

Capítulo 3

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

Unir
parrafos

de Cypress

Un sistema de comunicación?

En el Capítulo anterior, se ha descripto una comunicación que intercambia datos entre una PC y el controlador FX2LP a través del protocolo USB. Dicha comunicación constituye una primer etapa del desarrollo realizado.

En una segunda etapa, se debe lograr la comunicación de datos entre un FPGA y el controlador FX2LP. Con este enlace, los datos estarían en condiciones de fluir desde la PC al FPGA, a través de la interfaz. Los datos son transferidos a través de las memorias FIFO descriptas en el Capítulo 2. Yo aclararía para que es necesario

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 e implementar dentro de este dispositivo un módulo que sea capaz de interactuar con las memorias FIFO del controlador FX2LP.

El módulo que se implementó en el FPGA es una pequeña Máquina de Estados Finitos, a través de la cual, se leen las señales que provienen de la interfaz y se generan las señales necesarias para comandar su memoria FIFO.

A continuación, se justifica la elección del FPGA y la placa de desarrollo utilizados. También se detallan las señales que intervienen en el funcionamiento de la interfaz y los protocolos de lectura y escritura de modo asíncrono.

Los mecanismos que se deben seguir para las operaciones de intercambio de datos dan lugar a la elaboración de la máquina de estados se sintetiza en el FPGA. Se utiliza VHDL como lenguaje de descripción para elaborar el código que implementa de la máquina de estados en el FPGA se utiliza el lenguaje de descripción de hardware VHDL.

Además, se explica el desarrollo de un circuito impreso utilizado para la interconexión entre las distintas placas de desarrollo que se utilizan en este trabajo.

Yo uniría estos párrafos y detallaría en que sección se explica cada cosa

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. Así, no solo se facilita el

envío y la recepción del mensaje, sino también que determina a cada dispositivo los procedimientos que efectúa. Por este motivo, una vez definido que se utiliza una interfaz intermedia entre la PC y un FPGA (Capítulo 1), y que dicha interfaz es el circuito integrado EZ-USB FX2LP de Cypress (Capítulo 2), se determina cuál es el protocolo a través del cual se comunica cada uno de los dispositivos y se puede 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 diseñadas y comercializadas por la empresa Xilinx Inc. La placa Spartan-3E Starter, comercializada por Digilent es una placa que posee ~~que incorpora~~ 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. placa

Por su parte, la Nexys 3, tiene como núcleo un FPGA Spartan-6, es decir, un FPGA tecnológicamente superior a los FPGA Spartan-3. El FPGA Spartan 6 brinda mayor cantidad de bloques lógicos programables, el proceso CMOS en el que esta fabricada es mas moderno

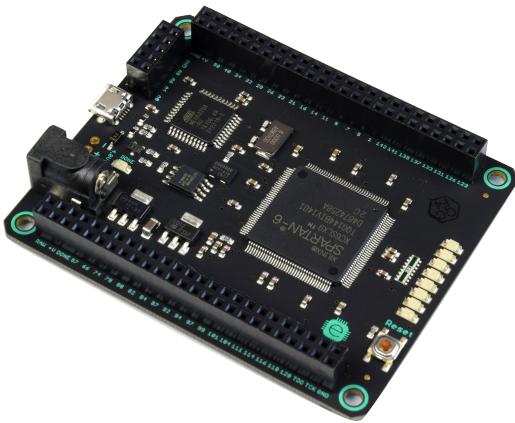
que el de la Spartan3 por lo que sus transistores son mas pequeños La miniaturización de los transistores, a su vez, otorga la posibilidad de una mayor velocidad de operación. La placa Nexys 3 también posee una gran gama de periféricos tales como pulsadores, interruptores, displays led de 7 segmentos, diferentes tipos de memorias, CODEC para comunicarse por Ethernet 10/100 o USB 2.0.

A diferencia de las anteriores, la placa de desarrollo Mojo v3 comercializada por la empresa Alchitry, es una placa de prototipado rápido. Esto quiere decir que en lugar de poseer una gran cantidad y variedad de periféricos, se dota a la placa de una gran cantidad de puertos para que el usuario pueda colocar los periféricos que desea. Al igual que la Nexys 3, cuenta con un FPGA Spartan 6 de Xilinx. Dispone 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 debido en gran medida a la ausencia de periféricos que, dependiendo la aplicación, pueden ser innecesarios. esto no lo entiendo

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 in Por otra parte, la placa Mojo 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 3.1, 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 circuitos 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



Esto me sigue sin quedar del todo claro: Se usan los conversores del micro para digitalizar las señales analógicas. ¿Como se trasmiten los datos a la FPGA?

Figura 3.1: Placa de prototipado rápido MOJO v3, diseñada por Embedded Micro

configuración del FPGA (cargada desde una memoria flash incorporada, o trasmisita 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. Las entradas analógicas que posee esta placa de desarrollo también son leídas a través del microcontrolador ATmega32U4, luego de que el FPGA es programado.

Es importante aclarar que si bien el sistema posee alguna forma de comunicación USB utilizando el microcontrolador ATmega32U4 como interfaz, el enlace que forma posee un menor ancho de banda que el sistema desarrollado en el presente trabajo. Esto se debe a que la línea de controladores ATmega incorpora puertos USB 2.0 full-speed, esto quiere decir que puede enviar datos a una tasa de 12 Mbit s^{-1} . Además, la comunicación entre ambos chips se realiza vía SPI (*Serial Peripheral Interface*, 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.

3.2. Señales de comunicación de la Máquina de Estados

Finitos

Yo pondria:

En esta sección se presentaran las señales de comunicación necesarias para efectuar una transmisión de datos entre la FPGA y el controlador FX2LP. Para realizar dicha transmisión se debe realizar la lectura y escritura de datos en la memoria FIFO del controlador mediante una secuencia de pasos (también llamada protocolo). Esta tarea fue llevada a cabo dentro del FPGA, con un bloque especialmente diseñado para este propósito. Este bloque contiene una máquina de estados finita que sigue la secuencia de pasos para efectuar el protocolo de lectura/escritura de la memoria FIFO. A continuación... explicar que vas a mostrar

Finitos (MEF). Por esto, a continuación se detallara la metodología a través de la cual se efectúan las operaciones de lectura y escritura en la memoria FIFO, y la comunicación interna de los diferentes módulos del FPGA, a fin de identificar las señales de entrada y de salida que dan lugar al programa que sigue la MEF diseñada en su funcionamiento.

no me queda claro que queres explicar aca

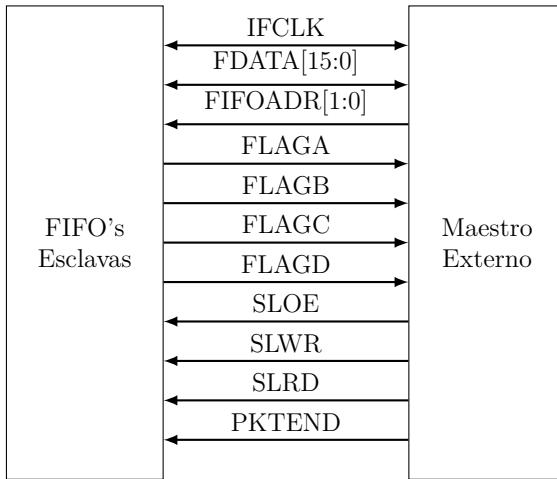


Figura 3.2: Señales de la interfaz entre las FIFO's y un maestro externo

3.2.1. Señales de comunicación el FPGA y el controlador FX2LP

La Figura 3.2 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. Segundo se programe, este puede ser de 8 o 16 bits, en forma independiente para cada EP.
- **FIFOADDR[1:0]:** puerto de direcciones. A través de él se selecciona la memoria activa en el bus.
- **FLAGx:** Los cuatro puertos de flag indican el estado de las memorias y son configurables.
- **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 por el μ C 8051 incorporado en el circuito integrado FX2LP. Las memorias FIFO tienen dirección binaria "00", "01", "10" y "11" en los puertos FIFOADR[1:0]. Se muestra en la Tabla 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 equivalen a niveles de tensión bajo y alto, respectivamente.

Los puertos que indican el estado de las memorias son programables. Pueden indicar si la memoria se encuentra llena o vacía. También es posible que señalen que se alcanzó una

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

Tabla 3.1: Direcciones de selección de memoria activa

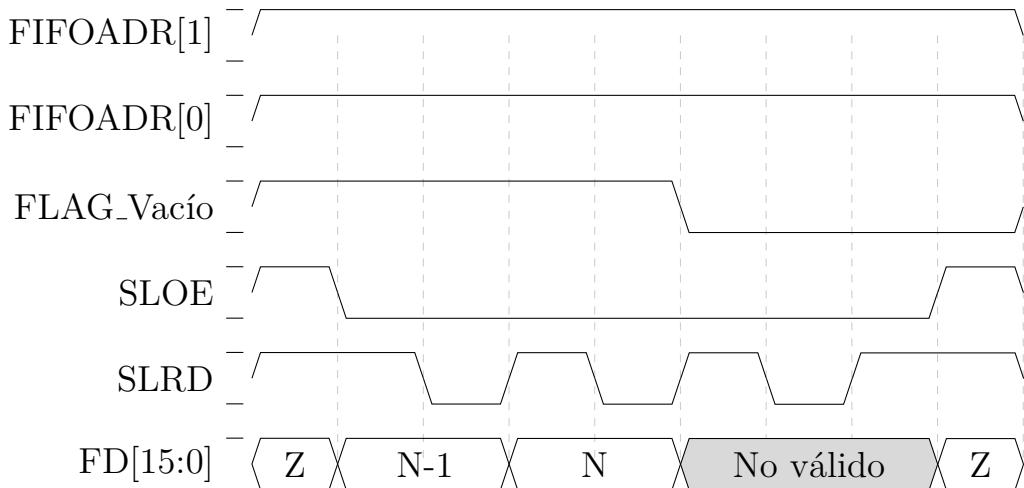


Figura 3.3: Diagrama temporal de la lectura de datos desde la memoria FIFO por un FPGA

cantidad de datos superior a un umbral programable. Según la configuración que el usuario realice, estarán asociadas a una memoria específica o a la memoria activa, seleccionada a través de FIFOADDR[1:0].

La configuración que se implementó en este trabajo, como ya se mencionó 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.

Se debe destacar que todas las señales que emite la memoria FIFO del controlador FX2LP son activas en bajo. Esto quiere decir que, por ejemplo, si la señal FLAGA posee un Cero logico? tensión bajo (cerca a 0 V), el espacio de memoria destinado al EP2 se encuentra lleno. Por su parte, si el valor es un uno logico? no a la tensión de alimentación del controlador (3.3 V), la memoria aún posee espacio para el almacenamiento de datos.

El que lea esta tesis ya va a saber de digitales
creo que si decís que son activo bajas y hablas
de niveles logicos ya se va a entender, podés
usar cero/uno logico o nivel bajo o alto

Lectura de datos desde la memoria FIFO

Para efectuar operaciones de lectura en régimen asíncrono, como se muestra en la Figura 3.3, 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 planteada en este trabajo, "11", la que corresponde al EP8. Luego, debe ser activada la señal SLOE, la cual coloca en los puertos FD[15:0] los datos almacenados en la memoria FIFO que fue activada mediante la señal.... El dato disponible en la salida de la memoria FIFO siempre será el más antiguo, es decir, el que se almacenó antes. En el flanco negativo de la señal SLRD, la memoria FIFO aumenta un

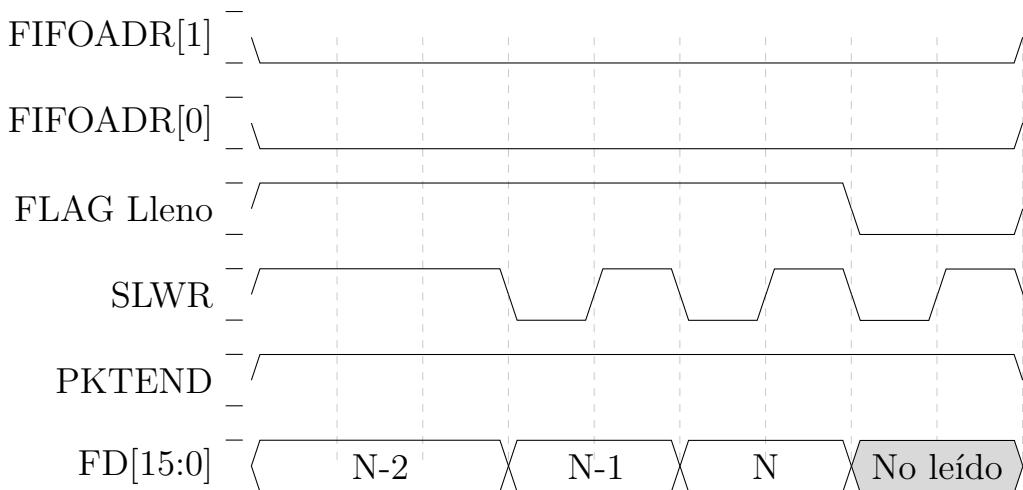


Figura 3.4: Diagrama temporal de la escritura de datos en la memoria FIFO desde un FPGA

contador que selecciona la dirección del próximo dato, y coloca este dato en el puerto FD[15:0].

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 FD[15:0] permanece en estado de alta impedancia para dejar el bus disponible a otros dispositivos.

Escritura de datos en la memoria FIFO

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 3.4. En primer lugar, el FPGA debe seleccionar la memoria FIFO, el FX2LP debe seleccionar la memoria a través de FIFOADR[1:0] en primer lugar. [En el sistema desarrollado en este trabajo se utiliza....] Luego, se coloca en el bus de datos, donde se encuentran conectados los puertos FD[15:0] del dato a escribir. Es importante aclarar que SLOE no debe estar activo, de modo tal que el bus FD[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 flanko negativo 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.

La interfaz FX2LP espera siempre un número determinado de datos, señalizado como N en los diagramas de la Figura 3.4 y la Figura 3.5. 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 3.5, cuando se activa PKTEND, también lo hace la señal FLAG_Lleno y la memoria FIFO ignora cualquier dato que se envíe a continuación.

3.2.2. Comunicación interna del FPGA

La MEF desarrollada es un nexo entre el controlador FX2LP y el FPGA. Por ello, la MEF yo pondria algo asi:

La máquina de estados finita (MEF) desarrollada es el nexo entre el controlador FX2LP y la FPGA, por este motivo debe ser capaz de proveer datos de lectura/escritura en ambas direcciones.

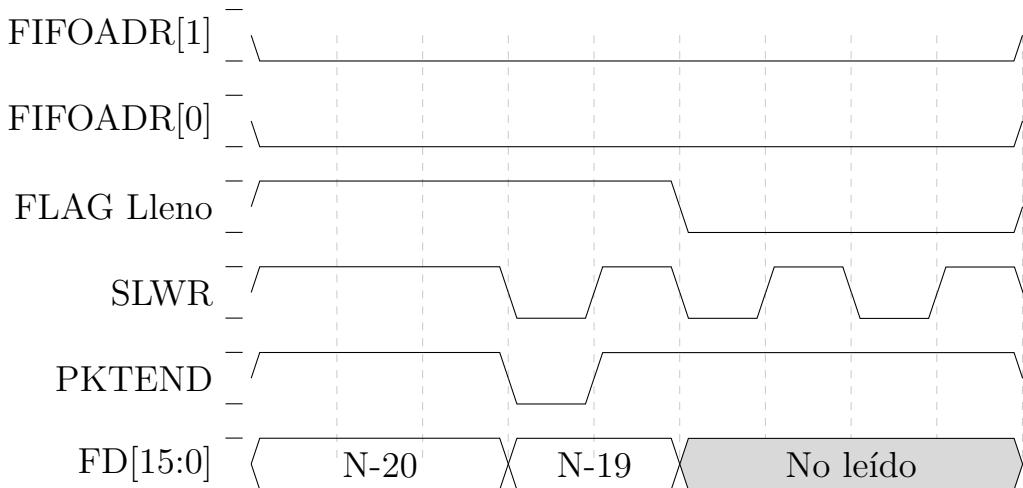


Figura 3.5: Diagrama temporal del funcionamiento del finalizado manual de mensajes

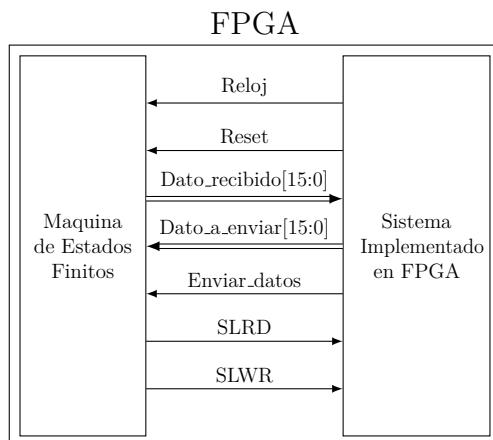


Figura 3.6: Diagrama de las señales que interconectan los módulos dentro del FPGA

inverso, poder escribir en la interfaz FX2LP los datos que le suministre un sistema sintetizado en el FPGA cuando lo indique. En adición, el FPGA debe proveerle al sistema una señal de reloj a fin de que la MEF tenga un funcionamiento sincronizado con el sistema.

La Figura 3.6 muestra un diagrama en bloques en donde se detalla cuáles son las señales internas a través de las cuales se comunican los distintos módulos que se implementan en el FPGA. La MEF desarrollada posee entrada y salida de datos independientes *Dato_a_enviar* y *Dato_Recibido*, entrada de reloj y reset que será suministrada por el FPGA, como así también ~~de~~ una señal que controla el envío de datos (*Enviar_Datos*)

Para indicar al FPGA que los datos a enviar fueron enviados, la MEF posee como salida *SLWR*. Para indicar que leyó datos de la interfaz FX2LP, se utiliza la señal *SLRD*. Se debe notar que ambas señales son las mismas a través de las cuales la MEF se comunica con la memoria FIFO del controlador.

A ver si entiendo bien:

De forma similar a la interfaz utilizada por la memoria FIFO del controlador FX.. la maquina de estados finita posee las señales *SLWR* y *SLRD* que le indican a los demás modulos implementados en la FPGA que los datos presentes en el bus fueron enviados/leidos.

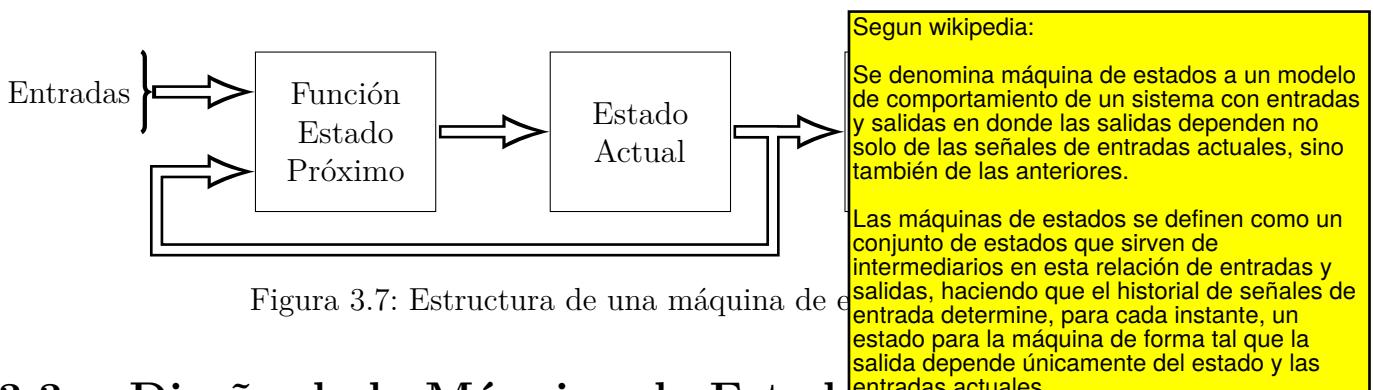


Figura 3.7: Estructura de una máquina de estados.

3.3. Diseño de la Máquina de Estados

No entiendo esta oracion

Una Máquina de Estados Finitos (MEF) es un tipo de dispositivo lógico secuencial que puede estar en uno de un conjunto finito de elementos. Tal como se observa en la Figura 3.7, el estado siguiente que adoptará la MEF será función de una combinación lógica entre el estado actual Yo sacaria esto, el que lea este trabajo ya sabe lo que es una maquina de estados y tambien sabe las diferencias entre Mealy y Moore. configurando la denominada máquina de Moore, aunque también pueden variar en función del estado de las entradas, lo que se conoce como máquina de Mealy[18]. Se podría decir que una máquina de Moore es una particularidad de la máquina de Mealy, en la cual las salidas no varían en función de las entradas.

Yo arrancaria desde aca:

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). Para el diseño de cada una de las operaciones de la MEF implementada en el presente trabajo, se recurrió a la confección del diagrama de flujo para un máquina de estados algorítmica.

Como se indica en la Sección 3.2, las señales de salida de la MEF diseñada que se comunican con el controlador FX2LP son *FIFOADR*, *SLOE*, *SLRD*, *SLWR* y *PKTEND*. El 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_Vacio*, *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 transmitir 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 órdenes 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 probables que los datos que se envíen.

Con las consideraciones mencionadas, se confeccionó la máquina de estados algorítmica que se observa en el diagrama de flujo de la Figura 3.8. En un estado inicial, todas las salidas se encuentran inactivas (en alto, dado que son activas en bajo). 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

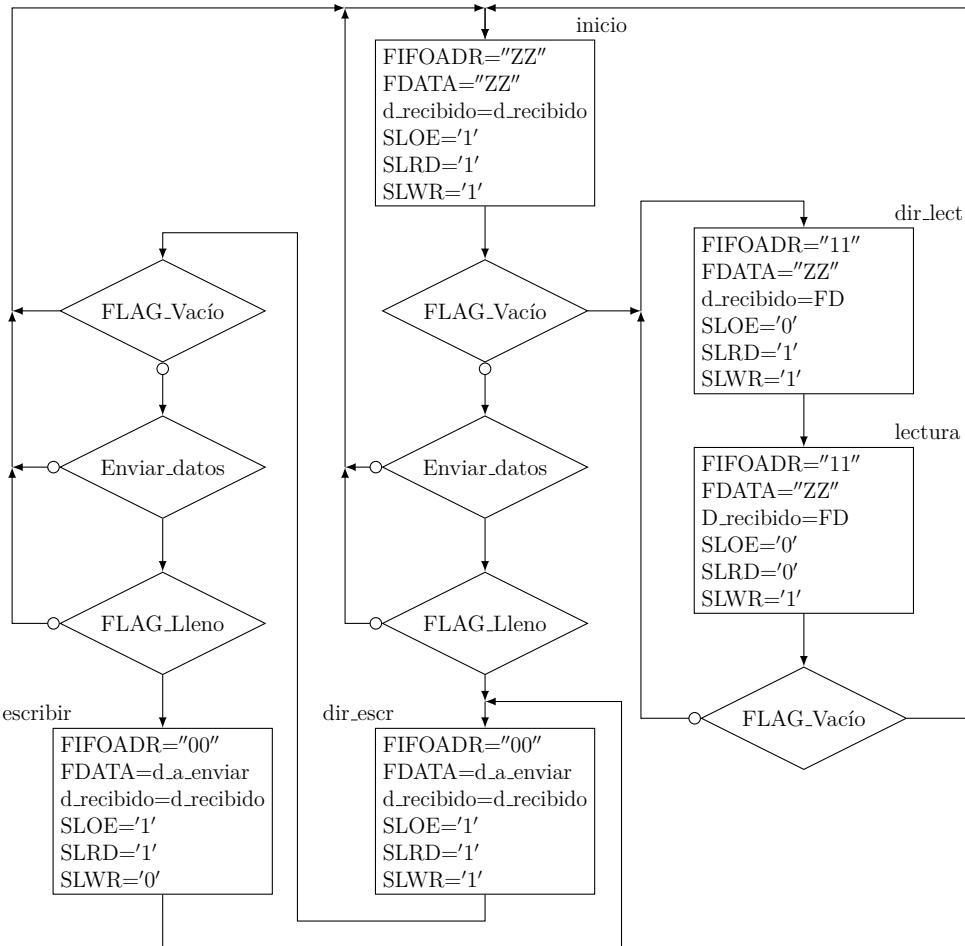


Figura 3.8: Diagrama de flujo de la máquina de estados desarrollada

No me queda en claro que es sistema genérico. Son los demás bloques implementados en la FPGA?

propio valor.

Cuando el *FLAG_Vacio* se activa, se procede a la operación de lectura. Si, en cambio, *FLAG_Vacio* está inactivo, 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* activo. 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 activa la señal *SLRD*, lo que incrementa el dato de la memoria FIFO. De esta manera, se puede volver a leer la señal de *FLAG_Vacio* y determinar si se vuelve a implementar una operación 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

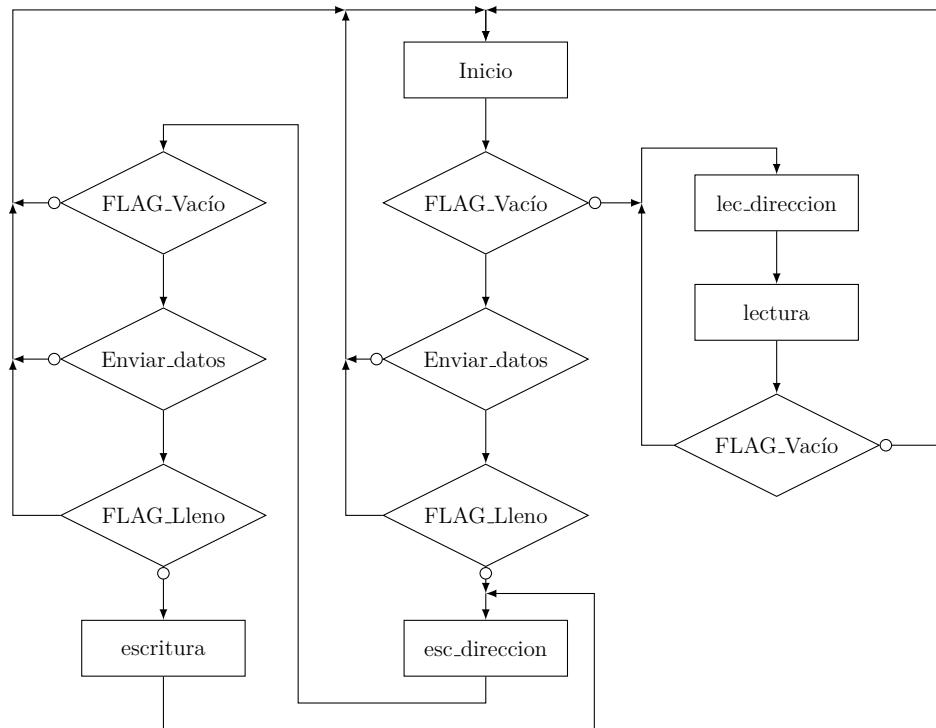


Figura 3.9: Diagrama de simplificado del flujo de la MEF

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.

3.4. Síntesis de la Máquina de Estados en VHDL

Yo uniría estos párrafos

Considerando las señales que se mencionaron 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.

Como se mencionó en la Sección anterior, una MEF se compone tres partes: la función del próximo estado, el estado actual y la función de salida. Cada uno de estas partes puede ser descrita 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.

Se presentan a continuación la función de próximo estado y la actualización del estado actual debido a que son, a criterio del autor, los procesos más importantes del desarrollo. El código completo, en donde se realiza la declaración de la entidad, la declaración de las señales, las conexiones internas y se asignan las funciones de salida a los estados en forma concurrente, se puede encontrar en el Apéndice A.

La función de próximo estado, es un proceso que lee el registro *estado_actual* y las señales de entrada y, en función de su valor, asigna el estado siguiente al registro *prox_estado*.

Para una mejor comprensión de la descripción de la función de próximo estado, 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 VHDL desarrollado. Además, para facilitar la lectura del desarrollo, se colocaron las dos señales de entrada *FLAG_Vacio* 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_mef := 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 ;
        end case;
    end process;

```

```

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;

```

La transición entre estados es un proceso que consta de un reloj que hace transfiere el valor del registro de *prox_estado* al registro *estado_actual*. A este reloj, se le acoplan dos temporizadores de habilitación. Esto se debe a que se algunas señales deben respetar ciertos tiempos de establecimiento y ancho de pulso. Cuando el próximo estado es *esc_direccion* 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 contador3 : natural range 0 to 4 := 0;
    signal contador2 : natural range 0 to 3 := 0;
    signal disp3      : std_logic := '0';
    signal disp2      : std_logic := '0';
begin
    reloj_mef: process (reloj_sist, reset)
    begin
        if reset = '0' then
            estado_actual <= inicio;
        elsif rising_edge(reloj_sist) then
            if contador2 = 0 and contador3 = 0 then
                estado_actual <= prox_estado;
            end if;
        end if;
    end process reloj_mef;

    tempo3: process(reloj_sist, reset, disp3)
    begin
        if reset = '0' then
            contador3 <= 0;

```

```

        elsif rising_edge(reloj_sist) then
            if contador3 > 0 then
                contador3 <= contador3 - 1;
            elsif disp3 = '1' then
                contador3 <= 4;
            end if;
        end if;
    end process tempo3;

    disp3 <= '1' when (prox_estado = esc_direccion) else '0';

tempo2: process(reloj_sist, reset, disp2)
begin
    if reset = '0' then
        contador2 <= 0;
    elsif rising_edge(reloj_sist)then
        if contador2 > 0 then
            contador2 <= contador2 - 1;
        elsif disp2 = '1' then
            contador2 <= 3;
        end if;
    end if;
end process tempo2;

with prox_estado select
    disp2 <=    '1' when lec_direccion | lectura | esc_direccion ,
    '0' when others;

end Behavioral

```

3.5. Verificación funcional de la síntesis

La verificación funcional es una rutina de control que sirve para que el desarrollador se asegure de lo que ha descripto a través de un lenguaje de alto nivel (VHDL en este trabajo), se corresponda con el funcionamiento esperado previo a la descripción. Para efectuar esta verificación, se describe el comportamiento esperado de las señales que de entrada al circuito desarrollado, se simula el sistema y se corrobora que las salidas y se comporten conforme a lo esperado.

Para efectuar la verificación funcional de la máquina de estados desarrollada, se describió en VHDL un ciclo típico de entrada de datos desde la interfaz y se realizó un chequeo de las salidas, tanto hacia el sistema exterior, como hacia el interior. También se simuló la entrada de datos desde el FPGA, generando la rutina de salida de datos hacia la interfaz.

~~Debido a que es una simulación y no si implementa en físico, la declaración de la entidad se deja en blanco.~~

(Modulos implementados dentro del
FPGA)

(Interfaz FX...)

En la implementación de la arquitectura, se declara e instancia el componente bajo prueba, en este caso la MEF y las señales necesarias para su correcta instanciación.

Una vez declarado e instanciado el componente bajo prueba, se generan los estímulos. De estos estímulos, las señales que emulan el reloj y el reset se establecen de forma concurrente. La señal de reloj es la que sincroniza el funcionamiento del sistema. La señal de reset es importante ya que asegura que el dispositivo inicie en el estado determinado por el diseño.

Finalmente, se [] el proceso que contiene los estímulos, emulando el funcionamiento de la interfaz. En un primer momento, las memorias FIFO se encuentran descargadas, es decir, los flags que indican memoria FIFO vacía se encuentran activos (*FLAGB* y *FALGD* en bajo), en tanto los flags que señalan memoria FIFO llena, se encuentran inactivos (*FLAGA* y *FLAGC* en alto). A su vez, el bus de datos *FDATA[15:0]* y el de dirección *FADDR[1:0]* se encuentran en estado de alta impedancia ('Z') y las salidas *SLWR*, *SLRD* y *SLOE* se encuentran inactivas. Estas condiciones se mantuvieron hasta que se desactiva la señal de reset, y luego, se mantienen por 5 ciclos de reloj. Así, se corrobora que ante la ausencia de estímulos, el sistema se mantiene invariable.

Yo pondría todo en pasado

Luego, se [] la operación de lectura de la memoria FIFO, con una serie de 3 datos aleatorios. Para este propósito, se [] la señal *FLAGB*, el flag vacío que corresponde al EP8, es decir, la memoria FIFO de salida (desde el Host). Seguidamente, se espera a la señal *SLOE*. Esta señal habilita a la interfaz a hacer uso del bus de datos *FDATA[15:0]*, por lo que se simulan datos aleatorios colocados en él, y luego se espera por la respuesta del sistema, el cual debería activar la señal de lectura *SLRD*.

Una vez realizada la operación de escritura se [] a describir la operación de escritura. Para esto, se [] un dato en el bus de envío de datos, *DATA_TO_TX* y se activa la señal de envío de datos *SEND_REQ*. Una vez que se activa la señal *SLWR*, se está en condiciones de cambiar el dato que se está enviando.

Se [] también dos pruebas extra, en el desempeño de la máquina de estados desarrollada. En primer lugar, se [] la interrupción del proceso de escritura a través de la señalización del ingreso de datos. Es decir, se debe detener el envío de datos cuando la señal *FLAGB* se active.

Además, se observa que también se debe detener el envío de datos cuando la memoria FIFO que recibe los datos que salen de nuestro sistema se llena. Esta situación es avisada por la interfaz a través de la señal *FLAGA*.

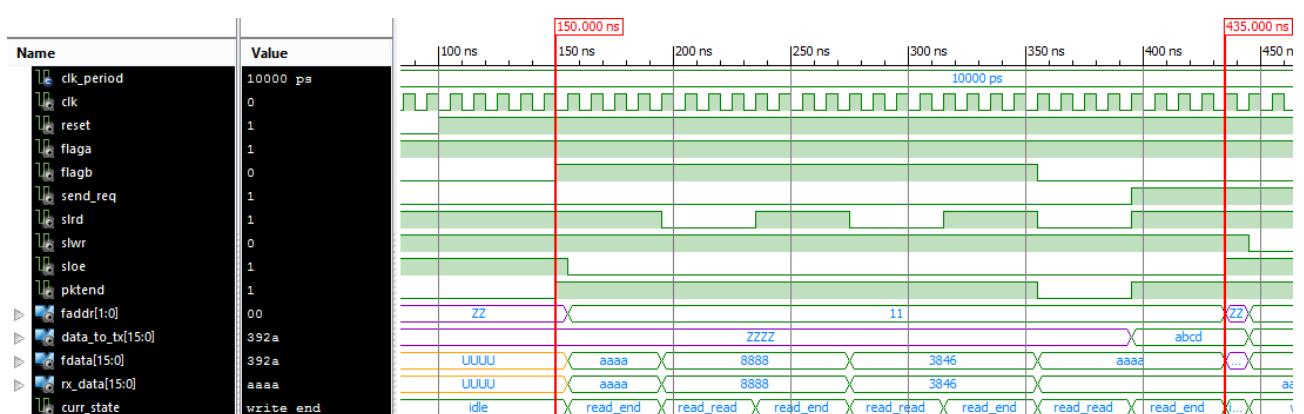


Figura 3.10: Diagrama temporal de la operación de lectura entregado por el simulador

Se puede observar en la Figura 3.10 el diagrama temporal obtenido a partir de la simulación. Se resalta en este diagrama que luego de liberada la señal de reset, el sistema conserva todas las variables en el estado esperado mientras no existe estímulo.

Una vez que es desactivada la señal *FLAGB*, el bus de dirección apunta a la memoria FIFO de entrada de nuestro sistema, es decir, *FADDR[1,0] = "11"*. La señal *SLOE* se coloca en bajo, ese decir que activa el bus de datos, *FDATA[15:0]* para que sea usado por la interfaz. Como se indicó en la descripción realizada, se _____ un dato en el bus y este es volcado en el bus que comunica los datos que ingresan al interior del sistema, *RX_DATA[15:0]*. Luego, se activa la señal *SLRD* en forma repetida, tal y como se espera.

También se observa que el estado actual en cada uno de los momentos es el adecuado. Además, se observa que, aunque se coloquen datos en el bus de salida interno *DATA_TO_TX[15:0]* y se active la señal de envío *SEND_REQ*, el bus de datos que se comunica con la interfaz *FDATA[15:0]* permanece con los datos externos.

Una vez que la MEF alcanzó el estado inicial (*IDLE*, como se ve en la Figura 3.11), procede a la escritura de datos. A partir de allí, mientras no existan datos en la memoria de entrada (*FLAGB* permanezca en alto), la memoria de salida no se encuentre llena (*FLAGA* se encuentre en alto) y sea requerido el envío de datos, el sistema procede a activar en forma sistemática los datos que encunetra en el bus de envío de datos *DATA_TO_TX*.

La Figura 3.12 muestra cómo el sistema finaliza la operación de escritura cuando un sucede al menos uno de los eventos considerados para su interrupción. Cuando se activa *FLAGB* mientras se ejecuta la operación de escritura, esta última es interrumpida y el sistema procede a realizar, el próximo estado, la operación de lectura. En el caso de que active la señal *FLAGA*, indicando que la memoria que recibe los datos que envía el FPGA alcanzó el máximo de capacidad, el proceso de escritura es detenido. En ambos casos, no es relevante el valor de la señal de envío de dato *SEND_REQ*, ya que ambos eventos poseen prioridad a esta.

Una vez realizada la simulación y verificación funcional de la síntesis desarrollada, se está en condiciones de grabar dicha síntesis en el FPGA y se puede proceder a realizar pruebas de funcionamiento más avanzadas. Las pruebas del sistema realizadas se detallan en el capítulo siguiente.

El código completo de la simulación, se pueden leer en el Anexo.

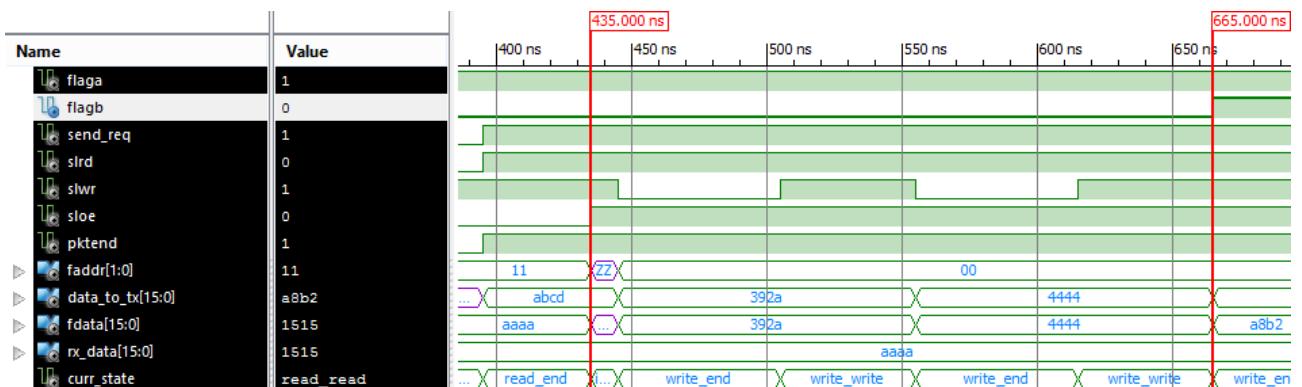


Figura 3.11: Diagrama temporal de la operación de escritura entregado por el simulador

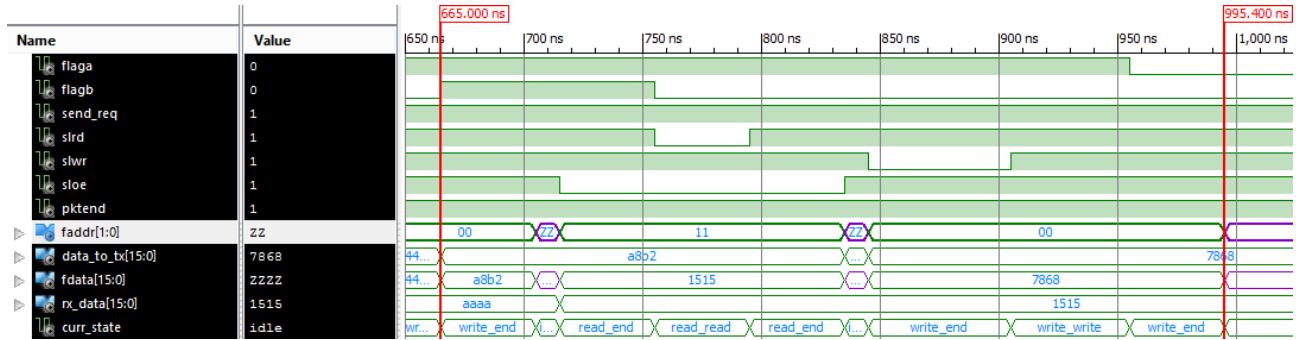


Figura 3.12: Diagrama temporal que muestra las interrupciones de la operación de escritura

3.6. Placa de Interconexión

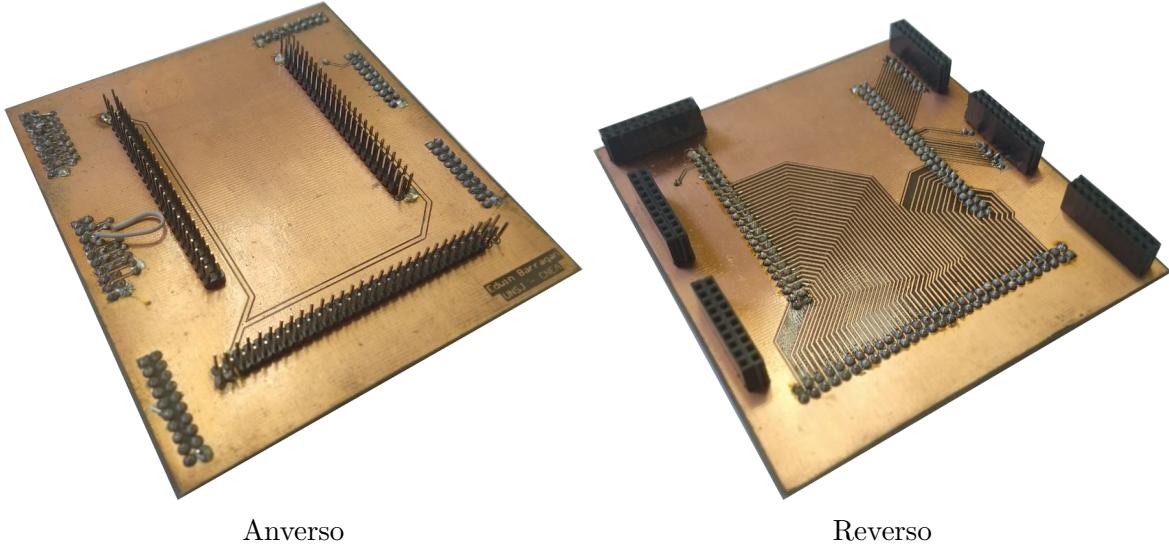


Figura 3.13: Versión 2 de la placa de interconexión

Hasta aquí, se tiene cada una de las partes que componen el desarrollo trabajando correctamente en forma individual. Sin embargo, previo a realizar pruebas del sistema en su totalidad, se debe conectar cada una de las partes en forma física. 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 inglés *Printed Circuit Board*) que conecta en forma eléctrica dos o más dispositivos. Esto brinda una conexión mucho más robusta y prolífica que si fuese realizada mediante cables cintas o alambres individuales. La Figura 3.13 muestra el circuito impreso desarrollado. El plano esquemático y el dibujo utilizado para su fabricación se puede consultar en el Anexo ??.

El circuito impreso desarrollado determina en forma ?? los puertos que se conectan entre la placa de desarrollo de la interfaz y la del FPGA. En la Tabla 3.2 se puede observar la correspondencia de cada una de las señales de interés del controlador FX2LP con los puertos del FPGA Spartan 6.

FX2LP	Spartan 6
FD15	P50
FD14	P51
FD13	P40
FD12	P41
FD11	P34
FD10	P35
FD9	P32
FD8	P33
FD7	P29
FD6	P30
FD5	P26
FD4	P27
FD3	P23
FD2	P24
FD1	P21
FD0	P22
SLWR	P17
SLRD	P16
SLOE	P6
FLAGA	P12
FLAGB	P14
FLAGC	P15
FLAGD	P11
PKTEND	P10
FIFOADR1	P9
FIFOADR0	P8

Tabla 3.2: Correspondencia entre las señales del controlador FX2LP y el FPGA Spartan 6 fijada por el PCB.

3.7. Sumario del capítulo

Durante el presente capítulo se desarrolló cuales son las señales de control que intervienen y como es su funcionamiento en las operaciones de lectura y escritura externa en las memorias FIFO. En base a ellas se diseño e implemento la máquina de estados finitos. Se utilizó VHDL como lenguaje de implementación de la MEF, para su posterior síntesis en FPGA. Se realizó también una verificación funcional de la síntesis realizada, a fin de corroborar su correcto funcionamiento. Finalmente 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 y como éste circuito impreso determina la correlación entre los puertos de cada uno de los circuitos integrados.

