TrueTask USB / MCCI USB DataPump Technical Overview

MCCI Corporation, June 2022

Introduction

TrueTask USB is MCCI's USB software platform for embedded systems. It's based on the MCCI USB DataPump ("DataPump"), MCCI's portable embedded USB framework. This article gives a technical overview of the DataPump for engineers and technical managers.

The DataPump combines comprehensive USB device support with the industry's most thoroughly verified embedded USB host and USB Type-C® stack. Its modular, reentrant design allows it to be deployed in many ways. It may be used as a stand-alone stack; or it may be used to augment, supplement, upgrade, or replace existing USB stacks, while maintaining compatibility with the existing stack.

The modular architecture of the DataPump allows it to scale from the simplest operating environments to the most complex. In the device stack and the Type C stack, all memory allocation happens during initialization. The host stack can be configured (for deeply embedded systems) to allocate all memory during initialization, or to allocate and free memory for device instance data while the stack is running.

In addition, by careful abstraction and layering, the components can be used independently. In a dual-role environment, the DataPump device component can be used with the DataPump host stack or with the native OS host stack.

All MCCI software is designed to serve as a platform for further development. APIs and data structures are stable from version to version, which means that system software built on the DataPump can be coded once, then reused as needed.

The careful design and implementation allow the DataPump to be readily deployed without source modification across a wide range of CPU architectures, USB host, device, and port controllers, and operating systems. A single code base supports a range of use cases, from USB 1.1 full speed to USB4® products, including Dual Role, Type C, OTG, and USB emulation applications.

The DataPump is the result of over 25 years of continuous development and refinement. It is accompanied by a large suite of test tools, regression tests, and development applications. This

ecosystem makes it the strongest engineering solution for supporting multiple USB products from a common source base.

TrueTask USB is a packaging of select components of the DataPump in a form that is easy to use in conjunction with select real-time operating systems. The components are pre-ported and integrated with the RTOS and the target SOC, and delivered as a software component. TrueTask USB inherits the stable APIs of the DataPump, and so insulates customer software from variations between hardware platforms, and variations between TrueTask USB releases.

MCCI's broad protocol support, standards leadership, and technical excellence are recognized by high-volume consumer product makers around the world, and have made the DataPump the "gold standard" for trouble-free applications of USB across a product line or throughout a corporation.

DataPump Components

The DataPump has the following components.

DataPump Port Manager

The DataPump Port Manager consists of the policy managers, finite state machines, and low-level drivers to support advanced Type C port implementations. The reference port controller architecture is the *USB Type-C Port Controller Specification* V2.0, but MCCI's flexible and robust implementation allows a variety of register models and a variety of connection strategies, including direct register access, I2C, SPI, or other indirect means of register access. The finite state machines used for port control are directly traceable to the reference finite state machines in chapter 4 of the *USB Type-C Specification* Release 2.0 and the USB Power Delivery specifications.

The reference port controller is the Faraday FTCPD210.

Many customers prefer a simple Type C Dual-Role Port implementation, which adds host/device flexibility without introducing the full scope and complexity (and interoperability test requirements) of power-delivery communications. MCCI's base implementation therefore supports the full range of Type C operation, without introducing Power Delivery communication over the CC line. However, the architecture directly scales to a full PD implementation for systems where this is desired.

DataPump Device Stack

The DataPump Device Stack consists of the Device Framework, Device Controller Drivers, support libraries, and device class protocols.

The Device Framework provides functionality common to all USB devices, including standard command support, suspend/resume, link-power-management support, composite device support, multiple device modes, and Microsoft OS Descriptor support.

Low speed, full speed, high speed, USB 3.2 SuperSpeed Gen1 and USB 3.2 Enhanced SuperSpeed Gen2 are fully supported including 96k Isochronous endpoints, as is USB tunneling over USB4. Using the DataPump Device stack and a suitable host controller, the MCCI Model 3411 USB Loopback Device can sustain 8 gigabits per second IN and OUT concurrently.

Device Controller Drivers

Device Controller Drivers (DCDs) provide a common, portable, low-level API to the Device Framework. This API is optimized for high throughput, zero-copy DMA operations.

Over thirty DCDs are available. Notable DCDs include:

- Synopsys DesignWare USB 3.2 "XDCI". This DCD supports the Synopsys SuperSpeed device IP. Support is included for streams and USB 3.2 Gen2. Xilinx FPGAs are supported, as are various Sitara chips from Texas Instruments.
- Synopsys DesignWare USB 2.0 IP. This DCD supports the industry standard Synopsys high-speed device and OTG IP. Versions 2.6 and later are supported. Depending on the version, PIO, DMA and scatter-gather DMA are supported. Normal USB and HSIC are supported. LPM is supported if supported by the hardware. This is the IP block used in USB-capable STM32 devices, as well as in the ST Telemaco automotive SOCs.
- Synopsys ChipIdea USB2.0 IP. This DCD supports the IP block used in the NXP i.MX6, the NXP LPC1850, and many other popular SOCs.
- Nvidia Jetson USB 3 gen1 Device Controller.
- Faraday FOTG210 device controller.
- Renesas R-Car3 SS DCI USB 3 gen1 device controller.
- Renesas r8a7795 USB3. This DCD supports the USB 3 superspeed device IP block used in the Renesas R-Car 3 SoC family
- Renesas '597. This DCD supports the Renesas high-speed OTG kit part (for lower volume designs and prototyping) and IP (for SOC applications).
- Mentor Inventra (currently marketed by SiFive). This DCD supports the popular Mentor MUSBMHDRC high-speed OTG core. A variety of PHY architectures allow support for common external PHYs. LPM is supported if supported by the hardware.
- Cadence USBHS-OTG-MPD. USB 2.0 device core with advanced DMA, and multi-device host controller for dual-role and USB On-The-Go applications supporting hubs.
- Cadence USBHS-OTG-SD. USB 2.0 device core with simple DMA. Simplified host controller supports dual-role device and USB On-The-Go applications that don't involve hubs or compound devices.
- Cadence USB 3.0 Device Controller, with or without SuperSpeed support.

DataPump Device Class Protocols

Device Class Protocol modules provide the support for device classes. MCCI offers over 20 device class protocols. All protocols can be freely combined to form composite multi-function devices, and to create sophisticated multi-mode devices.

- Audio Class 1.0 and 2.0, including asynchronous feedback endpoint support
- CDC (Communications Device Class) 1.2 Wireless Mobile Communication subclass (WMC) for multi-function 2.5G and 3G handsets
    - CDC WMC Abstract Control Model (ACM), for traditional modems and modem emulation
    - CDC WMC Device Management
    - CDC WMC OBEX (Object Exchange)
    - CDC WMC MDLM (debug ports, vendor specific functions, etc.)
- Abstract NIC family of virtual NICs over USB, with support for:
    - CDC ECM (Ethernet Control Model), for low throughput Ethernet-like networking, targeting cable modems and network bridges
    - CDC EEM (Ethernet Emulation Model), targeting accessories and local peripheral networking
    - CDC NCM (Network Control Model), for high throughput Ethernet-like networking
    - Microsoft RNDIS, for networking applications targeting Microsoft Windows systems
- Device Firmware Update (DFU) 1.1, for firmware update over USB
- Generic class (Vendor Specific Class), for implementing USB device behavior outside the DataPump
- Human Interface Device (HID) 1.1
- Mass Storage Bulk-Only Transport. This protocol is used both for read/write storage applications, and for CD-ROM.
- MCCI Loopback, for test and performance evaluation
- MCCI Virtual Serial Port (VSP), for migrating RS-232 devices to USB
- Mobile Broadband Interface Module (MBIM) 1.0
- Network Control Model (NCM) 1.0
- Still Image Class
- USB Attached SCSI (UAS) 1.0
- Video Class (UVC) 1.1 and 1.5, including multiple streams per function, bulk and isochronous data transfers, and Microsoft extensions.

DataPump Device Class Applications

Building on the Still Image Class implementation, MCCI offers PTP, PictBridge and Media Transport Protocol implementations. These implementations include a database suitable for removable media, and the software for automatic media file discovery and indexing.

DataPump Host Stack

The DataPump host stack, like the device stack, consists of the core functionality ("USBD"), host controller drivers (HCDs), and class drivers.

DataPump USBD

The DataPump USBD manages and operates the USB bus, including translating USBDI requests into the simpler commands used by host controller drivers, pipe management, default pipe management, bandwidth allocation, abstract scheduling for periodic traffic, and class driver management and matching.

Low speed, full speed, high speed, USB 3.2 Gen 1, and Gen 2 are fully supported, including 96k Isochronous transfers. Speeds of 8 Gbps IN + 8 Gbps OUT can be sustained concurrently on gen2 platforms with capable hardware.

Host Controller Drivers

Host Controller Drivers (HCDs) implement a common, portable, low-level API that is used by the USBD to access the physical Host Controller Interface (HCI). Like the DCD API, the HCD API is optimized for high throughput, zero copy DMA operations.

The DataPump HCD architecture has several unusual features. HCDs can run "stand alone", without a USBD. MCCI's "HCDVT" tool uses this feature to perform unit-testing of HCDs and HCIs without the limitations imposed by USBD.

Here are some of the HCDs available for the DataPump host stack:

- xHCI. This HCD supports any host controller conforming to the xHCI 0.96, 1.0 or 1.1 specifications. It has been qualified with Intel, Renesas, Fresco Logic, TI, Asmedia, Marvell, and Synopsys DesignWare cores.
- EHCI. This HCD supports any host controller conforming to the EHCI specification. It has been qualified with Intel, NXP, FTDI, Faraday, and Renesas platforms. It supports use with a separate OHCI companion controller, or with transaction translators embedded in the root hub.
- OHCI. This HCD supports any host controller conforming to the OHCI specification. It has been qualified with Renesas platforms.

- Synopsys DesignWare USB 2.0 IP. This HCD supports the Synopsys DesignWare high-speed USB host IP running in host mode, either as part of an OTG core or in a fixed host configuration. Depending on the HCI version, PIO, buffer DMA and scatter-gather DMA are supported. Transaction Translators are fully supported, to allow use of low- and full-speed devices behind high-speed hubs. High-bandwidth isochronous and interrupt pipes are fully supported, which allows use of standard PC webcams. It also includes support for the DWC_OTG core configured as a High-Speed Inter-Chip (HSIC) USB host. This HCD has been qualified on the Raspberry Pi with Windows 10 (32 and 64-bit), and on a variety of STM32 processors.
- Mentor Inventra IP. This HCD supports the Mentor Inventra core running as a dedicated USB host, or as part of an OTG device. PIO and external DMA are supported.
- Renesas '597. This HCD supports the Renesas '597 running as a dedicated host, or as an OTG device in host mode. It also supports the equivalent Renesas IP as part of an SOC. PIO or external DMA is supported.

A key part of the HCD architecture is the "HCDkit" library. This library provides a framework for "typical" host controller interfaces, including such functionality as a simulated root hub device and routines for managing scheduling of periodic transfers for HCIs that require software assistance.

DataPump Host Class Drivers

MCCI offers the following class drivers for the host stack.

- Hub driver – this is the only mandatory driver for the system. In restricted resource environments, the maximum hub depth can be pre-configured. This driver is normally configured to support transaction translators, however in full-speed only systems or HSIC systems with no embedded transaction translators, this support may be omitted.
- Composite driver – supports composite devices by dividing the device up into multiple virtual functions, which then match regular class drivers. This is only used when the host stack is configured as a native or hybrid stack.
- Mass Storage – these drivers support normal mass storage devices, for use when the host stack is configured as a native or hybrid stack.
- HID keyboard and mouse
- Abstract NIC (ECM, NCM, EEM)
- CDC ACM driver (for serial ports and modems) – in addition to connecting to any CDC ACM modem, supports Microchip and Holtek CDC ACM USB-serial adapters, and Linux USB device gadget ACM ports.
- Silicon Labs CP210x driver (for virtual serial ports)
- Generic Driver
- Audio Class 1
- Audio Class 2

- Video Class 1.0, 1.1 and 1.5
- ASIX USB 2.0 and USB 3 USB-to-Ethernet adapters
- Null Driver (for OS emulation)

Common Libraries

The common libraries for the DataPump environment include a rich set of primitives that simplify development.

- The DataPump object system provides a consistent behavior for structures that are registered with the system. Objects are named and discoverable. A standard message system allows objects to implement, inherit, and delegate abstract messaging services.
- Abstract memory allocators model memory pools (which may be implemented by the operating system or by library code in the DataPump working with a fixed amount of pre-allocated memory). Working with preallocated memory allows for "zero surprise" design; working with the operating system's allocators allows for a variable memory footprint that grows based on usage profile.
- Safe memory and string functions allow for runtime buffer overrun prevention.
- The abstract annunciator system allows multiple clients to register with event producers inside the stack, without being aware of the details of how the events are plumbed.
- A comprehensive debug logging system allows for a variety of approaches to runtime sequence-of-event recording.
- A compact, portable UTF8 library allows handling of Unicode strings in the common situations that arise for USB applications.

Operating System Integrations

MCCI offers a variety of pre-packaged operating system support packages, including:

- os/none – this is an MCCI nano-kernel that provides exactly and only the basic services needed for running the DataPump "on bare iron" without an operating system. It essentially provides an interrupt abstraction layer, hardware initialization services, and an event loop.
- Windows kernel – the DataPump host and device stacks can be embedded into WDM drivers. The host stack is provided with wrappers that completely emulate the standard Windows USB host stack, allowing use of standard Microsoft and third-party class drivers. The device stack uses MCCI proprietary APIs to expose the upper edges of the class protocols to user-mode applications.
- Green Hills INTEGRITY OS – special IODevices provide mediated access to registers from the kernel. The rest of the stack (HCDs, DCDs, protocols) run in user mode as part of the "TrueTask USB server". Client libraries link into customer address spaces to provide access to USB resources via INTEGRITY Connections.

- FreeRTOS, SafeRTOS, etc. – the DataPump is integrated as a task, communicating with clients using shared memory or classic device driver techniques.
- RTX – the DataPump is integrated as a task, communicating with clients using shared memory or classic device driver techniques. USB-UART drivers are integrated as CMSIS drivers to allow USB UARTs to be used interchangeably with traditional UARTs. Other drivers are integrated using MCCI's reentrant techniques, with APIs that mirror the standard RTX power management APIs.
- WindRiver VxWorks – the DataPump is integrated as a task, communicating with clients using a DataPump event queue and VxWorks semaphore.
- Linux kernel – the DataPump host and device stacks can be embedded into Linux kernel drivers, as loadable modules. The host stack is provided with wrappers that completely emulate the standard Linux USB host stack, while also allowing use of high-performance zero-copy MCCI drivers. The device stack is used with the MCCI DataPump native device class protocol implementations.
- Azure OS (ThreadX) – the DataPump is integrated as a service task, communicating with clients using shared memory or classic device driver techniques.
- MQX – the integration is like that for ThreadX
- µItron – the DataPump is integrated as a service task, communicating with clients using messages and event flags.
- RTEMS – the integration is integrated as a service task, communicating with clients using shared memory or classic device driver techniques.
- OSE – the DataPump is integrated as one or more OSE tasks, which communicate by clients using OSE signals (messages).

MCCI can readily port the DataPump to proprietary environments.

MMUs and multiple address spaces are represented by associating a "handle" with each buffer pointer. Handles are opaque to DataPump code, but are used by the operating system layer to record such things as MDLs (for Windows kernel-mode drivers), or signal buffer pointers (for OSE). Buffers without handles are treated as internal kernel-mode buffers. The handle approach also allows for easy migration to IOMMUs in hardened environments.

CPU and Compiler Support

MCCI supports a wide range of 32-bit and 64-bit CPUs, operating in either big- or little-endian mode. Endianness may be different for register accesses and DMA than for normal CPU operation, and requires no C compiler extensions.

Supported architectures include:

- ARM (32-bit, Thumb, and AARCH64)
- Intel (32-bit and 64-bit)

- RISC-V (32-bit and 64-bit)
- MIPS (32-bit and 64-bit)
- Sparc
- ARC
- Tensilica

Supported SOC implementations include:

- STM32
- Microchip SAMD
- Nvidia Jetson Tegra X1
- Renesas R-Car2 and R-Car3
- TI OMAP6 and Sitara
- NXP i.MX6 and i.MX7
- NXP PLC1850
- Xilinx Zynq UltraScale+ FPGAs (reference platform: ZCU106 or Ultra96-V2 boards)

Validation Tools

To help MCCI and MCCI customers to validate systems built with the DataPump, MCCI has created several software and hardware tools, including:

- Catena systems, including low-, full-, and high-speed test devices and hosts.
- The MCCI 3141 USB4™ Switch, and the MCCI 2101, 3101, 3102, and 3201 USB Connection Exercisers, used for automated plug/unplug testing of devices.
- The MCCI 3501 Type-C SuperMUTT, designed for host system API testing.
- The MCCI 3411 Gen2 Loopback Device, which enables automated performance testing at up to 16 Gbps aggregate bandwidth (8 Gbps IN and 8 Gbps OUT) as well as 96k Isochronous testing when using USB 3.2 gen2 speeds.

Build System and Build Tools

The DataPump is delivered with a complete build system that works on Windows or Linux, 32 bit or 64 bits. The build system includes the following tools.

- "mccimake" is a version of BSD make, as enhanced by MCCI for cross-platform Makefile portability
- "usbrc" is the USB resource compiler. It verifies, maps, and translates high-level descriptions of USB devices into the concrete descriptors and matching endpoint assignments for the DCI being used in the target system.

The build system treats the source tree as "read only", and integrates readily with OS-specific build systems. For example, the Windows version of the USB 3 host stack is built by creating the DataPump libraries, then building drivers using build.exe in the normal way.

Documentation

- User Guides
- Application Notes
- Test and Verification Procedures
- Detailed technical implementation documentation, in the form of CHM files.
- Source code, which averages more than one line of documentation commentary per line of code (as measured by the software metric tool CCCC).

Architectural Overview

The simplest DataPump device architecture is shown in Figure 1. The USB device hardware is modeled as a USB Device Controller Interface (DCI), which is controlled by a Device Controller Driver (DCD). Because USB transceiver control is often quite tricky in embedded systems, the transceiver (PHY) is explicitly modeled by a transceiver API. (Modeling the PHY also allows simple coordination of "car kit", battery charging, and other features that multiplex non-USB signaling over the USB connector.) The DCD is in turn controlled by the DataPump Device Framework, which in principle supplies all the chapter 9 functionality. (Some DCIs provide some of the chapter 9 functionality in hardware; in such cases, the DataPump Device Framework simply shadows the operations that are being performed by the DCI.)

The DCD does not implement any higher-level USB knowledge; that functionality lives in the DataPump Device Framework and the class protocol modules. DataPump Device Framework functions do not change based on the function being performed by the device. Rather, the function and configuration of the stack is driven (at the chapter 9 level) by the descriptors; and the provision of class protocols is driven by the "application initialization tables".

Modules in the DataPump programming environment are coded using a set of primitives and APIs that are independent of the native operating system. A simple object dictionary allows internal protocol instances (and external clients) to dynamically discover objects within the DataPump environment. A general-purpose messaging system allows external clients to send control messages to any of the indicated API points within the DataPump. This allows native OS management facilities (for example) to control the transceiver, to start/stop the USB device subsystem, to register for and to receive notifications from the any of the layers within the DataPump, all without having detailed knowledge of any of the internal data structures of the DataPump, and without having direct access to the memory being used by the DataPump.

Any USB device must have a static set of descriptors. The DataPump allows these descriptors to vary over time, but it requires that the set of device and configuration descriptors be set before

beginning device operation. While initializing, the DataPump then builds data structures that model the device's topology, by examining the descriptors. Finally, it scans the descriptors and notifies each of the provisioned protocol modules, which in turn bind themselves to configurations, interfaces and endpoints, using common code provided by the DataPump framework.

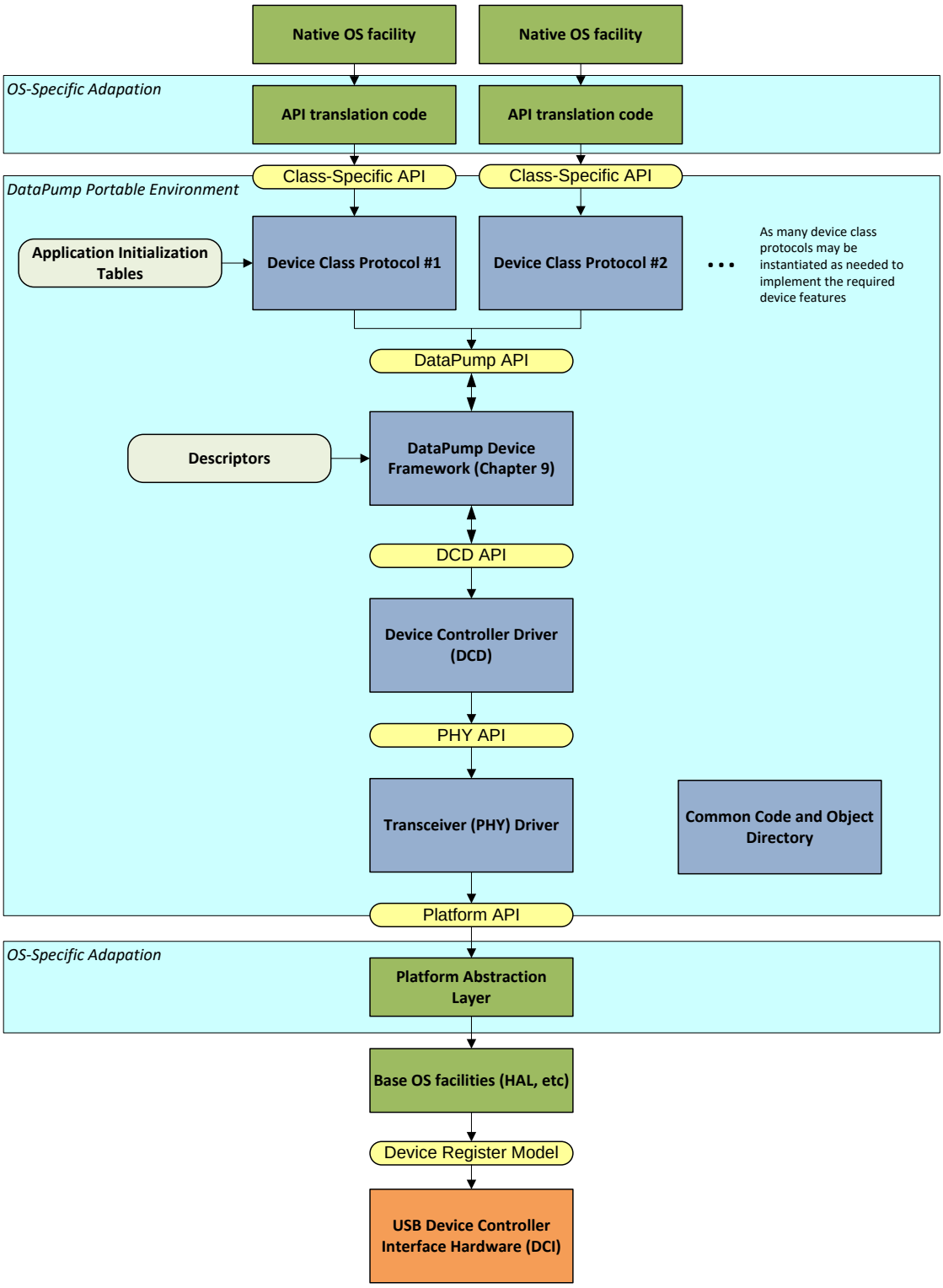

Figure 1. DataPump Device Architecture

(In most cases, the actual "examination of descriptors" is done at compile time by a special tool called "USBRC" – but the architecture of the DataPump also allows this to be done at run time,

if desired. Using USBRC allows a much larger number of consistency checks to be done at compile time, and allows for complex and correct mapping onto the limited hardware found in most SOCs. After parsing the description of the device, USBRC generates the actual descriptors to be used, and the initialization code, as C source code.)

After initialization, device class protocols work primarily with UDATASTREAM structures. These structures model active endpoints, abstracting alternate settings and alternate configurations that may allow the application to deal with logical data paths. For example, a bulk endpoint has a different maximum packet size in full speed mode than it does in high-speed. UDATASTREAMs allow the class protocol modules (and external clients) to deal with devices of arbitrary complexity in an extremely simple and intuitive way.

High-volume device data transfer uses the same techniques used with USB host stacks. The client prepares a buffer, or a scatter-gather list, then builds a USB data request (called a UBUFQE). The client then submits the UBUFQE for asynchronous processing. When the request is complete, the DataPump calls a client-supplied callback function to complete processing. The structure and APIs are optimized for DMA-based DCIs. Because of the short code paths, and the copy-free architecture, data transfers normally operate at bus speed.

The native operating system is modeled through an abstraction layer. This layer has two primary responsibilities: to model interrupts for the DCD, and to provide an "event dispatch" facility. In addition, it provides bindings for the normal OS services (such as allocating and freeing memory).

The DataPump is essentially singly threaded, although it operates well in symmetric multiprocessing (SMP) environments such as the Windows kernel. USB devices, at the USB level, have limited concurrency available, and memory interlocks are expensive; so not much concurrency is sacrificed. DataPump computations are event-driven and asynchronous, rather than a blocking and synchronous, so if a second CPU offers load to the USB while the first CPU is working, the work is simply queued, and the second CPU is immediately released for other work.

The DataPump device stack may be used in conjunction with the TrueTask USB host stack. Figure 2 shows the DataPump configured to support USB On-The-Go.

The diagram has been slightly reorganized compared to Figure 1, to make it more concrete. We show the DataPump configured for WMC and Vendor-Specific device functions (either as a composite or multi-configuration device); and the host stack configured to support HID and Mass Storage class devices.

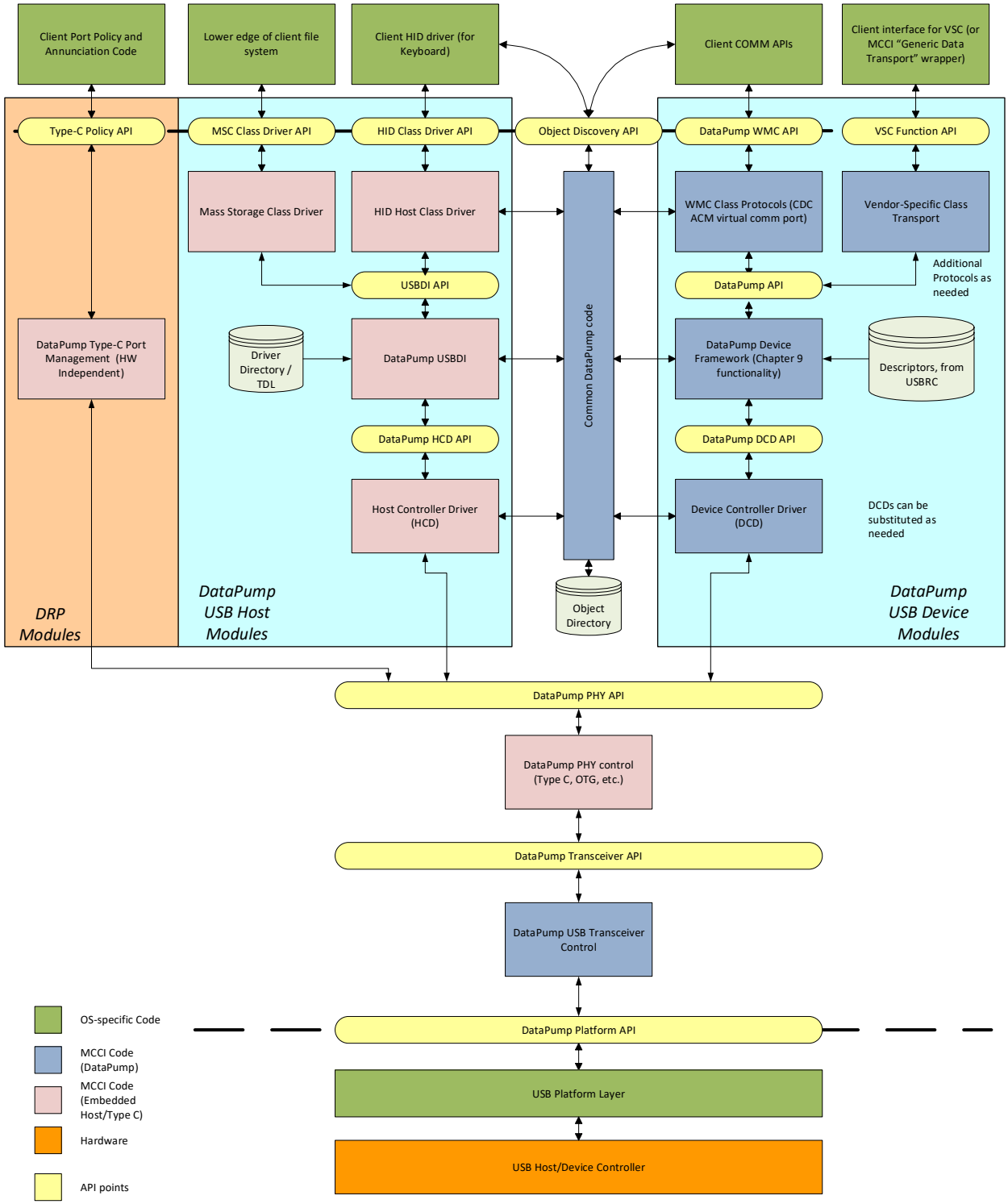

Figure 2. DataPump Type-C Dual-Role Architecture

DataPump Coding Conventions

MCCI's coding style has several features that contribute to portability.

- ANSI C 89 with selected C 99 features
- No use of global or static variables (global and static constants are, of course, allowed)
- No conditional compiles in C files (except for the usual checked/free configuration)
- Dynamic initialization and configuration
- Library-based
- Abstract types for portability
- Avoidance of type casts
- Type cloaking allows DataPump code to be integrated cleanly with any operating system.
- API versioning discipline provides stable APIs. Ensures strict compatibility for client source code as APIs evolve to meet new requirements.
- Minimal duplication of code

Licensing and re-deployment options

The DataPump product was designed as an OEM ("white label") software package, to allow our customers the most flexibility. It can be licensed in several forms.

A full source license is available, which gives licensees the most flexibility in adapting the DataPump to their needs. This approach may also simplify debugging.

The DataPump can also be licensed as header files plus libraries. The DataPump is not configured at compile time, so this license gives full functionality. Source licensees who need to sublicense the DataPump can also take advantage of this mode of operation for their sublicensees.

In certain configurations, the DataPump can be delivered as a fully pre-compiled executable. For example, when used as a Windows-compatible host stack, the DataPump is fully compiled and pre-configured. The DataPump device stack can be similarly pre-configured.

Modes of Integration

When integrating the DataPump into an existing system, MCCI offers the following approaches:

Native stack

Configured in this mode, the DataPump serves as a complete USB subsystem. Clients use the MCCI class protocols and APIs directly. This approach is the most efficient.

Emulation stack

Configured in this mode, the DataPump provides core USB functionality, but class protocols are implemented by client code outside the DataPump, typically enabling existing OS-native class drivers to be used without changes. The most typical example of this is using the DataPump in the Windows kernel for USB 3 host support.

Hybrid

When configured as a hybrid stack, the DataPump uses MCCI class protocol and class driver modules for certain key functions, and uses client code for the remaining functions. An example might be to use the MCCI NCM class driver for high performance zero-copy scatter/gather support, and to use OS-native drivers for other, less performance-critical functions.

Special Applications

Microsoft OS Descriptors

The DataPump device stack fully supports the Microsoft OS descriptors that facilitate driverless installation on Windows 10 and 11.

Device Firmware Update

The DataPump device stack includes standard support for the Device Firmware Update ("DFU") protocol. This allows firmware update using standard tools on Windows, Linux and macOS. Vendor-specific firmware update can also be implemented using CDC ACM comm ports, the vendor-specific class, or mass storage.

The DataPump host stack allows firmware update over Mass Storage. In mass storage update, the user attaches a properly formatted thumb drive; the application notices the drive and copies the firmware.

When combined with MCCI's trusted bootloader, ec25519 code signing and reliable system update can be implemented from a variety of sources.

Network support (Abstract NIC API)

MCCI's support for network drivers is extensive. The API upper edge allows zero copy, gather on write for very fast throughput. In addition, the same API is exported by host and device implementations allowing use of a single integration for dual-role products. The same API is used for NCM, EEM, ECM and RNDIS (device only) support. It's also used with the ASIX USB 2 88772A/B/C 100Base-T Ethernet devices, and the ASIX USB 3 88179 gigabit Ethernet devices.

CarPlay®/MirrorLink®

MCCI's high-performance NCM support, combined with the DataPump's flexible implementation style, makes it easy to implement CarPlay and MirrorLink hosts and devices. In addition to supporting NCM, MCCI also offers the Audio and vendor-specific drivers needed for a complete solution.

About MCCI

MCCI Corporation (www.mcci.com) is the USB software provider of choice to the world's leading companies. The MCCI USB DataPump device stack and TrueTask USB host stacks are the most comprehensive stacks available, covering the full spectrum of USB applications. In addition to its embedded software for USB, MCCI offers a comprehensive range of USB host class drivers for Windows and macOS, and manufactures specialized test equipment for USB development. MCCI is also a leader in open-source software and hardware development for LoRaWAN® technology long-range wireless IoT networks in the United States, and is the primary corporate sponsor for The Things Network New York. Founded in 1995, MCCI is privately held. It has development offices in Ithaca, NY, New York City, Chennai, India, and additional presence in Tokyo, Seoul, and Taipei.

Contact: sales@mcci.com Tel: +1-607-277-1029 x105

Twitter: @MCCI