Capítulo 1

Programación y configuración de la interfaz EZ-USB FX2LP

La interfaz que este trabajo utiliza para que los datos fluyan entre un FPGA y una PC está compuesta por el controlador EZ-USB FX2LP de Cypress, el cual viene incorporado en el kit de desarrollo CY3684.

El kit de desarrollo CY3684 puede ser descompuesto en dos partes: una de hardware, que posibilita la conexión eléctrica entre los componentes y una parte de software que facilita al desarrollador tanto la elaboración del programa que es cargado y ejecutado por el microcontrolador, denominado firmware, como las pruebas del sistema en desarrollo.

Este Capítulo aborda algunos aspectos conceptuales sobre la estructura y arquitectura del circuito integrado EZ-USB FX2LP y desarrolla la configuración seleccionada, la elaboración del firmware y las herramientas utilizadas.

1.1. Arquitectura FX2LP EZ-USB

El núcleo del Kit de Desarrollo CY3684 es el controlador EZ-USB FX2LP. La serie de controladores FX2LP se caracteriza por brindar una conexión USB 2.0 de alta velocidad y bajo consumo energético. Está diseñada, preferentemente más no exclusivamente, para periféricos

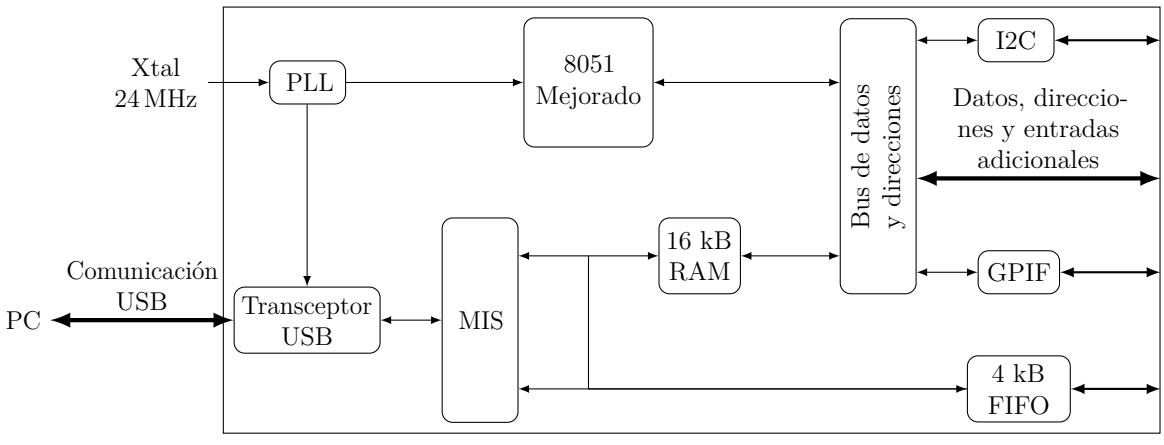

Figura 1.1: Arquitectura FX2LP

con autonomía limitada.

La arquitectura de controlador FX2LP, tal como se presenta en la Figura 1.1, integra un controlador USB completo. Incluye un transceptor USB, un Motor de Interfaz Serie (MIS), buffers de datos configurables, un microcontrolador 8051 que contiene registros y funciones adicionales orientadas a mejorar el rendimiento de la comunicación USB y una interfaz programable hacia los periféricos implementada con memoria tipo FIFO ($FirstInFirstOut$; Primero Entrado, Primero Salido). Además posee un PLL con divisor configurable a través del cual provee las señales de reloj adecuadas para el correcto funcionamiento del sistema.

El flujo de datos posee dos puntas (una PC y un FPGA) entre las cuales el controlador cumple el rol de interfaz. Para ello necesita poder comunicarse tanto con el host como con los periféricos. El usuario puede trasmitir datos desde y hacia el host a través del mismo puerto USB. Sin embargo, también posee dos puertos DE-9 que permiten comunicarse con la PC a través del protocolo UART (acrónimo de Transmisión y Recepción Asíncrona Universal, en inglés) que facilitan en gran medida la tarea de depuración del desarrollo gracias a que posee una configuración simple.

En cuanto a la interfaz con uno o mas perifericos, el controlador posee un puerto I^2C, una interfaz de propósito general (GPIF), para sistemas que necesitan ser comandados en forma externa; y una interfaz con memorias FIFO esclavas, a través de las cuales se puede conectar sistemas que cumplen un rol activo en el envío y recepción de información. Estas tres interfaces posibilitan la conexión de dispositivos que poseen tanto puertos estandarizados (ATA, PCMCIA, EPP, etc.), cómo personalizables (DSP, FPGA, microcontroladores, etc).

Este trabajo utiliza particularmente las memorias FIFO en modo esclavo, que responden a las diferentes señales que les proporciona un maestro externo implementado con un FPGA; por lo que a continuación se explicitan algunos detalles referidos a ellos, con lo que se busca aclarar el funcionamiento y que el lector comprenda los fundamentos de las configuraciones que se plasman en el código del firmware.

1.1.1. Motor de Interfaz Serial

El Motor de Interfaz Serial (MIS) es un módulo incorporado al circuito integrado que se encarga de tomar datos en paralelo y convertilos en una secuencia seriada.

La comunicación USB entre el controlador FX2LP y la PC se realiza a través del transceptor, unido al MIS. Para realizar el intercambio de datos, el firmware solo debe colocar o extraer los datos de buffers programables y modificar las banderas de handshaking. En forma automática, el MIS se encargan de empaquetar, enviar, recibir y desempaquetar toda la información, así como leer los tokens que emite el host, calcular y corroborar los códigos cíclicos de detección de errores y todo lo relacionado al protocolo propiamente dicho. El transceptor codifica y decodifica todo a nivel físico.

La Figura 1.2 muestra la función del MIS. Toma los datos colocados en los buffers de extremos, agrega la información que corresponde al encabezado y a la cola y, finalmente, coloca el registro de handshaking. Esto último, se observan como ACK (abreviación del ingles *acknowledge*, que significa reconocer, aceptar o agradecer) en la Figura 1.2. En el extremo del controlador, estas banderas se colocan en un registro especial que indica si el sistema está disponible, si los datos fueron colocados o leídos, dependiendo el caso tratado.

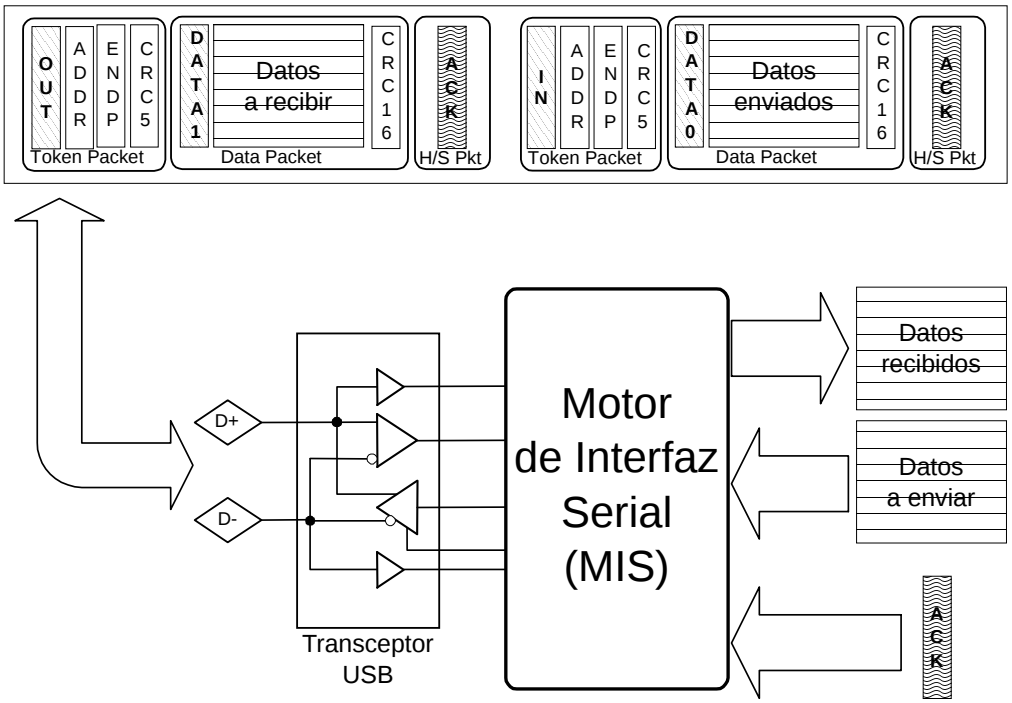

Figura 1.2: Implementación del enlace USB realizado por el EZ-USB[30]

1.1.2. Buffers de extremos

El MIS guarda los datos que aún no han sido enviados y/o los que han sido recibidos pero no leídos por ningún periférico en una memoria RAM específica, denominada buffer de extremo.

La norma USB define a un dispositivo extremo como una porción exclusiva e identificable de una dispositivo USB que es fuente o un sumidero de información. En otras palabras, USB ve a cada extremo como una memoria FIFO de donde surge o finaliza la información. En ingles, el termino extremo recibe el nombre de *endpoint*, por lo que, en adelante, cuando se hable de ellos se abreviara como EP o EPx, siendo la x un número que indica la dirección del extremo.

La serie de controladores FX2LP dispone de hasta 7 EP programables, los cuales deben poseer al menos dos buffers. La norma USB indica que cualquier dispositivo USB debe poseer un EP con dirección 0 que se destina para control y configuración, por lo que el controlador está dotado de 64 B para este fin. Es el único EP que puede ser bidireccional en el sentido del flujo de datos. A través de él, host y dispositivo envían y reciben transferencias de control. Luego, se incorporan dos EP1, que poseen un buffer de 64 B cada uno. Estos EP se identifican por la dirección de los datos, ya que uno de ellos es de salida y el otro de entrada de datos.

Finalmente, se incorpora una memoria de 4 KiB que debe ser configurada para los EP2, EP4, EP6 y EP8. La configuración de los EP la realiza el microcontrolador una vez que su programa se encuentra en ejecución. Las variables, conforme a los requerimientos de ancho de banda y acceso al bus son:

- Tamaño: Dependiendo del extremo a configurar puede ser de 512 o 1024 bytes.
- Tipo de acceso al bus: Definido según la norma USB, este tipo puede ser por bultos, isocrónico o de interrupción. No se admiten en estos EP paquetes de control.

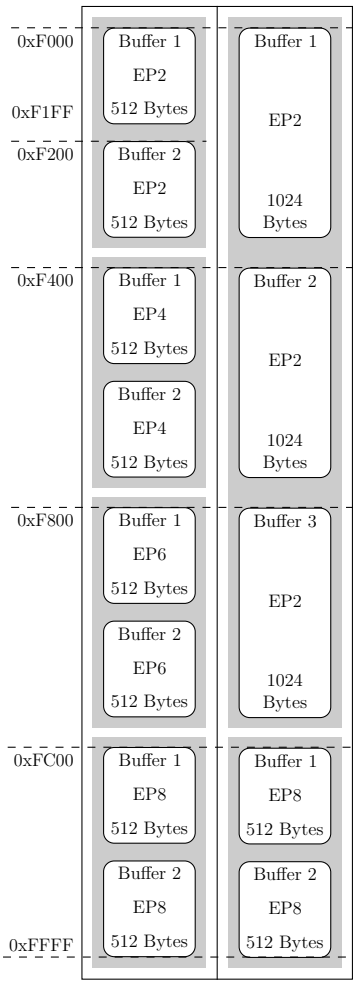

Figura 1.3: Buffers de extremos con sus direcciones de memoria. El cuadro de la izquierda muestra la configuración por defecto. El derecho, la implementada en este trabajo.

- Cantidad de buffers: Dependiendo del extremo, puede ser dos, tres o cuatro buffers por extremo.
- Habilitación: Se debe indicar al sistema si los extremos se usan o no. El EP no valido, no responderá a un pedido de entrada o salida.

La Figura 1.3 muestra solo dos de las posibles configuraciones de los EP. A la izquierda se observa la configuración por defecto del controlador FX2LP. Esto es, los cuatro EP habilitados, con 512 bytes cada uno, buffers dobles y comunicación por bultos. A la derecha se muestra la configuración elegida para este trabajo, es decir, solo son EP válidos el EP2 y EP8. EP2 posee tres buffers de 1024 bytes y el EP8 dos buffers con 512 bytes de capacidad cada uno. Siempre se debe considerar que se dispone hasta 4 KiB de memoria.

La característica de los buffers múltiples evita la congestión de datos. Con doble buffer, un periférico (o el microcontrolador) coloca o extrae datos de un buffer, mientras otro, del mismo EP, se encuentra enviando o recibiendo datos mediante el MIS. Cuando se configura un triple o cuádruple buffer, se agrega una o dos porciones mas de memoria a la reserva, respectivamente. De esta forma, se le otorga al sistema una gran capacidad de datos y ancho de banda.

Un detalle importante de los buffers múltiples es que, a la vista del controlador y/o de un periférico, el buffer posee una sola y única dirección y, es la propia interfaz FX2LP quien se encarga de seleccionar el buffer que corresponde en cada caso. Esto quiere decir que, por ejemplo, teniendo 4 buffers de 512 B cada uno, el 8051 verá solo uno de 512 B, sin necesidad de identificar a traves de su firmware con cuál de los cuatro está trabajando.

1.1.3. Memorias FIFO esclavas

Desde el punto de vista de la electrónica digital, el MIS es un dispositivo que recibe y envía datos desde y hacia el puerto USB utilizando una señal de reloj de 24 MHz. Esta señal, es provista por un cristal de cuarzo incorporado en el circuito impreso del Kit de Desarrollo CY3684 EZ-USB FX2LP. Por su parte, un sistema externo puede o no proveer una señal de reloj y manejo de datos propio cuy a fuente de reloj es a priori desconocida por quien configura el circuito integrado. El controlador USB incorpora memorias FIFO que se encargan de proveer una interfaz entre el MIS y un dispositivo externo, salvando el problema de poseer dos relojes diferentes e independientes.

Estas memorias funcionan en modo esclavo, es decir, se debe conectar un dispositivo capaz de proveer una lógica maestra externa que comande la entrada y salida de datos desde una memoria FIFO hacia o desde el exterior. Para los fines del presente trabajo, este modo de funcionamiento es óptimo ya que, dotando al FPGA de una máquina de estados, se logra la transferencia de datos en los tiempos requeridos.

El sistema de bus permite conectar a estas memorias hasta cuatro dispositivos diferentes. Por esto, existe un registro que permite seleccionar una porción de memoria FIFO para cada uno de los EP programables en el buffer de extremos.

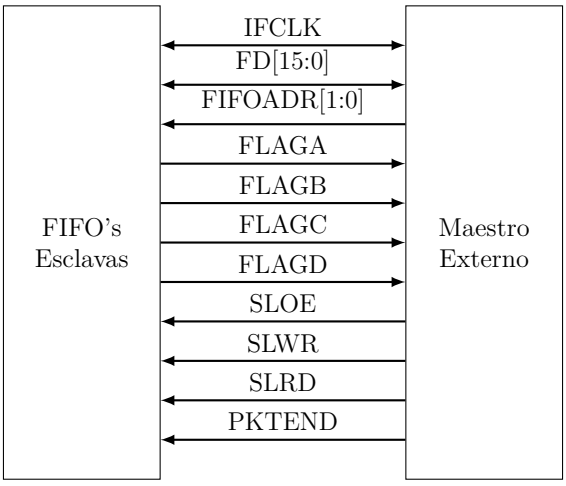

Figura 1.4: Puertos de interfaz entre las FIFO's y un maestro externo

La Figura 1.4 muestra las señales de la interfaz entre las memorias FIFO's y un maestro esclavo. Estas 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.

- FD[15:0]: constituye el bus de datos. Según 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 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.

1.1.4. Modos de entrada y salida automáticos

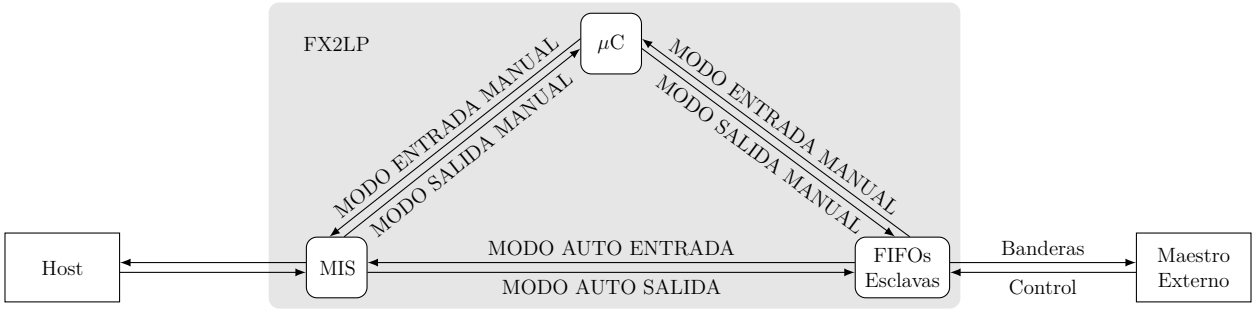

Figura 1.5: Modos de conexión de la memoria FIFO, el microntrolador y el MIS

Los datos que se reciben o envían a través del MIS. Pueden ser enviados en forma automática desde y hacia las memorias FIFO, o bien, pueden ser dirigidos a través del microcontrolador. Esto último permite leer, modificar, suprimir, agregar y/o generar nuevos datos antes de ser remitidos como paquete, es decir, todos juntos, a su respectivo EP. Estos caminos se pueden ver en la Figura 1.5.

Los fabricantes llaman a estos caminos "MODO MANUAL", en caso de enviar los datos a través del 8051, y "MODO AUTOMÁTICO", cuando la comunicación es directa entre el MIS y las FIFO. Además, se programan en forma independiente para cada extremo, sea este de salida o entrada.

Se debe notar en la Figura 1.5 que se refiere a paquetes de entrada cuando estos poseen una dirección que se inicia en un periférico y termina en el host y de salida cuando llevan el sentido contrario. Esto se debe al rol central que ejerce el host en la comunicación USB.

1.2. Kit de desarrollo de Software de Cypress

A través del kit de desarrollo CY3684 EZ-USB FX2LP, Cypress provee un amplio conjunto de herramientas que facilitan en gran medida la implementación del software. Estas soluciones

se denominan Kit de Desarrollo de Software (SDK, acrónimo del habla inglesa, *Software Development Kit*). Este SDK abarca una gran cantidad de aspectos, como ser la elaboración del firmware implementado en el 8051 del controlador, el desarrollo de programas de control para PC, la conversión de archivos de programación y ejecutables que graban la información en las diferentes memorias que posee la placa de desarrollo, sea RAM o EEPROM, para cargar programas no volátiles que posteriormente serán ejecutados por el 8051. No todas las herramientas se utilizan en este trabajo, por lo que se desarrollan las más destacadas.

1.2.1. Framework Cypress

Con la intención de dotar al desarrollador con una herramienta que le potencie la velocidad de diseño, Cypress provee una plantilla de código en lenguaje C para microcontroladores 8051.

Esta plantilla posee precargados todos los registros que posee la serie de controladores FX2LP con los mismos nombres que figuran en el manual de usuario. Además incorpora las funciones que el microcontrolador debe llevar a cabo para efectuar la comunicación USB y algunas que permiten interactuar con la placa de desarrollo CY3684. Los archivos que incorpora son:

- fw.c: es el código fuente principal. Contiene la función main() y lo necesario para manejar la comunicación USB.
- periph.c: contiene la implementación de las funciones invocadas por fw.c. Aquí se encuentran TD_Init() y TD_Poll(), las funciones a través de las cuales el usuario implementa el programa que necesita. También contiene todas las funciones de interrupción vectorizadas.
- fx2.h: posee la definición de constantes, macros, tipos de datos y funciones prototipo de la biblioteca.
- fx2regs.h: declara registros y máscaras.
- dscr.a51: este archivo es el descriptor de dispositivo. Es la información que USB necesita para poder registrar el dispositivo en el host, asignarle dirección y establecer los parámetros.
- ezusb.lib: biblioteca que implementa funciones provistas por el fabricante.
- syncdely.h: macro de sincronismo. Algunos accesos de registro requieren un tiempo de establecimiento específico que en el código se implementa deteniendo el microcontrolador ciertos ciclos de reloj.
- usbjmptb.obj: especifica las direcciones en memoria de las interrupciones vectorizadas.

La Figura 1.6 muestra una versión modificada del diagrama de flujo que sigue el código fuente provisto por Cypress. De ella se quitan funciones que no son necesarias para los objetivos del presente trabajo.

Cuando el programa es cargado al controlador, este se encarga de inicializar todas la variables de estado a su valor por defecto. También en este punto establece la comunicación con el anfitrión y le envía los descriptores provistos en el archivo `dscr.a51`. Acto seguido, ejecuta la función `TD_Init()`, a través de la cual el usuario programa e inicia la configuración del sistema. Luego, es necesario habilitar las interrupciones necesarias y finalmente, se invoca repetidamente la función `TD_Poll()`, en donde el usuario escribe las tares que ejecutará el 8051.

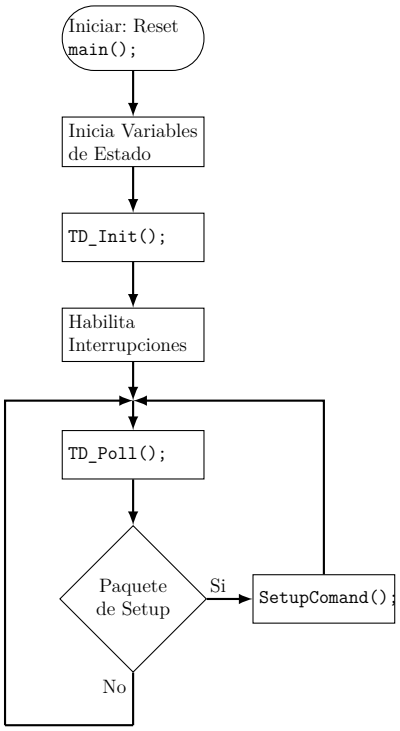

Figura 1.6: Diagrama en bloques simplificado

1.2.2. Entorno de desarrollo y compilador

Si bien Cypress no desarrolla software para escribir, compilar y depurar códigos, distribuye junto con el kit CY3684 una versión para evaluación de Keil μVision con el compilador C51 para programar microcontroladores basados en 8051. Aún la versión limitada de este entorno, resulta suficiente para la programación del controlador USB.

Keil μVision es un entorno de desarrollo integrado (IDE). Se entiende por IDE a un software que integra en un entorno gráfico las herramientas que permiten elaborar un programa que ejecutará un procesador, desde la escritura del algoritmo en uno o más lenguajes, su compilación, las pruebas y el depurado.

El programa utilizado posee, entre otras cosas, editor de textos con atajos de teclado, comandos que aceleran la escritura de código y resaltado de palabras claves para diferentes lenguajes de programación, navegador de archivos. También ejecuta, con solo un click, el compilador con la sintaxis correcta, y posee un depurador que, a través de un intérprete, permite ir ejecutando el código línea por línea o en bloques.

Para realizar un programa en este entorno, Cypress provee, junto con su framework, un proyecto vacío que puede ser copiado y pegado. Sin embargo, se puede realizar la configuración manual. Las instrucciones de este procedimiento se ubican en el Apéndice **??**.

En cuanto al compilador se refiere, el utilizado es C51. Éste es un programa que otorga un archivo hexadecimal con un código que será ejecutado por microcontroladores que estén implementados con la misma estructura que un Intel 8051, cómo lo es el microcontrolador que posee el FX2LP.

1.2.3. Cypress USB Control Center

Para grabar los programas desarrollados en el microcontrolador, el SDK provee de la aplicación USB Control Center. Además de la capacidad para programar el microcontrolador, este programa posee posibilita el envío y la recepción de datos a través del puerto USB conectado con el dispositivo programado, y muestra la configuración que el mismo envió en forma de descriptores. Esta característica brinda una realimentación, aunque mínima, de la entrada y salida de datos, facilitando las pruebas sobre el sistema en desarrollo. En el Apéndice **??** se explica su funcionamiento y manejo.

1.3. Desarrollo del firmware

A continuación se desarrollan los aspectos más relevantes del código final elaborado, compuestos por la función de inicialización y los descriptores del dispositivo USB.

Al establecer el modo automático de funcionamiento, la función `TD_Poll()` se encuentra vacía.

Luego, en el Capitulo 5 se abordará nuevamente algunos detalles realizados para la depuración del presente código, tales como la conexión del puerto serie y la utilización de algunas interrupciones específicas.

1.3.1. Inicialización del dispositivo

La inicialización del dispositivo se realiza a través de la función `TD_Init()`. Ésta es invocada solo una vez en el código, antes de ejecutar el loop principal, donde el programa ejecuta tareas específicas una y otra vez.

En primer lugar, se debe configurar la frecuencia a la que corre el reloj principal. Esta puede ser de 12 MHz, 24 MHz o de 48 MHz. Para ello se deben colocar los registros CPUCS.4=1 y CPUCS.3=0. A través del framwork de Cypress, esto puede ser escrito de la siguiente forma:

```
CPUCS = ((CPUCS & ~bmCLKSPD) | bmCLKSPD1); // 48 MHz
```

Luego se debe configurar el funcionamiento del reloj de la interfaz. El esquema de la Figura 1.7 muestra la configuración del hardware. Cómo se observa, La interfaz puede funcionar con frecuencias de 48 MHz o 30 MHz provistas por el FX2LP. Además, la señal puede ser dirigida al exterior, o bien, ser provista por un periférico. Los inversores se programan de forma tal que la interfaz sea activa en el flanco positivo o negativo del reloj fuente. En este trabajo, el registro se programa para tomar como reloj la señal de 48 MHz provisto por el mismo chip, la polaridad es de flanco ascendente y no se posee señal de reloj en el pin externo IFCKL. Además, se programa de modo asíncrono y en modo FIFO esclavo, a cuya configuración se accede por este puerto.

El autor entiende sobre la posibilidad y conveniencia de utilizar el modo sincrónico para conectar la interfaz. Más aún, es factible proveer la señal necesaria de reloj desde la FPGA y evitar así problemas de desajustes y fallas de sincronismo. Sin embargo, por error del alumno en el diseño del impreso de interconexión, la entrada de reloj no quedó conectada al chip de la FPGA. Durante la escritura de este informe, se encuentra en viaje el impreso con las correcciones pertinentes y espera ser implementado en trabajos futuros.

La configuración, entonces, queda definida por la sentencia de código:

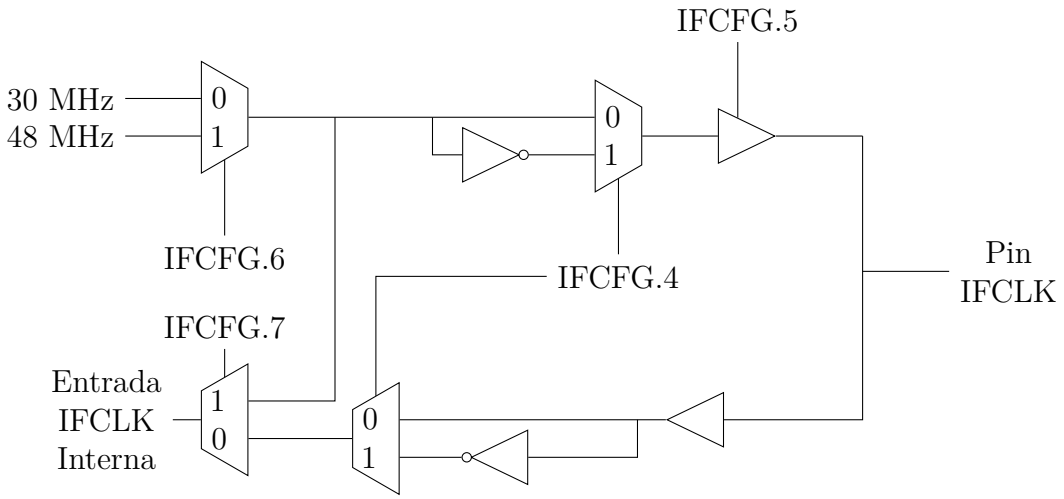

Figura 1.7: Esquema funcional para la entrada de reloj de la Interfaz

;

```
//colocar Interfaz FIFO esclava a 48MHz
//clk interno, no_salida, no_invertir clk, no_asincr
//fifoesclava(11) = 0xC3
//0xCB; para asincr.
IFCONFIG = 0xCB;
SYNCDELAY;
```

A continuación, se debe establecer el funcionamiento de los pines bandera. Estos avisan cuando un EP está vacío, completo o a un nivel programable. Para este trabajo, son necesarios solo una bandera que señale el nivel vacío del puerto por el que entran los datos a la FPGA y otra que señale el nivel completo del EP donde escribe los datos a enviar. Si bien no son leídos en ningún momento, para evitar problemas se configuran las banderas vacías y completas para ambos EP's:

```
//Pin Flags Configuratión
PINFLAGSAB = 0xBC;   // FLAGA <- EP2 Full Flag
                     // FLAGD <- EP2 Empty Flag
SYNCDELAY;
PINFLAGSCD = 0x8F;   // FLAGC <- EP8 Full Flag
                     // FLAGB <- EP8 Empty Flag
```

Luego, se deben programar los EP's. La configuración por defecto define a todos los EP como transferencias por bultos con doble buffer de 512 B. El EP2 y EP4 son salidas (desde la PC). El EP6 y EP8 son entradas (hacia la PC).

La programación de este trabajo es con una entrada isocrónica de dos buffers con 512 B de capacidad, cada uno, definida en el EP2 y una salida por bultos de dos buffers de 512 B configurada en el EP8. Los otros EP's son deshabilitados. Sin embargo, EP1IN y EP1OUT se dejan configurados por defecto, ya que no interfieren en nada con el presente trabajo.

```
EP1OUTCFG = 0xA0;
SYNCDELAY;
```

```
EP1INCFG = 0xA0;
SYNCDELAY;
EP4CFG &= 0x7F;
SYNCDELAY;
EP6CFG &= 0x7F;
SYNCDELAY;
EP8CFG = 0xA0; //EP8 is DIR=OUT, TYPE=BULK, SIZE=512, BUF=2x
SYNCDELAY;
EP2CFG = 0xD2;   // EP2 is DIR=IN, TYPE=ISOC, SIZE=512, BUF=2x
SYNCDELAY;
```

Como siguiente paso, se limpian las memorias FIFO de cualquier dato espúreo que contengan al momento del inicio del programa.

```
FIFORESET = 0x80;
SYNCDELAY;
FIFORESET = 0x02;
SYNCDELAY;
FIFORESET = 0x04;
SYNCDELAY;
FIFORESET = 0x06;
SYNCDELAY;
FIFORESET = 0x08;
SYNCDELAY;
FIFORESET = 0x00;
SYNCDELAY;
```

De esta forma, el controlador se encuentra listo para configurar las memorias asignadas a cada uno de los EP's. Se debe notar que en primer lugar se coloca 0x00 y luego el valor estipulado. Esto se basa en que el modo automático se prepara para transmitir ante un flanco ascendente del registro que lo habilita.

```
EP8FIFOCFG = 0x00;
SYNCDELAY;
EP2FIFOCFG = 0x00;
SYNCDELAY;

//setting on auto mode. rising edge is necessary
EP8FIFOCFG = 0x11;
SYNCDELAY;
EP2FIFOCFG = 0x0D;
SYNCDELAY;
```

Finalmente, se habilitan las interrupciones necesarias.

```
USBIE |= bmSOF;
```

Así, queda completa la inicialización y el dispositivo listo para enviar y recibir datos de forma automática.

1.3.2. Encabezado y declaraciones importantes

Para el correcto funcionamiento de este código, es necesario incorporar el encabezado que se observa a continuación.

```
#pragma noiv                    // No generar vectores de interrupción
#include "fx2.h"
#include "fx2regs.h"
#include "syncdly.h"            // SYNCDELAY macro
#include "leds.h"
```

Las primeras 4 líneas de encabezados son provistas por Cypress, a través de su framework. En ellas, la directiva de ensamblador noiv(identificada con `#pragma`), le indica al compilador que no debe habilitar las interrupciones vectorizadas. Estas, en cambio, serán manejadas y direccionadas a través del archivo objeto *usbjmptb.obj*.

El encabezado *leds.h* cuyo código se muestra a continuación, sirve para encender y apagar las luces de la placa de desarrollo. Estos leds se encuentran conectados a través de un decodificador y su funcionamiento se da con la sola lectura de direcciones específicas.

```
xdata volatile const BYTE D2ON  _at_ 0x8800;
xdata volatile const BYTE D2OFF _at_ 0x8000;
xdata volatile const BYTE D3ON  _at_ 0x9800;
xdata volatile const BYTE D3OFF _at_ 0x9000;
xdata volatile const BYTE D4ON  _at_ 0xA800;
xdata volatile const BYTE D4OFF _at_ 0xA000;
xdata volatile const BYTE D5ON  _at_ 0xB800;
xdata volatile const BYTE D5OFF _at_ 0xB000;
```

Luego, el framework define algunas variables globales que utiliza en las funciones implementadas para el manejo de las tareas relacionadas al protocolo USB. Se listan estas variables a continuación.

```
extern BOOL   GotSUD;           // Received setup data flag
extern BOOL   Sleep;
extern BOOL   Rwuen;
extern BOOL   Selfpwr;

BYTE    Configuration;          // Current configuration
BYTE    AlternateSetting = 0;   // Alternate settings

//-----------------------------------------------------------------------------
// Task Dispatcher hooks
//   The following hooks are called by the task dispatcher.
//-----------------------------------------------------------------------------

WORD blinktime = 0;
BYTE inblink = 0x00;
BYTE outblink = 0x00;
WORD blinkmask = 0;          // HS/FS blink rate
```

1.3.3. Descriptores USB

Los descriptores son una estructura definida de datos. A través de ellos, el dispositivo USB le comunica al anfitrión sus atributos, tales como velocidad de trabajo, cantidad de configuraciones e interfaces posibles, número de EPs, dirección de cada uno de ellos, tamaño máximo en bytes de paquetes que puede enviar en una comunicación, entre otros.

El framework de Cypress coloca toda la información sobre los descriptores de el dispositivo desarrollado en un archivo escrito en lenguaje de ensamblador, denominado *dscr.a51*. Este archivo contiene una plantilla en donde el desarrollador coloca la descripción de la configuración realizada en el firmware. Luego, el archivo es enviado por el dispositivo al host comunicando las características del sistema desarrollado.

En primer lugar, posee un encabezado en donde se establecen etiquetas que hacen más legible el código para el programador:

```
1     DSCR_DEVICE     equ   1    ;; Descriptor type: Device
2     DSCR_CONFIG     equ   2    ;; Descriptor type: Configuration
3     DSCR_STRING     equ   3    ;; Descriptor type: String
4     DSCR_INTRFC     equ   4    ;; Descriptor type: Interface
5     DSCR_ENDPNT     equ   5    ;; Descriptor type: Endpoint
6     DSCR_DEVQUAL    equ   6    ;; Descriptor type: Device Qualifier
7
8     DSCR_DEVICE_LEN     equ   18
9     DSCR_CONFIG_LEN     equ    9
10    DSCR_INTRFC_LEN     equ    9
11    DSCR_ENDPNT_LEN     equ    7
12    DSCR_DEVQUAL_LEN    equ   10
13
14    ET_CONTROL      equ    0    ;; Endpoint type: Control
15    ET_ISO          equ    1    ;; Endpoint type: Isochronous
16    ET_BULK         equ    2    ;; Endpoint type: Bulk
17    ET_INT          equ    3    ;; Endpoint type: Interrupt
```

Luego, el programador se debe asegurar que el código se guarda en un lugar de memoria adecuado.

```
1  DSCR    SEGMENT    CODE PAGE
2
3  ;;-----------------------------------------------------------------------
4  ;; Global Variables
5  ;;-----------------------------------------------------------------------
6      rseg DSCR          ;; locate the descriptor table in on-part memory.
```

Debido a la estructura rígida del formato de los descriptores, impuesto por la norma USB y por la implementación de Cypress,la memoria debe contener en primer lugar el descriptor de dispositivo.

```
1  DeviceDscr:
2      db    DSCR_DEVICE_LEN        ;; Largo del descriptor
3      db    DSCR_DEVICE    ;; Typo de descriptor
```

```
4     dw    0002H        ;; Versión de la norma (BCD)
5     db    00H          ;; Clase de Dispositivo
6     db    00H           ;; Sub-Clase de Dispositivo
7     db    00H           ;; Sub-sub-Clase de dispositivo
8     db    64           ;; Tamaño máximo de paquete
9     dw    0B404H        ;; Identificador de Vendedor (Cypress)
10    dw    0310H        ;; Identificador de Producto (Sample Device)
11    dw    0000H        ;; Identificador de versión del producto
12    db    1           ;; Indice de Fabricante en string
13    db    2           ;; Indice de Producto en string
14    db    0           ;; Indice de número de serie en string
15    db    1           ;; Número de configuraciones
```

El descriptor de tamaño máximo de paquete está referido especialmente al EP0, es decir, al extremo que el host y el dispositivo utilizan para intercambiar mensajes de control.

Se debe notar que los penúltimos tres parámetros mostrados corresponden a una descripción realizada en cadena de caracteres al final del archivo, a fin de poder mostrar un mensaje que pueda ser leído por un usuario humano.

El último parámetro está relacionado con el número de configuraciones que posee este dispositivo. Esto determina también la cantidad de descriptores de configuración que debe tener el archivo de descripción. No obstante, antes de comenzar con el descriptor de configuración, se debe especificar el descriptor Calificador de Dispositivo, el cual dá información sobre otras velocidades de operación. Este descriptor es necesario debido a que el sistema implementado cumple con la versión 2.0 de la norma USB.

```
1  org (($ / 2) +1) * 2
2  DeviceQualDscr:
3     db    DSCR_DEVQUAL_LEN    ;; Largo del descriptor
4     db    DSCR_DEVQUAL    ;; Tipo de descriptor
5     dw    0002H        ;; Versión de la norma (BCD)
6     db    00H          ;; Clase de dispositivo
7     db    00H           ;; Sub-clase de dispositivo
8     db    00H           ;; Sub-sub-clase de dispositivo
9     db    64           ;; Tamaño máximo de paquetes
10    db    1           ;; Número de comunicaciones
11    db    0           ;; Reservado
```

Debido a la implementación hecha por Cypress, cada descriptor debe encontrarse en una dirección de memoria par. Por ello se recurre al comando de la línea 1 del código de la sección anterior. A continuación, se especifica el descriptor de Calificador de Dispositivo, con información muy similar al descriptor de Dispositivo.

Luego, se procede a detallar cada una de las configuraciones y sus Interfaces que se indicaron en los descriptores anteriores. En este caso, se deben especificar dos: una de alta velocidad, indicada en el Descriptor de Dispositivo y otra de velocidad completa, indicada en el Calificador de Dispositivo. Se muestra en seguida el Descriptor de la configuración de alta velocidad, unido al Descriptor de Interfaz (línea 16 en adelante) y los Descriptores de Extremos a partir de la línea 27, que determinan la configuración total de Alta Velocidad.

```
1  org ((($ / 2) +1) * 2
2  HighSpeedConfigDscr:
3      db    DSCR_CONFIG_LEN      ;; Largo del descriptor
4      db    DSCR_CONFIG          ;; Tipo de descriptor
5      db    (HighSpeedConfigDscr_End-HighSpeedConfigDscr) mod 256
6                                 ;; Largo total (LSB)
7      db    (HighSpeedConfigDscr_End-HighSpeedConfigDscr)  /  256
8                                 ;; Largo total (MSB)
9      db    1        ;; Número de interfaces
10     db    1        ;; Indice de configuración
11     db    0        ;; String de configuración
12     db    80H      ;; Atributos (b7->1, b6 - selfpwr, b5 - rwu)
13     db    50       ;; Consumo de potencia (div 2 ma)
14
15     ;; Alt Interface 0 Descriptor - Bulk IN
16     db    DSCR_INTRFC_LEN      ;; Largo del descriptor
17     db    DSCR_INTRFC          ;; Tipo de decriptor
18     db    0                    ;; Indice de interfaz
19     db    0                    ;; Indice de ajuste alternativo
20     db    2                    ;; Número de extremos
21     db    0ffH                 ;; Clase de interfaz
22     db    00H                  ;; Sub-clsae de interfaz
23     db    00H                  ;; Sub-sub-clase de interfaz
24     db    0                    ;; Indice de string descriptr de interfaz
25
26     ;; Isoc IN Endpoint Descriptor
27     db    DSCR_ENDPNT_LEN      ;; Largo del descriptor
28     db    DSCR_ENDPNT          ;; Tipo de descriptor
29     db    82H                  ;; Extremo de entrada EP2
30                                ;; b7 -> IN/OUT, b[4:0] -> dir
31     db    ET_ISO               ;; Tipo de trasferencia
32     db    00H                  ;; Tamaño máximo de paquete (LSB)
33     db    02H                  ;; Tamaño máximo de paquete (MSB)
34     db    01H                  ;; Intervalo de consulta
35
36     ;; Bulk OUT Endpoint Descriptor
37     db    DSCR_ENDPNT_LEN      ;; Largo del descriptor
38     db    DSCR_ENDPNT          ;; Tipo de descriptor
39     db    08H                  ;; Extremo de salida EP8
40     db    ET_BULK              ;; Tipo de transferencia
41     db    00H                  ;; Tamaño máximo de paquete (LSB)
42     db    02H                  ;; Tamaño máximo de paquete (MSB)
43     db    00H                  ;; Intervalo de consulta
44
45 HighSpeedConfigDscr_End:
```

En la línea 12 del código anterior se debe colocar cuál es la fuente de la potencia que consume el dispositivos, es decir, de donde proviene la energía utilizada para el funcionamiento. El bit7 debe estar siempre establecido a 1 por razones históricas de la norma USB[27]. El bit6 en 1 define que el dispositivo está energizado por una fuente propia. En el caso contrario, toma potencia del bus. El bit5, por su parte, señala que el dispositivo tiene modo de baja energía y que es posible establecer el modo de funcionamiento de mayor consumo con un comando del host. La línea 13 se informa cuánta potencia consume, lo que le brinda al host la posibilidad de establecer un control de la potencia suministrada en el bus.

El último campo del descriptor de extremo, correspondiente al intervalo de consulta, se utiliza para establecer cada cuanto tiempo el host debe asignar ancho de banda para transferencias isocrónicas.

Luego de enviar la configuración de Alta Velocidad, se informa de la misma manera el descriptor de Velocidad Completa, cuyo código se observa a continuación.

```
1  org (($ / 2) +1) * 2
2  FullSpeedConfigDscr:
3      db    DSCR_CONFIG_LEN      ;; Largo del descriptor
4      db    DSCR_CONFIG          ;; Tipo de descriptor
5      db    (FullSpeedConfigDscr_End-FullSpeedConfigDscr) mod 256
6                                 ;; Largo total (LSB)
7      db    (FullSpeedConfigDscr_End-FullSpeedConfigDscr)  /  256
8                                 ;; Largo total (MSB)
9      db    1         ;; Número de interface
10     db    1         ;; Numero de configuraciones
11     db    0         ;; Indice de string de configuración
12     db    80H       ;; Atributos (b7 -<'1', b6 - selfpwr, b5 - rwu)
13     db    50        ;; Requerimiento de pontencia(div 2 ma)
14
15     ;; Interface Descriptor
16     db    DSCR_INTRFC_LEN ;; Largo del descriptor
17     db    DSCR_INTRFC     ;; tipo de descriptor
18     db    0               ;; Indice de interfaz
19     db    0               ;; Indice de ajuste alternativo
20     db    2               ;; Número de extremos
21     db    0ffH            ;; Clase de interfaz
22     db    00H             ;; Sub-clase de interfaz
23     db    00H             ;; Sub-sub-clase de interfaz
24     db    0               ;; Indice de string de interfaz
25
26     ;; Endpoint Descriptor
27     db    DSCR_ENDPNT_LEN    ;; Largo del descriptor
28     db    DSCR_ENDPNT        ;; Tipo de descriptor
29     db    82H                ;; Dirección y sentido de extremo
30     db    ET_ISO             ;; Tipo de extremo
31     db    0FFH               ;; Tamaño máximo de paquete(LSB)
32     db    03H                ;; Tamaño máximo de paquete(MSB)
```

```
33      db    01H                 ;; Intervalo de consulta
34
35      ;; Endpoint Descriptor
36      db    DSCR_ENDPNT_LEN     ;; Largo del descriptor
37      db    DSCR_ENDPNT         ;; tipo de descriptor
38      db    08H                 ;; Dirección y sentido de extremo
39      db    ET_BULK             ;; Tipo de extremo
40      db    040H                ;; Tamaño máximo de paquete (LSB)
41      db    00H                 ;; Tamaño máximo de paquete (MSB)
42      db    01H                 ;; Intervalo de consulta
43
44  FullSpeedConfigDscr_End:
```

Finalmente, el diseñador puede escribir todos los mensajes en formato de cadena de caracteres, para una lectura más sencilla por parte del usuario. En este trabajo solo se usan dos que sirven como ejemplo pero no se profundizó más en el estudio de estos mensajes debido a que no son relevantes para los objetivos.

```
1   org (($ / 2) +1) * 2
2   StringDscr:
3
4   StringDscr0:
5       db    StringDscr0_End-StringDscr0     ;; Largo del descriptor
6       db    DSCR_STRING                     ;; Tipo de descriptor
7       db    09H,04H
8   StringDscr0_End:
9
10  StringDscr1:
11      db    StringDscr1_End-StringDscr1     ;; Largo del descriptor
12      db    DSCR_STRING                     ;; Tipo de descriptor
13      db    'E',00                          ;; Mensaje
14      db    'd',00
15      db    'w',00
16      db    'i',00
17      db    'n',00
18      db    '␣',00
19      db    'B',00
20      db    'a',00
21      db    'r',00
22      db    'r',00
23      db    'a',00
24      db    'g',00
25      db    'a',00
26      db    'n',00
27  StringDscr1_End:
28
29  StringDscr2:
```

```
30      db    StringDscr2_End-StringDscr2     ;; Largo del descriptor
31      db    DSCR_STRING                     ;; Tipo de descriptor
32      db    'L',00                          ;; Mensaje
33      db    'a',00
34      db    '-',00
35      db    'T',00
36      db    'e',00
37      db    's',00
38      db    'i',00
39      db    's',00
40  StringDscr2_End:
```

1.4. Sumario del capítulo

En el presente capítulo se explicaron las herramientas provistas por Cypress a través de su Kit de Desarrollo CY3684 EZ-USB FX2LP. Este kit consiste en una parte física cuya paarte más destacada es el controlador USB FX2LP y una parte informática consistente en un set de herramientas que permiten la programación del controlador provisto.

También se fundamentó y se explicó en forma detallada la configuración seleccionada y la implementación de esta configuración en el código desarrollado.

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

[31] Cypress Semiconductor, "CY3684/CY3684 EZ-USB Development Kit User Guide," tech. rep., 2014.

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