

# Connect your embedded system to a PC with USB

---

*Stephen Martin, Quadros Systems, Inc.*

A popular use for USB in an embedded system is to allow a PC to read log files or similar data stored on the resident flash memory in the embedded system. This is useful for allowing occasional access by technicians (since RS-232/DB9 connectors have disappeared from laptops) or it can be a routine way of transferring log data to a PC for analysis.

## **System Definition**

1. When the embedded system is connected by a USB cable to a PC it must be recognized (or more accurately, "enumerated") by the PC as a remote drive.
2. The PC must be able to read and/or write to the resident flash on the embedded system using standard file system routines in Windows Explorer (drag and drop, move, copy, etc.)

## **System Components**

Building such a system requires several components and an understanding of how they interact.

- USB device (or OTG) controller and USB device stack with a mass storage class driver
- FAT32 file system
- Resident NAND flash memory
- File translation software

Let's examine each of these components in more detail. Figure 1 shows a diagram of the system.

## **Device Controller and USB Stack**

The functionality of a USB Device (or peripheral) controller is defined by the Universal Serial Bus specification. Because USB is a master/slave structure, the USB Device must be initialized and made ready to receive activity from the host. (Note: An OTG controller may also be used because it supports full USB Device capability.) The Device controller together with USB Device software (communications stack and device driver) allows data transfer to and from an embedded device over the Universal Serial Bus.

Depending on how it is configured, the USB Device can present itself to the Host as a member of one of many USB device classes. This determines how the Host will interact with the embedded system. In this case, the USB Device must present itself, using the mass storage class driver, as a remote storage device or flash drive. This requires integration between the USB Device software and a PC-compatible file system.

### **FAT32 File System, File Translation Software, NAND Flash**

A key requirement of our specification is that the PC must be able to read the resident files on the embedded system. That means the embedded system must write and manage its local files in exactly the same way the PC would. A FAT (File Allocation Table) file system is the standard format for PC-based file systems. FAT32 is the latest variation.

The only remaining file system issue is that a standard FAT file system cannot write natively to resident flash. A special file translation layer is helpful to prepare the memory device to accept data from the FAT system in 512 byte sectors and to manage the sectors such that the device does not fail from overuse of one or more sectors. NAND flash is a very cost effective flash storage media. However, it has certain characteristics which mean that it cannot be used as a simple continuous storage array. NAND flash may contain bad blocks; bad bits may develop in previously good blocks; and heavily used blocks will wear out. To allow a higher level system – such as a file system – to use NAND flash effectively it is necessary to use an FTL which carefully manages the flash while providing a simple logical sector interface to the host system.

This leads us to a discussion of the flash device itself. We have specified NAND flash because NOR flash is a poor design choice for this application. We do not support running FAT + FTL on NOR devices and advise against our customers designing such a system because it will be unstable and prone to problems. NAND flash is a much more desirable medium for a FAT file system.

Here are some of the issues in using NOR flash with a FAT file system:

**Long erase times.** Typically it takes 1 second to erase a NOR flash block. The time gets longer as the device gets older — sometimes as long as 20 seconds. The long erase time means that the system is waiting before it can write data, thus backing up other operations. During this time the system is vulnerable to data loss.

**Fragmentation.** NOR flash blocks do not have “housekeeping” space set aside in their blocks. This means that a separate array of blocks must be set aside for this purpose. The end result is a big fragmentation problem that will grow over time.

### **System Management**

A final design consideration is that the PC and Embedded System cannot address the NAND Flash at the same time. Because some file system information is cached by the system, the file system must be dismounted prior to relinquishing control over it. This flushes all file system tables and data to the NAND flash, which allows the new owner of the file system to see the most recent changes and prevents the old owner of the file system from ignoring the changes that will be made by the new owner. This means the local file system must be dismounted before the PC is allowed to connect and then remounted after the PC has disconnected. This is a simple routine you can write as part of your application.

## Summary

Designing your embedded system to allow a PC to read resident flash memory requires the correct components and interface code so that they work seamlessly together. Quadros Systems offers each of the discussed software components, integrated and ready to work in your system. Our “Getting Started” service assists new customers on how to open the sample project provided with the software distribution and get started modifying the code to meet their specific application requirements. To find out more about products and services from Quadros Systems, contact your local sales office or send an email to [sales@quadros.com](mailto:sales@quadros.com).

Figure 1

