Índice general

Yo redactaría de nuevo esa oración: En nucleo del kit de desarrollo(Que es un kit de desarrollo???) Cypres FX es un circuito integrado cy.. el mismo es un controlador USB de la serie FX.
Hya que citar las hojas de datops de todo.

Esta oración es muy corta, la iunegraría al texto de otra forma

Que son los perifericos con "autonomia limitada"?

Cuando se usan palabras en otro idioma hay que utilizar italics

Capítulo 1

Supongo que esto lo pone asi porque lo compilaste sin lo anterior.

Programación y configuración del Controlador USB

En el principio del cada capítulo se deberia escribir una breve introduccion explicando de que se trata el mismo, y al final de cada capítulo de debería escribir una breve conclusión del mismo

1.1. Arquitectura FX2LP EZ-USB

Del kit de desarrollo (Cypress??)....

El núcleo del Kit de Desarrollo FX2LP EZ-USB es un CY7C68013A. Este circuito integrado es un controlador USB y pertenece a la serie FX2LP del catálogo de integrados EZ-USB comercializado por Cypress Semiconductors. Su arquitectura se presenta en la Figura 1.1.

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 con autonomía limitada. Se integra un controlador USB completo. Esto incluye un transceptor USB, un Motor de Interfaz Serie (MIS), buffers de datos configurables, un microcontrolador 8051 mejorado 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 de los cuales proveen las señales de reloj adecuadas para el correcto funcionamiento del sistema.

a traves del cual se ...

Para que se entienda hay que explicar que hace cada uno de los módulos.

anfitrion a traves del mismo

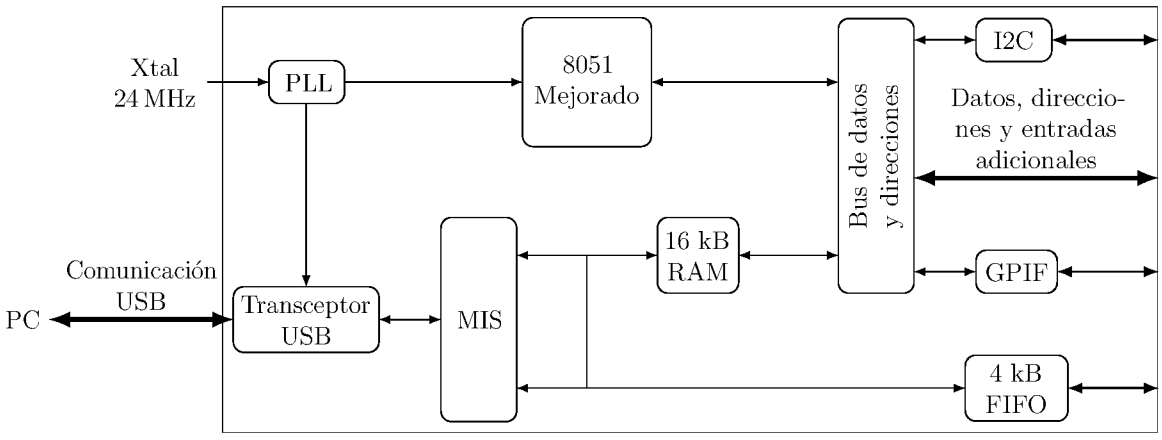

Figura 1.1: Arquitectura FX2LP

Arquitectura del controlador USB Cypress FX2LP

Habria que colocar un texto antes de colocar las variantes porque queda poco claro esto

puerto USB, o bien via RS-232. Para comunicarse con sistemas perifericos se puede aprovechar el puerto I^2C, la interfaz de proposito general, que actua como maestro y a la cual se le puede acoplar un periferico esclavo, y/o las memorias FIFO en modo esclavo que puede ser conectada a un sistema maestro. Esto brinda muchas alternativas de conexion, desde puertos estandar, como ser ATA, PCMCIA, EPP, etc. hasta dispositivos personalizables como DSP's y FPGA's.

variante 2

El flujo de datos posee dos puntas entre las cuales el controlador hace de nexo. Para ello necesita poder comunicarse tanto con el HOST como con los perifericos.

El intercambio de informacion con el HOST se lleva a cabo a traves del mismo puerto USB, objetivo principal de este trabajo. Sin embargo, tambien posee dos puertos UART que facilitan en gran medida la tarea de depuracion del desarrollo.

En cuanto a la interfaz con uno o mas perifericos, el controlador posee un puerto I^2C, una interfaz de proposito general (GPIF), para sistemas que necesitan ser comandados en forma externa; y una interfaz con memorias FIFO esclavas, a traves de las cuales se puede conectar sistemas que cumplen un rol activo en el envio y recepcion de informacion.

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 plasmarán en el código del firmware.

1 de Interfaz Serial

La comunicación USB entre el controlador FX2LP y la PC se realiza a través del transceptor, unido al MIS. Con el objetivo de intercambiar 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 los buffers de extremos, agrega todo el encabezado y 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.

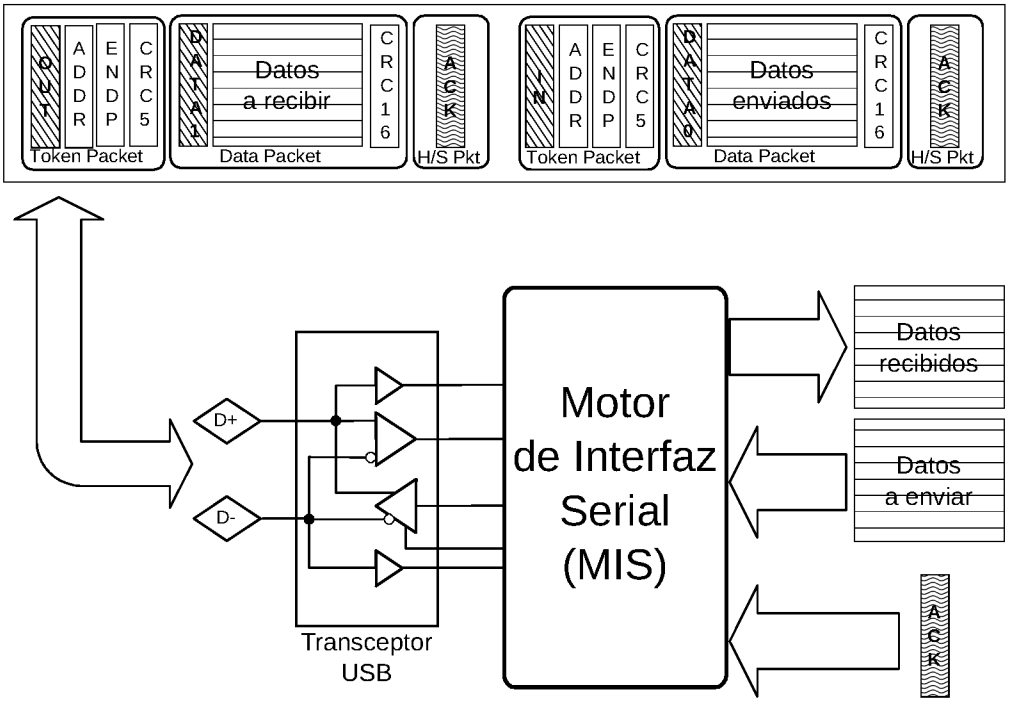

Figura 1.2: Implementación del enlace USB realizado por el EZ-USB (se copiaria con tikz para mejorar prolijidad)

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.

extremo(End Point)

La norma USB define a un dispositivo extremo como una porción exclusiva e identificable de un 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 se escribe 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's 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, anfitrión USBy dispositivo intercambiarán solo transferencias de control.

Luego, se incorpora un EP1, que posee dos buffers fijos, o sea no configurables, de 64 bytes, uno como entrada y el otro como salida.

Finalmente, 4 KiB de memoria debe ser configurada para los EP2, EP4, EP6 y EP8. La configuración de los EP la realiza el firmware en tiempo de ejecución. Las variables, conforme a

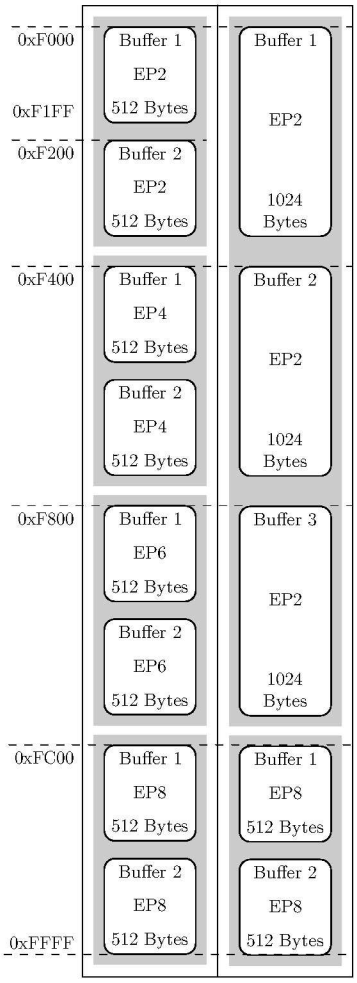

Figura 1.3: Buffers de extremos con sus direcciones de memoria. El cuadro de la izquierda muestra la configuración por defecto. El derecho, la implementada en este trabajo.

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 EPs paquetes de control.
- 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 EPs. A la izquierda se observa la configuración por defecto del controlador FX2LP. Esto es, los cuatro EP's 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 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 hay

Redactar de nuevo

que considerar que no se dispone más que 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 el sistema mismo quien se encarga de seleccionar el buffer en uso. Esto quiere decir que, por ejemplo, teniendo 4 buffers de 512 bytes cada uno, el 8051 verá solo uno de 512 bytes, sin necesidad de identificar con cuál de los cuatro está trabajando.

Que es un "punto de vista digital"?

1.1.3. Memorias FIFO esclavas

Desde un punto de vista digital, el MIS recibe y envía datos desde y hacia el puerto USB. Para ello utiliza un cristal de 24 MHz. Por su parte, un sistema externo puede o no proveer una señal de reloj y manejo de datos propio. 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, al controlador, 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.

Cual modo de funcionamiento?

Para los fines del presente trabajo, este modo de funcionamiento es óptimo ya que, dotando al FPGA de una máquina de estados mínima, 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 una memoria FIFO para cada uno de los EP programables en el buffer de extremos.

La Figura 1.4 muestra la interfaz entre las memorias FIFO's y un maestro esclavo. Estos 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 el se selecciona la memoria activa en el bus.

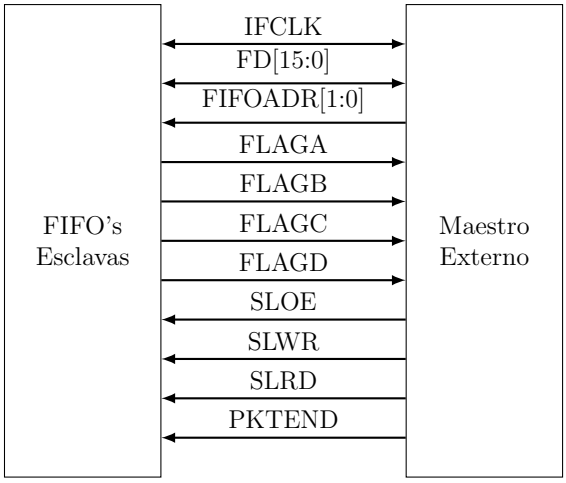

Figura 1.4: Puertos de interfaz entre las FIFO's y un maestro externo

- FLAGx: Los cuatro puertos de flag son configurables e indican memoria llena, vacía o un nivel programable. También pueden indicar el estado sobre una memoria en particular o sobre la activa.
- 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

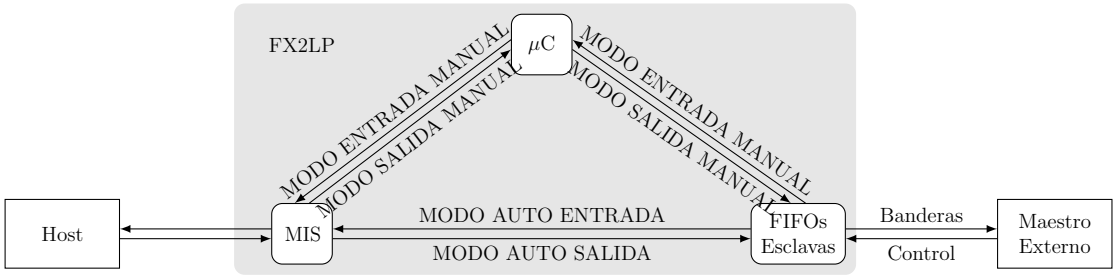

Figura 1.5: Modos de conexión de la memoria FIFO, el microntrolador y el MIS

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 anfitrión y de salida cuando llevan el sentido contrario. Esto se debe al carácter *anfritión-céntrico* de la comunicación USB, en donde el principal componente es el anfitrión USB y a él se acoplan los diferentes dispositivos.

1.2. Kit de desarrollo de Software de Cypress

Dentro del kit de desarrollo avanzado EZ-USB DVK, Cypress provee un amplio conjunto de herramientas que facilitan la implementación del software a la persona que lo produce. Cypress denomina, a estas soluciones, Kit de Desarrollo de Software (o SDK, ~~acrónimo del habla inglesa~~, *Software Development Kit*).

El kit abarca una gran cantidad de aspectos, como ser la elaboración del firmware implementado en el 8051 del controlador, la elaboración de drivers, la conversión de archivos de programación y ejecutables que graban la información en los diferentes chip, ya sea en la RAM del microcontrolador o en EEPROM, para cargar programas no volátiles que posteriormente serán ejecutados por el 8051. microprosesador 8051

Debido a que no todo se utiliza en este trabajo, a continuación se desarrollarán las herramientas 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 codigo 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 anfitrión USB, 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

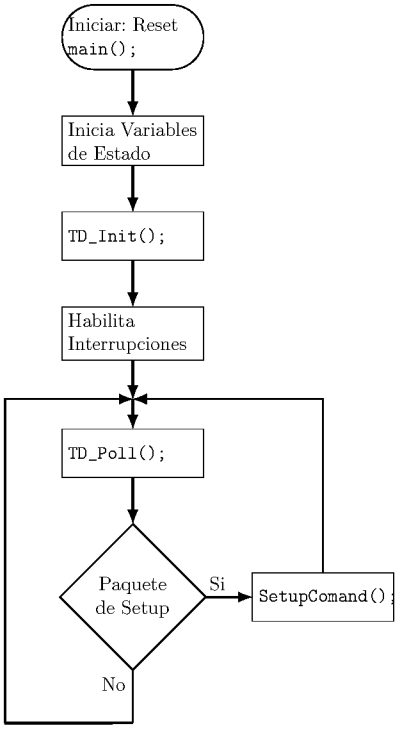

Figura 1.6: Diagrama en bloques simplificado

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 funcion `TD_Init()`, a través de la cual el usuario programa e inicia la configuración del sistema. Luego, es necesario habilitar todas las interrupcionnes necesarias y finalmente, se invoca repetidamente la función `TD_Poll()`, en donde el usuario escribe las tares que ejecutará el 8051.

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 3684 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, las pruebas, el depurado y el compilado del mismo.

El programa utilizado posee, entre otras cosas, editor de textos con atajos de teclado, comandos que aceleran la escritura de código y resaltador 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 por 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

Dentro del kit de desarrollo de software que brinda Cypress se encuentra el programa USB Control Center. Este software se utiliza durante este trabajo para cargar un programa compilado en el controlador FX2LP, para transmitir y recibir mensajes.

La utilidad de este programa radica en la 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
```

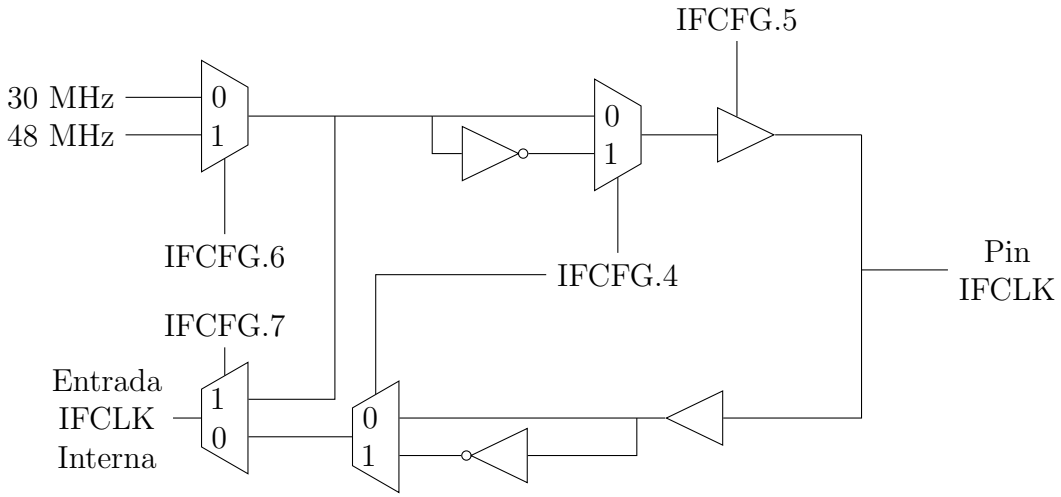

Figura 1.7: Esquema funcional para la entrada de reloj de la Interfaz

;

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:

```
//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*.

Se observa a continuación los descriptores usados en el presente trabajo.