Anexo A

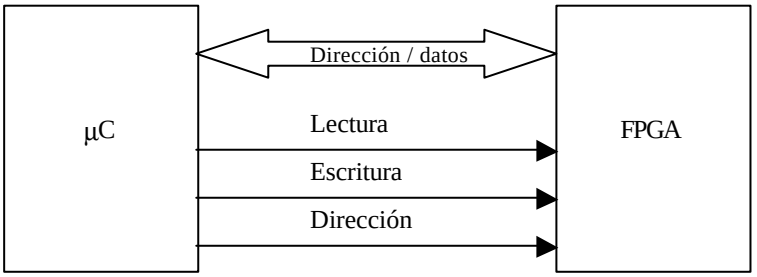
Protocolo de comunicaciones entre FPGA y mC

El protocolo de comunicaciones entre ambos dispositivos se realiza en paralelo, las características de esta comunicación son:

- Comunicación paralela y bidireccional
- 8 bits de datos
- 8 bits de dirección
- Dirección y datos multiplexados
- 3 bits de control

El control de las comunicaciones lo tiene el µC, y es la FPGA la que atiende a las peticiones de este. Se ha implementado de este modo para que la ocupación del módulo en la FPGA sea lo menor posible. El hecho de tener un espacio de direcciones de hasta 8 bits nos permite observar la FPGA desde el punto de vista del µC como un conjunto de 256 registros de escritura y 256 registros de lectura. Sin embargo normalmente se utilizarán muchos menos, por lo que ambos módulos manejan 8 registros de lectura y 13 de escritura, ya que 5 de los registros de escritura se utilizan para proveer a la FPGA de los canales ADC. El aumento del número de registros puede hacerse a nivel de código de una forma muy sencilla.

La conexión entre ambos dispositivos es la siguiente:



Dónde las señales tienen el siguiente significado:

Bus Dir/Datos

Bus de 8 bits en el que confluyen las direcciones y los datos de forma multiplexada. Por tanto se comportan como líneas bidireccionales, por lo que un tercer dispositivo podría hacer uso de este mismo bus

Dirección (dir)

Esta señal valida la dirección que se encuentra en el bus dir/dato. Es activa por flanco de subida y provoca la lectura de la dirección en la FPGA.

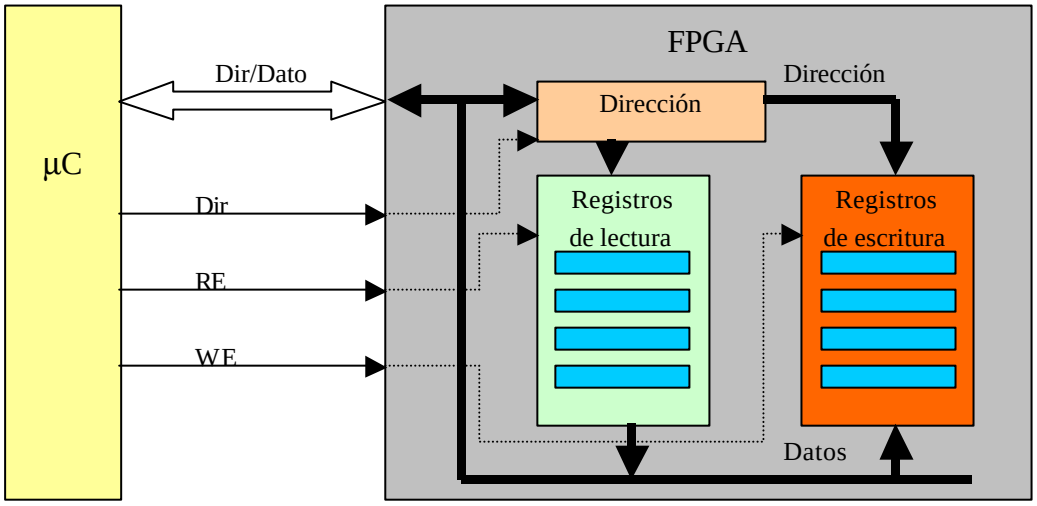
Lectura, RE

Línea de validación de lectura. Es activa por flanco de subida y su activación produce la lectura de un dato del bus dir/dato, escrito por la FPGA.

Escritura, WE

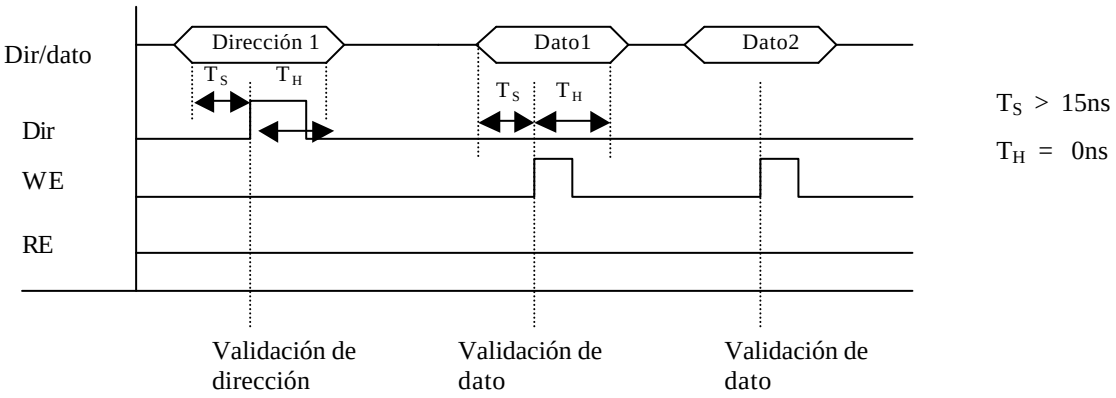
Línea de validación de escritura. Es activa por flanco de subida y su activación produce la escritura del dato del bus dir/dato en la FPGA.

Las comunicaciones se realizan de un modo muy sencillo. El µC coloca en primer lugar una dirección sobre el bus, a continuación la valida, y posteriormente realiza una escritura ó lectura sobre dicha dirección. En el caso de las lecturas toma el dato direccionado por la FPGA, mientras que en las escrituras graba el dato en el registro direccionado por la FPGA. Gráficamente:



Como se puede observar en la FPGA se implementa un registro que almacena la dirección, mientras que existen 2 bancos de registros, de lectura y escritura, direccionados por el registro mencionado y habilitados por las señales de control. Mostramos ahora la temporización de las acciones de escritura y lectura:

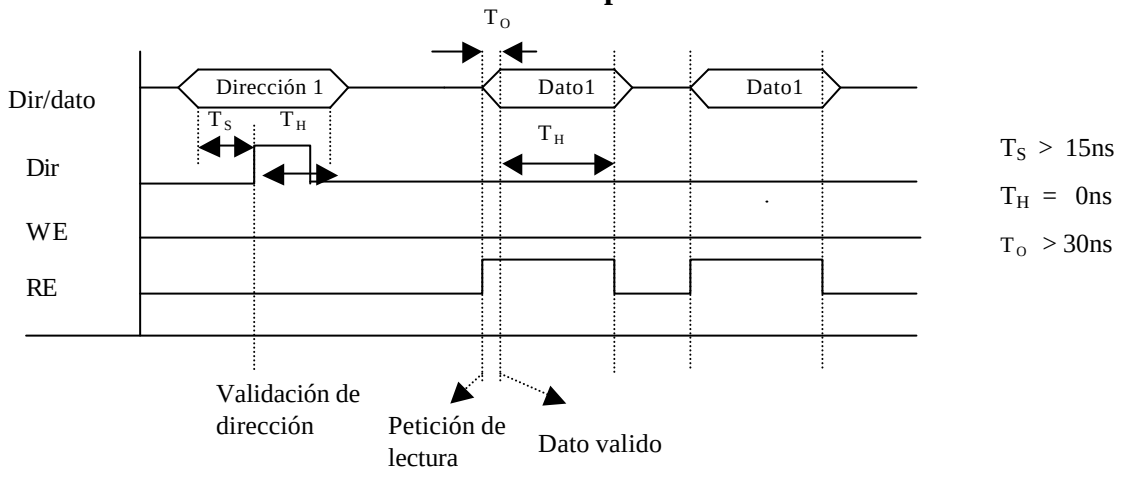
Acción de escritura sobre una dirección desde el punto de vista del mC



En este caso realizamos la escritura del dato1 sobre la dirección 1, y seguidamente realizamos la escritura del dato2, por lo que al final la dirección1 contendrá el dato2.

Los tiempos de setup y hold dependen de 2 motivos, por un lado de la FPGA utilizada, que en este caso no es relevante pues siempre será del mismo tipo (EPF10K10LC84-4). Y también depende del ruteado del diseño, y este puede variar según la implementación completa. El entorno Max-Plus II provee de una herramienta para calcular estos tiempos, el *timing analyzer*, a través del cual podemos calcular estos tiempos según el diseño. Sin embargo estos no variarán mucho respecto a los tomados como mínimos (T_S = 15ns y T_H = 0ns), y sería extraño que fuesen mayores. De todos modos el µC no puede ejecutar instrucciones en un tiempo inferior a 200ns, por tanto siempre cumplirá los tiempos.

Acción de Lectura de una dirección desde el punto de vista del mC



En este caso aparece el tiempo de output T_O, que es el tiempo necesario para que el de lectura para el µC sea válido. Igual que antes no tendremos problemas con los tiempos mínimos. Por otro lado, si no cambiamos el registro de direcciones, continuaremos leyendo el mismo registro, cuyo valor puede cambiar a lo largo del tiempo.

Implementación del módulo de comunicaciones en el mC

A continuación mostramos las funciones escritas en C para el compilador utilizado en el proyecto (CCS), que implementan la lectura y la escritura de un dato en uno de los registros de la FPGA. Constituyen el módulo de comunicaciones del µC.

```
#ifndef _COMUNICA_H
#define _COMUNICA_H

/*******************************************************

      COMUNICA.H

      Autores: Juan Maria Castro & Jesus Donate
      Fecha: 1-5-2002
      Version: 1.0
-------------------------------------------------------

      Modulo de comunicaciones entre uC y FPGA.

      Contiene 2 funciones para la lectura y escritura
      de un dato de 8 bits en una direccion de los
      registros de la FPGA.

      Si el modulo de la FPGA no contiene la direccion
      enviada el dato no es escrito, en caso de ser leido
      el resultado es impredecible

*******************************************************/

//Definimos las señales de transmision de datos con la FPGA
#byte dir_dato=0x08
#define entrada 0xFF
#define salida  0x00

#byte porte=0x09
#bit  write=porte.1
#bit  read =porte.0
#bit  dir  =porte.2

#use fast_io(d)
#use fast_io(e)

/************************************

      inicia_comunicaciones

      Configura las señales del modulo
      de comunicaciones en sus estados
      apropiados.
-------------------------------------
      Parametros: Ninguno
      Valor devuelto: Ninguno
************************************/
void inicia_comunicaciones(void)
{
      set_tris_d(entrada); //Lo definimos de entrada
      read=write=dir=0;
      set_tris_e(salida); //Configurado de Salida
}
```

```
/**************************************

      escribe_dato_fpga

      Escribe en la FPGA el dato que
      se le pasa en la direccion
      indicada.
---------------------------------------
      Parametros:
            direccion: E,Direccion del
                     registro de la FPGA
            dato: E, dato a enviar

      Valor devuelto: Ninguno
**************************************/
void escribe_dato_fpga(int direccion,int dato)
{
      //Colocamos la direccion
      dir_dato=direccion;
      set_tris_d(salida);
      //Escritura de la direccion
      dir=1;
      dir=0;
      //Colocamos el dato
      dir_dato=dato;
      //Escritura del dato
      write=1;
      write=0;
      set_tris_d(entrada);
}


/**************************************

      lee_dato_fpga

      Lee un dato de la FPGA de la
      direccion que se le pasa como
      parametro.
---------------------------------------
      Parametros:
            direccion: E,Direccion del
                     registro de la FPGA

      Valor devuelto: valor leido del
                  registro de la FPGA
**************************************/
int lee_dato_fpga(int direccion)
{
      int dato_entrada;

      //Colocamos la direccion
      dir_dato=direccion;
      set_tris_d(salida);
      //Escritura de la direccion
      dir=1;
      dir=0;
      set_tris_d(entrada);
      //leemos el dato
      read=1;
      dato_entrada=dir_dato;
```

```
	read=0;
	return(dato_entrada);
}

#endif _COMUNICA_H
```

Estas funciones pueden ser utilizadas para cualquier aplicación de usuario.

Implementación del módulo de comunicaciones en la FPGA

Mostramos el código del módulo para la FPGA. Este ha sido implementado en VHDL[1] y a sido programado como módulo genérico[2] y para un máximo de 13 registros de entrada (8 genéricos y 5 para los valores de los conversores A/D del µC), y 8 de salida. En el siguiente punto se muestra como implementar más registros.

```
library ieee;
use ieee.std_logic_1164.all;

entity comunicaciones_pic is
	generic( ad1_on: integer:=1;
		 ad2_on: integer:=1;
		 ad3_on: integer:=1;
		 ad4_on: integer:=1;
		 ad5_on: integer:=1;
		 registros_recepcion: integer :=8;
		 registros_transmision: integer :=8);

	port(
		dir_dato		:INOUT std_logic_vector(7 downto 0);
		write,dir		:IN	std_logic;
		read			:IN	bit;
		rx1,rx2,rx3,rx4,rx5,rx6,rx7,rx8	:bufferstd_logic_vector(7 downto 0);
		ad1,ad2,ad3,ad4,ad5	:buffer std_logic_vector(7 downto 0);
		tx1,tx2,tx3,tx4,tx5,tx6,tx7,tx8	:IN	std_logic_vector(7 downto 0)
		);
end comunicaciones_pic;

architecture rtl of comunicaciones_pic is
	SIGNAL direccion:  std_logic_vector(3 downto 0);
	SIGNAL dir_dato_e: std_logic_vector(7 downto 0);
	SIGNAL dir_dato_s: std_logic_vector(7 downto 0);
	SIGNAL dir_dato_f: std_logic_vector(7 downto 0);

begin
	process(dir)
	begin
		if dir'event and dir='1' then
			direccion <= dir_dato_e(3 downto 0);
		end if;
	end process;

	process(write)
	begin
		if write'event and write='1' then
			if direccion= "0000" and registros_recepcion >=1 then
				rx1 <= dir_dato_e;
			elsif direccion= "0001" and registros_recepcion >=2 then
				rx2 <= dir_dato_e;
			elsif direccion= "0010" and registros_recepcion >=3 then
				rx3 <= dir_dato_e;
			elsif direccion= "0011" and registros_recepcion >=4 then
				rx4 <= dir_dato_e;
			elsif direccion= "0100" and registros_recepcion >=5 then
```

[1] Lenguaje de descripción hardware. Es el lenguaje más utilizado para la programación de FPGAs y CPLDs

[2] Los módulos genericos son configurables en algunos parámetros, adaptándose a las necesidades del momento.

```
                        rx5 <= dir_dato_e;
                elsif direccion= "0101" and registros_recepcion >=6 then
                        rx6 <= dir_dato_e;
                elsif direccion= "0110" and registros_recepcion >=7 then
                        rx7 <= dir_dato_e;
                elsif direccion= "0111" and registros_recepcion >=8 then
                        rx8 <= dir_dato_e;
                elsif direccion= "1000" and ad1_on=1 then
                        ad1 <= dir_dato_e;
                elsif direccion= "1001" and ad2_on=1 then
                        ad2 <= dir_dato_e;
                elsif direccion= "1010" and ad3_on=1 then
                        ad3 <= dir_dato_e;
                elsif direccion= "1011" and ad4_on=1 then
                        ad4 <= dir_dato_e;
                elsif direccion= "1100" and ad5_on=1 then
                        ad5 <= dir_dato_e;
                else
                        if registros_recepcion >=1 then
                                        rx1 <= rx1;
                        end if;
                        if registros_recepcion >=2 then
                                        rx2 <= rx2;
                        end if;
                        if registros_recepcion >=3 then
                                        rx3 <= rx3;
                        end if;
                        if registros_recepcion >=4 then
                                        rx4 <= rx4;
                        end if;
                        if registros_recepcion >=5 then
                                        rx5 <= rx5;
                        end if;
                        if registros_recepcion >=6 then
                                        rx6 <= rx6;
                        end if;
                        if registros_recepcion >=7 then
                                        rx7 <= rx7;
                        end if;
                        if registros_recepcion >=8 then
                                        rx8 <= rx8;
                        end if;
                        if ad1_on =1 then
                                        ad1<=ad1;
                        end if;
                        if ad2_on =1 then
                                        ad2<=ad2;
                        end if;
                        if ad3_on =1 then
                                        ad3<=ad3;
                        end if;
                        if ad4_on =1 then
                                        ad4<=ad4;
                        end if;
                        if ad5_on =1 then
                                        ad5<=ad5;
                        end if;

                end if;
        end if;
end process;

process(read)
begin
        if read'event and read='1' then
                        if    direccion= "0000" and registros_transmision >=1 then
                                dir_dato_s <= tx1;
                        elsif direccion= "0001" and registros_transmision >=2 then
                                dir_dato_s <= tx2;
                        elsif direccion= "0010" and registros_transmision >=3 then
                                dir_dato_s <= tx3;
                        elsif direccion= "0011" and registros_transmision >=4 then
                                dir_dato_s <= tx4;
                        elsif direccion= "0100" and registros_transmision >=5 then
                                dir_dato_s <= tx5;
                        elsif direccion= "0101" and registros_transmision >=6 then
                                dir_dato_s <= tx6;
```

```
                        elsif direccion= "0110" and registros_transmision >=7 then
                               dir_dato_s <= tx7;
                        elsif direccion= "0111" and registros_transmision >=8 then
                               dir_dato_s <= tx8;
                        else
                               dir_dato_s <= "00000000";
                        end if;
               end if;
        end process;

        with read select
               dir_dato_f <= dir_dato_s when '1',
                                 "ZZZZZZZZ" when '0';


        dir_dato_e <= dir_dato;
        dir_dato   <= dir_Dato_f;

end rtl;
```

Como se puede observar en este código se han generado 3 procesos que corren en paralelo, y que corresponden a la lectura y escritura de datos y a la escritura de una dirección.

Aumento del numero de registros de comunicación

Como ya sabemos el módulo de comunicaciones está limitado a 8 registros de entrada + 8 de salida + los 5 conversores A/D. Este numero puede ser reducido simplemente disminuyendo las variables de control del módulo de la FPGA. Sin embargo para aumentarlo es necesario retocar el código. A continuación se explica como realizar este proceso.

Solamente es necesario realizar cambios en el módulo de la FPGA, ya que el módulo del µC está preparado para los 256 registros de E y S. El programa de comunicaciones de la FPGA consiste en 3 procesos que corren en paralelo, así que será necesario retocar los tres procesos, además de la instanciación[1] del componente. Pero hemos de tener en cuanta, que si solamente añadimos registros de recepción no es necesario retocar el proceso de lectura, del mismo modo no será necesario retocar el proceso de escritura si solamente se añaden registros de transmisión.

1. **Modificación de la instanciación**
   La actual tiene la forma:

```
port(
dir_dato            :INOUT std_logic_vector(7 downto 0);
write,dir           :IN          std_logic;
read                :IN          bit;
rx1,rx2,rx3,rx4,rx5,rx6,rx7,rx8   :buffer std_logic_vector(7 downto 0);
ad1,ad2,ad3,ad4,ad5               :buffer std_logic_vector(7 downto 0);
tx1,tx2,tx3,tx4,tx5,tx6,tx7,tx8   :IN     std_logic_vector(7 downto 0)
);
```

   Puesto que se deben definir todos los puertos del componente, al añadir registros tendremos que darles un nombre en esta sección. Si añadimos un registro de transmisión, le pondremos el nombre *tx9* o cualquier otro con el se desee identificar.

[1] La instanciación es la declaración de los puertos de un componente en VHDL, suele comenzar con la directiva PORT

2. Modificación del proceso de direccionamiento

Su forma actual es:

```
process(dir)
begin
	if dir'event and dir='1' then
		direccion <= dir_dato_e(3 downto 0);
	end if;
end process;
```

Este proceso lee la dirección del bus y la pasa al registro de direcciones, pero solo toma los bits necesarios, en este caso son 4 porque solamente es necesario direccionar 13 posiciones, por tanto tendremos que modificar su extensión solamente si es necesario, según el numero de registros que se desee añadir. Por ejemplo, para dar cabida a 32 registros tendríamos:

```
process(dir)
begin
	if dir'event and dir='1' then
		direccion <= dir_dato_e(4 downto 0);
	end if;
end process;
```

3. Modificación del proceso de lectura

El proceso actual es:

```
process(read)
begin
	if read'event and read='1' then
			if    direccion= "0000" and registros_transmision >=1 then
				dir_dato_s <= tx1;
			elsif direccion= "0001" and registros_transmision >=2 then
				dir_dato_s <= tx2;
			elsif direccion= "0010" and registros_transmision >=3 then
				dir_dato_s <= tx3;
			elsif direccion= "0011" and registros_transmision >=4 then
				dir_dato_s <= tx4;
			elsif direccion= "0100" and registros_transmision >=5 then
				dir_dato_s <= tx5;
			elsif direccion= "0101" and registros_transmision >=6 then
				dir_dato_s <= tx6;
			elsif direccion= "0110" and registros_transmision >=7 then
				dir_dato_s <= tx7;
			elsif direccion= "0111" and registros_transmision >=8 then
				dir_dato_s <= tx8;
			else
				dir_dato_s <= "00000000";
			end if;
	end if;
end process;
```

Como se puede observar, se deben tener en cuanta todos los estados de direccionamiento posibles, por ello si un caso no es contemplado el registro *dir_dato_s* contendrá el valor 0, como sería el caso de lectura del registro 9, que no está implementado. Por tanto para aumentar el numero de registros es necesario aumentar los casos posibles, añadiendo las líneas:

```
elsif direccion= "DIRECCION EN BINARIO" and registros_transmision >= NUMERO DE REGISTRO
then
			dir_dato_s <= Nombre del registro;
```

Donde Dirección en binario es la dirección del registro añadido en binario, numero de registro es la dirección del registro añadido en números naturales, y nombre del registro es el nombre que se le de en la instanciación.

4. Modificación del proceso de escritura

Este proceso es el más largo:

```
process(write)
begin
	if write'event and write='1' then
		if direccion= "0000" and registros_recepcion >=1 then
			rx1 <= dir_dato_e;
		elsif direccion= "0001" and registros_recepcion >=2 then
			rx2 <= dir_dato_e;
		elsif direccion= "0010" and registros_recepcion >=3 then
			rx3 <= dir_dato_e;
		elsif direccion= "0011" and registros_recepcion >=4 then
			rx4 <= dir_dato_e;
		elsif direccion= "0100" and registros_recepcion >=5 then
			rx5 <= dir_dato_e;
		elsif direccion= "0101" and registros_recepcion >=6 then
			rx6 <= dir_dato_e;
		elsif direccion= "0110" and registros_recepcion >=7 then
			rx7 <= dir_dato_e;
		elsif direccion= "0111" and registros_recepcion >=8 then
			rx8 <= dir_dato_e;
		elsif direccion= "1000" and ad1_on=1 then
			ad1 <= dir_dato_e;
		elsif direccion= "1001" and ad2_on=1 then
			ad2 <= dir_dato_e;
		elsif direccion= "1010" and ad3_on=1 then
			ad3 <= dir_dato_e;
		elsif direccion= "1011" and ad4_on=1 then
			ad4 <= dir_dato_e;
		elsif direccion= "1100" and ad5_on=1 then
			ad5 <= dir_dato_e;
		else
			if registros_recepcion >=1 then rx1 <= rx1;
			end if;
			if registros_recepcion >=2 then rx2 <= rx2;
			end if;
			if registros_recepcion >=3 then rx3 <= rx3;
			end if;
			if registros_recepcion >=4 then rx4 <= rx4;
			end if;
			if registros_recepcion >=5 then rx5 <= rx5;
			end if;
			if registros_recepcion >=6 then rx6 <= rx6;
			end if;
			if registros_recepcion >=7 then rx7 <= rx7;
			end if;
			if registros_recepcion >=8 then rx8 <= rx8;
			end if;
			if ad1_on =1 then ad1<=ad1;
			end if;
			if ad2_on =1 then ad2<=ad2;
			end if;
			if ad3_on =1 then ad3<=ad3;
			end if;
			if ad4_on =1 then ad4<=ad4;
			end if;
			if ad5_on =1 then ad5<=ad5;
			end if;
		end if;
	end if;
```

Por una parte se debe añadir el nuevo registro para acceder a el mediante el direccionamiento, para ello se añade la siguiente línea:

```
elsif direccion= "Dirección en binario" and registros_recepcion >=num de registro
then   nombre del registro <= dir_dato_e;
```

Donde Dirección en binario es la dirección del registro añadido en binario, numero de registro es la dirección del registro añadido en números naturales, y nombre del registro es el nombre que se le de en la instanciación.

Y por otro lado se debe añadir unas líneas para evitar que el registro pierda sus datos cuando no es direccionado, asignanso a cada registro su propio valor:

```
if registros_recepcion >=numero de registro then
       nombre del registro <= nombre del registro;
end if;
```

Estos son los cambios que se deben realizar, sin embargo el aumento del numero de registro trae consigo un aumento del tamaño de ocupación de la FPGA, que en muchos casos puede ser innecesario.