Application Note

USB Design with MSPM0 MCUs

ABSTRACT

This guide offers a high-level introduction for users seeking to design USB-based devices using MSPM0 microcontrollers (MCUs). This document highlights the hardware and software resources available from TI MSPM0, explains the built-in USB features of MSPM0 devices, and outlines the tools that simplify development.

This guide covers:

- Overview of USB systems
- The MSPM0 USB hardware module and available MCU families
- USB hardware design recommendations
- Outline of MSPM0 USB software developer's kit and supporting systems
- A hardware reference design for USB devices based on the MSPM0 MCU

1 USB Makes a Complex System Appear Simple

1.1 Why Has USB Been so Successful?

USB is one of the most widely used industry standards today. The simplicity of use and streamlined experience make this an essential tool for many users. There are also several other benefits including:

- Affordability: The low cost has made USB attractive to users, which has driven mass adoption
- Widespread compatibility: The ubiquity of USB made it a standard feature in many products.
- Extended use cases: Beyond main functionalities, USB has also enabled innovative use cases such as:
  - Power delivery: Providing power to charge small gadgets and devices.
  - Peripheral connectivity: Enabling connections for various peripherals such as keyboards, mice, and so on.
  - Data storage: Offering various storage options, from flash drivers to external hard drives.
- Reliability: Establishing a stable connection between USB devices without errors and failures.

Despite the simplicity for end users, the USB internal design can be complex. This complexity arises from the need to provide a fast, reliable data bus that automates common behaviors and tolerates hot plugging, which requires layers of protocol.

1.2 Why Does USB Look Simple?

USB hides complexity from the users, but the developers often see what lies underneath. Compared to other protocols such as UART, SPI, or I2C, USB requires far more handling for data transfer. USB takes more effort to send data than writing to a register.

On-chip USB modules help reduce some of this complexity, but the modules cannot eliminate the complexity. Significant portions of the USB stack still require software to manage the intricacies of USB communication. A well-designed software can insulate the application developer from many of these complexities, but they still face key challenges such as:

- Handling device connection and disconnection events
- Maintaining stable communication and reliable data transfer even in challenging conditions, such as a busy or unreliable bus, to prevent data loss.
- Developing strategies to manage and support multiple host operating systems.

These considerations are crucial for developers to verify the USB-based applications are reliable, efficient, and user-friendly.

2 MSPM0 USB Silicon

MSPM0 microcontroller families include an integrated USB module. The module is compatible with the latest MSPM0-SDK, which includes the development package called Tiny USB.

Although it is feasible to implement USB communication through bit-banging, this approach is not preferred for full-speed operations and likely consumes much of the capacity of the processor. The majority of the USB applications rely on the on-chip USB module, which provides a more efficient and reliable means of managing USB communication.

2.1 How MSPM0 Devices are Documented

The MSPM0 device documentation is divided into three locations for any given MSPM0 device family members:

- Technical Reference Manual: Contains all the architectural information for the MSPM0 MCU family. All USB-equipped devices can be found in the *MSPM0 G-Series 80MHz Microcontrollers Technical Reference Manual*.
- Datasheet: The datasheet contains all the parameters and details specific to this device family's members, providing a detailed understanding of features and capabilities.
- SDK User Guide: The SDK user's guide contains an overview of the different drivers and example programs that can be used as a starting point for creating software for new projects.

The combination of these documents provides a comprehensive understanding of the MSPM0 device family.

2.2 MSPM0 USB Module

The MSPM0 USB module offers the following features:

- USB 2.0 Full-speed (12Mbps) operation in host and device modes, as well as low-speed (1.5Mbps) operation in host mode
- Complies with USB-IF certification standards
- Four transfer types: Control, Interrupt, Bulk, and isochronous
- Sixteen endpoints
  - One dedicated control IN endpoint and one dedicated control OUT endpoint
  - Seven configurable IN endpoints and seven configurable OUT endpoints
- Two kB dedicated endpoint memory
- Dedicated hardware triggers for DMA support
- Support for USB-based device firmware update (DFU) through bootstrap loader (BSL)
- The USBFS module includes an integrated full-speed PHY
- Support Suspend and Resume Signaling feature

3 MSPM0 USB Hardware Design

3.1 Block Diagram

The USB block diagram is shown in Figure 3-1.

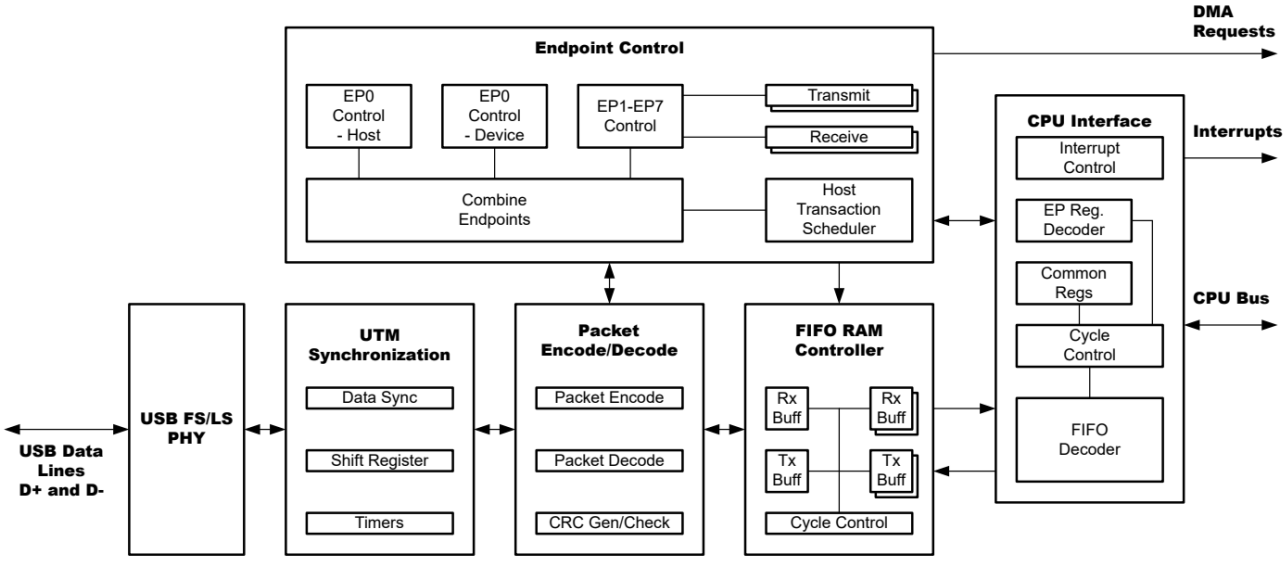

Figure 3-1. Block Diagram of USB Peripheral in MSPM0

3.2 USB Mode of Operation

The USB IP supports both device mode and host mode in a static configuration.

For USB-C type connectors, the device and the host mode need to be indicated by pull-up/down resistors on the CC1/CC2 lines on the PCB system, which limits the USB IP for the dynamic configuration of the mode. Therefore, USB IP does not support on-the-go (OTG) switching between device and host mode.

3.2.1 USB Device Mode: Bus Powered

In a bus-powered USB application, the device receives power from the USB host through the USB connector. This power delivery method is limited to USB device applications, which only operate while connected, as the USB host is required to provide power to the bus. A typical example of a bus-powered USB application is a USB mouse or keyboard. These devices only operate when the devices are connected to the computer, and the devices receive power supply from the USB host.

The diagram below shows the system components for a bus-powered application. This system requires an external 3.3V LDO to supply additional onboard components, as the voltage on VBUS is 5V. In this mode, no USB connection detection circuit is required as the SoC only operates when it is powered.

3.2.2 USB Device Mode: Self-Powered

Another option for power on a USB device can be *self-powering*. This means that on the device, there is some external power supply apart from the USB power, which provides power to the chip and the connection when necessary.

A typical example of such a mode can be an audio or storage device, where power is required despite being disconnected. These designs require additional power management when compared to the bus-powered mode, with some form of primary power regulator and sequencing depending on the complexity of the power tree.

Oftentimes, self-powered devices can also be considered *hybrid*, meaning they either use the bus power when connected or charge a battery when connected. For this reason, oftentimes a VBUS monitoring circuit is necessary to detect a connection and switch to the power path provided by the bus. This can be accomplished by something as simple as a resistor divider hooked up to an ADC pin on the MSPM0G5187, which toggles a switch on the power path.

3.2.3 USB Host Mode Power Considerations

Unlike the MSP430, the MSPM0Gx is capable of operating as a host in a USB design. While the majority of complexity comes from the software design of these applications, there are additional hardware requirements when it comes to designing a host.

Power is one of the main needs when designing a host device, as the host *must provide 5V VBUS* power to connected devices, with up to 500mA required for USB 2.0 full speed. This oftentimes means pairing the device with a dedicated power source, such as a separate switching power supply. As USB devices are often bus-powered, ensuring that the host can support the power requirements of any device is critical for consistent operation.

Protection must be considered as there is no guarantee that the device being connected is designed correctly and can introduce an overcurrent or overvoltage scenario. Integrating protection circuitry and a current sense resistor on the power line assists in verifying that the device is not exposed to such over-power situations.

3.2.4 ESD Considerations

Hosts require an additional degree of robustness when it comes to ESD protection on data lines and VBUS, as multiple, unknown, and potentially dubious devices can be causing ESD events downstream. Typically, a higher ESD rating is recommended, around 8kV contact and 15kV air.

3.2.5 Layout Considerations

Similar to USB devices, matching trace lengths on the data lines to verify the two lines are synchronized is critical.

Designers must be careful to keep USB data traces clean, with minimal vias (in pairs if one must be used), routed away from high-speed signals. The DP/DM signals must have a clean and complete ground plane for reference in a layer behind or above the signals. The differential resistance on these signals must be as close to 90Ω as possible, which can be calculated using most PCBCAD software.

3.3 USB Clock Implementation

USB imposes strict timing requirements, but those requirements differ materially between a USB host and a USB device due to their respective roles in bus timing and synchronization. At a high level, the host is the timing authority on the bus, while devices are made to synchronize themselves to the timing of the host.

A USB host is responsible for generating Start-of-Frame (SOF) packets at a precise 1kHz rate for full-speed devices. SOF packets define the global time base of the USB bus, and thus the host's clock must meet USB accuracy requirements independently, without relying on any external reference from the bus. Due to the strict timing requirements of a host, a high-accuracy external oscillator is required. This stable clock can be used as the reference input to the MSPM0's FLL or PLL, which synthesizes the higher-frequency clocks required by the USB module. The internal LFOSC is insufficient for this role, as the frequency error and temperature drift exceed what is acceptable for long-term SOF timing accuracy.

On the other hand, USB devices do not generate their own SOF packets and thus are not the timing master. This allows the device to derive its timing from the host-generated SOF packets. The MSPM0 utilizes the SOF packet as an input reference for the USBFLL, which will be used as the clock source. Devices do not typically require an extremely precise external oscillator as the host determines their timing, allowing for the use of less accurate, internal clocks. The USB peripheral on MSPM0 contains a clock unique to the peripheral, *USBCLK*, which operates at 60MHz. The USBCLK needs a precise reference clock, which can either be supplied by the SYSPLL block or the USB-specific USBFLL, which has lower power consumption through the FLL protocol as opposed to the PLL. More information on the clock tree can be found in section 2.3 of the Technical Reference Manual.

3.3.1 Selecting a Clock Source

The biggest consideration when selecting a clock source is precision. The USB specification requires the clock have a tolerance of 2500ppm. Sources outside of this specification can cause decreases in consistent performance and fail to verify USB compliance.

There are three choices to source the reference clock:

Table 3-1. Clock Source Differences

Source	Frequency Range	When to Use?
External Clock Source	4 to 48MHz crystal oscillator, PLL to higher frequencies	Crystal oscillators have the best precision and USB compliance, allowing the user to meet USB specification compliance.
Internal Clock Source	4 to 32MHz, PLL to higher frequencies	Not typically recommended as the internal clock is often outside of 2500ppm, usable for prototyping.
USBFLL	48 to 60MHz	No space for an external crystal, as the device leverages SOF packets for synchronization. Space and cost-constrained designs, however, require an active USB connection for clock synchronization.

If using an external crystal, make sure that the Crystal Oscillator guide is followed for maximum performance.

3.3.2 Selecting a Clock Frequency

A wide range of frequencies is possible, as the MSPM0's USBFLL clock source leverages SOF packets for synchronization. Each MSPM0 header file contains predefined constants for either SYSPLL or USBFLL sources, with additional configuration being available on SYSPLLCLK1, which allows divisors as high as 32.

While 48MHz is required for USB functionality, there are multiple ways to achieve this goal. Oftentimes, using a 48MHz crystal may be out of the scope of a design, as they tend to be more expensive, have higher EMI, and more strict design requirements, despite the advantages of signal integrity and clock simplicity.

Oftentimes, designers employ multiple HFXT crystals in a PLL (therefore, SYSPLLCLK), which allows the user to employ a combination of lower frequency crystals, such as six 8MHz crystals in a PLL, or three 16MHz crystals in a PLL. These have the advantage of being lower cost and lower EMI. However, increases clock tree complexity, as well as layout complexity.

In many cases, utilizing the USBFLL clock source is a reliable option for devices. With no additional crystal (saving space and money), it is a great fit for space and cost-constrained devices. However, it must be considered that there is higher latency, as the FLL requires multiple SOF frames to lock and synchronize to the host. This is a great option for bus-powered devices, which only operate while connected to USB.

3.4 Example Implementation

Figure 3-2 is a pseudo-schematic based on the evaluation kit, LP-MSPM0G5187:

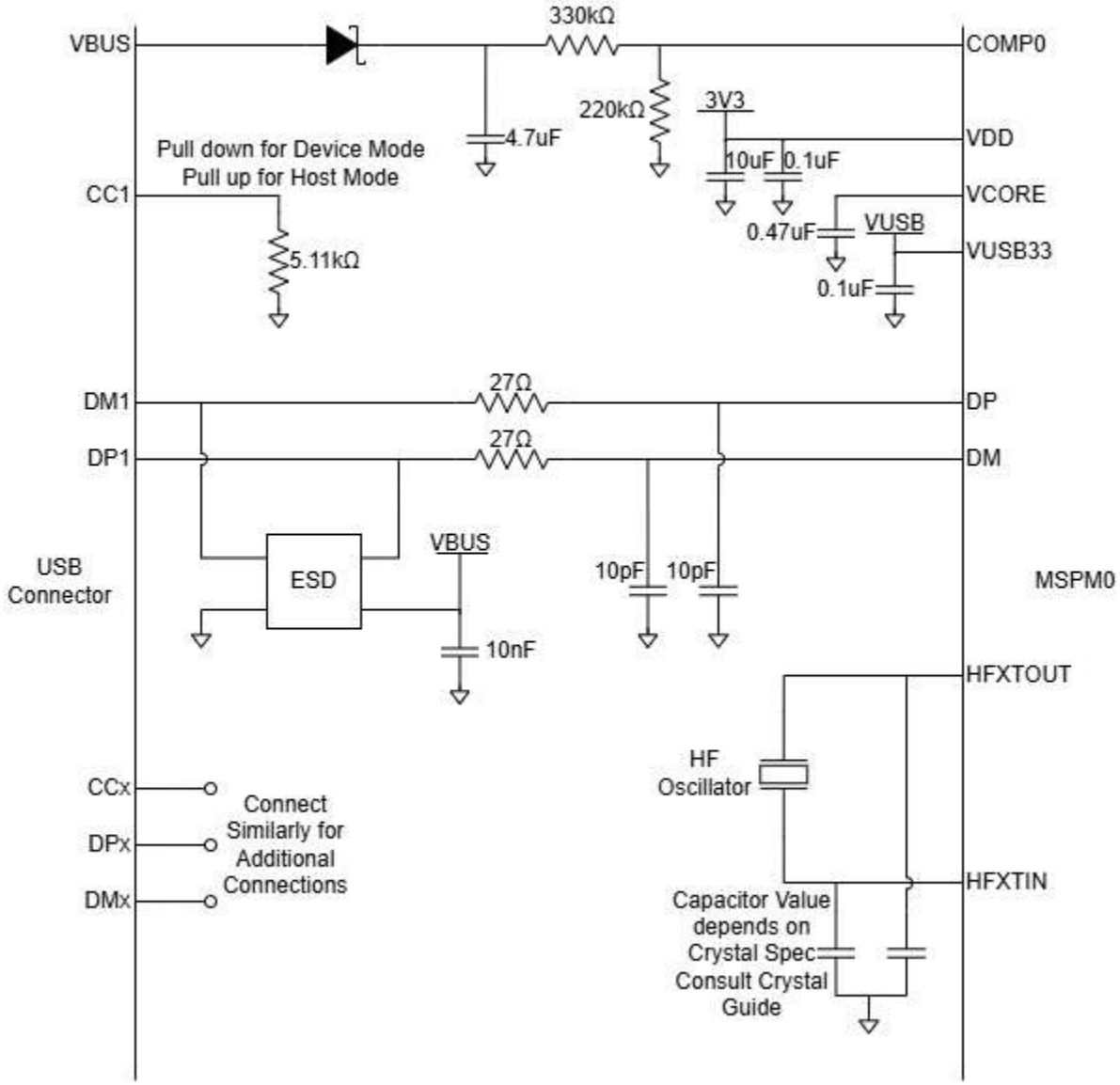

Figure 3-2. Example Device Configuration Reference Design

Figure 3-2 while simplified, has multiple important features to note and verify implementation within the circuit.

- VDD, VCORE, and VUSB33 are left disconnected in the previous example. Follow the datasheet of the device to verify the correct decoupling capacitor values are used.
- D+ Must be pulled up on DP on the device side, as this is the signal that informs the host when a new connection has been established.
- ESD is critical in USB devices, especially those commonly handled by humans such as mice, keyboards, and headsets. The device in the Figure 3-2 is TDP4E05U06.
- External clock sources connected to HFXTOUT and HFXTIN may be a crystal, resonator, or external clock source. Follow the crystal guide in Section 3.3.1 if using a crystal.

4 Software Overview

4.1 USB Stacks: Features

The TinyUSB Library is the foundation for developing USB devices using the MSPM0SDK. This API supports four of the most common USB device classes:

- Communications Device Class (CDC): A USB device class used for serial communication devices. CDC allows USB devices to emulate traditional serial ports (COM), enabling data exchange between host and device.
- Human Interface Device class (HID): A USB device class designed for user input devices such as keyboards, mice, and game controllers. HID devices use a standardized protocol that allows for driverless installation on most operating systems.
- Audio Device Class (UAC): A USB device class for transmitting and receiving digital audio data between a host and device. UAC allows USB devices to appear as standard audio IO devices to the host.
- Mass Storage Class (MSC): A USB device class that allows a device to appear as an external storage drive to the host operating system, enabling standard file system operations.

These classes provide a good selection for general-purpose use. See section *Choosing a Device Class* for a discussion of how to select an interface.

Features of the API include:

- Cross-platform support
- Host and Device stack
- Multiple device classes
- Clean and readable codebase
- MIT licensed
- Minimal resource requirements
- Multi-configuration support

There are several examples in the MSPM0SDK, which at the time of writing features the following examples:

Table 4-1. Examples Provided in MSPM0SDK

CDC Examples	HID Examples	MSC Examples	UAC Examples	General Purpose Examples
cdc_acm_uart	hid_cdc_composite_ti	msc_dual_lun	uac_microphone_i2s	device_billboard
cdc_dual_ports	device_hid_composite	sd_card_bridge	uac2_headset	
hid_cdc_composite_ti	device_hid_generic_inout	msc_file_explorer	uac2_speaker_fb	
billboard_cdc	hid_keyboard_ti		audio_test	
	hid_mouse_ti			
	hid_multiple_interface			

These examples are available for rapid testing and development using the MSPM0G5187 LaunchPad. Please see Section 5 for more information on getting started with prototyping.

4.2 SysConfig Descriptor Tool

TI provides simple configuration of descriptors using *SysConfig.* This tool allows for configuring many peripherals on MSPM0 devices, and for USB-enabled MSPM0 devices, there is a section titled *TinyUSB* which allows for configuration and generation of descriptors, rather than needing to code them manually.

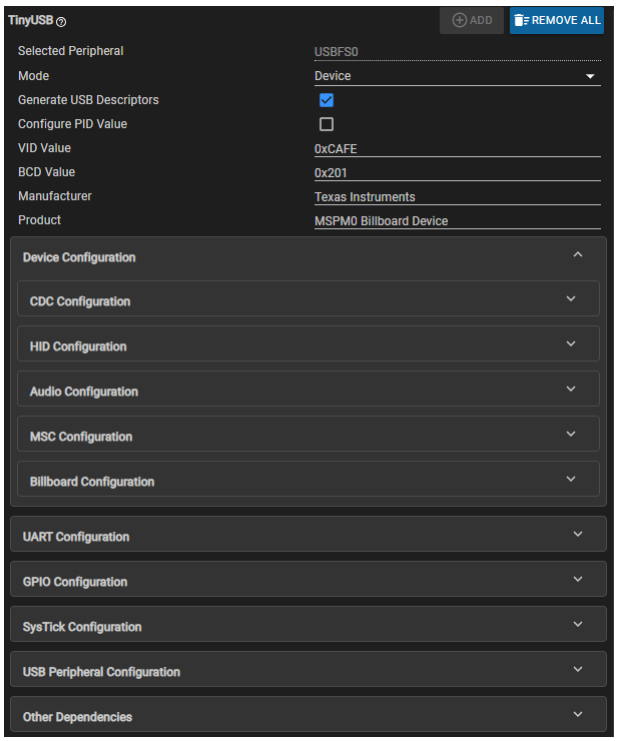

Figure 4-1. Screen Capture of the SysConfig Editor provided through Code Composer Studio

This tool allows for the configuration of the following features:

- Mode: Device/Host
- VID value
- BCD Value
- Product and Manufacturer codes
- Device class configuration
  - RX FIFO size (CDC ONLY)
  - TX FIFO size (CDC ONLY)
  - Endpoint transfer buffer size
  - Enabling speaker feedback, encoding, and decoding (UAC ONLY)
- Default and Alternate Billboard Identifiers
- Configure UART instance and pinmux selection
  - Clock Source
  - Clock Divider
  - Baud Rate
  - Word Length
  - Parity
  - Stop Bits
  - HW Flow control
  - DMA Configuration
  - Interrupt Configuration
- Configure GPIO Dependencies
  - Direction
  - Initial value
  - Pinmux

  - Interrupts
  - Internal Resistors
- Configure Systick dependencies
- Configure USB Clock Source

Utilizing SysConfig to generate USB descriptors can be a big help, as writing descriptors can be tedious and cause additional mistakes. Failures resulting from incorrect descriptors tend to be obfuscated, and tracking down those mistakes can cause a significant increase in debugging time.

SysConfig generates reliable descriptors, on the first try, for any combination of CDC, HID, UAC, and MSC interfaces with significantly less effort.

Think of the tool as building the USB interfaces an application interacts with. This is the first step to develop an MSPM0 TinyUSB project.

4.3 Selecting a Device Class

One of the first steps of creating a USB design is to select the correct device class, as this selection confines a device to a certain set of characteristics, as listed in Table 4-2.

Table 4-2. Comparison of Four Supported Device Classes

Characteristic	CDC (Communications Device Class)	HID (Human Interface Device Class)	MSC (Mass Storage Class)	UAC (Audio Device Class)
The interface generated on the host	Virtual COM port	A human interface device	Storage volume	An audio device
Industry expertise for this interface	COM ports are common in the industry; widely supported and well understood	Unlike COM ports or storage volumes, HID interfaces are somewhat USB-specific and less well-known in the industry	Storage volumes are common in the industry; widely supported and well understood	UAC devices have been in use for nearly two decades; widely supported and well understood
Installation on the host	Windows PCs must undergo a device installation process that requires end-user interaction. (1) Admin rights on this Windows PC are required Despite the actual binary files already existing in Windows, the user must supply an INF file	Loads silently in most operating systems – simply begins working. No driver files required	Loads silently in most operating systems – simply begins working. No driver files required	Class-compliant loading on most hosts, no installation required
How the end user interacts with it	An application on the host that interfaces with COM ports The application can be custom, or any existing application that uses COM ports	A custom application on the host that interfaces with HID devices	The device mounts a storage volume onto the system; applications read and write files on the volume. The application can be custom, or any application that reads or writes files	Audio is streamed to or from the device with the UAC interface
Driver certification needs	Unless the INF file is WHQL certified (signed), Windows reports that the driver is 'uncertified	No *uncertified* message is generated	• No *uncertified* message is generated	• No *uncertified* message is generated
Code footprint and complexity	Small code footprint (4K to 6K) Simple architecture	Small code footprint (4K to 6K) Simple architecture	Larger code footprint (8K to 15K) Requires a file system, increasing cost, size, and complexity.	Even larger footprint Requires interfaces Isochronous transfers, which increase complexity
Throughput	Fast (hundreds of KB/sec) Uses bulk USB transfers.	Slower (64 KB/sec) Uses interrupt USB transfers.	Fast (hundreds of KB/sec) Uses bulk USB transfers.	Dependent on sample rate and bit depth Typically hundreds of KB/sec

Table 4-2. Comparison of Four Supported Device Classes (continued)

Characteristic	CDC (Communications Device Class)	HID (Human Interface Device Class)	MSC (Mass Storage Class)	UAC (Audio Device Class)
Good for point-to-point communication between the host and the device	Yes	Yes	No	Yes
Good for bulk data transfer	Yes	No	Yes	Yes (audio)

4.3.1 Example Process for Deciding on a USB Device Class

Sometimes the choice of device class is clear. In other cases, the application can be considered general-purpose, giving the developer options. Although there are many ways to approach this decision, users can use Figure 4-2 to select a device class.

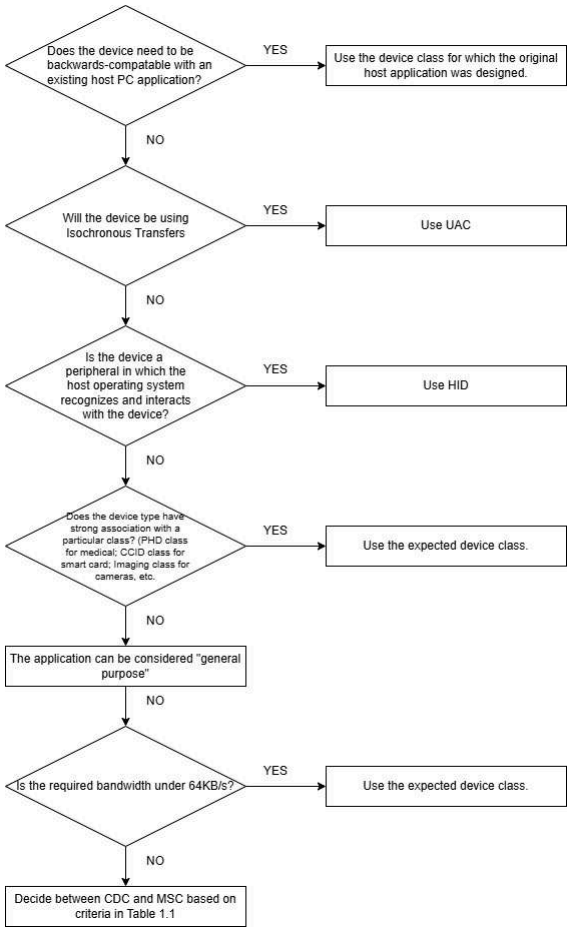

Figure 4-2. Example Decision Tree for Selecting the Correct Device Class

4.4 How to Select a Vendor ID (VID) and Product ID (PID)

A common question about USB is how to select the VID and PID. When a USB device is attached to a host, the host asks for USB descriptors. These tell the host the nature of the device and the capabilities of the device.

Included in the descriptors are the 16-bit VID and PID values. The VID is associated with a particular vendor and OEM, and a PID is associated with a product sold by that vendor.

For example, if vendor *Vendor1* sells the first USB product (*Product1*), the vendor obtains a VID, which is now associated with the company, and the vendor needs to then choose a PID to associate with *Product1*. When the vendor later releases *Product2*, the vendor uses the same VID but now should use a new PID. It is the responsibility of the vendor to make sure the vendor does not duplicate PIDs, which can result in conflicts in the field.

Therefore, a unique combination of a VID and PID allows a USB host to discern one USB product type from another. If the VID and PID of *Product1* and *Product2* are the same, and a host in the field encounters both products, conflicts can result from the host confusing the two products and loading an inappropriate driver. As a general rule, if devices have any differences in USB descriptors, the devices must have different PIDs.

4.4.1 Choosing and Obtaining VID and PID

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 $5000 annually, and the latter costs $3500 for a two-year license. (See http://www.usb.org/developers/vendor/ for more information.)

Unlike the predecessor MSP430 devices, MSP no longer runs a VID sharing program where customers can go online and request a PID unique to them and use this under TI's VID. The customer is free to register a VID with the USB-IF as mentioned above and use that VID and associated PIDs with our devices.

4.4.2 Using VIDs and PIDs During Development

Having a unique VID and PID pair on a USB device is important to prevent conflicts. A given USB host stores information about the driver requirements of the USB device after the first encounter with a given VID and PID. It must be able to assume that any subsequent devices with the same VID and PID require the same host driver setup. Therefore, once released to market, a VID and PID of a product must not be changed.

During development, however, the VID and PID can sometimes need to change as the developer arrives at the final USB descriptor set. The developer must prevent conflicts on the host machine being used. This can be done either by using a new PID value any time the USB descriptors change, or the original PID can be used, but the device must be uninstalled from the system and reinstalled. See the USB API Programmer's Guide in the USB Developers Package for more information.

4.5 TinyUSB API Programmer's Guide and Examples

The TinyUSB users guide, found in the SDK, includes more information on developing software using TinyUSB, as well as expanding on many of the examples mentioned in Table 4-1, as well as breaking down the USB descriptor tool in Figure 4-1.

Additionally, there are numerous examples online and deeper dives into the API through TinyUSB's website. This includes a number of additional definitions, software concepts, and resources related to application design.

5 Getting Started: Evaluating MSPM0 USB

The LP-MSPM0G5187 evaluation kit is equipped with all of the required breakouts to begin evaluating and developing both host and device designs.

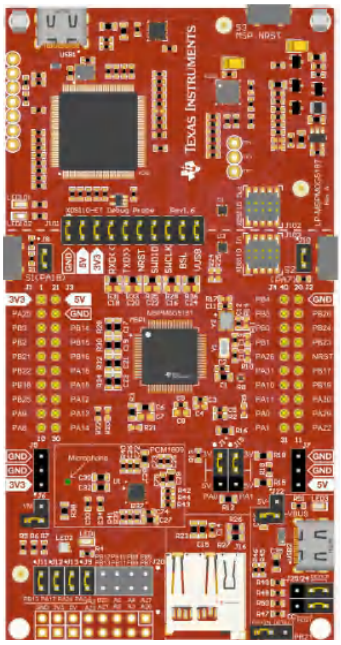

Figure 5-1. LP-MSPM0G5187 Evaluation Kit

The evaluation kit contains multiple jumpers, which allow for debugging numerous hardware and software features:

- Microphone and I2S-based audio ADC for UAC devices
- A microSD slot for MSC applications
- Two LEDs (One RGB-capable LED)
- Three buttons
- 40+ accessible pins for testing general devices.
- Two USB-C connectors, one for debugging, one for MSPM0 connection
- Onboard debug probe for programming, debugging, and EnergyTrace™ technology.

Rapid prototyping is simple, thanks to 40-pin expansion headers on the LaunchPad development kit that connect to a wide range of available BoosterPack™ plug-in modules. You can quickly add features like wireless, displays, sensors, and much more. Design a BoosterPack plug-in module or select among many already available from TI and elsewhere. The 40-pin interface is compatible with any 20-pin BoosterPack plug-in module that complies with the standard.

TI's Code Composer Studio IDE (CCS) is the primary development environment for designing and testing using LaunchPads. The USB API examples are provided through the MSPM0SDK, and downloading the latest version is generally a good idea to make sure the most recent examples are available.

6 Summary

The MSPM0 family of USB-enabled devices are capable of supporting multiple types of applications, including audio, human interface, and communications devices, alongside with mass storage hosts. The USB 2.0 full-speed module has up to 16 endpoints, dedicated endpoint RAM, DMA triggers, an integrated full-speed PHY, and DFU/BSL support. For more information on each of these modules, consult the technical reference manual and datasheet relevant to the USB part. MSPM0's USB peripheral coupled with the CCS development environment and TinyUSB architecture allows for simplicity of development for rapid development of USB applications when coupled with the hardware guidances throughout this app note. There are many important considerations in these designs, including USB-C mode indication, self and bus powered power architecture considerations, and routing and layout rules. While the clocking module within the MSPM0 is capable of handling device applications, careful consideration is required to maintain compliance with the USB protocol.

7 References

1. Texas Instruments, Starting a USB Design Using MSP430 MCUs, application note.
2. Texas Instruments, *LP-MSPM0G5187 LaunchPad*, development kit.
3. Texas Instruments, *MSPM0 G-Series 80MHz Microcontrollers*, technical reference manual.
4. Texas Instruments, *MSPM0 Academy*, resource explorer.
5. TinyUSB, *TinyUSB Concepts*, webpage.

8 USB Glossary

1. Bulk Transfers: One of four data transfer types on the USB bus. Bulk transfers are designed for moving high volumes of data. They are capable of using any free bandwidth on the bus (that is, bandwidth not already used by the other transfer types). This allows them to achieve the highest data rates, but they are given no reserved bandwidth, so on a busy bus, bulk transfers can receive small bandwidth or experience high latency. Transfer types are determined by the choice of USB interface type; for example, CDC and MSC interfaces use bulk transfers.
2. Composite USB Device: A physical USB device (one USB connector) that contains more than one USB interface – for example, two CDC interfaces or CDC+HID. The host enumerates each interface as a separate logical entity.
3. Control Transfers: One of four data transfer types on the USB bus. Control transfers handle the administrative tasks of setting up the connection, like reporting USB descriptors. The host also sends other USB device requests, and the device responds using control transfers. There is a USB endpoint dedicated to these transfers: endpoint 0 (EP0).
4. Device Class: A defined USB protocol for a particular class of devices. Common device classes include the Communications Device Class (CDC), Human Interface Device (HID) class, USB Audio Class (UAC), and Mass Storage Class (MSC).
    a. CDC: A USB device class used for serial communication devices. CDC allows USB devices to emulate traditional serial ports (COM), enabling data exchange between host and device.
    b. HID: A USB device class designed for user input devices like keyboards, mice, and game controllers. HID devices use a standardized protocol that allows for driverless installation on most operating systems.
    c. UAC: A USB device class for transmitting and receiving digital audio data between a host and device. UAC allows USB devices to appear as standard audio IO devices to the host.
    d. MSC: A USB device class that allows a device to appear as an external storage drive to the host operating system, enabling standard file system operations.
5. Device Installation: The first time a USB device is enumerated, the host may perform one-time functions to install the device. For example, Windows records information about the device in the system registry, using the device's VID and PID as an index. In subsequent enumerations, the host draws from the registry for much of its information about the device. Device installation may be silent (mostly invisible to the end user) or, in the case of CDC on Windows, may require user action.
6. Endpoint: The end of a pipe. This acts as a *mailbox* on the USB device for that pipe. A device usually has more than one active endpoint. When the host communicates on the bus, it first identifies the physical USB device, then the endpoint number within that device that it wishes to speak to. Endpoints are assigned specific functions according to the USB interfaces that were created. HID/MSC each use one IN and one OUT endpoint, while CDC uses two IN and one OUT endpoint. In the MSP430 API stacks, endpoint management is fully automated by the Descriptor Tool.
7. Enumeration: The process by which a host interrogates a physical USB device to determine what it is and loads an appropriate driver so that the host application can interface with it. Enumeration happens every time the device is attached.
8. Interrupt Transfers: One of the four USB data transfer types. Interrupt transfers are designed for latency, bandwidth, and delivery. However, the bandwidth is limited to only a single USB packet (64 bytes for full-speed USB) per frame (1 ms). Transfer types are determined by the choice of USB interface type; for example, HID interfaces use interrupt transfers.
9. Isochronous Transfers: One of the four USB data transfer types. Isochronous transfers provide guaranteed latency and bandwidth but not delivery. That is, if error checking shows corrupted data, the attempt is not retried. This type is intended for streaming audio and video -- applications in which a retry would result in an interruption and thus be more noticeable to the user than simply missing the packet.
10. INF (*.inf) file: A text-based file required during any USB device installation on Windows, allowing Windows to associate the device with a particular driver. For some device classes, Windows contains the INF internally, allowing for a silent device installation. For CDC, Windows prompts the end user for the INF file.
11. Pipe: A single line of communication between host and device. Pipes are either IN (into the host) or OUT (out of the host). They are characterized by a particular transfer type (for example, bulk or interrupt).

12. Descriptor: Data structures that the USB device provides to the host during enumeration, describing the device's capabilities, configurations, interfaces, and endpoints. Common descriptors include: Device, configuration, interface, and endpoint descriptors.
13. Full-Speed USB: USB operating at 12Mbps, the standard speed for USB 1.1 and optional for USB 2.0 devices.
14. Host: The USB controller that manages the bus and initiates all data transfers. Typically a PC, but can also be embedded systems with USB host capability.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025