Capítulo 3

Desarrollo del sistema maestro para la comunicación entre la FPGA y la interfaz Cypress

En los cap´itulos XX YY

para que?

Hasta el momento, se ha descripto una comunicación que intercambia datos entre una PC y el controlador FX2LP de Cypress. Sin embargo, esto sólo no es suficiente, ya que el sistema debe estar dotado, además, de un dispositivo que sea emisor y receptor de los datos que el controlador intercambia con la PC.

capaz de intercambiar datos entre el controlador y la PC

Para el desarrollo de una comunicación funcional de sistemas basado en FPGA es necesario implementar un módulo de comunicación dentro de la lógica que se implementa en el FPGA mismo. Por este motivo es mandante la utilización de un FPGA. Luego, es necesario identificar cuales son los mecanismos para la lectura y escritura de datos, las señales que intervienen y los puertos con los que debe interactuar el FPGA.

no entiendo esta oraci´on

La Figura 3.1 muestra un esquema en el cual se observa, como productor y consumidor de datos, un desarrollo genérico, implementado dentro del FPGA. Los datos fluyen desde el FPGA al controlador FX2LP a través de una máquina de estados finitos (MEF), que también provee las señales de control.

A continuación, se justifica la elección del FPGA y la placa de desarrollo utilizados, se detallarán las señales que intervienen en el funcionamiento de la interfaz y los protocolos de lectura y escritura de modo asíncrono. Esto dará lugar a la elaboración de una máquina de estados que luego podrá ser plasmada en un código que será sintetizado en el FPGA. Para la elaboración del código de síntesis se utiliza el lenguaje de descripción de hardware VHDL.

Además, se explica el desarrollo de un circuito desarrollado como interconexión entre las distintas placas de desarrollo que se utilizan en este trabajo.

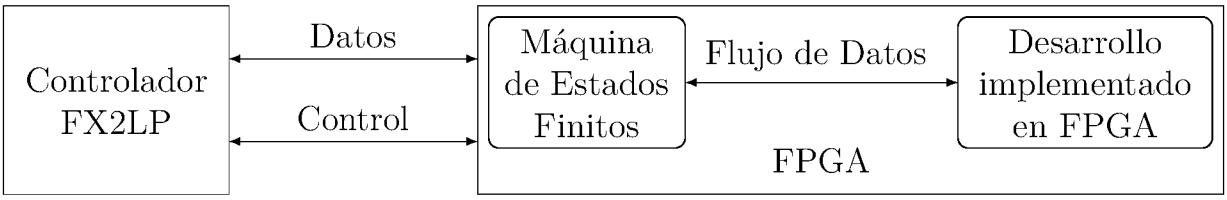


Figura 3.1: Esquema conceptual del flujo de datos hasta el controlador

3.1. Elección de la FPGA

En la implementación de una comunicación, para poder transmitir y recibir datos, los componentes que intervienen deben seguir un protocolo establecido, de forma tal que cada dispositivo sepa que procedimiento efectuar. Por este motivo, una vez definido que se utiliza una interfaz intermedia entre la PC y un FPGA y que dicha interfaz es el circuito integrado EZ-USB FX2LP de Cypress, se deberá configurar un FPGA para que reciba y envíe datos a la interfaz.

En base a los materiales disponibles, se evaluaron tres placas de desarrollo diferentes, todas con FPGA de Xilinx. La placa Spartan-3E Starter, comercializada por Digilent es una placa muy potente debido a la gran cantidad de periféricos que incorpora, entre los que se destacan su pantalla LCD, los 18 MB de memoria flash sumados a 64 MB de SDRAM y la gran cantidad de transceptores tales como Ethernet, PS/2 para teclados y JTAG para programación y depuración.

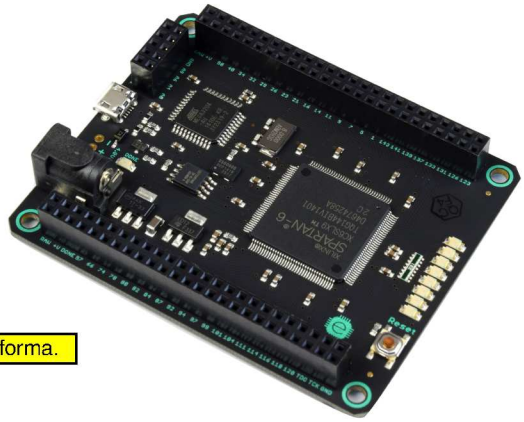
Por su parte, la Nexys 3, tiene como núcleo un FPGA Spartan-6, es decir, un FPGA tecnológicamente superior, lo que le brinda mayor cantidad de bloques lógicos programables, debido a que posee transistores más pequeños en su circuito integrado. La miniaturización de los transistores, a su vez, otorga la posibilidad de una mayor velocidad de operación. La placa Nexis 3 también posee una gran gama de periféricos tales como pulsadores, interruptores, displays led de 7 segmentos, diferentes tipos de memorias, CODEC para comunicare por Ethernet 10/100 o USB 2.0.

A diferencia de las anteriores, la placa de desarrollo Mojo v3 de Alchitry, es más modesta, pero no menos potente. Al igual que la Nexys 3, cuenta con un FPGA Spartan 6 de Xilinx. Dispone también de 84 puertos digitales configurables como entrada y/o salida, 8 entradas analógicas, 8 LED's de propósito general, un botón de tipo pulsador. La principal ventaja de esta placa de desarrollo es que posee un costo notablemente inferior a las anteriores.

Se elige para el desarrollo que se presenta en este trabajo la placa de desarrollo Mojo v3 debido a que se incorporan varios periféricos al kit CY3684 utilizado para el desarrollo de la interfaz. Además de esto, posee un FPGA superior a la placa Spartan 3E Starter, por lo que brinda la posibilidad de elaborar sistemas más complejos y veloces. En otras palabras la Mojo se selecciona por su bajo costo, versatilidad y por que está dotada por un Spartan-VI de Xilinx, que es un FPGA con una buena relación entre recursos, rendimiento, velocidad y precio.

La Mojo v3, la cual se observa en la Figura 3.2, es una placa de desarrollo muy económica para prototipado, es decir, la fabricación de modelos funcionales. Para ello los puertos se disponen en un arreglo de pines a través de los cuales es posible acoplar el dispositivo que sea necesario. Se dispone en el mercado de otros circuitos impresos que se conectan a los pines y contienen un grupo de periféricos para propósito general. Estos impresos, se denominan shields (*escudo* traducido al castellano). se obtiene así una placa de desarrollo a la medida de las necesidades de cada proyecto. El usuario también puede diseñar sus shields o conectar las entradas y salidas de otros dispositivo mediante cables.

Además de los shields, los diseñadores pensaron en que no sea necesario ninguna herramienta extra a la hora de programar la FPGA. Para ello, dotaron al sistema de un microcontrolador ATmega32U4 de Atmel con un programa de tipo bootloader, que se encarga de transferir la configuración del FPGA cargada desde una memoria flash incorporada, o trasmitida por el usuario desde una PC, a través de un transceptor USB que contiene el microcontrolador. Luego, el controlador es colocado en modo esclavo y se configura de forma tal que dota al sistema de una comunicación entre la FPGA y una PC, vía USB y se utiliza su ADC para leer los puertos analógicos.



Yo explicaria esto de otra forma.

Figura 3.2: Placa de prototipado rápido MOJO v3, diseñada por Embedded Micro

Una vez llegado a este punto, el lector podría preguntar con toda razón ¿por qué es necesario realizar un sistema de comunicación USB extra, si ya cuenta con un microcontrolador que se encarga de dicho asunto? La respuesta se basa en el ancho de banda del sistema de comunicación que dispone la placa. La línea de controladores ATmega incorpora puertos USB 2.0 full-speed. Esto quiere decir que puede enviar datos a una tasa de $12\,\mathrm{Mbit\,s^{-1}}$. Además, la comunicación entre ambos chips se realiza via SPI (*SerialPeripheralInterface*, o en español Interfaz Serie de Periféricos), comandada por un cristal de cuarzo de 50 MHz, ofreciendo una velocidad de salida que puede resultar insuficiente a los fines de este trabajo. Se pretende dotar al sistema del mayor ancho de banda posible, utilizando la capacidad de USB 2.0 High-Speed, de hasta 480 Mbps.

Maquina de estados finita (MEF)

para que?

3.2. Señales de Control

Resulta necesario identificar las entradas y salidas del módulo de comunicación que será implementado en FPGA. El diagrama en bloques que se observa en la Figura 3.1 muestra que la MEF diseñada posee señales que se comunican con el controlador FX2LP por un lado. Por el otro, tiene otras señales que intercambian datos con un sistema genérico que será desarrollado dentro el FPGA. Por lo tanto, en esta sección busca determinar cuales son y como operan las señales que intervienen en cada caso.

y otras señales

En esta seccion se detallara cuales son las señales....

Señales entre el FPGA y la interfaz FX2LP

La Figura 3.3 muestra los puertos a través de los cuales se conectan las memorias FIFO del controlador FX2LP con un dispositivo de control externo, el cual se implementa en este desarrollo a través del FPGA Spartan-IV de la placa Mojo. Las señales de control son:

- IFCLK: señal de reloj. No es necesario en caso de conectar la interfaz en modo asincrónico. La señal de reloj puede ser provista por el controlador o por el dispositivo de control en forma programable.
- FDATA[15:0]: constituye el bus de datos. Según se programe, este puede ser de 8 o 16 bits, en forma independiente para cada EP.

End point ?

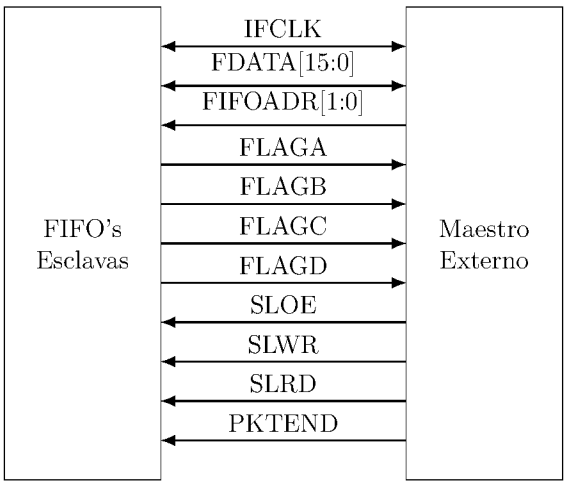


Figura 3.3: Señales de la interfaz entre las FIFO's y un maestro externo

- FIFOADDR[1:0]: puerto de direcciones. A través de él se selecciona la memoria activa en el bus.
- FLAGx: Los cuatro puertos de flag son configurables e indican memoria llena, vacía o un nivel programable. También pueden indicar el estado de una memoria específica o de la que se encuentra activa a través de FIFOADDR.
- SLOE, SLWR, SLRD: son las señales de control. A través de ellas el maestro entrega las ordenes de lectura y escritura.
- PKTEND: a través de este puerto el maestro indica que terminó una transferencia de datos.

Las señales FIFOADR[1:0] se utilizan para seleccionar la memoria FIFO sobre la que se escriben o leen los datos. Cada una de estas memorias está asociada a un extremo (EP) determinado. Estos extremos poseen dirección hexadecimal 02, 04, 06 y 08 para el sistema USB comandado microcontrolador 8051 incorporado en el circuito integrado FX2LP. Las memorias FIFO tienen dirección binaria ″00″,″ 01″,″ 10″ y ″11″ en los puertos FIFOADR[1:0]. En la Tabla 1 se muestran... bla 3.1 las direcciones asociadas entre cada una de las memorias FIFO y los EP. Se destaca que ′0′ y ′1′ en cada puerto FIFOADR equivale a niveles de tensión bajo y alto, respectivamente.

FIFOADR[1:0]	EP (USB)
00	0x02
01	0x04
10	0x06
11	0x08

Tabla 3.1: Direcciones de selección de memoria activa

Los puertos que indican la ocurrencia de eventos particulares en las memorias, como que la memoria se encuentra llena, vacío o el alcance de una cantidad de datos determinada, son programables. Es decir, al momento de realizar la configuración del controlador FX2LP, el

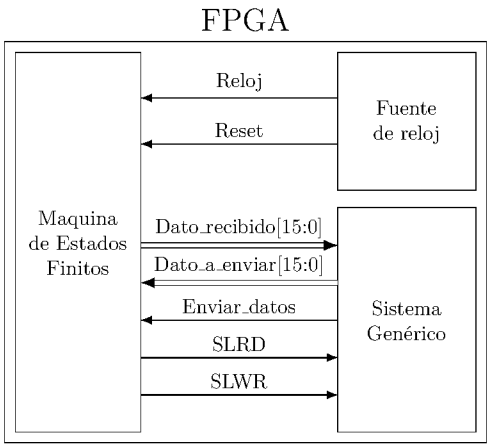


Figura 3.4: Señales internas que intercambian datos dentro del FPGA

desarrollador puede seleccionar que señales estarán presentes en los puertos FLAGA, FLAGB, FLAGC y FLAGD.

La configuración que se implementó en este trabajo, tal como se menciona en el Capítulo 2, dispone al EP 0x02 como puerto de entrada USB (es decir, salida desde el FPGA) y al EP 0x08 como salida USB (o sea, entrada para el FPGA). A su vez, el puerto FLAGA señala que la memoria FIFO relacionada al EP 0x02 está llena y el FLAGB indica que la memoria FIFO relacionada al EP 0x08 está vacía.

Señales de comunicación interna en el FPGA

No entiendo este parrafo

Desde el punto de vista de los sistemas que serán desarrollados en el FPGA, es decir, los sistemas para los cuales tendrán mayor relevancia los datos que fluyan a través de la comunicación establecida por este trabajo, será necesario poder leer los datos que arriben. A su vez, debe ser capaz de poder colocar datos en el extremo de la MEF al que puede accede y emitir la orden de que los envíe. También debe poder conocer cuando los datos que colocó en la MEF fueron enviados, a fin de poder colocar datos nuevos y cuando arriban datos nuevos que deben ser leídos.

Yo explicaria mejor esto debido a que no se entiende bien

Por su parte, la MEF necesitará para su correcto funcionamiento, de una señal de reloj. Debido a que la comunicación con la interfaz es asíncrona y no se dispone de señal de reloj entre ello, el generador de la señal de sincronismo lo debe proveer el FPGA. Es por ello, que también la MEF contará con una señal de reloj. A fin de asegurar una correcta inicialización del dispostivo, también se incorpora una señal de reset asíncrono.

Tal como se observa en la Figura 3.4, los datos cuentan con entradas y salidas independientes. La señal *Enviar_datos* se compone de un bit de entrada para la MEF que lo provee un sistema genérico, que ocupa el lugar que tendrán los sistemas que se desarrollen. Por su parte, el sistema genérico tomará los datos que genera la MEF para comunicarse con la interfaz, a fin de economizar recursos. Además, se incorpora un bloque que genera la fuente de reloj y sirve, a su vez, para emitir la señal de reset que inicializa el dispositivo.

La maquina de estados cuenta con salida y entrada de datos independiente no?

Esto tampoco lo entiendo bien Por que economiza recursos?

3.2.1. Descripción del puerto en VHDL

Sigo sin entender, creo que aca habria que hacer una introduccion un poco mas detallada, viendo el vhdl creo que lo que estas mostrando es la definicion de las entradas y salidas del bloque de la maquina de estados no?

Hasta acá, se pudieron identificar todos los puertos que intervienen en el funcionamiento de la comunicación que se implementó. Por lo tanto, se encuentran dadas las condiciones para poder describir la entidad en VHDL. Dicha entidad se detalla a continuación.

```
entity fx2lp_interfaz is
generic(
    constant in_ep_addr:  std_logic_vector(1 downto 0) := "00";
    constant out_ep_addr: std_logic_vector(1 downto 0) := "11";
    constant port_width:  integer := 16
    );
port(
    reloj: in std_logic;
    reset: in std_logic;
    -- desde y hacia la interfaz
    fdata:    inout std_logic_vector(port_width-1 downto 0);
    fifoaddr: out   std_logic_vector(1 downto 0);
    sloe:     out   std_logic;
    slrd:     out   std_logic;
    slwr:     out   std_logic;
    pktend:   out   std_logic;
    -- EP2 isoc in (hacia pc)
    -- EP8 bulk out (desde pc)
    flaga: in   std_logic;   -- EP2_full-->FLAG_Lleno
    flagb: in   std_logic;   -- EP8_empty-->FLAG_Vacio
    flagc: in   std_logic;   -- EP8_full-->sin uso
    flagd: in   std_logic;   -- EP2_empty-->sin uso
    -- desde y hacia el sistema
    enviar_dato: in  std_logic;
    d_recibido:  out std_logic_vector(port_width-1 downto 0);
    d_a_enviar:  in  std_logic_vector(port_width-1 downto 0)
    );
end fx2lp_interfaz;
```

Debido a que los EP son configurables y podrían variar, conforma al criterio de algún desarrollador, se declaran como constantes las direcciones de las memorias FIFO correspondientes a cada EP, es decir, el de salida y el de entrada. A su vez, ya que en ancho de bus de las memorias FIFO pueden variar, este se define como constante y puede valer 8 o 16 bits. La configuración por defecto, se colocó la implementación detallada en este trabajo. Esto es, 16 bits como ancho de bus, "00" es la dirección de la memoria FIFO conectada al EP de entrada (respecto al Host) y "11" la dirección correspondiente a la memoria FIFO relacionada al EP de salida.

Me cuesta entender que quiere decir este parrafo

3.3. Diseño de la Máquina de Estados Finitos

A modo conceptual, la máquina de estados finitos (MEF) que se implementa en este trabajo es capaz de realizar dos tareas, bien definidas: leer datos desde la memoria FIFO destinada al EP de salida (desde la PC) y escribir datos en la memoria FIFO que corresponde al EP de entrada (hacia el host). Resulta entonces del interés de este trabajo, comprender los protocolos de lectura y escritura que debe seguir el sistema, los que se detallan en las secciones siguientes.

Luego de ello, se abordará una descripción narrativa de las señales y, a su vez, se la reforzará con el diagrama de flujo con el funcionamiento de la máquina de estados finitos. Lo que dará lugar a proceder con la descripción de la misma.

No llego a entender que significa descripcion narrativa, y la ultima oraci´on

3.3.1. Lectura de datos desde la memoria FIFO

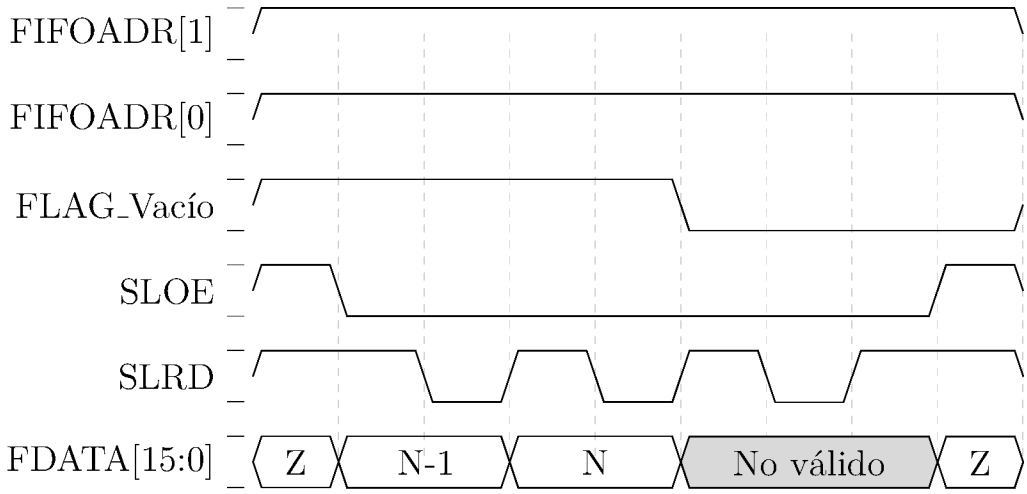


Figura 3.5: Diagrama temporal de la lectura de datos desde la memoria FIFO por un FPGA

flanco negativo?

Para efectuar operaciones de lectura en régimen asíncrono, como se muestra en la Figura 3.5, en primer lugar, el FPGA debe colocar en los puertos FIFOADR[1:0] la dirección de la memoria sobre la que desea efectuar esta operación. En el caso de la configuración realizada en este trabajo, ″11″, la que corresponde al EP8. Luego, debe ser activada la señal SLOE, la cual coloca en los puertos FDATA[15:0] los datos almacenados en la memoria FIFO apuntada por FIFOADR[1:0]. El dato disponible en la salida de la memoria FIFO siempre será el más antiguo, es decir, el que se almacenó antes. En el cambio de asertiva a negativa de la señal SLRD, la memoria FIFO aumenta un contador que selecciona la dirección del próximo dato, y coloca este dato en el puerto FDATA[15:0]. Cuando el contador es aumentado, los datos antiguos se descartan y no pueden ser recuperados luego.

Una vez que todos los datos fueron leídos, es decir, que el contador de la memoria ha alcanzado un valor N de datos, iguales a los almacenados, se activa la señal FLAG_Vacío (para este trabajo, FLAGB). Mientras SLOE no está activo, el puerto FDATA[15:0] permanece en estado de alta impedancia. En la Figura 3.5 se puede observar también que tanto las señales FLAG_Vacío, SLOE y SLRD son asertivas en '0'. En otras palabras,dichas señales son activas cuando tienen un bajo nivel de tensión.

activo bajas?

3.3.2. Escritura de datos en la memoria FIFO

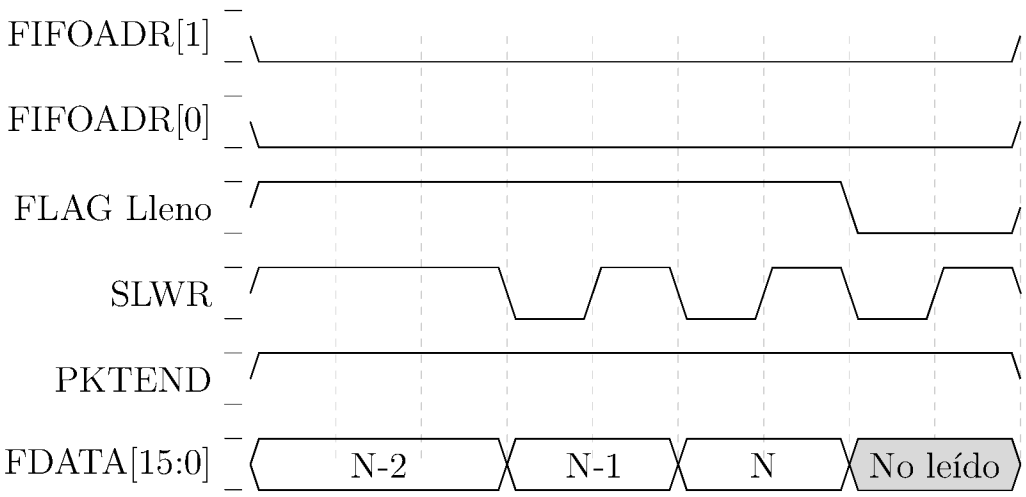


Figura 3.6: Diagrama temporal de la escritura de datos en la memoria FIFO desde un FPGA

Las señales que intervienen en el proceso de escritura de datos en la memoria FIFO, se encuentran detalladas en el diagrama temporal de la Figura 3.6. Para escribir datos en una memoria FIFO, el FPGA debe seleccionar la memoria a través de FIFOADR[1:0] en primer lugar. Para la configuración de este trabajo, esto es ″00″, correspondiente al EP2. Luego, se coloca en el bus de datos, donde se encuentran conectados los puertos FDATA[15:0], el dato a escribir. Se debe tener en cuenta que SLOE debe estar no asertivo, de modo tal que el bus FDATA[15:0] se encuentre en modo de alta impedancia por parte del controlador FX2LP y no interfiera con la escritura.

Una vez colocado el dato en el bus, se debe activar la señal SLWR. En el flanco positivo flanco asertivo de SLWR, el controlador incrementa el contador que indica la dirección de memoria en donde será almacenado el dato siguiente y deja guardado el dato que leyó en los puertos del bus FD. Como se observa en el diagrama de la Figura 3.6, SLWR es activo en bajo.

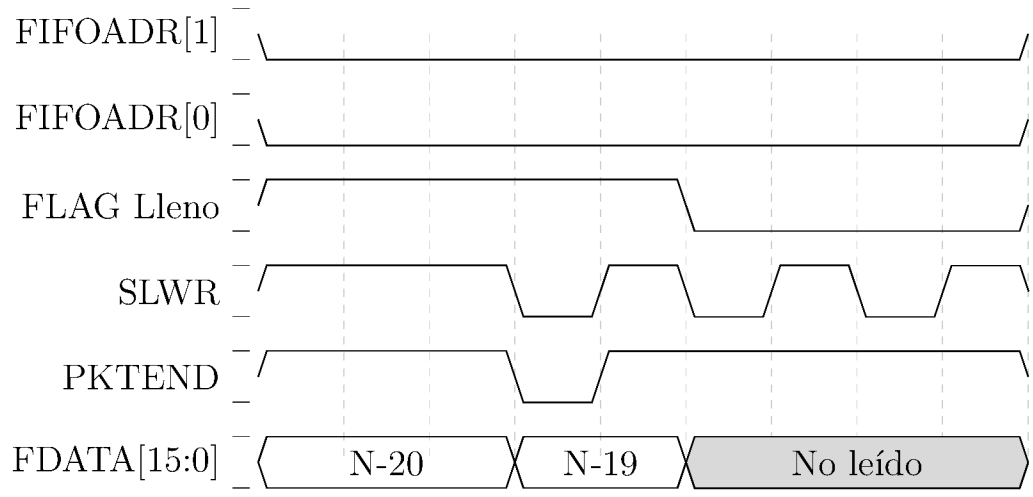


Figura 3.7: Diagrama temporal del funcionamiento del finalizado manual de mensajes

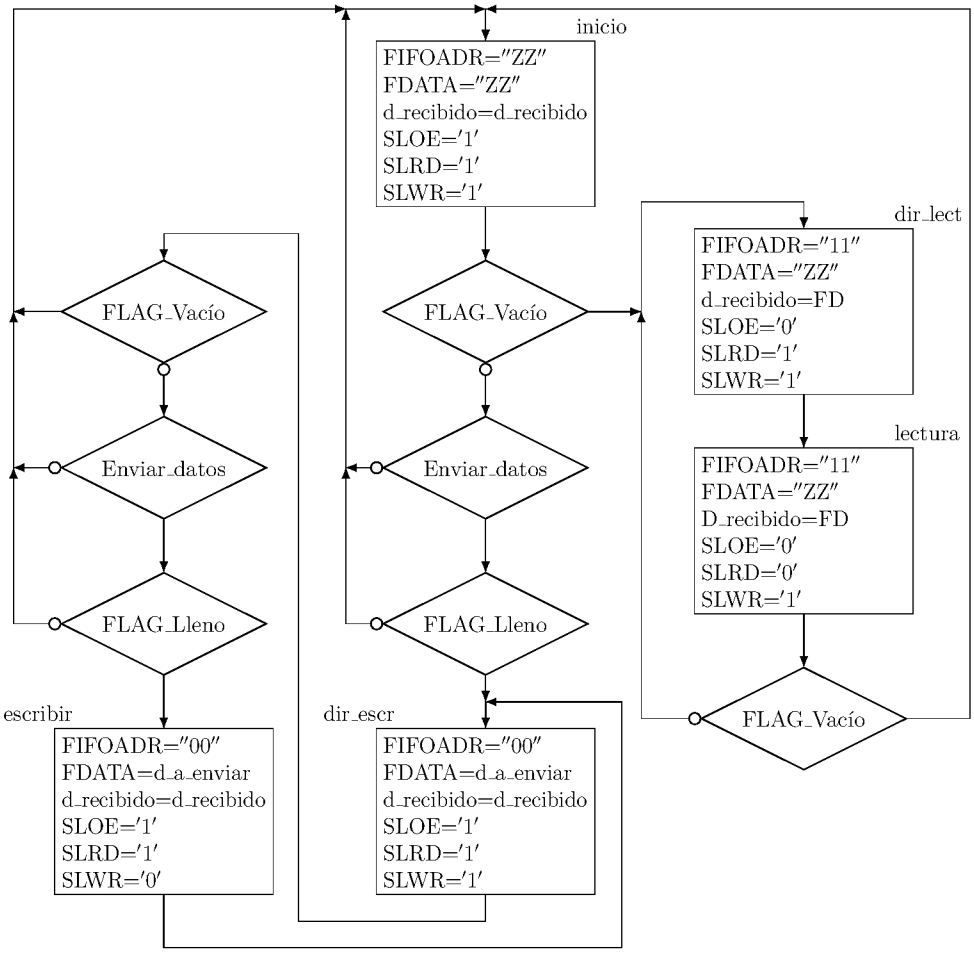


Figura 3.8: Diagrama de flujo de la máquina de estados desarrollada

La interfaz FX2LP espera siempre un número determinado de datos, señalizado como N en los diagramas de la Figura 3.6 y la Figura 3.7. Una vez alcanzado dicho número, el paquete queda listo para ser enviado cuando el host lo solicite. Sin embargo, puede ser enviado un número menor de datos en forma manual. Este funcionamiento es provisto a través de la señal PKTEND. Como se observa en la Figura 3.7, cuando PKTEND es asertiva (activa en bajo), la señal FLAG_Lleno se activa y la memoria FIFO ignora cualquier dato que se envíe a continuación.

3.3.3. Diagrama de Flujo de la MEF

Que se emplea para la comunicacion con..

Como se indica en la Sección 3.2, las señales de salida de la MEF que se diseña son, *FIFOADR*, *SLOE*, *SLRD*, *SLWR* y *PKTEND*. La bus de comunicación de datos hacia el interior del FPGA, *dato_recibido[15:0]* también es una salida del sistema desarrollado en este trabajo. Si bien *FDATA[15:0]* es un puerto de entrada y salida, se maneja también como puerto de salida, cuidando que, cuando funciona como puerto de entrada, se encuentre en alta impedancia el buffer que maneja la salida. Por su parte, los puertos de entrada son *FLAG_Vacío*, *Enviar_Datos* y *Flag_Lleno*.

Una consideración que se hizo en la implementación de este trabajo es que la lectura de las memorias FIFO es prioritaria con respecto a su escritura. Esto se basa en que se espera

que este desarrollo sirva de manera fundamental para la lectura de sensores. Dichos sensores serán configurados a través de los datos que lleguen al FPGA y, una vez configurados, deberán trasmitir los datos que adquieran del medio en que se encuentre. Se espera entonces, que la información que contienen los datos de configuración posea mayor importancia, ya que podría tener ordenes que detengan los sensores o cambien su funcionamiento. Así mismo, los datos deben ser enviados durante todo el tiempo que el sensor esté adquiriendo, por lo que se espera que los datos de entrada al FPGA sean menos proba bles que los datos que se envíen.

Así, se diseña la MEF que se observa en el diagrama de flujo de la Figura 3.8. En un estado inicial, todos las salidas se encuentran no asertivas. En el caso de salidas que se conectan a un bus (*FIFOADR[1:0]* y *FDATA[15:0]*) se colocan en estado de alta impedancia. El puerto *dato_recibido*, señalado en el diagrama de la Figura 3.8 como *d_recibido*, retiene su propio valor.

Cuando el *FLAG_Vacío* se activa, se procede a la operación de lectura. Si, en cambio, *FLAG_Vacío* es no asertivo, se debe conocer si el sistema genérico indica que envía datos, a través de *Enviar_datos*. Si esto ocurre, el sistema de comunicación debe corroborar que la memoria FIFO se encuentra en condiciones de recibir los datos, es decir, que no se encuentre *FLAG_Lleno* asertivo. Si estas condiciones ocurren, se debe proceder a la operación de escritura.

La operación de lectura coloca la dirección de la memoria FIFO relacionada al EP de salida (desde el Host), es decir ″00″. A su vez, se debe activar la salida de la memoria FIFO, activando la señal *SLOE*. El buffer de salida del bus *FDATA[15:0]* debe encontrarse en modo de alta impedancia, para no interferir con la lectura y un registro debe almacenar el valor que se indica en el buffer de entrada. El registro utilizado para almacenar la información leída es *d_recibido*. Luego, se debe activar la señal *SLRD*, lo que incrementa el dato de la memoria FIFO. De esta manera, se puede volver a leer la señal de *FLAG_Vacío* y determinar si se vuelve a implementar una operacion de lectura, o bien, se vuelve al inicio del programa.

Para efectuar la operación de escritura, en primer lugar se debe colocar la dirección de memoria FIFO que apunte al EP de entrada (hacia el Host). La dirección de la memoria FIFO en donde este trabajo escribe datos es ″11″. El bus de datos se conecta con el puerto interno *dato_a_enviar*, representado en el diagrama de la Figura 3.8 por *d_a_enviar*. Si las variables de entrada no se ven alteradas, el estado siguiente activará la señal *SLWR*, de forma tal que los datos colocados en el bus *FDATA* queden almacenados en la memoria FIFO. Luego, el estado siguiente desactiva *SLWR* y vuelve a consultar las variables de entrada.

3.3.4. Descripción de la Máquina de Estados en VHDL

Considerando las señales descriptas la Sección 3.2 y el diseño de la MEF cuyo diagrama de flujo se observa en la Figura 3.8, se procedió a describir el comportamiento del sistema en VHDL.

Conceptualmente, una MEF se compone tres partes, el estado actual, la función del próximo estado y la función de salida. Cada uno de estas partes puede ser descripta en VHDL todo junto en un mismo proceso, o bien en procesos separados. Este trabajo fue implementado mediante un proceso para la función de próximo estado y otro para actualizar el registro del estado actual. Las funciones de salida se implementaron mediante estados combinacionales.

De esta forma, se presenta a continuación la función de próximo estado. Para su mejor comprensión, se puede utilizar la Figura 3.9 en donde se observa una simplificación de cada uno de los estados del diagrama en bloques de la Figura 3.8, en donde se quitaron las variables de salida y se incorporó en cada uno de los estados el nombre con el que se lo asigna en el código

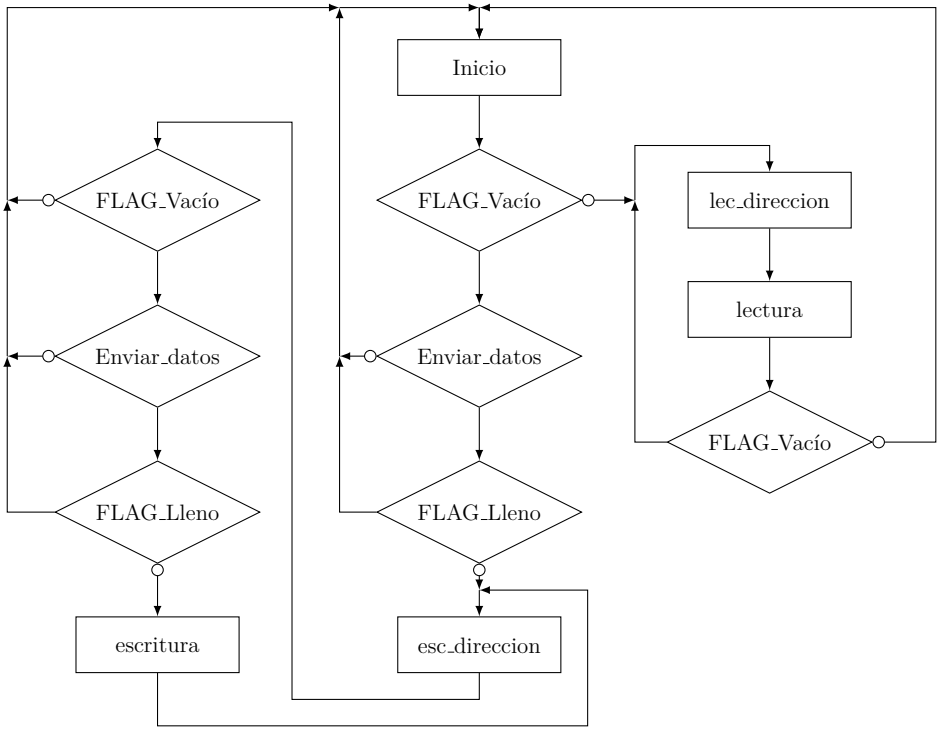


Figura 3.9: Diagrama de flujo de la máquina de estados desarrollada

VHDL desarrollado. Además, para facilitar la lectura del desarrollo, se colocaron las dos señales de entrada *FLAG_Vacío* y *FLAG_Lleno* como activos en alto.

```
architecture Behavioral of fx2lp_interfaz is
    -- maquina de estados de la interfaz
    type estados_mef is
    (
        inicio,
        lec_direccion, lectura,
        esc_direccion, escritura
    );
    signal estado_actual, prox_estado: estados_mea := inicio;
begin
    --implementacion de funcion de proximo estado
    proximo_estado: process(estado_actual, flag_lleno,
     flag_vacio, enviar_dato)
    begin
        case estado_actual is
            when inicio =>
                if flag_vacio = '0' then
                    prox_estado <= lec_direccion;
                elsif enviar_dato = '1' then
                    if flag_lleno = '0' then
                        prox_estado <= esc_direccion;
```

```
                    else
                        prox_estado <= inicio;
                    end if;
                else
                    prox_estado <= inicio;
                end if;

            when lec_direccion =>
                prox_estado <= lectura;

            when lectura =>
                if flag_vacio = '0' then
                    prox_estado <= lec_direccion;
                else
                    prox_estado <= inicio;
                end if;

            when esc_direccion =>
                prox_estado <= escritura;

            when escritura =>
                if enviar_dato = '1' then
                    if flag_vacio = '1' and flag_lleno = '0' then
                        prox_estado <= esc_direccion;
                    else
                        prox_estado <= inicio;
                    end if;
                else
                    prox_estado <= inicio;
                end if;

            when others =>
                prox_estado <= inicio;
        end case;
    end process proximo_estado;
end Behavioral;
```

Como se menciona anteriormente, las señales *FLAG_Vacío* y *FLAG_Lleno* se hicieron activos en alto. Sin embargo, las señales que toma la interfaz son activas en bajo. Entonces, se asignaron las señales mencionadas a los puertos *flaga* y *flagb* mediante un inversor. Todo esto apuntó a facilitar la lectura y el desarrollo de la descripción.

```
architecture Behavioral of fx2lp_interfaz is
    signal flag_vacio: std_logic;
    signal flag_lleno: std_logic;
begin
    flag_lleno  <= not flaga;
```

```
    flag_vacio <= not flagb;
end Behavioral;
```

La función de salida se implementa con lógica combinacional, utilizando el registro de estado actual. Debido a esto, se describe cada una de las salidas por separado. Como lo que modifica estas salidas son los estados de la MEF definidos, se debe recurrir a señales que sirvan como conectores internos desde los puertos hacia los diferentes componentes que se describen.

```
architecture Behavioral of fx2lp_interfaz is
    signal slwr_int:     std_logic := '1';
    signal slrd_int:     std_logic := '1';
    signal sloe_int:     std_logic := '1';
    signal pktend_int:   std_logic := '1';
    signal faddr_int:    std_logic_vector(1 downto 0) := "ZZ";
    signal fdata_sal:    std_logic_vector(port_width-1 downto 0);
    signal fdata_inent:  std_logic_vector(port_width-1 downto 0);
    signal reloj_sitema: std_logic;
begin
    reloj_sistema <= reloj;
    slwr   <= slwr_int;
    slrd   <= slrd_int;
    sloe   <= sloe_int;
    faddr  <= faddr_int;
    pktend <= pktend_int;
    d_recibido <= fdata_ent;
    fdata_sal <= d_a_enviar;

end Behavioral
```

Con todas las señales definidas y asignadas, y la máquina de estados que se detalló anteriormente, se pueden asignar las señales de salida:

```
architecture Behavioral of fx2lp_interfaz is
    with estado_actual select
        faddr_int <=    out_ep_addr when lec_direccion | lectura,
                        in_ep_addr  when esc_direccion | escritura,
                        (others => 'Z') when others;

    slwr_int <= '0' when prox_estado = esc_direccion else
                '1';

    slrd_int <= '0' when estado_actual = lec_direccion else
    '1';

    pktend_int <= ((not falg_vacio) or enviar_dato);

    with estado_actual select
    sloe_int <= '0' when lectura | lec_direccion,
```

```
                '1' when others;

    with estado_actual select
        fdata <=    fdata_sal       when escritura | esc_direccion,
                    (others => 'Z') when others;

    with estado_actual select
        fdata_ent <=    fdata     when lectura | lec_direccion,
                        fdata_ent  when others;
end Behavioral
```

Finalmente, resta el reloj que hace avanzar la MAE. A este reloj, se le acoplan dos temporizadores de habilitación. Esto se debe a que se espera que el sistema trabaje a 50 MHz. Sin embargo, para respetar los tiempos de establecimiento y ancho de pulso de las distintas señales[32], cuando el próximo estado es esc_dirección se deben esperar tres ciclos de reloj y en el caso de que el próximo estado sea escritura, lec_direccion o lectura, se debe esperar dos ciclos de reloj. Esto se implementa con dos contadores diferentes, los cuales habilitan o no el cambio de estado. Esto se detalla a continuación:

```
architecture Behavioral of fx2lp_interfaz is
    signal cont3:    natural range 0 to 4 := 0;
    signal cont2:    natural range 0 to 3 := 0;
    signal disparo3: std_logic := '0';
    signal disparo2: std_logic := '0';
begin
    contador3: process(reloj_sistema, reset, disparo3)
    begin
        if reset = '0' then
            cont3 <= 0;
        elsif rising_edge(reloj_sistema) then
            if cont3 > 0 then
                cont3 <= cont3 - 1;
            elsif disparo3 = '1' then
                cont3 <= 4;
            end if;
        end if;
    end process contador3;

    disparo3 <= '1' when (prox_estado = esc_direccion) else '0';

    counter2: process(reloj_sistema, reset, disparo2)
    begin
        if reset = '0' then
            cont2 <= 0;
        elsif rising_edge(reloj_sistema)then
            if cont2 > 0 then
                cont2 <= count2 - 1;
```

```
            elsif disparo2 = '1' then
                cont2 <= 3;
            end if;
        end if;
    end process contador2;

    with prox_estado select
    disparo2 <= '1' when lec_direccion | lectura | esc_direccion,
                '0' when others;

    reloj_mea: process (reloj_sistema, reset)
    begin
        if reset = '0' then
                estado_actual <= idle;
        elsif rising_edge(reloj_sistema) then
            if cont2 = 0 and cont3 = 0 then
                estado_actual <= prox_estado;
            end if;
        end if;
    end process reloj_mea;
end Behavioral
```

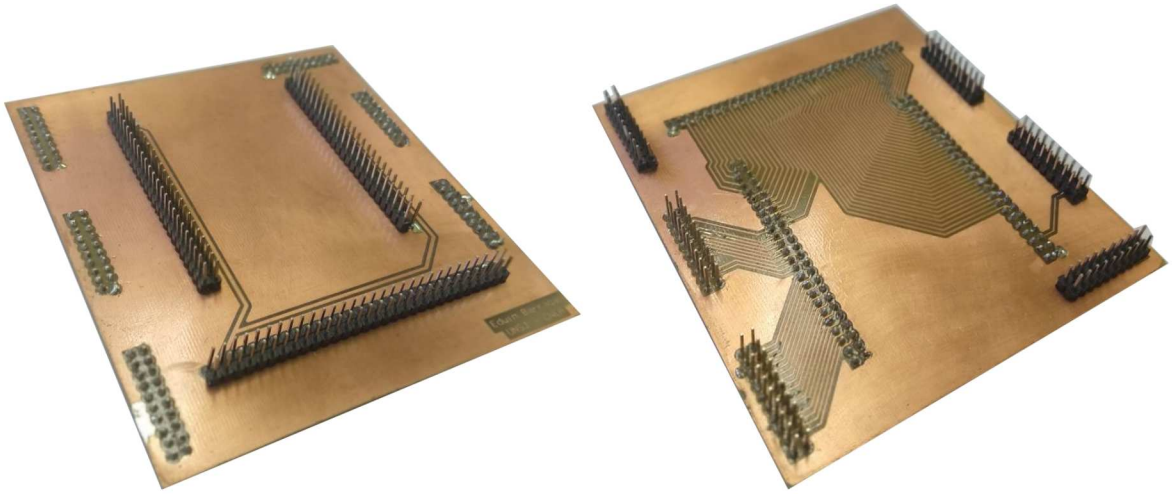
El código completo se puede encontrar en el Anexo **??**

3.4. Placa de Interconexión

Los circuitos integrados utilizados para la implementación de la comunicación USB, estos son el controlador FX2LP de Cypress y el FPGA Spartan VI de Xilinx, vienen incorporados en sendas placas de desarrollo. Para la conexión eléctrica de estos dos chips, se desarrolló una placa de interconexión, es decir, un circuito impreso (PCB, del ingles *Printed Circuit Board*) que conecta en forma eléctrica dos o más dispositivos. Esto brinda una conexión mucho más robusta y prolija que si fuese realizada mediante cables cintas o alambres individuales.

El desarrollo de la placa de interconexión, necesitó de tres versiones para obtener un correcto funcionamiento. La versión número 1, la cual se observa en la Figura 3.10, presenta un problema de contacto eléctrico entre sus dos caras conductoras, debido a no se contaba con la tecnología suficiente para realizar la metalización de los agujeros que conducen la señal de un lado al otro del circuito impreso durante el proceso de fabricación. Por otro lado, en la etapa de montaje, el alumno confunde los pines que debe ser colocados, ya que el reverso necesita pines hembra en lugar de pines macho.

Se realiza una segunda versión, la que se observa en la Figura 3.11. A este PCB se le incorporan vías pasantes para poder conectar las distintas pistas que recorren el circuito mediante la soldadura de alambres, lo que soluciona el problema de conexión eléctrica. En esta placa se tiene mayor cuidado en la etapa de montaje de los pines. Sin embargo, durante la revisión de los pines se encuentra un defecto en el puerto asignado a la señal de reloj, la cual posee un terminal en un pin que no se encuentra disponible. Esto obliga a la implementación de la comunicación del presente trabajo de forma asíncrona.



Anverso Reverso

Figura 3.10: Versión 1 de la placa de interconexión

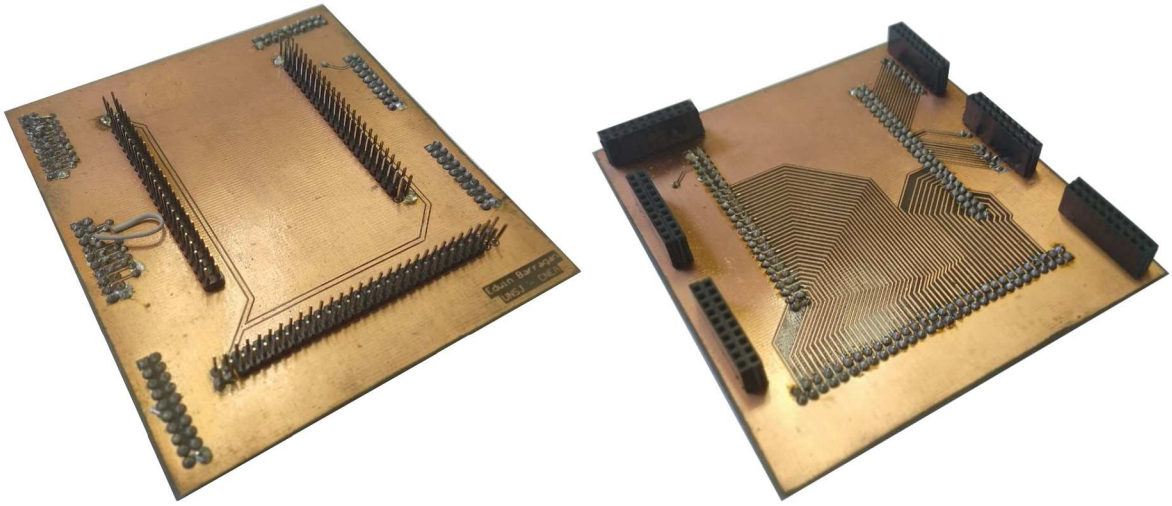
Otro defecto que presenta la versión 2 de la placa de interconexión es la inexistencia de un punto para soldadura, que se ocasiona al momento del perforado del impreso. Esto obliga a realizar una conexión mediante un pequeño cable, el cual se puede observar en el anverso de la Figura 3.11.

Finalmente, se decide rehacer el circuito de conexión en una tercera versión y solicitar su fabricación en una empresa especializada en la manufactura de PCB para prototipo, radicada en China. En esta versión, fue redirigida la línea de conexión defectuosa, lo que permite conectar las fuentes de reloj e implementar una comunicación síncrónica. Esto no fue realizado al momento de la escritura del presente informe, aunque se espera su implementación en trabajos futuros. Además, debido a restricciones en el proceso de fabricación, se debe redimensionar el PCB y hacerlo más compacto.

Gracias a la mejora que brinda la empresa en el proceso de fabricación, se eliminan las conexiones entre las dos caras del impreso mediante la soldadura de alambre y se cambian por agujeros metalizados. Además, se incorporaron conexiones adicionales entre el controlador y el FPGA. Esto permite utilizar el μC 8051 incorporado al controlador FX2LP para realizar tareas adicionales, junto al FPGA. Se adjunta en el Anexo **??** un plano esquemático con las conexiones de la versión 3 del circuito impreso. elaborado.

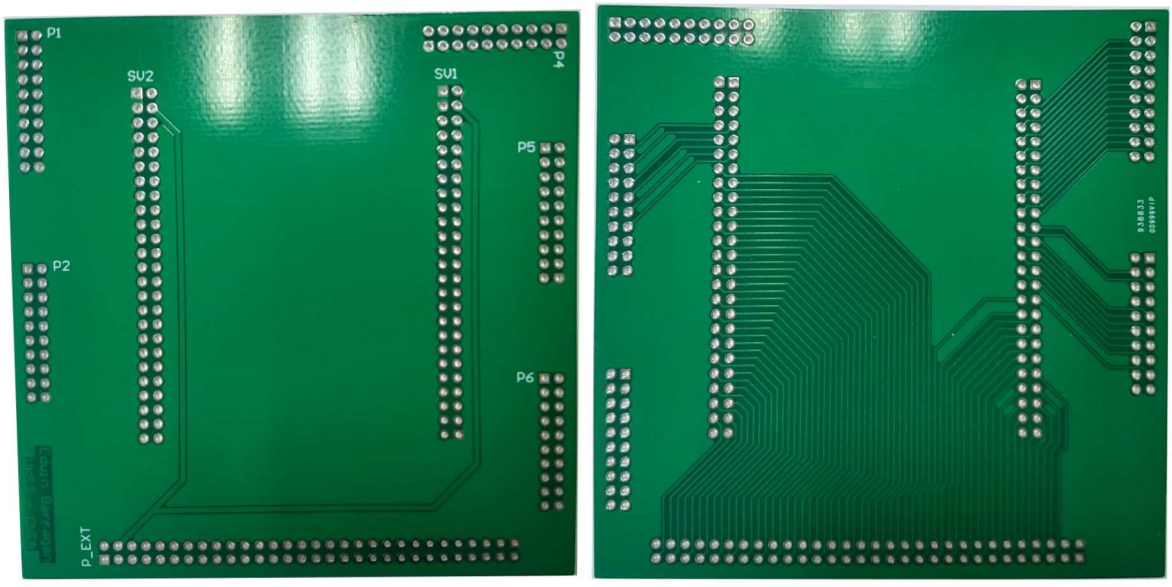
3.5. Sumario del capítulo

Durante el presente capítulo se desarrolló cuales son las señales de control que intervienen en las operaciones de lectura y escritura externa en las memorias FIFO. En base a ellas se explicó el diseño de la máquina de estados finitos y su descripción en lenguaje VHDL, para su posterior síntesis en FPGA. Luego se detalló la placa de interconexión realizada para la conexión eléctrica de las placas de desarrollo que contienen al controlador FX2LP y al FPGA Spartan VI



Anverso Reverso

Figura 3.11: Versión 2 de la placa de interconexión



Anverso Reverso

Figura 3.12: Versión 3 de la placa de interconexión

Capítulo 4

Verificación y validación del sistema

4.1. Depuración de la configuración de la interfaz FX2LP

4.2. Pruebas de la síntesis de la MEA en el FPGA

4.3. Pruebas de la comunicación entre el FPGA y la PC

4.3.1. Elección de la biblioteca libusb-1.0

La tercer parte en la que se divide el trabajo es relativa a la comunicación entre la interfaz y una PC. Ya que la interfaz se encarga en gran medida de lo relativo al empaquetamiento, codificación y decodificación y que las PC, por su parte, vienen equipadas con el hardware necesario, este trabajo debe implementar el software que comande y gestione, desde el sistema operativo el correcto acceso a los datos que se envían y reciben. Para la elaboración de software que permita el manejo de los puertos USB, se utiliza la biblioteca `libusb`.

`libusb` es una biblioteca de código abierto, muy bien documentada, escrita en C, que brinda acceso genérico a dispositivos USB. Las características de diseño que persigue el equipo de desarrollo que mantiene la biblioteca es que sea multiplataforma, modo usuario y agnóstico de versión[33].

- Multiplataforma: Se apunta a que cualquier software que contenga esta biblioteca pueda ser compilado y ejecutado en la mayor cantidad de plataformas posibles, dotando al software de portabilidad, es decir, esta biblioteca puede ser ejecutada en Windows, Linux, OS X, Android y otras plataformas sin necesidad de realizar cambios en el código.

- Modo usuario: No se requiere acceso privilegiado de ningún tipo para poder ejecutar programas escritos con esta biblioteca.

- Agnótisco de versión: Sin importar la versión de la norma USB que se utilice, el programa se podrá comunicar siempre con el dispositivo USB que se requiera.

La biblioteca `libusb` no posee un autor formal. Es decir, no hay una persona, empresa u organización formal que se encargue de la creación y el mantenimiento del software. Existe una comunidad de más de 130 desarrolladores que en forma voluntaria cooperan en el mantenimiento

y desarrollo de esta biblioteca. Se garantiza así que el proyecto esté documentado en forma detallada, existiendo amplios ejemplos y tutoriales de su uso.

Se elige esta biblioteca para la realización del software que gestionara el envío y la recepción de datos debido a su amplio soporte, la factibilidad de ejecutarlo en diferentes sistemas operativos y por ser totalmente gratuito.

4.4. Resultados

4.5. Conclusiones

4.6. Trabajos Futuros

Bibliografía

[1] R. Pallàs-Areny and J. G. Webster, *Sensors and signal conditioning*. Wiley-Interscience, 2001.

[2] D. M. Considine, *Encyclopedia of instrumentation and control*. McGraw-Hill, Inc., 1971.

[3] A. Perez Garcia, "Curso de instrumentación," p. 261, 2008.

[4] J. Fraden, *Handbook of modern sensors: physics, designs, and applications*. New York, NY: Springer New York, 2010.

[5] E. Slawiñski and V. Mut, *Humanos y máquinas inteligentes: conocimiento educativo sobre el comportamiento interno de robots que actúan juno y para el hombre*. Saarbrücken, Alemania: Editorial Académica Española, 2011.

[6] K. Ogata, *Modern control engineering*. Aeeizh, 2002.

[7] G. Binnig and H. Rohrer, "Scanning tunneling microscopy," *Surface Science*, vol. 126, pp. 236–244, mar 1983.

[8] R. Turchetta, K. R. Spring, and M. W. Davidson, "Digital Imaging in Optical Microscopy - Introduction to CMOS Image Sensors," (accessed in July 2019).

[9] S. Mendis, S. Kemeny, and E. Fossum, "CMOS active pixel image sensor," *IEEE Transactions on Electron Devices*, vol. 41, pp. 452–453, mar 1994.

[10] C. Hu-Guo, J. Baudot, G. Bertolone, A. Besson, A. S. Brogna, C. Colledani, G. Claus, R. D. Masi, Y. Degerli, A. Dorokhov, G. Doziere, W. Dulinski, X. Fang, M. Gelin, M. Goffe, F. Guilloux, A. Himmi, K. Jaaskelainen, M. Koziel, F. Morel, F. Orsini, M. Specht, Q. Sun, I. Valin, and M. Winter, "CMOS pixel sensor development: a fast read-out architecture with integrated zero suppression," *Journal of Instrumentation*, vol. 4, pp. P04012–P04012, apr 2009.

[11] J. Baudot, G. Bertolone, A. Brogna, G. Claus, C. Colledani, Y. Değerli, R. De Masi, A. Dorokhov, G. Dozière, W. Dulinski, M. Gelin, M. Goffe, A. Himmi, F. Guilloux, C. Hu-Guo, K. Jaaskelainen, M. Koziel, F. Morel, F. Orsini, M. Specht, I. Valin, G. Voutsinas, and M. Winter, "First test results of MIMOSA-26, a fast CMOS sensor with integrated zero suppression and digitized output," *IEEE Nuclear Science Symposium Conference Record*, pp. 1169–1173, 2009.

[12] M. Pérez, J. Lipovetzky, M. Sofo Haro, I. Sidelnik, J. J. Blostein, F. Alcalde Bessia, and M. G. Berisso, "Particle detection and classification using commercial off the shelf CMOS image sensors," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 827, pp. 171–180, aug 2016.

[13] M. Pérez, J. J. Blostein, F. A. Bessia, A. Tartaglione, I. Sidelnik, M. S. Haro, S. Suárez, M. L. Gimenez, M. G. Berisso, and J. Lipovetzky, "Thermal neutron detector based on COTS CMOS imagers and a conversion layer containing Gadolinium," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 893, pp. 157–163, jun 2018.

[14] C. L. Galimberti, F. Alcalde Bessia, M. Perez, M. G. Berisso, M. Sofo Haro, I. Sidelnik, J. Blostein, H. Asorey, and J. Lipovetzky, "A Low Cost Environmental Ionizing Radiation Detector Based on COTS CMOS Image Sensors," in *2018 IEEE Biennial Congress of Argentina (ARGENCON)*, pp. 1–6, IEEE, jun 2018.

[15] T. Hizawa, J. Matsuo, T. Ishida, H. Takao, H. Abe, K. Sawada, and M. Ishida, "32 × 32 pH image sensors for real time observation of biochemical phenomena," *TRANSDUCERS and EUROSENSORS '07 - 4th International Conference on Solid-State Sensors, Actuators and Microsystems*, pp. 1311–1312, 2007.

[16] ON Semiconductor, "NOIP1SN0300A Global Shutter CMOS Image Sensors," 2014.

[17] N. Ida, *Engineering Electromagnetics*. Cham: Springer International Publishing, 3th ed., 2015.

[18] J. F. Wakerly, *Digital Design: principles and practices*, vol. 1. Pearson, 1999.

[19] M. Perez, F. Alcalde, M. S. Haro, I. Sidelnik, J. J. Blostein, M. G. Berisso, and J. Lipovetzky, "Implementation of an ionizing radiation detector based on a FPGA-controlled COTS CMOS image sensor," in *2017 XVII Workshop on Information Processing and Control (RPIC)*, pp. 1–6, IEEE, sep 2017.

[20] R. Biswas, *An Embedded Solution for JPEG 2000 Image Compression Based Back-end for Ultrasonography System*. PhD thesis, IIT, Kharagpur, 2018.

[21] T. Yanagisawa, T. Ikenaga, Y. Sugimoto, K. Kawatsu, M. Yoshikawa, S.-i. Okumura, and T. Ito, "New NEO search technology using small telescopes and FPGA," in *2018 IEEE Aerospace Conference*, vol. 2018-March, pp. 1–7, IEEE, mar 2018.

[22] H. H. Goldstine and A. Goldstine, "The Electronic Numerical Integrator and Computer (ENIAC)," *Mathematical Tables and Other Aids to Computation*, vol. 2, p. 97, jul 1946.

[23] S. of Cable Telecommuniocations Engineers, *American National Standard ANSI/SCTE 07 2006. Digital Tansmission Standard for Cable Television*. Society of Cable Telecommunications Engineers, Inc., 2006.

[24] I. Micron Technology, "1 / 2-Inch Megapixel CMOS Digital Image Sensor MT9M001C12STM (Monochrome)," pp. 1–35, 2004.

[25] IEEE Computer Society, *IEEE Standard for Ethernet*, vol. 2018. 2018.

[26] IEEE Computer Society, *Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications IEEE Computer Society Specific requirements Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, vol. 2012. 2016.

[27] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, and Philips, *Universal Serial Bus Specification*, vol. Revision 2.0. 2000.

[28] "Usb hardware." `https://en.wikipedia.org/wiki/USB_hardware`. Ingreso: 8 de agosto del 2019.

[29] T. Riihonen, *Desing and analysis of duplexing Modes and Forwarding Protocols for OFDM(A) Relay Links*. PhD thesis, 2015.

[30] B. Sklar, *Digital communications: fundamentals and applications*. 2001.

[31] Cypress Semiconductor, "EZ-USB ® Technical Reference Manual," tech. rep., 2014.

[32] Cypress, "CY7C68013A, CY7C68014A, CY7C68015A, CY7C68016A, EZ-USB(R) FX2LP(TM) USB Microcontroller High-Speed USB Peripheral Controller," 2017.

[33] libusb, "libusb 1.0 https://libusb.info/ - acceso: 04/11/2019."