Resumen teórico USB

USB

Universal Serial Bus, es una interfaz plug & play entre un HOST (PC por ejemplo) y otros dispositivos.
Es una interfaz de intercambio de datos y distribución de energía. Esta consta de 4 hilos (VBUS, +D, -D, GND) y distribuye una tensión de +5V y una corriente máxima (para el estandar 2.0) de 500 mA.

¿Como funciona?

Es un BUS basado en el paso de un testigo tal como se utiliza en otros tipos de BUS. El controlador USB (HOST) distribuye testigos por el BUS, de esta manera el dispositivo cuya dirección coincide con la que porta el testigo responde aceptando o enviando datos al controlador. Es el HOST quien gestiona la distribución de la energía de los periféricos que lo requieran.
Cuenta con una topología de estrella apilada, esto permite el funcionamiento de hasta 127 dispositivos en simultaneo.
Existe un HOST principal en la punta de la pirámide que es quien centraliza toda la actividad del BUS. Además este tipo de topología permite que muchos dispositivos se conecten a un único BUS LÓGICO y sin que los dispositivos que están más abajo en la pirámide sufran retardos respecto de sus superiores.
En todo esquema general del BUS USB pueden encontrarse solo tres partes:

1. El controlador (HOST)
2. Los Hubs o concentradores
3. Los periféricos

El controlador

Para el caso mas genera y habitual reside dentro de la PC. Es el responsable de la comunicación entre los periféricos USB y la CPU. Es quien se encarga de la detección de conexión y desconexión de periféricos del BUS. Para cada uno conectado, el HOST debe determinar su tipo y asignarle una dirección lógica para sus futuras comunicaciones. El controlador es quien suministra al dispositivo los recursos del sistema que este precisa.

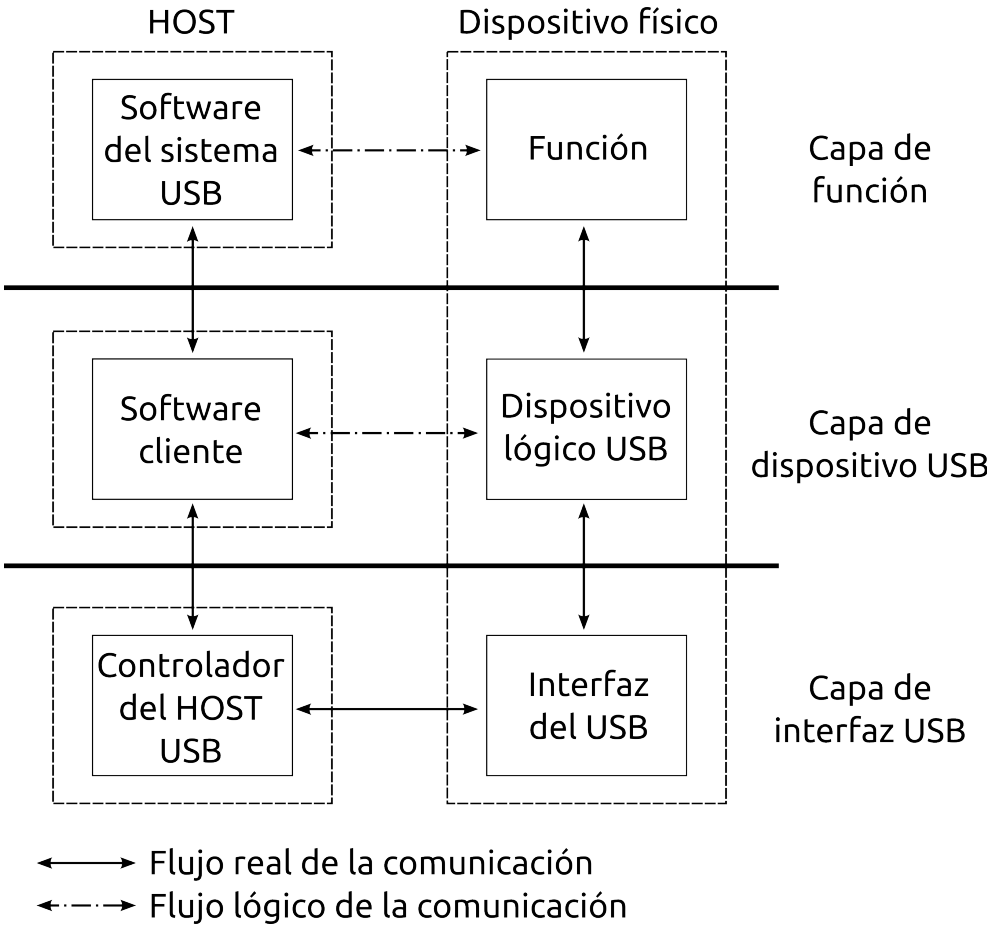
Los Hubs

Son distribuidores inteligentes de datos y alimentación. Los reparten de manera selectiva hacia sus puertas descendentes (hacia los periféricos) o a la inversa a través de su puerta ascendente (hacia el PC, HOST).
Cabe destacar que el controlador o HOST es un Hub/concentrador y se denomina RAÍZ, es el primero de toda la cadena.

Los periféricos

Existen de varios tipos y se pueden clasificar según su categoría de velocidad de trabajo. Algunos como teclados y mouse no requieren ungran uso de datos por lo que utilizan una velocidad de 1,5 Mbps liberando el resto del BUS para otros dispositivos con mayor demanda como monitores, escaners, audio, etc.

Diagrama de capas



Descriptores

Son estructuras de datos que le permiten al HOST aprender sobre el dispositivo. Cada descriptor contiene información del dispositivo como un todo o sobre una parte específica de este. TODOS los dispositivos USB DEBEN responder para los pedidos del HOST sobre los descriptores USB estándar.

Esta información varía y depende de cada dispositivo y de la aplicación particular en la que se utilice. Un descriptor para una configuración típica está compuesto como mínimo por:

- Al menos un descriptor de configuración
- Uno o más descriptores de interfaz
- Uno o más descriptores de endpoints

Puede existir además un descriptor de cadena de texto (USB string descriptor) que contiene una descripción del dispositivo. De esta forma el dispositivo puede pasar esta información descriptiva al SO. Tiene una estructura particular y trabaja en conjunto a otro descriptor de Offset (USB string descriptor offset).
Es en este archivo que también son definidos el Vendor ID (VID) y el Product ID (PID), números que sirven para la identificación de un dispositivo en particular y así también su asociación con un driver determinado para asegurar su funcionamiento.
Microchip entrega dos valores de uso masivo para VID y PID, estos se pueden usar de

manera libre y son:

VID = 04D8h
PID = 0011h o **000Bh**

Existen varios tipos de descriptores que pueden ser de uso obligatorio (o no) y que definen distintos aspectos de funcionamiento del USB del dispositivo. Los mas comunes son los de:

- Dispositivo (obligatorio)
- Configuración (obligatorio)
- String (opcional, texto descriptivo)
- Interfaz (obligatorio)
- Endpoint (obligatorio para usar otro endpoint además del de configuración)

Excepto para dispositivos compuestos, cada dispositivo contiene SOLO un descriptor de dispositivo que contiene la información de este y especifiica el número de configuraciones que este soporta. Para cada configuración este deberá contar con un descriptor de configuración con información sobre el consumo de energía del dispositivo, el número de interfaces que esta configuración soporta y para cada interfáz a su vez se deberá contar con un descriptor de interfáz que especifique el número de endpoints en cada una de ellas. Finalmente cada endpoint tiene un descriptor de endpoint que contiene la información necesaria para comunicarse con este. Una interfáz sin un descriptor de endpoint utiliza el endpoint de configuración (0), el dispositivo debe devolverlo asó como también todos los de interfáz, endpoin y otros descriptores subordinados tantos como bytes haya solicitado el HOST. El HOST no puede solicitar solo un endpoint (su descriptor) en particular.

En general la estructura de los descriptores es la misma en cuanto a los datos que contiene y el orden en el que estos se presentan, este es:

- bLength: Byte que da la longitud del descriptor en bytes
- bDescriptorType: Byte que identifica el tipo de descriptor

El resto de los campos varían en función del tipo de descriptor (device, configuration, interface, endpoint, string, etc).

Partes necesarias para un desarrollo basado en un microcontrolador PIC

1. Programa de la aplicación para el microcontrolador (PIC)
2. Programa de la aplicación para la PC
3. Edición de las librerías suministradas por Microchip

Consideraciones para la programación del microcontrolador PIC

- Los microcontroladores PIC de la serie 18 (que es la utilizada para esta aplicación) que soportan la comunicación por USB se denominan genéricamente 18F**XX**5**X** (en nuestro caso 18F4550)
- Incluír los siguientes archivos (suponiendo que se utiliza el compilador proporcionado por PIC C):
    - pic_usb.h → Serie 16
    - pic18_usb.h → Serie 18
    - usb.h → Definiciones comunes y prototipos

        - usb.c → Manejo del stack USB, la interrupción USB y el pedido de configuración del endpoint de configuración (0)
        - usb_cdc.h → Incluye los archivos anteriores para generar un dispositivo USB CDC
- Interrupciones válidas
    - int_usb() → Un evento USB ha ocurrido (requiere atención del programa). CCS maneja automáticamente el control de la interrupción.
- Funciones relevantes de las librerías
    - usb_init() → Inicializa el hardware USB. Debe esperar infinitamente para que el dispositivo USB se conecte al BUS, esto no significa que el USB sea enumerado por el HOST. Hace uso de la interrupción USB
    - usb_init_cs() → Realiza la misma función que usb_init() pero no espera a que el dispositivo se conecte al BUS. Esto es útil para un dispositivo que no es alimentado por el BUS y puede operar sin una conexión USB
    - usb_task() → Si se utiliza el sensado de conexión y usb_init_cs() para la inicialización esta función, debe ser continuamente revisada para verificar por una posible conexión al BUS. Cuando se desconecte del BUS resetéa el stack USB y el periférico. Hace uso de la interrupción USB. DEBE definirse USB_CON_SENSE_PIN en la aplicación para referír este pin
    - usb_detach() → Remueve al PIC del BUS. Esta función es llamada automáticamente por usb_task() si se pierde la conexión con el BUS pero puede ser llamada manualmente
    - usb_attach() → Conecta el PIC al BUS. Es llamada automáticamente por usb_task() pero puede ser llamada automáticamente
    - usb_attached() → Si se usa el pin de sensado, se devuelve un TRUE si este se encuentra en alto, de lo contrario se devuelve FALSE
    - usb_enumerated() → Devuelve TRUE si el dispositivo fue enumerado por el HOST. Si esta etapa se realizó, la conexión se encuentra en modo normal de operación y se permite el envío y recepción de paquetes
    - usb_put_packet(endpoint, data, len, tgl) → Coloca un paquete de datos en el buffer del endpoint establecido. Devuelve TRUE si hay éxito, retorna FALSE si el buffer aún está lleno con el último paquete.
    - usb_puts(endpoint, data, len, timeout) → Envía información al endpoint seleccionado. Se diferencia de usb_put_packet() en que va a enviar múltiples paquetes si la información no entra en un solo paquete
    - usb_kbhit(endpoint) → Devuelve TRUE si el endpoint especificado tiene información en su BUS de recepción
    - usb_get_packet(endpoint, ptr, max) → Lee bytes hasta una cantidad especificada por “max” del endpoint especificado y lo almacena en el puntero especificado por “ptr”. Devuelve el número de bytes almacenados en ptr
    - usb_gets(endpoint, prt, max, timeout) → Lee el mensaje disponible en el endpoint especificado. La diferencia con usb_get_packet() es que esta instrucción en cambio espera hasta la recepción completa del mensaje donde este puede contener mas de un paquete. Devuelve el número de bytes recibidos

Consideraciones para la programación de la PC

Tener en cuenta que el programa debe estar vinculado a la dll de Microchip en el caso de usar su driver por medio de la **mpusbapi.dll**.

Para cada plataforma de PC (esto es basado en su arquitectura y el SO implementado en esta) es necesario contar con el driver correspondiente, no es lo mismo tener un driver para 2 arquitecturas (x32 y x64) o SO diferentes. Es recomendable tener el del SO mas nuevo y de la arquitectura mas compleja (Windows 7 y 64 bits respectivamente). Se puede obtener de la página de Microchip bajo el nombre de “USB Framework”.

Consideraciones a nivel Hardware del PIC

Configuración y consideraciones de hardware para el Clock y osciladores

- Para el uso del USB con el protocolo 2.0 es necesario que el periférico cuente con una señal de Ck (USB) de 48 MHz. Esto puede lograrse mediante la configuración correcta de ciertos bytes (fuses) que van relacionados al valor de resonador/cristal colocado externamente. Este cristal (según su frecuencia de trabajo) será necesario colocar dos capacitores que estabilizarán la frecuencia del cristal.
- Para el uso del modo de alta velocidad (USB 2.0) es imperioso que el Ck del USB sea configurado a 48 MHz, esto no es necesario para el modo de trabajo en baja velocidad, por otra parte no es necesario que el Ck del CPU sea configurado a la misma frecuencia que el USB por lo que el USB (según su modo configurado) tendrá una frecuencia mientras que el Ck de la CPU puede tener un valor menor.

Consideraciones de hardware de conexión, estabilización y alimentación del PIC

- Si se utiliza el periférico USB es OBLIGATORIO que la tensión de la patita VUSB sea estabilizada (ya que estabiliza la tensión utilizada por el módulo USB) y lleva como MÍNIMO un capacitor de 220nF a GND.
- El pin de la alimentacióndel BUS USB es el encargado de la alimentacióndel USB. Sin embargo, dependerá del tipo de alimentación del USB (del proyecto/placa). Si el diseño tiene alimentación propia, no es necesario conectar VUSB, por otra parte, si se desea que el BUS USB alimente a la placa, será necesario conectar VBUS a VDD y de esta forma no se precisará una fuente de alimentación externa. Puede utilizarse un JUMPER que conecte o abra la conexión entre VBUS y VDD. No se recomienda utilizar AMBOS medios de alimentación
- Se puede (o no) utilizar el regulador interno de tensión y las resistencias internas de pull-up (del USB) pero debera SOLO utilizarse o las internas o una disposición similar externa, NO AMBAS

Registros y flags vinculadas a la conexión USB

- Registro UCON → El bit USBEN pasa de 0 a 1 cuando se conecta el PIC al BUS USB (attach). Al desconectar es necesario resetear esa bandera a bajo mediante una interrogación de actividad del USB. Esto esta relacionado a la conexión y desconexión del USB en caliente sin reiniciar el PIC

¿FALTAN?