# Proyecto de Grado

## Interfaz USB genérica para comunicación con dispositivos electrónicos

## Estado del Arte

A/C Andrés Aguirre, A/C Pablo Fernández y A/C Carlos Grossy

Tutores: MSc Ing. Gonzalo Tejera y MSc Ing. Alexander Sklar

Instituto de Computación - Facultad de Ingeniería

Universidad de la República Oriental del Uruguay

14 de diciembre de 2007

# Índice general

# Índice de figuras

# Índice de cuadros

# Parte I

# Tecnología USB

# Capítulo 1

# Introducción al USB

A continuación se presentan las caracteristicas más relevantes sobre la tecnología USB relevadas de la bibliografía consultada [2, 3, 21].

## 1.1. Motivación:

Muchas PCs diseñadas hoy en día aún implementan periféricos con interfaces basadas en el IBM PC original diseñado a comienzos de los ochenta. Esta implementación tiene numerosas desventajas que causan a diseñadores y a usuarios muchas frustraciones. USB emerge como el resultado de las dificultades asociadas al costo, configuración y posibilidad de conexión de un dispositivo periférico en el entorno de las computadoras personales.

**Recursos Limitados del sistema.** La figura 1.1 ilustra la herencia del paradigma de E/S donde los dispositivos periféricos eran típicamente mapeados en espacio de direcciones de E/S del CPU y se asignaba una línea de IRQ especifica, y un canal de DMA en algunos casos. Estos recursos del sistema eran asignados a dispositivos particulares por IBM y otros fabricantes y se convertían en direcciones de E/S, IRQs y canales de DMA estándar usados por desarrolladores de software para acceder a un dispositivo en particular. Otra limitación de este enfoque es el limitado número de dispositivos periféricos que puede ser conectado a los conectores estándar. Por ejemplo, el conector serial y el paralelo soportan solo un dispositivo, limitando de esta manera el número de dispositivos que pueden ser conectados de forma sencilla y barata.

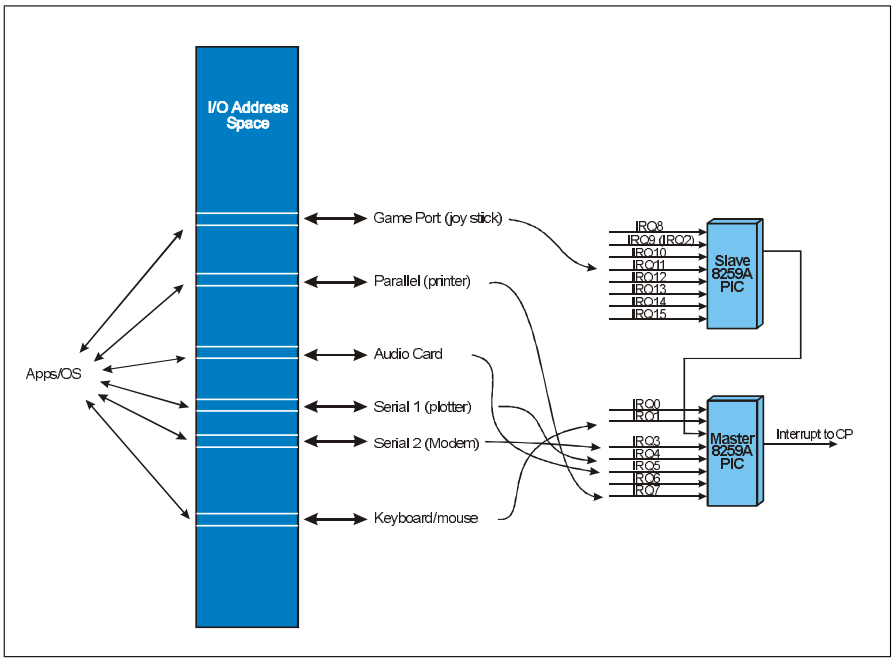



Figura 1.1: E/S Mapeada

**Interrupciones** Quizás el problema más crítico acerca de los recursos del sistema, gira alrededor de la asignación de interrupciones requeridas por la gran cantidad de dispositivos que se ponen en ejecución típicamente en las PC. Esto es particularmente cierto en los dispositivos periféricos que se conectan mediante el BUS de ISA, puesto que el bus ISA no apoya confiablemente interrupciones compartidas. El cuadro 1.1 enumera cada línea de IRQ y los dispositivos que la utilicen típicamente. Como puede verse, muchas de las líneas de IRQ se dedican a los dispositivos particulares basados en convenciones de la herencia, mientras que otras líneas de IRQ se pueden utilizar por una variedad de dispositivos periféricos.

**Direcciones de E/S:** Los conflictos por direcciones de E/S son muy comunes en los PCs. Los dispositivos periféricos usualmente requieren un bloque de direcciones de E/S para reportar información del estado y emitir comandos al dispositivo.

**Interfaces no compatibles:** Las interfaces estándar usadas por los dispositivos típicos (conexiones seriales y paralelas) soportan la conexión de un solo dispositivo. Como solo un dispositivos puede ser conectado por vez, la flexibilidad de estas conexiones es minimizada, esto trae como consecuencia la costosa decisión de construir una tarjeta de expansión para conectar al bus de expansión (ISA o PCI) para crear un punto de conexión para el diseño de un periférico nuevo.

**Preocupaciones del lado del usuario:** El usuario se enfrenta a varios problemas a la hora de conectar un dispositivo, como ser:

- Demasiados conectores.
- El sistema debe ser apagado para conectar la mayoría de los periféricos.
- El sistema debe ser reiniciado para instalar o cargar software.
- Costo.

| Línea de IRQ | Dispositivo |
|---|---|
| IRQ0 | Timer del sistema. |
| IRQ1 | Teclado. |
| IRQ2 | Canal para conectar en cascada con un controlador de interrupciones esclavo. |
| IRQ3 | Mouse serial, modem, plotter, impresora serial, puerto de joystick, lápiz, puerto infrarrojo. |
| IRQ4 | Mouse serial, modem, plotter, impresora serial. |
| IRQ5 | Bus mouse, impresora paralela, tarjeta de sonido, adaptador de LAN, controlador de cinta, puerto de joystick. |
| IRQ6 | Disquetera. |
| IRQ7 | Impresora paralela. |
| IRQ8 | Alarma RTC. |
| IRQ9 | Adaptador LAN, adaptador de vídeo, controlador de cinta, puerto de joystick |
| IRQ10 | Adaptador de LAN, tarjeta de sonido. |
| IRQ11 | Adaptador de LAN, controlador SCSI, controlador PCMCIA. |
| IRQ12 | Mouse PS/2, controlador PCMCIA. |
| IRQ13 | Errores de coprocesador numérico. |
| IRQ14 | Disco duro. |
| IRQ15 | Controlador SCSI, controlador PCMCIA. |

Cuadro 1.1: Asignación de IRQs

**El paradigma USB:** Los objetivos de diseño de un nuevo estándar deben solucionar, los defectos percibidos por los fabricantes y usuarios, deben proveer crecimiento a futuro, performance, y expansión. Los objetivos de diseño de USB incluyen:

- Un solo tipo de conector para todos los periféricos.
- Habilidad para conectar varios dispositivos periféricos al mismo conector.
- Un método para facilitar los conflictos por recursos.
- Conexión en caliente.
- Detección y configuración automática de los periféricos.
- Bajo precio para la implementación del sistema y los periféricos.
- Aumento en la capacidad de performance.
- Soporte para conexión de nuevos diseños de periféricos.
- Soporte para hardware y software legado.
- Implementación de bajo consumo energético.

USB rompe con los problemas asociados con la implementación de la E/S legada. Las restricciones relacionadas con el espacio de E/S, líneas de IRQ, y canales de DMA no existen más en la implementación de USB. Cada dispositivo que reside en el USB tiene asignado una dirección solo conocida por el subsistema USB y no consume ningún recurso del sistema. USB soporta hasta 127 dispositivos simultáneamente mediante la utilización de HUB. Los dispositivos USB contienen números de registros individuales o puertos que pueden ser accedidos indirectamente por los controladores de dispositivo USB. Estos registros son conocidos como los endpoints USB.

Cuando una transacción es enviada en el USB, todos los dispositivos (exceptuando los de baja velocidad) van a ver la transacción. Cada transacción comienza con la trasmisión de un paquete que define el tipo de transacción a ser realizada por el dispositivo USB y la dirección del endpoint. Esta dirección es manejada por el software USB. Cada dispositivo USB debe tener una dirección interna por defecto (llamada endpoint cero) que es reservada para configurar el dispositivo. Vía el endpoint cero, el software del sistema USB lee los descriptores estándar del dispositivo. Estos

descriptores proveen la información de configuración necesaria para la inicialización del software y el hardware. De esta manera el software del sistema puede detectar el tipo de dispositivo (o información de la clase) y determinar como el dispositivo se intenta acceder.

**Velocidad:** El bus serial universal (USB) crea una solución para conectar periféricos a una PC balanceando performance y costo. USB soporta tres tipos de velocidades de transmisión.

1. 1.5 Mbps (low speed)
2. 12 Mbps (full speed)
3. 480 Mbps (high speed)

La versión 1.0 y la 1.1 (1.x) de USB soportan solo las velocidades 1.5 Mbps y 12 Mbps, la versión 2.0 de la especificación de USB define una velocidad de 480 Mbps.

**Soporte Plug and Play en caliente:** USB puede detectar la conexión de un nuevo periférico y automáticamente instalar el software necesario para acceder al dispositivo. Este proceso también elimina la necesidad de setear switches y jumpers cuando se está configurando un dispositivo periférico y elimina la necesidad de reiniciar el sistema cuando el periférico es conectado.

**Expansión:** Los dispositivos HUBs proveen puertos adicionales al ser conectados a un puerto USB. Estos pueden ser dispositivos independientes o integrados a otros como impresoras o teclados.

**Sin necesidad de alimentación de energía:** La interface USB incluye, líneas de alimentación y tierra que proveen un valor nominal de +5V. Un periférico que requiere hasta 500 miliamperios puede tomar todo su poder desde el bus, en lugar de tener que proveerle una fuente externa de poder.

**Versatilidad:** Los cuatro tipos de transferencias de USB y los tres tipos de velocidad, hacen a la interfase apta para muchos tipos de periféricos. Hay tipos de transferencia aptos para intercambiar bloques de datos largos y pequeños, con y sin restricciones de tiempo. Para datos que no pueden tolerar retardos, USB puede garantizar ancho de banda o tiempo máximo entre transferencias. Aunque el sistema operativo, los controladores de dispositivo, y el software de aplicación pueden introducir retardos inevitables, USB hace sencillo y posible de alcanzar transferencias que están cerca del tiempo real.

A diferencia de otras interfaces, USB no asigna funciones especiales a las líneas de señal o hace otras suposiciones acerca de cómo la interfase va a ser usada, de esta manera es adecuada para cualquier tipo de periférico.

Para comunicarse con periféricos comunes como impresoras, teclados, y dispositivos de almacenamiento, USB ha definido clases que especifican los requerimientos y protocolos. Los desarrolladores pueden usar estas clase como guía en lugar de reinventar todo desde el principio.

**Soporte Periférico:** Del lado del periférico, cada dispositivo hardware USB debe incluir un chip controlador que maneje los detalles de la comunicación USB. Algunos controladores son microcontroladores completos que incluyen una CPU, memoria de datos y de programa, e interface USB. Otros controladores deben comunicarse mediante una interface a un CPU externo que se comunica con el controlador USB según sea necesario.

El periférico es responsable de responder pedidos de envío y recepción de datos usados en identificar y configurar el dispositivo para lectura y escritura de otros datos en el bus. En algunos controladores, algunas funciones son microprogramadas en el hardware y no necesitan ser programadas.

## 1.2. Puntos débiles de USB:

Toda interfase tiene sus limitaciones que la hacen impráctica para algunas aplicaciones. Para USB, los límites a tener en cuenta son la velocidad y la distancia, la falta de soporte para comunicaciones peer-to-peer, inhabilidad para broadcast, y falta de soporte en hardware y sistemas operativos viejos.

**Velocidad:** USB es versátil, pero no esta diseñado para hacer todo. La gran velocidad de USB lo hacen competitivo con IEEE-1394a (Firewire) de 400 Mbps, pero IEEE-1394b es aun más rápido, a 3.2 Gbps.

**Distancia:** USB fue diseñado como una expansión de escritorio para el bus, con la hipótesis de que los periféricos iban a estar relativamente cerca de la mano. Un segmento de cable puede ser como máximo de 5 metros. Otras interfaces, incluyendo el RS-232, RS-485, IEEE-1394b, y Ethernet, permiten cables mucho más largos. Se puede incrementar el largo de un link USB como máximo a 30 metros usando cables que unen cinco hubs y un dispositivo

**Comunicación Peer-to-Peer:** Toda comunicación es entre una computadora host y un periférico. El host es un PC u otra computadora con hardware de controlador de host. El periférico contiene el hardware de controlador de dispositivo. Los hosts no pueden hablar uno al otro directamente, y los periféricos tampoco pueden hablar uno al otro directamente. Otras interfaces, como IEEE-1394, permiten comunicación directa entre periféricos. USB provee una solución parcial con USB On-The-Go. Un dispositivo On-The-Go puede funcionar como un periférico y como un host de capacidad limitada, que puede comunicarse con otros dispositivos.

**Broadcasting:** USB no provee un mecanismo para enviar un mensaje simultáneamente a múltiples dispositivos en el bus. El host debe enviar el mensaje a cada dispositivo de manera individual. Si se necesita capacidad de broadcasting, se debe usar IEEE-1394 o Ethernet.

**Complejidad del Protocolo:** En contraste, algunas interfaces viejas permiten conectar circuitos muy simples con protocolos muy básicos. Por ejemplo, el puerto paralelo de la PC original es solo una serie de entradas y salidas digitales. Se puede conectar circuitos de entrada, salida básicos sin necesidad de inteligencia computacional del lado del periférico. El software de la PC, puede monitorizar y controlar los bits individuales en los puertos.

Con Aplicaciones USB, no se puede solo leer y escribir a direcciones de un puerto, y los dispositivos no pueden solo presentar una serie de entradas y salidas para leer y escribir directamente. Para acceder a un dispositivo USB, las aplicaciones se deben comunicar con una clase o un driver de dispositivo que en turnos se comunica con el driver USB de bajo nivel que maneja la comunicación en el bus. El dispositivo debe implementar los protocolos que habilitan la detención del PC, identificación y comunicación con el dispositivo.

## 1.3. Evolución de la interface

**USB Original:** La versión 1.0 de la especificación USB fue lanzada en enero de 1996. La versión 1.1 data de septiembre de 1998. USB 1.1 agrego un nuevo tipo de transferencia (interrupt OUT). En abril del 2000 se lanzo el USB 2.0 que agrego la opción de high speed.

Engineering Change Notices (ECNs) contienen revisiones y agregados a la especificación, incluyendo la definición de un nuevo conector mini-B.

**USB 2.0:** Mientras USB 1.X ganaba popularidad, era claro que un bus de mayor velocidad iba a ser útil. Investigaciones mostraron que una velocidad de bus cuarenta veces más rápida que la full speed podía mantener la compatibilidad hacia atrás con las interfases low y full-speed. El soporte para un bus de 480 Mbps hicieron a USB mucho más atractivo para periféricos como escaners, dispositivos de almacenamiento, y video.

**USB on-the-go:** Mientras USB se convertía en la interface de elección para todos los tipos de periféricos, los desarrolladores se comenzaron a preguntar por una forma de conectar sus periféricos directamente entre ellos y entre otros periféricos USB. Por Ejemplo, un usuario puede querer conectar una cámara directamente a una impresora, o dos unidades de almacenamiento para intercambiar archivos. El suplemento OTG para la especificación 2.0 de USB, fue lanzado en el 2001 y define una funcionalidad limitada de host que los dispositivos pueden implementar para comunicarse con periféricos.

**Wireless USB:** Se esta desarrollando una especificación, para conexión inalámbrica sobre USB hasta una velocidad de 480 Mbps.

## 1.4. Comparación con tecnologías relacionadas

**USB versus IEEE-1394:** Otra opción popular para nuevos periféricos es la IEEE-1394. La implementación de Apple a esta interface es llamada Firewire. Generalmente IEEE-1394 es más rápido y más flexible que USB pero es más caro de implementar. Con USB, un solo host controla la comunicación con muchos dispositivos. El host maneja la mayor parte de la complejidad, por tanto la electrónica del dispositivo puede ser relativamente simple y barata. Los dispositivos IEEE-1394 pueden comunicarse con los otros directamente, y una única comunicación puede ser dirigida a múltiples receptores. El resultado es una interface más flexible, pero la electrónica de los dispositivos es más compleja y cara. IEEE-1394 se ajusta mejor para aplicaciones que requieren comunicaciones extremadamente rápidas, o broadcasting para muchos receptores. USB se ajusta mejor para periféricos comunes como teclados, impresoras, y escaners, como también aplicaciones de velocidad baja a moderada y sensibles a los costos.

**USB versus Ethernet:** La ventajas de Ethernet incluyen la habilidad de usar cables muy largos, habilidad de broadcast, y soporte para protocolos de Internet. Sin embargo el hardware requerido para soportar Ethernet es más complejo y caro que el hardware típico de los periféricos USB. USB es también más versátil con cuatro tipos de transferencias y una variedad de clases definidas para diferentes propósitos.

## 1.5. Arquitectura, topología, componentes y conceptos

**Componentes del BUS:** El host es una PC u otra computadora que contiene un controlador host USB y un hub root. Estos componentes trabajan juntos para permitir al sistema operativo comunicarse con los dispositivos en el bus. El controlador de host da formato a los datos para trasmitir en el bus y traduce los datos recibidos a un formato que los componentes del sistema operativo pueden entender. El controlador de host también realiza otras funciones relacionadas con el manejo de comunicaciones en el bus. El hub root tiene uno o más conectores para conectar dispositivos. El hub root, en combinación con el controlador de host, detecta dispositivos conectados y desconectados, lleva a cabo pedidos del controlador de host, y pasa datos entre dispositivos y el controlador host.

**Topología:** La topología en el bus es de tipo tiered star. En el centro de cada estrella está un hub. Cada punto en una estrella es un dispositivo que conecta con un puerto en un hub, como puede verse en la figura 1.2.

La topología de tipo tiered star describe solamente las conexiones físicas. En el momento de programación, lo que importa es la conexión lógica. Para comunicarse, el host y el dispositivo no necesitan saber o preocuparse por cuántos hubs pasa la comunicación. Solamente un dispositivo a la vez puede comunicarse con un regulador del host.

**Definiendo términos:** Junto con el host, definido anteriormente como la computadora que controla la interface, otros tres también son importantes estos son: función, Hub, y dispositivo. Es también trascendente entender el concepto de un puerto del USB y cómo se diferencia de otros puertos tales como RS-232.

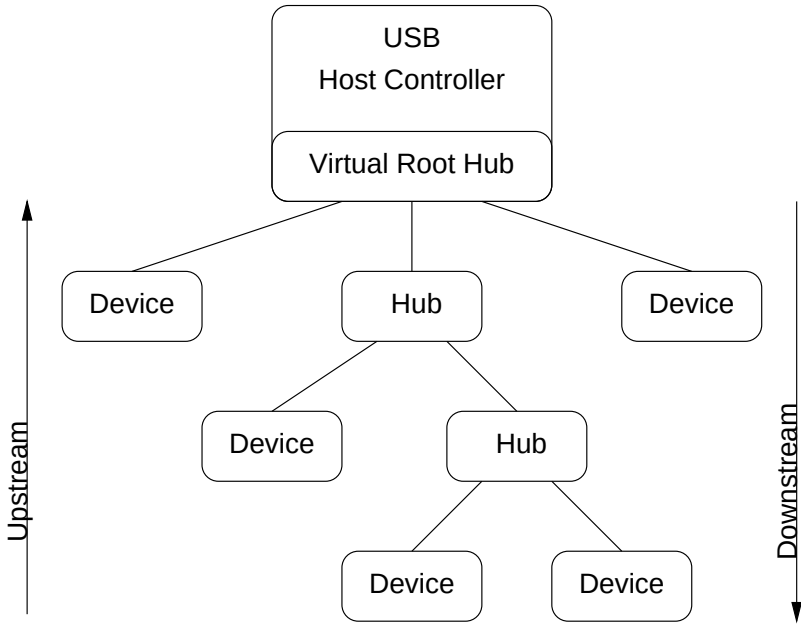



Figura 1.2: Topología USB

1. Función:

   La especificación de USB define una función como un dispositivo que provee una capacidad al host. Ejemplos de funciones son un mouse, un par de parlantes.

2. Hub:

   Un hub tiene un conector upstream para comunicarse con el host y uno o más conectores downstream o conexiones internas para dispositivos.

3. Dispositivo (Device):

   La especificación de USB define que un dispositivo es una función o un hub, a excepción del caso especial de un dispositivo compound, que contiene un hub y una o más funciones. Cada dispositivo en el bus tiene una dirección única, a excepción de nuevo de los dispositivos compound en los que el hub y las funciones tienen direcciones únicas. Los dispositivos composite son multi función con múltiples interfaces independientes.

4. Puerto:

   En sentido general un puerto es una entidad direccionable que esta disponible para conectar circuitos adicionales.

   Los puertos USB difieren de muchos otros puertos, porque todos ellos comparten un único camino hacia el host y no son direccionables directamente. Cada controlador de host maneja un único bus, o camino de datos. Cada conector en un bus, representa un puerto USB. A diferencia de RS-232 donde todos los dispositivos comparten el ancho de banda del bus. En USB sólo un dispositivo o el host, puede trasmitir en un momento dado.

**División del trabajo:** El host y sus dispositivos tienen definido responsabilidades. El host lleva la mayor parte del manejo de la carga de comunicaciones, pero un dispositivo debe tener la inteligencia de responder a las comunicaciones y otros eventos del bus del host y el hub al cual el dispositivo esta unido.

#### Las tareas del host

El host esta a cargo del bus, este debe saber que dispositivos se encuentran en el bus y las capacidades de cada uno. También debe hacer todo lo posible para asegurarse que todos los dispositivos en el bus puedan enviar y recibir información

cuando lo necesiten. Un bus puede tener muchos dispositivos, cada uno con diferentes requerimientos, y todos queriendo transferir datos al mismo tiempo.

Las aplicaciones no necesitan preocuparse de los aspectos específicos de la comunicación USB con dispositivos. Todo lo que debe hacer la aplicación es enviar y recibir datos usando las funciones estándar del sistema operativo accesibles mediante los lenguajes de programación. Muchas veces la aplicación no tiene que saber o preocuparse si el dispositivo usa USB u otra interface.

1. Detección de dispositivos:

   En el ciclo inicial, el hub hace que el host se de cuenta de todos los dispositivos USB conectados. En un proceso llamado enumeración, el host asigna una dirección y solicita información adicional de cada dispositivo. Luego del ciclo inicial, cada vez que un dispositivo es removido o conectado, el host aprende del evento y enumera cualquier nuevo dispositivo conectado y remueve los dispositivos desconectados de su lista de dispositivos disponibles para las aplicaciones.

2. Manejo del flujo de datos:

   Varios dispositivos pueden querer transferir datos al mismo tiempo. El controlador de host, divide el tiempo disponible en segmentos llamados frames y microframes y le da a cada transmisión una porción de un frame o microframe.

3. Detección de Errores

4. Suministro de Energía

5. Intercambio de Datos con Periféricos

**Las tareas del periférico:**

Las tareas del periférico son un espejo de las del host. Un periférico no puede iniciar una comunicación por si solo. En cambio este debe esperar y responder a una comunicación del host ( una excepción es la característica de remote wakeup, que le permite al periférico solicitar comunicación del host)

1. Detectar comunicaciones dirigidas al chip:

   Cada dispositivo monitorea la dirección de dispositivo contenida en cada comunicación en el bus. Si la dirección no coincide con la almacenada por el dispositivo, el dispositivo ignora la comunicación. Si la dirección coincide, el dispositivo almacena los datos en su buffer receptor y lanza una interrupción para indicar que han llegado datos. En casi todos los chips, estas funciones están incorporadas dentro del hardware y no requieren soporte en el código. El firmware no tiene que hacer acciones o tomar decisiones hasta que el chip ha detectado una comunicación conteniendo la dirección del dispositivo.

2. Detección de errores:

   Como el host, un dispositivo agrega bits para el control de errores de los datos que envía. Al recibir datos que incluyen bits de control de error, el dispositivo hace los cálculos para detectar errores. La respuesta o la ausencia de esta, le informan al host cuando retransmitir. Estas funciones típicamente están implementadas en el controlador de hardware y no necesitan ser programadas.

# Capítulo 2

# Usando USB

## 2.1. Introducción a las Transferencias:

Las comunicaciones USB pueden ser divididas en dos categorías: Comunicaciones usadas para enumerar un dispositivo y comunicaciones utilizadas por aplicaciones que llevan a cabo los propósitos de los dispositivos. Durante la enumeración el host aprende acerca del dispositivo y lo prepara para el intercambio de datos. La comunicación de la aplicación se lleva a cabo cuando el host intercambia datos que realizan la función para la cual el dispositivo fue diseñado. Por ejemplo, para un teclado, la comunicación de aplicación esta enviando al host datos acerca de las teclas presionadas para decirle a la aplicación que muestre el carácter o realice otra acción.

**Comunicaciones utilizadas para enumerar un dispositivo:** Durante la enumeración, el firmware del dispositivo responde a una serie de pedidos estándar del host. El dispositivo debe identificar cada pedido, retornar la información solicitada, y tomar otras acciones especificadas por el pedido.

**Comunicaciones utilizadas por aplicaciones:** Luego que el host ha intercambiado la información de enumeración con el dispositivo y el controlador del dispositivo ha sido asignado y cargado, la comunicación de aplicación puede comenzar. En el host las aplicaciones pueden usar las funciones estándar de la API del sistema operativo u otros componentes de software para leer y escribir al dispositivo. Para el dispositivo la transferencia requiere de colocar los datos a enviar en el buffer de transmisión de la controladora o recuperar datos del buffer de recepción, y al completar una transmisión, asegurarse de que el dispositivo esta listo para la siguiente transmisión. Cada transferencia de datos usa uno de los cuatro tipos de transferencias: control, interrupt, bulk, o isochronous. Cada una tiene un formato y un protocolo para ajustarse a las diferentes necesidades.

## 2.2. Manejando los datos en el BUS:

Las dos lineas de señal del USB llevan datos desde y hacia todos los dispositivos en el BUS. Los cables forman una sola linea de trasmisión que todos dispositivos deben compartir. En contraste con RS-232, que tiene una linea TX para llevar datos en una dirección y una linea RX para la otra dirección.

El host esta encargado de asegurar que las transferencias ocurran lo más rápido posible. El host maneja el tráfico dividiendo el tiempo en intervalos llamados frames (para para low y full speed) o microframes (para high speed). El host reserva una porción de cada frame o microframe para cada transferencia (figura 2.1). Un frame tiene un periodo de un milisegundo. Para tráfico high speed, el host divide cada frame en ocho microframes de 125 microsegundos. Cada frame o microframe comienza con una referencia de tiempo del tipo start-of-frame. Cada transferencia consiste de una o más transacciones.

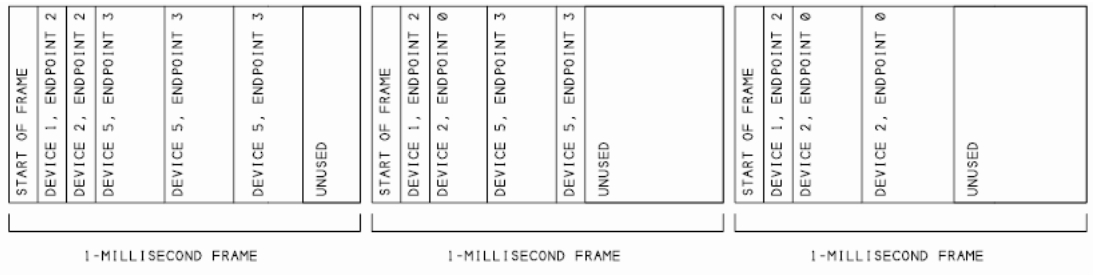



Figura 2.1: Frames USB

En low y full speed, el host planifica las transacciones en frames de un milisegundo. El host puede organizar transacciones en donde quiera dentro del frame. El proceso es similar a high speed, pero usando microframes de 125 microsegundos.

Dado que todo el tráfico comparte una linea de datos, cada transacción debe incluir una dirección del dispositivo que identifica al destinatario de la transacción. Cada dispositivo tiene una dirección asignada por el host, y todos los datos viajan hacia o desde el host. Cada transacción comienza cuando el host envía un bloque de información que incluye la dirección del dispositivo receptor y una localización especifica llamada endpoint. Todo lo que envía un dispositivo son respuestas a paquetes enviados por el host.

## 2.3. Elementos de una Transferencia:

Cada transferencia USB consiste de una o más transacciones, y cada transacción contiene paquetes que contienen información. Para comprender las transacciones, los paquetes y sus contenidos primero debemos conocer los endpoints y los pipes.

### 2.3.1. Device Endpoints:

Todo el tráfico en el bus viaja desde o hacia un device endpoint. El endpoint es un buffer que guarda múltiples bytes. Típicamente un endpoint es un bloque de memoria de datos o un registro en el chip del controlador. Los datos almacenados en un endpoint son datos recibidos o datos esperando a ser enviados. El host también tiene buffers que mantienen los datos recibidos y los que están esperando para ser enviados, pero el host no tiene endpoints. El host sirve de comienzo y fin para la comunicaciones con los device endpoints.

Como el bus es controlado por el host no es posible escribir simplemente en el bus, lo que se hace es escribir datos a un endpoint IN donde los datos son almacenados en el buffer hasta que el host envía un paquete de tipo IN a ese endpoint solicitando los datos.

La especificación USB define un device endpoint como "una porción única direccionable del dispositivo USB que es origen o destino de la información en un flujo de comunicación entre el host y el dispositivo." Esta definición sugiere que un endpoint conduce información solo en una dirección. Pero como veremos más adelante un endpoint de control es un caso especial que es bidireccional.

La dirección de un endpoint consiste de un número de endpoint y un sentido, el número es un valor entre 0 y 15. La dirección es definida desde el punto de vista del host: un endpoint de **entrada (IN)** provee datos para ser enviados al host y un endpoint de **salida (OUT)** guarda datos recibidos desde el host. Un endpoint configurado para transferencias de control debe transferir datos en ambos sentidos, por lo tanto un endpoint de control en realidad consiste de un par de sentidos de endpoints de entrada y salida que comparten el número de endpoint.

Otros tipos de transferencias envían datos solo en una dirección, la información de status y de control fluyen en direcciones opuestas. Un número simple de endpoint puede soportar tanto direcciones de **entrada** como de **salida**. Por ejemplo un dispositivo puede tener una dirección de endpoint uno de entrada para enviar datos al host y una dirección de endpoint uno de salida para recibir datos del host.

Toda transacción en el bus comienza con un paquete que contiene el número de endpoint y un código que indica el sentido del flujo de datos y cuando ó no la transacción esta iniciando una transferencia de control. Los códigos son IN, OUT y SETUP.

| Tipo de transacción | Origen de datos | Tipos de transferencias que usan este tipo de transferencia | Contenido |
|---|---|---|---|
| IN | Dispositivo | Todas | Datos o información de estado |
| OUT | Host | Todas | Datos o información de estado |
| SETUP | Host | Control | Pedidos |

Cuadro 2.1: Transacciones

### 2.3.2. Pipes: Conectando Endpoints y Hosts

Antes que una conexión pueda ocurrir el host y el dispositivo deben establecer un pipe. Un pipe USB es una asociación entre un endpoint y el software en el host controlador.

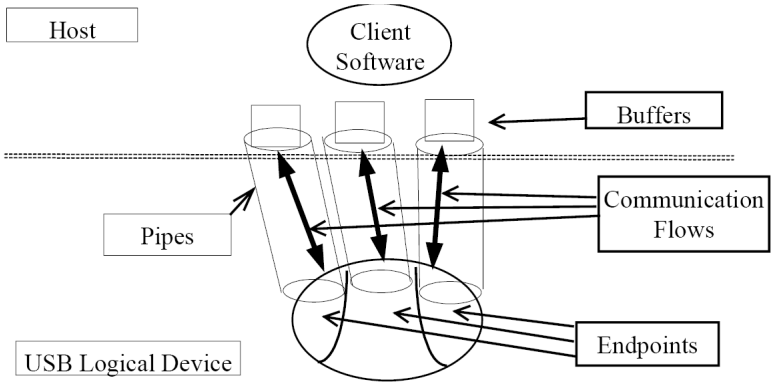



Figura 2.2: Flujo de comunicación USB

El host establece pipes durante la enumeración, si el dispositivo es removido del bus, el host remueve los pipes que ya no son necesarios. El host puede pedir nuevos pipes o remover los ya no utilizados en otros momentos mediante la solicitud de una configuración alternativa o interface para un dispositivo. Cada dispositivo tiene un pipe de control por defecto que usa el endpoint cero.

En la figura 2.2 se muestra como el flujo de información es conducido por los pipes entre los endpoints y los buffers del lado del host.

## 2.4. Tipos de transferencias:

USB esta diseñado para manejar distintos tipos de periféricos con una gran variedad de requerimientos para la frecuencia de transferencia, tiempo de respuesta y corrección de errores. Cada uno de los cuatro tipos de transferencias manejan diferentes necesidades, y los dispositivos pueden utilizar los tipos de transferencias que mejor se adecuen para sus propósitos. La tabla 2.2 muestra los distintos tipos de transferencias y sus características.

Las transferencias de ***control*** son las únicas que tienen funciones definidas por la especificación USB. Las transferencias de control permiten al host leer información acerca del dispositivo, asignar una dirección a un dispositivo, seleccionar configuraciones y otras características. Las transferencias de control también pueden enviar pedidos específicos del vendedor. Todos los dispositivos USB deben soportar transferencias de control.

Las transferencias ***bulk*** están pensadas para situaciones donde la latencia de la transferencias no es critica, como enviar un archivo a una impresora, recibir datos de un scanner, o acceder a un archivo en un disco. Para estas aplicaciones son llamativas las transferencias rápidas pero los datos pueden esperar si es necesario. Si el bus esta muy ocupado las transferencias bulk son

retardadas, pero si el bus esta libre las transferencias bulk son muy rápidas. Solo los dispositivos full y high speed pueden hacer transferencias bulk.

Las transferencias ***interrupt*** son para dispositivos que deben la atención del host o dispositivos periódicamente. Aparte de las transferencias de control, las transferencias Interrupt son la única forma de transferir datos para los dispositivos low-speed. Teclados y mouses utilizan este tipo de transferencias para enviar información.

Las transferencias ***isochronous*** tienen un tiempo de envío garantizado, pero no poseen control de errores. Son usadas para transmitir datos multimedia en aplicaciones de tiempo real. Es el único tipo de transferencia que no soporta retransmisión de datos recibidos con error. Solo los dispositivos full y high speed pueden hacer transferencias Isochronous.

| Tipo de transferencia | Control | Bulk | Interrupt | Isochronous |
|---|---|---|---|---|
| Uso típico | Identificación y configuración | Impresoras, scanners | Mouse, teclados | Streaming de audio y video |
| Requerido? | Si | No | No | No |
| Soporta low-speed? | Si | No | Si | No |
| bytes de datos/ms por transferencia, máximo posible por pipe (high speed).* | 15872 (31 transacciones / microframe de 64-byte) | 53248 (30 transacciones / microframe de 512-byte) | 24576 (3 transacciones / microframe de 1024-byte) | 24576 (3 transacciones / microframe de 1024-byte) |
| bytes de datos/ms por transferencia, máximo posible por pipe (full speed).* | 832 (30 transacciones / microframe de 64-byte) | 1216 (19 transacciones / microframe de 64-byte) | 64 (1 transacciones / microframe de 64-byte) | 1023 (1 transacciones / microframe de 1023-byte) |
| bytes de datos/ms por transferencia, máximo posible por pipe (low speed).* | 24 (31 transacciones / microframe de 8-byte) | No permitido | 0.8 (8 bytes cada 10 ms) | No permitido |
| Sentido del flujo de datos | Entrada y salida | Entrada o salida | Entrada o salida (USB 1.0 sólo soporta entrada) | Entrada o salida |
| Ancho de banda reservado para todas las transferencias del tipo (porcentaje) | 10 a low / full speed, 20 a high speed (mínimos) | No | 90 para low / full speed, 80 a high speed combinado para isochronous e interrupt (como máximo) | 90 para low / full speed, 80 a high speed combinado para isochronous e interrupt (como máximo) |
| Corrección de errores? | Si | Si | Si | No |
| Message o stream data? | Message | Stream | Stream | Stream |
| Frecuencia de entrega garantizada | No | No | No | Si |
| ¿Latencia garantizada (tiempo máximo entre transferencias)? | No | No | Si | Si |
| * Se asume que las transferencias usan el tamaño de paquete máximo | | | | |

Cuadro 2.2: Tipos de transferencias

Además de clasificar un pipe por el tipo de transferencia que lleva, la especificación del USB define los pipes como de *stream* o de *message*, según si la información viaja en uno o en ambas

direcciones. Las transferencias de control utilizan pipes bidireccionales de message; el resto de los tipos de la transferencia utilizan pipes unidireccionales stream.

**Inicializando una transferencia:** Cuando un controlador de dispositivo quiere comunicarse con un dispositivo, el controlador inicializa la transferencia. La documentación USB define una transferencia como el proceso de realizar y llevar a cabo un pedido de comunicación. Una transferencia USB consiste de transacciones. Las transacciones consisten en uno, dos, o tres paquetes.

Existen tres tipos de transacciones que se definen según el sentido del flujo de datos y el propósito. Las transacciones de Setup envían pedidos de control-transferencia a un dispositivo. Las transacciones OUT envían a un dispositivos datos o información de estado. Las transacciones IN envían a un host datos o información de estado.

La especificación USB define una transacción como la entrega de servicios a un endpoint. Servicio en este caso puede significar tanto el host enviando información al dispositivo, como el host requiriendo y recibiendo información desde el dispositivo.

Cada transacción incluye identificación, chequeo de error, estado e información de control tanto como cualquier información a ser intercambiada. Una transferencia completa ocupa varios frames o microframes, pero una transacción se debe completar sin interrupciones. Ninguna otra comunicación en el bus puede interrumpir en medio de una transacción. Por lo tanto los dispositivos tienen que ser capaces de responder rápido a los pedidos de información o a la información de estado en una transacción. Una transferencia con poca cantidad de datos puede requerir solo una transacción. Otras transferencias requieren múltiples transacciones con una porción de datos en cada una.

**Fases de una transacción:** Cada transacción tiene hasta tres fases, o partes que ocurren en secuencia: token, data, y handshake. Cada fase consiste en la trasmisión de uno o dos paquetes. Cada paquete es un bloque de información con un formato definido. Todos los paquetes comienzan con un Packet ID (PID) que contiene información identificatoria. Dependiendo de la transacción, el PID puede ser seguido por una dirección de endpoint, datos, información de estado, o un numero de frame, y bits de chequeo de error (ver figura **??**).

En la fase de token de una transacción, el host inicia la comunicación enviando un paquete de token. El PID indica el tipo de transacción, como Setup, IN, OUT, o Start-of-Frame.

En la fase de data, el host o el dispositivo pueden transferir cualquier tipo de información en un paquete de datos. El PID incluye un data-toggle (ver 2.5.3) o un valor de secuenciamiento de los datos usado para recuperarse de paquetes perdidos o duplicados cuando una transferencia tiene múltiples paquetes de datos.

En la fase de handshake, el host o dispositivo envía información de estado en un paquete de handshake. El PID contiene un código de estado (ACK, NAK, STALL o NYET). La especificación USB a veces usa los términos fase de estado y paquete de estado para referirse a la fase de handshake y el paquete de handshake.

La fase de token tiene un uso adicional. Un paquete de token puede llevar una marca de Start-of-Frame (SOF), la cual es una referencia de tiempo que el host envía a intervalos de un milisegundo a full speed y a intervalos de 125 microsegundos a high speed. Este paquete contiene también un numero de frame. Un endpoint puede sincronizarse al paquete Start-of-Frame o usar el contador de frames como una referencia de tiempo. El marcador de Start-of-Frame también mantiene a los dispositivos para no entrar en estado suspendido cuando no hay tráfico USB.

Los dispositivos Low-speed no ven los paquetes SOF. En su lugar el hub al que el dispositivo esta conectado utiliza un End-of-Packet (EOP) llamado señal keep-alive para dispositivos low-speed, la cual es enviada una vez por cada frame. Al igual que el SOF para los dispositivos full speed, el keep-alive mantiene a los dispositivos low-speed para no entrar en el estado de suspensión.

Toda transacción tiene un paquete de token. El host es siempre la fuente de este paquete, el cual configura la transacción identificando el tipo de transacción, el dispositivo que lo recibe, el endpoint, y el sentido de cualquier dato que la transacción va a transferir.

Dependiendo del tipo de transferencia y si es el host o el dispositivo que tiene información a enviar, un paquete de datos precede al paquete de token. El sentido especificado en el paquete de token determina cuando el host o el dispositivo envía el paquete de datos.

En todos los tipos de transferencias exceptuando las isochronous, el receptor de el paquete de datos (o el dispositivos si no hay paquetes de datos) retorna un paquete de handshake conteniendo un código indicando si la transacción fue exitosa o falló. La ausencia de paquete de handshake indica un error más serio.

**Restricciones de tiempo y garantías:** Los retardos permitidos entre los paquetes de token, data, y handshake de una transacción son muy pequeños, previstos para permitir solo los retardos del cable y el breve tiempo para permitir al hardware preparar una respuesta, como un código de estado en respuesta de un paquete recibido.

Un error común al escribir el firmware es asumir que el firmware debe esperar una interrupción antes de proveer los datos a enviar al host.

En cambio, antes que el host solicite los datos, el firmware debe copiar los datos a enviar en el buffer del endpoint y configurar el endpoint para enviar los datos al recibir un paquete de token IN. La interrupción ocurre luego que la transacción se completa, para decirle al firmware que el buffer del endpoint puede guardar datos para la próxima transacción. Si el firmware espera una interrupción antes de proveer los datos iniciales, la interrupción nunca ocurre y ningún dato es transferido.

## 2.5. Asegurando que las transferencias sean exitosas

### 2.5.1. Handshaking:

Como otras interfases, USB usa información de estado y control, o handshaking para ayudar a manejar el flujo de datos. En handshaking por hardware, lineas dedicadas transportan la información de handshaking.

USB usa handshaking por software. Donde un código indica el éxito o la falla de todas las transacciones exceptuando las transferencias isochronous. Además de esto en las transferencias de control, en la etapa de de Status se habilita al dispositivo a reportar el éxito o fallo de una transferencia completa.

Las señales de handshaking se trasmiten en los paquetes de handshake o de datos. Los códigos de estado definidos son: ACK, NAK, STALL, NYET, y ERR.

**ACK** (acknowledge) indica que el host o el dispositivo ha recibido los datos sin error.

**NAK** (negative acknowledge) significa que el dispositivo esta ocupado o no tiene datos para retornar.

**STALL** El handshake STALL puede tener cualquiera de estos tres significados: pedido de control no soportado, pedido de control fallido, o fallo de endpoint.

**NYET** Solo los dispositivos high-speed usan NYET, que significa not yet. Las transferencias high-speed bulk y control tiene un protocolo que permite al host darse cuenta antes de enviar datos si el dispositivo esta listo para recibir los datos.

**ERR** Es usado solo por los hubs high-speed en transacciones complete-split.

**NO RESPONSE** Otra forma de indicar el estado ocurre cuando el host o el dispositivo esperan recibir un handshake pero no reciben nada. Esta falta de respuesta puede ocurrir si el calculo de control de error del receptor detecta un error. Al no recibir respuesta, el emisor sabe que debe intentar nuevamente. Si luego de enviar en varias ocasiones continua fallando, el emisor puede tomar otra acción.

### 2.5.2. Control de error:

Los paquetes de token, data, y Start-of-Frame incluyen bits para ser usados en el control de error. Los valores de los bits son calculados usando el algoritmo llamado chequeo de redundancia cíclica (CRC).

El CRC es aplicado a los datos a ser chequeados. El dispositivo transmisor realiza los cálculos y envía el resultado junto con los datos. El receptor realiza los mismos cálculos sobre los datos recibidos. Si los resultados son iguales, los datos han sido recibidos sin error y el dispositivo retorna un ACK. Si los resultados no son iguales, el dispositivo receptor no envía handshake. La ausencia del handshake esperado indica al emisor que tiene que reintentar.

Típicamente, el host reintenta un total de 3 veces, pero la especificación USB le da al host cierta flexibilidad en determinar el número de reintentos. Al darse por vencido el host informa al driver del problema.

El campo de PID en paquetes token usa un método más simple de control de error. Los cuatro bits menos significativos en el campo de PID, y los 4 bits más significativos son su complemento.

### 2.5.3. La conmutación de datos:

En transferencias que requieren transacciones múltiples, el valor de data-toggle permite asegurar que ninguna transacción sea perdida manteniendo los dispositivos emisor y receptor sincronizados. El valor de data-toggle es incluido en el campo de PID de los paquetes de data para las transacciones IN y OUT. Cada endpoint mantiene su número de data toggle propio.

Tanto el emisor como el receptor conservan el data toggle.

Cuando el host configura un dispositivo, al encender o conectarse, el host y el dispositivo setean sus data toggles a DATA en t = 0. Al detectar un paquete entrante, el host o el dispositivo comparan el estado de su data toggle, con el data toggle recibido. Si los valores coinciden el receptor conmuta su valor y retorna un paquete de handshake ACK al emisor. El ACK causa al emisor a conmutar su valor para la próxima transacción.

El próximo paquete recibido en la transferencia debe contener un data toggle de DATA en t = 1, y nuevamente el receptor conmuta su bit y retorna un ACK. El data toggle continua alternando hasta que la transferencia es completada.

Si el receptor esta ocupado y retorna un NAK, o si el receptor detecta datos corruptos y no retorna respuesta, el emisor no conmuta su bit y reintenta con el mismo dato y data toggle.

Si el receptor retorna un ACK pero por alguna razón el emisor no ve el ACK, el emisor va a pensar que el receptor no recibió los datos y va a intentar de nuevo usando el mismo dato y data toggle. En este caso, el receptor del dato repetido no conmuta el data toggle e ignora el dato, pero retorna un ACK. Este ACK resincroniza los data toggles.

## 2.6. Transferencias con restricciones de tiempo criticas:

Solo porque un endpoint es capaz de una frecuencia de transferencia no significa que un dispositivo particular y un host van a ser capaces de alcanzar esa frecuencia. Muchas cosas pueden limitar la habilidad de una aplicación para enviar o recibir datos a la frecuencia que un dispositivo necesita. Los factores limitantes pueden ser el ancho de banda del bus, las capacidades del dispositivo, las capacidades del controlador de dispositivo y las latencias del software de aplicación, y latencias en el hardware del host y en el software.

### 2.6.1. Ancho de banda del bus:

Cuando un dispositivo solicita más ancho de banda para transferencias interrupt o isochronous del que esta disponible, el host se niega a configurar el dispositivo.

### 2.6.2. Capacidades del dispositivo:

Si el host promete que el ancho de banda USB solicitado va a estar disponible, de todas maneras no hay garantía que el dispositivo va a estar listo para enviar o recibir datos cuando sea necesario.

Para usar transferencias interrupt e isochronous eficazmente, tanto el emisor como el receptor tienen que ser capaces de enviar y recibir a la frecuencia deseada. Un dispositivo que envía datos debe escribir los datos a enviar en el buffer de transmisión del endpoint a tiempo para permitir al controlador colocar los datos en el bus al recibir un paquete de token IN. Un dispositivo que esta recibiendo datos debe leer los datos previos del buffer del endpoint antes de que el nuevo dato llegue.

Una manera para ayudar a asegurar que un dispositivo esta siempre listo para una transferencia es usar buffering doble (o cuádruple). Buffers múltiples le dan al firmware tiempo extra para cargar el siguiente dato a transmitir o recuperar los datos recién recibidos.

### 2.6.3. Capacidades del host:

Las capacidades del controlador de dispositivo y el software de aplicación en el host pueden también afectar cuando todas las transferencias disponibles toman lugar.

Un controlador de dispositivo solicita una transferencia mediante el envío de un paquete de pedido de E/S (IRP) hacia el controlador de un nivel inferior. Para transferencias interrupt e isochronous, si no hay un IRP pendiente para un endpoint cuando su tiempo programado llega a fin, el controlador de host saltea el intento de transacción. Para asegurarse que no se pierden oportunidades de transferencia, los drivers solicitan nuevos IRP inmediatamente después de completado el anterior.

El software de aplicación que usa los datos también debe mantenerse al ritmo de las transferencias, lo que se suele manejar es aumentar el tamaño del buffer que el driver usa.

Una forma de asegurar que una aplicación envíe o reciba datos con retrasos mínimos es colocar el código que se comunica con el controlador de dispositivo en un thread de programa independiente. El thread debe tener pocas responsabilidades aparte de administrar estas comunicaciones.

Haciendo pocas transferencias largas, en lugar de múltiples transferencias pequeñas puede también ayudar.

### 2.6.4. Latencias del host:

Otro factor en la performance de las transferencias criticas en tiempo en USB es la latencia debido a como el sistema operativo maneja la multitarea.

Por ejemplo Windows nunca fue diseñado como un sistema operativo de tiempo real que pueda garantizar una frecuencia de transferencias con un periférico.

En general es mejor dejar al dispositivo manejar cualquier requerimiento de procesamiento en tiempo real y hacer que el tiempo en las comunicaciones con el host sean lo menos criticas posibles. Por ejemplo si tenemos un dispositivo full-speed que lee un sensor una vez por milisegundo. El dispositivo puede intentar enviar cada lectura al host en una transferencia interrupt separada, pero si la transferencia es salteada por alguna razón, las transferencias no van a ponerse al día. Si el dispositivo en cambio junta las lecturas y las transfiere usando menos frecuencia, pero transferencias más largas, el tiempo de las transferencias es menos critico.

# Capítulo 3

# Enumeración

Antes que las aplicaciones puedan comunicarse con un dispositivo, el host necesita aprender acerca de los dispositivos y asignar un controlador al dispositivo. La enumeración es el intercambio de información que acompaña a estas tareas. El proceso incluye asignación de una dirección para el dispositivo, lectura de descriptores desde el dispositivo, asignación y carga de un controlador de dispositivo, y selección de una configuración que especifique los requerimientos de consumo de energía, endpoint y otras características. El dispositivo luego esta listo para transferir datos usando cualquiera de los endpoints en su configuración.

## 3.1. El Proceso:

Una de las tareas del host es detectar la conexión y la remocion de dispositivos. Cada hub tiene un endpoint interrupt IN para reportar estos eventos al host. Al iniciar el sistema, el host poolea su root hub para aprender si algún dispositivo esta conectado, incluyendo hubs adicionales y dispositivos conectados a estos hubs. Luego de iniciado el sistema, el host continua pooleando periódicamente para aprender de cualquier nuevo dispositivo conectado o removido.

En el proceso de aprender de nuevos dispositivos, el host envía una serie de pedidos al hub del dispositivo, causando que el hub establezca un camino de comunicación entre el host y el dispositivo. El host luego intenta enumerar al dispositivo mediante el envío de transferencias de control conteniendo pedidos estándar de USB para el endpoint 0 del dispositivo. Todos los dispositivos USB deben soportar transferencias de control, los pedidos estándar, y el endpoint 0. Para una enumeración satisfactoria, el dispositivo debe responder a cada pedido retornando la información solicitada y tomando otras acciones requeridas.

Desde el punto de vista del usuario, la enumeración es invisible y automática excepto por posibles mensajes anunciando la detección de un nuevo dispositivo. Algunas veces cuando es la primera vez en usar un dispositivo el usuario necesita asistir en la selección de un controlador o especificando cuando el host debe buscar los archivos del controlador.

## 3.2. Pasos de la enumeración:

La especificación USB define seis estados para un dispositivo. Durante la enumeración, un dispositivo transita por cuatro de estos estados: Powered, Default, Address, y Configured como puede verse en la figura 3.1. (Los otros estados son Attached y Suspend). En cada estado, el dispositivo tiene definidas capacidades y comportamiento.

Los pasos siguientes son la secuencia clásica de eventos que ocurren durante la enumeración. Pero el firmware del dispositivo no puede asumir que los pedidos de enumeración y los eventos van a ocurrir en un orden en particular.

1. El usuario conecta un dispositivo a un puerto USB.
2. El hub detecta el dispositivo.

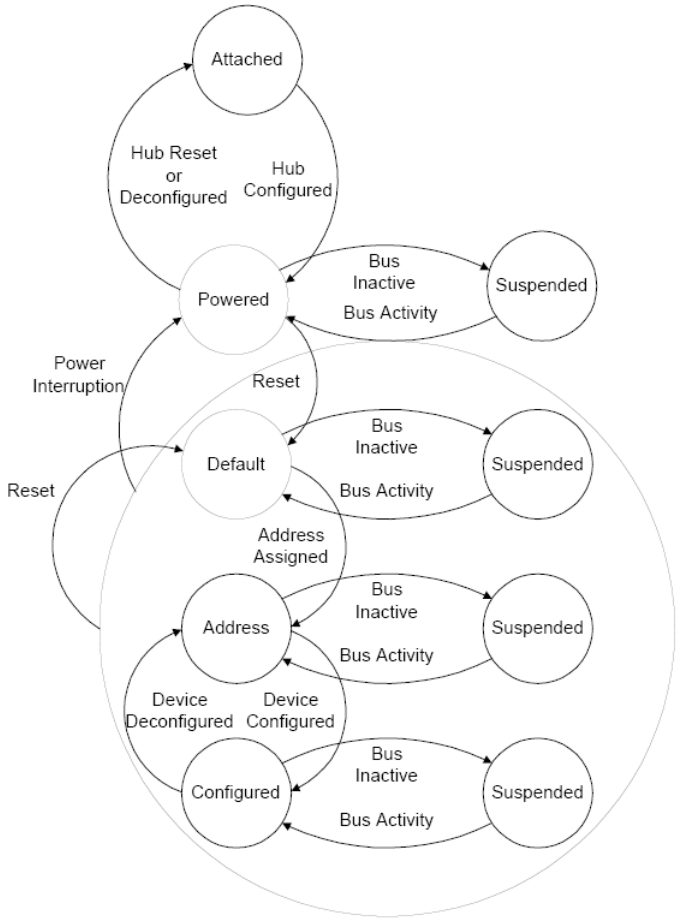



Figura 3.1: Diagrama de estados de un dispositivo USB

3. El host aprende del nuevo dispositivo.
4. El hub detecta cuando un dispositivo es low o full speed.
5. El hub resetea el dispositivo.
6. El host aprende si un dispositivo full-speed soporta high speed.
7. El hub establece una camino de señal entre el dispositivo y el bus.
8. El host envía un pedido Get_Descriptor para aprender el tamaño máximo de un paquete del pipe por defecto.
9. El host asigna una dirección
10. El host aprender acerca de las capacidades del dispositivo.
11. El host asigna y carga un controlador de dispositivo.
12. El controlador de dispositivo del host selecciona una configuración.

## 3.3. Descriptores:

Los descriptores USB son estructuras de datos, o bloques de información con formato, que le permiten al host aprender acerca de un dispositivo. Cada descriptor contiene información acerca del dispositivo como un todo o un elemento del dispositivo.

Todos los dispositivos USB deben responder a pedidos para los descriptores USB estándar. El dispositivo debe guardar información de los descriptores y responder a pedidos por los descriptores.

### 3.3.1. Tipos de Descriptores:

Durante la enumeración el host usa transferencias de control para solicitar descriptores a un dispositivo. A medida que avanza la enumeración, los descriptores requeridos están relacionados incrementalmente con pequeños elementos del dispositivo: primero el dispositivo entero, luego cada configuración, cada interface, y finalmente cada endpoint.

El nivel superior de descriptores informa al host de descriptores de bajo nivel adicionales. Excepto para los dispositivos compound.

#### 3.3.1.1. Device Descriptor

Contiene información básica acerca del dispositivo. Es el primero en leerse al conectarse el dispositivo y contiene información que el host necesita para obtener información adicional del dispositivo. El host solicita un device descriptor enviando un pedido de Get_Descriptor.

Un dispositivo puede tener un único device descriptor. Estos descriptores proveen información general como el fabricante, número de producto, número de serie, la clase de dispositivo y el número de configuraciones.

#### 3.3.1.2. Configuration Descriptor

Provee información acerca de los requerimientos de alimentación del dispositivo y cuantas interfaces son soportadas. Puede haber más de una configuración para un dispositivo.

#### 3.3.1.3. Interface Descriptor

Detallan el número de endpoints usados en la interface, como el tipo de interface. Puede haber más de una interface para una configuración.

#### 3.3.1.4. Endpoint Descriptor

Identifican el tipo de transferencia y su sentido, como otros datos específicos de un endpoint. Puede haber varios endpoints en un dispositivo y pueden ser compartidos en distintas configuraciones.

#### 3.3.1.5. String Descriptor

Varios de los descriptores previos referencian a uno o más string descriptors. Los string descriptors proveen información amigable acerca de la capa. Los string descriptors son generalmente opcionales.

| bDesciptorType | Tipo de Descriptor | Requerido? |
|---|---|---|
| 01h | Device | Si |
| 02h | Configuration | Si |
| 03h | String | No.Texto descriptivo opcional |
| 04h | Interface | Si |
| 05h | Endpoint | No, si el dispositivo usa solo en Endpoint 0. |
| 06h | Device_qualifier | Si, para dispositivos que soportan tanto full y high speeds. No permitido para otros dispositivos. |
| 07h | Other_speed_configuration | Si, para dispositivos que soportan tanto full y high speeds. No permitido para otros dispositivos |
| 08h | Interface_power | No. Implementa la administración de poder de la interface |
| 09h | OTG | Para dispositivos On-The-Go solamente. |
| 0Ah | Debug | No |
| 0Bh | Interface_association | Para dispositivos composite |

Cuadro 3.1: Tipos de descriptores

El campo bDescriptorType en un descriptor contiene un valor que identifica el tipo de descriptor.

# Capítulo 4

# Clases de Dispositivos

Cuando un grupo de dispositivos o interfaces comparten muchos atributos o proveen o requieren de servicios similares, tienen sentido definir los atributos y servicios en una especificación de clase. Los sistemas operativos pueden proveer driver para las clases en común, eliminando la necesidad de que los vendedores de dispositivos tengan que proveer los drivers para los dispositivos en esas clases. En la tabla 4.1 se puede ver una lista de clases incluídas en la especificación.

Cuando un dispositivo en una clase soportada tiene una característica única o habilidades no incluidos en el controlador de dispositivo, se puede proveer un driver de filtro para mantener las características agregadas y las habilidades, en lugar de escribir un controlador de dispositivo completo.

Una especificación de clase define el número y tipo de los endpoints requeridos u opcionales.

Una especificación de clase puede definir valores para los ítems en los descriptores estándar, como también descriptores class-specific, interfaces, usos de endpoints y pedidos de control.

| Clase | Descriptor donde la clase se encuentra definida |
|---|---|
| Audio | Interface |
| Chip/Smart Card Interface | Interface |
| Comunication | Dispositivo o Interface |
| Content Security | Interface |
| Device Firmware Upgrade | Interface (subclase de interface especifica de aplicación) |
| Human Interface (HID) | Interface |
| IrDA Bridge | Interface (subclase de interface especifica de aplicación) |
| Mass Storage | Interface |
| Printer | Interface |
| Still Image Capture | Interface |
| Test and Measurement | Interface (subclase de interface especifica de aplicación) |
| Video | Interface |

Cuadro 4.1: Clases con especificación aprobada

**Audio Class:** Esta clase define dispositivos que son origen o destino de información de audio en tiempo real.

Esta clase está definida en cuatro documentos separados [28]:

- Audio Device Document 1.0
- Audio Data Formats 1.0
- Audio Terminal Types 1.0
- USB MIDI ( music instruments device interface) Devices 1.0

**Comunications Device class** Esta clase define dispositivos que se conectan a la linea telefónica. Esta clase está definida en los siguientes documentos [28]:

- Class Definitions for Communication Devices 1.1
- Communications Device Class 1.

**Content Security** Esta clase define los mecanismos de transporte, descriptores y pedidos USB para soportar un método para la protección de la distribución de contenidos digitales mediante USB. La información a proteger es usualmente información con copyright. Esta clase está definida por los siguientes documentos [28]:

- Device Class Definition for Content Security Devices 1.0
- Content Security Method 1 - Basic Authentication Protocol 1.0
- Content Security Method 2 - USB Digital Transmission Content Protection Implementation 1.

**Human Interface Device Class (HID)** Esta clase define dispositivos manipulados por usuarios finales (como mouses, joystick). Las interfaces HID se comunican con el host usando tanto pipes de control como o pipes interrupt. Los pipes isochronous y bulk no son usados en los dispositivos de clase HID.

Está definido en los siguientes documentos [28]:

- Human Interface Devices 1.1
- HID Usage Tables 1.1
- HID Point of Sale Usage Tables 1.0

**Image Device Class** Esta clase define dispositivos que capturan imágenes fijas.

Está definido por el siguiente documento [28]:

- Still Image Capture Device Definition 1.0 document

**IrDA Class** Esta clase define una interface para transceivers infrarojos y está definida por el documento [28]:

- IrDA Bridge Device Definition 1.0 Document

**Mass Storage Device Class** Esta clase define dispositivos usados para almacenar grandes cantidades de información, como por ejemplo (disqueteras, discos duros, unidades de cinta).

Esta clase está definida por los siguientes documentos [28]:

- Mass Storage Overview 1.1
- Mass Storage Bulk Only 1.0
- Mass Storage Control/Bulk/Interrupt (CBI) Specification 1.0
- Mass Storage UFI Command Specification 1.

**Monitor Class** Esta clase es definida para controlar la configuración de un monitor y está especificada en el documento [28]:

- Monitor Device Document 1.0.

**Physical Interface Device Class (PID)** Define dispositivos que tienen respuesta táctil al operador. Como por ejemplo: Joysticks con resistencia variable para simular fuerzas y turbulencias.

Es parte de la clase HID, y está definida por el documento [28]:

- Device Class Definition for PID 1.0 document.

**Power Device Class** Dispositivos que alimenta de energía a sistemas o periféricos. Pueden ser tanto dispositivos independientes o integrados en la interface. El documento relacionado es:

- Power Device Class Document 1.0.

**Printer Device Class** Esta clase define los descriptores, endpoint y pedidos para impresoras. Está especificada en el documento [28]:

- Printer Device Class Document 1.

Otro documento documento importante que relaciona a todas las clases es "Universal Serial Bus Common Class Specification." [27] el cual tiene la intención de ser una guía para desarrollar especificaciones de clases, para promover implementaciones compatibles con drivers genéricos.

# Parte II

# Drivers

# Capítulo 5

# Introducción a los drivers

Se define como controlador (driver) de un dispositivo, a un componente de software que permite a los programas de aplicación tener acceso al hardware. Una de las principales funciones de un driver es ocultar a las aplicaciones todos los detalles referentes a la conexión física, señales y protocolos requeridos para poder comunicarse con el dispositivo. Esto permite que los códigos fuentes de las aplicaciones sean escritos de una forma independiente de los detalles concretos de cada dispositivo (puerto, interfases, control y manejo de señales). El código fuente de una aplicación podría ser el mismo para dispositivos funcionalmente similares pero que poseen distintas interfaces, pues los detalles específicos del hardware son manejados a bajo nivel. El driver cumple la función de traductor entre el código fuente de las aplicaciones y el código específico del hardware. Las aplicaciones se comunican con el driver de un dispositivo por medio de funciones provistas por el sistema operativo.

La primera gran separación de los drivers de dispositivos radica en donde ejecutan lo que define dos clases de drivers. La primera llamada comúnmente driver de modo usuario (User Mode Driver) que permite a los programas de aplicación comunicarse con dispositivos sin la necesidad de interactuar con el sistema operativo. Generalmente se implementan como bibliotecas de funciones en lenguajes de alto nivel, que poseen la lógica necesaria para interactuar con los sistemas operativos y acceder a los dispositivos. Esto permite integrarlas a las aplicaciones de usuario durante su desarrollo, logrando de este forma ocultar toda la comunicación de bajo nivel y facilitanto las tareas de depuración de las aplicaciones.

La otra clase se llama driver modo núcleo o kernel (User Mode Kernel) y ejecutan totalmente dentro del sistema operativo. Estos drivers son más complejos en su constitución que los user mode drivers, pues son implementados con lenguajes de programación de más bajo nivel como son C y Assembler que son utilizados en la construcción de los sistemas operativos. Algunas ventajas de estas clases de drivers, es que son mucho más performantes en su desempeño y además se pueden realizar cierto tipos de tareas que los user mode driver no pueden, como ser la intercepción y transformación de datos antes de que llegen a las aplicaciones. A continuación se describen brevemente algunos tipos o variantes de esta clase de driver, que se utilizan en las plataformas Windows y Unix:

- **Monolítico:** Es un driver que encapsula toda la funcionalidad necesaria para soportar un dispositivo de hardware. Es utilizado por una o más aplicaciones y maneja directamente el dispositivo de hardware. Para la comunicación con las aplicaciones utiliza comandos de control de entrada/salida (I/O control command)(IOCTL). Esta clase de drivers es soportada en las plataformas Unix y Windows.

- **Filtro:** Es un tipo de driver opcional que puede agregarse a la pila de drivers para agregar valor o modificar el comportamiento de un dispositivo. Básicamente su funcionamiento es la intercepción y manipulación de las entradas/salidas dek dispositivo. Como regla general, este clase de drivers no maneja directamente el hardware sino que trabajan con los datos y las entradas/salidas que pasan por ellos hacia la driver superior o inferior de la pila. El concepto de driver filtro es usado tanto en la plataforma Windows como Unix.

- **Clase y miniclase:** La pareja de estos dos drivers proporciona la lógica necesaria para dar soporte a un dispositivo específico. El driver clase satisface los requerimientos del sistema,

es decir, que da soporta un grupo de dispositivos con funcionalidades en común, como son todos los dispositivos USB de la clase HID o todos los dispositivos de red, pero tiene la particularidad de que son independientes del hardware. Un driver miniclase se ocupa de las operaciones de un determinado tipo de dispositivo de una clase en particular, de esta forma se obtine el beneficio de definir en un driver clase la lógica común y en cada driver miniclase solo lo específico del tipo de dispositivo. Estos tipos de drivers son utilizados únicamente en la plataforma Windows.

## 5.1. Modelos de Drivers

En la siguiente subsecciones se presentaran las características más importantes de los modelos de drivers de las plataformas Windows y Linux.

### 5.1.1. Plataforma Windows

Los drivers USB usan un modelo en capas llamado Modelo de Controladores de Windows (Windows Driver Model) (WDM) [30, 4], donde cada capa realiza una parte de la comunicación. La capa superior contiene al driver cliente o de función del dispositivo, el cual se encarga de manejar las comunicaciones entre las aplicaciones y el driver de bus. La capa inferior contiene al driver del bus que se encarga de manejar las comunicaciones entre el driver de función y el hardware. Además, pueden existir uno o más drivers filtro, que pueden complementar a los driver cliente y del bus.

El modelo en capas de los controladores USB simplifíca la tarea de su codificación, pues distintos dispositivos pueden compartir el código que realiza tareas en común. Por otro lado las aplicaciones no pueden acceder directamente al puerto USB, pues Windows no se lo permite y ésto obliga a que todas las comunicaciones de una aplicación deben ser realizadas por medio de un driver asignado a un dispositivo.

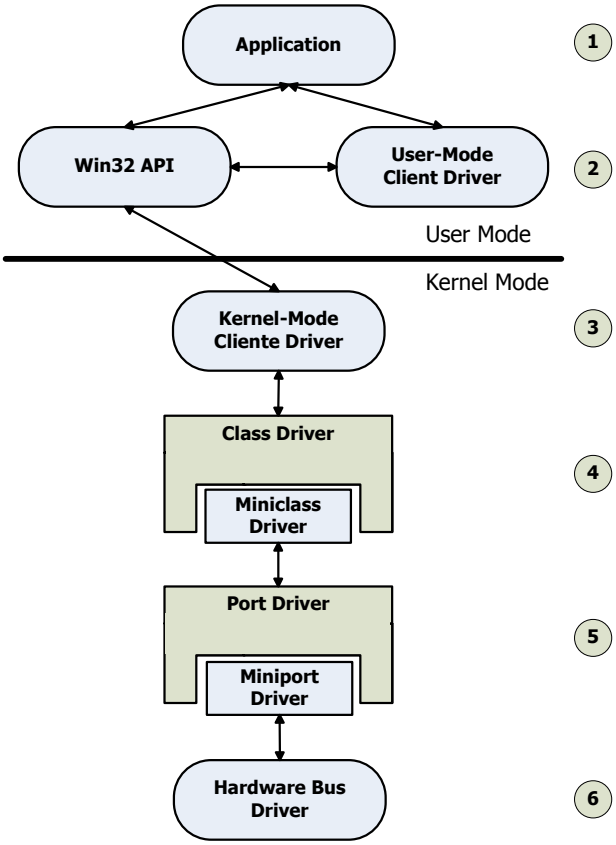



Figura 5.1: Arquitectura en capas de los drivers WDM.

Cada dispositivo es atendido por una cadena de drivers, que tipicamente se denomina pila de drivers (driver stack). Cada driver en la pila aisla alguna característica de hardware al driver que se encuentra por encima de él. La figura 5.1 muestra los tipos de drivers que potencialmente pueden existir en una pila de drivers de un dispositivo hipotético. En realidad, casi ningún dispositivo requiere que su pila de drivers contenga todos esos tipos de drivers. Como muestra la figura anterior.

1. Por encima de la pila de drivers se encuentran las aplicaciones. Estas manejan las peticiones de sus usuarios o de otras aplicaciones e invocan, ya se a la API de Windows (Win32 API) o a las funciones que exponen los drivers modo usuario.

2. Un driver modo usuario maneja los pedidos de las aplicaciones de usuario o de la API de windows. Para los pedidos que requieren servicios del sistema operativo, el driver modo usuario llama a la API de windows, la cual invoca el driver modo kernel o la rutina de soporte necesaria para atender el pedido.

3. Un driver modo kernel maneja los pedidos similares a los que reciben los drivers modo usuario, con la excepción de que dichos pedidos son atendidos dentro del sistema operativo que ejecuta en modo kernel o de privilegios.

4. El par de drivers: clase y mini-clase proveen la mayor parte de soporte de un dispositivo específico. El driver de clase suministra los requerimientos de sistema pero con un soporte independiente del hardware para una clase particular de dispositivos. Este tipo de driver es habitualmente suministrado por el sistema operativo.
   Un driver de mini-clase maneja las operaciones para un tipo específico de dispositivo de una clase en particular. Por ejemplo, el driver clase bateria soporta las operaciones comunes a cualquier bateria, mientras que el driver mini-clase de un dispositivo UPS en particular maneja los detalles particulares y únicos de ese dispositivo. Los driver mini-clase son habitualmente proporcionados por los fabricantes de dispositivos.

5. El driver de puerto correspondiente (para algunos dispositivos, esto es el host controller o host adapter driver) da soporte a las E/S requeridas para el puerto subyacente, hub u otro dispositivo físico a través del cual los dispositivos se conectan. Si cualquiera de esos drivers están presentes dependen del tipo de dispositivo y el bus en los que eventualmente se conectan. Para los dispositivos USB, la pareja de drivers roothub y host controller cumplen con las obligaciones del driver de puerto. Estos drivers manejan la E/S entre los dispositivos conectados al bus USB y el bus en si mismo. El driver mini-puerto correspondiente maneja las operaciones específicas del dispositivo para el driver puerto. Para la mayoria de los tipos de dispositivos, el driver puerto es suministrado con el sistema operativo, y el driver mini-puerto es suministrado por los fabricantes de dispositivos.

6. En la parte inferior de la imagen se encuentra el driver del bus hardware. Estos drivers son suministrados por el sistema operativo para la mayoria de los casos.

Por simplicidad, los drivers filtro no se mostraron en la figura 6.4. Sin embargo, un driver filtro puede incluirse en cualquiera de las capas que se encuentran por encima del driver de bus en la pila de drivers. Un driver filtro agrega valor a un driver existente pues permite la intercepción y manipulación de las E/S de los dispositivos. Como regla general, los drivers filtro no operan sobre el hardware directamente, pero trabajan únicamente sobre los datos y las pedidos de E/S que pasan por ellos hacia una capa superior o inferior.

# Capítulo 6

# Herramientas de Desarrollo

Con la aparición del USB se han multiplicado la cantidad de periféricos que se pueden conectar a un PC, esto es debido entre otras cosas a la facilidad de uso y configuración. Esta simplicidad se basa en una complejidad mayor a la hora de desarrollar un driver de un dispositivo, ya que se necesita tener conocimientos de como funciona la tecnología USB, así como de los detalles internos del funcionamiento del sistema operativo. A continuación se presentan algunas herramientas de desarrollo de drivers USB, las cuales se agrupan en tres grandes grupos por sus características:

- Básicas
- Drivers Genéricos
- Drivers Personalizados

## 6.1. Herramientas Básicas

Estos conjuntos de herramientas se caracterizan por ser gratuitas, utilizables para la construcción de cualquier driver de dispositivo y de acceso público. Además suelen ser aplicaciones de línea de comandos y de contar con una cantidad mínima de elementos, los cuales son indispensables para poder desarrollar un driver.

En la plataforma Windows este conjunto de herramientas básicas se conoce como Windows DDK [29] y está compuesto por un compilador C, un editor de vínculos (linker), utilitarios que asisten en la construcción de los drivers, documentación del WDM y código fuente de ejemplo de los distintos tipos de drivers, en particular para USB existen los ejemplos de: driver filtro, driver que utilizan transferencias bulk o Isochronous entre otros.

En la plataforma Linux la realidad es un poco distinta, en general las distintas versiones de Linux traen como parte de su distribución herramientas para la compilación y vinculación de código fuente escrito en el lenguaje C. Pero existía la necesidad de contar con información consolidada y ejemplos de código, para poder guiar la tarea de desarrollo de drivers. Recién en mayo del 2006, Greg Kroah-Hartman (uno de los desarrolladores del núcleo de Linux) hizo público el LDDK [17], que básicamente está compuesto por una recopilación de toda la documentación de la versión 2.6.16.18 del núcleo de Linux, de los códigos fuentes completos de esta versión del núcleo y de una copia en formato electrónico del libro sobre el tema [6].

La principal desventaja que presentan estos juegos de herramientas es que se necesita mucha experiencia en programación en C, una cantidad importante de conocimientos de como funcionan internamente los sistemas operativos, como son las comunicaciones entre las aplicaciones y el hardware, así como el conocimiento sobre los modelos de drivers de cada plataforma. Todo esto hace que este tipo de herramientas no sean la primera opción al momento de pensar en desarrollar un driver de dispositivo.

## 6.2. Drivers Genéricos

La idea básica de este tipo de herramientas es ahorrar la mayor cantidad de trabajo posible al desarrollador, tomando como base que todas las comunicaciones con los dispositivos USB

siguen el protocolo definido en la especificación USB, tiene sentido entonces, que un único driver genérico sea capaz de comunicarse con cualquier dispositivo. Además generan interfases o drivers específico para el dispositivo que ejecuta en modo usuario y permiten una comunicación con el driver genérico que ejecuta en modo kernel dentro del sistema operativo.

Un driver genérico completo debería soportar los cuatro tipos de transferencias USB incluyendo los pedidos de control definidos por terceros, también debería soportar la administración de energía y la capacidad de conectar y ejecutar (Plug and Play). Este enfoque permite mayor velocidad en el desarrollo y no requiere programación alguna para crear el driver, pero tiene como desventaja que no maneja todas las posibles situaciones. A continuación se presentan algunas de estas herramientas:

### 6.2.1. Jungo Ltd. - WinDriver USB 8.02

*WinDriver* [13] es un juego de herramientas de desarrollo que simplifica la creación de drivers monolíticos de dispositivos. Incluye un ambiente gráfico de desarrollo, API's, utilitarios de diagnóstico y depuración y ejemplos que permiten un rápido desarrollo de drivers de alto rendimiento.

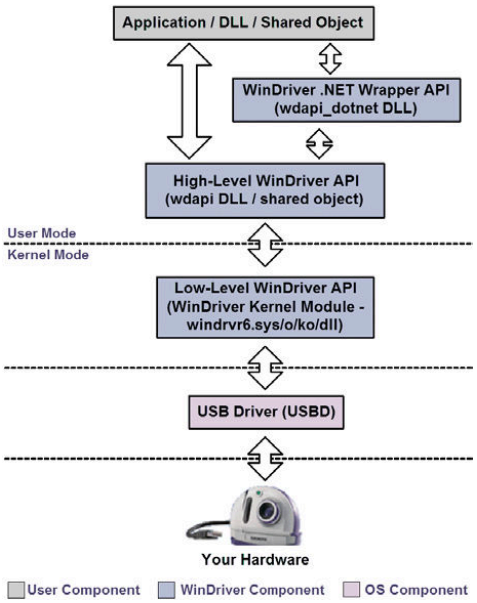



Figura 6.1: Arquitectura de la herramienta WinDriver USB 8.02.

#### Características

- Acceso inmediato al hardware: Se accede al hardware USB a través de una aplicación gráfica de usuario sin tener que escribir una sola línea de código.
- Generación de código específica para el hardware: *WinDriver* posee un ayudante que genera un esqueleto del código del controlador personalizado para su dispositivo.
- Fácil incorporación de funcionalidad al código generado: En modo usuario y en el ambiente de desarrollo de la preferencia del programador.
- Herramientas gráficas: Ayudantes y aplicaciones de usuario intuitivas que simplifican el acceso al dispositivo y la generación del código fuente del driver.
- Depuración: Monitor gráfico de depuración que permite observar las actividades que suceden en modo kernel y usuario.

- Ejemplos y generación de código para los ambientes de desarrollo más comunes: MS Visual C++, MS.Net (C# y VB), Borland C++ Builder, Borland Delphi, Visual Basic 6.0, Windows CE Plataform Builder, GCC, etc.
- Compatibilidad entre sistemas operativos: El código fuente generado es compatible entre los sistemas operativo soportados sin necesidad de realizar ningún cambio al mismo.
- Independencia del Hardware: Soporta cualquier hardware USB de base.
- Controlador certificable WHQL (Windows): Firma digitalmente el driver para poder presentarlo al testeo de Microsoft's WHQL.
- Detección de dispositivos USB: Además de los dispositivos conectados directamente al PC, puede detectar los dispositivos que se encuentran conectados más alla del puerto USB (conectados a Hubs USB).
- Información de cada dispositivo detectado: Ubicación física, identificador de fabricante y producto; información de configuraciones, interfases y endpoints.
- Verificación del hardware y depuración vía ayudantes:
  - Transferencia de paquetes de datos a través de los canales USB.
  - Lectura continua ("escucha") de los canales USB.
  - Restablecimiento de las operaciones de los canales.
- En conformidad con el WDM: Soporta el manejo de las caracterísiticas de gestión de energía y Plug and Play.
- Generación e instalación de los archivos INF.
- Soporta las tres velocidades de comunicación USB.
- Soporta todos los tipos de transferencias USB.
- Soporte para múltiples interfaces de dispositivos.
- Diseñado para cumplir con las especificaciones 1.1 y 2.0 de USB.
- Tipos de licencias y precios:
  - Node-Lock (Licencia individual para un desarrollador en un único PC): USD 3.000
  - Floating (Licencia individual para un desarrollador en varias PCs): USD 6.000
  - 3-Pack (Tres licencias Node-Lock): USD 6.000

**Requerimientos**

**Sistemas Operativos soportados:**

- Windows XP 64-bit / Server 2003 64-bit / Vista 64-bit
- Windows 98, Windows ME, Windows NT, Windows 2000, Windows XP, Windows Server 2003 y Windows Vista
- Windows CE 4.x-5.0 (x86 / MIPS / ARM CPU) y Windows Mobile 5.0 (ARMV4I CPU)
- Linux y Linux Embedded 2.4 - 2.6 (x86 32-bit y 64-bit; IA64; Power PC 32-bit)

**Compilador:**

- GCC o cualquier otro compilador ANSI C
- Borland Delphi
- Visual Basic 6.0
- Visual C#.Net, Visual Basic.Net

**Espacio en Disco:**

- Entre 26 y 34 MB dependiendo del sistema operativo.

### 6.2.2. Thesycon Systemsoftware & Consulting GmbH - USBIO Development Kit 2.31

*USBIO* [26] es un controlador genérico USB para Windows, que hace posible controlar cualquier tipo de dispositivo USB y provee una interfaz de programación que puede ser usada por cualquier aplicación que corra en la plataforma Windows.

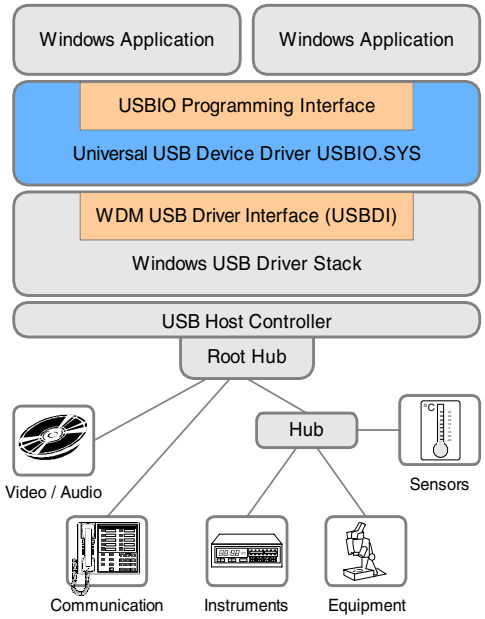



Figura 6.2: Arquitectura de la herramienta *USBIO* Development Kit 2.31.

**Características**

- Soporta las especificaciones de USB 1.1 y 2.0.
- Cumple con *WDM*.
- Soporta la funcionalidades Plug and Play y gestión de energía.
- El diseño y la implementación esta optimizada para una eficiente trasmisión de datos.
- La interfaz de programación es similar a la utilizada para el manejo de archivos entrada/salida.
- Múltiples dispositivos USB pueden ser controlados al mismo tiempo.
- Múltiples aplicaciones pueden utilizar *USBIO* al mismo tiempo.
- Todas las funciones USB del modo kernel están disponibles a nivel de las aplicaciones de usuario.
- Provee una interfaz con los endpoints USB similar a la de archivos.
- La interfaz nativa de programación soporta los lenguajes C, C++, Delphi y Java.
- Mediante la arquitectura COM, brinda una interfaz de programación de alto nivel para lenguajes como son Visual Basic, Delphi, C#, etc.

- Generación e instalación de los archivos INF.
- Soporta personalizaciones específicas del fabricante.
- Soporta todos los tipos de transferencias USB y modo asincrónico.
- Tipos de licencias y precios:
  - Developer (Licencia individual para un desarrollador en varias PCs durante el desarrollo[1]): EUR 600
  - Runtime (Es en los mismo terminos que la licencia Developer pero si se puede redistribuir como parte de productos comerciales): EUR 2300

**Requerimientos**

**Sistemas Operativos soportados:**

- Windows 98SE, WindowsME, Windows 2000, Windows XP, Windows Server 2003
- Windows XP 64-bit, Windows Server 2003 64-bit
- Windows XP Embedded

**Compilador:**

- Cualquier compilador de C, C++.
- Java
- Cualquier compilador de alto nivel que soporte COM (ej. Visual Basic, C#, Delphi, etc).

**Espacio en Disco:**

- Aproximadamente 8 MB.

### 6.2.3. EnTech Taiwan - RapidDriver Developer 2.1

La herramienta *RapidDriver* [8] fue creada para soportar desarrollos de nuevos dispositivos de hardware sin ser un experto en *DDK*. Provee un amplio soporte para dispositivos ISA, PCI, USB bajo Windows 2000, XP y Server 2003. Existen dos ediciones de la herramienta:

- *RapidDriver Explorer*: Permite extraer toda la información de los recursos requeridos por el hardware directamente desde el dispositivo, dejando al desarrollador libre para comenzar el desarrollo y testeo de las funciones especificas. Para los dispositivos USB, se pueden obtener todos los descriptores, realizar operaciones de lectura y escritura para los tipos de transferencias: bulk e interrupt, entre otras cosas.
- *RapidDriver Developer*: Incluye todas las características que la edición Explorer, pero permite además crear y distribuir aplicaciones propias que incorporan los drivers y bibliotecas dinámicas para controlar al hardware.

[1] No se permite la redistribución de ninguna parte del software. Dentro de los 12 meses después de la compra debe ser actualizada a la licencia Runtime pagando su la diferencia.

**Características**

- Soporta la especificación USB 1.1.
- Cumple con WDM.
- Múltiples dispositivos USB pueden ser controlados al mismo tiempo.
- Soporte para múltiples interfaces de dispositivos.
- Generación de ejemplos (código) para cada dispositivo.
- Soporta los tipos de transferencias USB: interrupt y bulk.
- Información de los descriptores de cada dispositivo detectado.
- Todas las funciones USB están disponibles a través de una biblioteca dinámica.
- Incluye un depurador del hardware para poder analizar más de cerca al dispositivo.
- Permite hacer depuración por medio de línea de comandos o usando el motor multi-lenguaje de scripts FastScript.
- Tipo de licencias y precios:
  - RapidDriver Explorer:
    - Personal (Un usuario en un único PC): USD 200
    - Group (Hasta 5 usuarios): USD 500
  - RapidDriver Developer:
    - Personal (Un desarrollador en un único PC con la facultad de redistribuir los drivers y dll's como parte de los productos): USD 350
    - Group (Hasta 5 desarrolladores, permitiendo la redistribucción de los drivers y dll's como parte de los productos): USD 950

**Requerimientos**

**Sistemas Operativos soportados:**

- Windows 2000, Windows XP y Windows Server 2003

**Compilador:**

- MS Visual C/C++
- Borland Delphi
- Borland C++ Builder
- Visual Basic 6.0
- Visual Basic.Net y Visual C#.Net

**Espacio en Disco:**

- Aproximadamente 6 MB

### 6.2.4. Icaste llc - JCommUSB 1.0

El JCommUSB [11] es una API que provee acceso al puerto USB a aplicaciones que funcionan en sistemas operativos Windows. La API provee medios para instalar y configurar los dispositivos USB así como métodos de lectura y escritura de los endpoints. Esta herramienta tiene como objetivo brindar el más alto nivel de abstracción para el desarrollador Java, proveyendo una interfase simple pero poderosa para comunicarse con los dispositivos USB.

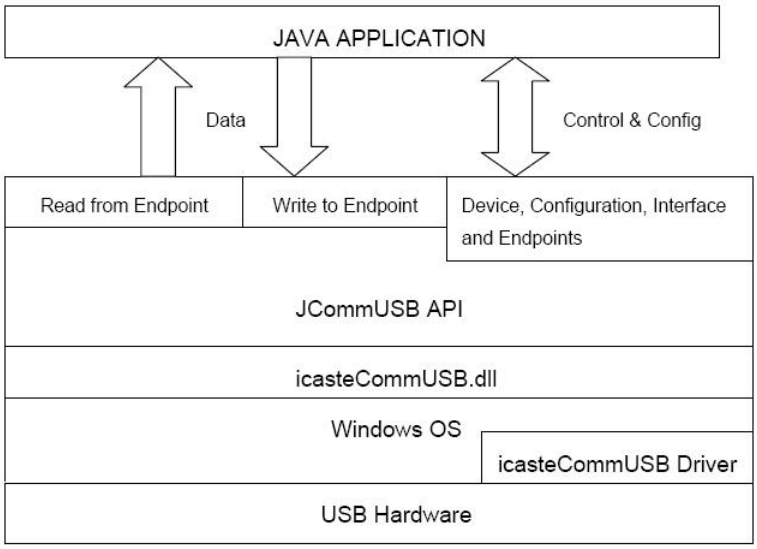



Figura 6.3: Arquitectura de la herramienta JCommUSB 1.0

**Características**

- La API utiliza la clase Dispositivo para representar a los dispositivos USB que se conectan y la clase Excepción para manejar cualquier excepción que se produzca.
- Sólo se puede utilizar la primera configuración del dispositivo, ya que no se proveen métodos para seleccionar configuraciones alternativas.
- Únicamente se soportan dispositivos en los que sus identificadores de hardware comiencen con la sigla 'USB\'.
- Soporta la especificación USB 2.0
- Soporta todos los tipos de transferencia USB.
- Tipos de licencias y precios:
  - Personal Edition (Un desarrollador en un único PC, no comerciable): USD 35
  - Commercial Edition 1 unidad (Un desarrollador en un único PC, comerciable): USD 40
  - Commercial Edition 5 unidades: USD 175

**Requerimientos**

**Sistemas Operativos Soportados:**

- Windows XP y Windows XP 64-bits
- Windows Server 2003 y Windows Server 2003 64-bits

**Compilador:**

- Java 2 Runtime Edition 1.5.0_05 o superior.
- Java 2 Standard Development Kit 1.5.0_05 o superior.

**Espacio en Disco:**

- Aproximadamente 1,5 MB

### 6.2.5. Otras herramientas

**Microchip Technology Inc. - MPUSBAPI Library 1.0**
Es una API que provee acceso al puerto USB a aplicaciones que funcionan en sistemas operativos Windows. Forma parte del paquete gratuito ***Microchip Full-Speed USB*** *Solutions (MCHPFSUSB)* [19], que permite el desarrollo de aplicaciones de usuario y de firmwares para la comunicación con dispositivos USB, además como parte del paquete cuenta con un driver USB de proposito general (clase Custom USB). Sus principales características son:

- Soporta los estándares USB 1.x y 2.0.
- Soporta los sistemas operativos: Windows 98SE, Windows ME, Windows 2000 y Windows XP.
- No brinda opciones para utilizar distintas interfases y configuraciones para la comunicación con los dispositivos.
- Soportan todos los tipos de transferencia USB y modo asincrónico para la del tipo interrupt.
- Conjunto de operaciones reducida y bloqueantes (timeout).
- Soporta el uso de 32 endpoints.
- La API es una biblioteca de vinculación dinámica (DLL), para su facil integración a los proyectos de desarrollo.
- Es una herramienta gratuita.

**LibUSB 0.1.12**
Es un proyecto [15] GNU-LGPL desarrollado para sistemas operativos del estilo UNIX y tiene como meta la creación de una biblioteca que permita a aplicaciones de usuario poder comunicarse con dispositivos USB sin importar el sistema operativo. La versión estable de este proyecto es la 0.1.12 y fue diseñada para tener una fuerte analogía con las especificaciones USB, sin embargo su implementación es muy pobre debido a la existencia de mucho código rígido (hardcode) lo que trae como resultado que se pierdan algunas características del diseño. Las principales características de este producto son:

- Soporta los sistemas operativos: Linux, FreeBSD, NetBSD, OpenBSD, Darwin/MacOS X
- Fue diseñado para soportar el estándar USB 1.x, aunque también puede soportar el 2.0.
- Puede utilizar simultáneamente distintas interfaces para la comunicación con un dispositivo.
- Los tipos de datos usan estructuras abstractas para mantener la portabilidad de representación de la información.
- Las funciones son bloqueantes y esperan la finalización del proceso o terminan por tiempo de espera.
- Soportan todos los tipos de transferencia y todas las peticiones estándar de la especificación USB.
- Es una herramienta gratuita.

**LibUSB-Win32**
Este proyecto [16] es una migración del proyecto LibUSB a la plataforma Windows, esta biblioteca permite a las aplicaciones de usuario comunicarse con cualquier dispositivo USB en Windows de una forma genérica sin la necesidad de codificar ninguna línea de un controlador modo núcleo. Las principales características de este producto son:

- Soporta los sistemas operativos: Windows 98SE, Windows ME, Windows 2000 ,Windows XP

- Las funcionalidades son 100 % compatibles con el proyecto LibUSB.
- Puede ser usado como un controlador filtro o como un controlador base de un dispositivo simultaneamente.
- Soporta transferencias del tipo control, bulk e interrupt y todas las peticiones estándar de la especificación USB.
- Soporta mensajes de control de terceros.
- Es una herramienta gratuita.

**jUSB**

Este proyecto [14] brinda una API gratuita y de código libre para Java que permite el manejo de dispositivos USB. Su diseño esta definido en términos de interacciones y descriptores encontrados en la especificación USB. Por abajo la API es una SPI (Service Provider Interface) que permite que la mayoría del código sea compartido, actualmente existen tres implementaciones de SPI: usb.linux que utiliza JNI para comunicarse con el usbdevfs, usb.remote que utiliza RMI para brindar un mecanismo de acceso remoto y usb.windows que utiliza una biblioteca de enlace dinámico y un controlador genérico de modo núcleo. Las principales características de este producto son:

- Soporta los estándares USB 1.1 y 2.0.
- Permite acceso a dispositivos nativos y remotos.
- Dispositivos con múltiples interfaces pueden soportar múltiples controladores.
- Soporta los tipos de transferencias: Control y Bulk, el tipo de transferencia Interrupt esta en construcción.
- Permite enumerar dispositivos y recibir notificaciones cuando el dispositivo se conecta o desconecta.
- Es un herramienta gratuita.

**JSR80 (javax.usb)**

Es una API Java [12] que permite acceder directamente al los dispositivos USB existentes en el sistema. Está compuesta por una parte totalmente codificada en Java que es común para todas las plataformas y luego para cada plataforma existen distintas implementación que realizan la conección a bajo nivel con el sistema operativo. Este producto esta estandarizado y fue creado utilizando el JCP (Java Community Process), lo que lleva a que en la actualidad sólo la implementación para Linux este certificada según el JCP. El diseño de la API tiene mucha concordancia con la estructura lógica interna de un dispositivo USB así como con la topología de la arquitectura USB.

## 6.3. Controladores Personalizados

Los juegos de herramientas totalmente automatizados no son utilizables para todos los dispositivos, ellos no pueden crear drivers filtro, ni permiten tener drivers complejos para obtener el mejor rendimiento posible. Es por eso que existen juegos de herramientas que proveen de bibliotecas y otros utilitarios para asistir en la codificación de drivers propios del dispositivo. En general estas herramientas presentan un esqueleto de driver, al cual se le pueden codificar sus partes y luego compilar, obteniendo controladores con un rendimiento casi igual como si hubieran sidos codificados desde cero. Este enfoque es mucho más flexible en cuanto al manejo de distintas situaciones pero requiere experiencia en la codificación. A continuación se presentan algunos productos con este enfoque:

### 6.3.1. Jungo Ltd. - KernelDriver 6.11

*KernelDriver* [13] es un juego de herramientas que permite desarrollar drivers que corren en el núcleo del sistema operativo en una forma genérica y para cualquier plataforma Windows que soporta nativamente USB (Win98SE, WinNT 4.0, WinMe, Win2000, WinXP, WinServer 2003).

Esta herramienta simplifica la tarea de creación de un driver de sistema ya que permite la generación automática de un esqueleto del código del driver, el cual cumple con el WDM. Además *KernelDriver* provee una API en el modo kernel del sistema operativo que permite acceso al hardware, la cual es portable a través de las distintas plataformas soportadas. En la figura 6.4 se aprecia la arquitectura de los componentes de este producto.

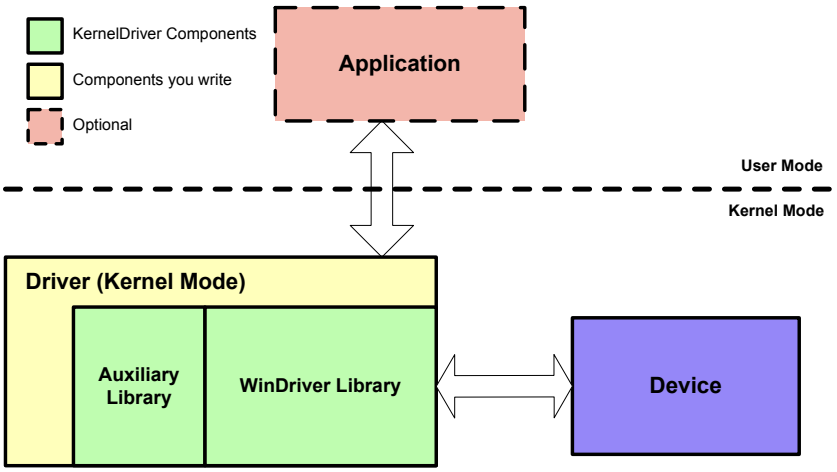



Figura 6.4: Arquitectura del KernelDriver 6.11

#### Características

- Soporta las especificaciones de USB 1.1 y 2.0.
- Acceso inmediato al hardware para un rápida revisión y diagnóstico del mismo por medio de una aplicación con entorno gráfica.
- Generación automática de un esqueleto del código del driver personalizado para su dispositivo.
- Soporte total de la API de la herramienta *WinDriver*, que permite crear y verificar el driver en modo usuario y luego poder usar el mismo código en el driver de modo kernel.
- API de modo kernel para facilitar el acceso y control del hardware.
- Los códigos fuentes y binarios generados con *KernelDriver* son compatibles a lo largo de las plataformas Windows soportadas.
- Carga y descarga dinámica del driver que facilita la tarea de revisión.
- Tipos de licencias y precios:
  - Node-Lock (Licencia individual para un desarrollador en un único PC): USD 3.000
  - Floating (Licencia individual para un desarrollador en varias PCs): USD 6.000
  - 3-Pack (Tres licencias Node-Lock): USD 6.000

#### Requerimientos

**Sistemas operativos soportados:**

- Windows 98SE, Windows ME
- Windows NT 4.0, Windows 2000, Windows XP y Windows Server 2003

**Compilador:**

- Microsoft Visual C++
- GCC
- Cualquier otro compilador C de 32-bit.

**Espacio en Disco:**

- Aproximadamente 44 MB

### 6.3.2. EnTech Taiwan - RapidDriver Source Builder

Incluía todas las características de las ediciones Explorer y Developer [8], mas la habilidad de generar código fuente que cumple con el WDM y que permite tener un driver totalmente personalizado para el dispositivo. Lamentablemente la empresa que desarrolla esta herramienta decidió discontinuar la versión en favor de poner a la venta los códigos fuentes de los drivers y bibliotecas dinámicas a un monto de USD 1.500.

# Capítulo 7

# Herramientas de Depuración

En esta sección se describiran algunas de la herramientas de depuración de drivers estudiadas como parte del relevamiento del estado del arte. Existen dos grandes clases de herramientas análisis: las que son un producto de software, que registran los eventos que se suceden así com el tráfico de información que se da entre las aplicaciones de usuario cuando invocan al driver hasta que los paquetes llegan al host controller USB del PC. Y las herramientas de depuración con componentes de hardware que permiten tener un grado de detalle más exacto pues pueden analizar los paquetes que se intercambian el dispositivo y el host controller USB del PC.

## 7.1. Analizadores vía Software

### 7.1.1. SourceQuest Inc.- SourceUSB 2.0

Es un analizador USB vía software [24] que trabaja a nivel del host y que permite registrar las peticiones entrada/salida USB, eventos y invocaciones de funciones de bajo nivel entre los componentes de la pila de drivers USB. Su funcionamiento se basa en un drivers modo núcleo que se instala en el sistema operativo y coexiste con la pila de drivers USB de Windows y una aplicación de alto nivel para visualizar la información recolectada. Es un complemento ideal para un analizador vía hardware que registra las transacciones del canal pues brinda una visión desde la perspectiva del host.

**Características**

- Soporta los estándares USB 1.x y 2.0.
- No utiliza drivers filtro para capturar la información, que se cargan junto con los drivers del dispositivo, esto le permite poder registrar todas las peticiones de entrada/salida que suceden durante la enumeración o remoción de un dispositivo.
- La visualización de la información capturada puede ser en tiempo real, al finalizar la captura de la misma.
- La captura se puede iniciar desde el arranque del sistema operativo o se pueden utilizar comandos del menú o atajos de teclado para comenzar y detener la captura, además se puede detener la captura al suceder algún evento de interés.
- Se puede ver en detalle la composición (paquetes Setup, URB's, descriptores, etc.) y las distintas etapas (junto con sus estados) por las que pasan todas las peticiones USB de entrada/salida.
- Se pueden aplicar filtros para controlar la cantidad y tipo de información que se quiere capturar y visualizar.
- Toda la información capturada posee un sello de tiempo para asistir en el análisis de performance.

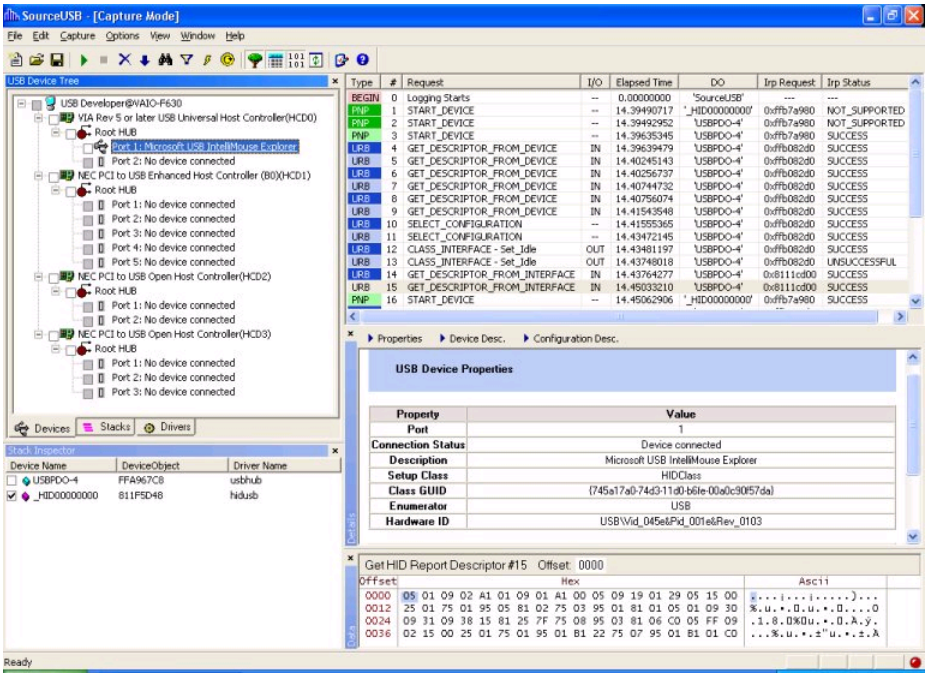



Figura 7.1: SourceUSB - Interfaz gráfica.

- Captura de todos las USB Request Blocks, URB's no documentados de Windows 2000, todas las URB's de los Hub y de la clase HID.
- Captura de todos los IRP de energía y Pnp y eventos remotos.
- Captura de las llamadas entre MiniPort-USBPort, Hub-USBPort.
- Captura de todas las IOCtrl internas de USB y IOCtrl modo usuario para los host controllers, hub y dispositivos HID.
- Se puede exportar toda la información capturada a archivos XML.
- Licencias y precios:
  - Single User: USD 595
  - 2-User: USD 995
  - 3-User: USD 1.495
  - 4-User: USD 1.995
  - 10-User: USD 3.995

**Requerimientos**

**Sistemas operativos soportados:**

- Windows 2000, Windows XP, Windows Server 2003 y Windows Vista

**Hardware requerido:**

- Placa base Intel o compatible de 32-bit.
- El archivo de captura es mapeado en memoria y el producto soporta desde archivos de 1 MB a 100 MB.
- USB Host Controller (UHCI, OHCI, EHCI).

### 7.1.2. HHD Software Ltd. - USB Monitor 2.36

Este software [10] es del tipo "sniffer" (husmeador) y está pensado para desarrolladores e ingenieros que diseñan, crean y conectan a una PC una variedad de dispositivos electrónicos por medio de USB. La herramienta permite capturar todas URB's transferidas desde el driver de un dispositivo al host controller y viceversa, para ello se vale de un driver filtro que se instala en la pila de drivers USB del sistema operativo logrando monitorear toda la información transferida y desplegar en un formato sencillo y legible al usuario de la herramienta. De esta forma se pueden detectar problemas a nivel del protocolo de comunicación o se puede utilizar como herramienta de ingeniería inversa o investigación de las características de un dispositivo de terceros.

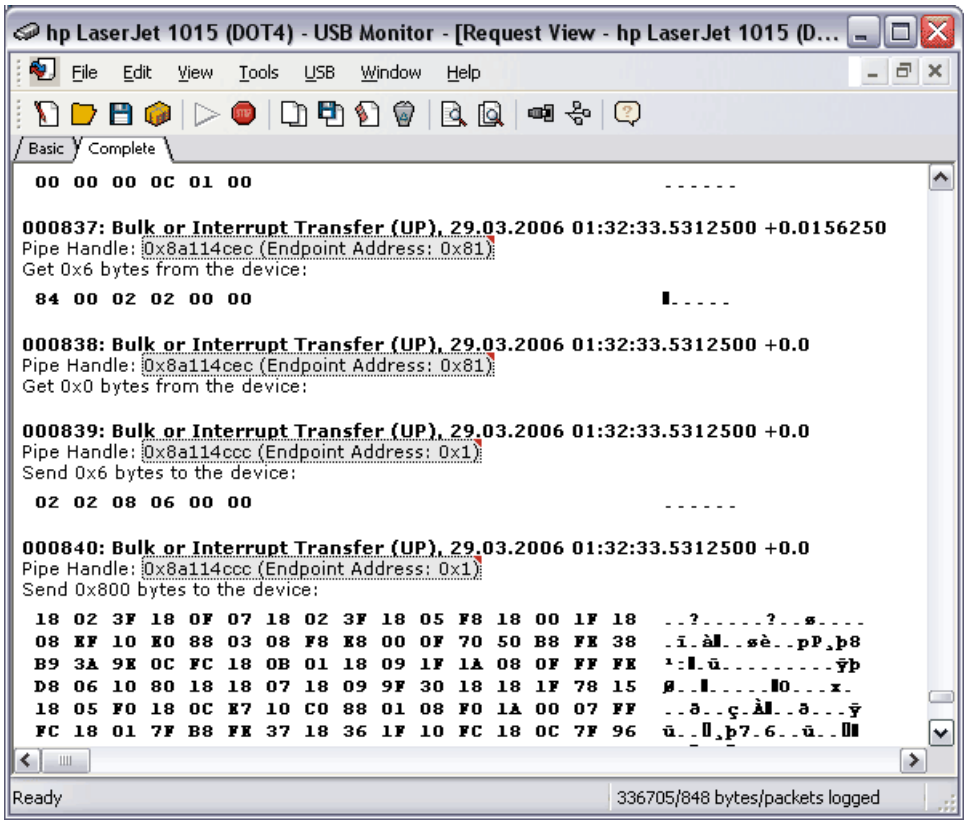



Figura 7.2: USB Monitor - Interfaz gráfica.

#### Características

- Intercepta toda la información leída de y escrita en el dispositivo USB.
- Decodifica cada URB que captura y muestra su contenido usando distintos esquemas desde básicos a detallados.
- Permite grabar toda la información capturada en bitácoras que luego se pueden ejecutar logrando una simulación de todo lo sucedido para un mejor estudio de la información.
- Aunque la computadora entre en un estado de reserva o hibernación, esta herramienta puede seguir ejecutando gracias a que es compatible con la tecnología ACPI de manejo de energía.
- Permite visualizar por medio de una estructura arborescente todos los host controllers, hubs, puertos y dispositivos conectados al PC.
- Para cada dispositivo conectado se puede visualizar el contenido de sus descriptores, incluyendo los de dispositivo, configuración, interfase, endpoint y de texto.

- Soporta la característica plug and play, lo que permite que cada vez que se conecta o desconecta un dispositivo se puedan ver todas las acciones que realiza el sistema operativo y el driver del dispositivo.
- Permite que la información capturada se exportada a varios formatos o copiada al block de notas de Windows.
- Tipos de licencias y precios:
  - Regular (Un usuario en varios PC o un único PC y varios usuarios): USD 80
  - Site (Cualquier cantidad de usuarios pero con registro de la dirección física de las PC): USD 720
  - Unlimited: USD 1040

**Requerimientos**

**Sistemas Operativos soportados:**

- Windows 2000, Windows XP, Windows Server 2003

**Hardware requerido:**

- Procesador de arquitectura x86
- 128 MB de memoria RAM
- USB Host Controller (UHCI, OHCI, EHCI).

### 7.1.3. Parallel Technologies Inc. - USBInfo 1.2

Esta herramienta es una analizador de performance de dispositivos USB [20] así como un visualizador de todos los componentes y dispositivos USB presentes en un PC. Esta visualización incluye una representación gráfica de los host controllers, hubs, puertos y dispositivos así como toda la información existente en el registro de Windows relacionada con cada componente y dispositivo. Para poder realizar las pruebas de performance se utiliza un driver filtro que monitorea y mide el tráfico de información entre el PC y los dispositivos. Como objetivos principales, éste producto busca el aumento de la performance de los dispositivos, así como la detección de problemas de instalación de los drivers de los mismos en el sistema operativo.

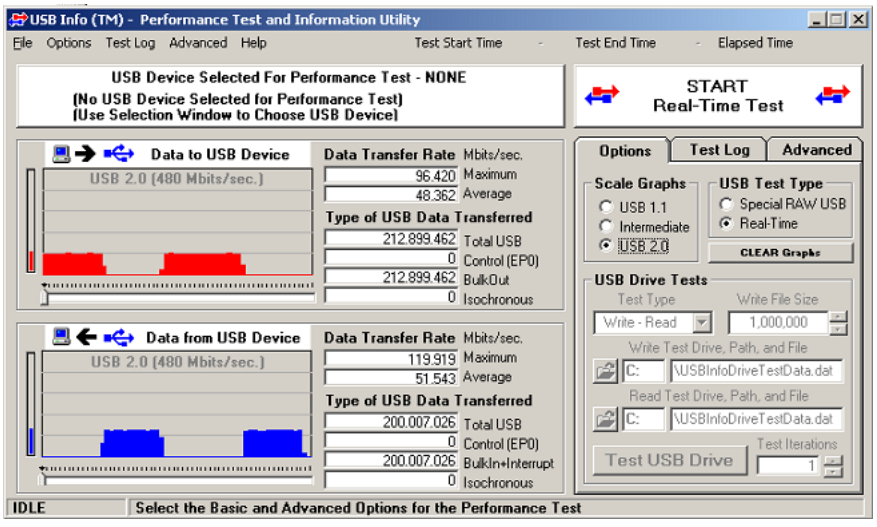



Figura 7.3: USBInfo - Test de Performance

**Características**

- Los dispositivos USB y sus componentes asociados se visualizan como unidades de disco.
- Se identifica claramante si los host controllers, hubs y dispositivos cumplen realmente con el estandar 2.0.
- Soporta tanto los drivers que vienen en los sistemas operativos de la plataforma Windows como los de otros fabricantes.
- Detecta hubs externos y los dispositivos conectados a ellos.
- Permite ejecutar pruebas de performance para cada dispositivo por separado.
- Permite ejecutar pruebas de performance para cada componente de un dispositivo compuesto.
- Permite visualizar en forma conjunta toda la información técnica del dispositivo, así como todas las entradas y claves existentes en el Registro de Windows.
- Permite registrar y salvar los resultados de las pruebas de performance para un posterior estudio y/o comparación con otros dispositivos.
- Encuentra entradas en el Registro de Windows que son incorrectas o estan incompletas o que no aplican más al hardware del dispositivo USB que esta actualmente instalado.
- Tipos de licencias y precios:
  - Personal Edition (Un usuario en un único PC): USD 20
  - Professional Edition (Un usuario en multiples PCs): USD 40

**Requerimientos**

**Sistemas operativos soportados:**

- Windows 98, Windows 98SE, Windows ME
- Windows 2000, Windows XP

**Hardware requerido:**

- Procesador compatible con la familia Intel x86 de al menos 1 Ghz de velocidad.
- USB Host Controller (UHCI, OHCI, EHCI).
- 128 MB de memoria RAM para Windows 98 / 98SE / ME y 256 MB para Windows 2000 / XP.
- 500 MB de espacio en disco.

### 7.1.4. Otras herramientas

**Fabula Tech Inc. - USB Monitor Pro 2.0**
Es un producto [9] muy similar al USB Monitor 2.36, algunas de sus diferencias estan en la interfaz gráfica la cual facilita bastante la comprensión de la información pues se puede visualizar en forma separada la información que proviene del dispositivo de la que fluye a él . Además permite capturar eventos, IOCTL internos, IRP's de Pnp y manejo de energía. Existen tres tipos de licencia y precios:

- Single (Una licencia por PC, menos de 50): USD 150
- Company Site (Cualquier cantidad de PC): Se debe consultar a sales@fabulatech.com.
- OEM (Redistribución ilimitada a terceros como parte del software): Se debe consultar a sales@fabulatech.com.

**Requerimientos**

- Sistemas operativos soportados: Windows 2000, Windows XP, Windows Server 2003
- USB Host Controller (UHCI, OHCI, EHCI).

**AGG Software - Advanced USB Port Monitor 2.1**

Esta herramienta de software [1] permite visualizar el tráfico de paquetes, decodificar los descriptores, detectar errores en los dispositivos o en los drivers y medir el rendimiento del dispositivo y del driver. Las principales características de este producto son:

- Driver filtro de modo núclo que soporta: WDM, WMI, manejo de energía y plug and play.
- Captura y visualización del tráfico de información en tiempo real.
- Vistas detalladas de los dispositivos USB asi como de los paquetes URB y los IRP del sistema de PnP y del sistema de manejo de energía.
- Exportación de la información a formatos XML, PDF y Microsoft Word.
- Tipo de licencia y precios (una licencia por PC):
  - Lite: USD 20
  - Standard: USD 48
  - Professional: USD 60

**Requerimientos**

- Sistemas operativos: Windows 2000, Windows XP, Windows Server 2003, Windows Vista
- Procesador compatible con la familia Intel x86 y con al menos 1 Ghz de velocidad.
- 256 MB de memoria RAM como mínimo y al menos 100 MB de espacio en disco.
- USB Host Controller (UHCI, OHCI, EHCI).

**Perisoft - Bus Hound 0.5**

Este software[22] fue pensado como un analizador de canales que captura peticiones de entrada/salida, protocolos y realiza mediciones de redimientos. Soporta una gran cantidad de canales como ser: USB 1.x & 2.0, SCSI & ATAPI, IDE & SATA, FireWire, Bluetooth, puerto serial, puerto paralelo y más. Sus principales usos son la inspección a bajo nivel de las peticiones de entrada/salida, detección de errores en los drivers o firmware de los dispositivos, protocolo para la ingeniería inversa de un driver, para adquirir medidas de rendimiento o para investigar las principales operaciones de un dispositivo. Existen dos tipos de licencias: por usuario a un precio de USD 800 y por sitio a un precio de USD 5.000.

**Requerimientos**

- Sistemas operativos soportados:
  - Windows 95, Windows 98, Windows ME
  - Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003
  - Windows XP Embebido
  - Windows Itanium (IA-64), Windows x64 (AMD-64)
- USB Host Controller (UHCI, OHCI, EHCI).

## 7.2. Analizadores vía Hardware

### 7.2.1. ElliSys Sàrl - USB Tracker 110

El *USB Tracker 110* [7] es un analizador de protocolo USB de bajo costo compatible con el estándar USB 2.0, que permite entre otras cosas: visualizar los paquetes enviados y recibidos, decodificar los descriptores y peticiones estándar, detectar errores en los dispositivos o drivers y medir sus rendimientos. Además, cuenta con un software de fácil uso para la visualización de toda la información del análisis, lo que facilita su utilización en el desarrollo de dispositivos USB, drivers o software embebido.


Figura 7.4: ElliSys - *USB Tracker 110*

**Características**

- Análisis del tráfico a velocidades: Low (1.5 Mbit/s), Full (12 Mbits/s) y combinado.
- Detección automática de la velocidad del dispositivo que se está analizando.
- Detección y medición de los estados del bus USB y protocolos de bajo nivel.
- Visualización amigable de las transacciones y en detalle de la información de las transferencias.
- Decodificación y visualización detallada de las peticiones estándar y de los descriptores de los dispositivos.
- Análisis no intrusivo del tráfico y con una alta impedancia en la prueba.
- Formato pequeño y robusto que facilita su traslado y alimentación por medio del bus USB.
- Precio y extensiones de software opcionales:
  - *USB Tracker 110*: USD 999.
  - Decodificación de clases estándar USB: USD 2.400.
  - Kit de desarrollo USB para análisis: USD 850.

**Requerimientos**

- Procesador Pentium III 600 MHz o superior.
- 128 MB de memoria RAM.
- Se recomienda que el PC tenga un host controller USB 2.0
- Resolución de pantalla: 800x600 pixeles en 256 colores como mínimo.
- Internet Explorer 5.0 o superior.
- Windows 2000 y Windows XP.

### 7.2.2. ElliSys Sàrl - USB Explorer 200

El *USB Explorer 200* [7] es un analizador de protocolo USB 2.0 de alto rendimiento, que ayuda a desarrollar de dispositivos USB de mejor calidad y en menos tiempo. Se encarga de monitorear los eventos USB y registrar el tráfico de información que intercambia por el cable USB, usualmente entre un PC y un dispositivo. Durante la captura del tráfico, se despliega en tiempo real en una estructura jerarquíca y ordenada cronológicamente la información de las transacciones, paquetes, así como la decodificación de las perticiones estándar o de clases y los descriptores USB.


Figura 7.5: ElliSys - USB Explorer 200

**Características**

- Análisis del tráfico a velocidades: Low (1.5 Mbit/s), Full (12 Mbits/s) y High (480 Mbits/s).
- Detección automática de la velocidad del dispositivo que se está analizando.
- Detección y medición de los estados del bus USB y protocolos de bajo nivel.
- Visualización amigable de las transacciones y en detalle de la información de las transferencias.
- Alto grado de decodificación de las peticiones estándar y descriptores.
- Alto grado de decodificación de las peticiones de las clases USB y descriptores.
- Diseñado para ser utilizado en cualquier lado por medio de un ordenador portátil.
- Análisis no intrusivo del tráfico y con una alta impedancia en la prueba.
- Formato pequeño y robusto que facilita su traslado y alimentación por medio del bus USB.
- Exportación de los datos analizados a varios formatos (XML, texto, raw, etc)
- Ediciones y Precio:
  - Standard: USD 2.999.
  - Professional: USD 5.999 (hardware trigger, decodificación de clases estándar USB y kit de desarrollo para análisis)
- Extensiones individuales a la edición Standard:
  - Hardware trigger: USD 1.500
  - Decodificación de clases estándar USD: USD 2.400
  - Kit de desarrollo para análisis: USD 850

**Requerimientos**

- Procesador Pentium III 600 MHz o superior.
- Como mínimo se necesitan 128 MB de memoria RAM, se recomienda el uso de 512 MB.
- Resolución de pantalla: 800x600 pixeles en 256 colores como mínimo.
- Host controller USB 2.0.
- Windows 2000 y Windows XP.

### 7.2.3. Catalyst Enterprise Inc. - Conquest USB

La serie Conquest [5] de analizadores de protocolo USB se presentan en 3 ediciones: clásica, estándar y avanzada. La primera fue diseñada para el análisis de dispositivos USB 1.1 (Low y Full speed) y permite la detección de errores y el registro e información de la información analizada. Las ediciones estándar y avanzada incluyen además: la decodificación de los paquetes de las clases USB o propias del usuario, la inyección de eventos en forma nativa o por un hardware trigger externo, análisis no intrusivo y filtrado en tiempo real de los eventos del bus USB.


Figura 7.6: Catalyst - Conquest USB

**Características**

- Análisis del tráfico a velocidades: Low, Full y High.
- Visualización amigable de las transacciones y en detalle de la información de las transferencias.
- Decodificación de las peticiones estándar y descriptores.
- Decodificación de las peticiones de las clases USB y definidas por el usuario.
- Análisis no intrusivo del tráfico, gracias a un circuito electrónico propietario de alta impedancia ubicado entre el bus y el analizador que permite no distorcionar las señales entre el PC y el dispositivo.
- Filtrado en tiempo real de datos, por medio de la definición de patrones de busqueda.
- Protocolo de detección de errores que permite detectar en tiempo real hasta 13 errores de protocolo y unos 20 más despues del proceso de captura.
- Diseño compacto y robusto que permite tener todos los componentes integrados y alimentación por medio del bus USB.
- Entradas / salidas externas ofrecen una forma de conectarse con una variedad de tipos de equipamientos.
- Exportación de los datos analizados a texto, ASCII o binario.
- Ediciones y precios:
  - Classic: USD 995
  - Standard: USD 2.950
  - Advanced: USD 5.950

**Requerimientos**

- Procesador Pentium III 500 MHz o equivalente.
- 128 MB de memoria RAM.
- Resolución de 1024x768 pixeles con 16bit de colores.
- Sistemas operativos Windows 2000 y Windows XP.
- Espacio en disco de 110 MB para el software SBAE y datos.

### 7.2.4. Catalyst Enterprise Inc. - SBAE-30B

El SBAE-30B [5] es un analizador del bus serial USB 2.0 que es capaz de analizar los datos de transferencias a velocidades de 480 Mb/s, realizar análisis de tiempos, rendimiento. Además posee las funcionalidades de: generar tráfico en el bus USB como lo hace un host controller, emular un dispositivo USB estándar y emular dispositivos USB OTG.


Figura 7.7: Catalyst - SBAE-30B

**Caracterísitcas**

- Análisis del tráfico a velocidades: low, full y high con detección automática de la adecuada para el dispositivo.
- Visualización amigable de las transacciones y en detalle de la información de las transferencias.
- Decodificación de las peticiones estándar y descriptores.
- Decodificación de las peticiones de las clases USB y definidas por el usuario.
- Análisis no intrusivo del tráfico, gracias a un circuito electrónico propietario de alta impedancia ubicado entre el bus y el analizador que permite no distorcionar las señales entre el PC y el dispositivo.
- Filtrado en tiempo real de datos, por medio de la definición de patrones de busqueda.
- Protocolo de detección de errores que permite detectar en tiempo real hasta 13 errores de protocolo y unos 20 más despues del proceso de captura.
- Diseño compacto y robusto que permite tener todos los componentes integrados y alimentación por medio del bus USB.
- Entradas / salidas externas ofrecen una forma de conectarse con una variedad de tipos de equipamientos.
- Soporte para el análisis de USB OTG.
- Analizador con dos puertos que permiten la completa caracterización de la conectividad de un Hub o puede ser usado como un canal secundario independiente de análisis.

- Ediciones y precios:
  - SBAE30-2x-ADC: USD 22.000
  - SBAE30-2x-EHD: USD 24.000

**Requerimientos**

- Procesador Pentium III 500 MHz o equivalente.
- 128 MB de memoria RAM.
- Resolución de 1024x768 pixeles con 16bit de colores.
- Sistemas operativos Windows 2000 y Windows XP.
- Espacio en disco de 110 MB para el software SBAE y datos.

# Parte III

# Opciones de Conectividad USB

# Capítulo 8

# Philips ISP1581

A continuación se muestra un resumen de principales características de la arquitectura del controlador de periféricos USB ISP1581 de Philips[23] :

- Compilante con la especificación USB 2.0 y soporta las velocidades hi-speed y full-speed.
- Soporta la detección de velocidad automática (480 Mbps o 12 Mbps).
- Soporta 7 Endpoints de entrada, 7 endpoints de salida (además del endpoint de control por defecto)
- Memoria FIFO integrada de 8 Kbytes con múltiples configuraciones .
- Soporta endpoints con doble buffering para incrementar el throughtput y transferencia de datos en tiempo real.
- Interfases:
  - Interfase de bus independiente para la mayoría de los microcontroladores/microprocesadores (12.5 MByte/s)
  - Interfase DMA de alta velocidad (12.8 Mbyes/s)
  - Interfase directa con periféricos ATA/ATAPI
- Conexión al bus USB controlada por software (SoftConnect tm)
- Transceiver de datos y regulador de voltaje de 3.3 V integrados.

## 8.1. Arquitectura del ISP1581

El ISP1581 es un controlador de periféricos de alta velocidad para el bus USB compilante con la version 2.0. Esta solución esta pensada para agregar la capacidad de comunicación a través de USB a un microcontrolador/microprocesador mediante una interfaz paralela. De esta forma se puede utilizar una aproximación modular para la implementación de un periférico junto a la reutilización de una arquitectura y firmware ya existente, lo cual reduce el tiempo de desarrollo y elimina sus riesgos asociados.

El ISP1581 soporta tanto la velocidad high-speed (480 Mbps) autodectectando si esta presente, y en caso de no estarlo permite un modo de compatibilidad con full-speed (12 Mbps). Está diseñado como un controlador de periféricos genérico, de tal forma que puede ser configurado como cualquiera las clases definidas dentro de la especificacion USB, como por ejemplo: Mass Strorage Devices, Communicaction Devices, Printing Devices, Human Interface Devices, etc. También incorpora la capacidad de conectarse/desconectarse por software (SoftConnect tm), lo que permite la reenumeración del mismo en caso de ser necesaria.

El ISP1581 tiene una interfaz de comunicación paralela configurable para la mayoría de los tipos de microcontrolador/microprocesador. Para aumentar el ancho de banda para la comunicación de datos, el manejador de DMA integrado puede ser invocado para transferir datos desde y hacia la memoria externa o dispositivos.

Este dispositivo cuenta con una memoria RAM FIFO interna de 8 Kbytes, que es compartida por los endpoints habilitados. Hay 7 endpoints de entrada, 7 endpoints de salida y 2 endpoints de control con un tamaño fijo de 64 bytes de largo. Cualquiera de los 7 endpoints de entrada o salida puede ser habilitado o deshabilitado en forma separada. El tipo de endpoint puede ser tanto de interrupt, isochronous o bulk y el tamaño de buffer de estos endpoints puede ser configurado individualmente dependiendo de los requerimientos de la aplicación. También tiene de manera opcional la posibilidad de utilizar doble buffering para incrementar el throughtput de datos a los endpoints. En la figura 8.1 se puede ver un diagrama de bloques de la arquitectura del ISP1581 y continuación se verá en detalle las interfases y componentes del mismo:

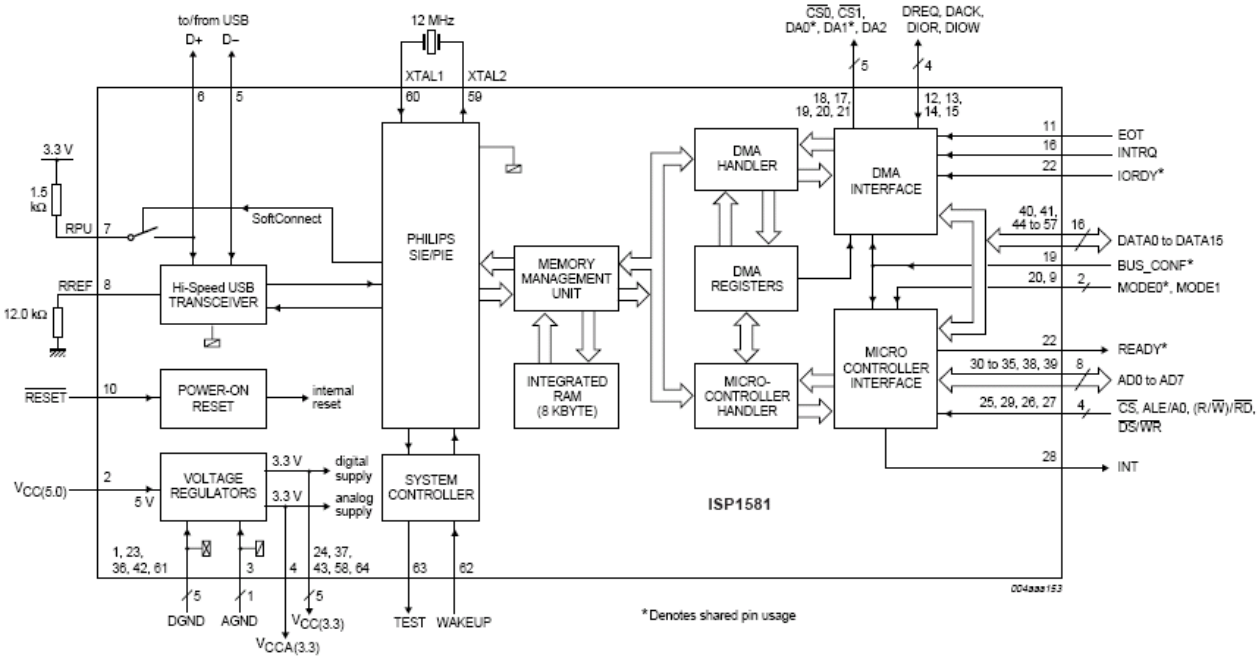



Figura 8.1: Diagrama de bloque de la arquitectura del ISP1581

1. Motor de interfase serial philips (SIE):
   El SIE implementa la capa de protocolo USB completa. Está completamente realizada en hardware para ser rápida y no requiere intervención del firmware. Sus funciones incluyen:

- Reconocimiento de patron de sincronización.
- Conversión paralela/serial.
- Bit (de-)stuffing.
- Chequeo y generación de CRC.
- Verificación y generación de identificación de paquete (PID) .
- Reconocimiento de dirección.
- Evaluación y reconocimiento de handshake.

2. Unidad de gestión de memoria (MMU) y RAM integrada:
El MMU y la RAM integrada proveen la conversión entre la velocidad de USB (full-spped: 12 Mbps, high-speed: 480 Mbps) y el manejador del microcontrolador o el manejador de DMA. Los datos del bus USB son guardados en la RAM integrada, la cual es borrada sólo cuando el microcontrolador ha leído/escrito todos los datos desde/hacia el buffer del endpoint correspondiente o cuando el manejador de DMA ha leído/escrito todos los datos desde/hacia el buffer del endpoint. Hay un total de 8 Kbytes de memoria para buffering.

3. Interfaz y manejadores para microcontroladores/microprocesadores.
La configuración de la interfaz se realiza por medio de pines durante el encendido. Ellos son BUS_CONF, MODE1 y MODE0 para lograr compatibilidad con la mayoria de los tipos de interfaz. Dos configuraciones de bus seleccionadas con la entrada BUS_CONF durante el encendido:

   - Modo procesador Generico (BUS_CONF=1) mostrado en la figura 8.2:
     - AD[7:0]: bus de direcciones de 8 bits (selecciona el registro destino)
     - DATA[15:0]: bus de datos de 16 bits (compartido por el procesador y el DMA)
     - Señales de control: /CS y
       - Si el pin MODE0=1 entonces los pines /RD y /WR son los strobes de lectura y escritura (estilo 8051)
       - Si el pin MODE0=0 entonces los pines R//W y /DS representan la direccion y el strobe de datos (estilo Motorola)
     - Interfase DMA (modo genérico esclavo solamente): usa las lineas DATA[15:0] como bus de datos y DIOR y DIOW dedicadas al strobe de lectura y escritura.
     - MODE1 debe ser igual a 1.
   - Modo Split Bus (BUS_CONF=0) ilustrado en la figura 8.3 en su versión esclavo:
     - AD[7:0]: bus local del microprocesador (direccionamiento y datos multiplexados)
     - DATA[15:0]: bus de datos de 16 bits DMA
     - Señales de control: /CS y
       - El pin MODE0 tiene la misma semántica en en el modo de procesador genérico.
       - Si el pin MODE1=0 entonces el pin ALE es usado para latchear las direcciones multiplexadas en los pines AD[7:0].
       - Si el pin MODE1=1 entonces el pin A0 es usado para indicar dirección o datos.
     - Interfase DMA (modo maestro o esclavo): usa DIOR y DIOW como pines dedicados a los strobes de lectura y escritura.

4. Interfase DMA y Manejador DMA:
El manejador del microcontrolador, permite al microcontrolador externo acceder al conjunto de registros en el SIE, así como también para el manejador de DMA. La inicialización de la configuración de DMA es realizada a través del manejador del microcontrolador.El bloque de DMA se puede dubdividir en dos bloques: el manejador DMA y la interfaz DMA. El firmware debe escribir al registro de comandos de MDA para empezar una transferencia DMA. El comando de operación (operation code) (opcode) determina si debe comenzar

una transferencia genérica DMA, PIO, MDMA o UDMA. El manejador realiza la comunicación con la RAM interna (FIFO) mientras es usada por el núcleo USB. Luego de ser recibido el comando DMA, el manejador DMA dirige los datos desde la RAM interna hacia el DMA externo o desde el DMA externo hacia la RAM interna.
La interfaz DMA configura los tiempos y el handshake del DMA. Los datos pueden ser transferidos usando strobes DIOR y DIOW o con los handshakes DACK y DREQ.
Tenemos 2 opciones para la interfaz DMA:

- Interfaz de almacenamiento basada en IDE: los modos DMA son PIO (E/S Paralela), MDMA (DMA multipalabra; ATA), y UDMA (Ultra DMA , ATA).
- Para una interfaz DMA genérica, los modos DMA que pueden ser usados son : DMA genérico (esclavo) o MDMA (maestro)

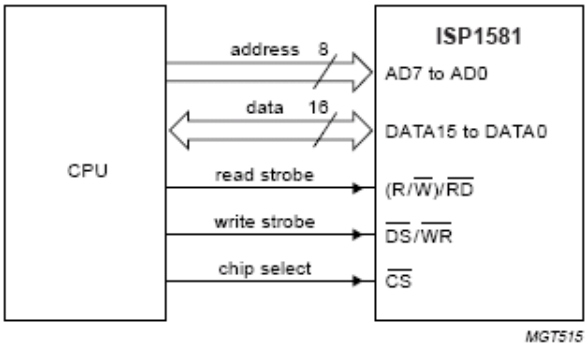



Figura 8.2: Modo procesador genérico

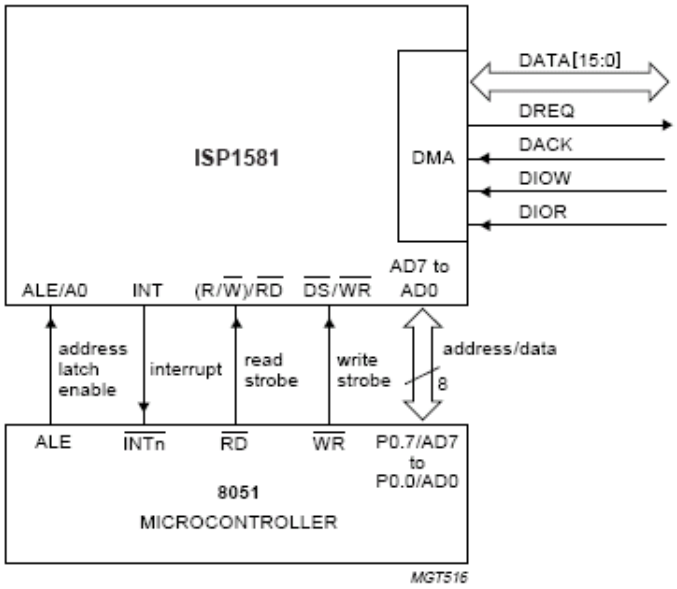



Figura 8.3: Modo split bus (modo esclavo)

# Capítulo 9

# Texas Instruments TUSB3210

A continuación se muestra un resumen de principales características de la arquitectura del microcontrolador TUSB3210 de Texas Instruments[25] :

- Arquitectura: CISC, basado en el núcleo del CPU 8052.
- Memoria de programa:
  - 6 K de ROM para funciones de USB y Bootloader (grabado de fábrica).
  - 8 K de RAM para el firmware de usuario.
- Memoria SRAM:
  - 256 bytes de uso general.
  - 512 de memoria compartida con modulo USB.
- Periféricos:
  - 4 Puertos de 8 bits de Entrada/Salida.
  - 3 Módulos de Timer/Counter.
  - 1 Controlador Maestro I2C
- USB:
  - Interfaz USB V2.0 que soporta alta velocidad.
  - Hasta 3 endpoints de entrada y 3 de salida.
  - Soporta transferencias: Interrupt y Bulk.
- Características del microcontrolador especiales:
  - Bootloader que permite cargar el firmware a través de I2C o del PC a través de USB
  - Soporte multiproducto.
  - Niveles de prioridad en las interrupciones.
  - Modo de bajo consumo.

## 9.1. Arquitectura

La arquitectura del TUSB3210 es CISC y basa en la del núcleo 8052. En la figura 9.1 se muestra un diagrama de los componentes de este microcontrolador

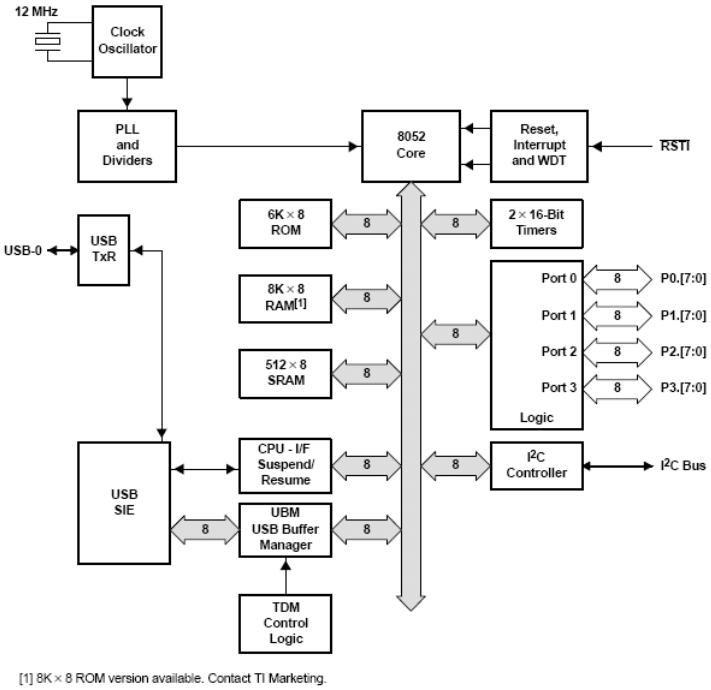



Figura 9.1: Diagrama del microcontrolador TUSB3210

### 9.1.1. Modos de operación y mapas de memoria asociados

Este microcontrolador cuenta con 2 modos de operación, el modo de boot y el modo normal. En cada uno de estos modos el mapa de memoria es diferente.

Cuando el bit SDW es igual a 0 (modo boot) tenemos los 6k de ROM mapeados en el área de código en 2 secciones, a partir de la direccion 0h y duplicada a partir de la 8000h. En el espacio de datos tenemos 8k de RAM (que permiten lectura y escritura) a partir de la direccion 0, los 512 bytes de RAM compartida con el módulo de USB a partir de los FD80h y un area para los Buffers, MMR y entrada/salida mapeada ubicada en la parte más alta de memoria entre los FF80h y los FFFFh

Cuando el bit SDW es igual a 1 (modo normal), los 8K de RAM pasan a la sección de código en la dirección 0 y son sólo de lectura. Luego tenemos sólo una copia de los 6K de ROM a partir de los 8000h, y la RAM compartida, los Buffers, MMR y Entrada/Salida mapeadas permanecen en el area de datos en la misma posición que en el modo de boot. En la figura 9.2 se muestran los mapas de memoria para cada modo de operación.

### 9.1.2. Operación de Boot

Luego del un reset del microcontrolador comienza en el modo de boot y ejecutando la primera instrucción del area de código en ROM. Inicialmente el dispositivo estará desconectado del bus USB debido a que la resistencia pull-up en D+ desactivada. Esta resistencia es controlada por el bit CONT, si está en 0 el dispositivo esta desconectado, si se setea en 1 se activa la resistencia pull-up conectandose al el dispositivo al bus. La finalidad del código de boot en ROM es cargar en la RAM el código del firmware y para ello ejecuta la siguiente secuencia de eventos:

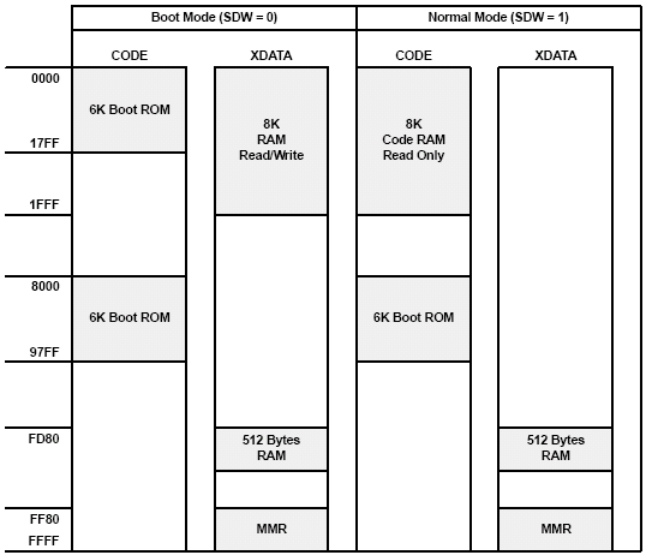



Figura 9.2: Mapa de memoria en los dos modos de operación del TUSB3210

1. Se verifica si hay una EEPROM conectada al bus I2C que contenga el código de firmware, si es así se procede a copiar el firmware a los 8K de RAM y se continua en el paso 3.
2. Se procede a conectar el dispositivo al bus de USB (se setea CONT en 1) y se obtiene el firmware que es copiado a los 8K de RAM, luego se desconecta del bus.
3. Luego que el firmware esta debidamente cargado en la memoria RAM se procede a realizar el cambio al modo normal, seteando el bit SDW en 1 y el CPU comienza a ejecutar el firmware.

Luego de que se setean adecuadamente módulo de USB por parte del firmware, éste debera setear el bit CONT en 1 para conectarse al bus y comenzar el proceso de enumeración normal.

### 9.1.3. Interrupciones

Toda las interrupciones no estandares del 8052 (USB, I2C, etc) son conectadas a través de una compuerta OR para generar la señal interna /INT0. Las señales externas /INT0 e /INT1 no son utilizadas. Se provee un registro de vector de interrupción para identificar todas estas fuentes de interrupción y poder despachar adecuadamente a la corresp diente rutina.

Luego de un reset el CPU comineza con la ejecución de la posición de memoria 0000H. Como se vé en la figura 9.3, cada interrupción tiene una posición fija en la memoria de programa. La interrupción interna 0 tiene asignada la posición 0003H.

| INTERRUPT SOURCE | DESCRIPTION | START ADDRESS | COMMENTS |
|---|---|---|---|
| ET2 | Timer-2 interrupt | 002Bh | |
| ES | UART interrupt | 0023h | |
| ET1 | Timer-1 interrupt | 001Bh | |
| EX1 | Internal $\overline{\text{INT1}}$ or INT1 | 0013h | Used for P2[7:0] interrupt |
| ET0 | Timer-0 interrupt | 000Bh | |
| $\overline{\text{INT0}}$ | Internal $\overline{\text{INT0}}$ | 0003h | Used for all internal peripherals |
| Reset | | 0000h | |

Figura 9.3: Mapa de interrupciones

Las posiciones de las rutinas de interrupción estan separadas por intervalos de 8 bytes: 0003H para la Interrupción Externa 0, 000BH para Timer 0, 0013H para Interrupcion Externa 1, 001Bh

para el Timer 1, etc. De esta forma si se tiene rutinas de manejo de interrupciones cortas pueden entrar en esos intervalos.

En los dispositivos de 8 Kbytes, si /EA = Vcc selecciona las direcciones 0000H hasta la 1FFFH como interna y las direcciones 4000H a FFFFH como externa. Si el pin /EA = Vss entonces todos los fetchs son diregidos a la rom externa. El strobe de lectura para la ROM externa, /PSEN, es usada en todos los fetchs a memoria externa, y no se activa para los fetchs de memoria interna. El diagrama de conexión con memoria externa se muestra en la figura 9.4.

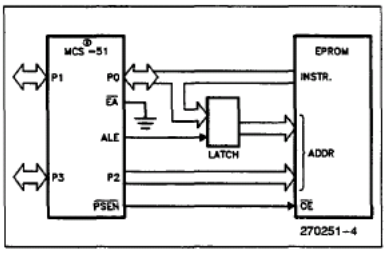



Figura 9.4: Acceso a la memoria externa.

Para tener acceso a la memoria externa se utilizan 16 lineas de E/S (puertos 0 y 2) dedicados a la funciones de bus durante los fetchs. El puerto 0 tiene las funciones de direccionamiento y lectura de datos de manera multiplexada. El funcionamiento es el siguiente: primero direcciona el byte bajo del contador de programa (PCL) y luego va a un estado flotante esperando la llegada de datos de la memoria de programa externa. Durante el tiempo que la señal del byte bajo del PC es valida, la señal ALE (Address Latch Enable) inserta este byte en el address Latch. Mientras tanto el puerto 2 direcciona la parte alta del Contador de Programa (PCH), entonces el pin /PSEN envia la señal para el Output Enable (/OE) de la memoria externa y el byte de datos es leido por el microcontrolador. Los accesos a memoria externa son siempre de 16 bits de ancho, a pesar de que la memoria usada sea menos de 64 Kbytes.

### 9.1.4. Acceso a la memoria de datos

En la figura 9.5 se muestra la configuración de hardware para acceder hasta 2 Kbytes de RAM externa. En este caso el CPU está ejecutando código de la ROM interna. El puerto tiene las mismas funciones que en acceso a memoria de programa externa y del puerto 2 se utilizan 3 lineas para paginar la RAM. También deben utilizarse las señales que genera el CPU en el puerto 3: /RD y /WR durante el acceso a la RAM externa.

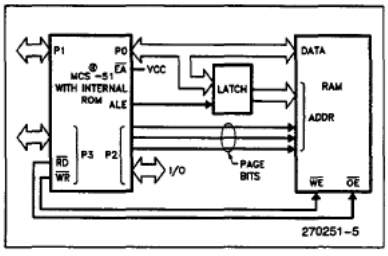



Figura 9.5: Acceso a RAM externa.

Puede haber hasta 64 Kbytes de memoria de datos externa que puede ser direccionada con 1 o 2 bytes de ancho de palabra.

La memoria de datos interna esta dividida en 3 bloques generalmente referidos como Lower 128, Upper 128 y espacio SFR. El direccionamiento de la RAM interna es de 1 byte de ancho, por

lo que sólo puede tener un espacio de 256 bytes, sin embargo los modos de direccionamiento de la RAM interna permite acomodar 384 bytes usando un truco simple. Accesos directos mayores de la direccion 7FH acceden a un espacio de memoria y el direccionamiento indirecto mayor a 7FH acceden a un espacio de memoria diferente. En la figura 9.6 se pueden ver ocupando el mismo bloque de direcciones, pero ellas son entidaddes separadas.

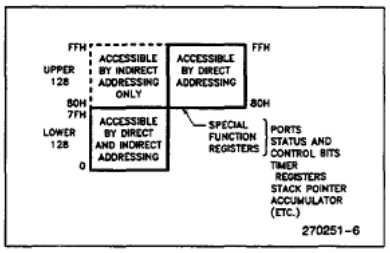



Figura 9.6: Memoria interna de datos

Los Lower 128 bytes de RAM estan mapeados segun la fugura 9.7. Los 32 bytes más bajos se agrupan en 4 bancos de 8 registros.

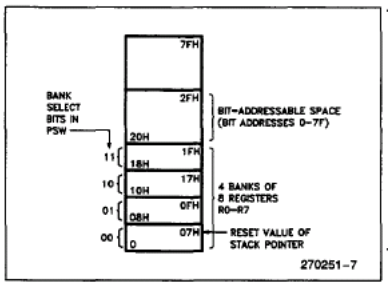



Figura 9.7: Lower 128 bits.

Instrucciones de programa realizan llamadas a estos registros como R0 hasta R7. Dos bits en la palabra de status (PSW) seleccionan que banco esta en uso. Esto permite un uso de espacio de código más eficiente, debido a que las instrucciones de registro son más cortas que las que usan direccionamiento directo.

Los siguientes 16 byes arriba de los bancos de registro forman un bloque de espacio de memoria direccionable por bit. El set de instrucciones incluye una amplia selección de instrucciones para manejo de a bits que acceden a los 128 bits de esta área directamente (00H a 7FH) .

Todos los bytes en el Lower 128 pueden ser accedidos mediante direccionamiento directo o indirecto. La parte Upper 128 puede ser accedida sólo por direccionamiento indirecto.

La figura 9.8 muestra parte del espacio de Registro de Funciones Especiales (FSR) dentro de los cuales estan los latches de los puestos de entradas salida, registros de Timers, controles de periféricos, etc. Dieciseis de las direcciones dentro del espacio de memoria SFR son direccionables tanto por bit como por byes. Ellas son las direcciones que terminan en 000BH.

### 9.1.5. Conjunto de Instrucciones

Todos los mienbros de la familia MCS-51 ejecutan el mismo set de instrucciones. Este set esta optimizado para aplicaciones de control de 8 bits que incluye una variedad de accesos rapidos a RAM interna para facilitar operaciones en pequeñas estructuras de datos. Este set de instrucciones posee además un gran soporte para variables de un bit como un tipo de datos

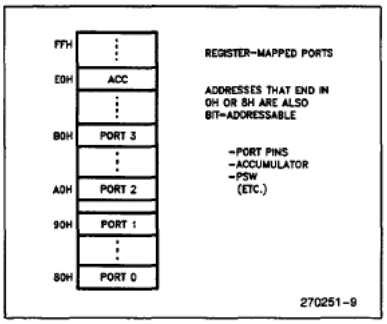



Figura 9.8: Espacio FSR

separado, permitiendo manipulaciones de a bit en sistemas de control y lógica que requieran procesamiento booleano.

Antes de empezar con los modos de direccionamiento daremos una descripción del PSW que se usa en muchas de las instrucciones del set. El PSW se encuentra en el espacio de SFR y contiene varios bits de status que reflejan el estado actual del CPU. Los bits que contiene y sus significado se muestran en la figura 9.9.

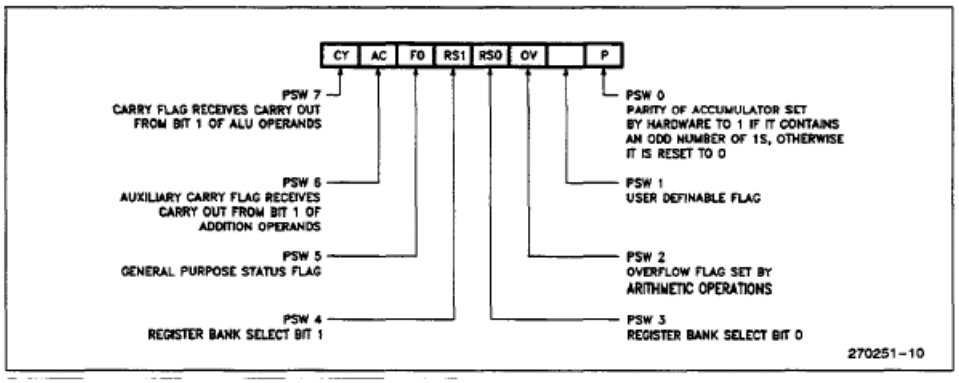



Figura 9.9: Bits de status

El mismo contiene el bit de Carry, el Carry Auxiliar (para operaciones BCD), 2 bits para seleccionar el banco de registros (RS0 y RS1), el bit de Overflow, el bit de Paridad y 2 bits extras definibles por el usuario.

#### 9.1.5.1. Modos de direccionamiento

- **Direccionamiento directo**:
  En direccionamiento directo el operando es especificado con un campo de direccion de 8 bits en la instrucción. Sólo la RAM de datos interna y los SFRs pueden ser directamente direccionados en este modo.

- **Direccionamiento indirecto**:
  El direccionamiento indirecto especifica un registro que contiene la direccion del operando. La RAM interna y externa pueden ser indirectamente direccionadas. El registro de dirección para direccionamiento de 8 bits puede ser R0 o R1 del banco seleccionado, o el Stack pointer. El registro de direccionamiento para direcciones de 16 bits puede ser sólo el registro "data pointer" de 16 bits DPTR.

- **Instrucciones de Registro**:
  Los bancos de registro que contienen los registros R0 a R7, pueden ser accedidos por algunas instrucciones que contienen un valor de 3 bits especificando el registro en el código

de operación. El banco accedido es el especificado por los 2 bits RS0 y RS1 que están en el PSW.

- **Instrucciones de registro especificos**:
  Algunas instrucciones son específicas a ciertos registros, por ejemplo las que siempre operan con el Acumulador, Puntero a Datos, etc, en donde no se necesita direccionar un registro, esta implícito en el opcode.

- **Constantes inmediatas**:
  El valor de una constante puede seguir a un opcode en la memoria de programa, por ejemplo:
  MOV A, #100
  Carga el acumulador con el decimal número 100.

- **Direccionamiento indexado**:
  Sólo la memoria de programa puede ser accedida con direccionmiento indexado, y sólo puede ser de lectura. Esta pensado para leer tablas de look-up. Un registro base de 16 bits (DPTR o PC) apunta a base de la tabla, y en el acumulador se setea el numero de entrada en la tabla. Otro tipo de direccionamiento indexado usado en el "case jump". En este caso la dirección de destino es calculada como la suma del puntero base y el dato cargado en el acumulador.

#### 9.1.5.2. Aspectos importantes del set de instrucciones

Esta microcontrolador cuenta con multiplicación y división de 8 bits realizada por hardware. Esto acelera programas que hacen uso de esta funcionalidad, ya que en hardware se pueden obtener tiempos de ejecución más rápidos que si se hiciera en software.

Otra característica interesante es que el manejo de stack es manejado por software, a diferencia de otros microcontroladores en donde tiene un área en hardware reservado de tamaño fijo a tales efectos, y no puede manipularse a través de software..

Por último este microcontrolador cuenta con una gran cantidad de instrucciones para manejo de variables de 1 bit como move, set, clear, complement, or y and.

### 9.1.6. Periféricos incorporados

En esta sección se describen los periféricos que incorpora el microcontrolador con detalles sobre sus caracterísitcas.

#### 9.1.6.1. Cuatro puertos de 8 bits bidireccionales

Los cuatro puertos en el 8051 son bidireccionales. Como ya se dijo antes los puertos 0 y 2 pueden ser utilizados para acceso a memoria externa, en cuyo caso dejan de utilizarse como puertos de entrada y salida generales. Todos los pines del puerto 3 son multifuncionales (2 entradas externas de timers 0 y 1, 2 del USART, 2 de interrupciones externas y 2 de acceso de memoria de datos) así como también 2 del puerto 0 (2 entradas externas de timer 2).

Todos los puertos poseen un schmitt-trigger para la entrada, resistencias de pull-up y salidas open-drain que pueden proveer hasta 8 miliamperes.

#### 9.1.6.2. Controlador Maestro I2C

Este microcontrolador cuenta con un módulo I2C que puede ser utilizado tanto en la etapa de booteo para leer una eeprom externa como para comunicarse en modo run con otros periféricos. El TUSB3210 sólo soporta una relación maestro-esclavo, por lo que no soporta el arbitraje del bus. Está implementado totalmente en hardware. Este puede comunicarse con dispositivos utilizando velocidades de bus de 100khz o 400khz, este módulo es de fácil manejo ya que para interactuar con el se utilizan 4 registros de 8 bits. Ellos son: uno para status y control, otro registro para direccionamiento, y luego 2 registros más, uno para entrada y otro para salida de datos.

#### 9.1.6.3. Timers/Counters

Como parte del núcleo del 8052, este microcontrolador posee 3 módulos de Timer/Counter de 16 bits que pueden ser utilizados en varios modos y para aplicaciones específicas como por ejemplo generadores de baud rate.

En la función de timer, su registro (formado por 2 bytes THn y TLn) es incrementado en cada ciclo de máquina. Como cada ciclo de máquina consiste en 12 períodos de oscilación, la velocidad de conteo es de 1/12 de la frecuencia del oscilador.

En la función de Contador, el registro es incrementado en respuesta a una transición de 1 a 0 del correspondiente pin de entrada externa.

**Timers 0 y 1:**

A continuación veremos brevemente los 4 modos de los timers 0 y 1:

- **Modo 0**: En este caso se utiliza el el módulo como un contador de 8 bits con un prescaler de 5 bits. Cuando se realiza una transición de todos los bits de 1s a 0s se setea labandera de interrupción TFn.
- **Modo 1**: Similar al modo 0, pero en este caso se utiliza el registro completo de 16 bits.
- **Modo 2**: Configura el registro de timer como de 8 bits con recarga automática. En este caso el overflow no sólo setea la bandera de interrupción TF1 sino que recarga el valor de TLn con el contenido de THn, manteniendo THn incambiado.
- **Modo 3**:
  - Aqui se hace una diferencia entre el timer 0 y el timer 1. Para el caso del timer 1 simplemente mantiene el contenido del registro, similar a desactivarlo. Para el caso del Timer 0, establece TL0 y TH0 como 2 contadores separados.
  - Este modo sirve para aplicaciones que requieran de un timer/counter de 8 bit extra.

**Timer 2:**

Este timer cuenta con 3 modos de operación: capture, recarga automática y como generador de baud rate.

#### 9.1.6.4. Interfaz serial

La interfase serial o USART es full duplex y cuenta con un buffer en la recepción, por lo que puede comenzarse a recibir un segundo byte antes de leer el dato previamente recibido en el registro de entrada. Los registros de entrada y salida son ambos accedidos a través del mismo SFR SBUF. Cuando el SBUF es escrito, éste carga el registro transmisor y cuando se lee SBUF, se accede físicamente a un registro de recepcion separado.

La interfase puede operar en 4 modos, y se utilizan los pines RXD y TXD para recepción y salida de datos:

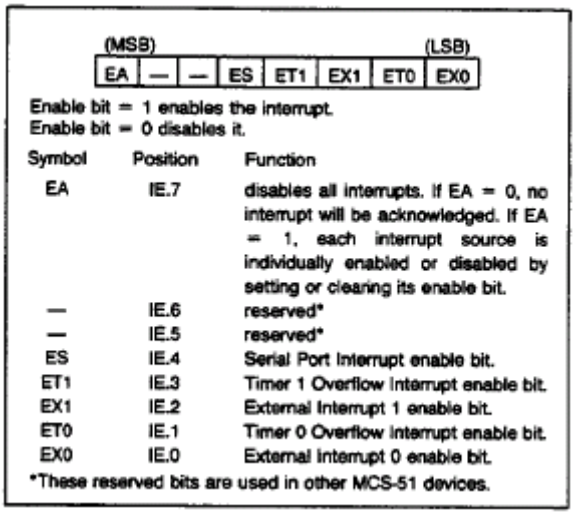
- **Modo 0**: Son enviados 8 bits de datos (LSB primero) con baud rate fijo a 1/12 de la frecuencia del oscilador.
- **Modo 1**: 10 bits son transmitidos/recibidos: un bit de start (0), luego 8 bits de datos (LSB primero) y finalmente un bit de stop (1). El baud rate es variable.
- **Modo 2**: 11 bits son transmitidos/recibidos: un bit de start (0), luego 8 bits de datos (LSB primero), un noveno bit de datos programable y finalmente un bit de stop (1).
  El baud rate es programable y puede ser 1/32 o 1/64 de la frecuencia del oscilador.
- **Modo 3**: Similar al modo 2 pero el baud rate es variable.

Los modos 2 y 3 permiten la comunicación entre procesadores, utilizando el noveno bit para diferenciar entre un byte de direccionamiento o uno de datos. De esta forma se puede lograr por software la espera por parte de los microprocesadores esclavos, en busca de su dirección en una primera instancia. En caso de ser la propia se continua procesando los bytes de datos, sino los bytes de datos son ignorados.

#### 9.1.6.5. Interrupciones

El núcleo 8052 provee 6 fuentes de interrupción, 2 interrupciones externas, 3 interrupciones de timer y una interrupción del puerto serial. La siguiente es la estructura de interrupciones del 8052.

Cada fuente de interrupción puede ser individualmente habilitada o deshabilitada mediante el seteo de los bits del SFR llamado IE (Interrupt Enable). Ademas este registro también contiene un bit para deshabilitar todas las interrupciones a la vez. Esto se muestra en la figura 9.10.



| (MSB) | | | | | | | (LSB) |
|---|---|---|---|---|---|---|---|
| EA | — | — | ES | ET1 | EX1 | ET0 | EX0 |

Enable bit = 1 enables the interrupt.
Enable bit = 0 disables it.

| Symbol | Position | Function |
|---|---|---|
| EA | IE.7 | disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit. |
| — | IE.6 | reserved* |
| — | IE.5 | reserved* |
| ES | IE.4 | Serial Port Interrupt enable bit. |
| ET1 | IE.3 | Timer 1 Overflow Interrupt enable bit. |
| EX1 | IE.2 | External Interrupt 1 enable bit. |
| ET0 | IE.1 | Timer 0 Overflow Interrupt enable bit. |
| EX0 | IE.0 | External Interrupt 0 enable bit. |

*These reserved bits are used in other MCS-51 devices.

Figura 9.10: Habilitación de interrupciones

Cada fuente de interrupción puede ser programada individualmente a uno de los dos niveles de prioridad, utilizando el bit en el SFR llamado IP (interrupt priority). Una interrupción de baja prioridad puede ser interrumpida por una de alta prioridad, pero no por otra interrupción de baja prioridad. Una interrupción de alta prioridad no puede ser interrumpida por ninguna otra.

Si dos fuentes de interrupción de diferente prioridad son recibidas en forma simultanea, el pedido de la de alta prioridad es servida. En el caso que dos interrupciones con el mismo nivel de prioridad, una segunda estructura de prioridad es determinada por el orden del polling. Esto se muestra en la figura 9.11.

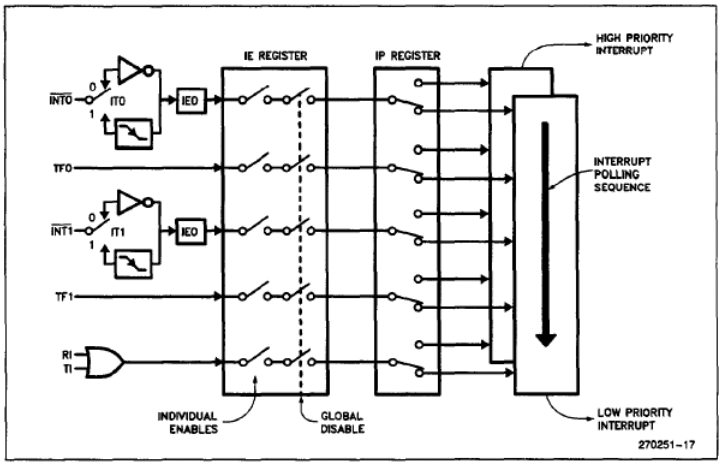



Figura 9.11: Prioridad en interrupciones.

En el caso del TUSB3210 todas las otras interrupciones que no son estandar en el 8052 (USB, I2C, etc) son conectadas como un OR para generar una interrupcion interna /INT0, lo que se muestra en la figura 9.12. Ademas las interrupciones /INT0 e /INT1 no son usadas y /INT0 debe ser programada como interrupción activada por nivel bajo. Para determinar cual fue la fuente de interrupción se cuenta con un registro vector de interrupciones (VECINT) que debe ser leido para despachar la rutina de atencion a interrupción adecuada. Este vector es de 8 bits y esta dividido de la siguiente manera:

Bits 7 al 4 como G[3:0] que define el grupo de la interrupción
Bit 3 al 1 como I[2:0] que define la fuente de la interrupción
Bit 0 reservado.

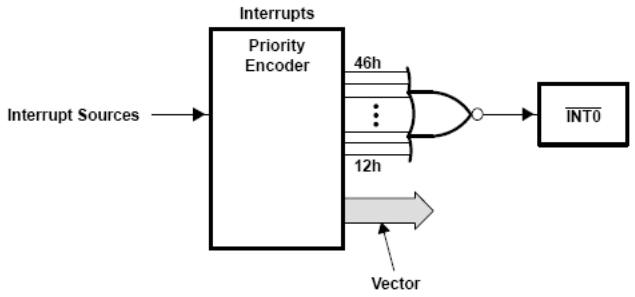



Figura 9.12: Otras interrupciones no estandar.

En la figura 9.12 se muestra como está conectada una lógica de prioridades en donde el campo I define la fuente de interrupción dentro de un grupo (FIFO) y el cambo G define el número de grupo. El G0 tiene la menor prioridad y G15 tiene la mayor.

Otra fuente de interrupciones son las entradas del puerto 2 que estan conectadas con un OR lógico o el bit 3 del puerto 3 para generar la interrupción /INT1 que también es programada como activa por nivel bajo, y se muestra en la figura 9.13.

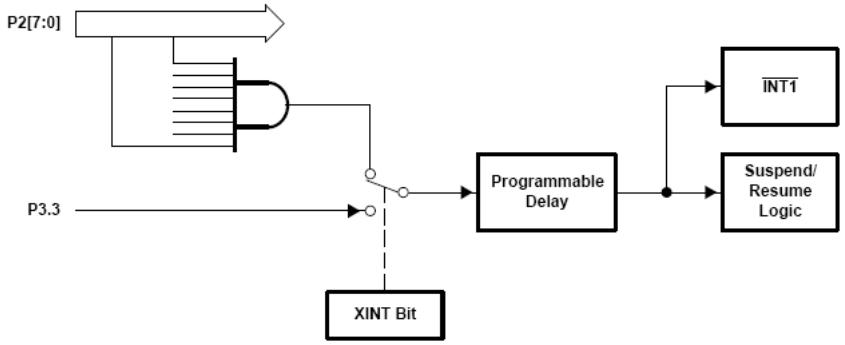



Figura 9.13: Interrupciones del puerto 2

#### 9.1.6.6. Característica especial del microcontrolador

- **Soporte multiproducto:**
  Se cuenta con 4 pines de entrada dedicados para seleccionar el uno de las identificaciones VID/PID guardadas en memoria. Esto sirve para proveer a los manofacturadores de equipamiento original (OEM) tener un dispositivo programado en ROM para soportar 16 diferentes lineas de producto, pudiendo selecconarlo facilmente seteando como 1 o 0 esas lineas.

# Capítulo 10

# Microchip PIC18F4550

Este dispositivo[18] pertenece a la gama alta de microcontroladores que desarrolla Microchip. A modo de resumen se presentan sus características más sobresalientes.

- Arquitectura: Harvard.

  Memoria de programa: 32k (16386 instrucciones).

  Memoria RAM: estática 2048 bytes.

  EEprom: 256 bytes.

- Periféricos:

  5 Puertos de Entrada/Salida.

  4 Módulos de Timer.

  1 Módulo Capture/Compare/PWM.(CCP)

  1 Módulo Enhanced Capture/Compare/PWM (ECCP).

  1 Módulo Enhanced USART (EUSART).

  Master Synchronous Serial Port (MSSP) que soporta SPI e I2C.

  Conversor analógico/digital de 10 bits y hasta 13 canales.

  Dos comparadores analógicos.

- USB:

  Interfaz USB V2.0 que soporta baja y alta velocidad.

  Soporta transferencias: Control, Interrupt, Isochronous y Bulk.

  Hasta 32 endpoints (16 bidireccionales).

  Streaming Parallel Port (SPP).

- Modos de Manejo de Energía:

  Run: CPU y periféricos encendidos.

  Idle: CPU apagado y periféricos encendidos.

  Sleep: CPU y periféricos apagados.

- Características de microcontrolador especiales:

  12 modos de reloj y la posibilidad de utilizar el modulo USB y el CPU a diferentes velocidades.

  Memoria de Programa auto-programable por software.

  Niveles de prioridad en las interrupciones.

  Multiplicador 8x8 en hardware y ejecuta en un solo ciclo.

  Una sola fuente 5V, programación y depuración en el circuito (ICSP e ICD).

En la figura 10.1 podemos ver un diagrama en bloques de los componentes del microcontrolador.

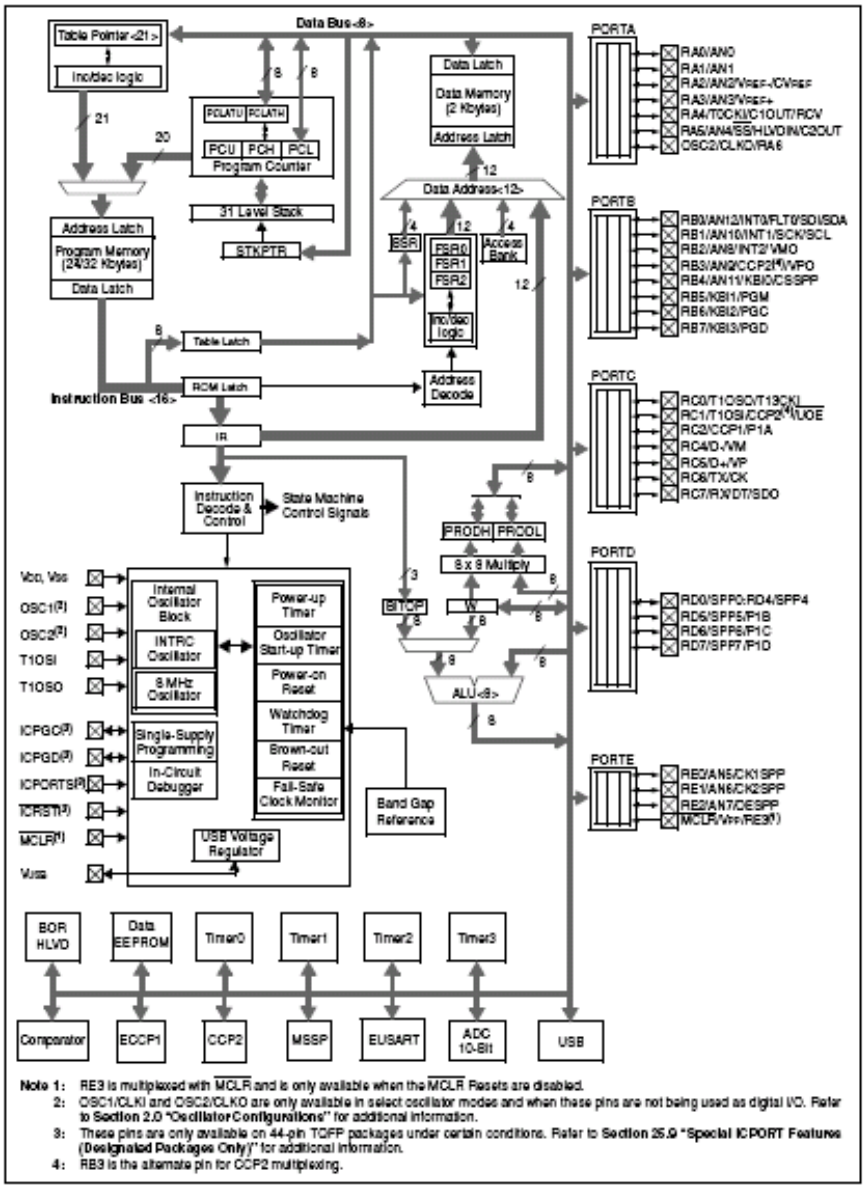



Figura 10.1: Diagrama en bloques del microcontrolador PIC18F4550

## 10.1. Organización de la memoria

Este dispositivo emplea la arquitectura Harvard, debido a ello las memorias para datos y programa usan diferentes buses y por tanto están en espacios de direccionamiento disjuntos.

La memoria de programa esta implementada mediante 32 Kbytes de memoria FLASH, lo que permite almacenar hasta 16.384 instrucciones de una palabra (16 bits). Este microcontrolador implementa un contador de programa (PC) de 21 bits, y mas allá del limite de la memoria implementada se lee como 0 (instrucción NOP). A continuación se muestra en la figura 10.2 el mapa de memoria de programa y stack.

Aquí se puede destacar un stack de 31 niveles que puede ser manejado por software y los vectores de reset e interrupciones de alta y baja prioridad.

Este microcontrolador incorpora pipeline por lo que en un ciclo de instrucción puede realizar el fetch y ejecución de la instrucción. Las instrucciones son de largo fijo y tienen el largo de una (2 bytes) o 2 palabras (4 bytes). De esta forma todas las instrucciones de 1 palabra se ejecutan en un ciclo de instrucción excepto los saltos, en que se incurre en una penalización y demoran

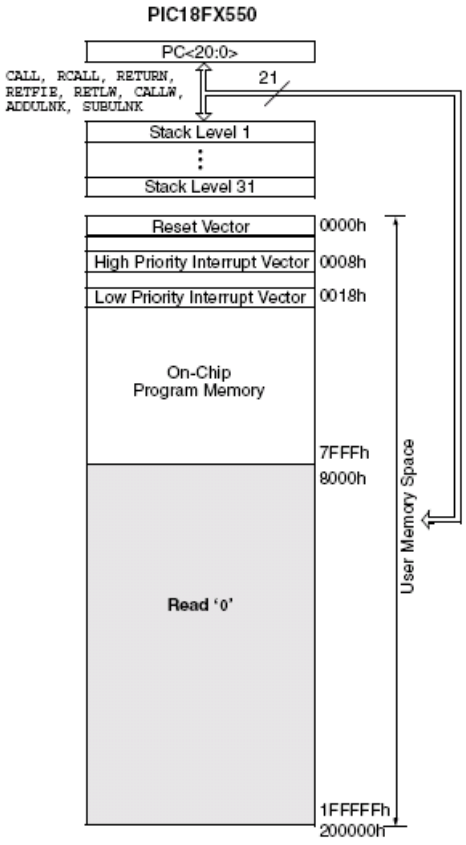



Figura 10.2: Mapa de memoria del programa y stack del PIC18F4550

2 ciclos de instrucción. Los saltos condicionales de instrucciones de 2 palabras demoran 3 ciclos de instrucción cuando la condición de salto es verdadera. Un ciclo de instrucción consiste en 4 períodos del oscilador, entonces si la frecuencia del oscilador es de 4 Mhz, la duración normal de ejecución de una instrucción es de un 1 us.

La memoria de datos esta implementada como RAM estática direccionada mediante 12 bits, lo que da un total de 4096 bytes de memoria que esta dividida en 16 bancos de 256 bytes cada uno. La memoria se divide en registros de función especial (SFRs) y registros de propósito general (GPRs). Los SFRs son utilizados para control y status del microcontrolador y funciones periféricas, mientras que los GPRs son utilizados para el almacenamiento de datos o variables de la aplicación del usuario.

Todo el espacio de memoria puede ser accedido en los modos: Inherente, Literal, Directo, Indirecto y además si se utiliza el conjunto de instrucciones extendido, se puede utilizar el modo Indexado de Offset Literal.

Para asegurar que los registros comúnmente usados (SFRs y algunos GPRs) sean accedidos en un solo ciclo, este dispositivo implementa un Banco de Acceso. Es un espacio de memoria de 256 bytes que provee dicho acceso. En la figura 10.3 se puede ver un esquema del mecanismo de acceso a la memoria.

Aquí el SFR BSR sirve para seleccionar el banco de memoria que se quiere utilizar.

Los bancos 4 al 7 están mapeados a una memoria especial la cual comparten el núcleo del microcontrolador y el USB serial interface engine (SIE)

De forma adicional este microcontrolador posee una memoria EEPROM de 256 bytes que es accedida en forma similar a un periférico, ya que es direccionada y accedida mediante un conjunto de de SFRs.

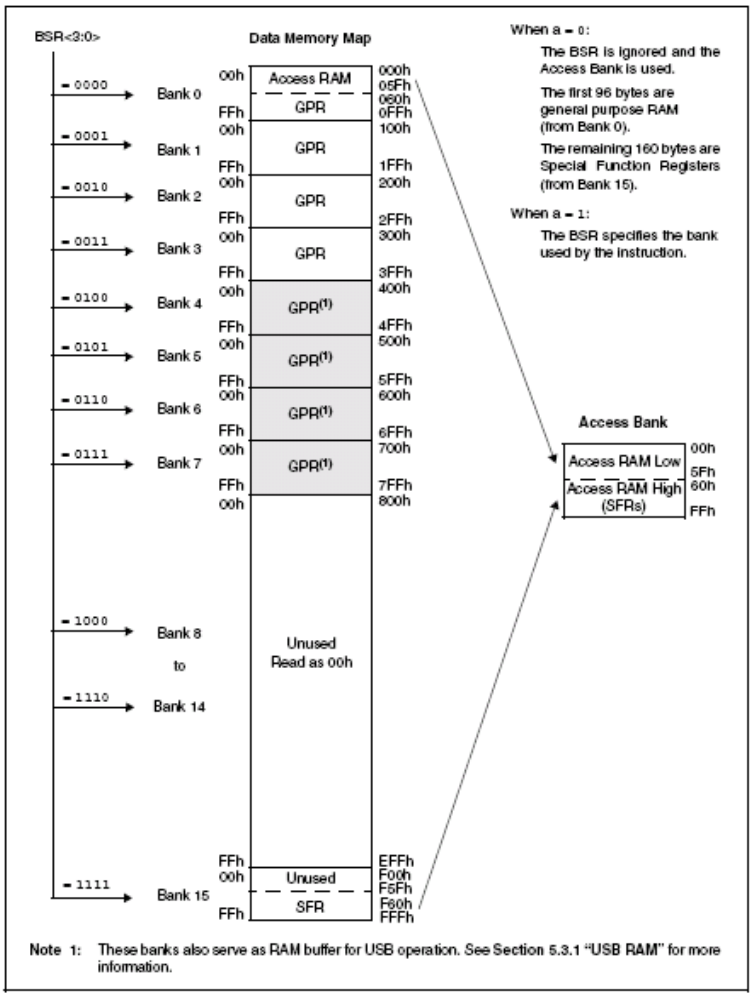



Figura 10.3: Acceso a la memoria

## 10.2. Periféricos

### 10.2.1. Puertos de entrada/salida

El PIC18F4550 cuenta con 5 puertos de Entrada/Salida, pero en general, sus pines están multiplexados con algunos periféricos específicos (ej. EUSART, ADC), de modo que si se habilita un periférico particular, esos pines dejan de funcionar como Entrada/Salida digital.

En la figura 10.4 se muestra un diagrama genérico de un puerto.

Cada puerto cuenta con 3 registros mapeados en memoria para su uso. El registro TRIS (registro de dirección de datos) sirve para setear cada uno de los pines como de entrada o salida, el registro PORT sirve para leer y escribir el puerto y el registro LATA mantiene un latched output value del PORT y sirve para funciones de lectura-modificación-escritura.

Los puertos son en general bidireccionales, pudiéndose setear cada uno de los pines pertenecientes al puerto como de entrada o salida. Cada pin seteado como salida puede brindar hasta 25 miliamperes, lo que permite alimentar en forma directa algunos dispositivos como por ejemplo LEDs o displays.

Algunos de los puertos tienen particularidades por lo que veremos a continuación los atributos y capacidades especiales de cada uno de ellos.

- PORTA: 8 bits de ancho, bidireccional. Todos sus pines tienen niveles de entrada TTL y drivers de salidas CMOS completos.

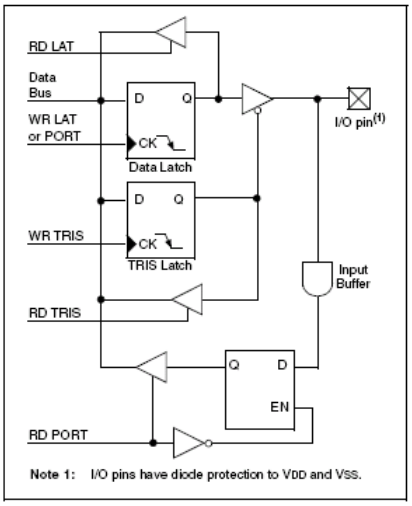



Figura 10.4: Puerto de entrada/salida digital genérico.

Este puerto esta multiplexado con los siguientes módulos: Timer0, módulo ADC, transceiver externo de USB y comparador analógico.
En particular 5 pines son multiplexados con canales y referencias de voltaje del modulo ADC, 1 con el modulo Timer0 si es utilizado con clock externo o la salida del un comparador de voltaje, y finalmente un pin que puede ser utilizado en el circuito de oscilador principal.

- PORTB: 8 bits de ancho, bidireccional. Cada uno de los pines posee una resistencia interna de pull-up que puede ser activada con un bit de control.
  Tiene 4 de sus pines que si son utilizados como entradas pueden generar interrupciones al cambiar (RB7:RB4). Dos de sus pines son multiplexados con un transceiver externo de USB y una es multiplexada con el modulo Streaming Parallel Port (SPP)

- PORTC: 7 bits de ancho, bidireccional (excepto RC4 y RC5 solo de entrada). Esta multiplexado con módulos de comunicación serial como por ejemplo EUSART, MSSP y USB, además de timers. Todas las entradas utilizan Schmitt Triggers excepto RC4 y RC5.

- PORTD: 8 bits de ancho, bidireccional. Todas las entradas utilizan Schmitt Triggers y poseen una resistencia pull-up configurable con un bit de control. Estas automáticamente se desconectan si el puerto es utilizado como salida u otro periférico.
  Este puerto esta multiplexado con el módulo Enhanced CCP. De manera adicional este puerto puede ser configurado como Streamming Parallel Port (SPP) de 8 bits, si se utiliza en este modo las entradas pasan a ser TTL.

- PORTE: 4 bits de ancho, bidireccional (excepto RE3 que sólo es entrada) con entradas Schmitt Trigger. Esta multiplexada con módulos ADC, SPP y Master Reset del dispositivo.

### 10.2.2. Timers/Counters:

Este dispositivo cuenta con 4 timers, a continuación se detallan las características de cada uno de ellos:

- Timer0:

  Seleccionable por software modos de 8 y 16 bits.

  Registros de lectura y escritura.

  Prescaler dedicado de 8 bits programable por software.

  Fuente de reloj interna o externa.

  Selección de flanco para clock externo.

  Interrupción programable por overflow.

- Timer1 y Timer3:

  Seleccionable como timer o contador por software.

  Registros de lectura y escritura.

  Fuente de reloj seleccionable con reloj del dispositivo o opciones de oscilador interno de Timer1: esto permite seleccionar un oscilador de 32khz para alimentar el CPU y disminuir el consumo de energía o realizar un RTC.

  Interrupción programable por overflow.

- Timer2:

  Timer de 8 bits y registro de periodo (TMR y PR2 respectivamente)

  Registros de lectura y escritura.

  Preescaler (1:1, 1:4, 1:16) y postscaler (1:1 a 1:16) programables por software.

  Interrupción en igualdad de TMR2 a PR2.

  Uso opcional para como shift clock para el modulo MSSP.

  Módulos Capture/compare/PWM (ccp):
  Este microcontrolador cuenta con 2 modulos ccp, uno de los cuales tiene una característica "enhanced" para el manejo de PWM. Cada uno de estos módulos tiene 1 registro de control (1 byte) y un registro de datos (2 bytes), ambos son de lectura y escritura. Estos módulos utilizan timers según el modo de funcionamiento mostrado en la tabla 10.5.

| CCP1 Mode | CCP2 Mode | Interaction |
|---|---|---|
| Capture | Capture | Each module can use TMR1 or TMR3 as the time base. The time base can be different for each CCP. |
| Capture | Compare | CCP2 can be configured for the Special Event Trigger to reset TMR1 or TMR3 (depending upon which time base is used). Automatic A/D conversions on trigger event can also be done. Operation of CCP1 could be affected if it is using the same timer as a time base. |
| Compare | Capture | CCP1 be configured for the Special Event Trigger to reset TMR1 or TMR3 (depending upon which time base is used). Operation of CCP2 could be affected if it is using the same timer as a time base. |
| Compare | Compare | Either module can be configured for the Special Event Trigger to reset the time base. Automatic A/D conversions on CCP2 trigger event can be done. Conflicts may occur if both modules are using the same time base. |
| Capture | PWM(1) | None |
| Compare | PWM(1) | None |
| PWM(1) | Capture | None |
| PWM(1) | Compare | None |
| PWM(1) | PWM | Both PWMs will have the same frequency and update rate (TMR2 interrupt). |

**Note 1:** Includes standard and Enhanced PWM operation.

Figura 10.5: Modos de funcionamiento de los timers/counters

Los 2 módulos pueden compartir el mismo timer si operan en el mismo modo al mismo tiempo.

- **Modo captura**: En este modo el registro de datos del módulo, captura el valor de 16 bits del Timer1 o Timer3 cuando ocurre un evento en el pin asociado. El evento puede ser: cada flanco de bajada, cada flanco de subida, cada 4 flancos de subida o cada 16 flancos de subida. De manera opcional se puede utilizar dicho evento como fuente de interrupción.

- **Modo comparación**: En este modo, el registro de datos de 16 bits es constantemente comparado con un timer, cuando son iguales el pin asociado al modulo puede ser: puesto en nivel alto, puesto en nivel alto, cambiado (alto a bajo o bajo a alto), dejar sin cambios. De manera opcional se puede utilizar dicho evento como fuente de interrupción y en caso de hacerlo el pin asociado no es afectado.

- **Modo PWM:**En el modo modulación por ancho de pulso (PWM) el pin asociado del modulo produce una salida de hasta 10 bits de resolución.

- **Modulo Enhanced PWM (ECCP)**: Este tiene la posibilidad de utilizar pines adicionales para utilizar la salida PWM. Esto incluye el uso de 2 a 4 canales de salida, con selección de polaridad, control de dead-band y apagado y recomienzo automático. Tiene 4 modos de funcionamiento:

  Salida simple (1 pin, similar al módulo CCP)

  Salida Medio-Puente (2 pines)

  Salida Puente completo (4 pines) Directo.

  Salida Puente completo (4 pines) Reverso.

### 10.2.3. Módulo Master Synchronous Serial Port (MSSP)

El modulo Master Synchronous Serial Port (MSSP) es una interfase serial útil para la comunicación con otros periféricos o microcontroladores. Estos periféricos pueden ser memorias seriales EEPROM, manejadores de display, conversores A/D, etc.

Este módulo puede operar en uno de estos dos modos:

- Interfase Periférica Serial (SPI)
- Inter-Circuito Integrado (I2C)
- Modo Maestro Full
- Modo esclavo (con llamada de dirección general)

La interfaz I2C soporta los siguientes modos de operación:

- Modo Maestro
- Modo Multi-Maestro
- Modo Esclavo
- Enhanced universal synchronous receiver transmitter (EUSART)

Un módulo USART puede ser configurado de manera asíncrona y full-duplex o sincrónica y half-duplex. El módulo EUSART además implementa otras funcionalidades como Detección de Baud-Rate Automática (ABD) y calibración, wake-up automático en recepción de Sync Break y transmisión de 12-bit break caracter. Estas características hacen posible su uso en sistemas Local Interconnect Network Bus (LIN bus).

El EUSART puede ser configurado en los siguientes modos:

- Asincrónico (full-duplex) con :

  Auto-wake-up en recepción de caracteres

  Calibración de baud automática

  12-bit Break Carácter transmisión

- Sincrónico Maestro (half-duplex) con paridad de reloj seleccionable.
- Sincrónico Esclavo (half-duplex) con paridad de reloj seleccionable.

### 10.2.4. Conversor analógico digital (ADC)

El microcontrolador cuenta con un conversor analógico digital de 10 bits de resolución y cuenta con 13 canales de entrada. Este módulo puede ser configurado para generar interrupciones y puede operar mientras el procesador se encuentra en sleep, de esta forma puede hacer un wake-up en el momento que la conversión es finalizada.

Para usar este módulo se deben seguir algunos pasos y además esperar un tiempo mínimo para la conversión. Este tiempo es de alrededor de 6.4 us, por lo que limita la conversión de una señal analógica a frecuencias menores de 75 khz.

### 10.2.5. Módulo Comparador

El módulo comparador esta formado por dos comparadores cuyas entradas pueden ser multiplexadas de variedad de formas. Así mismo también pueden ser multiplexadas sus salidas y asignadas a nivel de pines.

Este módulo cuenta con la posibilidad de interrumpir al procesador al detectarse algún cambio en cualquiera de los dos comparadores. Los comparadores pueden funcionar mientras el procesador esta en modo sleep y realizar un wake-up en caso de haberse seteado previamente.

### 10.2.6. Módulo de Referencia para comparar voltajes:

Este módulo es un 16-tap resistor ladder network que provee un referencia de voltaje seleccionable por software. Su principal función es la de proveer una referencia de voltaje para los comparadores analógicos, pero igualmente puede funcionar independientemente de ellos.

### 10.2.7. Módulo de detección de voltaje High/Low

Es un circuito programable que permite especificar un voltaje de referencia (utilizando el módulo de referencia) y una dirección de cambio con respecto a ella. De esta forma es posible detectar cuando un voltaje se supera o se hace inferior al voltaje de referencia, y en caso de haberse seteado, este puede producir una interrupción al CPU.

## 10.3. USB

El PIC18F4550 contiene un USB Serial Interface Engine (SIE) que proporciona comunicaciones rápidas entre un host USB y el micrcocontrolador. El SIE puede ser interfaseado directamente al USB (utilizando un transceiver interno) o a través de un transceiver externo. Además cuenta con un regulador de 3.3 volts incorporado para utilizar el transceiver interno en aplicaciones de 5 Volts.

Se mejoraron algunas características del hardware para aumentar su performance, como una memoria especial compartida entre el SIE y el microcontrolador y un puerto de streaming para permitir tranferencias ininterrumpidas de grandes volúmenes de datos a buffers externos.

La interfaz USB es compilante con la versión 2.0 y permite funcionar en alta (12 Mbps) y en baja (1.5 Mbps) velocidad. Se puede destacar que soporta los 4 tipos de transferencias descriptas en el estándar, pudiendose definir hasta 32 endpoints o hasta 16 caso de ser bidireccionales. Además cuenta con la posibilidad de utilizar ping-pong buffering. El ping pong buffering permite utlizar 2 buffers por endpoint, uno para las transferencias pares y otro para las tranferencias impares, aumentando así el throughtput de la comunicación.

El tranceiver interno cuenta con la posibilidad de utilizar resistencias pull-up programables por software en los pines D+ y D-, de esta forma se puede conectar y desconectar por software al bus. Esta caracteristica lo hace interesante para permitir la re-enumeración del dispositivo por software y da la posibilidad de programar un bootloader que cargue un programa a ejecutar a través de USB, y luego de cargado se re-enumere como el dispositivo propiamente a utilizar.

El módulo USB puede generar múltiples condiciones de interrupción. Para manejar todas las fuentes de interrupción, el módulo es provisto de su propia lógica, de estructura similar al del resto de las interrupciones del microcontrolador.

El dispositivo USB puede obtener la energía de tres formas: sólo del bus, self-powered o dual power con self-power dominante.

Una ruta alternativa para el manejo de datos es utilizar el puerto de Streaming (SPP). De esta forma permite posibilidades de diseño en donde el microcontrolador actúa como un gestor de datos, permitiendo al SPP pasar grandes bloques de datos sin que el microcontrolador este procesándolo. Por ejemplo, en un sistema de adquisición de datos, donde los datos son eviados mediante streaming de un buffer FIFO externo a la PC a través de USB.

Este puerto actúa como puerto maestro completo de 8 bits con las siguientes salidas de control:

- Dos salidas de reloj (CK1SPP y CK2SPP)
- Output Enable (OESPP)

- Chip Select (CSSPP)

De esta forma si todas las salidas son utilizadas, permiten las siguientes opciones:

- CLK1 como reloj de la información de dirección de endpoint mientras que CLK2 de los datos
- CLK1 como reloj de operaciones de escritura mientras que CLK2 de lectura.
- CLK1 como reloj de las direcciones impares mientras que CLK2 de direcciones pares.

En caso de comunicarse a través del puerto de streaming con dispositivos lentos se cuenta con la posibilidad de agregar estados de espera, desde 0 hasta 30 incrementando de a 2.

## 10.4. Interrupciones

El dispositivo tiene múltiples fuentes de interrupción y la posibilidad de utilizar la dos niveles de prioridad en las interrupciones, o utilizar el modo de compatibilidad con la línea media deshabilitando las prioridades.

Todas las interrupciones tienen tres bits de control para controlar la operación:

- Bit de bandera para indicar que un evento de interrupción ha ocurrido
- Bit de encendido para permitir saltar a la dirección del vector de interrupciones cuando el bit de bandera este encendido
- Bit de prioridad para seleccionar alta o baja prioridad.

En caso de utilizar niveles de prioridad, una interrupción de alta prioridad puede interrumpir a una de baja prioridad, por lo que algunos cuidados especiales deben seguirse en el momento de la programación.

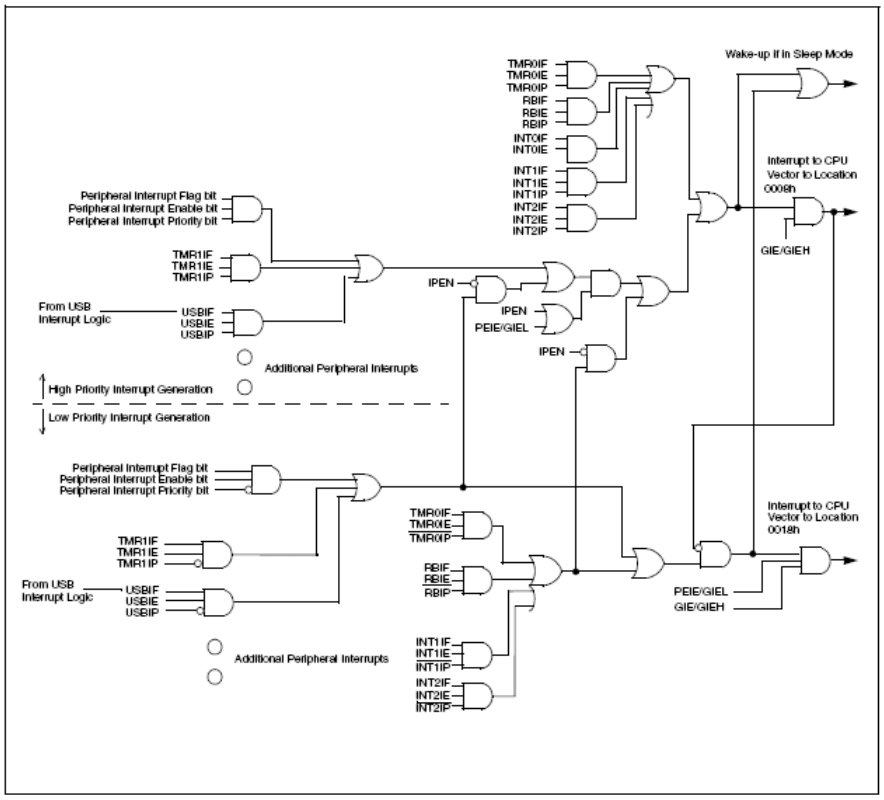



Figura 10.6: Lógica del manejo de las interrupciones en el PIC18F4550

Como se ve en la figura 10.6, la lógica de manejo de interrupciones agrega una salida que permite realizar un wake-up del microcontrolador en caso de haberse seteado el evento correspondiente, independientemente si se pasa a ejecutar la rutina de interrupción.

## 10.5. Modos de ahorro de energía:

Este microcontrolador cuenta con 7 modos para la gestión de la energía. Estos caen dentro de las siguientes categorías:

- Modos Run
- Modos Idle
- Modo Sleep

Estas categorías definen que porciones del dispositivo están recibiendo señal de reloj y a que velocidad. En modo sleep se desactivan las señales de reloj a todos los módulos. En la tabla 10.7 se muestran los bits que afectan los modos de ahorro de energía junto con las fuentes de reloj asociadas a cada modo.

| Mode | OSCCON Bits | | Module Clocking | | Available Clock and Oscillator Source |
|---|---|---|---|---|---|
| | IDLEN(1) | SCS1:SCS0 | CPU | Peripherals | |
| Sleep | 0 | N/A | Off | Off | None – all clocks are disabled |
| PRI_RUN | N/A | 00 | Clocked | Clocked | Primary – all oscillator modes. This is the normal full power execution mode. |
| SEC_RUN | N/A | 01 | Clocked | Clocked | Secondary – Timer1 oscillator |
| RC_RUN | N/A | 1x | Clocked | Clocked | Internal oscillator block(2) |
| PRI_IDLE | 1 | 00 | Off | Clocked | Primary – all oscillator modes |
| SEC_IDLE | 1 | 01 | Off | Clocked | Secondary – Timer1 oscillator |
| RC_IDLE | 1 | 1x | Off | Clocked | Internal oscillator block(2) |

Figura 10.7: Modos de ahorro de energía

## 10.6. Carateristicas especiales del microcontrolador:

### 10.6.1. Selección del oscilador

El microcontrolador cuenta con varias fuentes para generar la señal de reloj para el CPU y para el módulo de USB. Esto se realiza mediante los bits de configuración en el momento de la programación y por software se puede entrar en un modo de ahorro de energía. Allí se selecciona el tipo de oscilador y el uso de prescalers y postcalers. Esto permite alcanzar velocidades de reloj mayores que el cristal o resonador utilizado y a la vez permitir señales de reloj diferentes para cada módulo.

Esencialmente hay tres fuentes de reloj: osciladores primarios, secundarios y bloque oscilador interno. Dentro de los osciladores primarios encontramos los modos de cristal externo o resonador (alta y baja velocidad), modos de reloj externo y bloque de oscilador interno. En la figura 10.8 se muestra un esquema donde se muestran las opciones al momento de seleccionar la fuente de reloj del microcontrolador.

Los osciladores secundarios son aquellas fuentes que no están conectadas a los pines OSC1 y OSC2. Estas fuentes pueden continuar la operación cuando se setea al microcontrolador en modo de ahorro de energía. Dentro de esta categoría se encuentra el Timer1 cuando se setea como oscilador y normalmente funciona cristal de 32.768 khz para funcionar como reloj de tiempo real (RTC).

Además de ser una funcionar como fuente de reloj primaria, el bloque oscilador interno está disponible como fuente de reloj en modo de ahorro de energía.

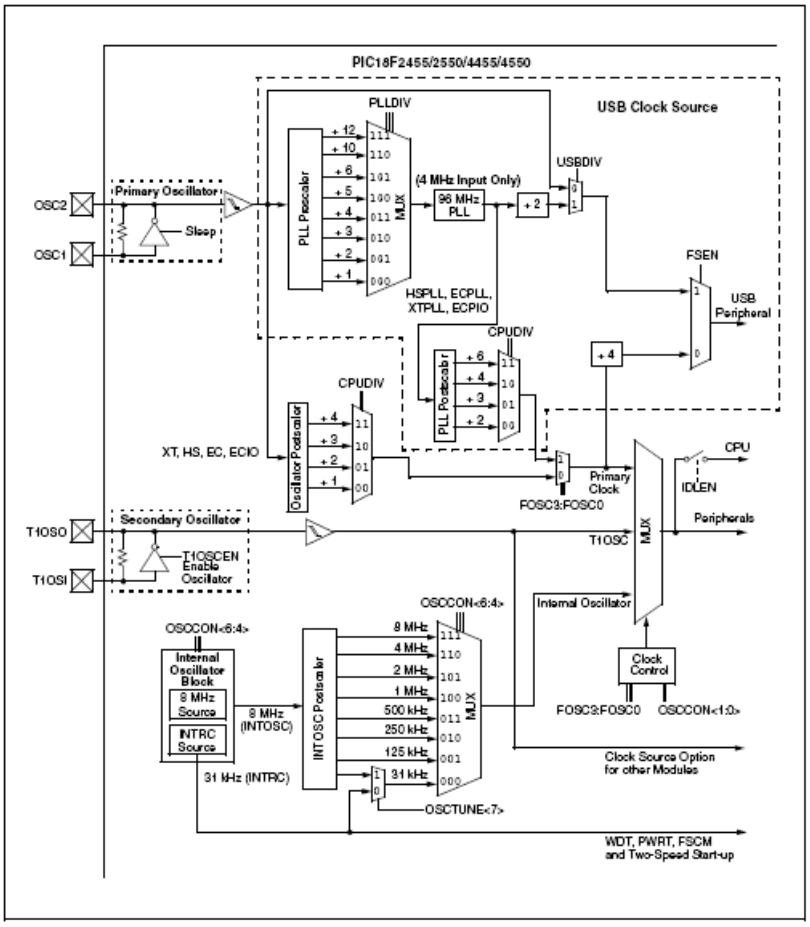



Figura 10.8: Selección de la fuente de reloj del microcontrolador

### 10.6.2. Multiplicador de 8x8 por hardware

Como parte de la ALU se cuenta con un multiplicador por hardware y permite que la multiplicación se realice en un solo ciclo. Esto aumenta el throughtput computacional además de reducir los algoritmos de dicha operación.

### 10.6.3. Memoria de Programa auto-programable por software.

Una capacidad adicional que tiene este microcontrolador es la posibilidad de leer, borrar y escribir la memoria de programa mediante software, esto brinda la capacidad poder hacer un pequeño programa bootloader que permita cargar o actualizar el software a través de el modulo serial o USB, así como también lectura y escritura de tablas de datos.

### 10.6.4. Programación In-circuit (ICSP) y depuracion In-circuit. (ICD)

Este microcontrolador puede ser programado en forma serial en el mismo circuito de la aplicación. Solo se precisan dos líneas para reloj y datos, y tres más para alimentación y voltaje de programación. Esto facilita la tarea de programación sin tener que retirar el microcontrolador del circuito.

En forma análoga se puede activar la funcionalidad de depurador cuando el microcontrolador se encuentra en el circuito y permite el sencillo depurado del programa cuando es utilizado con el MPLAB IDE.

## 10.7. Conjunto de instrucciones:

El conjunto de instrucciones esta compuesto por 75 instrucciones de núcleo PIC18, así como también por un conjunto de 8 nuevas instrucciones para la optimización de código recursivo o que utiliza el stack de software.

La mayoría de las instrucciones ocupan sólo una palabra (16 bits) y cuatro instrucciones requieren 2 palabras en memoria de programa.

- El conjunto de instrucciones es altamente ortogonal y se puede agrupar en 4 categorías básicas:
  - Operaciones orientadas a bytes.
  - Operaciones orientadas a bits.s
  - Operaciones con literales.
  - Operaciones de control.

La mayoría de las operaciones orientadas a bytes tienen 3 operandos:

El file register *f*, el destino del resultado *d* y me acceso a memoria *a*.

El registro *f* especifica que registro va a ser utilizado por la instrucción, el designador de destino *d* especifica donde será almacenado el resultado de la operación (en f o en el registro de trabajo) y en *a* el tipo de acceso a la memoria de datos.

Todas las operaciones orientadas a bits tienen 3 operandos:

El file register *f*, el bit en el file register *b* y el acceso a memoria *a*

El designador de campo de bit *b* selecciona el bit que será afectado por la operación dentro del byte designado por el file register *f*.

Las operaciones con literales pueden usar alguno de los siguientes operandos.

- Un valor literal a ser cargado en un file register *k*
- Un registro FSR para cargar el valor literal *f*
- No requieren operando

Las instrucciones de control pueden usar algunos de los siguientes operandos:

- Dirección de memoria de programa
- El modo de las instrucciones de CALL o RETURN
- El modo de la lectura o escritura de tabla.
- No requieren operando.

El conjunto de instrucciones extendido por defecto se encuentra deshabilitado y para habilitarlo se utiliza un bit de configuración al momento de programarlo.

Estas instrucciones pueden clasificarse como operaciones literales que manipulan los FSR o los usan para acceso de memoria indexado. Estas instrucciones están específicamente implementadas para optimizar código de programa re-entrante escrito en lenguajes de alto nivel, particularmente en C.

# Capítulo 11

# ATMEL AT90USB647

Resumen de las principales características del microcontrolador AT90USB647 de ATMEL:

- Arquitectura: RISC
- Memoria:
  - 128 KBytes de memoria Flash.
  - 4 KBytes de memoria EEPROM
  - 8 KBytes de SRAM
  - Hasta 64 KBytes de memoria externa opcional.
- Perifericos:
  - Hasta 48 pines de E/S digital
  - 4 Módulos de timer (2 de 8 bits y 2 de 16 bits)
  - 6 Canales de PWM con resolución programable de 2 a 16 bits y 2 de 8 bits
  - Salida de Modulador de comparación
  - Conversor A/D de 8 canales y 10 bits de resolución.
  - USART Serial Programable.
  - Interfaz SPI (Master/Slave)
  - Interfaz 2-Wire Serial orientada a bytes
- USB:
  - Interfaz USB V2.0 que soporta baja (1.5 Mbit/s) y alta velocidad (12 Mbit/s)
  - Módulo suplementario On-The-Go USB 2.0 Rev 1.0
  - 6 Endpoints programables que soportan transferencias Bulk, Interrupt e Isochronous
  - 832 bytes de DPRAM para allocacion de memoria de endpoint
- Caracteristicas de microcontrolador especiales:
  - Interfaz JTAG (IEEE std. 1149.1 compilante) que permite programar la memoria Flash, EEPROM, fusibles y bits de bloqueo.
  - Memoria Flash Auto-programable por software
  - Sección de código de Boot opcional con bits de bloqueo independientes
  - Multiplicador en hardware.
  - Ahorro de energia: 5 modos de Sleep.

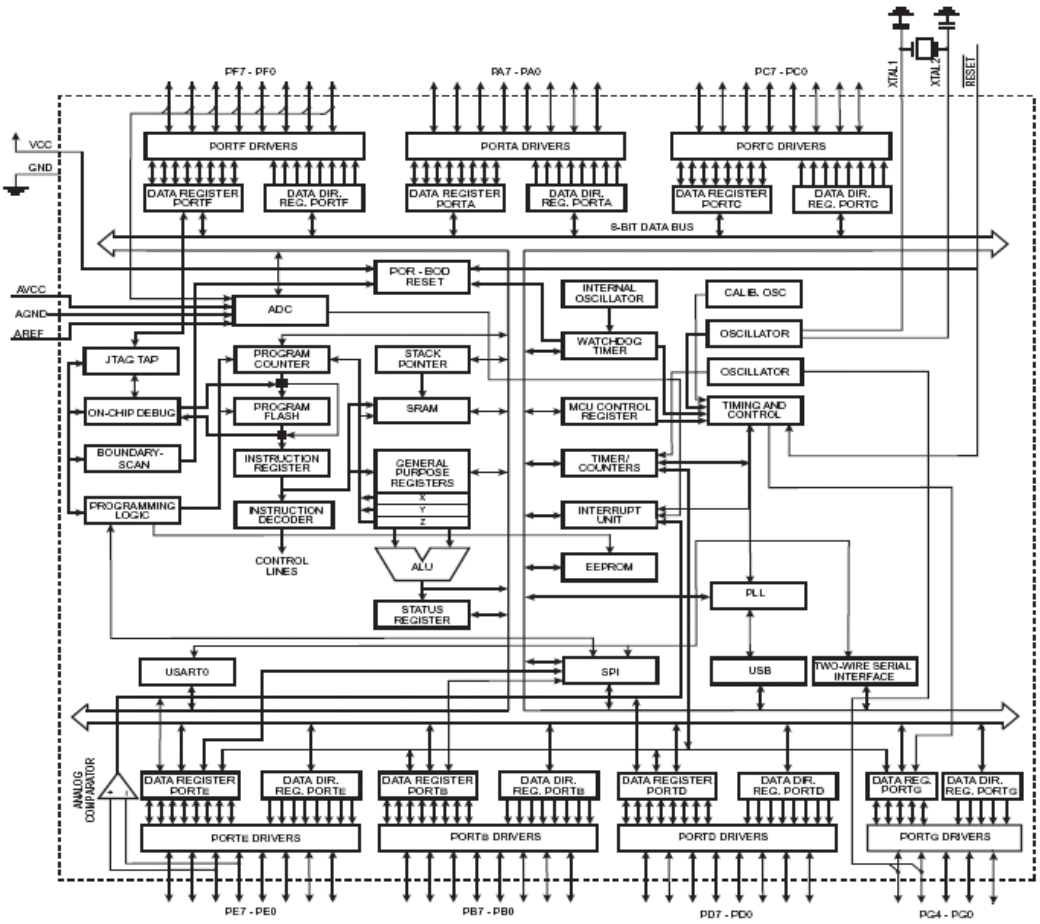



Figura 11.1: Diagrama de bloques

## 11.1. Núcleo y organizacion de la memoria

El AT90USB128 es un microcontrolador CMOS de 8 bits basado en la arquitectura enhanced RISC AVR. Cuenta con 135 instrucciones, la mayoria de ellas que ejecuta en un ciclo de reloj. A en la figura 11.1 vemos un diagrama de bloques del microcontrolador.

El núcleo AVR combina un rico set de instrucciones con 32 registros de trabajo de uso general. Los 32 registros están directamente conectados a la ALU, permitiendo que 2 registros independientes sean accedidos en una única instrucción en un ciclo de reloj. Este microcontrolador incorpora además un multiplicador por hardware que soporta multiplicación con y sin signo y formato fraccional.

El microcontrolador cuenta con una memoria flash que puede ser reprogramada en el circuito a traves de una interfaz SPI serial, por un programador de memoria convencional o por el programa de booteo que corre en el núcleo AVR. El software en la seccion de booteo puede continuar corriendo mientras la sección o área de aplicacion es actualizada, proveyendo una operacion Leo-Mientras-Escribo.

La función principal del núcleo del CPU es asegurar la ejecución correcta del programa. El CPU debe entonces ser capaz de acceder a las memorias, hacer cálculos, controlar periféricos y manejar interrupciones. En la figura 11.2 se muestra un diagrama de bloques de microcontrolador que se centra en la interconexión de CPU mediante el bus de datos de 8 bits con los demás componentes dentro del micrcontrolador.

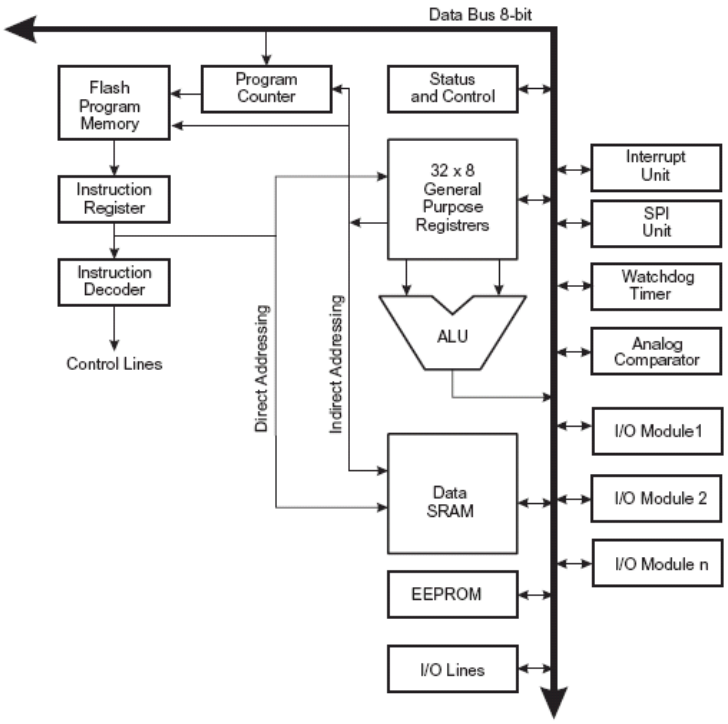



Figura 11.2: Diagrama de bloques de la arquitectura AVR

Para maximizar la performance y paralelismo, el AVR usa una arquitectura Harvard con buses separados para memorias de programa y datos. Instrucciones en la memoria de programa son ejecutados con un pipeline de un solo nivel. Cuando una instrucción está siendo ejecutada, la siguiente instruccion es pre-fetched de la memoria de programa.

Seis de los 32 registros pueden ser usados como tres punteros para acceso indirecto de 16 bits para direccionamiento de datos. Uno de esos punteros de dirección también puede ser usado como puntero para tablas de look-up en el área de memoria de programa. Estos registros son llamados X, Y y Z.

La mayoría de las intrucciones tienen un formato de una única palabra de 16 bits. Cada una de las direcciones de programa contienen una instrucción de 16 o 32 bits.

El espacio de memoria de programa Flash está dividido en 2 secciones, la sección de programa de booteo y la sección de programa de aplicación. Ambas secciones tienen bits de lock dedicados para protección de escritura y de lectura/escritura.

Este microcontrolador cuenta con un módulo de interrupciones flexible y tiene sus registros de control en el espacio de E/S con un bit de habilitación de interrupciones globales en el registro de status. Todas las interrupciones tienen un vector de interrupción separado en la tabla de vectores. Las interrupciones tienen prioridades en relación con la posición de dicho vector, mientras más baja sea la direccion del vector más alta es la prioridad.

El espacio de memoria de E/S contiene 64 direcciones para funciones de perifericos del CPU como registros de control, SPI, y otras funciones de E/S. Esta memoria puede ser accedida directamente, o como espacio de datos siguiendo las direcciones de Registro, 0x20-0x5f. Además el AT90USB128 tiene un espacio de E/S extendido desde 0x60-0x0ff en SRAM.

### 11.1.1. Reseteo y manejo de interrupciones

El AVR provee gran cantidad de diferentes fuentes de interrupciones. Todas las interrupciones tienen asignados bits de habilitación los cuales deben ser seteados junto con el bit de interrupciones globales para habilitar la interrupción.

Las direcciones más bajas de la memoria de programa son definidas por defecto como los de Reset y Vectores de interrupción. Mientras más baja es la dirección, más alta es la prioridad. De esta forma RESET tiene la prioridad más alta, y la siguiente es INT0.

Cuando una interrupcion ocurre el Bit de Interrupciones Globales (I) es borrado y todas las interrupciones son deshabilitadas. El software de usuario puede setear el bit I para permitir interrupciones anidadas. De esta forma todas las interrupciones pueden interrumpir la interrupción actual. El bit I es seteado automaticamente cuando la instruccion RETI (retorno de interrupción) es ejecutada.

Basicamente hay dos tipos de interrupciones. El primer tipo es gatillado por un evento que setea la bandera de interrupción. Para estas interrupciones el Contador de Programa (PC) es seteado en el vector de interrupción correspondiente de forma de ejecutar el manejador de interrupciones correspondiente, y el hardware borra la bandera de interrupción. Si una interrupción ocurre cuando su bit de habilitación no esta seteado, la bandera de interrupción permanecera seteada y será recordada cuando la interrupción sea habilitada. De forma similar si una o varias condiciones de interrupción ocurren mientras el bit I esté deshabilitado, las correspondiente/s banderas de interrupción seran seteadas y recordadas cuando el bit I sea seteado, y seran ejecutadas en orden de prioridad. En la figura 11.3 se muestran los vectores de interrupción de este microcontolador.

El segundo tipo de interrupciones sera gatillada mientras la condición de interrupción este presente. Estas interrupciones no necesariamante tienen banderas de interrupción. Si la condición de interrupción desaparece antes que la interrupción sea habilitada, la interrupción no será gatillada.

Cuando el AVR sale de una interrupción, siempre retorna al programa principal y ejecuta una instrucción más antes de que algua otra interrupción pendiente sea servida.

El registro de status no es automaticamente guardado cuando se entra a una rutina de interrupción ni restaurado cuando se vuelve, esto debe ser manejado por software.

### 11.1.2. Memoria del AT90USB64/128

La arquitectura AVR tiene 2 espacios de memoria principales, la memoria de datos y la memoria de programa. A esto se le agrega la memoria EEPROM para salvar datos. Los 3 espacios de memoria son lineales y regulares. En la figura 11.4 se muestra el mapa de memoria.

#### 11.1.2.1. Memoria de programa Flash reprogramable en el sistema

El AT90USB128 contiene 128 Kbytes de memoria flash reprogramable en el sistema para el almacenamiento de programa. Dado que todas las instrucciones AVR son de 16 o 32 bits de ancho, la memoria esta organizada como 64 K x 16. Para seguridad del software, esta memoria esta dividida en dos secciones, la de booteo y la del programa de aplicación.

#### 11.1.2.2. Memoria de datos SRAM

En la figura 11.5 se muestra como esta organizada la memoria de datos.

Los primeros 32 bytes conforman el file de registros, los siguientes 64 son las pociciones de la memoria estándar de E/S, luego 416 bytes de memoria de E/S extendida y luego los siguientes 8192 bytes de memoria de datos internos SRAM.

Puede utilizarse opcionalmente una SRAM externa. Esa SRAM ocupará el área del resto de las direcciones en el espacio de 64 Kbytes. Esta área comienza a partir de la dirección que sigue a la memoria SRAM interna. El acceso a la memoria SRAM externa agrega un ciclo de reloj más por byte comparado con el de la SRAM interna.

#### 11.1.2.3. Memoria de datos EEPROM

El AT90USB128 contiene 4 KBytes de capacidad. Esta organizada como un espacio de direcciones separada en donde los bytes puede ser leidos o escritos de a uno. La interacción entre el CPU y la EEPROM se realiza de manera indirecta mediante registros ubicados en las posiciones de memoria estándar de E/S.

| Vector No. | Program Address(2) | Source | Interrupt Definition |
|---|---|---|---|
| 1 | $0000(1) | RESET | External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset |
| 2 | $0002 | INT0 | External Interrupt Request 0 |
| 3 | $0004 | INT1 | External Interrupt Request 1 |
| 4 | $0006 | INT2 | External Interrupt Request 2 |
| 5 | $0008 | INT3 | External Interrupt Request 3 |
| 6 | $000A | INT4 | External Interrupt Request 4 |
| 7 | $000C | INT5 | External Interrupt Request 5 |
| 8 | $000E | INT6 | External Interrupt Request 6 |
| 9 | $0010 | INT7 | External Interrupt Request 7 |
| 10 | $0012 | PCINT0 | Pin Change Interrupt Request 0 |
| 11 | $0014 | USB General | USB General Interrupt request |
| 12 | $0016 | USB Endpoint/Pipe | USB ENdpoint/Pipe Interrupt request |
| 13 | $0018 | WDT | Watchdog Time-out Interrupt |
| 14 | $001A | TIMER2 COMPA | Timer/Counter2 Compare Match A |
| 15 | $001C | TIMER2 COMPB | Timer/Counter2 Compare Match B |
| 16 | $001E | TIMER2 OVF | Timer/Counter2 Overflow |
| 17 | $0020 | TIMER1 CAPT | Timer/Counter1 Capture Event |
| 18 | $0022 | TIMER1 COMPA | Timer/Counter1 Compare Match A |
| 19 | $0024 | TIMER1 COMPB | Timer/Counter1 Compare Match B |
| 20 | $0026 | TIMER1 COMPC | Timer/Counter1 Compare Match C |
| 21 | $0028 | TIMER1 OVF | Timer/Counter1 Overflow |
| 22 | $002A | TIMER0 COMPA | Timer/Counter0 Compare Match A |
| 23 | $002C | TIMER0 COMPB | Timer/Counter0 Compare match B |
| 24 | $002E | TIMER0 OVF | Timer/Counter0 Overflow |
| 25 | $0030 | SPI, STC | SPI Serial Transfer Complete |
| 26 | $0032 | USART1 RX | USART1 Rx Complete |
| 27 | $0034 | USART1 UDRE | USART1 Data Register Empty |
| 28 | $0036 | USART1TX | USART1 Tx Complete |
| 29 | $0038 | ANALOG COMP | Analog Comparator |
| 30 | $003A | ADC | ADC Conversion Complete |
| 31 | $003C | EE READY | EEPROM Ready |
| 32 | $003E | TIMER3 CAPT | Timer/Counter3 Capture Event |
| 33 | $0040 | TIMER3 COMPA | Timer/Counter3 Compare Match A |
| 34 | $0042 | TIMER3 COMPB | Timer/Counter3 Compare Match B |
| 35 | $0044 | TIMER3 COMPC | Timer/Counter3 Compare Match C |
| 36 | $0046 | TIMER3 OVF | Timer/Counter3 Overflow |
| 37 | $0048 | TWI | 2-wire Serial Interface |
| 38 | $004A | SPM READY | Store Program Memory Ready |

Figura 11.3: Vector de interrupciones

| Memory | | Mnemonic | AT90USB64 | AT90USB128 |
|---|---|---|---|---|
| Flash | Size | Flash size | 64 K bytes | 128K bytes |
| | Start Address | - | 0x00000 | |
| | End Address | Flash end | 0x0FFFF(1) 0x7FFF(2) | 0x1FFFF(1) 0xFFFF(2) |
| 32 Registers | Size | - | 32 bytes | |
| | Start Address | - | 0x0000 | |
| | End Address | - | 0x001F | |
| I/O Registers | Size | - | 64 bytes | |
| | Start Address | - | 0x0020 | |
| | End Address | - | 0x005F | |
| Ext I/O Registers | Size | - | 160 bytes | |
| | Start Address | - | 0x0060 | |
| | End Address | - | 0x00FF | |
| Internal SRAM | Size | ISRAM size | 4 K bytes | 8 K bytes |
| | Start Address | ISRAM start | 0x0100 | |
| | End Address | ISRAM end | 0x10FF | 0x20FF |
| External Memory | Size | XMem size | 0-64 K bytes | |
| | Start Address | XMem start | 0x1100 | 0x2100 |
| | End Address | XMem end | 0xFFFF | |
| EEPROM | Size | E2 size | 2 K bytes | 4K bytes |
| | Start Address | - | 0x0000 | |
| | End Address | E2 end | 0x07FF | 0x0FFF |

Notes: 1. Byte address.
2. Word (16-bit) address.

Figura 11.4: Mapa de memoria

#### 11.1.2.4. Memoria de E/S

Todos los periféricos de entrada y salida en el AT90USB128 se ubican en el espacio de E/S. Todas las posiciones pueden ser accedidas mediante instrucciones LD/LDS/LDD y ST/STS/STD, transfiriendo datos entre los 32 registros de trabajo y el espacio de E/S. Los registros dentro de las direcciones 0x00 - 0x1F son accesibles directamente de a bit usando instrucciones SBI y CBI. Cuando se usan comandos de E/S específicos IN y OUT, las direcciones 0x00 - 0x3F deben ser usadas. Como el AT90USB128 es un microcontrolador complejo con más unidades de periféricos como para utilizarse con las 64 posiciones reservadas para los comandos IN y OUT, deben usarse otras instrucciones para utilizar el espacio de E/S extendido desde la direccion 0x60 a 0x1FF en SRAM.

El AT90USB128 contiene 3 registros de E/S de proposito general (GPIORx). Estos registros pueden ser usados para guardar cualquier informacion y son particularmente útiles para guardar variables globales y bits de status.

#### 11.1.2.5. Interfase de memoria externa.

Debido a sus atributos, la interfaz de memoria externa es útil para operar como interfase con dispositivos de memoria como SRAM externa y Flash, asi como con periféricos como displays LCD, y conversores A/D y D/A.

Se pueden destacar las siguientes caracteristicas:

- Cuatro diferentes seteos de wait-state (incluyendo sin wait-state)
- Seteo de wait-state independientes para diferentes sectores de memoria externa (tamaño de sector configurable)
- El número de bits dedicado al byte alto es seleccionable.
- Bus keepers en las lineas de datos para minimizar el consumo de energía (opcional)

Cuando la memoria externa esta habilitada (XMEM), las direcciones fuera del espacio de la memoria interna SRAM se puede direccionar utilizando los pines dedicados a la memoria externa.

Esta interfaz consiste en:

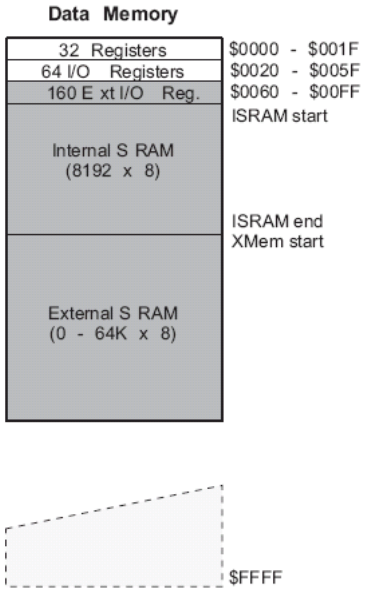



Figura 11.5: Mapa de memoria de Datos

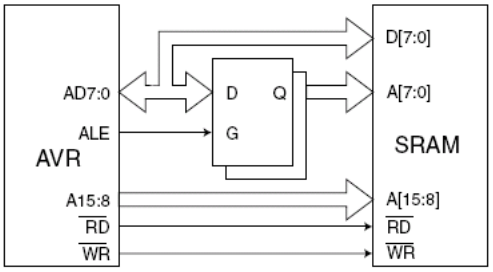



Figura 11.6: Interfaz de memoria externa.

- AD7:0: Byte bajo de direccionamiento y bus de datos multiplexados.
- A15:8: Byte alto de direccionamiento (numero de bits configurables)
- ALE: Adress latch enable
- /RD: Strobe de lectura.
- /WE: Strobe de escritura.

En la figura 11.6 se muestra como se conecta el microcontrolador con la memoria externa utilizando un latch octal (tipicamente un 74x573 o equivalente) que es transparente cuando G es alto.

Las resistencias de pull-up en el puerto AD7:0 pueden ser activadas si es escrito uno en el registro de puerto correspondiente. La interfaz XMEM tambien provee un bus-keeper en las lineas AD7:0 que puede ser configurado por software. Cuando está habilitado, el bus-keeper va a dejar el valor previo en el bus AD7:0 mientras esas lineas son puestas en 3er estado por la interfaz XMEM.

## 11.2. Reloj del sistema y Opciones de Reloj

En la figura 11.7 se muestra el sistema de reloj principal en el AVR y su distribución. No todos los relojes precisan estar activos en un tiempo dado. Para lograr reducir el consumo de

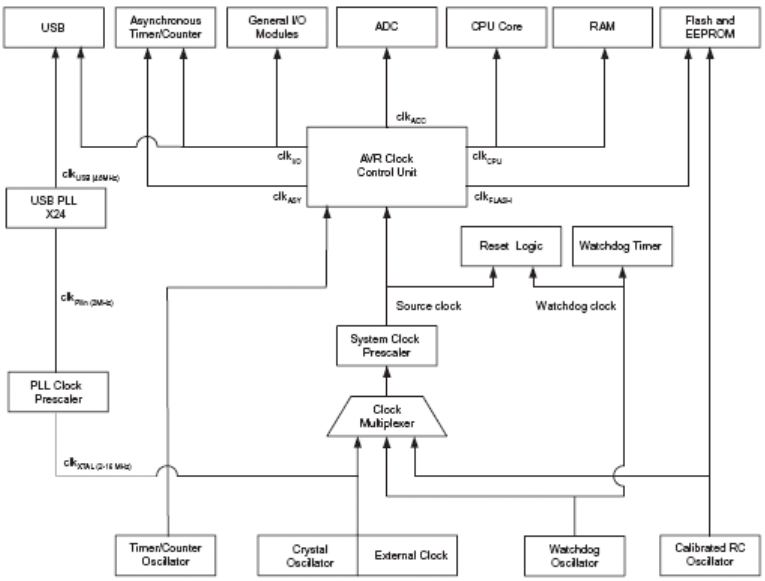



Figura 11.7: Distribucion de Reloj

energía, los relojes para los módulos que no están en uso pueden ser parados utilizando diferentes modos de sleep.

- **Reloj del CPU**: Este reloj es ruteado a partes del sistema relacionadas con la operacion del nécleo AVR. Ejemplos de esos módulos son los regitros de proposito general o el registro de status. Si se para (halt) este reloj, se inhibe al núcleo de poder hacer operaciones generales y cálculos.
- **Reloj de E/S**: Este reloj es usando en la mayoría de los módulos de E/S, como son los Timers/Counters, SPI y USART.
- **Reloj de Flash**: Controla la operación de la interfaz Flash. Usualmente esta activo simultaneamente con el reloj del CPU.
- **Reloj timer asincrónico**: Este reloj permite que el Timer/Counter asincrónico sea alimentado directamente de un reloj externo o un cristal de reloj de 32 Khz. El dominio dedicado del reloj permite usar este Timer/Counter como un contador de tiempo real a pesar de que el dispositivo esté en modo sleep.
- **Reloj de ADC**: El ADC es provisto con un dominio de reloj dedicado. Esto permite parar el CPU y los relojes de entrada y salida para reducir el ruido generado por la circuiteria digital. Esto permite una conversión ADC más exacta.
- **Reloj USB**: El USB está provisto con un dominio de reloj dedicado. Este reloj es generado con un PLL dentro del chip corriendo a 48 Mhz.

### 11.2.1. Fuentes de reloj

El dispositivo tiene las siguientes opciones de fuentes de reloj seleccionables por los bits o fusibles de configuración. La señal de reloj de la fuente seleccionada sirve como entrada del generador de clock AVR y ruteada a los módulos adecuados. A continuación se enumeran las fuentes de reloj que puede utilizar el microcontrolador:

- **Cristal de baja potencia**: El AT90USB128 puede funcionar con cristales de 1 hasta 16 Mhz.
- **Cristal de baja frecuencia**: Este dispositivo puede utilizar un cristal de reloj de 32.768 Khz como fuente de reloj.

| | Active Clock Domains | | | | | Oscillators | | Wake-up Sources | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sleep Mode | $clk_{CPU}$ | $clk_{FLASH}$ | $clk_{IO}$ | $clk_{ADC}$ | $clk_{ASY}$ | Main Clock Source Enabled | Timer Osc Enabled | INT7:0 and Pin Change | TWI Address Match | Timer2 | SPM/ EEPROM Ready | ADC | WDT Interrupt | Other I/O | USB Synchronous Interrupts | USB Asynchonous Interrupts(4) |
| Idle | | | X | X | X | X | X(2) | X | X | X | X | X | X | X | X | X |
| ADCNRM | | | | X | X | X | X(2) | X(3) | X | X(2) | X | X | X | | X | X |
| Power-down | | | | | | | | X(3) | X | | | | X | | | X |
| Power-save | | | | | X | | X(2) | X(3) | X | X | | | X | | | X |
| Standby(1) | | | | | | X | | X(3) | X | | | | X | | | X |
| Extended Standby | | | | | X(2) | X | X(2) | X(3) | X | X | | | X | | | X |

Cuadro 11.1: Características de los modos de ahorro de energía

- **Oscilador interno RC calibrado**: Por defecto provee un reloj de 8.0 MHz. Esta frecuencia es nominal a 3 volts y 25 grados celcius de temperatura. El dispositivo cuenta con un registro de calibración que permite que el oscilador pueda ser calibrado en el rango de frecuencias de 7.3 a 8.1 MHz con 1 % de tolerancia.
- **Oscilador interno de 128 KHz:** Este oscilador interno es un oscilador de baja potencia que tiene una salida nomina de 128 KHz alimentado con 3 volts y 25 grados celcius de temperatura.
- **Reloj Externo**: Este dispositivo puede utilizar una fuente externa.

## 11.3. Modos de ahorro de energia y modos sleep

Los modos de ahorro de energía permiten a las aplicaciónes apagar los módulos que no son utilizados en el microcontolador, ahorrando enegría. Esto es particularmente interesante en dispositivos alimientados por baterias. El AVR provee varios modos de sleep, permitiendo configurar los requierimientos a los requerimientos específicos de la aplicacion. Para entrar en uno de los 5 modos de sleep, se debe setear un valor en un registro y luego ejecutar la instrucción Sleep. En el cuadro 11.1 se muestran una tabla con las fuentes de reloj, osciladores y fuentes para salir del ahorro de energía en cada modo.

## 11.4. Periféricos

Todos los puertos AVR tienen una funcionalidad real Read-Modify-Write cuando son usados como puertos digitales de entrada/salida. Eso significa que la dirección de un pin de un puerto puede ser cambiada sin modificar de manera initencional cualquier otro pin con instrucciones SBI (Setear bit en registro de E/S) o CBI (Borrar bit en registro de E/S). Cada pin puede brindar hasta 20 mA de corriente, por lo que es capaz de manejar LEDs o displays directamente. La mayoría de los pines están multiplexados con funciones alternativas de periféricos en este dispositivo.

### 11.4.1. Puertos de entrada/salida digital

En la figura 11.8 se muestra un puerto de entrada y salida general.

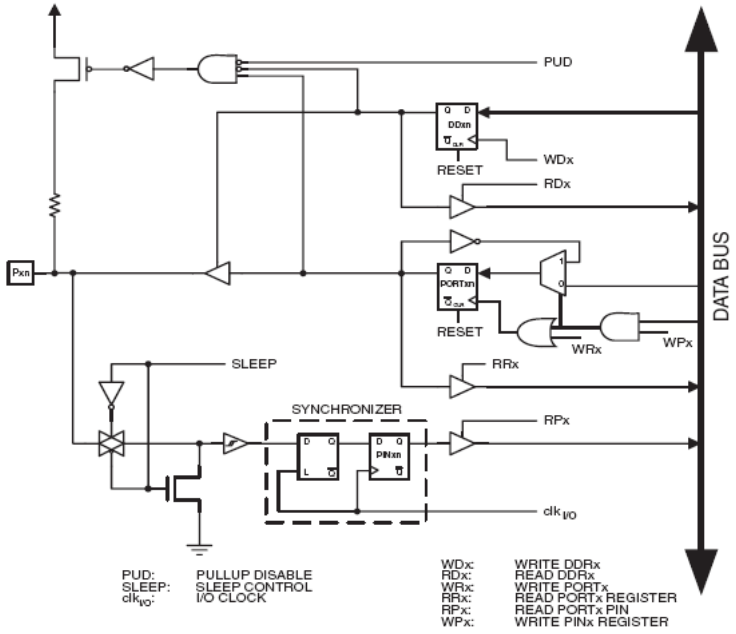



Figura 11.8: Puerto de entrada/salida digital general

## 11.4.2. Timers/counters

Este microcontrolador cuenta con varios timers y counters que se explican a continuación:

### 11.4.2.1. Dos timers/counters de 8 bits

Este microcontrolador posee dos contadores de propósito general de 8 bits y sus principales características son:

- Dos unidades de comparación de salida independientes.
- Registros de comparación de salida con doble buffer.
- Borrado de timer al matchear la comparación (recarga automática)
- Salida PWM con período variable y fase correcta.
- Generador de frecuencia
- Tres fuentes de interrupción independientes.

Vale destacar que uno de los 2 timers, además posee un prescaler de 10 bits e interfaz para un cristal de reloj de 32 Khz independiente de la entrada/salida de reloj.

### 11.4.2.2. Dos timers de 16 bits

Estos timers son de 16 bits lo que permite una temporizado de la ejecución de programa exacto (manejo de eventos), generación de ondas, y medidas de tiempos de señales. Sus principales caracterísiticas son:

- Diseño de 16 bits verdader (permite PWM de 16 bits)
- Tres unidades de comparación de salida independientes.
- Registros de comparación de salida con doble buffer.
- Unidad de captura con una entrada
- Cancelador de sonido de entrada de captura

- Borrado de timer al matchear la comparación (recarga automática)
- Salida PWM con período variable y fase correcta.
- Generador de frecuencia
- Contador de eventos externos
- Veinte fuentes de interrupción independientes.

#### 11.4.2.3. Modulador de comparación de salida

El modulador de comparación de salida (OCM) permite la generación de ondas con una frecuencia portadora. El modulador usa las salidas de de una unidad de comparación de salida de 16 bits y una de 8 bits.

#### 11.4.2.4. Interfase periférica serial (SPI)

La interfase periférica serial (SPI) permite la transferencia de datos sincrónicos a alta velocidad entre este microcontrolador y dispositivos perféricos u otros dispositivos AVR. Esta interfase dispone de las siguientes caracterísiticas:

- Transferencias sincrónicas de 3 cables full duplex.
- Operación en modos maestro o esclavo.
- Transferencias con LSB o MSB primero.
- Siete velocidades programables.
- Bandera de interrupción de final de transmición.
- Bandera de protección de colisión de escritura
- Despertado del modo Idle.
- Modo maestro de doble velocidad (CK/2).

#### 11.4.2.5. USART

Sus principales características son:

- Operación full duplex.
- Operación sincrónica u asincrónica.
- Operación maestro u esclava con reloj sincrónico.
- Generador de Baud Rate de alta resolución.
- Soporta frames con 5,6,7,8,9 bits de datos y 1 o 2 bits de stop.
- Generación de paridad por hardware.
- Detección de errores de framing y data overrun.
- Modo de comunicación multiprocesador.
- Soporta modo SPI Maestro.

#### 11.4.2.6. Interfaz 2-wire (TWI)

Esta interfaz es ideal para aplicaciones de microcontroladores. El protocolo TWI permitea al diseñador del sistema cnectar hasta 128 dispositivos usando solo dos lineas de bus bidireccionales. Esta interfaz tiene las siguientes caracterísicas:

- Operación en modo maestro o esclavo.
- Soporta arbitraje multi maestro.
- Velocidad de transferencia de hasta 400 Khz.
- Programable dirección de esclavo con soporte de llamadas generales.
- Reconocomiento de dirección permite despertar del modo sleep.

#### 11.4.2.7. Controlador USB

Esta interfase puede funcionar a velocidades low y full. El controlador usb provee el hardware para realizar una interfaz USB y la comunicación con el CPU se realiza mediante el mecanismo de memoria compartida (DPRAM). Este controlador posee las siguientes caracterísicicas:

- Soporta velocidades low y full.
- Soporta modo ping-pong buffering
- 832 bytes de memoria compartida
- Soporte de modo OTG
- Soporte de los cuatro tipos de transferencias (Control, Bulk, Interrupt, Isochronous)
- 7 endpoints:
  - 1 de 64 bytes maximo (endpoint de control por defecto)
  - 1 endpoint de 256 bytes máximo
  - 5 endpoints de 64 bytes máximo.

#### 11.4.2.8. Conversor analógico digital (ADC)

Este microcontrolar cuenta con un conversor analógico digital de aproximación sucesiva de 10 bits. Este conversor está conectado a un multiplexor analógico de 8 canales, lo que permite la obtención de 8 entradas de voltaje con referencia en 0V (GND). Este dispositivo también soporta 16 combinaciones de voltajes diferenciales. Otras características del conversor son las siguientes:

- Tiempo de conversión de entre 65 y 260 us
- Hasta 15 KSPS en máxima resolucion.
- Interrupción al finalizar la conversión.
- Cancelador de ruido en modo sleep.

## 11.5. Caracterísitcas especiales del microcontrolador

Este microcontrolador tiene algunas características especiales que vale la pena resaltar como lo son: soporte para boot loader, multiplicador por hardware e interfaz JTAG.

### 11.5.1. Soporte para Boot Loader

El soporte de Boot Loader provee un mecanismo de lectura-mientras-escribe real que permite la bajada o subida de código de programa por el MCU mismo. Esto permite tener aplicaciones flexibles que permite upgrades por software por el MCU usando un programa Boot Loader residente. El Boot loader de esta manera, puede utilizar cualquier interfaz y protocolo asociado para leer código y escribir ese código en la memoria Flash o viceversa. Este mecansimo incluye también el área del bootloader, por lo que éste se puede hasta modificar a si mismo, y borrarse si ya no es más necesario. Mediante fusibles se pueden configurar niveles de protección para áreas de Boot loader y aplicación para permitir escribir en éstas areas de memoria.

### 11.5.2. Multiplicador por hardware

Este microcontrolador posee en su ALU un multiplicador por hardware que permite operaciones con y sin signo. De esta forma se puede acelerar las aplicaciones que hagan uso de esta caracterísitca.

### 11.5.3. Interfaz JTAG y sistema de debug en el chip

Esta interfaz puede ser usada para: el testeo de PCBs usando la capacidad JTAG Boundary-scan, la programación de memoria no volátil (fusibles y bits de candado) y debugging.

# Bibliografía


[1] AGG Software, http://www.aggsoft.com, último acceso octubre 2006.
(Citado en la página 60.)

[2] ANDERSON, Don. USB System architecture. 2nd Edition. Addison-Wesley Professional, 2001.
(Citado en la página 13.)

[3] AXELSON, Jan. USB Complete. 3rd Edition. Lake View Research, 2001.
(Citado en la página 13.)

[4] BAKER, Art et al. The Windows 2000 device driver book, a guide for programmers. 3rd Edition. Prentice HALL PTR, 2000, .
(Citado en la página 40.)

[5] Catalyst Enterprise Inc., http://www.getcatalyst.com/index.html, último acceso octubre 2006.
(Citado en las páginas 63 y 64.)

[6] CORBET, Jonathan; RUBINI, Alessandro; KROAH-HARTMAN, Greg. Linux device drivers. 3rd Edition, O' Reilly Media, 2005.
(Citado en la página 43.)

[7] ElliSys Sàrl, http://www.ellisys.com, último acceso octubre 2006.
(Citado en las páginas 61 y 62.)

[8] EnTech Taiwan, http://www.entechtaiwan.com/index.shtm, último acceso octubre 2006.
(Citado en las páginas 47 y 53.)

[9] Fabula Tech Inc., http://www.fabulatech.com, último acceso octubre 2006.
(Citado en la página 59.)

[10] HHD Software Ltd., http://www.hhdsoftware.com, último acceso octubre 2006.
(Citado en la página 57.)

[11] Icaste llc, http://www.icaste.com, último acceso octubre 2006.
(Citado en la página 48.)

[12] JSR80, http://javax-usb.org, último acceso octubre 2006.
(Citado en la página 51.)

[13] Jungo Ltd., http://www.jungo.com, último acceso octubre 2006.
(Citado en las páginas 44 y 52.)

[14] jUSB, http://jusb.sourceforge.net, último acceso octubre 2006.
(Citado en la página 51.)

[15] LibUSB, http://libusb.sourceforge.net, último acceso octubre 2006.
(Citado en la página 50.)

[16] LibUSBWin32, http://libusb-win32.sourceforge.net, último acceso octubre 2006.
(Citado en la página 50.)

[17] Linux Driver Development Kit, http://www.kroah.com/log/2006/05/24/#ddk, último acceso octubre 2006.
(Citado en la página 43.)

[18] MicrochipDatasheets y Application notes de pic18f45xx. Disponible vía Web en http://www.microchip.com/stellent/idcplgidcplg?IdcService=SS_GET_PAGE&nodeId=1335&dDocName=en010300, visitada en julio 2006.
(Citado en la página 83.)

[19] Microchip Technology Inc., http://www.microchip.com, último acceso octubre 2006.
(Citado en la página 50.)

[20] Parallel Technologies Inc., http://www.usbinfo20.com, último acceso octubre 2006.
(Citado en la página 58.)

[21] PEACOCK, Graig. USB In a nutshell making sense of the USB standard. 3rd Edition, 2002. Disponible vía Web en www.beyondlogic.org, visitada en julio 2006.
(Citado en la página 13.)

[22] Perisoft, http://www.perisoft.net, último acceso octubre 2006.
(Citado en la página 60.)

[23] Philips. Datasheets y Application notes de Philips ISP158x USB peripheral controller. Disponible vía Web en http://www.semiconductors.philips.com/cgi-bin/pldb/pip/isp1581.html, visitada en julio 2006.
(Citado en la página 69.)

[24] SourceQuest Inc., http://www.sourcequest.com, último acceso octubre 2006.
(Citado en la página 55.)

[25] Texas Intruments. Datasheets y Application notes de la línea TUSB. Disponible vía Web en http://focus.ti.com/docs/prod/folders/print/tusb3210.html, visitada en julio 2006.
(Citado en la página 73.)

[26] Thesycon Systemsoftware & Consulting GmbH, http://www.thesycon.de/eng/home.shtml, último acceso octubre 2006.
(Citado en la página 46.)

[27] Universal Serial Bus Common Class Specification SYSTEMSOFT CORPORATION INTEL CORPORATION Revision 1.0 December 16, 1997. Disponible vía Web en http://www.usb.org/developers/devclass_docs/usbccs10.pdf visitada en octubre 2006.
(Citado en la página 35.)

[28] USB, Estándar y especificaciones técnicas. Disponible vía Web en http://www.usb.org, visitada en julio 2006.
(Citado en las páginas 33, 34 y 35.)

[29] Windows Driver Development Kit, http://www.microsoft.com/whdc/devtools/ddk/default.mspx, último acceso octubre 2006.
(Citado en la página 43.)

[30] Windows Driver Model, http://msdn2.microsoft.com/en-us/library/aa490248.aspx, último acceso octubre 2006.
(Citado en la página 40.)

# Apéndice A

# Glosario:

**ACK:** Abreviatura de ***Ack****nowledgement* (Asentimiento positivo), en comunicaciones entre computadores, es un mensaje que se envía para confirmar que un mensaje o un conjunto de mensajes han llegado. Si el terminal de destino tiene capacidad para detectar errores, el significado de ACK es "ha llegado y además ha llegado correctamente".

**ADC:** Acrónimo de ***A****nalog* ***D****igital* ***C****onversor* (Conversor Analógico-Digital), es un circuito integrado electrónico que convierte señales continuas a números digitales discretos.

**API:** Sigla de ***A****pplication* ***P****rogramming* ***I****nterface* (Interfaz de Programación de Aplicaciones), es el conjunto de funciones y procedimientos que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción. Una buena API hace más fácil desarrollar un programa mediante el suministro de todos los elementos de construcción.

**ATA:** Acrónimo de ***A****dvanced* ***T****echnology* ***A****ttachment* (Tecnología Avanzada de Fijación), es una interfaz estándar para conectar dispositivos de almacenamiento como discos duros y unidades de CD-ROM dentro de los computadores personales.

**ATAPI:** Acrónimo de ***A****dvanced* ***T****echnology* ***A****ttachment* ***P****acket* ***I****nterface* (Interfaz de Paquetes de Tecnología Avanzada de Fijación), es una extensión de la interfaz ATA y permite conectar medio removibles.

**B:** Lenguaje de programación creado en 1969 por Ken Thompson con contribuciones de Dennis Ritchie en los Laboratorios Bell, predecesor del lenguaje de programación C.

**Bit:** Acrónimo de ***B****inary* ***Digit*** (Dígito Binario), es un dígito del sistema de numeración binario.

**Bluetooth:** Es el nombre común de la especificación industrial IEEE 802.15.1, que define un estándar global de comunicación inalámbrica que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia segura, globalmente y sin licencia de corto rango.

**Boot-Loader:** Es un firmware que permite la rápida descarga de programas en los microcontroladores. En el caso de los PIC, el boot-loader permite descargar programas directamente desde el PC sin necesidad de utilizar ningún tipo de grabador.

**BSD:** Acrónimo de Berkeley Software Distribution (Distribución de Software Berkeley) y se utiliza para identificar un sistema operativo derivado del sistema Unix nacido a partir de los aportes realizados a este sistema por la Universidad de California en Berkeley.

**Byte:** Unidad básica de almacenamiento de información que hace referencia a una secuencia de ocho bits contiguos. También se utiliza el termino octeto como sinónimo preciso de la cantidad de bits que representa.

**ByteCode:** Es un código intermedio más abstracto que el código máquina.

**C:** Lenguaje de programación creado en 1972 por Ken Thompson y Dennis M. Ritchie en los Laboratorios Bell como evolución del anterior lenguaje B.

**C++:** Lenguaje de programación, diseñado a mediados de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C. Se puede decir que C++ es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos.

**Callback:** (Retro llamada) Es un código ejecutable que es pasado como argumento a otro código. Esto permite a una capa de software de más bajo nivel invocar una rutina (o función) definida en una capa de más alto nivel.

**CDC:** Acrónimo de ***C**omunications **D**evice **C**lass* (Clase de Dispositivo de Comunicaciones), es una de las clases que define el estándar USB y permite clasificar a los dispositivos según su comportamiento esperado.

**Chip:** Sinónimo de circuito integrado, es una pastilla muy delgada en la que se encuentran una enorme cantidad (del orden de miles o millones) de dispositivos microelectrónicos interconectados, principalmente diodos y transistores, además de componentes pasivos como resistencias o condensadores.

**COM:** Acrónimo de ***C**omponent **O**bject **M**odel* (Modelo de Objetos de Componentes), es una plataforma de Microsoft para componentes de software introducida en 1993. Permite la comunicación entre procesos y la creación dinámica de objetos en cualquier lenguaje de programación que soporte dicha tecnología.

**Compound:** Se llama dispositivos compound a aquellos que contienen tanto un dispositivo como un hub.

**Composite:** Se llama dispostivos composite a aquellos que contienen múltiples funciones e interfaces independientes en el mismo dispositivo. Posee una sola dirección en el bus pero cada interface tiene una función distinta y especifica su propio driver en el host.

**CPU:** Sigla de ***C**entral **P**rocessing **U**nit* (Unidad Central de Procesos), es el componente en una computadora digital que interpreta las instrucciones y procesa los datos contenidos en los programas de computadora. Es la parte que constituye el cerebro de cualquier computadora, es el encargado de realizar y dirigir todas las funciones.

**DAC:** Sigla de ***D**igital **A**nalog **C**onversor* (Conversor digital-analógico), es un dispositivo para convertir datos digitales en señales de corriente o de tensión analógica.

**DarwinBSD:** Es un sistema que subyace en MacOS, cuya primera versión fue liberada en 2001 para funcionar en ordenadores Macintosh.

**Descriptor:** Son estructuras de datos que describen las capacidades del usb y como van a ser usadas.

**DIL:** Véase DIP.

**DIP:** Acrónimo de ***D**ual **I**n-line **P**ackage*, es un tipo de circuito integrado alojado en una carcasa rectangular y de dos filas paralelas de pines de conexión eléctrica.

**DLL:** Acrónimo de ***D**ynamic **L**ink **L**ibrary* (Biblioteca de Vinculación Dinámica), por vinculación dinámica se entiende que las subrutinas de la biblioteca son cargadas en un programa de aplicación en tiempo de ejecución.

**DMA:** Acrónimo de ***D**irect **M**emory **A**ccess* (Acceso Directo a Memoria), es una característica de los computadores modernos, que permite a ciertos componentes de hardware dentro del computador, a acceder al sistema de memoria para lectura y/o escritura en forma independiente al CPU.

**Driver:** (Controlador) Programa que permite al sistema operativo interactuar con un dispositivo, haciendo una abstracción del hardware y proporcionando una interfaz -posiblemente estandarizada- para usarlo.

**E/S:** Abreviatura ***E**ntrada/**S**alida*, es la colección de interfaces que usan las distintas unidades funcionales de un sistema de procesamiento de información para comunicarse con otras, o las señales enviadas a través de esas interfaces.

**EEPROM:** Acrónimo de ***E****lectrically* ***E****rasable* ***P****rogrammable* ***R****ead* ***O****nly* ***M****emory* (Memoria de Sólo Lectura Programable y Borrable Eléctricamente), es un tipo de memoria ROM que puede ser programado, borrado y reprogramado eléctricamente. Sólo puede ser borrada y reprogramada entre 100.000 y 1.000.000 de veces.

**FIFO:** Acrónimo de ***F****irst* ***I****n,* ***F****irst* ***O****ut* (Primero en Entrar, Primero en Salir), es un método utilizado en estructuras de datos, contabilidad de costes y teoría de colas. Guarda la analogía con las personas que esperan en una cola y van siendo atendidas en el orden en que llegan.

**FireWire:** Véase IEEE 1394.

**Firmware:** Es un bloque de instrucciones de programa para propósitos específicos, grabado en una memoria tipo ROM, que establece la lógica de más bajo nivel que controla los circuitos electrónicos de un dispositivo.

**FreeBSD:** Es un sistema operativo libre basado en los sistemas BSD para ordenadores personales basado en los CPU de arquitectura Intel.

**Free_Software:** (Software Libre) Es la denominación del software , que una vez obtenido, puede ser usado, copiado, estudiado, modificado, mejorado y redistribuido libremente de forma de beneficiar a toda la comunidad.

**FSF:** Acrónimo de ***F****ree* ***S****oftware* ***F****oundation* (Fundación para el Software Libre), es una organización creada en octubre de 1985 por Richard Stallman y otros entusiastas del Software Libre con el propósito de difundir este movimiento. Una de las principales funciones de la FSF es dar cobertura legal, económica y logística al Proyecto GNU.

**GNU:** Acrónimo recursivo de ***G****NU is* ***N****ot* ***U****nix* (GNU No es Unix), es un sistema operativo de computadores compuesto enteramente por software libre. Se lo conoce también como Proyecto GNU y fue iniciado por Richard Stallman en 1983.

**GPL:** Acrónimo de ***G****eneral* ***P****ublic* ***L****icense* (Licencia Pública General), es una licencia creada por la FSF a mediados de los años 80, y está orientada principalmente a proteger la libre distribución, modificación y uso de software.

**HAL:** Acrónimo de ***H****ardware* ***A****bstraction* ***L****ayer* (Capa de Abstracción de Hardware), es un elemento del sistema operativo que funciona como interfaz entre el software y el hardware del sistema, proveyendo una plataforma de hardware consistente sobre la cual correr las aplicaciones.

**HEX:** Su nombre formal es *Intel HEX* y corresponde a un formato de archivo para transmitir información binaria de aplicaciones programadas para microcontroladores, ROMs o cualquier otro tipo de chip. Éste formato es uno de los más viejos disponibles para este propósito y se usa desde los años 1970. El formato es un archivo de texto, donde cada línea contiene valores hexadecimales que codifican una secuencia de datos y sus corrimientos o direcciones absolutas.

**HID:** Acrónimo de ***H****uman* ***I****nterface* ***D****evices* (Dispositivo de Interfaz Humana), es un tipo de dispositivo de un computador, que interactúa directamente con seres humanos, tomando y/o devolviendo información.

**Hot_Plug:** (Enchufado en caliente) Véase Hot Swap.

**Hot_Swap:** (Sustitución en caliente) Hace referencia a la capacidad de algunos componentes hardware para realizar su instalación o sustitución sin necesidad de detener o alterar la operación normal de la computadora donde se alojan.

**I/O:** Abreviatura de ***I****nput/* ***O****utput*, véase E/S.

**I2C:** Acrónimo de ***I****nter-****I****ntegrated* ***C****ircuit* (Inter Circuitos Integrados), es un bus de comunicaciones serie diseñado por Phillips en el año 1992.

**IDE:** Acrónimo de ***I**ntegrated **D**evelopment **E**nvironment* (Entorno integrado de desarrollo), es un programa compuesto por un conjunto de herramientas (un editor de código, un compilador, un depurador y un constructor de interfaz gráfica) para brindar un marco de trabajo amigable a un programador.

**IEEE_1394:** Conocido como FireWire por Apple Inc. y como i.Link por Sony, es un estándar multiplataforma para entrada/salida de datos en serie a gran velocidad. Suele utilizarse para la interconexión de dispositivos digitales como cámaras digitales y videocámaras a computadoras.

**IEEE_802:** Es un comité y grupo de estudio de estándares perteneciente al Instituto de Ingenieros Eléctricos y Electrónicos (IEEE), que actúa sobre Redes de Ordenadores, concretamente y según su propia definición sobre redes de área local y redes de área metropolitana.

**IRP:** Acrónimo de ***I**/O **R**equest **P**acket* (Paquete de Petición E/S), es una estructura de modo kernel que es usada por WDM para comunicarse internamente y con el sistema operativo.

**IOCTL:** Acrónimo de ***I**nput/**O**utput **C**ontrol* (Control entrada/salida), la llamada de sistema ioctl en los sistemas basados en Unix, permite a una aplicación controlar o comunicarse con un driver de dispositivo fuera de los usuales read/write de datos.

**ISR:** Acrónimo de ***I**nterrupt **S**ervice **R**outine* (Rutina de Servicio de Interrupción), es una rutina (callback) en un sistema operativo o driver de dispositivo encargada de atender una interrupción del hardware. Su ejecución es disparada por la recepción de la interrupción.

**Java:** Lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 1990. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel como punteros.

**JIT:** Sigla de ***J**ust **I**n **T**ime* (Compilación en tiempo de ejecución), es una técnica para mejorar el rendimiento de sistemas de programación que compilan a bytecode, consistente en traducir el bytecode a código máquina nativo en tiempo de ejecución.

**JNI:** Sigla de ***J**ava **N**ative **I**nterface*, es un framework de programación que permite que un programa escrito en Java ejecutado en la máquina virtual Java pueda interactuar con programas escritos en otros lenguajes como C, C++ y ensamblador.

**Jumper:** Elemento para interconectar dos terminales de manera temporal sin tener que efectuar una operación que requiera herramienta adicional, dicha unión de terminales cierran el circuito eléctrico del que forma parte.

**kB:** (kilobyte) Es una unidad de información o de almacenamiento informático equivalente a mil bytes o 1.024 bytes, dependiendo el contexto. Puede ser abreviado como: K, KB, Kbytes.

**Kernel:** (Núcleo) Es la parte fundamental de un sistema operativo. Es el software encargado de gestionar los recursos del sistema.

**kHz:** (kilohertz) Unidad de frecuencia equivalente a mil ciclos por segundo.

**LED:** Sigla de ***L**ight **E**mitting **D**iode* (Diodo emisor de luz), es un dispositivo semiconductor (diodo) que emite luz cuasi-monocromática, es decir, con un espectro muy angosto, cuando se polariza de forma directa y es atravesado por una corriente eléctrica.

**LGPL:** Acrónimo de ***L**esser **G**eneral **P**ublic **L**icense* (Licencia Pública General Menor), es una licencia que aplica generalmente a bibliotecas y permite a programas privativos (no libres) utilizar las bibliotecas como parte de sus trabajos. Esta licencia mantiene términos más laxos que la GPL en que necesariamente todas las partes deben ser Software Libre.

**Linux:** Es la denominación de un sistema operativo tipo Unix (creado en 1992) y el nombre de un kernel (creado por Linus Torvalds en 1991). Es uno de los paradigmas más prominentes del Software Libre y del desarrollo del código abierto.

**MacOS:** Acrónimo de ***M**acintosh **O**perating **S**ystem* (Sistema Operativo de Macintosh), es el nombre del primer sistema operativo de Apple Computer para las computadoras Macintosh.

**MB:** (megabyte) Es una unidad de información o de almacenamiento informático equivalente a $10^6$bytes o $2^{20}$ bytes dependiendo el contexto. Puede ser abreviado como Mbyte.

**Mb:** (megabit) Es una unidad de información o de almacenamiento informático equivalente un millón de bits. Puede ser abreviado como Mbit.

**Mbps:** (megabit per second) Es una unidad tasa de transferencia de datos equivalente a un millón de bits por segundo. Puede ser abreviado como Mbit/s o mbps).

**MHz:** (megahertz) Unidad de frecuencia que equivale a un millón de ciclos por segundo.

**MIDI:** Acrónimo de ***M**usical **I**nstrument **D**igital **I**nterface* (Interfaz Digital de Instrumentos Musicales). Es un protocolo estándar que permite a los computadores, sintetizadores, secuenciadores y otros dispositivos musicales electrónicos, comunicarse entre si para la generación de sonidos.

**MPLAB:** Es un editor IDE gratuito, destinado a productos de la marca Microchip. Este editor es modular, permite seleccionar los distintos microprocesadores soportados, además de permitir la grabación de estos circuitos integrados directamente al programador.

**MPLAB_C18:** Es un compilador C compatible con ANSI C y completo para la familia PIC18 de PICmicro de 8-bits.

**MPLAB_ICD2:** Depurador (con la característica de depuración directa en el circuito) y programador del firmware de los microcontroladores de la empresa Microchip.

**NACK:** Acrónimo de ***N**egative **Ack**nowledgement* (Asentimiento negativo), en comunicaciones entre computadoras, es un mensaje que se envía para informar de que en la recepción o procesamiento de los datos ha habido un error.

**NetBSD:** Es un sistema operativo del tipo Unix basado en los sistemas BSD, de distribución libre y de códigos fuentes abiertos.

**Ohm:** (Ohmnio) Es la unidad de resistencia eléctrica en el Sistema Internacional de Unidades. Su nombre se deriva del apellido del físico alemán Georg Simon Ohm, autor de la Ley de Ohm.

**OpenBSD:** Es un sistema operativo libre tipo Unix, multiplataforma, basado en los sistemas BSD. Es un descendiente de NetBSD, con un foco especial en la seguridad y la criptografía.

**PC:** Sigla de ***P**ersonal **C**omputer* (Computadora Personal), término genérico utilizado para referirse a microcomputadores que se basan en un microcontrolador Intel o compatible.

**PCB:** Sigla de ***P**rinted **C**ircuit **B**oard* (Circuito Impreso), es un medio para sostener mecánicamente y conectar eléctricamente componentes electrónicos, a través de rutas o pistas de material conductor, grabados desde hojas de cobre laminadas sobre un sustrato no conductor.

**PCMCIA:** Sigla de ***P**ersonal **C**omputer **M**emory **C**ard **I**nternational **A**ssociation.* Estándar para tarjetas de expansión para laptops.

**PIC:** Son una familia de microcontroladores tipo RISC fabricados por Microchip Technology Inc. y derivados del PIC1650, originalmente desarrollado por la división de microelectrónica de General Instruments. El nombre actual no es un acrónimo, en realidad el nombre completo es PICmicro.

**PID:** Acrónimo de ***P**roduct **ID*** (Identificador de Producto), es un identificador único asignado por el USB-IF a los productos fabricados por las empresas registradas.

**Plug_and_Play:** (Enchufar y listo) Se refiere a la tecnología que permite a un dispositivo informático ser conectado a una computadora sin tener que configurar el mismo.

**Plug-In:** (Enchufar en) Pequeño programa que añade alguna función a otro programa, habitualmente de mayor tamaño. Un programa puede tener uno o más conectores.

**Polling:** (Escrutiño) Refiere a la toma de muestras en forma activa del estado de un dispositivo externo, por medio de un programa sincrónico. Este concepto se usa normalmente asociados a escrutinios de E/S.

**Proxy:** (Intermediario, Apoderado) En su forma más general, es una clase que funciona como interfaz de otra, logrando ocultar su verdadera identidad.

**PWM:** Sigla de ***P****ulse* ***W****idth* ***M****odulation* (Modulación en el ancho del pulso). Forma de codificar un valor análogo en una onda digital, de forma que el ciclo de trabajo está en relación con el valor a codificar.

**Python:** Es un lenguaje de programación (interpretado) que soporta múltiples paradigmas de programación (funcional, orientado a objetos y imperativo) y fue creado por Guido van Rossum en el año 1990.

**QFN:** Acrónimo de ***Q****uad* ***F****lat Package* ***N*** *o Lead* (Encapsulado Cuadrado Plano Sin Conectores), es un encapsulado de circuito integrado para montaje superficial donde los conectores de componentes no se extienden fuera del encapsulado.

**RAM:** Sigla de ***R****andom* ***A****ccess* ***M****emory* (Memoria de acceso aleatorio), es una memoria de semiconductores en la que se puede escribir o leer información. Es volátil, es decir, pierde su contenido al desconectar la energía eléctrica.

**Ribbon_Cable:** (Cable Cinta) También conocido como cable plano multihilo, es un cable con muchos cables conductores que corren en forma paralela unos con otros sobre el mismo plano. Como resultado, el cable es ancho y chato en vez de redondo. Su nombre proviene de la semejanza de estos cables con un pedazo de cinta. Son normalmente utilizados en periféricos internos de un computador personal.

**ROM:** Sigla de ***R****ead* ***O****nly* ***M****emory* (Memoria de sólo lectura), es una memoria de semiconductores que puede leerse pero no modificarse. La ROM suele almacenar la configuración del sistema o el programa de arranque de la computadora.

**RTC:** Acrónimo de ***R****eal* ***T****ime* ***C****lock* (Reloj de Tiempo Real), es un reloj un computador (la mayoría de las veces en forma de circuito integrado) que permite mantener la pista del tiempo actual.

**SIE:** Acrónimo de ***S****erial* ***I****nterface* ***E****ngine* (Motor de Interfaz Serial), es un circuito integrado que normalmente se encarga de la recepción y envío de datos de las transacciones y trabaja en conjunto con el transceiver USB y los endpoints. El SIE no interpreta o utiliza los datos, sólo envía los datos que están disponibles para hacerlo y almacena los datos recibidos.

**SIL:** Acrónimo de ***S****ingle* ***I****n-****L****ine Package*, es un tipo de circuito integrado alojado en una carcasa que posee una única fila de pines de conexión eléctrica.

**SMT:** Sigla de ***S****urface* ***M****ount* ***T****echnology* (Tecnología de montaje superficial), es un método para la construcción de circuitos electrónicos en el que los componentes se montan directamente sobre la superficie de los circuito impreso.

**Solaris:** Es un sistema operativo desarrollado por Sun Microsystems. Es un sistema certificado como una versión de Unix. Aunque Solaris en sí mismo aún es software propietario, la parte principal del sistema operativo se ha liberado como Software Libre.

**SPI:** Sigla de ***S****erial* ***P****eripherical* ***I****nterface* (Bus de Interfaz de Periféricos Serie), es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos.

**TAD:** Acrónimo de ***T****ipo* ***A****bstracto de* ***D****atos* (Abstract Data Type), es una especificación de un conjunto de datos y un conjunto de operaciones que pueden ser realizadas sobre los datos.

**Through-Hole_Technology:** (Tecnología A Través de Orificio) Se refiere al esquema utilizado para el montaje de componentes electrónicos, que implica la inserción de los pines de los componentes en orificios perforados en el circuito electrónico impreso y el soldado al circuito en el lado opuesto.

**TimeOut:** (Tiempo vencido) Hace referencia a la conclusión intencional de una tarea incompleta luego de superado un tiempo límite estipulado para su conclusión en forma normal.

**Transceivers:** (Transceptores) Es un dispositivo que realiza funciones tanto de transmición como de recepción, utilizando componentes de circuito comunes para ambas funciones.

**UART:** Acrónimo de ***U****niversal* ***A****synchronous* ***R****eceiver* ***T****ransmitter* (Transmisor Receptor Asíncrono Universal), es un circuito integrado (o parte de uno) usado para comunicaciones seriales en computadores que forma parte de los puertos seriales de computadores personales o periféricos.

**Unix:** Es un sistema operativo portable, multitarea y multiusuario; desarrollado en principio por un grupo de empleados de los laboratorios Bell de AT&T, entre los que figuran Ken Thompson, Dennis Ritchie y Douglas McIlroy.

**URB:** Acrónimo de ***U****SB* ***R****equest* ***B****lock* (Petición en Bloque USB), son estructuras utilizadas a nivel de los sistemas operativos, que permiten la configuración de los dispositivos y transmición de datos. Estas estructuras son enviadas por medio de un IRP a la capas inferiores de la pila de drivers.

**USB:** Sigla de ***U****niversal* ***S****erial* ***B****us* (Bus Universal en Serie), es una interfaz que provee un estándar de bus serie para conectar dispositivos a un ordenador personal. Fue creado en 1996 por IBM, Intel, Northern Telecom, Compaq, Microsoft, Digital Equipment Corporation y NEC.

**USB_Hub:** (Concentrador USB) Es un dispositivo que permite tener varios puertos USB a partir de uno sólo.

**VID:** Acrónimo de ***V****endor* ***ID*** (Identificador de Vendedor), es un identificador único asignado por el USB-IF a las empresas (previo registro) que quieren fabricar dispositivos USB.

**WiFi:** Acrónimo de ***Wi****reless* ***Fi****delity*. Es un conjunto de estándares para redes inalámbricas basados en las especificaciones IEEE 802.11.

**WINE:** Acrónimo recursivo de ***W****ine* ***I****s* ***N****ot an* ***E****mulator* (Wine no es un emulador), es una reimplementación de la API de Windows (en sus versiones de 16-bits y 32-bits) para sistemas operativos basados en Unix bajo plataformas Intel.

**Windows:** (Microsoft Windows) Es un sistema operativo con interfaz gráfica para computadoras personales propiedad de la empresa Microsoft.

**Wireless_USB:** (USB Inalámbrico) Es un protocolo de comunicación inalámbrico por radio con gran ancho de banda que combina la sencillez de uso de USB con la versatilidad de las redes inalámbricas. En mayo del 2006 se aprobó la especificación del nuevo estándar que se abrevia como W-USB o WUSB.

**Wrapper:** (Envoltura) Es una clase que se utiliza para transformar una interfaz en otra, de tal modo que una clase que no pudiera utilizar la primera, haga uso de ella a través de la segunda.