

USB al Grano

Índice general

1. Introducción	5
2. Hardware	7
2.1. Conectores	7
2.2. Eléctrico	8
2.3. Indentificación de velocidad	8
2.4. Alimentación (V_{Bus})	9
2.5. Corriente de suspensión	10
2.6. Entrando al modo de suspensión	10
2.7. Tasa de señal de datos	11
3. Protocolos USB	13
3.1. Campos comunes de paquetes USB	13
3.2. Tipos de paquetes	14
3.3. Funciones USB	15
3.4. Terminales (endpoints)	16
3.5. Tuberías (pipes)	16
4. Tipos de EndPoints	19
4.1. Transferencias de Control	19
4.2. Transferencias de control, un panorama más amplio	21
4.3. Transferencias de interrupción	22
4.4. Transferencias Isócronas	23
4.5. Transferencias de bulto (bulk)	24
4.6. Administración de ancho de banda	25
5. Descriptores USB	27
5.1. Composición de los descriptores USB	29
5.2. Descriptores de dispositivo	29
5.3. Descriptores de configuración	30
5.4. Descriptores de Interfaz	31
5.5. Descriptores de Endpoint	32
5.6. Descriptores de Cadena (String)	33
6. Peticiones USB	35
6.1. El paquete de configuración (Setup)	35
6.2. Peticiones estándar	36
6.2.1. Peticiones estándar de dispositivo	36
6.2.2. Peticiones estándar de Interfaz	38
6.2.3. Peticiones estándar de Endpoint	38

Capítulo 1

Introducción

La versión de USB 1.1 soportaba dos velocidades: un modo de velocidad completa de 12Mbps/s y un modo de baja velocidad de 1.5Mbps/s. El modo de 1.5Mbps/s es más lento y menos susceptible a interferencia electromagnética (EMI, o Electromagnetic Interference), reduciendo así el costo de filtros de ferrita y componentes de calidad. Por ejemplo, los cristales pueden reemplazarse por resonadores más baratos. El USB 2.0 ha subido la barra hasta 480Mbps/s. Los 480Mbps/s se conoce como modo de Alta Velocidad y fue una adición para competir con el Bus Serial Firewire.

Velocidades del USB

- High speed - 480 Mb/s
- Full speed - 12 Mb/s
- Low speed - 1.5 Mb/s

El USB es controlado por un anfitrión (host). Solo puede haber un anfitrión por bus. La especificación en sí misma no soporta ninguna forma de arreglo multimaestro. Sin embargo la especificación On-The-Go, la cual es un estándar suplementario al USB 2.0, ha introducido el Protocolo de Negociación de Host el cual permite a dos dispositivos negociar el papel de anfitrión. Esto está enfocado hacia, y limitado a, conexiones individuales punto a punto tales como teléfonos móviles y agendas electrónicas, y no a configuraciones de hubs múltiples/dispositivos múltiples. El anfitrión USB es responsable de emprender todas las acciones y de fijar (schedule) el ancho de banda. Los datos pueden enviarse por varios métodos de transacción usando un protocolo basado en tokens.

Desde mi punto de vista, la topología de bus del USB es un tanto limitante. Una de las intenciones originales del USB era reducir la cantidad de cableado en la parte trasera de su PC. Los usuarios de Apple dirán que la idea vino del Bus de Escritorio de Apple (Apple Desktop Bus), en el que tanto el teclado, ratón y algunos otros periféricos podían conectarse conjuntamente en cadena (daisy chained) usando el mismo cable.

Sin embargo el USB es una topología de estrella por niveles (tiered star), similar a la de 10BaseT Ethernet. Esto impone el uso de un hub en algún lugar, lo cual supone un precio mayor, más cajas en su escritorio y más cables. Sin embargo no es tan malo como parece. Muchos dispositivos tienen hubs USB integrados. Por ejemplo, su teclado puede contener un hub que está conectado a su computadora. Su ratón y otros dispositivos tales como su cámara digital pueden conectarse fácilmente en la parte trasera de su teclado. Los monitores son solo otro periférico de una larga lista de los que comúnmente tienen hubs integrados.

Esta topología de estrella por niveles, en lugar de simplemente una con dispositivos conectados en cadena, tiene algunos beneficios. Primeramente, la alimentación a cada dispositivo puede monitorearse e incluso apagarse si una condición de sobrecarga de corriente ocurre, sin interrumpir otros dispositivos USB. Tanto dispositivos de velocidad alta, completa y baja pueden ser soportados, haciendo que el hub filtre transacciones de velocidad alta y completa para que los dispositivos de menor velocidad no las reciban.

Pueden conectarse hasta 127 dispositivos a cualquier bus USB en cualquier momento. ¿Necesita más dispositivos? Simplemente agregue otro puerto/host. Si bien la mayoría de anfitriones (hosts) USB más antiguos tenían dos puertos, la mayoría de fabricantes ha visto esto como una limitación y están comenzando a introducir tarjetas anfitrionas de 4 y 5 puertos con un puerto interno para discos duros, etc. Los primeros anfitriones tenían un controlador USB y de ese modo ambos puertos compartían el mismo ancho de banda USB. A medida que los requerimientos de ancho de banda aumentaban, comenzamos a ver tarjetas multipuerto con dos o más controladores permitiendo canales individuales.

Los controladores anfitrión USB tienen su propia especificación. Con el USB 1.1, había dos Especificaciones de Interfaz de Controlador Anfitrión: UHCI (Universal Host Controller Interface o Interfaz Universal de Controlador Anfitrión) desarrollada por Intel lo cual agrega más carga de software (Microsoft) y que permite hardware más barato; y OHCI (Open Host Controller Interface o Interfaz Abierta de Controlador Anfitrión) desarrollada por Compaq, Microsoft y National Semiconductor, la cual agrega más carga de hardware (Intel) y permite software más simple. Una relación típica de ingenieros de hardware/software.

Con la introducción del USB 2.0 se necesitaba una nueva especificación de Interfaz de Controlador Anfitrión, para describir los detalles a nivel de registros específicos al USB 2.0. Así nació EHCI (Enhanced Host Controller Interface o Interfaz Mejorada de Controlador Anfitrión). Entre los contribuyentes significativos están Intel, Compaq, NEC, Lucent y Microsoft, así que parece que se han unido para ofrecernos un estándar de interfaz y de esa forma solo un nuevo driver que implementar en nuestros sistemas operativos. Ya era hora.

USB como su nombre sugiere es un bus serial. Usa 4 cables blindados de los cuales dos son energía (+5V & GND). Los dos restantes son señales de datos diferenciales de par trenzado/cruzado. Este usa un esquema de codificación NZRI (Non Return to Zero Invert o No Retorno a Cero para Invertir Cero) para enviar datos con un campo de sincronización (sync) para sincronizar el anfitrión y relojes de receptor.

USB soporta Plug'n Play con drivers dinámicamente cargables y descargables. El usuario simplemente conecta el dispositivo en el bus. El host detectará esta adición, interrogará el dispositivo que se acaba de conectar y cargará el driver apropiado; todo eso en el tiempo que el puntero de reloj se lleva en parpadear en su pantalla, siempre que un driver esté instalado para su dispositivo. El usuario final no necesita preocuparse de terminaciones, términos como IRQs y direcciones de puertos, o reiniciar la computadora. Una vez que el usuario termina, puede simplemente desconectar el cable; el anfitrión detectará su ausencia y descargará automáticamente el driver.

La carga de driver apropiado se hace usando una combinación PID/VID (Product ID/Vendor ID, o Identificación de Producto/Vendedor). El VID se proporciona por el foro del Implementador USB a un costo y esto es visto como otro sticking point para el USB. La información más reciente de las cuotas se puede encontrar en el Sitio Web de Implementadores de USB.

Otras organizaciones estándar proporcionan un VID extra para actividades no comerciales tales como la enseñanza, investigación o experimentación (los aficionados). El foro de Implementadores de USB todavía está en espera de ofrecer este servicio. En estos casos puede desear usar uno asignado a su fabricante de sistema de desarrollo. Por ejemplo la mayoría de fabricantes de chips tendrán una combinación VID/PID que puede usar para sus chips la cual se sabe que no existe como un dispositivo comercial. Otros fabricantes de chips pueden incluso venderle un PID para usarse con su VID (el de ellos) para su dispositivo comercial (el suyo).

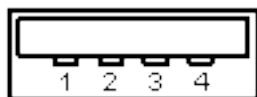
Otra característica más notable del USB son sus modos de transferencia. USB soporta transferencias de Control, Interrupción, e Isócrona. Si bien veremos los otros modos de transferencia más adelante, la isócrona permite a un dispositivo reservar una cantidad definida de ancho de banda con latencia garantizada. Esto es ideal en aplicaciones de Audio o Video en las que la congestión puede causar pérdida de datos o fotogramas descartados. Cada modo de transferencia ofrece al diseñador ventajas y desventajas en áreas tales como detección y recuperación de errores, latencia garantizada y ancho de banda.

Capítulo 2

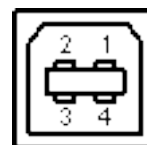
Hardware

2.1. Conectores

Comúnmente hay dos tipos de conectores, llamados tipo A y tipo B, que se muestran a continuación:



Conector A



Conector B

Los conectores Tipo A típicamente se encuentran en anfitriones (hosts) y hubs. Por ejemplo, los conectores Tipo A (hembra) son comunes en tarjetas madre de computadoras y en hubs. Los conectores Tipo B (hembra) se encuentran en dispositivos.

Es interesante notar que se encuentran cables Tipo A a Tipo A directamente cableados y un conjunto de adaptadores USB de género en algunas tiendas de computadoras. Esto va en contradicción de la especificación USB. Los únicos dispositivos con conectores Tipo A (macho) a Tipo A (macho) son puentes que se usan para conectar dos computadoras conjuntamente. Otros cables prohibidos son extensiones USB que tienen un macho a un extremo (ya sea Tipo A o Tipo B) y una hembra en el otro. Estos cables violan los requerimientos de longitud del USB.

El USB 2.0 incluía una errata extra que introduce los conectores B mini-USB. Los detalles de estos conectores pueden encontrarse en la Noticia de Cambio de Ingeniería del Conector Mini-B. El razonamiento detrás de los mini conectores vino del rango de dispositivos electrónicos miniatura tales como teléfonos móviles y agendas electrónicas. El conector actual tipo B es demasiado grande para integrarse fácilmente en estos dispositivos.

Recientemente se ha liberado la especificación On-The-Go la cual agrega funcionalidad punto a punto al USB. Esto introduce los anfitriones USB en teléfonos móviles y agendas electrónicas, y así ha incluido una especificación para machos mini-A, hembras mini-A y hembras mini-AB. Supongo que deberíamos estar inundados con cables mini USB muy pronto y un conjunto de cables convertidores mini y estándar.

Pin N°	Color de cable	Función
1	rojo	VBus (5V)
2	blanco	D-
3	verde	D+
4	negro	GND

Los colores internos estándar de los cables se usan en los cables USB, facilitando la identificación de los alambres de un fabricante a otro. El estándar especifica varios parámetros eléctricos para los cables. Es interesante leer el detalle de la especificación USB 1.0 original incluida. La entendería especificando atributos eléctricos, pero el párrafo 6.3.1.2 sugería que el color recomendado para el recubrimiento de los cables USB debería ser blanco escarcha - ¡qué aburrido! El USB 1.1 y USB 2.0 se relajaron para recomendar Negro, Gris o Natural.

Los diseñadores de circuitos impresos querrán referirse al capítulo 6 para esquemas y asignación de pines.

2.2. Eléctrico

A menos que esté diseñando el silicio para un dispositivo/transceptor o un anfitrión/hub USB, no hay mucho que necesite saber sobre las especificaciones eléctricas en el capítulo 7. Tratamos brevemente los puntos aquí.

Como hemos discutido, el USB usa un par de transmisión diferencial para los datos. Éste está codificado usando NZRI y está relleno con bits para asegurar transiciones adecuadas en el flujo de datos. En dispositivos low y full speed, un 1 diferencial se transmite al poner D+ mayor a 2.8V con una resistencia de 15K ohm puesta a tierra y D- menor a 0.3V con una resistencia de 1.5K ohm puesta a 3.6V. Un 0 diferencial por otro lado es un D- mayor a 2.8V y D+ menor a 0.3V con las mismas resistencias pull up/down.

El receptor define un 1 diferencial como D+ 200mV mayor que D- y un 0 diferencial como D+ 200mV menor que D-. La polaridad de la señal está invertida dependiendo de la velocidad del bus. Por eso los estados J y K se usan para significar los niveles lógicos. En baja velocidad un estado J es un 0 diferencial. En alta velocidad un estado J es un 1 diferencial.

Los transceptores USB tendrán tanto salidas diferenciales y de terminación individual. Ciertos estados de bus se indican por señales de terminación individual en D+, D- o ambos. Por ejemplo un cero terminado individualmente o SE0 puede usarse para significar un reinicio del dispositivo si se mantiene por más de 10mS. Un SE0 se genera al mantener tanto D- como D+ bajos (<0.3V). Las salidas terminadas individualmente y las diferenciales son importantes para notar si está usando un transceptor y FPGA como su dispositivo USB. No puede salirse con la suya y muestrear solo la salida diferencial.

El bus low/full speed tiene una impedancia característica de 90 ohms +/- 15 %. Por eso es importante observar la hoja de especificación cuando se seleccione una impedancia que coincida la serie de resistencias para D+ y D-. Cualquier hoja de especificaciones debería especificar estos valores y tolerancias.

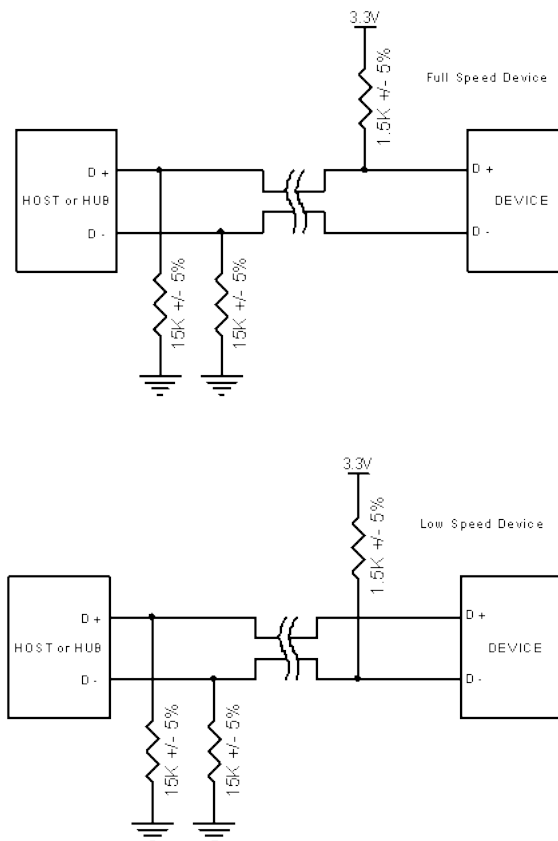
El modo de high speed (480Mbits/s) usa una corriente constante de 17.78mA para la señal para reducir el ruido.

2.3. Indentificación de velocidad

Un dispositivo USB debe indicar su velocidad al poner ya sea la línea D+ o D- en alto a 3.3 voltios. Un dispositivo full speed, ilustrado abajo, usará una resistencia pull up unida a D+ para especificarse a sí mismo como un dispositivo full speed. Estas resistencias pull up al lado del dispositivo también se usarán por el anfitrión o hub para detectar la presencia de un dispositivo conectado a su puerto. Sin una resistencia pull up, el USB asume que no hay nada conectado al bus. Algunos dispositivos tienen esta resistencia integrada en su silicio, la cual puede ser activada o desactivada bajo el control del firmware; otros requieren una resistencia externa.

Por ejemplo Philips Semiconductor tiene una tecnología SoftConnect™. Cuando se conecta por primera vez al bus, esto le permite al microcontrolador inicializar el dispositivo de función USB antes de que active la resistencia pull up de identificación de velocidad, indicando que un dispositivo está conectado al bus. Si la resistencia pull up estuviera conectada a VBUS, entonces esto indicaría un dispositivo que ha sido conectado al bus tan pronto como el enchufe es insertado. El anfitrión puede entonces intentar reiniciar el dispositivo y pedir un descriptor cuando el microprocesador ni siquiera ha comenzado a inicializar el dispositivo de función USB.

Otros vendedores tales como Cypress Semiconductor también usan una resistencia programable para propósitos de Re-Numeración™ en sus dispositivos EzUSB en donde el dispositivo puede ser enumerado para una función tal como programación In Field y luego ser desconectado del bus bajo control del firmware, y enumerarse como un dispositivo diferente, todo sin intervención del usuario y en un pestaño. Muchos de los dispositivos EzUSB no tienen flash ni OTP ROM para almacenar código. Estos son arrancados al tiempo de la conexión.



Notará que no hemos incluido identificación de velocidad para el modo de Alta Velocidad. Los dispositivos high speed comenzarán por conectarse como dispositivos full speed (1.5K a 3.3V). Una vez que se ha conectado, efectuará un "chirrido" de alta velocidad durante el reinicio y establecerá una conexión high speed si el hub lo soporta. Si el dispositivo opera en modo high speed, entonces la resistencia pull up es removida para balancear la línea.

Un dispositivo compatible con el USB 2.0 no está obligado a soportar el modo high speed. Esto permite producir dispositivos más baratos si la velocidad no es crítica. Éste también es el caso para dispositivos USB 1.1 low speed a los cuales no se les requiere soportar el modo full speed.

Sin embargo, un dispositivo high speed no debe soportar el modo low speed. Solo debería soportar el modo full speed necesario para conectarse por primera vez, y luego el modo high speed más tarde si se negocia exitosamente. Un dispositivo compatible con USB 2.0 con conector Tipo A (hembra) (hub o host) debe soportar los tres modos: high speed, full speed y low speed.

2.4. Alimentación (V_{BUS})

Uno de los beneficios del USB son dispositivos alimentados por el bus - dispositivos que obtienen su energía del bus y que no requieren adaptadores externos de enchufe o cables adicionales. Sin embargo, muchos saltan a esta opción sin antes considerar todos los criterios necesarios.

Un dispositivo USB especifica su consumo de energía expresado en unidades de 2mA en el descriptor de configuración que examinaremos más tarde en detalle. Un dispositivo no puede aumentar su consumo de energía, mayor que el que especifica durante la enumeración, aun utilizando alimentación externa. Hay tres clases de funciones USB:

- Funciones de bajo consumo alimentadas por bus
- Funciones de alto consumo alimentadas por bus
- Funciones autoalimentadas

Las funciones de bajo consumo alimentadas por bus toman toda su alimentación desde V_{BUS} y no pueden tomar más de una unidad de carga. La especificación USB define una unidad de carga como 100mA. Las funciones de bajo consumo alimentadas por bus también deben diseñarse para trabajar abajo de un voltaje de V_{BUS} de 4.40V y hasta un voltaje máximo de 5.25 medidos en el conector macho en el extremo

del dispositivo. Para muchos dispositivos de 3.3V, reguladores LDO (Low-dropout o de Salida con Baja Caída) son obligatorios.

Las funciones de alto consumo alimentadas por bus tomarán toda su alimentación del bus y no pueden tomar más de una unidad de carga hasta que haya sido configurada, luego de lo cual puede tomar hasta cinco unidades de carga (500mA Máximo) siempre y cuando lo pida en el descriptor. Las funciones de bus de alto consumo deben ser capaces de ser detectadas y enumeradas a un mínimo de 4.40V. Cuando se opera a una unidad de carga completa, se especifica un V_{BUS} mínimo de 4.75V con un máximo de 5.25V. Una vez más, éstas medidas se toman en el conector macho.

Las funciones autoalimentadas pueden tomar hasta 1 unidad de carga del bus y derivar el resto de su energía de una fuente externa. Si esta fuente externa falla, debe haber medidas para no tomar más de una unidad de carga del bus. Las funciones autoalimentadas son más fáciles de diseñar de acuerdo a la especificación ya que no hay muchos problemas con el consumo de energía. La carga alimentada por bus de 1 unidad permite la detección y enumeración de dispositivos sin alimentación principal/secundaria aplicada.

Ningún dispositivo USB, sea que esté alimentado por el bus o de forma externa, puede suministrar energía a V_{BUS} . Si V_{BUS} es desconectado, el dispositivo tiene un máximo de 10 segundos para cortar la alimentación de las resistencias pull up D+/D- usadas para identificación de velocidad.

Otras consideraciones de V_{BUS} son la corriente de irrupción (Inrush current) la cual debe ser limitada. Esto se indica en la especificación USB párrafo 7.2.4.1 y es comúnmente ignorado. La corriente de irrupción contribuye a la cantidad de capacitancia en su dispositivo entre V_{BUS} y tierra. Por eso la especificación indica que la capacitancia máxima de desacoplamiento (decoupling) que puede tener en su dispositivo es de 10uF. Cuando usted desconecta el dispositivo después de que la corriente está fluyendo a través del cable inductivo USB, puede surgir un voltaje grande de flyback en el extremo abierto del cable. Para prevenir esto, se especifica una capacitancia de desacoplamiento mínima de 1uF para V_{BUS} .

Para el típico dispositivo alimentado por el bus, este no puede consumir más de 500mA lo cual es razonable. ¿Cuál es la complicación, podría preguntarse? El Modo de Suspensión, ¿tal vez?

2.5. Corriente de suspensión

El modo de suspensión es obligatorio para todos los dispositivos. Durante la suspensión entran en efecto restricciones adicionales. La corriente máxima de suspensión es proporcional a la unidad de carga. Para un dispositivo de 1 unidad de carga (lo predeterminado) la corriente máxima de suspensión es 500uA. Esto incluye corriente desde las resistencias pull up en el bus. En el hub, tanto D- como D+ tienen resistencias pull up de 15K ohms. Para propósitos de consumo de energía, la resistencia pull up en el dispositivo está en serie con la pull up de 15K ohms, haciendo un total de 16.5K ohms en un VTERM típicamente de 3.3V. Por eso la corriente es de 200uA antes de que incluso empecemos.

Otra consideración para muchos dispositivos es el regulador de 3.3V. Muchos de los dispositivos USB funcionan con 3.3V. El PDIUSB11 es uno de tales ejemplos. Los reguladores lineales típicamente son muy ineficientes con corrientes de inactividad (quiescent) en el orden de los 600uA, y por eso se necesitan reguladores más eficientes y caros. En la mayoría de los casos, también debe ralentizar o detener relojes en los microcontroladores para caer dentro del límite de los 500uA.

Muchos desarrolladores preguntan en el Foro de Implementadores del USB: "¿cuáles son las complicaciones de exceder este límite?". Se entiende que la mayoría de anfitriones y hubs no tienen la capacidad de detectar una sobrecarga de semejante magnitud y por eso si consume 5mA o incluso 10mA todo debería estar bien, teniendo en mente que al final del día, su dispositivo viola la especificación USB. Sin embargo, en operación normal, si trata de exceder los 100mA o su carga permitida designada, entonces se espera que el anfitrión o hub lo detecte y desconecte su dispositivo, a favor de la integridad del bus.

Por supuesto que estos problemas de diseño se pueden evitar si elige diseñar un dispositivo autoalimentado. Las corrientes de suspensión pueden no ser una gran preocupación para computadoras de escritorio, pero con la introducción de la especificación On-The-Go, comenzaremos a ver anfitriones USB construidos en teléfonos móviles y agendas electrónicas. El consumo energético que se obtiene de estos dispositivos afectará adversamente la vida operativa de la batería.

2.6. Entrando al modo de suspensión

Un dispositivo USB entrará en suspensión cuando no hay actividad en el bus por más de 3.0ms. Entonces tiene unos 7ms adicionales para desactivar el dispositivo y consumir no más de la corriente de suspensión designada y así, debe sólo estar consumiendo la corriente de suspensión ya fijada del bus, 10mS después de que la actividad del bus se ha detenido. A fin de mantenerse conectado a un hub o

anfitrión suspendidos, el dispositivo aún debe proveer energía a su resistencia pull up de selección de velocidad durante la suspensión.

El USB tiene un paquete de inicio de frame o "keep alive" (mantener con vida) enviado periódicamente en el bus. Esto evita que un bus sin actividad entre en modo de suspensión en ausencia de datos.

- Un bus high speed tendrá micro frames enviados cada $125.0\ \mu\text{s} \pm 62.5\ \text{ns}$.
- Un bus full speed tendrá un frame enviado cada $1.000\ \text{ms} \pm 500\ \text{ns}$.
- Un bus low speed tendrá un "keep alive" el cual es un EOP (End of Packet, o Fin de Paquete) cada 1ms solo en la ausencia de datos de bajo nivel.

El término "suspensión global" se usa cuando el bus USB entero entra en modo de suspensión colectivamente. Sin embargo, se pueden suspender dispositivos selectivamente enviando un comando al hub al que el dispositivo está conectado. A esto se le llama "Suspensión Selectiva".

El dispositivo reanudará operaciones cuando reciba cualquier señal que no sea de inactividad. Si un dispositivo tiene despertador remoto activado (remote wakeup) entonces este puede señalarle al anfitrión que salga de la suspensión.

2.7. Tasa de señal de datos

Otra área que es a menudo pasada por alto es la tolerancia de los relojes USB. Esto se especifica en la especificación USB, sección 7.1.11.

- Los datos high speed tienen un reloj a 480.00Mb/s con una tolerancia de señal de datos de $\pm 500\text{ppm}$ (pulse position modulation, o modulación de posición de pulso).
- Los datos full speed tienen un reloj a 12.000Mb/s con una tolerancia de señal de datos de $\pm 0.25\%$ o $2,500\text{ppm}$.
- Los datos low speed tienen un reloj a 1.50Mb/s con una tolerancia de señal de datos de $\pm 1.5\%$ o $15,000\text{ppm}$.

Esto permite usar resonadores para dispositivos de bajo costo y low speed, pero los excluye para dispositivos full o high speed.

Capítulo 3

Protocolos USB

A diferencia del RS-232 e interfaces seriales similares, en las que el formato de los datos enviados no está definido, el USB está conformado de varias capas de protocolos. Si bien esto suena complicado, no se rinda ahora. Una vez que entienda lo que está pasando, realmente solo tiene que preocuparse de las capas de mayor nivel. De hecho, la mayoría de circuitos integrados del USB se encargarán de la capa más baja, haciéndola así casi invisible al diseñador final.

Cada transacción USB consiste de un:

- Paquete de Token (cabecera que define lo que se espera que siga)
- Paquete Opcional de Datos (que contiene el payload o "carga útil")
- Paquete de Estado (usado para reconocer transacciones y proporcionar un método de corrección de errores)

Como ya hemos discutido anteriormente, el USB es un bus céntrico a un anfitrión. El anfitrión (host) inicia todas las transacciones. El primer paquete, también llamado token, es generado por el anfitrión para describir lo que sigue y si la transacción de datos será una lectura o una escritura, y cuál es la dirección del dispositivo y el endpoint designado. El siguiente paquete generalmente es un paquete de datos que lleva el payload (carga útil) y es seguido por un paquete de saludo (handshaking), reportando si los datos o el token fueron recibidos exitosamente, o si el punto final (endpoint) está atascado o no está disponible para aceptar datos.

3.1. Campos comunes de paquetes USB

Los datos en el bus USB transmiten primero el bit LSB (el de menor peso o Least Significant Bit). Los paquetes USB consisten de los siguientes campos:

- **SYNC**
Todos los paquetes deben comenzar con un campo Sync. El campo Sync tiene 8 bits de longitud en low o full speed, o 32 bits de longitud para high speed y se usa para sincronizar el reloj del receptor con el del transmisor. Los últimos dos bits indican dónde comienzan los campos PID.
- **PID**
PID significa Packet ID, o IDentificación de Paquete. Este campo se usa para identificar el tipo de paquete que se está enviando.

Hay 4 bits para PID; sin embargo para asegurar que se reciba correctamente, los 4 bits son complementados y repetidos, haciendo un PID de 8 bits en total. El formato resultante se muestra a continuación:

PID0	PID1	PID2	PID3	nPID0	nPID1	nPID2	nPID3
------	------	------	------	-------	-------	-------	-------

La siguiente tabla muestra los valores posibles:

Grupo	Valor PID	Identificador del Paquete
Token	0001	Token OUT
	1001	Token IN
	0101	Token SOF
	1101	Token Setup
Datos	0011	DATA0
	1011	DATA1
	0111	DATA2
	1111	MDATA
Handshake	0010	Handshake ACK
	1010	Handshake NAK
	1110	Handshake STALL
	0110	NYET (No Response Yet, aún sin respuesta)
Especial	1100	PREambulo
	1100	ERR
	1000	Split (partir)
	0100	Ping

■ **ADDR**

El campo de dirección especifica para cuál dispositivo está designado el paquete. Teniendo una longitud de 7 bits, permite soportar 127 dispositivos. La dirección 0 no es válida, ya que cada dispositivo que todavía no tiene una dirección asignada debe responder a los paquetes enviados a la dirección 0.

■ **ENDP**

El campo de punto final (endpoint) está conformado de 4 bits, permitiendo 16 posibles puntos finales. Los dispositivos de baja velocidad, sin embargo, solo pueden tener 2 puntos finales adicionales en la parte superior de la tubería (pipe) por defecto (4 puntos finales como máximo).

■ **CRC**

Se efectúan Chequeos de Redundancia Cíclica en los datos dentro de la carga útil (payload) del paquete. Todos los paquetes de token tienen un CRC de 5 bits mientras que los paquetes de datos tienen un CRC de 16 bits.

■ **EOP**

Fin del Paquete (End Of Packet). Señalado por un Cero Terminado Individualmente (Single Ended Zero o SE0 por aproximadamente el tiempo de 2 bits seguido por una J por el tiempo de un bit).

3.2. Tipos de paquetes

El USB tiene 4 tipos diferentes de paquetes. Los paquetes de Token indican el tipo de la transacción a seguir, los paquetes de saludo (handshake) se usan para reconocer (acknowledge) datos o reportar errores; y los paquetes de inicio de marco (frame) indican el inicio de un nuevo marco (cuadro o frame).

■ **Paquetes de token**

Hay tres tipos de paquetes de token:

- **IN (entrada)** - informan al dispositivo USB que el anfitrión (host) desea leer información
- **OUT (salida)** - informan al dispositivo USB que el anfitrión (host) desea enviar información
- **Setup** - Usado para iniciar transferencias de control

Los paquetes de token deben apegarse al siguiente formato:

Sync	PID	ADDR	ENDP	CRC5	EOP
------	-----	------	------	------	-----

- **Paquetes de datos**

Hay dos tipos de paquetes de datos, cada uno capaz de transmitir hasta 1024 bytes de datos.

- **Data0**

- **Data1**

El modo de alta velocidad define otros dos PIDs de datos, DATA2 y MDATA.

Los paquetes de datos tienen el siguiente formato:

Sync	PID	Data	CRC16	EOP
------	-----	------	-------	-----

- El tamaño máximo del payload para dispositivos low speed es 8 bytes.
- El tamaño máximo del payload para dispositivos full speed es 1023 bytes.
- El tamaño máximo del payload para dispositivos high speed es 1024 bytes.
- Los datos deben enviarse en múltiplos de bytes.

- **Paquetes de saludo (Handshake)**

Hay tres tipos de paquetes de handshake que consisten simplemente del PID:

- **ACK** - Reconocimiento de que el paquete ha sido recibido exitosamente.
- **NAK** - Reporta que el dispositivo no puede temporalmente enviar o recibir datos. También usado durante transacciones de interrupción para informarle al anfitrión que no hay datos que enviar.
- **STALL** - El dispositivo encuentra que está en un estado que requiere intervención del anfitrión.
- **NYET** - Sólo dispositivos high speed. Es devuelto por un endpoint como parte del protocolo PING o puede ser devuelto por un hub cuando una transacción dividida no fue completada.

Los paquetes de handshake tienen el siguiente formato:

SYNC	PID	EOP
------	-----	-----

- **Paquetes de inicio de frame (Marco o cuadro)**

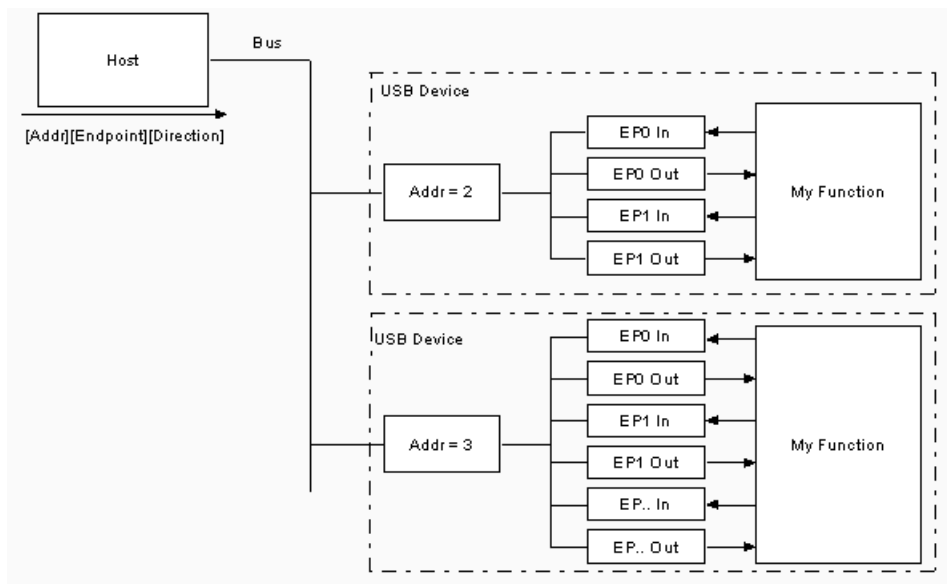
El paquete SOF (Start of Frame), que consiste de un número de frame de 11 bits, se envía por el host cada $1\text{ms} \pm 500\text{ns}$ en un bus full speed o cada $125\mu\text{s} \pm 0.0625\mu\text{s}$ en un bus high speed.

Sync	PID	Número de Frame	CRC5	EOP
------	-----	-----------------	------	-----

3.3. Funciones USB

Cuando pensamos en un dispositivo USB, pensamos en un periférico USB; pero un dispositivo USB podría significar un dispositivo transceptor USB usado en el lado anfitrión o periférico, un Hub USB o circuito integrado dispositivo Controlador Host, o un dispositivo periférico USB. Por eso el estándar hace referencia a las funciones USB las cuales pueden ser vistas como dispositivos USB las cuales ofrecen una capacidad o función tales como una Impresora, Drive Zip, Escáner, Módem u otro periférico.

Así que a estas alturas deberíamos saber el tipo de cosas que conforman un paquete USB. ¿No es así? ¿Ya olvidó cuántos bits conforman un campo PID? Bueno, no se alarme mucho. Afortunadamente la mayoría de funciones USB manejan los protocolos de bajo nivel del USB hasta la capa de transacción (la cual cubriremos en el siguiente capítulo) en el silicio. La razón por la que cubrimos esta información es porque la mayoría de controladores de función USB reportarán errores tales como un Error de Codificación de PID. Sin cubrir esto brevemente, uno podría preguntar: ¿Qué es un Error de Codificación PID? Si usted sugiriera que los últimos cuatro bits del PID no concordaban al inverso de los primeros cuatro bits entonces estaría en lo correcto.



La mayoría de funciones tendrán una serie de buffers, típicamente de 8 bytes. Cada buffer pertenecerá a un punto final (endpoint) - EP0 IN, EP0 OUT, etc. Digamos por ejemplo que el anfitrión (host) envía una petición de descriptor de dispositivo. El hardware de la función leerá el paquete de setup (configuración) y determinará a partir del campo de dirección (address) si el paquete va por sí mismo, y si es así copiará la carga útil (payload) del siguiente paquete de datos hacia el buffer del endpoint apropiado dictado por el valor en el campo de endpoint del token de configuración (setup). Entonces enviará un paquete handshake (saludo) para reconocer la recepción del byte y generará una interrupción interna dentro del semiconductor/microcontrolador para el endpoint apropiado señalando que ha recibido un paquete. Típicamente todo esto se hace en hardware.

El software ahora recibe una interrupción, y debería leer el contenido del buffer del endpoint y escanear (parse) la petición de descriptor de dispositivo.

3.4. Terminales (endpoints)

Las terminales pueden describirse como fuentes (source) o "sumideros" (sinks) de datos. Ya que el bus es céntrico a un anfitrión, las terminales ocurren al final del canal de comunicación, en la función USB. En la capa de software, su driver de dispositivo puede enviar un paquete al EP1 de su dispositivo por ejemplo. A medida que los datos fluyen fuera del anfitrión, estos terminarán en el buffer EP1 OUT. Su firmware entonces, en su tiempo "libre" y sin esforzarse, leerá estos datos. Si desea devolver datos, la función no puede simplemente escribir al bus ya que el bus es controlado por el anfitrión. Por eso escribe datos en EP1 IN los cuales permanecen en el buffer hasta el momento en el que el anfitrión envía un paquete IN a esa terminal pidiendo datos. Las terminales también pueden verse como la interfaz entre el hardware del dispositivo de función y el firmware corriendo en el dispositivo de función.

Todos los dispositivos deben soportar la terminal cero (endpoint zero). Esta es la terminal que recibe todas las peticiones de control de dispositivo y de estado durante la enumeración y a través y durante el tiempo en el que el dispositivo es operacional en el bus.

3.5. Tuberías (pipes)

Mientras que el dispositivo envía y recibe datos en una serie de puntos finales (endpoints), el software cliente transfiere datos a través de tuberías (pipes). Una tubería es una conexión lógica entre el anfitrión y el o los puntos finales. Las tuberías también tienen un conjunto de parámetros asociados con estos, tales como cuánto ancho de banda se ha reservado para éste, qué tipo de transferencia usa (Control, Bulk/Bulk, Isócrona o Interrupción), una dirección de flujo de datos y tamaños máximos de paquete/buffer. Por ejemplo la tubería por defecto es una tubería bidireccional hecha del punto final cero in (endpoint zero in) y el punto final cero out (endpoint zero out) con un tipo de transferencia de control.

El USB define dos tipos de tuberías:

- **Tuberías de flujo (stream)**

No tienen un formato USB definido; eso significa que puede enviar cualquier tipo de datos por una tubería de flujo y puede recuperar los datos en el otro extremo. Los datos fluyen secuencialmente y

tienen una dirección predefinida, ya sea dentro (in) o fuera (out). Las tuberías de flujo soportarán tipos de transferencia de bulto, isócronas y de interrupción. Las tuberías de flujo pueden controlarse ya sea por el anfitrión o el dispositivo.

- ***Tuberías de mensajes***

Tienen un formato USB definido. Son controladas por el anfitrión, y las cuales son iniciadas por una petición enviada desde el anfitrión. Los datos son luego transferidos en la dirección deseada, dictada por la petición. Por eso, las tuberías de mensajes permiten a los datos fluir en ambas direcciones pero solo soportarán transferencias de control.

Capítulo 4

Tipos de EndPoints

La Especificación USB define cuatro tipos de transferencia/endpoints (puntos finales):

- Transferencias de Control
- Transferencias de Interrupción
- Transferencias Isócronas
- Transferencias de Bulto (Bulk)

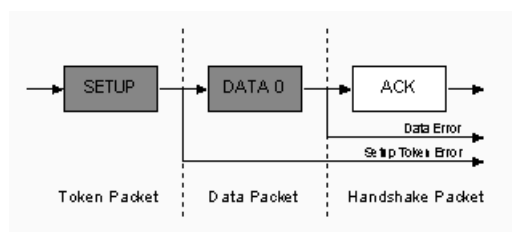
4.1. Transferencias de Control

Las transferencias de control típicamente se usan para operaciones de comandos y estado. Son esenciales para configurar un dispositivo USB con todas las funciones de enumeración, siendo efectuadas usando transferencias de control. Típicamente son paquetes de ráfagas aleatorias que se inician por el anfitrión y usan el mejor esfuerzo de entrega. La longitud de paquete de las transferencias de control en dispositivos low speed debe ser de 8 bytes; los dispositivos full speed permiten un tamaño de paquete de 8, 16, 32 o 64 bytes; y los dispositivos high speed deben tener un tamaño de paquete de 64 bytes.

Una transferencia de control puede tener hasta tres etapas:

- ***Etapas de configuración (setup)***

es cuando se envía la petición. Ésta consiste de tres paquetes. El token de configuración (setup) se envía primero el cual contiene la dirección y el número del endpoint. El paquete de datos se envía a continuación y siempre tiene un tipo PID de data0 e incluye un paquete de configuración el cual detalla el tipo de petición. Detallaremos el paquete de configuración (setup) más adelante. El último paquete es un saludo (handshake) usado para reconocer (acknowledge) la recepción exitosa o indicar un error. Si la función recibe exitosamente los datos de configuración (setup) - CRC y PID, etc. - esta responde con ACK; de lo contrario ignora los datos y no envía un paquete de saludo (handshake). Las funciones no pueden emitir un paquete STALL o NAK en respuesta a un paquete de configuración.

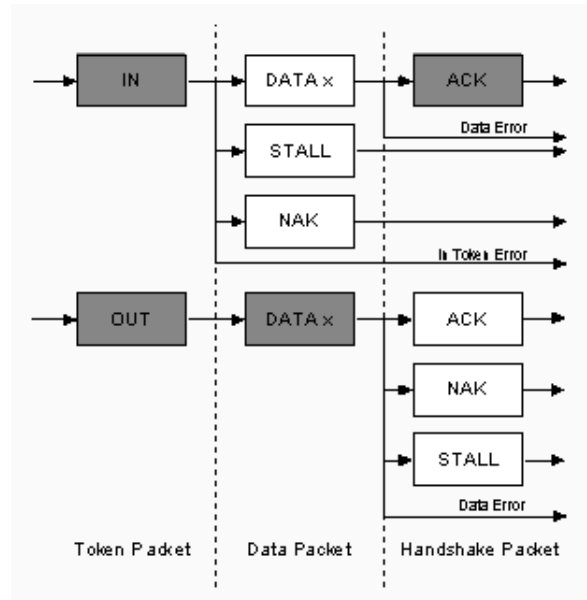


- ***Etapas opcionales de datos***

consiste de una o múltiples transferencias IN u OUT. La petición de configuración (setup) indica la cantidad de datos a ser transmitidos en esta etapa. Si excede el tamaño máximo del paquete, los datos se enviarán en múltiples transferencias, cada vez de la longitud máxima del paquete excepto por el último paquete.

La etapa de datos tiene dos escenarios diferentes dependiendo de la dirección de la transferencia de datos.

- **IN** - Cuando el anfitrión está listo para recibir datos de control emite un token IN. Si la función emite el token IN con un error, por ejemplo el PID no concuerda con los bits PID invertidos, entonces ignora el paquete. Si el token se recibió correctamente, el dispositivo puede ya sea responder con un paquete DATA que contiene los datos de control a enviarse, un paquete de atascamiento (stall) que indica que el endpoint ha tenido un error, o un paquete NAK que le indica al anfitrión que el punto final (endpoint) está trabajando, pero temporalmente no tiene datos que enviar.

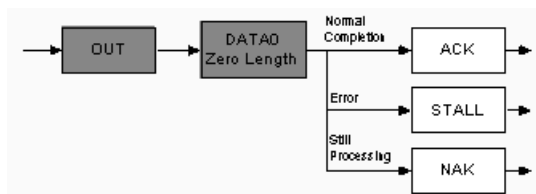


- **OUT** - Cuando el anfitrión necesita enviarle un paquete de datos de control al dispositivo, emite un token OUT seguido por un paquete de datos que contiene los datos de control como carga útil (payload). Si cualquier parte del token o paquete de datos OUT está corrompido entonces la función ignora el paquete. Si el buffer del punto final (endpoint) de la función estaba vacío y ha marcado los ciclos de reloj para los datos en el buffer del endpoint entonces emite un ACK informándole al anfitrión que ha recibido los datos con éxito. Si el buffer del endpoint no está vacío debido al procesamiento del paquete anterior, entonces la función devuelve un NAK. Sin embargo, si el endpoint ha tenido un error y su bit de detener (halt) ha sido activado, entonces devuelve un STALL (atasque).

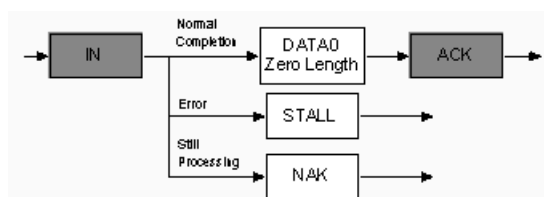
■ **Etapas de estado**

Reporta el estado total (overall) de la petición y esto una vez más varía de acuerdo a la dirección de la transferencia. El reporte del estado siempre es efectuado por la función.

- **IN** - Si el host envió tokens IN durante la etapa de datos para recibir datos, entonces el anfitrión debe reconocer (acknowledge) la recepción exitosa de los datos. Esto se hace por el host enviando un token OUT seguido por un paquete de datos con longitud cero. La función puede ahora reportar su estado en la etapa del saludo (handshaking). Un ACK indica que la función que ha completado el comando está ahora lista para aceptar otro comando. Si un error ocurriera durante el procesamiento de este comando, entonces la función emitirá un STALL. Sin embargo si la función todavía está procesando, devuelve un NAK indicando al host que repita la etapa de estado más adelante.



- **OUT** - Si el anfitrión envió uno o varios tokens OUT durante la etapa de datos para transmitir datos, la función reconocerá la recepción exitosa de los datos enviando un paquete con longitud cero en respuesta a un token IN. Sin embargo, si un error ocurrió, debería emitir un STALL o si todavía está procesando datos, debería emitir un NAK pidiéndole al anfitrión reintentar la etapa de estado más adelante.



4.2. Transferencias de control, un panorama más amplio

Ahora, ¿cómo queda esto aplicado? Digamos por ejemplo, que el Anfitrión desea pedir un descriptor de dispositivo durante la enumeración. Los paquetes que se envían se muestran a continuación.

El anfitrión enviará el token de Configuración (Setup) diciéndole a la función que el siguiente paquete es el paquete de Configuración (Setup). El campo Dirección (Address) mantendrá la dirección del dispositivo del que el anfitrión está pidiendo el descriptor. El número de punto final (endpoint) debería ser cero, especificando la tubería por defecto (default pipe). El anfitrión entonces enviará un paquete DATA0. Éste tendrá una carga útil (payload) de 8 bits la cual es la Petición de Descriptor de Dispositivo como se delinea en el Capítulo 9 de la Especificación USB. La función USB entonces reconoce (acknowledge) que el paquete de configuración (setup) ha sido leído correctamente sin errores. Si el paquete fue recibido corrompido, el dispositivo solo ignora este paquete. El anfitrión entonces reenviará el paquete después de un corto lapso.

1 Token de configuración	Sync	PID	ADDR	ENDP	CRC5	EOP	Dirección y número de endpoint
2 Paquete de Data0	Sync	PID	Data0		CRC16	EOP	Petición de descriptor de dispositivo
3 Handshake ACK	Sync	PID	EOP				Paquete Ack de configuración de dispositivo

Los tres paquetes anteriores representan la primer transacción USB. El dispositivo USB ahora decodificará los 3 paquetes recibidos, y determinará si fue una petición de descriptor de dispositivo. El dispositivo entonces intentará enviar el Descriptor de Dispositivo, el cual será la siguiente transacción USB.

1 Token in	Sync	PID	ADDR	ENDP	CRC5	EOP	Dirección y número de endpoint
2 Paquete Data1	Sync	PID	Data1		CRC16	EOP	Primeros 8 bytes de descriptor de dispositivo
3 Handshake ACK	Sync	PID	EOP				El host reconoce el paquete
4 Token in	Sync	PID	ADDR	ENDP	CRC5		Dirección y número de endpoint
5 Paquete Data0	Sync	PID	Data0		CRC16	EOP	Últimos 4 bytes mas relleno
6 Handshake ACK	Sync	PID	EOP				El host reconoce el paquete

En este caso asumimos que la carga útil máxima (payload) es de 8 bytes. El anfitrión (host) envía el token IN, diciéndole al dispositivo que ahora puede enviar datos a su endpoint. Ya que el tamaño máximo de paquete es 8 bytes, debemos partir el descriptor de dispositivo en porciones (chunks) de 8 bytes para enviarlo. Cada porción debe ser de 8 bytes excepto por la última transacción. El anfitrión reconoce (acknowledge) cada paquete que le enviamos.

Una vez que el descriptor de dispositivo es enviado, sigue una transacción de estado. Si las transacciones tuvieron éxito, el anfitrión enviará un paquete con longitud cero indicando que la transacción total (overall) tuvo éxito. La función entonces responde a este paquete con longitud cero indicando su estado.

1 Token Out	Sync	PID	ADDR	ENDP	CRC5	EOP	Dirección y número de endpoint
2 Paquete Data1	Sync	PID	Data1		CRC16	EOP	Paquete de longitud cero
3 Handshake ACK	Sync	PID	EOP				El dispositivo reconoce la transacción entera

4.3. Transferencias de interrupción

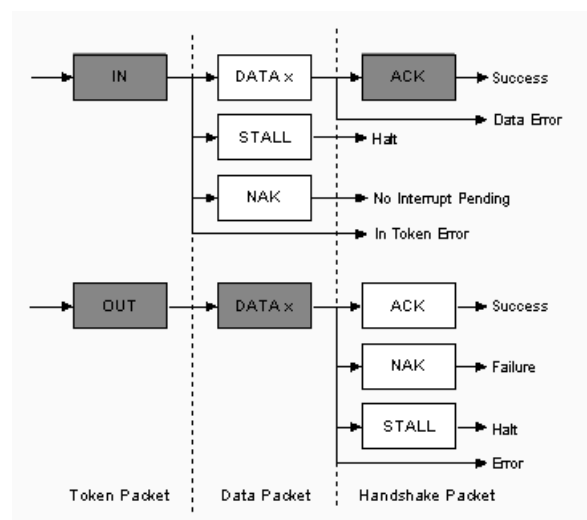
Cualquiera que haya tenido experiencia con peticiones de interrupción en microcontroladores sabrá que las interrupciones se generan por el dispositivo. Sin embargo bajo el USB si un dispositivo requiere la atención del anfitrión, ¡debe esperar hasta que el anfitrión vea manualmente su estado (poll) antes de poder reportar que necesita atención urgente!

Características

- Latencia garantizada
- Tubería de flujo unidireccional
- Detección de error y reintento en el próximo período

Las transferencias de interrupción típicamente comunicaciones pequeñas no periódicas, "iniciadas" por el dispositivo, que requieren latencia delimitada (bounded). Una petición de Interrupción se pone en cola por el dispositivo hasta que el anfitrión ve manualmente el estado (poll) del dispositivo USB, pidiendo datos.

- El tamaño máximo del payload para dispositivos low speed es de 8 bytes.
- El tamaño máximo del payload para dispositivos full speed es de 64 bytes.
- El tamaño máximo del payload para dispositivos high speed es de 1024 bytes.



El diagrama anterior muestra el formato de una transacción de Interrupción IN y OUT.

- **IN** - El anfitrión periódicamente evaluará (poll) el endpoint de interrupción. Esta tasa de evaluación manual (polling) se especifica en el descriptor de endpoint el cual se cubre más adelante. Cada evaluación manual (poll) implica que el anfitrión envíe un token IN. Si el token IN está corrompido, la función ignora el paquete y continúa monitoreando el bus por más tokens nuevos. Si una interrupción ha sido puesta en cola por el dispositivo, la función enviará un paquete de datos que contiene datos relevantes a la interrupción cuando recibe el token IN. Una vez que se recibe con éxito en el anfitrión, el anfitrión devolverá un ACK. Sin embargo, si los datos están corrompidos, el anfitrión no devolverá estado. Si por el contrario una condición de interrupción no estaba presente cuando el anfitrión evaluó manualmente (poll) el endpoint de interrupción con un token IN, entonces la función señala este estado enviando NAK. Si un error ha ocurrido en este endpoint, un STALL se envía en su lugar en respuesta al token IN.
- **OUT** - Cuando el anfitrión desea enviar datos de interrupción al dispositivo, emite un token OUT seguido por un paquete de datos que contiene los datos de interrupción. Si cualquier parte del token OUT está corrompido entonces la función ignora el paquete. Si el buffer del endpoint de la función estaba vacío y ha marcado ciclos de reloj para los datos al buffer del endpoint, este emite un ACK informándole al anfitrión que ha recibido con éxito los datos. Si el buffer del endpoint no está vacío debido al procesamiento de un paquete anterior, entonces la función devuelve NAK. Sin embargo si ocurrió un error con el endpoint y consecuentemente su bit de detener (halt) ha sido activado, este devuelve un STALL.

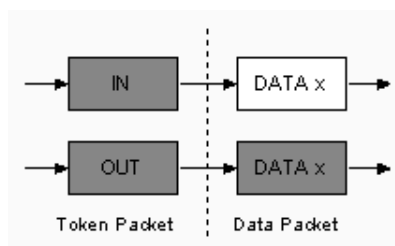
4.4. Transferencias Isócronas

Las transferencias isócronas ocurren continua y periódicamente. Típicamente contienen información sensible al tiempo, tal como una secuencia de audio o video. Si hubiera una demora o reintento de datos en una secuencia de audio, entonces esperaría audio errático que contiene fallas (glitches). El ritmo (beat) puede ya no estar en sincronía. Sin embargo, si un paquete o marco (frame) fue descartado de vez en cuando, es menos probable que el escucha lo note.

Las transferencias isócronas proporcionan:

- Acceso garantizado al ancho de banda USB
- Latencia delimitada (bounded)
- Tubería de datos unidireccional
- Detección de datos vía CRC, pero sin reintento o garantía de entrega
- Solo modos full y high speed
- Sin cambio en los datos (toggling)

El tamaño máximo de la carga útil (payload) se especifica en el descriptor del endpoint de un Endpoint Isócrono. Este puede ser de un máximo de 1023 bytes para un dispositivo full speed y 1024 bytes para un dispositivo high speed. Ya que el tamaño máximo de los datos del payload va a afectar los requerimientos de ancho de banda del bus, es sabio especificar un tamaño de payload conservador. Si está usando un payload grande, puede también serle ventajoso especificar una serie de interfaces alternativas con tamaños de payload isócronos diferentes. Si durante la enumeración el anfitrión no puede activar su endpoint isócrono preferido debido a restricciones de ancho de banda, tiene algo hacia lo cual degradarse (fall back) en lugar de solo fallar por completo. Los datos que se envían en un endpoint isócrono pueden ser menos que el tamaño negociado previamente y pueden variar en longitud de una transacción a otra.



4.5. Transferencias de bulto (bulk)

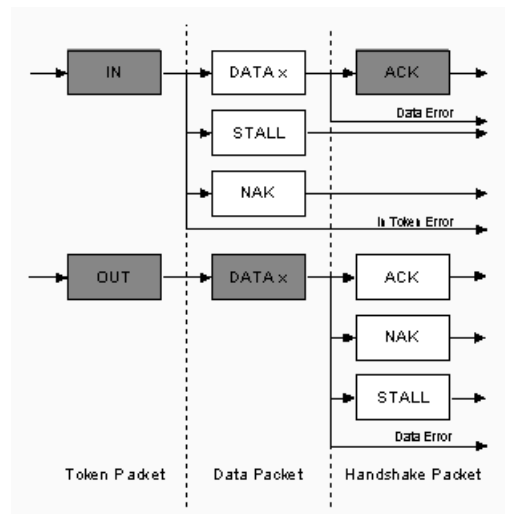
Las transferencias de bulto pueden usarse para grandes ráfagas de datos. Ejemplos de tales transferencias podrían incluir un trabajo de impresión enviado a una impresora o una imagen generada por un escáner. Las transferencias de bulto ofrecen corrección de errores en la forma de un campo CRC16 en el payload de datos y mecanismos de detección de errores/retransmisión asegurando que los datos se transmiten y reciben sin errores.

Las transferencias de bulto usarán ancho de banda sobrante sin asignar (spare) en el bus después de que todas las otras transacciones han sido asignadas. Si el bus está ocupado con transferencias isócronas o de interrupción entonces los datos de bulto pueden lentamente hacer que el bus trabaje limitadamente por "goteo" (trickle). Como resultado las transferencias de Bulto solo deberían usarse para comunicación no sensible al tiempo ya que no hay garantía de latencia.

Características de las transferencias de bulto

- Usadas para transferir grandes ráfagas de datos
- Detección de errores vía CRC, con garantía de entrega
- Sin garantía de ancho de banda o latencia mínima
- Tuberías de datos unidireccional
- Solo modos full y high speed

Las transferencias de bulto solo son soportadas por dispositivos full y high speed. Para los endpoints full speed, el tamaño máximo del paquete de bulto puede ser de 8, 16, 32 o 64 bytes. Para los endpoints high speed, el tamaño máximo de paquete puede ser hasta de 512 bytes. Si el payload de datos no usa el tamaño máximo de paquete, este no necesita rellenarse con ceros. Una transferencia de bulto se considera completa cuando ha transferido la cantidad exacta de datos solicitados, cuando ha transferido un paquete menor que el tamaño máximo del endpoint, o cuando ha transferido un paquete con longitud cero.



El diagrama anterior muestra el formato de una transacción de bulto IN y OUT.

- **IN** - Cuando el anfitrión está listo para recibir datos de bulto, emite un token IN. Si la función recibe el token IN con un error, ignora el paquete. Si el token se recibió correctamente, la función puede ya sea responder con un paquete DATA que contiene los datos de bulto a ser enviados, o un paquete de atasco (stall) que indica que el endpoint ha tenido un error o un paquete NAK que le indica al anfitrión que el endpoint está funcionando, pero temporalmente no tiene datos que enviar.
- **OUT** - Cuando el anfitrión quiere enviar a la función un paquete de datos de bulto, emite un token OUT seguido por un paquete de datos que contiene los datos de bulto. Si cualquier parte del token OUT o paquete de datos está corrompido entonces la función ignora el paquete. Si el buffer del endpoint de la función estaba vacío y ha contado ciclos de reloj para los datos al buffer del endpoint, emite un ACK informándole al anfitrión que ha recibido los datos con éxito. Si el buffer del endpoint no está vacío debido al procesamiento de un paquete anterior, entonces la función devuelve un NAK. Sin embargo, si el endpoint ha tenido un error y su bit de detener (halt) ha sido activado, devuelve STALL (atasque).

4.6. Administración de ancho de banda

El anfitrión es responsable de administrar el ancho de banda del bus. Esto se hace en la enumeración cuando se configuran Endpoints Isócronos y de Interrupción y a través (durante) la operación del bus. La especificación impone límites en el bus, permitiendo no más del 90 % de cualquier marco (frame) a ser asignado para transferencias periódicas (Interrupción e Isócrona) en un bus full speed. En buses high speed esta limitación se reduce a que no más del 80 % de un micro marco (micro frame) pueda asignarse a transferencias periódicas.

Así que puede rápidamente ver que si tiene un bus altamente con transferencias periódicas, el 10 % restante se deja para transferencias de control y una vez que esas han sido asignadas, las transferencias de bulto obtendrán su rebanada de lo que quede.

Capítulo 5

Descriptorios USB

Todos los dispositivos USB tienen una jerarquía de descriptorios que describe al anfitrión información tal como qué es el dispositivo, quién lo fabrica, qué versión de USB soporta, de cuántas formas puede configurarse, el número de endpoints y sus tipos, etc.

Los descriptorios USB más comunes son:

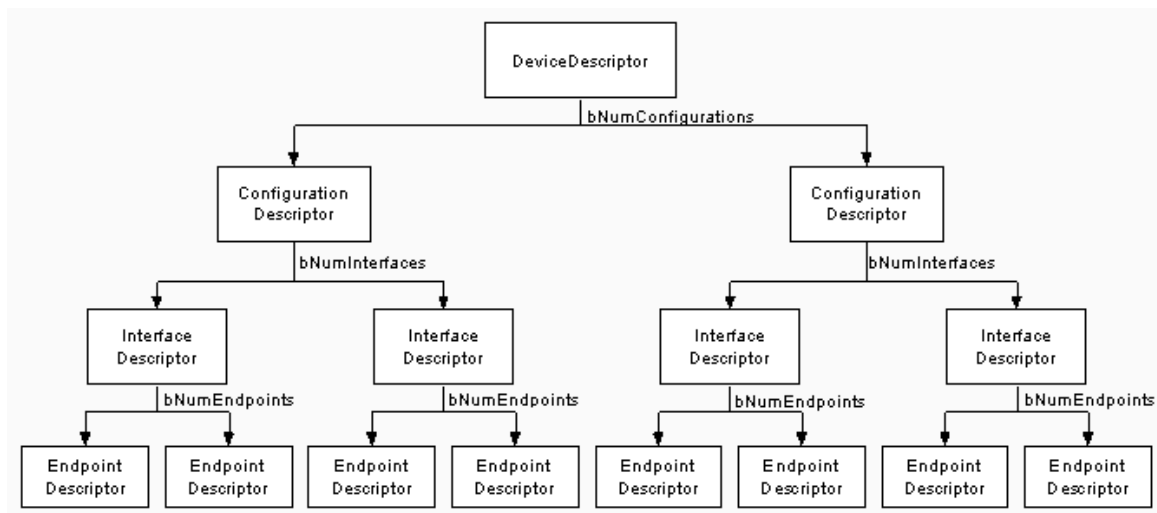
- Descriptorios de dispositivo
- Descriptorios de configuración
- Descriptorios de interfaz
- Descriptorios de endpoint
- Descriptorios de cadena (string)

Los dispositivos USB solamente pueden tener un descriptor de dispositivo. El descriptor de dispositivo incluye información tal como a qué revisión de USB se apegó el dispositivo, las IDs de Producto y Vendedor usadas para cargar los drivers apropiados y el número de configuraciones posibles que el dispositivo puede tener. El número de configuraciones indica cuántas ramas de descriptorios de configuración hay que seguir.

El descriptor de configuración especifica valores tales como la cantidad de energía que usa esta configuración particular, si el dispositivo es autoalimentado o alimentado por el bus, y el número de interfaces que tiene. Cuando se enumera un dispositivo, el anfitrión lee los descriptorios del dispositivo y puede hacer una decisión sobre qué configuración habilitar. Solo puede habilitar una configuración a la vez.

Por ejemplo es posible tener una configuración de alto consumo alimentada por el bus y una configuración autoalimentada. Si el dispositivo está conectado a un anfitrión con una fuente principal, el driver de dispositivo puede escoger activar la configuración de alto consumo alimentada por el bus permitiendo al dispositivo ser alimentado sin una conexión externa, pero si está conectada a una laptop o agenda electrónica podría habilitar la segunda configuración (autoalimentado) lo que requeriría que el usuario conecte su dispositivo de forma externa.

Las opciones de configuración no están limitadas a diferencias de alimentación. Cada configuración podría alimentarse de la misma forma y consumir la misma corriente, y aún así tener combinaciones de interfaz o endpoint diferentes. Sin embargo debería notarse que cambiar la configuración requiere que toda actividad en cada endpoint sea detenida. Si bien el USB ofrece esta flexibilidad, muy pocos dispositivos tienen más de una configuración.

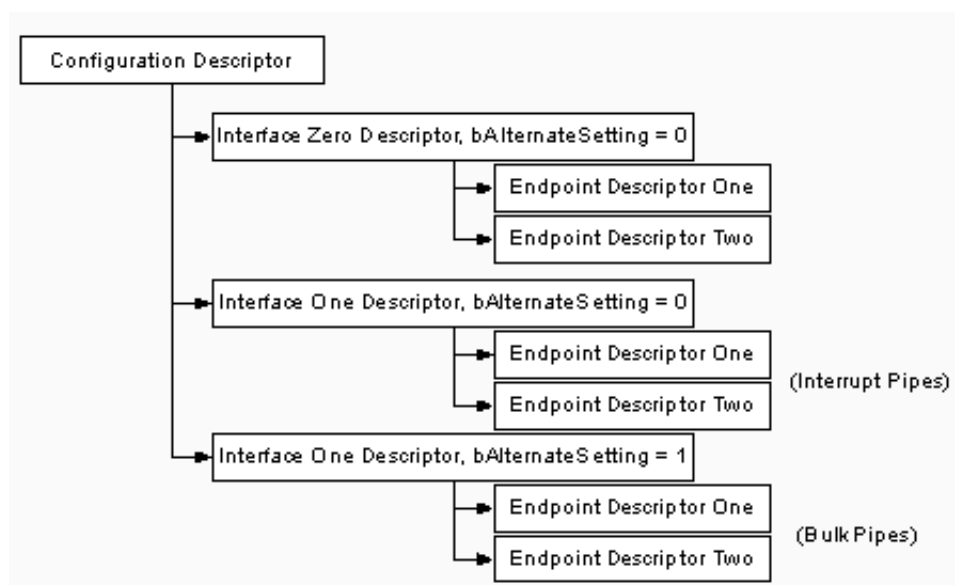


El descriptor de interfaz podría ser visto como una cabecera o agrupación de los endpoints en un grupo funcional que lleva a cabo una única característica del dispositivo. Por ejemplo podría tener un dispositivo multifunción con fax/escáner/impresora. El descriptor de interfaz 1 podría describir los endpoints de la función de fax, el descriptor de interfaz 2 la función de escáner, y el descriptor de interfaz 3 la función de impresora. A diferencia del descriptor de configuración, no hay limitaciones de tener solo una interfaz habilitada a la vez. Un dispositivo podría tener 1 o muchos descriptors de interfaz habilitados a la vez.

Los descriptors de interfaz tienen un campo `bInterfaceNumber` que especifica el número de interfaz y un `bAlternativeSetting` que permite a una interfaz cambiar configuraciones sobre la marcha (on the fly). Por ejemplo podríamos tener un dispositivo con dos interfaces, interfaz 1 e interfaz 2. La interfaz 1 tiene `bInterfaceNumber` a cero indicando que es el primer descriptor de interfaz, y un `bAlternativeSetting` de cero.

La interfaz 2 tendría un `bInterfaceNumber` a uno indicando que es la segunda interfaz y un `bAlternativeSetting` de cero (por defecto). Podríamos entonces incluir otro descriptor, también con un `bInterfaceNumber` a uno indicando que es la segunda interfaz, pero esta vez estableciendo `bAlternativeSetting` a uno, indicando que este descriptor de interfaz puede ser una configuración alternativa al otro descriptor de interfaz dos.

Cuando se activa esta configuración, los dos primeros descriptors de interfaz con `bAlternativeSetting` igual a cero se usan. Sin embargo, durante la operación el host puede enviar una petición `SetInterface` dirigida a la de la Interfaz 1 con una configuración alternativa de 1 para activar el otro descriptor de interfaz.



Esto da una ventaja por sobre tener dos configuraciones, en que podemos estar transmitiendo datos sobre la interfaz 0 mientras cambiamos las configuraciones de endpoint asociadas con la interfaz 1 sin afectar la interfaz 0.

Cada descriptor de endpoint se usa para especificar un tipo de transferencia, dirección, intervalo de monitoreo manual (polling) y tamaño máximo de paquete para cada endpoint. El endpoint 0, el endpoint de control por defecto, siempre se asume que es un endpoint de control y como tal nunca tiene un descriptor.

5.1. Composición de los descriptors USB

Todos los descriptors tienen un formato común. El primer byte especifica la longitud del descriptor, mientras que el segundo byte indica el tipo del descriptor. Si la longitud del descriptor es menor a lo que define la especificación, entonces el anfitrión ha de ignorarla. Sin embargo, si el tamaño es mayor de lo esperado, el anfitrión ignorará los bytes extra y comenzará a buscar el siguiente descriptor al final de la longitud devuelta en cuestión.

Offse	Campo	Tamaño	Valor	Descripción
0	bLength	1	Número	Tamaño del descriptor en bytes
1	bDescriptorType	1	Constante	Inicio de parámetros para el descriptor
2	...	n	...	Inicio de parámetros para el descriptor

5.2. Descriptores de dispositivo

El descriptor de dispositivo de un dispositivo USB representa el dispositivo entero. Como resultado, un dispositivo USB solo puede tener un descriptor de dispositivo. Este especifica alguna información básica pero importante sobre el dispositivo tal como la versión de USB soportada, el tamaño máximo de paquete, IDs de vendedor y de producto, y el número de configuraciones posibles que el dispositivo puede tener. El formato del descriptor de dispositivo se muestra a continuación.

Offset	Campo	Tamaño	Valor	Descripción
0	bLength	1	Número	Tamaño del descriptor en bytes (18)
1	bDescriptorType	1	Constante	Descripción de dispositivo (0x01)
2	bcdUSB	2	BCD	Número de especificación USB al que el dispositivo se apeg
4	bDeviceClass	1	Clase	Código de clase asignado por USB.org Si es 0, cada interfaz especifica su propio código Si es 0xFF, el vendedor especifica el código De lo contrario el campo es un código de clase invalido
5	bDeviceSubClass	1	Subclase	Código de subclase asignado por USB.org
6	bDeviceProtocol	1	Protocolo	Código de protocolo asignado por USB.org
7	bMaxPacketSize0	1	Número	Tamaño máximo de paquete para el EndPoint 0 Los tamaños válidos son 8, 16, 32 y 64 bytes
8	idVendor	2	ID	ID de vendedor asignada por USB.org
10	idProduct	2	ID	ID de producto asignada por USB.org
12	bcdDevice	2	BCD	Número de versión (release) del dispositivo
14	iManufacturer	1	Indice	Indice del descriptor de cadena de caracteres del fabricante
15	iProduct	1	Indice	Indice del descriptor de cadena de caracteres del producto
16	iSerialNumber	1	Indice	Indice del descriptor del número de serie
17	bNumConfigurations	1	Entero	Número de configuraciones posibles

- El campo bcdUSB reporta la versión más alta de USB que soporta el dispositivo. El valor está en decimal codificado en binario (BCD) con un formato de 0xJJMN en donde JJ es la versión mayor, M es el número menor de versión y N es el número de sub-versión menor. Por ejemplo, el USB 2.0 se reporta como 0x200, USB 1.1 como 0x0110 y USB 1.0 como 0x0100.

- Los campos bDeviceClass, bDeviceSubClass y bDeviceProtocol se usan por el sistema operativo para encontrar un driver de clase para su dispositivo. Típicamente solo bDeviceClass está establecido a nivel de dispositivo. La mayoría de especificaciones de clase eligen identificarse a sí mismas al nivel de la interfaz y como resultado establecen bDeviceClass a 0x00. Esto permite que el mismo dispositivo individual soporte múltiples clases.
- El campo bMaxPacketSize reporta el tamaño máximo del paquete para el endpoint cero. Todos los dispositivos deben soportar el endpoint cero.
- Los campos idVendor e idProduct se usan por el sistema operativo para encontrar un driver para su dispositivo. El ID de Vendedor se asigna por el Foro de Implementadores de USB (USB Implementors Forum o USB-IF).
- El campo bcdDevice tiene el mismo formato que bcdUSB y se usa para proporcionar un número de versión del dispositivo. Este valor es asignado por el desarrollador.
- Existen tres descriptores de cadenas de caracteres para proporcionar detalles del fabricante, el producto y el número de serie. No hay un requerimiento sobre tener descriptores de cadenas de caracteres. Si no hay ningún descriptor de cadenas presente, debería usarse un índice de cero.
- bNumConfigurations define el número de configuraciones que el dispositivo soporta en su velocidad actual.

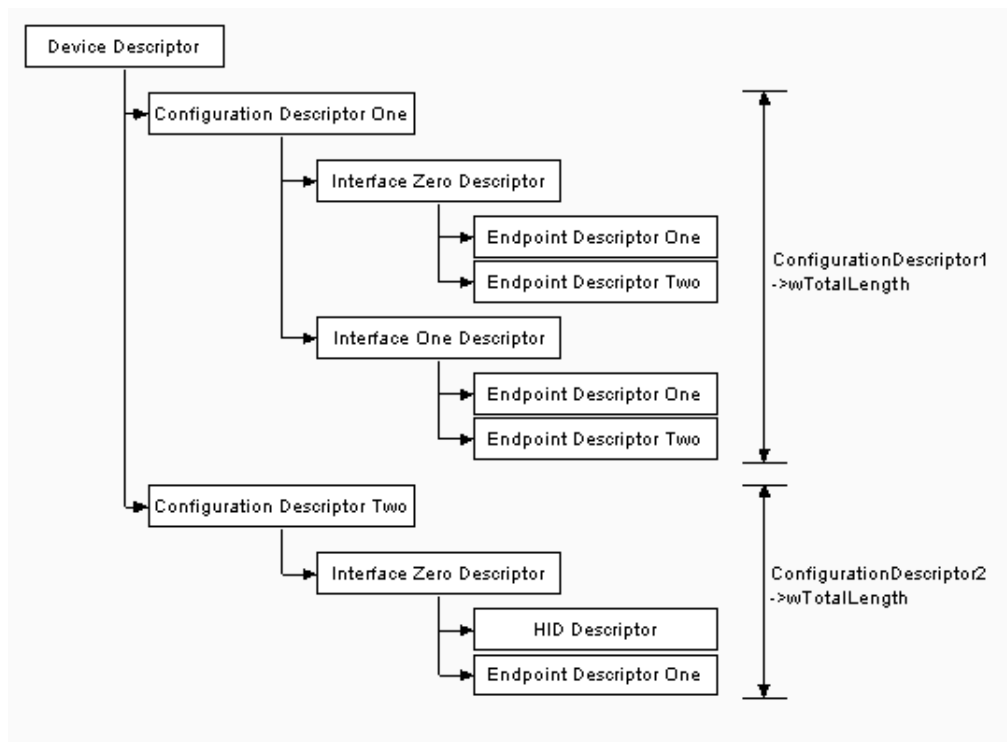
5.3. Descriptores de configuración

Un dispositivo USB puede tener varias configuraciones diferentes aunque la mayoría de dispositivos son simples y solo tienen una. El descriptor de configuración especifica cómo se alimenta el dispositivo, cuál es el consumo máximo de energía, y el número de interfaces que tiene. Por eso es posible tener dos configuraciones: una para cuando el dispositivo esté alimentado por el bus y otra cuando está alimentado de forma externa. Ya que esta es una "cabecera" a los descriptores de Interfaz, también es factible tener una configuración que use un modo de transferencia diferente al de otra configuración.

Una vez que todas las configuraciones han sido examinadas por el anfitrión, el anfitrión enviará un comando SetConfiguration con un valor diferente de cero que coincida con el bConfigurationValue de una de las configuraciones. Este se usa para elegir la configuración deseada.

Offset	Campo	Tamaño	Valor	Descripción
0	bLength	1	Número	Tamaño del descriptor en bytes
1	bDescriptorType	1	Constante	Descriptor de configuración (0x02)
2	wTotalLength	2	Número	Longitud total en bytes de los datos devueltos
4	bNumInterfaces	1	Número	Número de interfaces
5	bConfigurationValue	1	Número	Valor a usar como argumento para seleccionar esta configuración
6	iConfiguration	1	Índice	Índice del descriptor de cadena de caracteres que describe esta configuración
7	bmAttributes	1	Mapa de bits	D7 - Reservado, valor 1 (USB 1.0 alimentado por Bus) D6 - Autoalimentado D5 - Despertado remoto (Remote Wakeup) D4..D0 - Reservado, valor 0
8	bMaxPower	1	mA	Consumo máximo de energía en unidades de 2mA

- Cuando se lee el descriptor de configuración, devuelve la jerarquía de configuración entera la cual incluye todos los descriptores de interfaz y endpoint. El campo wTotalLength refleja el número de bytes en la jerarquía.



- **bNumInterfaces** especifica el número de interfaces presentes para esta configuración.
- **bConfigurationValue** es usado por la petición **SetConfiguration** para elegir esta configuración.
- **iConfiguration** es un índice a un descriptor de cadena de caracteres que describe la configuración en forma humanamente legible.
- **bmAttributes** especifica parámetros de energía para la configuración. Si un dispositivo es autoalimentado, establece **D6**. El bit **D7** fue usado en el USB 1.0 para indicar un dispositivo alimentado por el bus, pero esto es hecho por **bMaxPower** ahora. Si un dispositivo usa cualquier cantidad de energía del bus, ya sea un dispositivo alimentado por el bus o autoalimentado, debe reportar su consumo de energía en **bMaxPower**. Los dispositivos también pueden soportar despertado remoto (remote wakeup) el cual le permite al dispositivo despertar el anfitrión cuando se encuentra suspendido.
- **bMaxPower** define el consumo energético máximo que el dispositivo tomará del bus. Este es en unidades de 2mA, así que se puede especificar un máximo aproximado de 500mA. La especificación le permite a un dispositivo de alto consumo alimentado por el bus consumir no más de 500mA desde VBUS. Si un dispositivo pierde la alimentación externa, entonces no debe consumir más de lo indicado en **bMaxPower**. Cualquier operación que no puede llevar a cabo sin energía externa debería fallar.

5.4. Descriptores de Interfaz

El descriptor de interfaz podría verse como una cabecera o agrupación de los endpoints dentro de un grupo funcional que lleva a cabo una única característica del dispositivo. El descriptor de interfaz se apega al siguiente formato:

Offset	Campo	Tamaño	Valor	Descripción
0	bLength	1	Número	Tamaño del descriptor en bytes (9)
1	bDescriptorType	1	Constante	Descriptor de interfaz (0x04)
2	bInterfaceNumber	1	Número	Número de interfaz
3	bAlternateSetting	1	Número	Valor usado para elegir una configuración alternativa
4	bNumEndpoints	1	Número	Número de endpoints usados por esta interfaz
5	bInterfaceClass	1	Clase	Código de clase, asignado por USB.org
6	bInterfaceSubClass	1	SubClase	Código de subclase, asignado por USB.org
7	bInterfaceProtocol	1	Protocolo	Código de protocolo, asignado por USB.org
8	iInterface	1	Indice	Indice del descriptor de cadena de caracteres que describe esta interfaz

- bInterfaceNumber indica el índice del descriptor de interfaz. Este debería estar basado en cero e incrementado una vez por cada nuevo descriptor de interfaz.
- bAlternateSetting puede usarse para especificar interfaces alternativas. Estas interfaces alternativas pueden seleccionarse con la petición SetInterface.
- bNumEndpoints indica el número de endpoints usados por la interfaz. Este valor debería excluir el endpoint cero y se usa para indicar el número de descriptores de endpoint a seguir.
- bInterfaceClass, bInterfaceSubClass y bInterfaceProtocol pueden usarse para especificar clases soportadas (por ejemplo HID, comunicaciones, almacenamieto masivo, etc.). Esto le permite a muchos dispositivos usar drivers de clase evitando la necesidad de escribir drivers específicos para su dispositivo.
- iInterface permite una descripción de cadena de caracteres para la interfaz.

5.5. Descriptores de Endpoint

Los descriptores de endpoint se usan para describir endpoints aparte del endpoint cero. El endpoint cero siempre se asume que es un endpoint de control y se configura antes de siquiera se solicite cualquier descriptor. El anfitrión usa la información devuelta desde estos descriptores para determina los requerimientos de ancho de banda del bus.

Offset	Campo	Tamaño	Valor	Descripción
0	bLength	1	Número	Tamaño del descriptor en byte (7)
1	bDescriptorType	1	Constante	Descriptor de Endpoint (0x05)
2	bEndpointAddress	1	Endpoint	Dirección del Endpoint Bits 3..0 - Número de Endpoint 6..4 - Reservado 7 - Dirección (0 = Out, 1 = in) ignorado para Endpoint de control
3	bmAttributes	1	Mapa de Bits	Bits 1..0 - Tipo de transferencia 00b - Control 01b - Isócrono 10b - Bulto 11b - Interrupción 5..2 - Deben ser 0 si no es un Endpoint isócrono 3..2 - Tipo de sincronización (modo isócrono) 00b - Sin sincronización 01b - Asíncrona 10b - Adaptiva 11b - Síncrona 5..4 - Tipo de uso (modo isócrono) 00b - Endpoint de datos 01b - Endpoint de feedback (retroalimentación) 10b - Endpoint explícito de datos de feedback 11b - Reservado
4	wMaxPacketSize	2	Número	Tamaño máximo del paquete que este Endpoint es capaz de enviar o recibir
6	bInterval	1	Número	Intervalo para monitorear manualmente (polling) las transferencias de datos del Endpoint. El valor está en cuenta de marcos (frames). Ignorado para Endpoint de bultos y control. La isócrona debe ser igual a 1 y el campo puede estar en el rango de 1 a 255 para Endpoint de interrupción

- bEndpointAddress indica qué endpoint está describiendo este descriptor.
- bmAttributes especifica el tipo de transferencia. Este puede ser ya sea de transferencias de Control, Interrupción, Isócronas o de Bulto. Si se especifica un endpoint isócrono, se pueden seleccionar atributos adicionales tales como los tipos de Sincronización y Uso.
- wMaxPacketSize indica el tamaño máximo de payload (carga útil) para este endpoint.
- bInterval se usa para especificar el intervalo de monitoreo manual (polling) de ciertas transferencias. Las unidades se expresan en marcos (frames), así que esto es igual ya sea a 1ms para dispositivos low o full speed y 125us para dispositivos high speed.

5.6. Descriptores de Cadena (String)

Los descriptores de cadena ofrecen información humanamente legible y son operacionales. Si no se usan, cualesquiera campos de índice de cadena de caracteres de los descriptores deben estar a cero indicando que no hay descriptor de cadena de caracteres disponible.

Las cadenas están codificadas en el formato Unicode y se pueden hacer productos que soporten múltiples idiomas. El Índice de Cadena 0 debería devolver una lista de idiomas soportados. Una lista de IDs de Idioma puede encontrarse en los Identificadores de Idioma del Bus Serial Universal o Universal Serial Bus Language Identifiers (LANGIDs) version 1.0.

Offset	Campo	Tamaño	Valor	Descripción
0	bLength	1	Número	Tamaño del descriptor en bytes
1	bDescriptorType	1	Constante	Descriptor de cadena (0x03)
2	wLANGID[0]	2	Número	Código 0 del lenguaje soportado ej (0x0409 es ingles - EE UU)
4	wLANGID[1]	2	Número	Código 1 del lenguaje soportado ej (0x0C09 es ingles - Australia)
6	wLANGID[2]	2	Número	Código 2 del lenguaje soportado ej (0x0407 es Alemán - estandar)

El Descriptor de Cadena de Caracteres muestra el formato del Descriptor Cero de Cadena de Caracteres. El anfitrión debería leer este descriptor para determinar qué idiomas están disponibles. Si se soporta un idioma. Si ese es el caso, entonces puede referenciarse al enviar el ID del idioma en el campo wIndex de una petición GetDescriptor(String).

Todas las cadenas de caracteres subsecuentes toman el formato siguiente:

Offset	Campo	Tamaño	Valor	Descripción
0	bLength	1	Número	Tamaño del descriptor en bytes
1	bDescriptorType	1	Constante	Descriptor de cadena de caracteres (0x03)
2	bString	n	Unicode	Cadena codificada en unicode

Capítulo 6

Peticiones USB

6.1. El paquete de configuración (Setup)

Cada dispositivo USB debe responder a paquetes de configuración (setup) en la tubería por defecto. Los paquetes de configuración se usan para la detección y configuración del dispositivo y llevan a cabo funciones comunes tales como establecer la dirección del dispositivo USB, solicitando un descriptor de dispositivo o verificando el estado de un endpoint.

Un anfitrión compatible con el USB espera que todas las peticiones sean procesadas dentro de un período máximo de 5 segundos. También especifica una temporización más estricta para peticiones específicas:

- Peticiones estándar de dispositivo sin una etapa de datos deben completarse en 50ms.
- Peticiones estándar de dispositivo con una etapa de datos deben comenzar a devolver datos 500ms después de la petición.
- Cada paquete de datos debe enviarse dentro de 500ms de la transmisión exitosa del paquete anterior.
- La etapa de estado debe completarse dentro de 500ms después de la transmisión del último paquete de datos.
- El comando SetAddress (que contiene una fase de datos) debe procesar el comando y devolver el estado dentro de 50ms. El dispositivo entonces tiene 2ms para cambiar la dirección antes de que la siguiente petición sea enviada.

Estos períodos de timeout son aceptables incluso para los dispositivos más lentos, pero pueden ser una restricción durante la depuración. 50ms no permite que se envíen muchos caracteres a 9600bps en un puerto serial asíncrono o para un Depurador/Emulador en el circuito (In Circuit) para efectuar la operación paso a paso (single step) o detener la ejecución (break) para examinar los registros internos. Como resultado, el USB requiere algunos métodos de depuración diferentes a los de otros proyectos de microcontroladores.

Casualmente leyendo a través del DDK de XP, uno puede notar que el Driver de Controlador Host ahora tiene un comando `USBUSER_OP_SEND_ONE_PACKET` con un comentario que dice "Esta API se usa para implementar el 'single step' de la herramienta de desarrollo de transacciones USB". Si bien dicha herramienta todavía no ha sido publicada, solo podemos esperar a verla pronto.

Cada petición comienza con un Paquete de Configuración (Setup) largo que tiene el siguiente formato:

Offset	Campo	Tamaño	Valor	Descripción
0	bmRequestType	1	Mapa de bits	D7 - Dirección de transferencia de la fase de datos 0 - Host a dispositivo 1 - Dispositivo a host D6..D5 - Tipo 00b - Estandar 01b - Clase 10b - Vendedor 11b - Reservado D4..D0 - Recipiente 00000b - Dispositivo 00001b - Interfaz 00010b - Endpoint 00011b - Otro 00100b..11111b - Reservado
1	bRequest	1	Valor	Petición
2	wValue	2	Valor	Valor
4	wIndex	2	Indice	Indice
6	wLength	2	Cuenta	Número de bytes a transferir si hay una fase de datos

El campo `bmRequestType` determinará la dirección de la petición, el tipo de la petición y el receptor designado. El campo `bRequestField` determina la petición que se está haciendo. `bmRequestType` normalmente se escanea (parse) y la ejecución se ramifica a un número de manejadores (handlers) tales como el manejador de petición estándar de dispositivo, un manejador de petición estándar de interfaz de dispositivo, un manejador de petición estándar de endpoint, un manejador de petición estándar de clase de dispositivo, etc. El modo en el que escanee el paquete de configuración (setup) depende totalmente de usted. Otros podrían elegir escanear (parse) `bRequest` primero y luego determinar el tipo y el receptor basado en cada petición.

Las peticiones estándar son comunes a todos los dispositivos USB y se detallan en las siguientes páginas. Las peticiones de clase son comunes a clases de drivers. Por ejemplo, todo dispositivo que se apegue a la clase HID tendrá un conjunto común de peticiones de clase específicas. Estas serán diferentes a las de un dispositivo que se apegue a la clase de comunicaciones y será nuevamente diferente a la de un dispositivo que se apegue a la clase de almacenamiento masivo.

Y la última de todas son las peticiones, definida por el vendedor. Estas son peticiones que usted como diseñador de dispositivo USB puede asignar. Normalmente son diferentes entre un dispositivo y otro, pero depende totalmente de su implementación y su imaginación.

Una petición común puede ser direccionada a diferentes receptores y basados en el receptor, efectuar funciones diferentes. Una petición estándar `GetStatus` por ejemplo, puede dirigirse al dispositivo, interfaz o endpoint. Cuando se dirige a un dispositivo, este devuelve banderas indicando el estado del despertado remoto (remote wakeup) y si el dispositivo es autoalimentado. Sin embargo si la misma petición se direcciona a la interfaz esta siempre devuelve cero, o si se dirigiera a un endpoint, devolverá la bandera de detener (halt) para el endpoint.

Los campos `wValue` y `wIndex` permiten a los parámetros ser pasados con la petición. `wLength` se usa para especificar el número de bytes a ser transferidos, de haber una fase de datos.

6.2. Peticiones estándar

La sección 9.4 de la especificación USB detalla las "peticiones estándar de dispositivo" que se requiere implementar para todo dispositivo USB. El estándar proporciona una única tabla agrupando los elementos por petición. Considerando que la mayoría de firmware escaneará (parse) el paquete de configuración (setup) por receptor, optaremos por descomponer las peticiones basándonos en el receptor para un examen e implementación más fáciles.

6.2.1. Peticiones estándar de dispositivo

Actualmente hay ocho peticiones estándar de dispositivo, todas las cuales se detallan en la siguiente tabla.

bmRequestType	bRequest	wValue	wIndex	wLength	Datos
1000 0000b	GET_STATUS (0x00)	0	0	2	Estado del dispositivo
0000 0000b	CLEAR_FEATURE (0x01)	Selector de Característica	0	0	Nada
0000 0000b	SET_FEATURE (0x03)	Selector de Característica	0	0	Nada
0000 0000b	SET_ADDRESS (0x05)	Dirección del dispositivo	0	0	Nada
1000 0000b	GET_DESCRIPTOR (0x06)	Tipo de Descriptor e índice	0 o ID de idioma	Longitud de Descriptor	Descriptor
0000 0000b	SET_DESCRIPTOR (0x07)	Tipo de Descriptor e índice	0 o ID de idioma	Longitud de Descriptor	Descriptor
1000 0000b	GET_CONFIGURATION (0x08)	0	0	1	Valor de Configuración
0000 0000b	SET_CONFIGURATION (0x09)	Valor de Configuración	0	0	Nada

- La petición GetStatus dirigida hacia el dispositivo devolverá 2 bytes durante la etapa de datos con el siguiente formato:

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Reservado														Despertado remoto	Autoalimentado

Si D0 es 1, entonces eso indica que el dispositivo es autoalimentado. Si es 0, el dispositivo tiene activado el despertado remoto y puede despertar al anfitrión durante la suspensión. El bit de despertado remoto puede configurarse por las peticiones SetFeature y ClearFeature con un selector de característica de DEVICE_REMOTE_WAKEUP (0x01)

- Las peticiones ClearFeature y SetFeature pueden usarse para establecer características booleanas. Cuando el receptor designado es el dispositivo, los únicos dos selectores de característica disponibles son DEVICE_REMOTE_WAKEUP y TEST_MODE. El modo de prueba le permite al dispositivo exhibir varias condiciones. Estas se documentan adicionalmente en la Especificación USB Revisión 2.0.
- SetAddress se usa durante la enumeración para asignar una única dirección al dispositivo USB. La dirección se especifica en wValue y solo puede ser un máximo de 127. Esta petición es única en que el dispositivo no establece su dirección sino hasta después de completar la etapa de estado (ver Transferencias de Control). Todas las otras peticiones deben completarse antes de la etapa de estado.
- SetDescriptor/GetDescriptor se usa para devolver el descriptor especificado en wValue. Una petición para el descriptor de configuración devolverá el descriptor de dispositivo y todos los descriptors de interfaz y endpoint en una sola petición.
 - Los Descriptores de Endpoint no pueden accederse directamente por una petición GetDescriptor/SetDescriptor.
 - Los Descriptores de Interfaz no pueden accederse directamente por una petición GetDescriptor/SetDescriptor.
 - Los Descriptores de Cadena de Caracteres incluyen un ID de idioma en wIndex para permitir el soporte de múltiples idiomas.
- GetConfiguration/SetConfiguration se usa para solicitar o establecer la configuración actual del dispositivo. En el caso de una petición GetConfiguration, se devolverá un byte durante la etapa

de datos indicando el estado del dispositivo. Un valor de cero significa que el dispositivo no está configurado y un valor diferente de cero indica que el dispositivo está configurado. SetConfiguration se usa para habilitar un dispositivo. Este debería contener el valor de bConfigurationValue del descriptor de configuración deseado en el byte de menor peso de wValue para elegir qué configuración activar.

6.2.2. Peticiones estándar de Interfaz

La especificación actualmente define cinco peticiones estándar de interfaz las cuales se detallan en la siguiente tabla. Es muy interesante notar que solo 2 peticiones hacen algo inteligible.

bmRequestType	bRequest	wValue	wIndex	wLength	Datos
1000 0001b	GET_STATUS (0x00)	0	Interfaz	2	Estado de la interfaz
0000 0001b	CLEAR_FEATURE (0x01)	Selector de característica	Interfaz	0	Nada
0000 0001b	SET_FEATURE (0x03)	Selector de característica	Interfaz	0	Nada
1000 0001b	GET_INTERFACE (0x0A)	0	Interfaz	1	Interfaz alterna
0000 0001b	SET_INTERFACE (0x0A)	Configuración alterna	Interfaz	0	Nada

- wIndex normalmente se usa para especificar la interfaz referida para peticiones dirigidas a la interfaz. Su formato se muestra a continuación:

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Reservado								Autoalimentado							

- GetStatus se usa para devolver el estado de la interfaz. Tal petición a la interfaz debería devolver 2 bytes 0x00 0x00 (ambos bytes están reservados para uso futuro).
- Las peticiones ClearFeature y SetFeature pueden usarse para establecer características booleanas. Cuando el recipiente designado es la interfaz, la Especificación USB Revisión 2 actual no especifica características de interfaz.
- GetInterface y SetInterface establecen la configuración de Interfaz Alternativa la cual se describe en mayor detalle bajo el Descriptor de Interfaz.

6.2.3. Peticiones estándar de Endpoint

Las peticiones estándar de Endpoint vienen en cuatro variedades como se listan a continuación.

bmRequestType	bRequest	wValue	wIndex	wLength	Datos
1000 0010b	GET_STATUS (0x00)	0	Endpoint	2	Estado del Endpoint
0000 0010b	CLEAR_FEATURE (0x01)	Selector de característica	Endpoint	0	Nada
0000 0010b	SET_FEATURE (0x03)	Selector de característica	Endpoint	0	Nada
1000 0010b	SYNCH_FRAME (0x12)	0	Endpoint	2	Número de Frame

- wIndex normalmente se usa para especificar el endpoint referido y la dirección para peticiones dirigidas a un endpoint. Su formato se muestra a continuación:

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Reservado								Dir	Reservado			Número de Endpoint			

- GetStatus devuelve 2 bytes que indican el estado (detenido/atascado, o halted/stalled) de un endpoint. El formato de los dos bytes devueltos se ilustra a continuación.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Reservado															Halt

- Las peticiones ClearFeature y SetFeature se usan para establecer características de endpoint. El estándar actualmente define un selector de característica de endpoint, ENDPOINT_HALT (0x00) que le permite al host atascar (stall) y limpiar un endpoint. Solo se recomienda que los endpoints aparte del endpoint por defecto tengan esta funcionalidad.
- Una petición de Synch Frame (marco de sincronización) se usa para reportar un marco de sincronización de endpoint.