V-USB Firmware driver para AVR.

Aplicaciones en Comunicaciones.

Jacobi Diego

diego@jacobi.com.ar

Universidad Tecnológica Nacional Facultad Regional Paraná
Argentina – Entre Ríos - Paraná

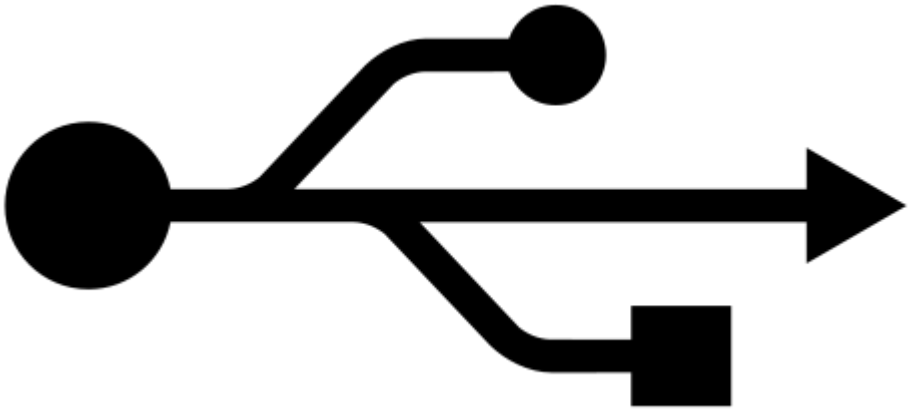
keywords: usb, comunicaciones, protocolo, implementacion, avr

Resumen:

Con este documento se explica y ejemplifica un método de programación utilizado para conectar una circuito electrónico que incluya cualquier microcontrolador AVR con una interfaz de usuario en una computadora mediante el protocolo USB.

Abstract:

This document explains and illustrates a programming method used to connect an electronic circuit that includes any AVR microcontroller with a user interface on a computer using the USB protocol.



Son muchas las formas de poder establecer una comunicación USB con una computadora. Algunas familias de microcontroladores de diferentes marcas ya poseen módulos internos por hardware capaces de funcionar como esclavo haciendo mínimo uso de código del programa. Otras familias van mas allá e implementan USB On-The-Go capaz de comunicarse como Host con otro dispositivo, como lo hace celulares y PDAs por ejemplo. Existen también otras soluciones externas al microcontrolador como la ampliamente usada interfaz USB a UART de FTDI, el FT2232H, que permite a cualquier microcontrolador con un puerto UART poseer una comunicación USB, o incluso por JTAG, I2C, SPI o paralelo. O finalmente la solución en la que este texto se concentra.

La solución por software tiene una serie de ventajas únicas que los métodos anteriores no poseen, así como también una serie de limitaciones.

Ventajas y desventajas de un driver USB:

Ventaja: Un driver bien programado podrá ser usado en todas las familias de microcontroladores de una marca o incluso de varias marcas. La interfaz con el driver (o API) puede ser constante entre marcas, permitiendo al programador portar fácilmente su firmware a otra familia, posiblemente, muy diferente.

Desventaja: No existe actualmente un driver multimarca. En cambio los modulos externos como el FT2232H si puede ser usado en cualquier familia o marca.

Ventaja: Al no estar conformado por transistores, un driver no ocupa tamaño físico en la electrónica interna del microcontrolador, permitiendo que éste posea otros módulos en su lugar.

Desventaja: En cambio, un driver ocupa tiempo de procesamiento. Aunque mientras no hay transferencias, el tiempo es un 5% o menos, y mientras hay transferencias puede ser mucho mayor, pero el driver no detiene el firmware.

Ventaja: Los pines utilizados no están fijos y pueden ser cambiados por software a conveniencia, por lo que usar comunicación USB no te quitará ese valioso pin de ADC o PWM que te serviría de mucho usar.

Desventaja: Una interrupción de alta prioridad debe ser usada.

Ventaja: Un driver puede no implementar completamente el protocolo haciendo uso solamente de lo que necesita, puede incluso violar algunas de las especificaciones, como por ejemplo realizar una transmisión en masa en un dispositivo de baja velocidad.

Desventaja: Violar especificaciones cuando se implementa USB por software en dispositivos de muy baja velocidad es casi forzado para obtener el mejor rendimiento, y nunca es buena idea.

Ventaja: Puede ser completamente desconectado por software, evitando así que consuma energía cuando no es usado, a diferencia de las demás soluciones que requieren de una señal de clock adicional ya sea por cristal externo o por PLL, hace que el modo idle o de bajo consumo sea mas difícil de lograr.

Desventaja: Se puede usar solo un conjunto finito de frecuencias de clock para poder mantener la sincronización, y ésta de vez en cuando puede que se pierda, pero el driver se encarga de recuperarla.

Ventaja: Un driver tiene costo de producción nulo.

Desventaja: Las licencias y el par VID/PID para aplicaciones comerciales no son siempre gratuitos.

Ventaja: Un driver no se puede quemar y no se suelda.

Ventaja: Puede ser usado en familias con encapsulado DIP, ideal para pruebas en protoboards.

Ventaja: Puede ser usado junto con otro modulo interno de USB y con otro externo como el FT2232H pudiendo tener mas de una interfaz USB. Por ejemplo, un modulo interno On-The-Go para comunicar con otro dispositivo como host y el driver como esclavo.

Ventaja: No usa el puerto UART, por lo cual el mismo puede ser usado con otros propósitos, como para conectar con otro dispositivo, como interfaz de debugueado mandando caracteres a una PC, como interfaz de bootloader por puerto serie, etc.

Ventaja: Puede programarse para funcionar como bootloader por USB. Esto presenta una ventaja adicional. Se puede programar para simular ser un grabador ISP por USB como el USBasp y usar el mismo software de grabación.

Hardware:

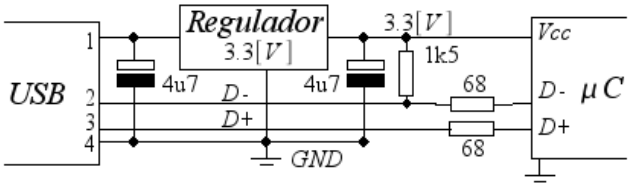
El hardware adicional requerido por un driver USB no es mas que una adaptación niveles de tensión e impedancias entre el microcontrolador y el bus.

El host envía 0 [V] para un nivel bajo y 3.3 [V] para un nivel alto. Por suerte aquí no tenemos mucho problema ya que los AVR, incluso en 5V de alimentación, reconocen estas tensiones como nivel alto.

El único problema se encuentra en los datos enviados por el dispositivo, ya que el host espera un nivel bajo de entre 0 y 0.8 [V] y un nivel alto de entre 2 y 3.6 [V].

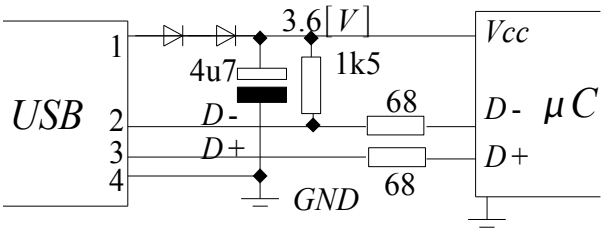
Hay dos soluciones para este problema.

Una solución es alimentar el microcontrolador con 3.3 a 3.6 [V], de esta forma las salidas tendrán un nivel compatible con USB, mejor inmunidad a ruido, transiciones rápidas de niveles, y si el circuito se alimenta con batería recargable, entonces podría ser recargado desde el mismo bus del USB.

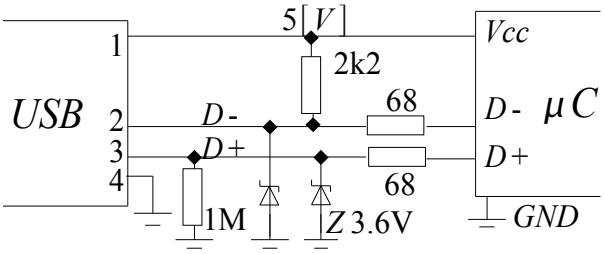


Algunos reguladores pueden por si mismos consumir mas de 500 uA, lo que viola el limite de bajo consumo del protocolo. 200 uA son consumidos por la resistencia pull up de 1k5 que determina la conexión low speed del protocolo, quedando un máximo de 299 uA para el resto del dispositivo en modo bajo consumo. Un AVR de la familia ATmega requiere un mínimo de 1 uA en su mas bajo modo de bajo consumo.

Otro circuito opcional es no usar regulador y usar 2 diodos en serie desde la alimentación de 5V. Esto genera una caída de tensión entre 1.2 y 1.4 [V], pero puede generar muchos problemas para usar con ADCs, ya que estos necesitan de una fuente estable, y los diodos varían la caída de tensión con la corriente que circula.



La tercera solución es convertir los niveles a la salida de cada pin y operar con el microcontrolador en 5[V].



Se puede usar Zenners de 3.6 V o de 3.3 V, pero puede ocurrir que con estos últimos el Host no detecte la señal correctamente.
La ventaja de usar Zenners es que todo el circuito puede funcionar a 5 V, se tienen una alta velocidad de subida y bajada de los niveles, son de bajo costo y fáciles de obtener.
Las desventajas es que los zenners no son todos iguales y no son lineales y consumen mas corriente en los niveles altos.

Frecuencia de clock:

La limitación que aquí tenemos es que en low speed el bus trabaja a 1.5 Mbps, por lo tanto el clock debe ser un múltiplo de esta frecuencia para máximo sincronismo.
Un microcontrolador requerirá como mínimo de 12 Mhz de clock para leer y guardar los datos del bus.
Una conexión en full speed es de 12 Mbps y requiere de un clock de 48 Mhz, el cual es frecuentemente generado con un PLL usando un cristal externo de 12 Mhz.
Es por eso que la mayoría de los dispositivos USB poseen un cristal oscilador de 12 Mhz, sin embargo, gracias a que se trata de una implementación por software, podemos usar otras frecuencias de clock haciendo trucos

especiales para mantener el sincronismo.
El driver V-USB para AVR posee un segmento de código para cada frecuencia soportada. Entre ellas 12, 15, 16, 16.5 y 20 Mhz. Por ello dependiendo de la frecuencia de clock usada puede variar el tamaño compilado del driver. El menor tamaño es para las frecuencias de 16 y 20 Mhz.
Los valores se consideran precisos. Esto significa que un cristal de 11.9 Mhz no serviría.
Algunas de las frecuencias listadas, como la de 20 Mhz, solo están disponibles como osciladores RC internos. Estos osciladores deben ser calibrados automáticamente por el driver.
Hay ejemplos sobre ellos en la página oficial del driver.

Comunicación con la computadora:

La comunicación con la computadora se establece siempre a través del API del sistema operativo.
Esto agrega limitaciones en el mercado de nuestro producto, por lo que si queremos abarcar todo el mercado tendremos que programar el software para cada uno.

LibUSB es un proyecto de código abierto nacido en linux que posee actualmente 2 versiones mayores del API que define. Ambas incompatibles. La serie libusb-0.1 y la serie libusb-1.0.
La serie libusb-0.1 está actualmente soportada por una capa de compatibilidad (libusb-compat-0.1) en los sistemas Unix debido a que gran cantidad de software aun lo usa, pero para software nuevo y drivers no es aconsejable su uso.
Por esa razón en los ejemplos se incluyen códigos para ambas versiones.
La serie libusb-1.0 define un API mucho mas simple, limpio y completo que su predecesor, pero lamentablemente aun no es posible su uso en sistemas Windows.
Actualmente solo la serie 0.1 es funcional en Windows.
libusb-0.1 y libusb-win32-0.1 son proyectos independientes que implementan el mismo API en todo aspecto posible, de forma que podemos programar bajo el API 0.1 y hacer nuestro código portable entre distintas plataformas, permitiéndonos llegar a un más amplio mercado.

En el futuro se prevee que libusb-1.0 sea portado a Windows, esto significa que formarían parte del mismo proyecto, pero ese futuro aún no ha comenzado y mientras tanto el API mas portable es el de la serie 0.1.

Clases de dispositivos:

Cuando diseñas un dispositivo, debes decidir cómo debe presentarse ante el Host.
USB es un protocolo mucho mas complicado que RS232. No solo define los parámetros eléctricos o como viaja la información, si no también como obtener información y como trabajar con un dispositivo determinado agrupándolos en clases.
Mientras que el protocolo básico está manejado por el driver, la comunicación y la implementación de la clase respectiva es tarea del programador. Así, este puede elegir implementar entre varios tipos de clases:

- Custom Class Devices
- Standard HID Class Devices
- Custom HID Class Devices
- Other Devices

Custom Class Devices:

Es la forma mas directa de implementar un dispositivo que no puede ser clasificado en ninguna otra clase. Cualquier dispositivo que no planee trabajar bajo el standard puede ser implementado de esta forma.
Ventajas: Fácil de implementar. Fácil de crear software en sistemas Unix.
Desventajas: En windows, se requiere crear un driver e instalarlo en cada computadora que se quiera el software. Si o si, un software debe ser programado solo para dicho dispositivo.

Como código de ejemplo se incluye el firmware y software en el anexo USB_HelloWorld.

Standard HID Class Device:

Puedes diseñar tu dispositivo para que se comunique como si fuera un dispositivo

estandard, como un mouse, teclado, joystick, una fuente, etc.
Como para el host tu dispositivo es un HID, entonces ya sabe como comunicarse con él.
Ventajas: No se requiere drivers.
Desventaja: Solo puede implementarse HIDs. Se debe crear un HID Report Descriptor.

Hay muchos ejemplos de implementaciones en la página oficial del driver V-USB.

Custom HID class device:

Es posible "mal usar" transacciones de datos definidas por la clase HID (originalmente diseñadas para describir el tipo de información que contiene un dato) de forma que podamos enviar y recibir datos arbitrarios.
Con esta técnica se puede evitar los inconvenientes con Microsoft Windows de forma que éste asigne el driver para dispositivos HID automáticamente, y no necesites instalar tu propio driver.
Ventajas: No se necesita drivers.
Desventajas: El largo de los bloques de datos esta definido en el Report Descriptor. Algunos sistemas operativos (como BSD) limitan el acceso de libusb cuando el driver hid se ha cargado. Es ligeramente mas complejo que el primer caso.

Como código de ejemplo se incluye el firmware y software en el anexo USB_HID_HelloWorld y ligeramente mas avanzado en el anexo USB_HID_BufferTransfers.

Otra forma es implementar peticiones de tipo Vendor en un dispositivo HID tonto.
Esto significa que el dispositivo se presenta como HID pero no cumple la función como tal. Esto ayuda a evitar las ventanas emergentes de Windows cada vez que detecta el dispositivo y permite que sea accedido por libusb, pero requiere de la instalación del driver de todas formas.

Other Devices:

Una vez entendidas las cualidades del driver, queda a capacidad del programador implementar otros tipos de dispositivos del estandard.

Ejemplos:

Con este documento se proveen 4 paquetes de ejemplos.
Todos están explicados usando comentarios y pensados para que el lector estudie el programa en un orden dado, ya que los ejemplos mas avanzados no explican detalles explicados en ejemplos menos avanzados.

Cada ejemplo se compone de al menos 2 partes, el Firmware y el Software. El Hardware es para todos el mismo.
Las herramientas utilizadas para el firmware y software son las siguientes:

- Codeblocks,
- V-USB,
- libusb / libusb-win32
- WinAVR

En algunos ejemplos se provee en forma duplicada el software, para la versión 0.1 de libusb, la cual se encuentra disponible tanto para Windows como para Unix, y la versión 1.0 que aún no se encuentra disponible para Windows.
El código está hecho para ser sencillo, multiplataforma y mostrar las características de la comunicación y no persigue ninguna aplicación especial.

El lector debe estudiar el código antes de obtener conclusiones con el software ejecutado, y se recomienda al lector en cada caso que realice modificaciones a gusto sobre cada ejemplo para obtener una experiencia de aprendizaje mas rápida y rica.

USB_ListDevices:
Se trata de un programa de introducción a la programación con libUSB. No posee firmware.
Su única función es listar algunas de las propiedades de los dispositivos USB que se encuentran conectado al sistema.
No se llevan a cabo transacciones.

USB_HelloWorld:

Es el primer ejemplo con Firmware que muestra la implementación mas simple de un Custom Class Device.
Se llevan a cabo las transacciones mas simples.

USB_HID_HelloWorld:
Este segundo ejemplo implementa el mismo anterior pero en forma de un Custom HID Class Device, por lo tanto no requiere instalar drivers.

USB_HID_BufferTransfers:
Este ejemplo implementa un Custom HID Class Device y agrega un conjunto mas avanzado de transacciones de datos, incluyendo transferencias de buffers.

Referencias:

Especificaciones USB

Objective Development V-USB

FTDI Published Articles

WinAVR

Code::Blocks

LibUsb

LibUsb-Win32