Miguel Ángel Lara Guarino

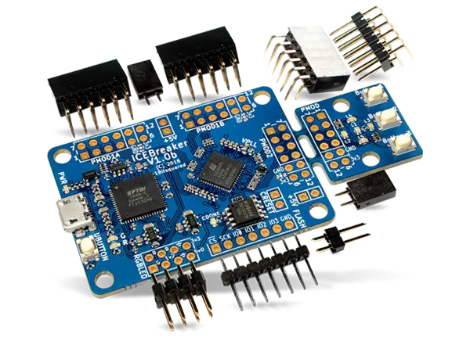
Alfonso Manchado Pérez

**Icono

Descripción generada automáticamente**

**SEPA 2023:**

**Memoria de Prácticas y Proyecto**



SoftCore

En esta memoria, nos adentramos en el emocionante mundo de la FPGA Lattice IceBreaker, una placa de desarrollo poderosa y compacta, y exploramos en detalle su integración con el procesador Neorv32 como SoftCore. Este procesador de 32 bits, diseñado específicamente para su implementación en FPGA, proporciona una sólida base de cómputo y control, ampliando las capacidades de la IceBreaker y ofreciendo una plataforma adaptable para numerosas aplicaciones.

A lo largo de este informe, comentaremos las diferentes prácticas realizadas durante la asignatura y acabando con la explicación de nuestro proyecto final, donde la combinación de la FPGA IceBreaker y el procesador Nerov32 ha demostrado ser excepcionalmente efectiva.

Pero, antes de nada, ¿por qué el uso de un procesador dentro de una FPGA? La inclusión de un procesador en una FPGA (Field-Programmable Gate Array) ofrece una versatilidad única al permitir la personalización del hardware para aplicaciones específicas, combinando la lógica personalizada con el procesamiento de propósito general. Es decir, tenemos por un lado el poder de paralelizar múltiples tareas con la FPGA y por el otro, el desarrollo secuencial por parte del procesador que nos permite tener más control sobre el sistema.

**Índice**

[1. Práctica 1: Toma de contacto y uso de los leds 3](#_Toc153830226)

[**Hardware** 3](#_Toc153830227)

[**Software** 4](#_Toc153830228)

[2. Práctica 2: Uso del Teclado numérico mediante el GPIO 5](#_Toc153830229)

[**Hardware** 5](#_Toc153830230)

[**Software** 7](#_Toc153830231)

[3. Práctica 3: Uso de un periférico Wishbone 8](#_Toc153830232)

[**Hardware** 8](#_Toc153830233)

[**Software** 10](#_Toc153830234)

[4. Proyecto Final 10](#_Toc153830235)

[**Hardware** 10](#_Toc153830236)

[**Software** 12](#_Toc153830237)

[5. Anotaciones adicionales 12](#_Toc153830238)

# Práctica 1: Toma de contacto y uso de los leds

Para esta primera práctica, entraremos en contacto con el manejo de las herramientas que se nos proporcionan para realizar la síntesis, implementación y generación del bitstream por parte de los códigos VHDL y la compilación para la parte del procesador Neorv32.

Los programas que nos permitirán efectuar estos pasos son:

* Síntesis 🡪 Yosys + Ghdlsynth
* Place & Route 🡪 Nextpnr
* Bitstream 🡪 Icestorm
* Build 🡪 GNU Make

## **Hardware**

Para la parte del hardware, se hará uso de los pines de comunicación GPIO para crear una conexión directa entre los Leds y el soft processor Neorv32. Además, se agregará una señal de reset controlada por el soft processor y, finalmente, se comunicará también mediante el GPIO, el valor del botón pulsado.



Gpio\_o

Gpio\_i

Leds (4 downto 0)



Reset signal

**ICE40UP5k**

**FPGA**

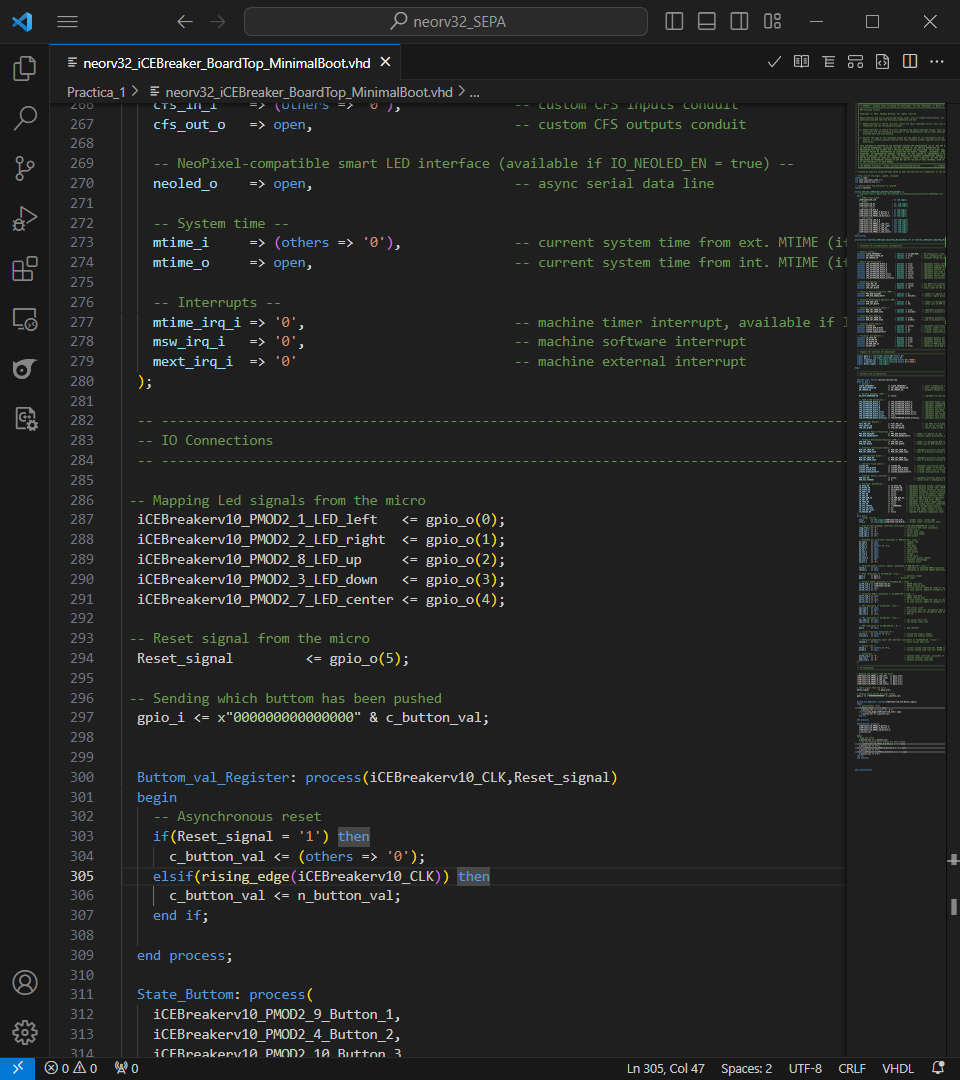
Button (2 downto 0)

1. Texto

   Descripción generada automáticamenteAñadimos las señales necesarias:
2. Asignamos a la señal ‘c\_button\_val’ el valor del botón que haya sido pulsado:

Texto

Descripción generada automáticamente

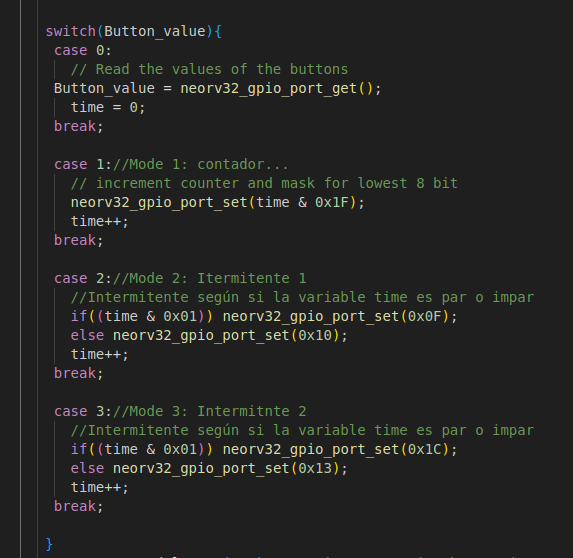
1. Asignamos valor a las salidas:

## **Software**

En la parte software habrá un único cometido, recibir los datos enviados a través de la GPIO y traducirlo en una secuencia de leds.

Para esto hay 3 secuencias de leds a elegir con los 3 botones de nuestra FPGA mediante una máquina de estados, y tras esto, aprovechamos la señal de reset diseñada en hardware para volver al inicio pasado unos segundos.

1. Máquina de estados en función de la pulsación de los botones



1. Timer para resetear la GPIO

Texto

Descripción generada automáticamente

# Práctica 2: Uso del Teclado numérico mediante el GPIO

Una vez adaptados a este nuevo entorno de trabajo, en esta segunda práctica tocaremos el uso de periféricos externos instanciados en nuestro hardware, en concreto, usaremos el teclado de 16 valores PMOD KYPD. Nuestro objetivo personal para esta práctica es conseguir el funcionamiento correcto de este, y aprovecharlo para realizar una calculadora a nivel software.

## **Hardware**

Para la parte del hardware, se hará uso de los pines de comunicación GPIO para crear una conexión directa entre el teclado y el soft processor Neorv32. Además, se instanciará una entidad llamada “peripheral\_teclado.vhd” la cual realizará el proceso de lectura del teclado.

s\_leds [4:0]

Gpio\_o

Gpio\_i



**ICE40UP5k**

**FPGA**

c\_button\_val [2:0]

s\_reset

Key\_value [15:0]



Peripheral\_teclado.vhd

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza bajaPara realizar la lectura del teclado en hardware, se ha diseñado la siguiente entidad en VHDL:

Texto

Descripción generada automáticamenteCaptura de pantalla de computadora

Descripción generada automáticamentePara poder realizar la lectura sobre el teclado, utilizaremos un contador de 2 bits para realizar la secuencia: ‘1110’, ‘1101’, ‘1011’ y ‘0111’ según el contador sea ‘00’, ‘01’, ‘10’ y ‘11’, respectivamente. Por cada secuencia se realizará una lectura de las señales col\_1\_o, col\_2\_o, col\_3\_o y col\_4\_o para comprobar si existe algún cero, en cuyo caso, invertiremos la señal y la guardaremos en el registro c\_key como codificación One Hot, el cual volcará sus datos sobre el registro c\_key\_value cada 4 ciclos (Tras un barrido de lectura sobre el teclado) y limpia el registro c\_key. Finalmente, el valor leído del teclado es enviado directamente al top, en el cual se envía al Gpio\_i para que el soft processor pueda leer la tecla pulsada.

|  |  |  |  |
| --- | --- | --- | --- |
| Tecla | Codificación | Tecla | Codificación |
| Ninguna | 0x0000 | Ninguna | 0x0000 |
| “1” | 0x0008 | “2” | 0x0800 |
| “4” | 0x0004 | “5” | 0x0400 |
| “7” | 0x0002 | “8” | 0x0200 |
| “0” | 0x0001 | “F” | 0x0100 |
| “A” | 0x0080 | “3” | 0x8000 |
| “B” | 0x0040 | “6” | 0x4000 |
| “C” | 0x0020 | “9” | 0x2000 |
| “D” | 0x0010 | “E” | 0x1000 |

## **Software**

A nivel software se diseña una función para decodificar lo recibido a través de “Key\_value”, y con una máquina de estados, en función de lo pulsado, realizamos las cuentas de la calculadora.

Para esto hemos tratado cada letra del teclado como una operación:

1. 🡪 Ans(guardamos el resultado anterior)
2. 🡪Suma
3. 🡪Resta
4. 🡪Multiplicación
5. 🡪Ac(Reseteo)
6. 🡪Resultado
7. Actualización del valor pulsado.

Texto

Descripción generada automáticamente

1. Máquina de estados para la pulsación de las letras(operaciones)

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

1. Función auxiliar para evitar repeticiones en las pulsaciones.

Texto

Descripción generada automáticamente

# Práctica 3: Uso de un periférico Wishbone

Esta práctica se basa en el uso de los periféricos mediante el bus Wishbone, un bus lógico que usaremos para conectar hardware con software y, además, guardar datos en direcciones conocidas a las que podamos acceder con facilidad. Nuestro objetivo para realizar con este bus es crear un candado virtual, almacenando las claves en los registros del bus.

## **Hardware**

Para la parte del hardware, se hará uso de un periférico WishBone. De este modo, creamos una nueva entidad llamada “wb\_peripheral\_teclado.vhd” el permitirá establecer una comunicación segura entre el soft processor y el periférico sobre los registros 0, 1, 2, 3 y 4 instanciados en la dirección 0x90000000. Por tanto, nuestro periférico estará situado entre 0x90000000 y 0x9000001F.



**wb\_tag\_m2s [2:0]**

**wb\_we\_m2s**



s\_leds [4:0]

Gpio\_o

Gpio\_i

s\_reset

c\_button\_val [2:0]

**ICE40UP5k**

**FPGA**

**Wb\_Peripheral\_teclado.vhd**

**wb\_adr\_m2s [31:0]**

**wb\_dat\_m2s [31:0]**

**wb\_sel\_m2s [3:0]**

**wb\_stb\_m2s**

**wb\_cyc\_m2s**

**wb\_lock\_m2s**

**wb\_err\_s2m**

**wb\_dat\_s2m [31:0]**

**wb\_ack\_s2m**

Para poder realizar esta comunicación, primero realizamos una comprobación de seguridad de sobre las constantes definidas en el generic map:

Texto

Descripción generada automáticamente

Luego, realizaremos una comprobación de que la dirección a la que están acudiendo es la de nuestro periférico:

Texto

Descripción generada automáticamente con confianza baja

Ahora, en un process comprobaremos (en caso de que acces\_req = ‘1’) si están realizando una escritura o una lectura, y a cuál de los registros se está realizando:

Texto

Descripción generada automáticamente

|  |  |  |
| --- | --- | --- |
| Registro 0 | Tecla Pulsada | Tecla pulsada en el periférico Teclado |
| Registro 1 | Contraseña escrita | Almacena la contraseña escrita por el usuario |
| Registro 2 | Comprobación | Se utiliza en el Proyecto (se explica más adelante) |
| Registro 3 | Contraseña real | Almacena la contraseña correcta |
| Registro 4 | Resultado | Se utiliza en el Proyecto (se explica más adelante) |

## **Software**

En la parte software, recurriremos a los registros que tenemos para ir almacenando la clave introducida y después compararla con la clave almacenada. Para esto recibimos con en la práctica anterior los caracteres pulsados y según sea el valor, escribimos la clave, cambiamos esta, o introducimos lo que llevamos. En caso de acertar o fallar se encenderá una serie de leds avisando, con verde o rojo respectivamente, acompañado de un aviso por la uart.

# Proyecto Final

## **Hardware**

Para la parte del hardware, se hará uso del mismo periférico wishbone del teclado explicado en la práctica 3 (con algunas mejoras explicadas más adelante) y un nuevo periférico wishbone para el uso de un display 7 segmentos de dos dígitos.



**wb\_tag\_m2s [2:0]**

**wb\_we\_m2s**



s\_leds [4:0]

Gpio\_o

Gpio\_i

s\_reset

c\_button\_val [2:0]

**ICE40UP5k**

**FPGA**

**Wb\_Peripheral\_teclado.vhd**

**wb\_adr\_m2s [31:0]**

**wb\_dat\_m2s [31:0]**

**wb\_sel\_m2s [3:0]**

**wb\_stb\_m2s**

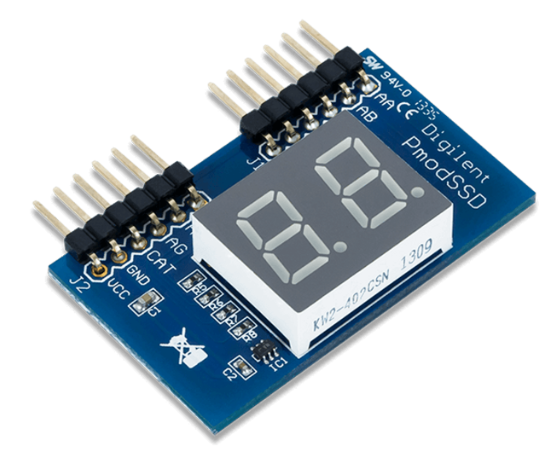
**wb\_cyc\_m2s**

**wb\_lock\_m2s**

**wb\_err\_s2m**

**wb\_dat\_s2m [31:0]**

**wb\_ack\_s2m**



|  |  |
| --- | --- |
| ‘xxx1’ | Protocolo A correcto |
| ‘xx1x’ | Protocolo B correcto |
| ‘x1xx’ | Protocolo C correcto |
| ‘1xxx’ | Protocolo D correcto |

Las mejoras incluidas en el periférico wishbone del Teclado es la comprobación dinámica de la contraseña insertada por el usuario en los 4 protocolos. Para esto, añadiremos un process en el que comprobaremos si nos indican mediante el registro 2 qué protocolo debemos comprobar (A, B, C o D) y el resultado de la comparación del registro 1 (Contraseña insertada por el usuario) con el registro 3 (Contraseña correcta) se guardará en el registro 4 según el protocolo que se esté comprobando.

Siendo x cualquier valor.

Texto

Descripción generada automáticamente

Esta metodología permite una comprobación de la contraseña de los 4 protocolos en paralelo.

Para el periférico wishbone del display 7 segmentos dividiremos el código en dos partes, el primero es el encargado de establecer la comunicación con el Neorv32 y los registros de nuestro periférico y el segundo es el encargado de leer los registros 0, 1 y 2, y actuar sobre el display.

Texto

Descripción generada automáticamenteLa primera parte será igual a la detallada en la practica 3 sobre el periférico wishbone de teclado a excepción de que ahora este periférico lo tendremos ubicado en la posición 0x900000020 y de tamaño 16 bits.

En la segunda parte dispondremos de dos multiplexores encargados de decodificar los diferentes dígitos del display y un process encargado de generar un contador de 183 Hz y la asignación de los pines del display en función del registro de control 2.

Dígitos [6:0]

Registro 0

Registro 1

Digit Select

Centenas

Decenas

**S\_decod\_num**

**Interfaz de usuario gráfica, Texto

Descripción generada automáticamente**

## **Software**

En la parte software se ha diseñado la máquina de estados que recibe los caracteres y se los envía tanto al display, como a los registros. Para esto se comunica con el hardware mediante registros para saber cómo representarlos, ayudado de una función auxiliar que aprovecha los valores establecidos en nuestro top.

Además, se ocupa de determinar cuál es la clave de la caja y asignar, según el protocolo elegido, las diferentes claves a sus bits correspondientes del registro 3.

Texto

Descripción generada automáticamenteDe forma externa al main, creamos dos funciones, una primera “Lee\_teclado”, explicada en las prácticas anteriores, para poder obtener un valor de una pulsación.

Además, tenemos la función “Represent\_Display”, que tendrá como entradas el valor a representar en las unidades, el de las decenas y enable. Estos valores no serán literales, si no que mandaremos el código de dichos valores(coincide con números, pero no con letras).

En esta función se trabaja con dos registros dedicados únicamente al display, en uno guardaremos el valor a representar en las unidades y en el otro el de las decenas, de esta forma, en el código hardware se hará la asignación de bits a activar.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza mediaIniciamos el bucle, y para futuras necesidades, en cada ciclo guardamos el valor de todos nuestros registros dedicados a teclado y contraseñas.

El código en c se basará en una máquina de estados, el estado inicial comprobará en un inicio si hemos acertado la contraseña, en este caso abrirá la caja durante 5z, y tras esto hará un reset.

Texto

Descripción generada automáticamenteEn caso contrario leeremos el valor pulsado en el teclado y dividiremos entre números y letras.

Texto

Descripción generada automáticamenteSi pulsamos un número entramos al estado 0, en el que iremos escribiendo por pantalla el valor que llevamos pulsado, y además lo mostramos en el display.

Junto a esto se muestra el valor de nuestros registros para poder ir comprobando a nivel código que todo funciona correctamente, y cómo función.

Si se hubiese pulsado una tecla entre A y D, pasamos a los estados de protocolo, tenemos uno para cada una de estas letras.

Esta parte se encarga de guardar en el registro 1 lo que hemos leído por teclado junto al valor anterior, y enviamos mediante el registro 2 una señal al hardware avisando de que se ha pulsado dicha letra.

Texto

Descripción generada automáticamenteCómo vemos en la figura, en este caso se envía al registro 2 un 1, activando el byte menos significativo del registro, que se refiere al protocolo A, en este orden, el segundo byte corresponderá con el protocolo B, y así con los C y D.

Tras esto, pasamos a los estados de comprobación del protocolo.

Texto

Descripción generada automáticamenteSe lee el registro 4, en donde como se explica anteriormente, se leerá un 1 su bit primer bit en caso de haber acertado la contraseña asignada a este protocolo, es decir, en caso de que el valor que hemos introducido en el registro 1 coincida con los el bit menos significativos del registro 4. Si acertamos vamos al estado inicial, de lo contrario, pasamos al estado de fallo.

En el estado inicial, como vimos antes, comprobamos si el registro 4 tiene sus 4 primeros bits a 1, y en ese caso mostramos por el display la apertura de la caja con un OP, y mantenemos encendidos los leds de cada acierto, hasta tener 4 verdes en caso de abrir la caja.

Si hubiésemos fallado en cualquiera de las claves, pasamos al estado de fallo, que lo único que hará será mandar un reset síncrono, poner las variables a cero y volver a establecer la contraseña inicial.

Texto

Descripción generada automáticamentePara saber que esto ha pasado, encendemos un led rojo durante 3s y mostramos un mensaje por pantalla, y CL(close) por el display.

Texto

Descripción generada automáticamentePor último, si quisiéramos resetear lo escrito sin tener que fallar, tenemos la opción de la tecla E, borrará las claves introducidas y reseteará todos los registros.

# Anotaciones adicionales

A lo largo de la memoria, se ha utilizado el concepto de la codificación One-Hot en el apartado hardware. Esto es porque simplifica la representación de estados en circuitos digitales al usar un solo bit alto para cada estado, facilitando la implementación de lógica secuencial y máquinas de estado. Esto reduce la propagación de señales, mejora la velocidad de operación y la detección de errores.

Por último, para poder tener acceso de forma más cómoda al código correspondiente a cada práctica, se facilita un enlace público al repositorio de GitHub en el que hemos estado trabajando de forma conjunta.

https://github.com/Miguellarag02/neorv32\_SEPA.git