# Starting a USB Design Using MSP430™ MCUs

*MSP430 Applications*

**Contents**





## 1 Introduction

This document is a roadmap to starting a design with MSP430's USB solution. It provides the system designer:

- a summary of MSP430's USB module
- a brief description of the *MSP430 USB Tools Suite,* and how the tools relate to one another
- information to help in starting any MSP430 USB design

It is a "front door" through which the system designer can understand MSP430's USB offering and how to get started with a design. In most cases it does not go into detail, but rather summarizes the subject and instructs the reader of where to find more information.

# 2 MSP430 USB Module

A high-level overview of the MSP430 USB module is as follows:

- **Full-speed USB device (12Mbps).** Full-speed is a great match for the MSP430 and its typical applications. It facilitates communication with a PC (or other USB host) with simplicity and low system cost.
- **Supports control, interrupt, and bulk transfers.** This enables support of the most popular USB device classes. (The only transfer type not supported is isochronous, which is generally for audio/video applications.)
- **Eight input / eight output endpoints.** This is more than enough for the vast majority of applications.
- **Integrated 3.3V LDO, for operation directly from 5V VBUS.** Many competing USB devices require an external LDO. The MSP430's LDO can source not only the USB module, and not only the rest of the MSP430, but in some cases the entire system – up to 12mA.
- **Integrated D+ pullup.** Competing USB devices sometimes require a considerable amount of external circuitry.
- **Uses a crystal.** The USB module requires that a crystal be placed on the MSP430's XT2 oscillator. A wide variety of frequencies can be used, because of the programmable PLL. This allows selection of a crystal by other criteria – perhaps another frequency used in the system (the clock can be output to other devices), or the crystal with the lowest cost.
- **Integrated transceiver.** There is no need to buy an external PHY / line driver – just attach directly to a USB connector (with discretes).

A system block diagram is shown in Fig. 1.

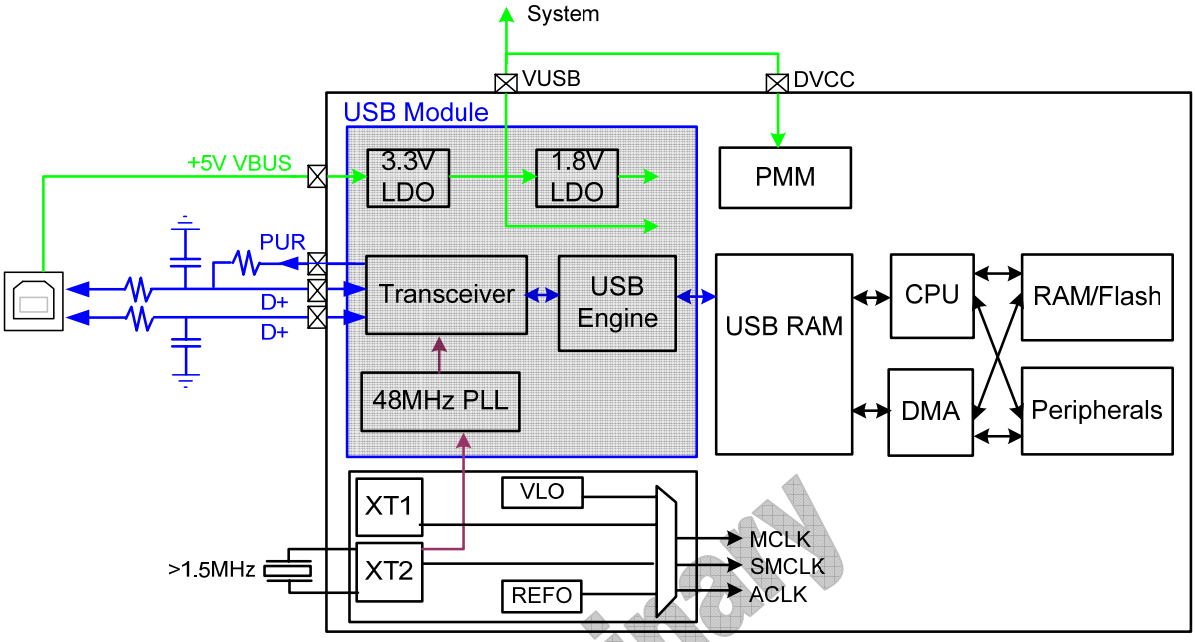


**Figure 1. Block Diagram**

Please see the *MSP430F5xx Family USB User's Guide* for more information.



## 3 *MSP430 USB Tools Suite* Overview

USB is an excellent example of a complex system that's been made to look very simple. But designing with USB is often harder than it looks, and so TI attempts to ease the process with a complete tools suite. The tools suite is intended to streamline common tasks, as well as assist the designer new to USB in knowing how to proceed.

Fig. 2 shows a high-level view of the tools in the suite. Items available as of this writing (July 2009) are shown in black; items available later this year are shown in grey.

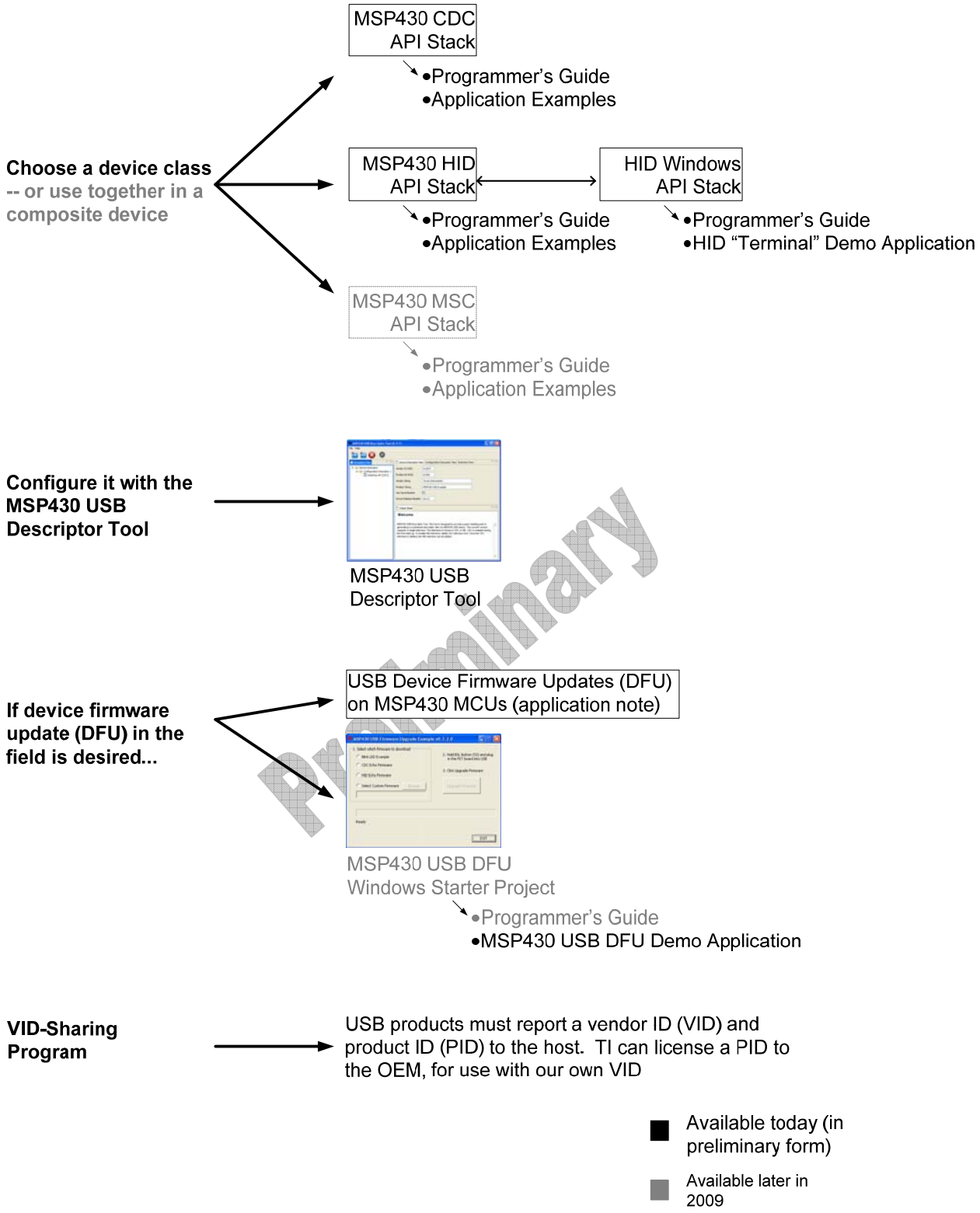




**Figure 2. MSP430 USB Tools Suite**

## 3.1 MSP430 USB API Stacks

The foundation of the tools suite is a set of API stacks that implement the most common USB device classes. These USB-certified stacks are implemented as APIs with accessible source code. Most of the USB functionality is “under the hood”, and most designers shouldn’t need to edit the source directly. (However, the designer can do so if desired.) This carries an advantage in that, if the designer doesn’t modify, it maximizes the probability of the USB certification staying intact. *(Note: certification of the CDC/HID stacks will be completed in August 2009.)*

The API stacks are written to make USB as intuitive as is possible. The programmer’s guides help explain the challenges USB presents, and how the API can assist in dealing with them.

The *MSP430 HID Windows API* is provided to complement the MSP430 HID API stack. Together, they overcome some of HID’s disadvantages, allowing it to be very an excellent general-purpose choice for many applications.

See Sec. 4 for a full discussion of the MSP430 USB API stack options.

## 3.2 MSP430 USB Descriptor Tool

Every USB device must report a set of *descriptors,* which identify it to the host. The descriptors can be sensitive to coding error; any formatting mistake can cause the device to fail enumeration. Therefore, the design/debug process can become time-consuming.

TI provides a *Descriptor Tool* to make descriptor generation for the MSP430 fast and reliable. The user inputs the necessary information in a graphical user interface, and then tells the tool to generate the output code. The resulting *descriptors.c/h* files are then used to replace the defaults in the API stacks. Inputs can be saved in a **.dat* format and edited at a later time.



The Tool can also be thought of as a sort of ”control panel” for the API stacks. The stacks have various “configuration constants” that can be used to “tweak” the stack’s performance; the Tool can configure those as well.

The Tool includes a help pane, with commentary that aids the user in making each decision.

It is recommended to use the Tool once prior to beginning application development, since some of the chosen items affect the way the API stack works. However, assuming the **.dat* file is saved, the tool can be run at any later time by opening it and re-generating the *descriptors.c/h* files. For example, often the PID isn’t chosen until late in the development cycle, and so the tool may need to be run again at that time.

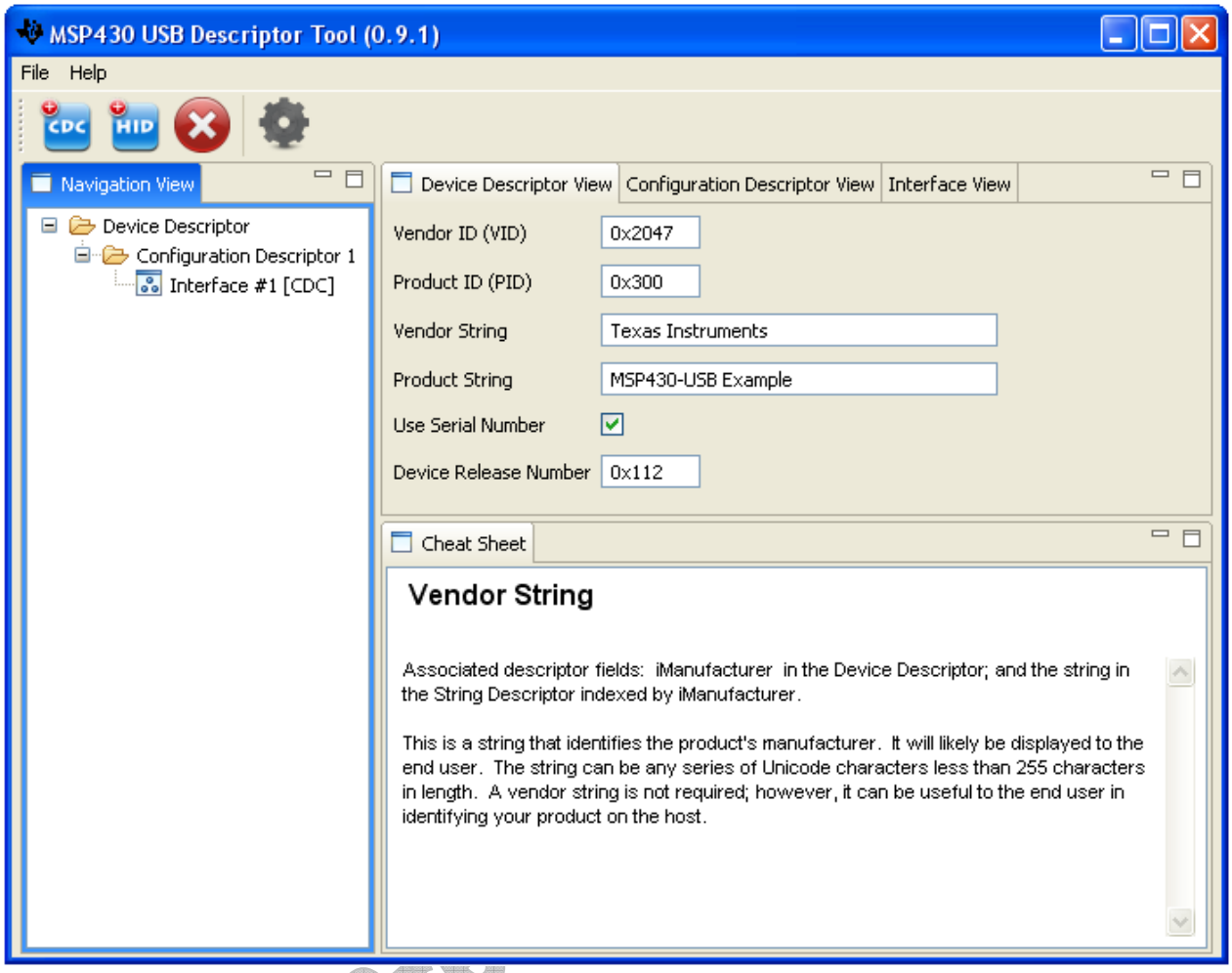


**Figure 3. The MSP430 USB Descriptor Tool**

*(NOTE: v1.0 of the Descriptor Tool supports only single-interface CDC and single-interface HID devices; composite devices are not supported. The configuration constant controls are also not yet included. v2.0 of the tool, scheduled for September 2009, is planned to add both these features.)*

## 3.3 MSP430 USB Device Firmware Upgrade Tools

A common USB application is to provide end users a means of updating firmware in the field. This carries many advantages for the OEM – for example, product returns can be reduced by pushing bug fixes to the field.

All USB-equipped MSP430 devices are equipped with a bootstrap loader (BSL) that downloads firmware from a USB host. This requires software on the host that can perform this function, and so TI provides the *MSP430 USB DFU Windows Starter Project*, a Visual Studio project that implements all the basic firmware update functionality. The project can be quickly customized, or integrated into a larger application.

The DFU functionality is described in the application note *USB Device Firmware Update (DFU) on MSP430 MCUs.*

Another document, the *MSP430 Memory Programming User's Guide,* discusses all aspects of memory programming on all MSP430 devices. It covers the USB bootstrap loader at a lower level. The system designer may not need to use it for implementing USB DFU, but it is available as a detailed reference.

*(NOTE: As of July 2009, the application note, as well as a demo application compiled from the starter project, have been released (in preliminary form). The starter project itself is planned for August 2009.)*

### 3.4 MSP430 VID-Sharing Program

All USB devices must report a vendor ID (VID) and product ID (PID) to the USB host. TI can help, by licensing a PID from our own VID space. See Sec. 5.2 for more information.

## 4 How to Choose a USB Device Class

One of the reasons for USB's success is the definition of "device classes" – a clear protocol definition for common applications. The most popular ones are supported natively by most operating systems, and this means most OEMs can avoid writing host drivers. This is a great advantage, because such drivers can be difficult to write, and any bugs they contain can crash the PC due to residing in the OS kernel. In contrast, device class drivers supported in the OS are well-proven and require no installation on the part of the end user. They "just work".

Of course, the USB device itself needs to have software that can talk to these device class drivers. TI provides software for the three most popular device classes. These are shown in Table I, in a comparison matrix. In the matrix, straight text represents advantages while italics represents disadvantages relative to the other classes.

**Table 1. Device Class Tradeoffs**

| Characteristic | CDC (Communications Device Class) | HID (Human Interface Device class) | MSC (Mass Storage Class) |
|---|---|---|---|
| Interface generated on the PC | • Virtual COM port<br>• Common in the industry; widely-supported and well-understood<br>• An existing PC application for backward-compatibility might already use it | • *Generates a HID interface on the PC*<br>• *Less familiar than COM ports or storage volumes* | • Ordinary storage volume<br>• Common in the industry; widely-supported and well-understood |
| Installation on the PC | • *OEM must supply an INF file for Windows*<br>• *Despite the actual binary files already existing in Windows, the user must go through a device installation process.* [1] | • Loads "silently" in most operating systems.<br>• No driver files required | • Loads "silently" in most operating systems.<br>• No driver files required |
| How the end user interacts with it [2] | • Via a PC application<br>• OEM might supply this application, or might rely on widely-available terminal applications | • Via a PC application<br>• OEM must supply this application | • The device mounts on the system like a storage volume, which means it can be seen by the end user<br>• OEM might supply an application to interface with the volume, or could rely on the end user interacting with it directly through the OS (i.e., dragging files). |
| Driver certification needs | • *Unless the INF file is WHQL certified, Windows will report that the driver is 'uncertified'.* [3] | • No 'uncertified' message is generated | • No 'uncertified' message is generated |
| Code footprint and complexity | • Small code footprint (4-5K)<br>• Simple architecture | • Small code footprint (4-5K)<br>• Simple architecture | • *Larger code footprint (~15K)*<br>• *Must obtain a file system. Architecture, development, and testing are more complex.* |
| Throughput [4] | • Fast (hundreds of KB/sec)<br>• Uses bulk USB transfers. | • *Slow (64KB/sec)*<br>• Uses interrupt USB transfers. | • Fast (hundreds of KB/sec)<br>• Uses bulk USB transfers. |



Notes:
1. This *device installation process* generally consists of the user pressing the "Next…" button several times. The user must also locate the INF file provided by the OEM. It's a fairly easy process, but it does create the possibility of a failed install, which will require the OEM to support the end user. In contrast, HID/MSC are automatically loaded.
2. This characteristic can be considered an advantage or disadvantage, depending on the situation. Especially, this applies to MSC being accessible as a storage volume to the end user – there are ways in which it could cause significant problems or be a very useful feature.
3. WHQL certification refers to Microsoft's Windows Hardware Quality Labs. See http://www.microsoft.com/whdc/winlogo/drvsign/drvsign.mspx.
4. Throughput on bulk transfers is often significantly affected by host characteristics and therefore harder to predict.

*NOTE: As of July 2009, TI has released the CDC and HID API stacks. MSC is scheduled for 4Q09; until then, MSC is supported by our 3rd parties. Check with TI for more information.*

If a device is very clearly of a certain type – for example, flash storage drives are always MSC devices – then this may drive the selection of the class. But for many MCU applications, the selection isn't as clear. Although there are many ways to approach the decision (depending on an OEM's priorities), TI suggests the following process as a rule of thumb, for deciding between CDC/HID/MSC.

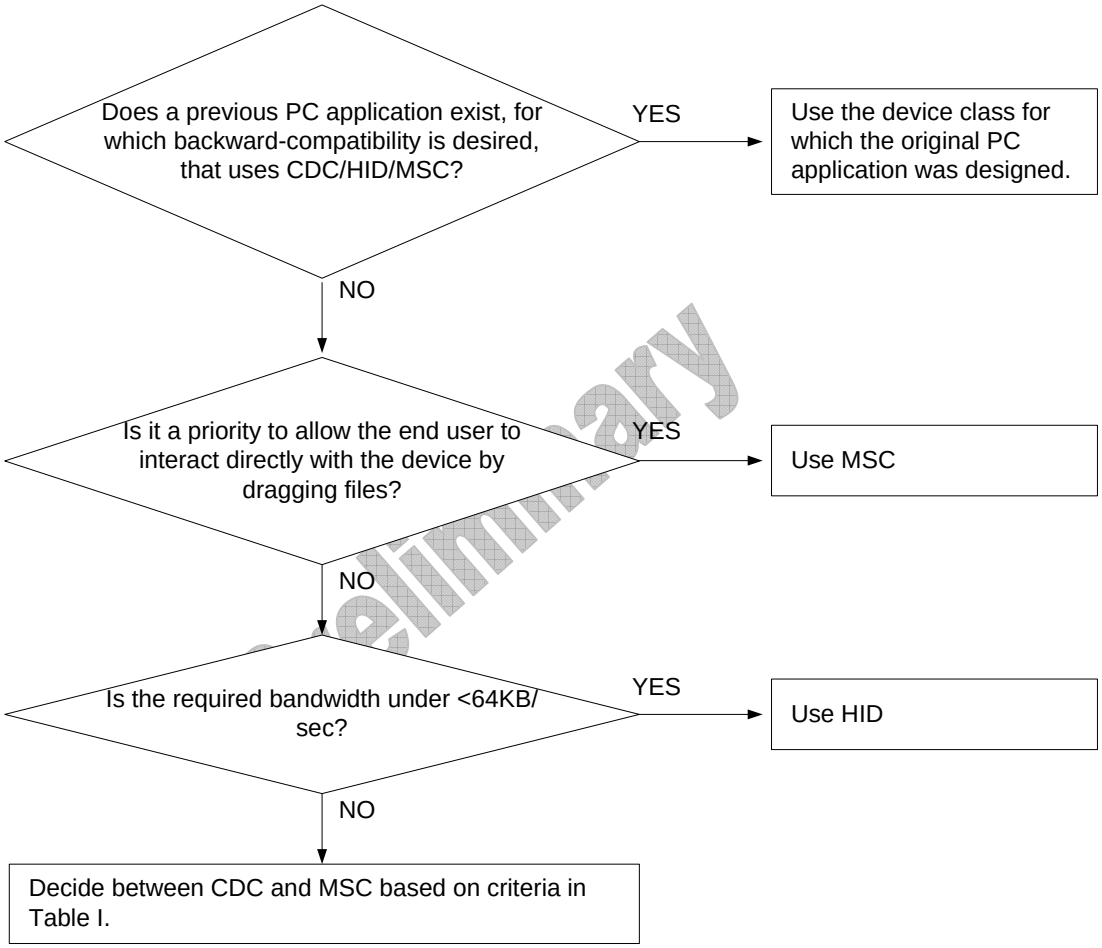


**Figure 4. Suggested Device Class Decision Process**

When a decision has been made, the API stack and programmer's guide can be downloaded from TI's website. They're located on the product pages for USB-equipped MSP430 devices. (Go to http://www.ti.com and do a part number search for the device -- for example, MSP430F5529.)

# 5 How to Select a Vendor ID / Product ID (VID/PID)

A common question about USB is how to select the VID and PID.

## 5.1 What are the VID and PID?

When a USB device is attached to a host, the host asks for its USB *descriptors*. These identify to the host what the device is (i.e., which device class) and what its capabilities are (i.e., does it contain a local power source, etc.). Included in the descriptors are the 16-bit VID and PID values. The VID is associated with a particular vendor/OEM, and a PID is associated with a product sold by that vendor.

For example, if vendor "Vendor1" sells their first USB product ("Product1"), they will obtain a VID, which will now be associated with their company; and they need to then choose a PID to associate with "Product1". When they later release "Product2", they'll use the same VID, but now should use a new PID. It's up to the vendor to ensure they don't duplicate PIDs, otherwise conflicts in the field could result.

A unique combination of a VID and PID allows a USB host to identify one USB product type from another, and this is why it's important that vendors obtain their own VID, and why PIDs shouldn't be duplicated. If the VID/PID of "Product1" and "Product2" are the same, and a host in the field encounters both products, conflicts might result from the host confusing the two products and loading an inappropriate driver.



## 5.2 How are the VID/PID Chosen?

Speaking generally, VIDs are assigned by the USB Implementers Forum (USB-IF), which is the standards body that oversees USB. The vendor can choose to obtain the VID by joining the USB-IF, or to license a VID without joining. At the time of writing, the former costs $4000 annually, and the latter costs $2000 for a two-year license. (See http://www.usb.org/developers/vendor/ for more information.)

Or, TI will license a PID for use with a TI-owned VID, as part of MSP430's VID-sharing program. The license is free, with the basic stipulations that it only be used with MSP430-based devices. The program is intended to make the VID a non-issue for smaller customers trying to go to market with their USB device. To obtain a copy of the license, visit http://www.ti.com/msp430 and click on the MSP430F5xx link.

## 5.3 How are They Used with the *MSP430 USB Tool Suite*?

The MSP430 API stacks make it clear how to use the VID/PID with the software. But in most cases, the easiest way is with the *MSP430 USB Descriptor Tool*, which accepts the VID/PID in an entry field and automatically places them in the descriptors.

Some applications implement USB firmware upgrade using the USB BSL. This requires an additional VID/PID combination. (Or, if firmware upgrade is the only purpose for USB on the device, with no USB implemented within the main application, then the BSL VID/PID pair is the only one required.) Use of the VID/PID with the USB BSL is covered in the document *USB Device Firmware Updates on MSP430 MCUs*.

# 6 Selecting a Power Configuration

The USB host provides 5V power over the USB cable, from which the device can be powered. This power source is referred to as “VBUS”. For devices that are permanently “tethered” to the host, or that never need to function when not attached to the host, this can eliminate the need for a battery or other local source. Such a device is called *bus-powered*. By strict definition, a bus-powered device is one which derives *all* its power from VBUS.

A USB device can also have its own power source, whether from mains power or from a battery. Such a device is called *self-powered*. In the strict sense of the word, according to the USB specification, a self-powered device is one that derives any of its power from a local source (i.e., a source other than VBUS). More informally, such a device can be thought of as *mixed-powered.*

These definitions apply only to how the device is powered *while attached to USB*. A device might derive all its power from VBUS while attached, but then operate fully from a local source when not attached. From a USB standpoint, this device is still considered bus-powered. But from a system standpoint, MSP430 documentation refers to this as *switched-context powering.* This setup is advantageous in the vast majority of battery-powered applications, because being attached to the USB host can keep it running active continuously, and this is a constant drain on the battery. VBUS can help relieve this power burden.

A very common power configuration is shown in Fig. 4. This is both:

- mixed-powered (partly bus-powered and partly self-powered, while attached to the host); and
- switched-context (when disconnected from the host, the entire system is powered from a local source)

For a battery-powered device it usually represents the opportunity for longest battery life.



TBD.

**Figure 5. Example Power Circuit that is Mixed- and Switched-Context Powered**

Other aspects must be considered when selecting a power supply circuit. If the rest of the system consumes more than the maximum allowed for the LDO, then an external LDO might be needed.

Another aspect is if it's desired to use a rechargable battery that is charged from VBUS. For example, this is a model common with cell phones. An example circuit for this is shown in Fig. 5.

TBD.



**Figure 6. Example Power Circuit for a USB-Rechargeable Battery**

# 7 Basic USB Debugging

This section covers some basic concepts that apply to any USB device.

If a USB device fails to operate as expected, the first action is always to determine whether the device enumerated successfully on the host. *Enumeration* is the process by which the host identifies the device and associates it with the appropriate driver.

The basic events related to enumeration are as shown in Fig. 6.

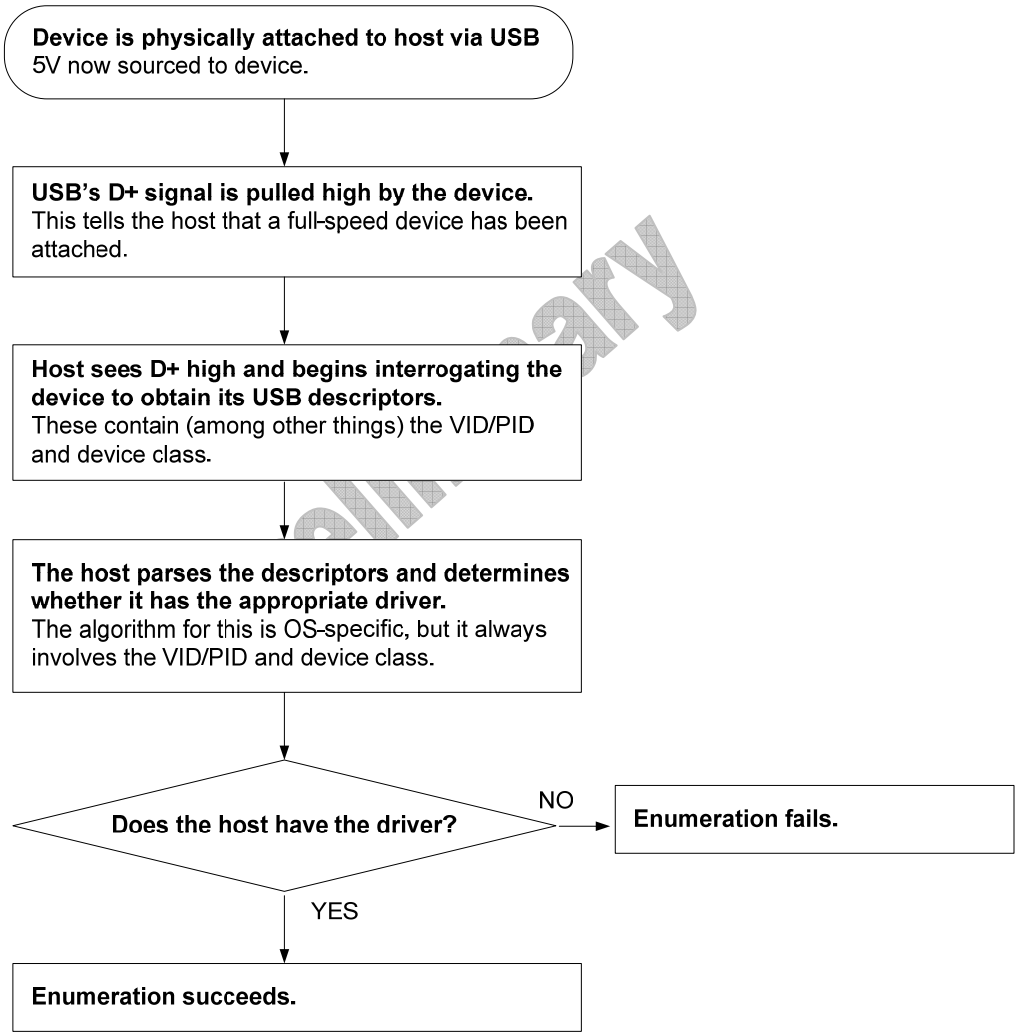


**Figure 7. Basic Enumeration Flow**

Speaking from a Windows standpoint, it's possible to see how far the process gets by observing certain clues. If enumeration succeeds, Windows plays audible alert tones. It can also be verified by checking the Windows Device Manager; in Windows XP, the Device Manager can be opened by right-clicking on My Computer → Properties item → Hardware tab → Device Manager button. It is shown in Fig. 7.

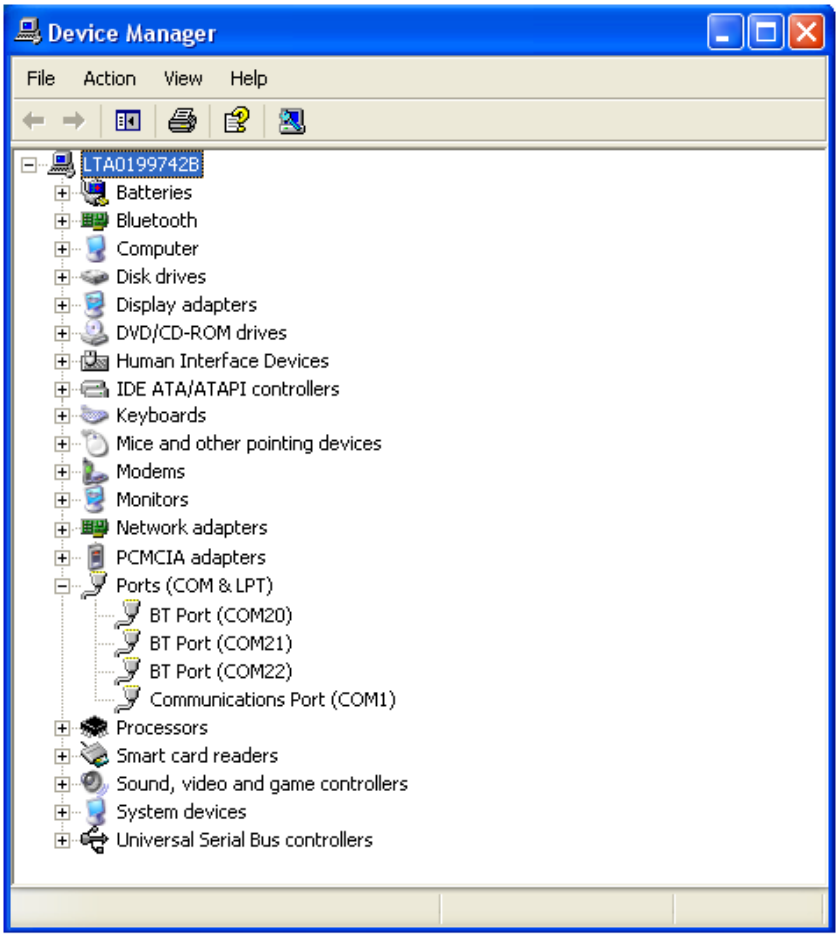


**Figure 8. Device Manager**

The section under which the device should appear depends on the device; CDC devices enumerate under “Ports”, HID devices enumerate under “Human Interface Devices”, and MSC devices enumerate under “Disk Drives”. Whichever category it’s listed under, attaching or detaching should both cause a refresh action in the display. (Try this with a known good USB device to observe it.)

If the refresh doesn’t occur, then the next question is whether the host even saw the device being attached; in other words, was D+ pulled high? (See Fig. 6.) This can be observed with a voltmeter or oscilloscope. If no activity at all can be seen on the host from attaching the device, it’s possible that D+ is not being pulled high.

If the device was seen, but enumeration didn’t complete successfully, then somehow the OS failed to associate the device with the appropriate driver. In this case, in Windows, it’s likely the Device Manager contains a device that has a “yellow bang” or other negative result; this indicates the device was seen but not identified. This is shown in Fig. 8.

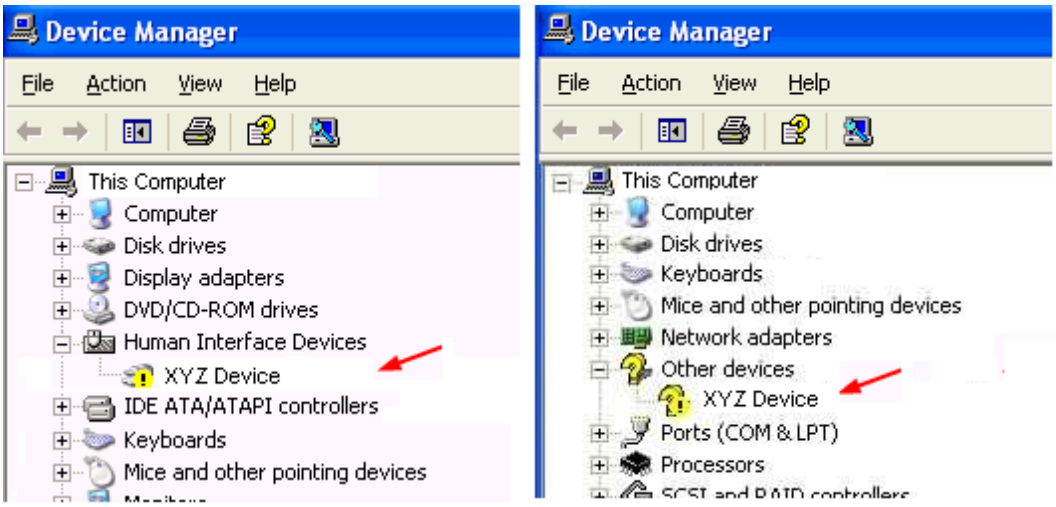


**Figure 9. Failed Enumeration Results in the Device Manager**

Perhaps the reason for the failed enumeration is that the OS prompted for a driver and the end user wasn't able to provide it – in which case the cause of the problem will be obvious. If not this, then it's possible the USB descriptors simply contained an error.

In unusual situations when using Windows, another possible explanation is that a previous failed attempt to enumerate this device resulted in its system registry entry to become corrupted. The application "Registry Editor" ("Regedit"), included in the Windows install, can be used to delete this device's entry; or the VID/PID of the device can be altered, and this will cause Windows to think it sees a new device.

If it's determined that the descriptors are in error, it's best to try to return to known good values, such as the defaults provided with the MSP430 USB API stacks. Then, work toward figuring out where the problem is between the two. (Errant descriptors should not be a problem if using TI's API stacks with the *MSP430 USB Descriptor Tool.*)

Another possible explanation is that the MSP430 software is being prevented from replying to the host's descriptor requests. Even if using the MSP430 USB API stacks with default descriptors, if the application in some way interferes with USB interrupts, the API may not be able to respond to USB requests.

If enumeration completed successfully, and the device still isn't operating as expected, the problem is likely at the application level. (Again, using the MSP430 USB API stacks, with the *MSP430 USB Descriptor Tool,* helps minimize the risk of this occurring by containing most of the USB functionality within the API.) If problems at this level can't be resolved through software debugging techniques at the host or MSP430, a USB bus protocol analyzer might be required.

# 8 USB Compliance and Certification Testing

Part of what has made USB successful is a well-written specification and widespread adherence to the standard, including USB certification. These ensure that end users see a consistently good experience with all USB devices.

This section is intended as a high-level guide to the compliance/certification process, and also contains MSP430-specific information. For a complete understanding of USB compliance testing and certification, please visit http://www.usb.org/developers/compliance/.

## 8.1 USB Compliance Testing

The USB Implementers Forum (USB-IF), which is the standards body that oversees USB, has specified a procedure that determines whether a device is compliant or not. Such testing is called *USB compliance testing.* This testing is done at the certification venues, and an OEM might choose to do internal testing prior to submission.

The compliance procedure uses a set of hardware, called the *goldtree*, on which the testing is performed. (It is referred to as a tree because a group of USB devices attached to each other using one or more hubs takes on the form of a tree structure.) It uses off-the-shelf USB products, with specific information to help obtain the exact model to be used. This makes the components easy to obtain.

There is also a Windows application provided by the USB-IF as part of the compliance procedure, called USBCV.

At the time of this writing (July 2009), there is an "official" procedure and a "candidate" procedure. Certification venues might use either one. They are shown in Table 2.



**Table 2. USB Compliance Procedures**

| | Procedure Document |
|---|---|
| **Official procedure** | USB Compliance Specification v1.3<br>http://www.usb.org/developers/compliance/peripheral_low/ |
| **"Candidate" procedure** | Gold Suite Test Procedure v0.91<br>http://compliance.usb.org/index.html, click on "Interoperability" |

The candidate may at some point become official or be replaced by another candidate. USB products are sometimes discontinued by their manufacturers, and so the USB-IF periodically updates the goldtree list, and sometimes the procedure as well.

## 8.2 USB Certification

*Certification* can essentially be thought of as "officially" passing compliance testing, such that the USB-IF recognizes it and grants the product certified status. This involves submitting the device to the USB-IF or one of its certified testhouses (Sec. 8.3. has more information).

In addition to passing the compliance test, the product must be submitted with a filled-out form called the *compliance checklist.* This nine-page document mostly contains a series of yes/no questions asking if the device adheres to various parts of the USB specification. Some of these questions pertain to characteristics controlled by the MSP430, and some pertain to system-level inputs.

To assist in completing this form, TI provides commentary in Table 3 below. The elements in the table correlate with the elements in the official checklist, v1.08 (dated November 2001).

Many of the comments make reference to the TI reference design, located in Sec. 9. Any deviation from this reference design should be cause for the designer to re-evaluate the checklist.

Similarly, many of the comments make reference to the MSP430 USB API stacks, as the stacks play a large role in the relevant behavior. However, since the system designer has full access to the API stack software, any changes made to the code in question could affect the API stack's ability to properly respond, making its compliance comments below no longer true.

**NOTE:** This commentary is meant only as assistance to the system designer in assessing compliance – TI assumes no responsibility for the values entered. Ultimately, the system designer must be the one to ensure the device adheres to all USB specifications.

**Table 3. Commentary for the USB Peripheral Compliance Checklist**

| Field ID | Comments |
|---|---|
| M3 | Yes, if the TI reference design is used. |
| M4 | |
| M5 | |
| M6 | Probably, if the TI reference design is used. However, there is a system dependency in these characteristics – the designer should take extra care ot ensure they are true. |
| M7 | |
| M8 | |
| M9 | |
| MLS1-4 | N/A |
| MFS1 | Yes, if the TI reference design is used. |
| MFS2 | This is dependent on the MSP430 application firmware. (However, the device couldn't enumerate on the host if this weren't true.) |
| MFS3 | Yes. |
| MT1 | Application-specific. |
| MT2 | Yes, if the TI reference design is used. |
| MUT1 | Application-specific. |
| MUT2 | Yes, if the TI reference design is used. |
| MUT3 | Yes, if the TI reference design is used. |
| D1 | Yes. |
| D2 | Yes. (It's derivation from VUSB, which is driven by VBUS, ensures this is true.) |
| D3 | Yes. (See $V_{LAUNCH}$ parameter in MSP430 datasheet.) |
| D4 | The pullup function is software-driven, and therefore the MSP430 application must ensure the pullup occurs within the time limit. (If using the MSP430 USB API stacks, this occurs when calling *USB_connect()*. ) |
| D5 | The figure referenced by this requirement, Fig. 7-29 in the USB specification, involves several requirements that might be affected by the MSP430 application, in particular how quickly the device enters USB suspend. The system designer should ensure these are true. |
| D6 | Yes. |
| D7 | |
| D8 | |
| D9 | |
| D10 | Yes. (However, entering suspend also involves reducing VBUS draw below 500uA, and this most likely involves the MSP430 application – see D11.) |
| D11 | The MSP430 immediately prompts the application with an interrupt that a suspend event has occurred; but the application must ensure that VBUS power draw is properly reduced within the time limit. |
| D12 | Yes. |
| D13 | |
| D14 | |
| D15 | Yes, if the MSP430 API stacks are used. |
| D16 | Yes. (MSP430 application must restore the system and be responsive.) |
| D17 | Yes. |
| D18 | Yes. |
| D19 | Yes. |
| D20 | Yes, if the TI reference design is used. |

| | |
|---|---|
| D21<br>D22<br>D23 | Yes, if the MSP430 API stacks are used. However, the MSP430 application has the ability to impede this – care must be taken. |
| D24 | Application-dependent. The certification process will reveal if this fails, and it's recommended the OEM perform this prior to submitting for certification. |
| D25 | Yes. |
| D26 | Yes. |
| LS1-4 | N/A |
| FS1<br>FS2<br>FS3 | Yes. |
| W1 | This is controlled by the MSP430 application. |
| W2 | Yes (if remote wakeup is used by the application.) |
| W3 | |
| W4 | Yes, if the MSP430 USB API stacks are used. (The API's remote wakeup function call will not send a remote wakeup signal if the host has not enabled it.) |
| P1 | Yes, if the TI reference design is used. (The diode prevents this from occurring.) |
| LP1<br>LP2<br>LP3<br>LP4<br>LP5<br>LP6<br>LP7<br>LP8<br>LP9 | From an MSP430 perspective, the answer to these questions is yes. But these are system-level questions and are therefore application-dependent. |
| HP1<br>HP2<br>HP3<br>HP4<br>HP5<br>HP6<br>HP7<br>HP8<br>HP9<br>HP10<br>HP11<br>HP12 | From an MSP430 perspective, the answer to these questions is yes. But these are system-level questions and are therefore application-dependent. |
| R1 | Yes. |
| R2 | Yes, because the reference design does not contain ferrite beads. |
| R3 | This is controlled by the MSP430 application. |
| R4 | This is controlled by the MSP430 application. |

## 8.3 How & When to Test: In-House, Testhouses, and "Plugfests"

It's always a good idea to perform USB compliance testing in-house prior to submitting a device for certication or going to market with a product. If the testing discovers an error, it can save expense and lost time down the road. Using TI's materials can greatly minimize the risk of failing certification, but the OEM's application built on TI's materials also has the ability to affect compliance, and so the silicon vendor's material is usually not a guarantee.

Certification must be performed through one of these methods:

- A testhouse certified by the USB-IF (http://www.usb.org/developers/compliance/labs/)
- A Compliance Workshop (also known as a "plugfest") sponsored by the USB-IF (http://www.usb.org/developers/events/compshop)

Some pros and cons of the two methods are shown in Table 4.

**Table 4. Pros and Cons of the Certification Methods**

| | |
|---|---|
| **Testhouse** | *Advantages*:<br>• Results can usually be obtained within a few weeks<br>• Anonymity of results; failures do not need to be reported to the USB-IF |
| | *Disadvantages*:<br>• A fee must be paid to the testhouse |
| **Plugfest** | *Advantages*:<br>• No cost to attend (other than travel costs, if applicable) |
| | *Disadvantages*:<br>• Plugfests are scheduled periodically; this might mean waiting weeks or months until the next one occurs.<br>• Plugfests are held regionally; travel expenses might be involved<br>• Plugfests are relatively public; a testsuite might even be run by an employee of your competitor |

# 9 TI Reference Design for USB Interface

TI recommends certain basic circuitry for USB interface. This is shown in Fig. 9.

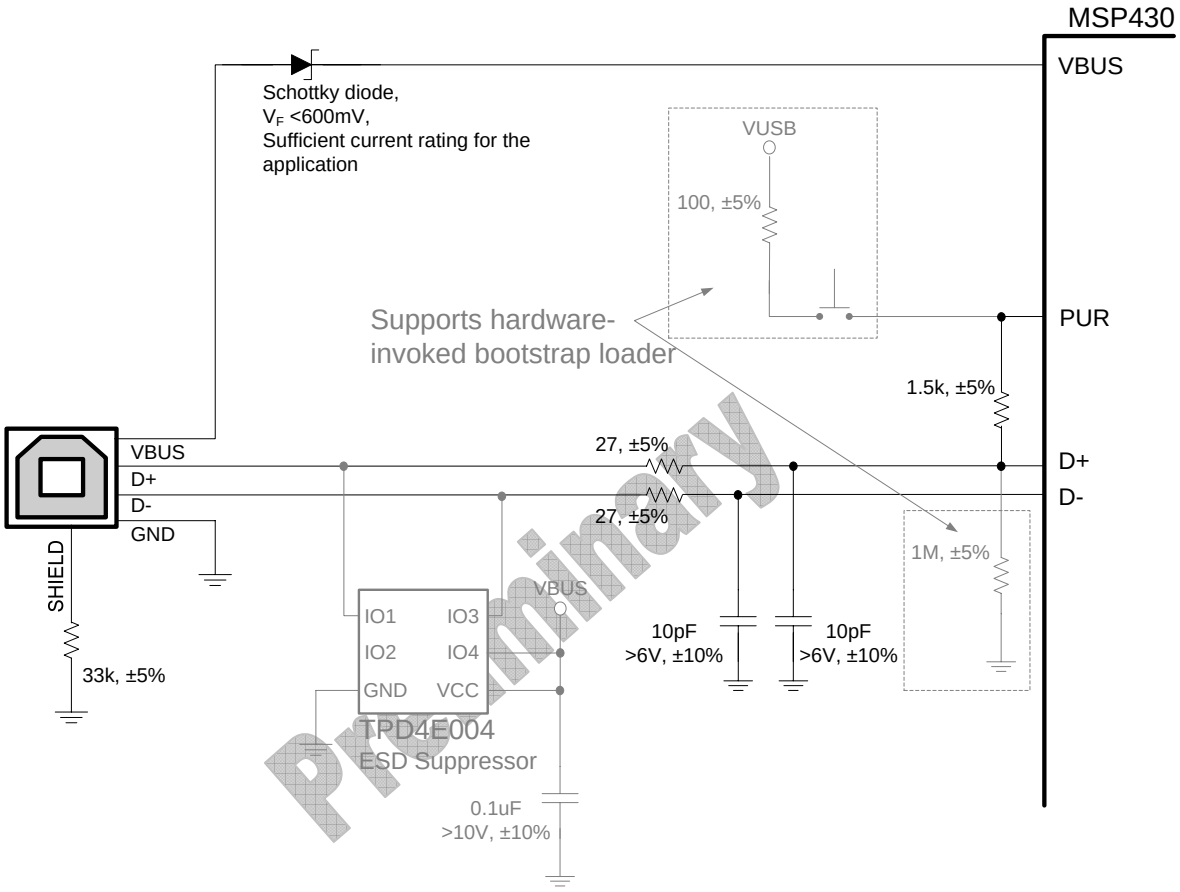


**Figure 10. TI USB Reference Design**

In the figure, black elements are always the recommendation, while gray elements are somewhat application-specific and provided in order to show a complete system picture.

ESD protection is optional. USB ports are often a vulnerable point, because it extends a conductive path to the outside of the enclosure.

The other gray areas relate to circuitry for hardware invocation of the bootstrap loader. This is covered in *USB Device Firmware Updates (DFU) on MSP430 MCUs.* The circuitry shown allows the PUR pin to be seen as pulled either high or low, immediately after a BOR reset event.

The elements in black can be changed by the system designer, but it should be noted that the compliance checklist assumes these elements are in place. Therefore, any deviation should be cause for the designer to re-evaluate the checklist.

# 10 Evaluation of the Device

At the time of this writing (July 2009), TI supports evaluation of the MSP430F552x devices with a sample kit. Contact the TI Product Information Center to obtain one. (http://www.ti.com/support)

# 11 References

- Information on MSP430's USB devices
  - *MSP430F5xx Family User's Guide (slau208d)*
  - *MSP430F5xx USB Module User's Guide (slau284) (will be integrated with the User's Guide by September 2009)*
  - *MSP430F552x/1x Preliminary Datasheet (slas590)*
- Find these tools on the MSP430F5529 product page, at http://www.ti.com/msp430:
  - *MSP430 CDC/HID API Stack software, with Programmer's Guides*
  - *MSP430 USB Descriptor Tool*
  - *MSP430 Windows HID API, with Programmer's Guide*
- Information on USB device firmware updates, also on the MSP430F5529 product page:
  - *USB Device Firmware Updates (DFU) on MSP430 MCUs*
  - *USB Device Firmware Update (DFU) Demo*
  - *USB Device Firmware Update (DFU) Windows Starter Project* (coming August 2009)
  - *MSP430 Memory Programming User's Guide (slau265)* (will be updated for USB by September 2009)
- http://www.usb.org
- http://www.ti.com/msp430