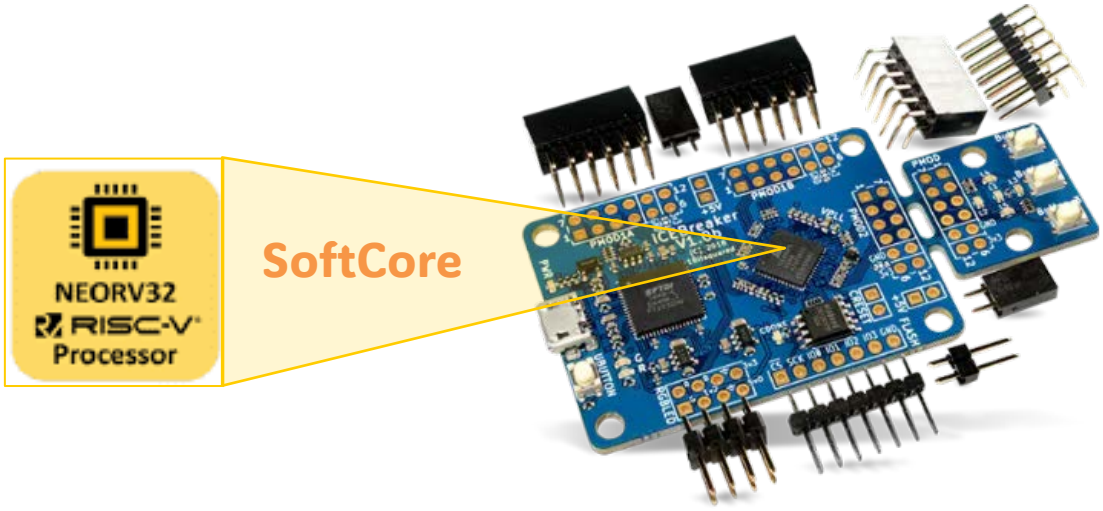
Miguel Ángel Lara Guarino
Alfonso Manchado Pérez

SEPA 2023:

Memoria de Prácticas y Proyecto



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

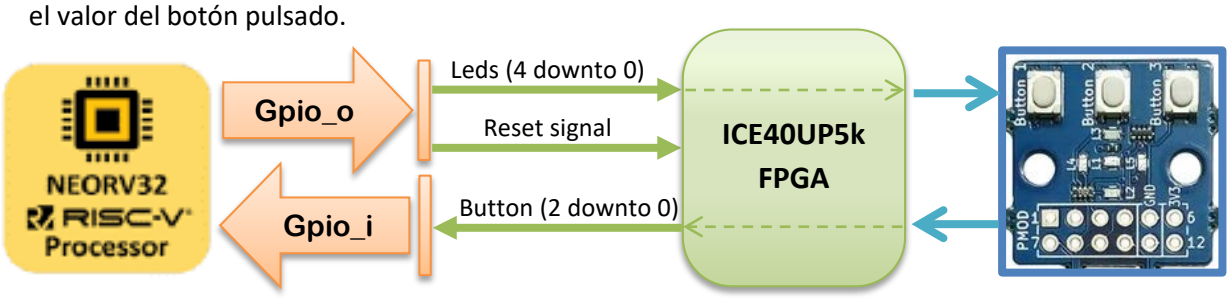
Para esta primera práctica, entraremos en contacto con el manejo de las herramientas que se nos proporcionan para realizar la síntesis, implementación y generación del bitstream por parte de los códigos VHDL y la compilación para la parte del procesador Neorv32.

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.



1. Añadimos las señales necesarias:

```
-- ------------------------------------------------------------
-- Signals for internal IO connections
-- ------------------------------------------------------------
signal gpio_o : std_ulogic_vector(63 downto 0);
signal gpio_i : std_logic_vector(63 downto 0);
signal n_button_val : std_logic_vector(3 downto 0):="0000";
signal c_button_val : std_logic_vector(3 downto 0):="0000";
signal Reset_signal : std_logic;
```

2. Asignamos a la señal ‘c_button_val’ el valor del botón que haya sido pulsado:

```
buttom_val_Register: process(iCEBreakerv10_CLK,Reset_signal)
begin
  -- Asynchronous reset
  if(Reset_signal = '1') then
    c_button_val <= (others => '0');
  elsif(rising_edge(iCEBreakerv10_CLK)) then
    c_button_val <= n_button_val;
  end if;

end process;

State_Buttom: process(
  iCEBreakerv10_PMOD2_9_Button_1,
  iCEBreakerv10_PMOD2_4_Button_2,
  iCEBreakerv10_PMOD2_10_Button_3,
  c_button_val
  )
begin
-- Keep the state
  n_button_val <= c_button_val;
  -- We are sending which buttom has been pushed
  if (iCEBreakerv10_PMOD2_9_Button_1 = '1') then
  n_button_val <= x"1";
elsif (iCEBreakerv10_PMOD2_4_Button_2 = '1') then
  n_button_val <= x"2";
elsif (iCEBreakerv10_PMOD2_10_Button_3 = '1') then
  n_button_val <= x"3";
end if;
end process;
```

3. Asignamos valor a las salidas:

```
-- Mapping Led signals from the micro
 iCEBreakerv10_PMOD2_1_LED_left   <= gpio_o(0);
 iCEBreakerv10_PMOD2_2_LED_right  <= gpio_o(1);
 iCEBreakerv10_PMOD2_8_LED_up     <= gpio_o(2);
 iCEBreakerv10_PMOD2_3_LED_down   <= gpio_o(3);
 iCEBreakerv10_PMOD2_7_LED_center <= gpio_o(4);

-- Reset signal from the micro
 Reset_signal         <= gpio_o(5);

-- Sending which buttom has been pushed
 gpio_i <= x"00000000000000" & c_button_val;
```

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

```
switch(Button_value){
 case 0:
   // Read the values of the buttons
 Button_value = neorv32_gpio_port_get();
   time = 0;
 break;

 case 1://Mode 1: contador...
   // increment counter and mask for lowest 8 bit
   neorv32_gpio_port_set(time & 0x1F);
   time++;
 break;

 case 2://Mode 2: Itermitente 1
   //Intermitente según si la variable time es par o impar
   if((time & 0x01)) neorv32_gpio_port_set(0x0F);
   else neorv32_gpio_port_set(0x10);
   time++;
 break;

 case 3://Mode 3: Intermitnte 2
   //Intermitente según si la variable time es par o impar
   if((time & 0x01)) neorv32_gpio_port_set(0x1C);
   else neorv32_gpio_port_set(0x13);
   time++;
 break;
```

2. Timer para resetear la GPIO

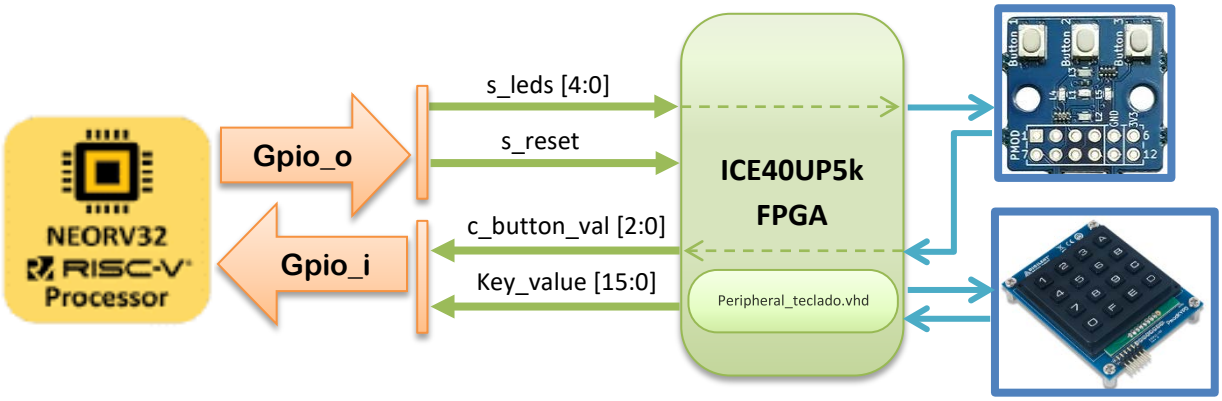
```
    neorv32_cpu_delay_ms(200); // wait 500ms using busy wait

    if (time > 30){ //After 6 seconds in a mode...
      for(i=0; i<3; i++)
      {
        //leds sequence to announce the automatic reset
        neorv32_cpu_delay_ms(300);
        neorv32_gpio_port_set(0x1F);
        neorv32_cpu_delay_ms(300);
        neorv32_gpio_port_set(0x00);
      }

      neorv32_gpio_port_set(0x20); // Asynchronous Reset
      neorv32_uart0_print("\nReset activado");
      neorv32_cpu_delay_ms(10); // wait 500ms using busy wait
      neorv32_gpio_port_set(0); // clear gpio output
      time = 0;
      Button_value = 0;
```

2. 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.



Para realizar la lectura del teclado en hardware, se ha diseñado la siguiente entidad en VHDL:

```
entity peripheral_teclado is
  port (
    -- 12MHz Clock input
    clk_i               : in std_logic;
    reset_i             : in std_logic;

    -- Rows
    en_i                : in std_logic;
    Row_1_i             : in std_logic;
    Row_2_i             : in std_logic;
    Row_3_i             : in std_logic;
    Row_4_i             : in std_logic;

    -- Cols
    Col_1_o             : out std_logic;
    Col_2_o             : out std_logic;
    Col_3_o             : out std_logic;
    Col_4_o             : out std_logic;

    -- Key codificated in One Hot
    Key_o     : out std_logic_vector(15 downto 0)

    );
end entity;
```

Para 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.

```
-- Read key processs
peripheral_teclado_decode: process(
    c_counter,
    s_row,
    c_col,
    c_key,
    c_key_value
    )
begin
    n_key       <= c_key;
    n_col       <= (others => '0');
    n_counter   <= (others => '0');
    n_key_value <= c_key_value;

    -- Sampling the value each four cycles.
    if(c_counter = "00" and c_key /= "00000000" ) then
        n_key_value <= c_key;
        n_key <= (others => '0'); -- Reset de value
    end if;

    if (en_i = '1') then
        n_counter   <= c_counter + 1;

        case (c_counter) is
            when "00" =>
                n_col   <=  "1110";
                if (s_row /= "1111") then
                    n_key <= x"000" & not(s_row);
                end if;

            when "01" =>
                n_col   <=  "1101";
                if (s_row /= "1111") then
                    n_key <= x"00" & not(s_row) & x"0";
                end if;

            when "10" =>
                n_col   <=  "1011";
                if (s_row /= "1111") then
                    n_key <= x"0" & not(s_row) & x"00";
                end if;

            when others =>
                n_col   <=  "0111";
                if (s_row /= "1111") then
                    n_key <= not(s_row) & x"000";
                end if;
        end case;
    end if;

end process;
```

```
-- Concurrents Outputs
Col_1_o <= c_col(0);
Col_2_o <= c_col(1);
Col_3_o <= c_col(2);
Col_4_o <= c_col(3);

s_row   <= Row_4_i &
           Row_3_i &
           Row_2_i &
           Row_1_i;

Key_o   <= c_key_value;

peripheral_teclado_sinc: process(clk_i, reset_i)
begin
    if (reset_i = '1') then
        c_counter   <=  (others => '0');
        c_col       <= (others => '0');
        c_key       <= (others => '0');
        c_key_value <= (others => '0');

    elsif ( rising_edge(clk_i)) then
        c_counter   <= n_counter;
        c_col       <= n_col;
        c_key       <= n_key;
        c_key_value <= n_key_value;

    end if;
end process;
```

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:

A. → Ans(guardamos el resultado anterior)
B. →Suma
C. →Resta
D. →Multiplicación
E. →Ac(Reseteo)
F. →Resultado

1. Actualización del valor pulsado.

```
while (1) {
q_caracter_value = maquina_boton1();
Key_value = Lee_teclado();
if(q_caracter_value == 1){
    if(Key_value<=9)
    {
      int decimal = 0;
      decimal = Key_value;
      //Value update on every touchS
      total_value = total_value*10 +decimal;

      neorv32_uart0_printf("Has pulsado: %u\n",decimal);
      neorv32_uart0_print("Total value: ");
      neorv32_uart0_printf("%u\n",total_value);

    }
```

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

```
else if(Key_value>9 && Key_value<71)
{
  //Switch about different operations
  switch(Key_value){
    case 65: //ANS
      total_value = result;
      neorv32_uart0_printf("Guardado el resultado anterior: \n",result);
    break;

    case 66://ADD
      pADD=1;
      operando1 = total_value;
      total_value = 0;
      pSUBS=0;
      pPRO=0;
      neorv32_uart0_print("+\n");
    break;

    case 67://SUBSTRACTION
      if(total_value==0)
      {
        sign=-1;
      }
      else{
        pSUBS=1;
        operando1 = total_value;
        total_value = 0;
      }
      pADD=0;
      pPRO=0;
      neorv32_uart0_print("-\n");
    break;
```

```
    case 68://PRODUCT
      pPRO=1;
      operando1 = total_value;
      total_value = 0;
      pSUBS=0;
      pADD=0;
      neorv32_uart0_print("x\n");
    break;

    case 69://AC
      total_value=0;
      pSUBS=0;
      pADD=0;
      pPRO=0;
      sign=1;

      neorv32_uart0_print("Variables reseteadas\n");
    break;

    case 70://RESULT
      if(pADD==1){result=sign*operando1+total_value;}
      else if(pSUBS==1){result=sign*operando1-total_value;}
      else if(pPRO==1){result=sign*operando1*total_value;}
      total_value=0;
      pSUBS=0;
      pADD=0;
      pPRO=0;
      sign=1;
      neorv32_uart0_print("-----------------------\n");
      neorv32_uart0_printf("El resultado es: %u\n",result);
    break;
}
```

3. Función auxiliar para evitar repeticiones en las pulsaciones.

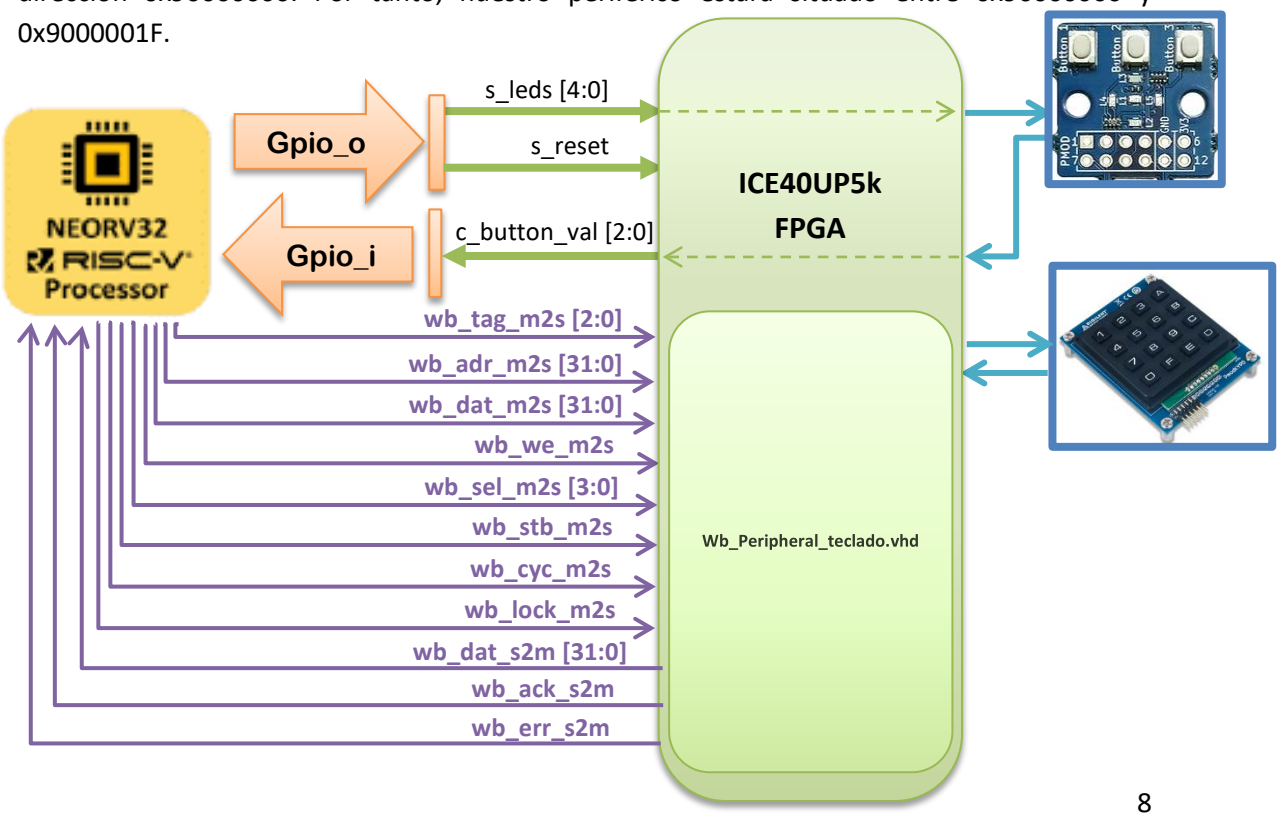
```
uint8_t maquina_boton1(void)
{
  uint8_t valor = Lee_teclado();
    switch(estadoB1)
    {
    case 0:
        if(valor != 0xFF)
        {
          estadoB1=1;
          }
        break;
    case 1:
        {estadoB1=2;}
        break;
    case 2:
        if(valor == 0xFF) {estadoB1=0;}
        break;
        }
    if(estadoB1==1) {return 1;}
    return 0;
```

3. 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.



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

```
-- Sanity Checks --------------------------------------------------------------
-- -----------------------------------------------------------------------------
assert not (WB_ADDR_SIZE < 4) report "wb_regs config ERROR: Address space <WB_ADDR_SIZE> has
assert not (is_power_of_two_f(WB_ADDR_SIZE) = false) report "wb_regs config ERROR: Address sp
assert not ((WB_ADDR_BASE and addr_mask_c) /= all_zero_c) report "wb_regs config ERROR: Modul
```

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

```
-- Device Access? -------------------------------------------------------------
-- -----------------------------------------------------------------------------
access_req <= '1' when ((wb_adr_i and (not addr_mask_c)) = (WB_ADDR_BASE and (not addr_mask_c))) else '0';
```

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:

```
s the peripheral selected?
f (wb_cyc_i = '1') and (wb_stb_i = '1') and (access_req = '1') then

    -- Write access, only full-word accesses
    if (wb_we_i = '1' and wb_sel_i = "1111") then
        case to_integer(unsigned(wb_adr_i(index_size_f(WB_ADDR_SIZE)-1 downto 2))) is
            when 0 =>
                n_reg0 <= wb_dat_i;
            when 1 =>
                n_reg1 <= wb_dat_i;
            when 2 =>
                n_reg2 <= wb_dat_i;
            when 3 =>
                n_reg3 <= wb_dat_i;
            when 4 =>
                n_reg4 <= wb_dat_i;
            when others =>
                null;
        end case;
        wb_ack_o <= '1';
    else
    -- Read access
        case to_integer(unsigned(wb_adr_i(index_size_f(WB_ADDR_SIZE)-1 downto 2))) is
            when 0 =>
                wb_dat_o <= c_reg0;
            when 1 =>
                wb_dat_o <= c_reg1;
            when 2 =>
                wb_dat_o <= c_reg2;
            when 3 =>
                wb_dat_o <= c_reg3;
            when 4 =>
                wb_dat_o <= c_reg4;
            when others =>
                null;
        end case;
        wb_ack_o <= '1';
    end if;

d if;
```

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.

```
while(1){
  int registro0 = neorv32_cpu_load_unsigned_word (WB_BASE_ADDRESS + WB_REG0_OFFSET);
  int registro1 = neorv32_cpu_load_unsigned_word (WB_BASE_ADDRESS + WB_REG1_OFFSET);
  int registro2 = neorv32_cpu_load_unsigned_word (WB_BASE_ADDRESS + WB_REG2_OFFSET);
  int registro3 = neorv32_cpu_load_unsigned_word (WB_BASE_ADDRESS + WB_REG3_OFFSET);



  Key_value = Lee_teclado();
  if(Key_value != 0xFF){

    if(Key_value != q_key_value){
      if(Key_value < 10){
        int decimal = 0;
        decimal = Key_value;
        total_value = total_value*10 + decimal;
        neorv32_uart0_printf("Clave introducida: %u\n",total_value);
        neorv32_cpu_store_unsigned_word (WB_BASE_ADDRESS + WB_REG1_OFFSET, total_value);
      }
```

Comenzamos el bucle y, en cada ciclo, leemos nuestros 4 registros para ir mostrando que el funcionamiento es correcto.

Tras esto, leemos nuestra tecla como en la práctica anterior, y vamos acumulando el valor en la clave que hemos escrito.

Lo que vamos pulsando se guarda además, en cada pulsación, en el registro 1.

Una vez hemos escrito nuestra contraseña y queramos terminar, hay dos opciones, la primera es la tecla ‘E’, que enviará un 1 al registro 2 avisando de que queremos hacer una comprobación, y llamamos a “compara_valores”.

```
else if(Key_value > 64 && Key_value < 71){
    if(Key_value == 70)
    {
     neorv32_uart0_print("Cambio de clave realizado\n");
     neorv32_cpu_store_unsigned_word (WB_BASE_ADDRESS + WB_REG3_OFFSET, total_value);
     total_value = 0;

    }
    else if(Key_value == 69)
    {
     neorv32_uart0_print("Comprobacion de la clave\n");
     neorv32_cpu_store_unsigned_word (WB_BASE_ADDRESS + WB_REG2_OFFSET, 0x00000001);
     total_value = 0;
     compara_valores();
```

En caso de pulsar ‘F’, significa que queremos cambiar la contraseña, por lo que en este caso se guardará en el registro 3, donde tenemos la contraseña, lo último que hemos tecleado.

```
_t compara_valores(void){
uint32_t password = 0;
uint32_t introducido = 0;
uint32_t flag = 0;

//Read register 0
introducido = neorv32_cpu_load_unsigned_word (WB_BASE_ADDRESS + WB_REG1_OFFSET);
password = neorv32_cpu_load_unsigned_word (WB_BASE_ADDRESS + WB_REG3_OFFSET);
flag = neorv32_cpu_load_unsigned_word (WB_BASE_ADDRESS + WB_REG2_OFFSET);

// If the user push a key:
if (flag == 0x00000001){
  // Read the key value:
  if(introducido == password)
  {
    neorv32_uart0_print("\nCLave correcta\n");
    // Reset the register 1
    neorv32_cpu_store_unsigned_word (WB_BASE_ADDRESS + WB_REG2_OFFSET, 0x00000000);
    neorv32_cpu_store_unsigned_word (WB_BASE_ADDRESS + WB_REG1_OFFSET, 0x00000000);

    neorv32_gpio_port_set(0x0F);
    neorv32_cpu_delay_ms(1000);
    neorv32_gpio_port_set(0x00);
  }
  else
  {
    neorv32_uart0_print("\nClave incorrecta\n");
    // Reset the register 1
    neorv32_cpu_store_unsigned_word (WB_BASE_ADDRESS + WB_REG2_OFFSET, 0x00000000);
    neorv32_cpu_store_unsigned_word (WB_BASE_ADDRESS + WB_REG1_OFFSET, 0x00000000);

    neorv32_gpio_port_set(0x10);
    neorv32_cpu_delay_ms(1000);
    neorv32_gpio_port_set(0x00);
  }
```

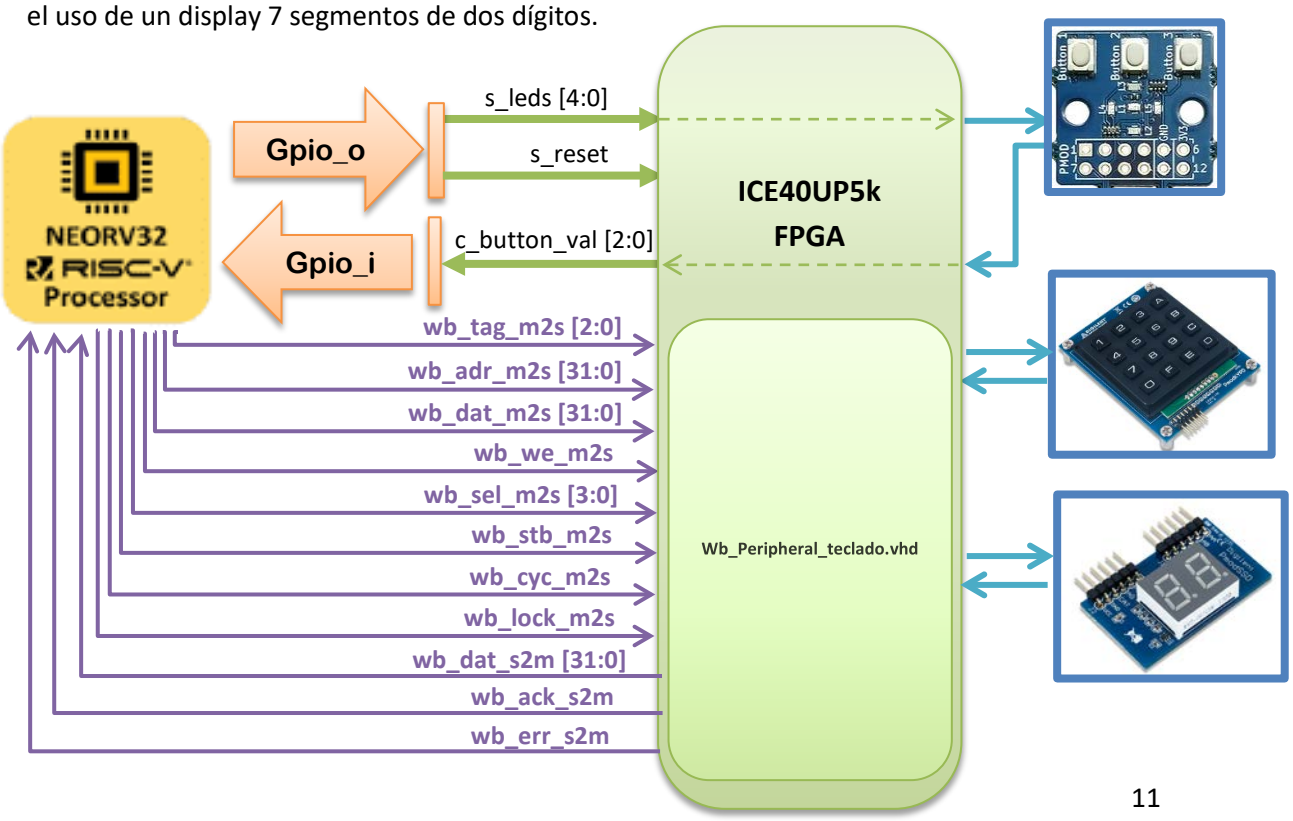
Cuando llamamos a "compara_valores", esta guarda los registros 1, 2 y 3. Con esto lo primero que hace es comprobar si le hemos pedido que compruebe la contraseña.

Tras esto, directamente comparamos los registros 1 y 3, es decir, la contraseña guardada y la introducida, y avisar al usuario mediante un mensaje y una señal de leds, si se ha acertado o se ha fallado.

4. 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.



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.

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

```
-------------------------------------------------
--- COMPARE THE PASSWORD                      ---
-------------------------------------------------
wb_peripheral_teclado_Compare_password_comb: process(
    c_reg1,
    c_reg2,
    c_reg3,
    c_Password_result
    )
begin

    n_Password_result <= c_Password_result;

        if (c_reg2(0) = '1') then  -- Command "A"
            if ((c_reg1(7 downto 0) xnor c_reg3(7 downto 0)) = "11111111") then
                n_Password_result <= c_Password_result(3 downto 1) & "1";
            end if;
        end if;

        if (c_reg2(1) = '1') then -- Command "B"
            if ((c_reg1(15 downto 8) xnor c_reg3(15 downto 8)) = "11111111") then
                n_Password_result <= c_Password_result(3 downto 2) & "1" & c_Password_result(0);
            end if;
        end if;

        if (c_reg2(2) = '1') then  -- Command "C"
            if ((c_reg1(23 downto 16) xnor c_reg3(23 downto 16)) = "11111111") then
                n_Password_result <= c_Password_result(3) & "1" & c_Password_result(1 downto 0);
            end if;
        end if;

       if (c_reg2(3) = '1') then -- Command "D"
            if ((c_reg1(31 downto 24) xnor c_reg3(31 downto 24)) = "11111111") then
                n_Password_result <= "1" & c_Password_result(2 downto 0);
            end if;
        end if;


d process;
```

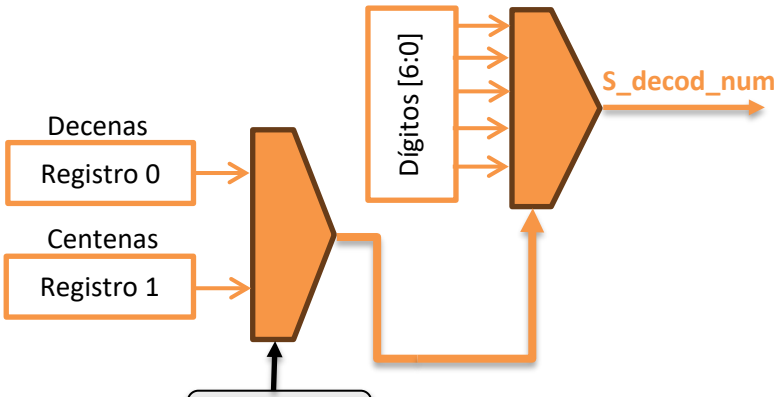
Esta metodología permite una comprobación de la contraseña de los 4 protocolos en paralelo.

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.

La 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.

```
-------------------------------------------------------
 7 segment display                                  ---
-------------------------------------------------------

wb_7segmentDisplay_comb: process(c_counter,c_ds,s_decod_num,c_reg2)
begin
    -- Counter
    n_counter   <= c_counter + 1;

    -- Digit Select
    n_ds        <= c_ds;
    if (c_counter = x"10000") then
        n_ds        <= not(c_ds); -- 183 Hz
        n_counter   <= (others => '0');
        -- 12 MHz / 0x10000 = 12MHz / 65.536 = 183 Hz
    end if;

    -- Display
    aa_o        <= '0';
    ab_o        <= '0';
    ac_o        <= '0';
    ad_o        <= '0';
    ae_o        <= '0';
    af_o        <= '0';
    ag_o        <= '0';
    case(to_integer(unsigned(c_reg2(1 downto 0)))) is
        when 0 => -- Represent:  --
            ag_o    <= '1';

        when others => -- Represent the number
            aa_o        <= s_decod_num(6);
            ab_o        <= s_decod_num(5);
            ac_o        <= s_decod_num(4);
            ad_o        <= s_decod_num(3);
            ae_o        <= s_decod_num(2);
            af_o        <= s_decod_num(1);
            ag_o        <= s_decod_num(0);

    end case;
nd process;
```




```
-------------------------------------------------------
-- D                                              ---
-------------------------------------------------------

WITH (c_ds) SELECT
s_num           <= std_logic_vector(c_reg0(11 downto 0)) when '0',
                   std_logic_vector(c_reg1(11 downto 0)) when others;


WITH (s_num) SELECT
s_decod_num     <=   "1111110" when x"000", -- 0
                     "0000110" when x"001", -- 1
                     "1101101" when x"002", -- 2
                     "1001111" when x"004", -- 3
                     "0010111" when x"008", -- 4
                     "1011011" when x"010", -- 5
                     "1111011" when x"020", -- 6
                     "0001110" when x"040", -- 7
                     "1111111" when x"080", -- 8
                     "0011111" when x"100", -- 9
                     "1111000" when x"200", -- C
                     "1110000" when x"400", -- L
                     "0111101" when others; -- P
```

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.

De forma externa al main, creamos dos funciones, una primera “Lee_teclado”, explicada en las prácticas anteriores, para poder obtener un valor de una pulsación.

```
void Represent_Display(uint8_t Decenas, uint8_t Centenas, uint8_t Enable){

  uint16_t FPGA_display;
  uint16_t Mask;
  uint8_t i;

  for (i=3, Mask = 0x0004, FPGA_display = Decenas ; i<13 ; i++){
    FPGA_display = Decenas == i ? Mask : FPGA_display;
    Mask = (Mask << 1);
  }

  neorv32_cpu_store_unsigned_word (WB_DISPLAY_BASE_ADDRESS + WB_DISPLAY_REG0_OFFSET,  FPGA_display); // Write tens

    for (i=3, Mask = 0x0004, FPGA_display = Centenas ; i<13 ; i++){
    FPGA_display = Centenas == i ? Mask : FPGA_display;
    Mask = (Mask << 1);
  }

  neorv32_cpu_store_unsigned_word (WB_DISPLAY_BASE_ADDRESS + WB_DISPLAY_REG1_OFFSET,  FPGA_display); // Write units

  if(Enable != 0){
      neorv32_cpu_store_unsigned_word (WB_DISPLAY_BASE_ADDRESS + WB_DISPLAY_REG2_OFFSET, 0x00000001); // Order to write on display
  }
  else{
    neorv32_cpu_store_unsigned_word (WB_DISPLAY_BASE_ADDRESS + WB_DISPLAY_REG2_OFFSET, 0x00000000); // Order to write on display
  }
```

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.

Iniciamos el bucle, y para futuras necesidades, en cada ciclo guardamos el valor de todos nuestros registros dedicados a teclado y contraseñas.

```
while(1){
  //to know always whats happening on the registers
  uint32_t registro0 = neorv32_cpu_load_unsigned_word (WB_TECLADO_BASE_ADDRESS + WB_TECLADO_REG0_OFFSET);
  uint32_t registro1 = neorv32_cpu_load_unsigned_word (WB_TECLADO_BASE_ADDRESS + WB_TECLADO_REG1_OFFSET);
  uint32_t registro2 = neorv32_cpu_load_unsigned_word (WB_TECLADO_BASE_ADDRESS + WB_TECLADO_REG2_OFFSET);
  uint32_t registro3 = neorv32_cpu_load_unsigned_word (WB_TECLADO_BASE_ADDRESS + WB_TECLADO_REG3_OFFSET);
  uint32_t registro4 = neorv32_cpu_load_unsigned_word (WB_TECLADO_BASE_ADDRESS + WB_TECLADO_REG4_OFFSET);
```

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.

En caso contrario leeremos el valor pulsado en el teclado y dividiremos entre números y letras.

```
switch(estado)
{
  case 10: //Initial case-->waiting for caracters
    if(registro4 == 0xF)
    {
      neorv32_cpu_store_unsigned_word (WB_TECLADO_BASE_ADDRESS + WB_TECLADO_REG4_OFFSET,0x00000000);
      neorv32_uart0_printf("\nPuerta abierta, tiene 5s...\n");
      Represent_Display(0,12,1);
      neorv32_cpu_delay_ms(5000);
      v_gpio = 0x00;
      neorv32_gpio_port_set(0x20);
      neorv32_gpio_port_set(0x00);
      Represent_Display(0,12,0);
    }
    else{
      //New pulse on the keypad
      Key_value = Lee_teclado();
      if(Key_value != 0xFF){
        if(Key_value != q_key_value){
          if(Key_value < 10){estado=0;}
          else{estado = Key_value;}
          q_key_value = Key_value;
        }
      }
      else{q_key_value = 0xFF;}
    }
  break;
```

```
case 0:  //Number
  Represent_Display(decena,Key_value,1);  //Displays the numbers
  total_value=(total_value<<4)+Key_value;   //Move units to tens
  total_value = total_value & 0xFF; //Take only last numbers and discard the rest
  neorv32_uart0_printf("Total_value: %x\n",total_value);
  decena = Key_value;
  estado = 10;

  //Show registers to see how it works easier
  neorv32_uart0_printf("Registro 1: %x\n",registro1);
  neorv32_uart0_printf("Registro 2: %x\n",registro2);
  neorv32_uart0_printf("Registro 3: %x\n",registro3);
  neorv32_uart0_printf("Registro 4: %x\n",registro4);


break;
```

Si 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.

Cómo 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.

```
//Case 65-69 only for letters
case 65:  //A
  registro1 = neorv32_cpu_load_unsigned_word (WB_TECLADO_BASE_ADDRESS + WB_TECLADO_REG1_OFFSET);
  registro1 = registro1+total_value;
  neorv32_cpu_store_unsigned_word (WB_TECLADO_BASE_ADDRESS + WB_TECLADO_REG1_OFFSET, registro1);

  registro2 = neorv32_cpu_load_unsigned_word (WB_TECLADO_BASE_ADDRESS + WB_TECLADO_REG2_OFFSET);
  registro2 = registro2 + 1;
  neorv32_cpu_store_unsigned_word (WB_TECLADO_BASE_ADDRESS + WB_TECLADO_REG2_OFFSET, registro2);

  estado = 1;
break;
```

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

Se 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.

```
case 1:  //Check A protocol
  neorv32_cpu_delay_ms(500);
  registro4 = neorv32_cpu_load_unsigned_word (WB_TECLADO_BASE_ADDRESS + WB_TECLADO_REG4_OFFSET);
  if((registro4 & 1) != 0) //Condition specified on hardware
  {
    neorv32_uart0_print("\nClave A correcta\n");
    v_gpio = v_gpio+ led1;  //To not disturb other leds
    neorv32_gpio_port_set(v_gpio);
    neorv32_cpu_delay_ms(1000);
    Represent_Display(10,11,0);
    decena = 0;
    Key_value = 0xFF;
    total_value=0;
    estado = 10;
  }
  else
  {
    total_value=0;
    estado = 5;
  }
break;
```

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.

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

```
case 5: //Fail
  neorv32_uart0_print("\nClave incorrecta->Claves reseteadas\n");
  Represent_Display(10,11,1);  //-->CL
  neorv32_gpio_port_set(0x10);  //Red led
  neorv32_cpu_delay_ms(3000);
  neorv32_gpio_port_set(0x20); //General reset except to reg3
  neorv32_gpio_port_set(0x00);
  neorv32_cpu_store_unsigned_word (WB_TECLADO_BASE_ADDRESS + WB_TECLADO_REG3_OFFSET, 0x00555555);
  Represent_Display(10,11,0); //-->--
  v_gpio = 0x00;
  decena = 0;
  Key_value = 0xFF;
  total_value=0;
  estado = 10;

break;
```

```
case 69:  //E-->Reset
  neorv32_gpio_port_set(0x20);  //General reset except to reg3
  neorv32_gpio_port_set(0x00);
  neorv32_cpu_store_unsigned_word (WB_TECLADO_BASE_ADDRESS + WB_TECLADO_REG3_OFFSET, 0x00555555);
  v_gpio = 0x00;
  estado = 10;
  neorv32_uart0_print("\nVariables y claves reseteadas\n");
break;
```

Por 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.

5. 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