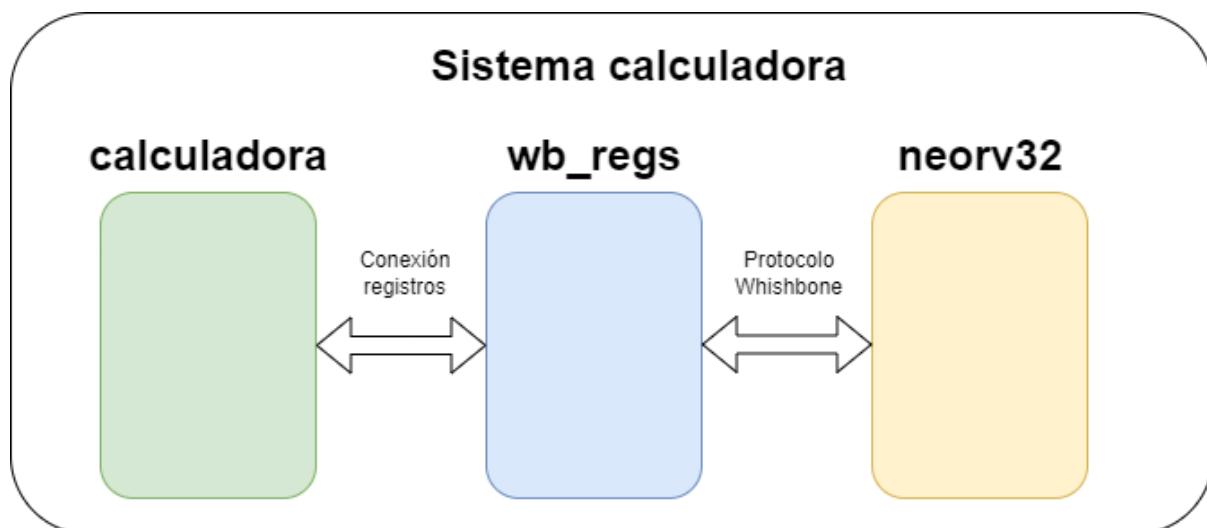
Ampliaciones realizadas

Proponemos como mejora a esta tercera práctica la implementación a nivel hardware de un módulo que haga las veces de calculadora y la anexión al esquema que tenemos hasta ahora. Dicho módulo estará conectado con el periférico de registros que estamos usando. Será un bloque combinacional que a partir de los operandos y la operación matemática seleccionada devolverá el resultado de dicha operación. Los operandos corresponderán a los dos primeros registros del periférico wb_regs y la operación a realizar estará contenida en el tercer registro. El resultado se almacena en todo momento en el cuarto de los registros. De esta manera, ya hemos definido cual será el papel de cada registro en el sistema. Podemos intuir cuales serán las entradas y salidas del nuevo bloque.



Por tanto, la forma de cambiar los operandos y la función (suma, resta o multiplicación) será escribiendo el valor del registro correspondiente del periférico. Cuando se necesita conocer el resultado de la operación se leerá el último registro.

El esquema que se muestra a continuación recoge un resumen a nivel de bloques funcionales del sistema completo.



En la próxima práctica se incluirán nuevos bloques para hacer un sistema más completo.

Hardware:

En primer lugar, vemos el módulo calculadora.vhd realizado. Como puede verse, el código es sencillo, en un proceso combinacional se comprueba en la lista de sensibilidad si ha habido cambios en el operando 1, operando 2 o la función, y de ser el caso, actualiza el resultado sumando (función=0), restando (función=1) o multiplicando (función=3).

Vemos la entidad del módulo, simplemente cuenta con las entradas y salida que como se ha dicho irán conectadas de forma directa con el periférico de registros.

```
entity calculadora is
    Port ( operando_1 : in STD_LOGIC_VECTOR(9 downto 0);
           operando_2 : in STD_LOGIC_VECTOR(9 downto 0);
           funcion : in STD_ULOGIC_VECTOR(1 downto 0);
           resultado : out STD_LOGIC_VECTOR(31 downto 0));
end calculadora;
```

La arquitectura cuenta con tres procesos. El primero de ellos se encarga de estructurar las señales asociadas a los dos operandos en función del signo de los mismos. Esto es necesario para que las operaciones se realicen en el siguiente process correctamente, teniendo en cuenta la codificación en complemento a 2.

```
signo_op: process (operando_1,operando_2)
begin
    if operando_1(9) = '1' then --negativo
        operando_1_s(9) <= '1';
    else --positivo
        operando_1_s(9) <= '0';
    end if;
    operando_1_s(10) <= (operando_1(9));
    operando_1_s(8 downto 0) <= signed(operando_1(8 downto 0));

    if operando_2(9) = '1' then --negativo
        operando_2_s(9) <= '1';
    else --positivo
        operando_2_s(9) <= '0';
    end if;
    operando_2_s(10) <= (operando_2(9));
    operando_2_s(8 downto 0) <= signed(operando_2(8 downto 0));
end process;
```

En este segundo proceso es donde se realizan las operaciones de suma, resta y multiplicación con signo.

```
calculo: process(operando_1_s,operando_2_s, funcion)
begin
    resultado_s <= (others => '0');
    case to_integer(unsigned(funcion)) is
        when 0 => --sumar
            resultado_s(10 downto 0) <= (operando_1_s+operando_2_s);
        when 1 => --restar
            resultado_s(10 downto 0) <= (operando_1_s-operando_2_s);
        when 2 => --multiplicar
            resultado_s(21 downto 0) <= (operando_1_s*operando_2_s);
        when others =>
            resultado_s <= (others => '1');
    end case;
end process;
```

En el tercero de los procesos interpretamos el resultado obtenido del anterior según el tipo de operación que se esté realizando y el signo del resultado, adaptamos el resultado a complemento a 2.

```
resul: process (funcion,resultado_s)
begin
    if funcion(1) = '0' then --caso de suma o resta el bit de signo será el 10
        if resultado_s(10) = '1' then --negativo
            resultado(30 downto 10) <= (others => '1');
        else --positivo
            resultado(30 downto 10) <= (others => '0');
        end if;
        resultado(31) <= (resultado_s(10));
        resultado(9 downto 0) <= std_logic_vector(resultado_s(9 downto 0));
    else --caso de multiplicación el bit de signo será el 22
        if resultado_s(21) = '1' then --negativo
            resultado(30 downto 21) <= (others => '1');
        else --positivo
            resultado(30 downto 21) <= (others => '0');
        end if;
        resultado(31) <= (resultado_s(21));
        resultado(20 downto 0) <= std_logic_vector(resultado_s(20 downto 0));
    end if;
end process;
```

Cabe destacar que si en lugar de haber realizado las operaciones sin tener en cuenta los signos o suponiendo que simplemente los operandos serán positivos en todo momento se podría haber resuelto con un solo proceso más sencillo. Sin embargo, hemos querido que nuestra calculadora fuera capaz de realizar operaciones teniendo en cuenta el signo de los operandos.

Hemos tenido que modificar para nuestra aplicación el fichero original wb_regs.vhd incluyéndole las entradas y salidas necesarias para la conexión con el módulo anterior.

Las diferencias a destacar son:

- En la entidad, se ha añadido en el port las entradas y salidas para la conexión directa de los datos de los registros con el módulo calculadora.vhd.

```
entity wb_regs is
    generic (
        WB_ADDR_BASE : std_ulegic_vector(31 downto 0); -- module base address, size-aligned
        WB_ADDR_SIZE : positive -- module address space in bytes, has to be a power of two, min 4
    );
    port (
        -- wishbone host interface --
        (...)

        -- calculator i/o --
        dat_reg_0 : out std_ulegic_vector(31 downto 0); -- register 1 data output
        dat_reg_1 : out std_ulegic_vector(31 downto 0); -- register 2 data output
        dat_reg_2 : out std_ulegic_vector(31 downto 0); -- register 3 data output
        dat_reg_3 : in std_ulegic_vector(31 downto 0) -- register 4 data input
    );
end wb_regs;
```

- En la arquitectura, de forma combinacional y asíncrona, se conecta directamente los valores de los registros a los puertos de salida y entrada asociados a la calculadora.

```
-- Direct connection
dat_reg_0 <= reg0;
dat_reg_1 <= reg1;
dat_reg_2 <= reg2;
reg3 <= dat_reg_3;
```

También debemos hacer los cambios pertinentes en el .vhd top partiendo de la parte básica. Modificamos la instancia de wb_regs para incluir las nuevas entradas y salidas, creamos una instancia del bloque combinacional calculadora.vhd e interconectamos por medio de señales internas ambos módulos.

```
calculadora_inst : entity neorv32.calculadora
port map(
    operando_1 => operando_1_s(9 downto 0),
    operando_2 => operando_2_s(9 downto 0),
    funcion     => funcion_s(1 downto 0),
    resultado   => resultado_s
);
```

Por último, incluimos en el package el nuevo componente, corregimos el correspondiente a wb_regs y editamos el filesets.mk para incluir el nuevo archivo.

Software:

El código desarrollado lee valores por uart desde cutecom para la función y los operandos 1 y 2 y muestra el resultado de la operación.

Dentro del bucle sin fin del código, en primer lugar, leemos los operandos 1 y 2. Para la lectura de dichos operandos usamos la función neorv32_uart0_scan(). Dicha función nos devuelve la cadena de caracteres leída por la uart desde cutecom. Para traducirla a valores numéricos con signo hemos desarrollado la función char2int(). Tenemos también en cuenta los límites del rango numérico permitidos. En el caso en el que el operando recibido sea adecuado, se almacena en el registro correspondiente perteneciente a wb_regs.

```
while (1)
{
    // Operando1
    int32_t operando1 = 0;
    flagNumSup = 0;
    do
    {
        neorv32_uart0_print("Operando 1: ");
        neorv32_uart0_scan(buffer, 20, 1);
        neorv32_uart0_print("\n");
        operando1 = (uint32_t)char2int(buffer);
        if ((operando1 > 511) || (operando1 < -512))
        {
            neorv32_uart0_print("Número demasiado grande, rango entre -512 y 511. Vuelva a introducirlo.\n");
            flagNumSup = 1;
        }
        else flagNumSup = 0;
    } while (flagNumSup == 1 || flagError == 1);
    neorv32_cpu_store_unsigned_word(BASE_ADDR, operando1);
```

De forma simétrica a la mostrada, también se procede con el operando 2.

Por otro lado, la función char2int() comentada se ha desarrollado para evitar recurrir a las funciones de alto nivel de C, las cuales supondrían un consumo excesivo de recursos. Teniendo en cuenta la codificación ASCII, filtra todo carácter distinto a un número, a excepción del signo '-' al principio permitido para indicar negativos, mostrando error en caso necesario.

```

int32_t char2int(char buffer[])
{
    int32_t num = 0;
    int i = 0;
    char flagSigno = 0;

    flagError = 0;
    while (buffer[i] != '\0')
    {
        if (buffer[i] == '-' && i==0)
        {
            flagSigno = 1;
        }
        else
        {
            if((buffer[i] < 48) || (buffer[i] > 57)) //si no es un número y no es signo -
            {
                neorv32_uart0_print("Carácter no válido, vuelve a introducirlo.\n");
                flagError = 1;
                return 0;
            }
            else
                num = num * 10 + (buffer[i] - 48);
        }
        i++;
    }
    if (flagSigno)
    {
        num *= -1;
    }
    return num;
}

```

Para finalizar, comprobamos que el valor de los operandos se ha escrito correctamente en los registros. Una vez ya escritos en los registros correspondientes, el resultado estará preparado inmediatamente en el registro 3, por lo que en un bucle aplicamos cada una de las funciones disponibles, para verificar que todas las operaciones de la calculadora funcionan. Leyendo los registros del periférico se le muestra al usuario por pantalla con uso de la UART.

```

neorv32_uart0_print("Valores obtenidos de los registros: \n");
    int i;
    for (i = 0; i < 3; i++)
    {
        // Funcion
        neorv32_cpu_store_unsigned_word(BASE_ADDR + 0x8, i);
        // Resultado
        memoryValue = neorv32_cpu_load_unsigned_word(BASE_ADDR);
        neorv32_uart0_printf("OPERANDO 1: %i\n", memoryValue);
        memoryValue = neorv32_cpu_load_unsigned_word(BASE_ADDR + 4);
        neorv32_uart0_printf("OPERANDO 2: %i\n", memoryValue);
        memoryValue = neorv32_cpu_load_unsigned_word(BASE_ADDR + 8);

        switch (memoryValue)
        {
        case 0:
            neorv32_uart0_print("FUNCION SUMA\n");
            break;
        case 1:
            neorv32_uart0_print("FUNCION RESTA\n");
            break;
        case 2:
            neorv32_uart0_print("FUNCION MULTIPLICACION\n");
            break;
        }
        memoryValue = neorv32_cpu_load_unsigned_word(BASE_ADDR + 12);
        neorv32_uart0_printf("RESULTADO: %i\n", memoryValue);
        neorv32_uart0_print("/*-----*/\n");
    }
    neorv32_uart0_print("\n*****\n");
}

```

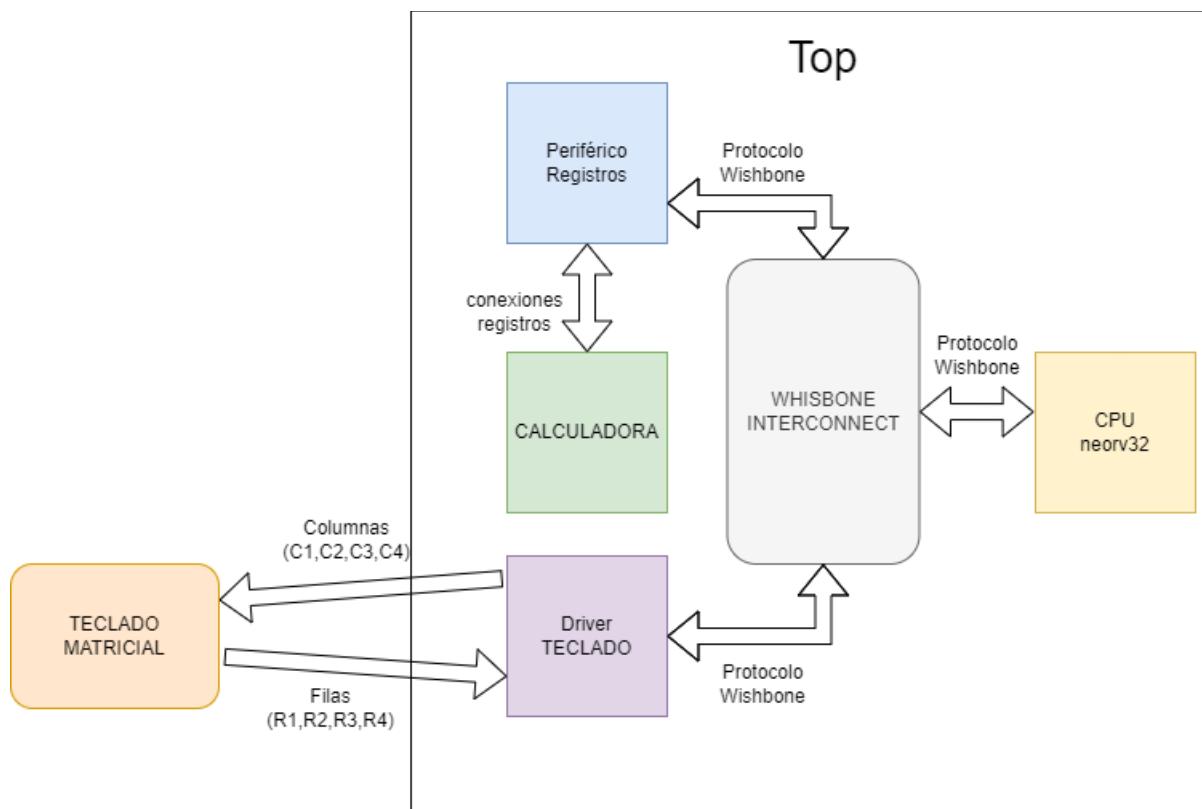
A continuación, se muestran algunos ejemplos de resultados obtenidos, verificando de esta forma el comportamiento deseado.

```
[20:32:51:408] Operando 1: -43h f
[20:33:03:119] Operando 2: 12h f
[20:33:06:047] Valores obtenidos de los registros: r8 f
[20:33:06:064] OPERANDO 1: -43h f
[20:33:06:079] OPERANDO 2: 12h f
[20:33:06:079] FUNCION SUMA r8 f
[20:33:06:096] RESULTADO: -31h f
[20:33:06:096] /*-----*/ r8 f
[20:33:06:111] OPERANDO 1: -43h f
[20:33:06:111] OPERANDO 2: 12h f
[20:33:06:127] FUNCION RESTA r8 f
[20:33:06:127] RESULTADO: -55h f
[20:33:06:143] /*-----*/ r8 f
[20:33:06:159] OPERANDO 1: -43h f
[20:33:06:159] OPERANDO 2: 12h f
[20:33:06:175] FUNCION MULTIPLICACION r8 f
[20:33:06:191] RESULTADO: -516h f
[20:33:06:191] /*-----*/ r8 f
[20:33:06:207] r8 f
[20:33:06:207] =====*
```

Práctica 4

El objetivo escogido para esta práctica será el de ampliar el sistema que se consiguió en la práctica 3 ampliada usando componentes con los que se han trabajado a lo largo de las prácticas de la asignatura. Incluiríremos el teclado numérico hexadecimal PMOD Keypad de Digilent de la práctica 2 como parte de la interfaz usuario-máquina. La interpretación de la tecla pulsada en este caso será vía hardware. Realizaremos un nuevo periférico que se utilizará de driver entre teclado y neorv32. Al incluir dicho nuevo periférico, necesitamos incorporar un interconnect Wishbone que gestione tanto al driver del teclado como al wb_regs.

La arquitectura funcional del sistema completo quedaría de la siguiente forma:



MODO DE FUNCIONAMIENTO CALCULADORA (MANUAL DE USUARIO)

- Tecla A: Seleccionar modo SUMA
- Tecla B: Seleccionar modo RESTA
- Tecla C: Seleccionar modo MULTIPLICACION
- Tecla F: Seleccionar escritura del primer operando
- Tecla E: Seleccionar escritura del segundo operando
- Tecla D: Mostrar operación con resultado por pantalla
- Resto teclas: Escribir operandos (tras seleccionar F o E)
- Tecla A (tras pulsar F o E): Poner signo negativo al operando
- Combinación Tecla 1 + Tecla 0: “Easter-egg”

Hardware:

El bloque driver_teclado tiene como entradas las filas del teclado y como salidas las columnas del mismo. Es el encargado de dar las combinaciones de columnas valores altos y bajos necesarios en el teclado para que al leer las entradas del teclado sepamos si una tecla está activada o no. En resumen, se encarga de realizar la función que en la práctica 2 se hacía de manera software. Además, este periférico cuenta con más entradas y salidas, las correspondientes al protocolo wishbone.

```

entity driver_teclado is
  generic (
    WB_ADDR_DRIVER : std_ulegic_vector(31 downto 0) -- driver address
  );
  port (
    -- wishbone host interface --
    wb_clk_i : in std_ulegic; -- clock
    wb_rstn_i : in std_ulegic; -- reset, async, low-active
    wb_adr_i : in std_ulegic_vector(31 downto 0); -- address
    wb_dat_i : in std_ulegic_vector(31 downto 0); -- read data
    wb_dat_o : out std_ulegic_vector(31 downto 0); -- write data
    wb_we_i : in std_ulegic; -- read/write
    wb_sel_i : in std_ulegic_vector(03 downto 0); -- byte enable
    wb_stb_i : in std_ulegic; -- strobe
    wb_cyc_i : in std_ulegic; -- valid cycle
    wb_ack_o : out std_ulegic; -- transfer acknowledge
    wb_err_o : out std_ulegic; -- transfer error
    -- rows/columns keypad i/o --
    rows : in std_ulegic_vector(3 downto 0);
    columns : out std_ulegic_vector(3 downto 0)
  );
end driver_teclado;

```

En la zona de la architecture hemos usado la técnica de diseño con dos procesos, combinacional y síncrono. Cada vez que un contador de 16 bits desborda (5 ms aproximadamente teniendo en cuenta la frecuencia de clk de trabajo) leemos las filas del teclado y almacenamos el valor leído en una señal de datos de 16 bits, de forma ordenada, según la columna actual de las 4 que estamos poniendo a 0. Así mismo realizamos un desplazamiento a izquierdas de la señal de 4 bits que almacena la combinación de valores que tomarán las columnas en cada momento. De esta manera, tendremos en todo momento un conjunto de datos de 16 bits ordenados desde 0 (bit menos significativo) a F (bit más significativo) que recogen la información para saber si la tecla correspondiente está pulsada (0) o no (1). En el siguiente ejemplo se observa el contenido que tendría si se pulsa la tecla A y 4 simultáneamente.

La interpretación de esta información se hará vía software para tomar decisiones en el sistema completo.

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1

Centrándonos en el combinacional podemos ver la forma de trabajar que hemos seguido.

```

comb : process (contador, datos, columns_s, cont_columns, rows, wb_cyc_i, wb_stb_i, wb_adr_i)
begin

    p_contador <= contador + 1;
    -- Valores por defecto (los próximos serán los actuales)
    p_columns_s <= columns_s;
    p_cont_columns <= cont_columns;
    p_datos <= datos;

    -- Cada vez que desborda, se leen filas y se cambian columnas (5ms)
    if (contador = 0) then

        --Lee filas
        case to_integer(unsigned(cont_columns)) is
            when 0 => --Column 0
                --Datos 1470
                p_datos(1) <= rows(0); --1
                p_datos(4) <= rows(1); --4
                p_datos(7) <= rows(2); --7
                p_datos(0) <= rows(3); --0
            when 1 => --Column 1
                --Datos 258F
                p_datos(2) <= rows(0); --2
                p_datos(5) <= rows(1); --5
                p_datos(8) <= rows(2); --8
                p_datos(15) <= rows(3); --F
            when 2 => --Column 2
                --Datos 369F
                p_datos(3) <= rows(0); --3
                p_datos(6) <= rows(1); --6
                p_datos(9) <= rows(2); --9
                p_datos(14) <= rows(3); --E
            when 3 => --Column 3
                --Datos ABCD
                p_datos(10) <= rows(0); --A
                p_datos(11) <= rows(1); --B
                p_datos(12) <= rows(2); --C
                p_datos(13) <= rows(3); --D
            when others =>
                p_datos <= datos;
        end case;

        --Desplaza columnas
        p_columns_s <= columns_s(2 downto 0) & columns_s(3);

        --Apunta columna por la que vamos
        p_cont_columns <= cont_columns + 1;
    end if;

    if (wb_cyc_i = '1') and (wb_stb_i = '1') and (wb_adr_i = WB_ADDR_DRIVER) then
        wb_ack_o <= '1';
    else
        wb_ack_o <= '0';
    end if;
end process;

```

En el proceso síncrono simplemente actualizamos el valor de los estados en cada flanko de la señal de reloj. También se da la inicialización de todo con el botón de reset de la placa.

```

sync : process (wb_clk_i, wb_rstn_i)
begin
    -- Inicialización
    if wb_rstn_i = '0' then
        columns_s <= "1110";
        cont_columns <= "00";
        contador <= (others => '0');
        datos <= (others => '1');

    elsif rising_edge(wb_clk_i) then
        contador <= p_contador;
        datos <= p_datos;
        columns_s <= p_columns_s;
        cont_columns <= p_cont_columns;

    end if;
end process;

```

Por último, conectamos las señales internas con las salidas del bloque.

```
● ● ●

columns <= columns_s;
wb_dat_o(15 downto 0) <= datos;
wb_dat_o(31 downto 16) <= (others => '0');
wb_err_o <= '0';
```

Explicaremos ahora el bloque wb_switch hecho. Éste selecciona un periférico u otro en función de la dirección que ponga el master (neov32) para realizar las tareas de comunicación de lectura y escritura.

Las entradas y salidas se dividen en dos bloques: las que estarán conectadas por un lado al neov32 y las que se conectarán a los dos periféricos disponibles.

```
● ● ●

entity wb_switch is
port (
    -- Whishbone Master Interface
    wb_mstr_adr_i      : in std_ulegic_vector(31 downto 0);
    wb_mstr_dat_i      : in std_ulegic_vector(31 downto 0);
    wb_mstr_dat_o      : out std_ulegic_vector(31 downto 0);
    wb_mstr_we_i       : in std_ulegic;
    wb_mstr_sel_i      : in std_ulegic_vector(03 downto 0);
    wb_mstr_stb_i      : in std_ulegic;
    wb_mstr_cyc_i      : in std_ulegic;
    wb_mstr_ack_o      : out std_ulegic;
    wb_mstr_err_o      : out std_ulegic;

    -- Whishbone Slave Interface
    wb_slv_adr_o       : out std_ulegic_vector(31 downto 0);
    wb_slv_dat_o       : out std_ulegic_vector(31 downto 0);
    wb_slv_dat_i_p0    : in std_ulegic_vector(31 downto 0);
    wb_slv_dat_i_p1    : in std_ulegic_vector(31 downto 0);
    wb_slv_we_o        : out std_ulegic;
    wb_slv_sel_o       : out std_ulegic_vector(03 downto 0);
    wb_slv_stb_o       : out std_ulegic_vector(1 downto 0);
    wb_slv_cyc_o       : out std_ulegic_vector(1 downto 0);
    wb_slv_ack_i       : in std_ulegic_vector(1 downto 0);
    wb_slv_err_i       : in std_ulegic_vector(1 downto 0)
);
end wb_switch;
```

En este caso únicamente necesitamos un proceso combinacional:

```
● ● ●

process
(wb_mstr_adr_i,wb_mstr_dat_i,wb_slv_dat_i_p0,wb_slv_dat_i_p1,wb_mstr_stb_i,wb_mstr_cyc_i,wb_slv_ack_i,wb_slv_err_i)
begin

--Escritura p0 / p1
wb_mstr_dat_o <= (others => '0');
wb_slv_dat_o <= wb_mstr_dat_i;

--p0 (wb_regs)
if (wb_mstr_adr_i(4) = '0') then
    if (wb_mstr_we_i = '0') then --Lectura
        wb_mstr_dat_o <= wb_slv_dat_i_p0;
        wb_slv_dat_o <= (others => '0');
    end if;
    -- Comunicación
    wb_slv_stb_o(0) <= wb_mstr_stb_i;
    wb_slv_cyc_o(0) <= wb_mstr_cyc_i;
    wb_slv_stb_o(1) <= '0';
    wb_slv_cyc_o(1) <= '0';
    wb_mstr_ack_o <= wb_slv_ack_i(0);
    wb_mstr_err_o <= wb_slv_err_i(0);
```

```
--p1 (driver_teclado)
else
  if (wb_mstr_we_i = '0') then --Lectura
    wb_mstr_dat_o <= wb_slv_dat_i_p1;
    wb_slv_dat_o <= (others => '0');
  end if;
  -- Comunicación
  wb_slv_stb_o(1) <= wb_mstr_stb_i;
  wb_slv_cyc_o(1) <= wb_mstr_cyc_i;
  wb_slv_stb_o(0) <= '0';
  wb_slv_cyc_o(0) <= '0';
  wb_mstr_ack_o <= wb_slv_ack_i(1);
  wb_mstr_err_o <= wb_slv_err_i(1);
end if;
end process;

--Conexiones directas
wb_slv_adr_o <= wb_mstr_adr_i;
wb_slv_we_o <= wb_mstr_we_i;
wb_slv_sel_o <= wb_mstr_sel_i;
```

Una vez que tenemos los dos bloques nuevos diseñados, en el fichero .vhd top debemos crear las instancias de estos e interconectar con señales internas el neorv32 con el wishbone interconnect, y este a su vez con ambos periféricos.

Por último, debemos incluir los nuevos components en el fichero package y añadir la ruta a los .vhd en fichero fileset.

Software:

El programa desarrollado se encarga de administrar las pulsaciones de teclas recibidas para cambiar la funcionalidad de la calculadora, cambiar los operandos de la cuenta que se hace y mostrar el resultado de acuerdo con el manual de usuario básico que se comentó antes.

Para ello, podemos dividir el programa desarrollado en 2 partes principales.

La primera de ellas se corresponde con la función principal, en la cual se lee continuamente el valor que proporciona el driver del teclado y se interpreta cuando se ha dado una pulsación en el mismo. Para ello, teniendo en cuenta que la información de las pulsaciones viene dada como vimos antes, con una ristra de 0 y 1 en las posiciones correspondientes a las teclas, usando una máscara variable que seleccione cada una de las posiciones y comparando dicha máscara con un AND lógico de la propia máscara y el negado del valor que proporciona el driver teclado, es fácil conseguir identificar la tecla pulsada en cada caso. Comprobamos todas las posibles teclas en un bucle for de 15 iteraciones. En el caso en el que se haya detectado alguna de las teclas, distinguimos en un switch...case las diferentes acciones que tenemos que tomar en función de la tecla pulsada. De esta forma, en el caso de recibir A, B o C se cambiará el tipo de operación a realizar, si se pulsa D se mostrará la operación realizada y resultado. Observar que pulsaciones de teclas numéricas (0 a 9) son ignoradas mostrando un mensaje de aviso. Si se pulsan F o E entonces se entrará en el modo de selección de operando numérico.

Además, se ha tenido en cuenta la funcionalidad de solamente contar una vez por pulsación de tecla para evitar múltiples lecturas de una misma entrada.



```

while (1)
{
// Lectura e Interpretación de datos de driver_teclado
START AGAIN:
    datos_keypad = neorv32_cpu_load_unsigned_word(WB_ADDR_DRIVER);
    if (datos_keypad == datos_keypad_anterior) // compruebo que se haya dejado de pulsar la misma tecla
    {
        goto START AGAIN;
    }
    if (datos_keypad == 0xffff) // easter-egg (pulsando 10)
        neorv32_uart0_print("Has descubierto un easter egg...\n");
    else
    {
        mask = 1;
        for (uint8_t tecla = 0; (tecla < 16); tecla++)
        {
            if (((~datos_keypad) & mask) == mask) // Detectada tecla pulsada
            {
                tecla_pulsada = (uint32_t)tecla;

                switch (tecla_pulsada)
                {
                    case 0xA: // Modo SUMA
                        neorv32_uart0_print("MODO SUMA\n");
                        neorv32_cpu_store_unsigned_word(BASE_ADDR + 8, 0);
                        break;
                    case 0xB: // Modo RESTA
                        neorv32_uart0_print("MODO RESTA\n");
                        neorv32_cpu_store_unsigned_word(BASE_ADDR + 8, 1);
                        break;
                    case 0xC: // Modo MULTIPLICACIÓN
                        neorv32_uart0_print("MODO MULTIPLICACION\n");
                        neorv32_cpu_store_unsigned_word(BASE_ADDR + 8, 2);
                        break;
                    case 0xF: // Selección Operando 1
                        neorv32_uart0_print("OPERANDO 1 SELECCIONADO\n");
                        datos_keypad_anterior = datos_keypad;
                        operando1 = cambia_operando(1);
                        neorv32_cpu_store_unsigned_word(BASE_ADDR, operando1);
                        neorv32_uart0_printf("Operando 1: %i\n", operando1);
                        goto START AGAIN;
                    case 0xE: // Selección Operando 2
                        neorv32_uart0_print("OPERANDO 2 SELECCIONADO\n");
                        datos_keypad_anterior = datos_keypad;
                        operando2 = cambia_operando(0);
                        neorv32_cpu_store_unsigned_word(BASE_ADDR + 4, operando2);
                        neorv32_uart0_printf("Operando 2: %i\n", operando2);
                        goto START AGAIN;
                    case 0xD: // Mostrar resultado
                        neorv32_uart0_print("RESULTADO:\n");
                        función = neorv32_cpu_load_unsigned_word(BASE_ADDR + 8);
                        resultado = neorv32_cpu_load_unsigned_word(BASE_ADDR + 12);
                        (...) //muestra por pantalla
                        break;
                    default:
                        neorv32_uart0_print("Primero selecciona operando a guardar.\n");
                        break;
                }
                mask = mask << 1;
            }
        }
        datos_keypad_anterior = datos_keypad;
    }
}

```

Para realizar esta tarea cambio de operandos, se ha creado una función auxiliar, que supone la segunda gran parte del código. La función recibe el modo en el que se encuentra (operando1 u operando2) y devuelve el valor numérico con signo que deberá almacenarse en el registro del periférico wb_regs correspondiente. Una vez seleccionado el cambio de operando, podemos usar las teclas numéricas 0 a 9 para introducir el número que queramos en el rango permitido, por cuestión de diseño en la dimensión de los datos con los que tratamos (rango de -512 a 511 = 2^{10}). Podemos introducir hasta números de 3 cifras en dicho rango. También podemos seleccionar el signo que tendrá el número pulsando la tecla A. Si el valor sale del rango se reinicia a 0 para volver a introducirlo. Para almacenar el valor del operando1 o 2 basta con pulsar de nuevo la tecla F o E respectivamente.

```

uint32_t cambia_operando(int modo)
{
    (...) //inicialización variables

    while (1)
    {
        START AGAIN:
        datos_keypad = neorv32_cpu_load_unsigned_word(WB_ADDR_DRIVER);
        if (datos_keypad == datos_keypad_anterior)
            goto START AGAIN;
        mask = 1;
        for (uint8_t tecla = 0; tecla < 16; tecla++)
        {
            if (((~datos_keypad) & mask) == mask)
            {
                if (tecla <= 9)
                {
                    flagKeep = 0;
                    if (((int)operando * 10 + (int)tecla) > 511)
                    {
                        neorv32_uart0_print("Número excede límites \n");
                        operando = 0;
                    }
                    else
                        operando = operando * 10 + (uint32_t)tecla;
                    neorv32_uart0_printf("Operando actual: %i\n", signo * operando);
                }
                else if (((tecla != 14) && (modo == 0)) || ((tecla != 15) && (modo == 1))) && (tecla != 10))
                {
                    neorv32_uart0_print("Tecla no válida \n");
                }
                else if (tecla == 10) // si se pulsa A se invierte el signo del operando
                {
                    if (signo == 1)
                    {
                        signo = -1;
                    }
                    else
                    {
                        signo = 1;
                    }
                    neorv32_uart0_print("Operando actual: ");
                    (...) //distinción de casos para mostrar el número en pantalla
                }
                else
                {
                    datos_keypad_anterior = datos_keypad;
                    if (flagKeep == 1)
                    {
                        if (modo == 1)
                            operando = operando1;
                        else
                            operando = operando2;
                    }
                    return operando * signo;
                }
                mask = mask << 1;
            }
            mask = 1;
            datos_keypad_anterior = datos_keypad;
        }
    }
}

```

Por último, se ha incluido un detalle para mostrar la capacidad de extensión de nuestro sistema. Podemos detectar combinaciones de teclas para realizar otro tipo de funcionalidades. En nuestro caso al pulsar las teclas 1 y 0 muestra un mensaje por cutecom. En lugar de eso se podría hacer cualquier otra funcionalidad extra en el sistema global, así como añadir las combinaciones de teclas deseadas siguiendo este ejemplo demostrativo. No obstante, existe una limitación debido al conexionado propio del teclado, por la cuál no se pueden detectar simultáneamente 2 teclas de la misma fila. Sin embargo, sí será posible entre teclas de la misma columna o filas distintas.

A continuación, se muestran pruebas realizadas del sistema enseñando así los resultados alcanzados y verificando el cumplimiento de las especificaciones marcadas.

[20:05:22:076] OPERANDO 1 SELECCIONADO $\text{c}_R \text{ } \text{l}_F$
[20:05:22:476] Operando actual: 3 $\text{c}_R \text{ } \text{l}_F$
[20:05:22:700] Operando actual: 35 $\text{c}_R \text{ } \text{l}_F$
[20:05:25:980] Operando actual: 356 $\text{c}_R \text{ } \text{l}_F$
[20:05:28:331] Tecla no valida $\text{c}_R \text{ } \text{l}_F$
[20:05:30:091] Operando 1: 356 $\text{c}_R \text{ } \text{l}_F$
[20:05:36:027] Primero selecciona operando a guardar. $\text{c}_R \text{ } \text{l}_F$
[20:05:36:395] Primero selecciona operando a guardar. $\text{c}_R \text{ } \text{l}_F$
[20:05:40:538] OPERANDO 2 SELECCIONADO $\text{c}_R \text{ } \text{l}_F$
[20:05:43:018] Operando actual: 7 $\text{c}_R \text{ } \text{l}_F$
[20:05:43:465] Operando actual: 75 $\text{c}_R \text{ } \text{l}_F$
[20:05:44:394] Número excede límites $\text{c}_R \text{ } \text{l}_F$
[20:05:44:409] Operando actual: 0 $\text{c}_R \text{ } \text{l}_F$
[20:05:47:305] Operando actual: 2 $\text{c}_R \text{ } \text{l}_F$
[20:05:47:625] Operando actual: 23 $\text{c}_R \text{ } \text{l}_F$
[20:05:48:745] Operando actual: -23 $\text{c}_R \text{ } \text{l}_F$
[20:05:50:025] Operando actual: 23 $\text{c}_R \text{ } \text{l}_F$
[20:05:51:401] Operando actual: -23 $\text{c}_R \text{ } \text{l}_F$
[20:05:53:257] Operando actual: -236 $\text{c}_R \text{ } \text{l}_F$
[20:05:54:904] Tecla no valida $\text{c}_R \text{ } \text{l}_F$
[20:05:56:472] Operando 2: -236 $\text{c}_R \text{ } \text{l}_F$
[20:05:58:072] MODO SUMA $\text{c}_R \text{ } \text{l}_F$
[20:05:59:144] RESULTADO: $\text{c}_R \text{ } \text{l}_F$
[20:05:59:144] $356 + (-236) = 120 \text{ c}_R \text{ l}_F$
[20:05:59:160] *-----* $\text{c}_R \text{ } \text{l}_F$
[20:06:02:999] MODO RESTA $\text{c}_R \text{ } \text{l}_F$
[20:06:03:639] RESULTADO: $\text{c}_R \text{ } \text{l}_F$
[20:06:03:639] $356 - (-236) = 592 \text{ c}_R \text{ l}_F$
[20:06:03:655] *-----* $\text{c}_R \text{ } \text{l}_F$
[20:06:04:887] MODO MULTIPLICACION $\text{c}_R \text{ } \text{l}_F$
[20:06:05:671] RESULTADO: $\text{c}_R \text{ } \text{l}_F$
[20:06:05:671] $356 \times (-236) = -84016 \text{ c}_R \text{ l}_F$
[20:06:05:687] *-----* $\text{c}_R \text{ } \text{l}_F$

[22:09:25:587] OPERANDO 1 SELECCIONADO $\text{c}_R \text{ } \text{l}_F$
[22:09:27:747] Operando actual: 1 $\text{c}_R \text{ } \text{l}_F$
[22:09:28:051] Operando actual: 15 $\text{c}_R \text{ } \text{l}_F$
[22:09:30:067] Operando 1: 15 $\text{c}_R \text{ } \text{l}_F$
[22:09:31:091] Primero selecciona operando a guardar.
[22:09:31:107] Has descubierto un easter egg... $\text{c}_R \text{ } \text{l}_F$
[22:09:37:299] OPERANDO 2 SELECCIONADO $\text{c}_R \text{ } \text{l}_F$
[22:09:40:003] Operando actual: 3 $\text{c}_R \text{ } \text{l}_F$
[22:09:40:419] Operando actual: 31 $\text{c}_R \text{ } \text{l}_F$
[22:09:41:219] Operando actual: -31 $\text{c}_R \text{ } \text{l}_F$
[22:09:42:403] Operando 2: -31 $\text{c}_R \text{ } \text{l}_F$
[22:09:44:211] MODO RESTA $\text{c}_R \text{ } \text{l}_F$
[22:09:45:139] RESULTADO: $\text{c}_R \text{ } \text{l}_F$
[22:09:45:155] $15 - (-31) = 46 \text{ c}_R \text{ l}_F$
[22:09:45:155] *-----* $\text{c}_R \text{ } \text{l}_F$
[22:09:48:355] OPERANDO 1 SELECCIONADO $\text{c}_R \text{ } \text{l}_F$
[22:09:48:851] Operando actual: 5 $\text{c}_R \text{ } \text{l}_F$
[22:09:49:347] Operando actual: 52 $\text{c}_R \text{ } \text{l}_F$
[22:09:50:275] Operando 1: 52 $\text{c}_R \text{ } \text{l}_F$
[22:09:51:507] RESULTADO: $\text{c}_R \text{ } \text{l}_F$
[22:09:51:507] $52 - (-31) = 83 \text{ c}_R \text{ l}_F$
[22:09:51:523] *-----* $\text{c}_R \text{ } \text{l}_F$
[22:09:54:403] OPERANDO 2 SELECCIONADO $\text{c}_R \text{ } \text{l}_F$
[22:09:56:627] Operando actual: 9 $\text{c}_R \text{ } \text{l}_F$
[22:09:57:139] Operando actual: -9 $\text{c}_R \text{ } \text{l}_F$
[22:09:58:627] Operando 2: -9 $\text{c}_R \text{ } \text{l}_F$
[22:10:00:003] RESULTADO: $\text{c}_R \text{ } \text{l}_F$
[22:10:00:003] $52 - (-9) = 61 \text{ c}_R \text{ l}_F$
[22:10:00:019] *-----* $\text{c}_R \text{ } \text{l}_F$
[22:10:02:371] OPERANDO 2 SELECCIONADO $\text{c}_R \text{ } \text{l}_F$
[22:10:02:867] Operando actual: 9 $\text{c}_R \text{ } \text{l}_F$
[22:10:03:715] Operando 2: 9 $\text{c}_R \text{ } \text{l}_F$
[22:10:04:595] RESULTADO: $\text{c}_R \text{ } \text{l}_F$
[22:10:04:595] $52 - 9 = 43 \text{ c}_R \text{ l}_F$
[22:10:04:611] *-----* $\text{c}_R \text{ } \text{l}_F$