Myriad™ 2 Development Kit (MDK)

MDK USB Overview

v1.1

Intel Movidius
2200 Mission College Blvd
M/S SC11-201
Santa Clara, CA 95054
http://www.movidius.com/

Revision History

Date	Version	Description
November 2017	1.1	Added section 4.3 Vendor specific class performance Added section 4.4 Host driver performance for mass storage
October 2017	1.0	Updated Table 1 Added section 3.2.2 Selecting host or device mode Added section 3.3.1 USB Clock configuration Added section 3.3.2 USB PHY controller initialization Added section 3.3.3 DataPump stack initialization Updated Table 2 Added section 7 Reference Documentation

Table of Contents

1 Introduction

This document gives an overview of the USB support in Myriad™ 2 Movidius® Development Kit (MDK) and references the further documentation.

2 Overview

The USB driver stack in MDK for Myriad™ 2 is implemented using MCCI DataPump USB Device Driver stack.

MCCI USB DataPump is a multi-layered embedded USB software framework which provides a total solution for designing high-performance USB peripheral devices. It is suitable to develop application, protocol, and/or hardware interface software to create a specific USB device.

MDK port of DataPump is using RTEMS OS. At least one RTEMS thread is created in the DataPump. This thread is used for receiving and dispatching DataPump events. Optionally, when debug build is used, another thread is used to receive debug events and print debug information.

3 MDK integration of MCCI USB DataPump

3.1 USB DataPump Device Stack configuration

The DataPump USB Device core environment has been ported to Myriad™ 2 and integrated into MDK. Its main components:

- Platform Port Layer
- USB Device Controller Driver (DCD)
- USB Device Framework

On the top of the DataPump core environment the following specific USB Device Class Protocols are supported:

- USB Video Class (UVC) Protocol
- USB Human Interface Device Class (HID) Protocol
- USB Mass Storage Class (MSC) Protocol
- USB Device Firmware Upgrade Class (DFU) Protocol
- USB Vendor Specific Class (VSC) Protocol
- USB Communication Device Class (CDC) Protocol

Validated operation modes:

- HighSpeed (USB2)
- SuperSpeed (USB3)

3.1.1 Application-MDK-DataPump relationship

Since the direct use of the DataPump's API is the optimal choice for application to interface with, there is no MDK wrapper around it. It means the developers program directly against MDK and DataPump APIs and then link with the DataPump libraries.

Figure 1 illustrates the relationship between the Applications, DataPump and MDK.

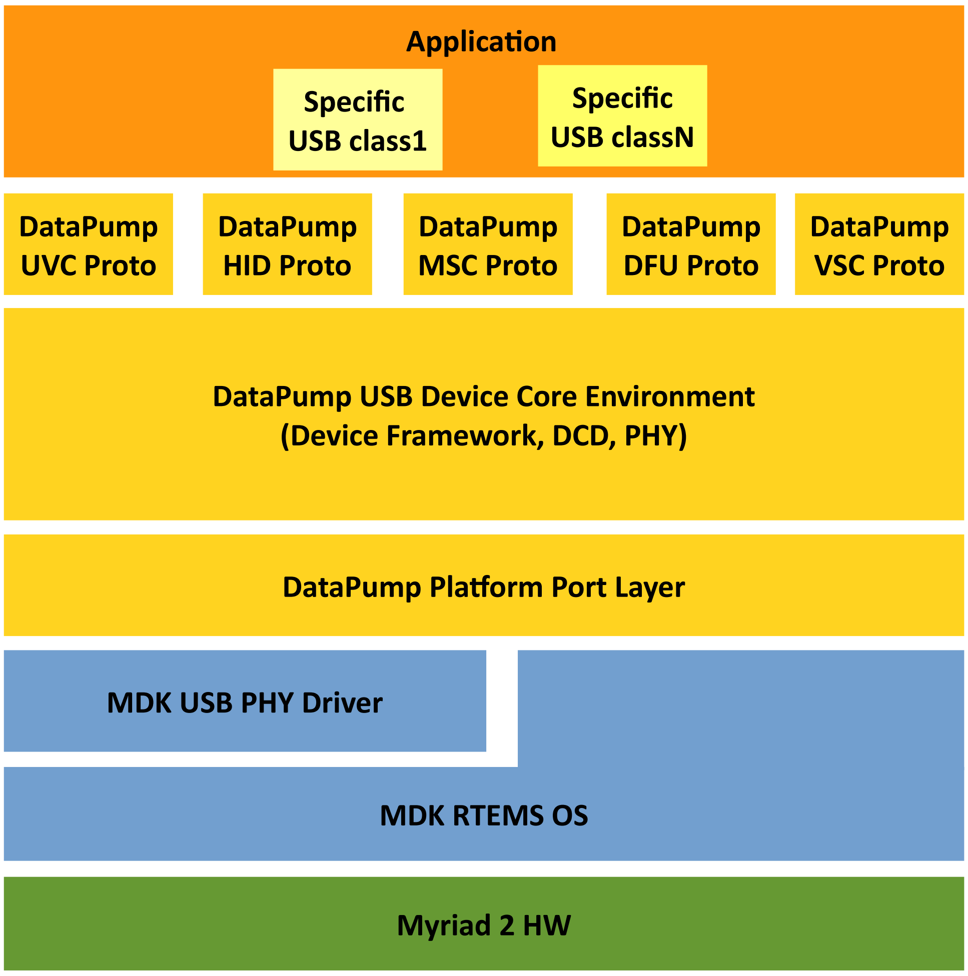

Figure 1: MDK integration of MCCI USB DataPump
(green: HW; blue: MDK; yellow: MCCI DataPump; orange: Application)

For detailed documentation of the DataPump architecture and port, see the MCCI USB DataPump Technical Overview.

3.2 Build system integration

3.2.1 USB DataPump libraries

The DataPump `include` files and libraries are located under:

- mdk/packages/movidius/usb/include
- mdk/packages/movidius/usb/lib/release
- mdk/packages/movidius/usb/lib/debug

The libraries come in two forms: debug and release. Debug libraries should be used only for debug. Using them will print some debug messages when the application is ran. The verbosity of these messages depends on the debug mask that was set at DataPump initialization. A complete list of debug mask flags can be found in:

```
mdk/packages/movidius/usb/include/datapump/usbpumpdebug.h
```

By default MDK applications are using release libraries. To use the debug libraries, the following line must be added in the application makefile:

```
RTEMS_USB_LIB_BUILD_TYPE = debug
```

USB DataPump is organized into the following libraries:

USB DataPump Library	Type	Description
datapump.a	Core	DataPump code library
xdci.a	DCD	DCD (Device Controller Driver) for Synopsys xDCI
osrtems.a	OS	RTEMS OS library
apidownload.a	Device API	DFU class API library
apistorage.a	Device API	MSC class API library
protovideo.a	Device Protocol	UVC class protocol library
protohid.a	Device Protocol	HID class protocol library
protomsc.a	Device Protocol	MSC class protocol library
daprotondfu.a	Device Protocol	DFU class protocol library
protovsc2.a	Device Protocol	VSC class protocol library
protowmc.a	Device Protocol	WMC protocol library
usbseri.a	serial API	USBSERI API library
classkit.a	Host	Class driver development kit
classcomposite.a	Host	Multi-function devices support
class_generic.a	Host	General purpose USB driver that provides access to any USB device
class_msd.a	Host	Mass storage class driver
class_uvc.a	Host	Video Class driver
hcd.a	Host	Host controller driver
hcdkit.a	Host	HCD driver framework
transaction_translator.a	Host	Hub translation between standards
usbd.a	Host	USB driver (USB management module)

USB DataPump Library	Type	Description
xhcd.a	Host	Host controller driver

Table 1: USB DataPump Libraries

3.2.2 Selecting host or device mode

USB can be configured either as device or host, but not both at the same time. The default configuration assumed USB to be in device mode (RTEMS_USB_HOST_BUILD variable is set to no). The build system will emit an error when bot host and device are enabled.

3.2.2.1 Configuring USB as device

In order to use the USB DataPump libraries in an MDK application the 'MV_USB_PROTOS' makefile variable has to be set by the application Makefile. The value of this variable should be the name of the used USB class proto(s):

- protovideo
- protohid
- protomsc
- protondfu
- protovsc2
- protowmc

If more protos are used, then all of them need to be specified separated by spaces, e.g.

```
MV_USB_PROTOS = protovideo protomsc
```

The build system automatically adds the needed `include` paths and libs to the build including the needed core and API libs as well.

3.2.2.2 Configuring USB as host

For host configuration RTEMS_USB_HOST_BUILD build variable should be set to yes.

3.2.3 USBRC target

A new MDK `make` target called 'usbrc' has been added for MCCI's USBRC tool. This `make` target invokes USBRC for all .urc files located under the application directory and generates the corresponding .h and .c files with the same name beside the .urc files. This files shouldn't be used directly. The build system uses it when needed.

When it's invoked the usbrc target generates two files: a .c and a .h, containing the USB descriptors information obtained from the .urc file. These files are automatically generated and deleted by the build system whenever it is needed.

USBRC is not a mandatory tool, but could be useful for generating the class descriptor. For more details see the 950000061r_(USBRC-User-Guide).pdf.

3.3 DataPump initialization

Any application that uses USB stack should perform two initialization steps:

- USB PHY controller initialization
- DataPump initialization

3.3.1 USB Clock configuration

The current configuration of the PHY driver only allows a reference clock of 20 MHz. Each application must enable `USB_CTRL_SUSPEND auxiliary clock and set it to a frequency of 20MHz.`

Example for a `600MHz system clock:`

```
// for a 600 MHz system clock the frequency of USB_CTRL_SUSPEND clock
is
// 600 * 1 / 30 = 20MHz
static tyAuxClkDividerCfg auxClk[] = {
  {
    .auxClockEnableMask     = AUX_CLK_MASK_USB_CTRL_SUSPEND_CLK,
    .auxClockSource         = CLK_SRC_PLL0,
    .auxClockDivNumerator   = 1,
    .auxClockDivDenominator = 30
  },
  {0, 0, 0, 0}, // Null Terminated List
};
```

`The USB PHY driver will check if the frequency for USB_CTRL_SUSPEND is correctly set and will return an error if not.`

3.3.2 USB PHY controller initialization

This part consists of the initialization of the USB PHY controller by calling the OsDrvUsbPhyInit() function. This function must be called before UsbPump_Rtems_DataPump_Startup. It takes two parameters: the first parameter should be set to 1 if OTG block is enabled and 0 otherwise. The second parameter should tell if an external clock is used, when set to 1, or an internal one, when 0.

OsDrvUsbPhyInit alows a default initialization if the parameter passed to it is NULL. This case is equivalent with the following code:

```
osDrvUsbPhyParam_t initParam =
{
    .enableOtgBlock    = USB_PHY_OTG_DISABLED,
    .useExternalClock  = USB_PHY_USE_EXT_CLK,
    .fSel              = USB_REFCLK_20MHZ,
    .refClkSel0        = USB_SUPER_SPEED_CLK_CONFIG,
    .forceHsOnly       = USB_PHY_HS_ONLY_OFF
};
OsDrvUsbPhyInit(&initParam);
```

USB PHY driver contains an option that allows to ignore the VBUS voltage level, which is needed for some boards. This option can be enabled by adding a define:

```
#define USB_PHY_IGNORE_VBUS 1
```

in the application code.

3.3.3 DataPump stack initialization

DataPump allows some parameter configuration by passing a:

USBPUMP_APPLICATION_RTEMS_CONFIGURATION structure to the UsbPump_Rtems_DataPump_Startup function for device or UsbPump_Rtems_DataPump_HostStartup for host.

The USBPUMP_APPLICATION_RTEMS_CONFIGURATION declaration is the following:

```
struct _USBPUMP_APPLICATION_RTEMS_CONFIGURATION
{
  unsigned int               nEventQueue;
  void *                     pMemoryPool;
  unsigned int               nMemoryPool;
  unsigned int               DataPumpTaskPriority;
  unsigned int               DebugTaskPriority;
  unsigned int               UsbInterruptPriority;
  const char *               pDeviceSerialNumber;
  USBPUMP_APPLICATION_RTEMS_USE_BUS_POWER_FN *pUseBusPowerFn;
  unsigned int               fCacheEnabled;
  unsigned int               DebugMask;
  USBPUMP_APPLICATION_RTEMS_PLATFORM_IOCTL_FN *pPlatformIoctlFn;
  unsigned int               fDoNotWaitDebugFlush;
};
```

USBPUMP_APPLICATION_RTEMS_CONFIGURATION structure field description:

Field name	Description
nEventQueue	Number of the DataPump event queue elements
pMemoryPool	Platform memory pool information. The pMemoryPool is a pointer of the platform memory pool. If pMemoryPool is not NULL a platform memory pool will be created and used for dynamic memory allocation. If pMemoryPool is NULL then RTEMS system memory allocation will be used.
nMemoryPool	The size of platform memory pool. If this field is 0 then RTEMS system memory allocation will be used.
DataPumpTaskPriority	DataPump task priority. The priority range from a high of 1 to a low of 255.
DebugTaskPriority	Debug task priority. The priority range from a high of 1 to a low of 255. This debug task priority should be lower than DataPump task priority.
UsbInterruptPriority	USB interrupt priority value. This will be used to configure USB interrupt.
pDeviceSerialNumber	This is USB device serial number string. If pDeviceSerialNumber is NULL, DataPump will use serial number string in the URC file.
pUseBusPowerFn	This function will be called by DataPump when host sends GetDeviceStatus command. This function returns current device power state which use USB bus power or self power.
fCacheEnabled	This flag represents system enabled cache or not. The user should set this flag to TRUE if system cache is enabled. If this flag is TRUE, device controller driver will flush and invalidate data buffer before and after an USB IO operation.
DebugMask	DataPump debug mask. A complete list of debug mask flags can be found in: mdk/packages/movidius/usb/include/datapump/usbpumpdebug.h

Field name	Description
pPlatformIoctlFn	This function is user supplied UPLATFORM ioctl handler. The DataPump platform object ioctl handler calls this function if DataPump platform object doesn't support IoctlCode. If this function doesn't support IoctlCode, it should returns USBPUMP_IOCTL_RESULT_NOT_CLAIMED.
fDoNotWaitDebugFlush	If enabled DataPoint will not wait for all the data in the debug queue to be flushed

Table 2: USBPUMP_APPLICATION_RTEMS_CONFIGURATION structure field description

3.4 Critical memory sections

By default, DataPump does not use any specific memory sections for its code/data, but there are some critical functions and data structures that are recommended to be placed in a fast memory to achieve the best performance. A linker script that contains this list is available in MDK: common/scripts/ld/myriad2collection/myriad2_usb_critical_memory_sections.ldscript. This script, if included in the application custom linker script, will place the critical functions and data structures into CMX. The addresses at which this section will be placed in can be configured (to other CMX or DDR addresses) by defining __USB_FAST_TEXT_START, __USB_FAST_RODATA_START and/or __USB_FAST_DATA_START in the local linker script, before including the script that contains the sections. If these variables are not changed, a default set address will be used.

The default section placement is at the end of the CMX memory, after all other CMX sections.

The following sections will be created:

Section name	Description	Section size
.usb.fast_text_mem	Contains the function that should be placed in a fast memory section.	26888 bytes
.usb.fast_rodata_mem	Contains the read only data structures that should be placed in a fast memory	576 bytes
.usb.fast_data_mem	Contains the read/write data structures that should be placed in a fast memory	Variable size

NOTE: .usb.fast_data_mem contains the statically allocated memory pool, if that memory pool is used. Its size is variable depending on the pool size that is requested at DataPump initialization (see section 3.3).

4 USB performance

4.1 USB Mass Storage Class performance benchmark using RAM disk

4.1.1 HighSpeed measurements

The measure results may be different, depending on the used host controller.

The tests were made for two different cases: one with all code and data placed int CMX and another one with code and data placed into DDR.

The tool used to measure the speed on the host was HD_speed. EHCI controller was on a Windows 7 machine and xHCI controller on Windows 8.

The table below contains the measured values for and USB 2.0 device (block size represents the size of the data chunks that the application requests from the OS):

Code + data in CMX				
	EHCI controller		xHCI controller	
Block Size(KB)	Read speed (MiB/s)	Write speed (MiB/s)	Read speed (MiB/s)	Write speed (MiB/s)
512	31.9	25.1	37.8	35.6
265	31.9	24.8	37.8	35.4
128	31.6	24.7	37.3	34.8
64	31.1	24.7	36.1	33.7

Code + data in DDR				
	EHCI controller		xHCI controller	
Block Size(KiB)	Read speed (MiB/s)	Write speed (MiB/s)	Read speed (MiB/s)	Write speed (MiB/s)
512	32.0	25.0	37.5	35.4
265	32.0	25.0	37.3	35.4
128	31.8	25.0	36.6	34.4
64	31.5	24.6	35.3	33.6

The small difference between the DDR and CMX is caused by the fact that the test application used for this benchmark is small. Also the RAM disk size is very small (256 KB only), so the code and data are cached efficiently.

4.1.2 SuperSpeed (USB 3) measurements

Windows test environment:

- OS version: Windows 8.1
- testing tool: HD_Speed
- 64MiB RAM disk Myriad™ 2 application

Linux test environment:

- OS version: Ubuntu 14.04 LTS
- testing tools: gnome-disks (Benchmark option)
- 64MiB RAM disk Myriad™ 2 application

Operation	Windows 8.1	Linux (Ubuntu 14.04)
Read	377 MiB/s	326 MiB/s
Write	385 MiB/s	331 MiB/s

4.2 Video class performance

Isochronous transfers are used to transmit a fix amount of data periodically. The time on the USB bus is split into so-called bus intervals. For HighSpeed (USB 2) and SuperSpeed (USB 3) a bus interval is 125µs. This time interval is also called microframe. 1ms intervals represent a frame which is composed by 8 microframes.

An isochronous endpoint contains information on the amount of data that is to be sent on each bus interval and the period of the transfers. In terms of period, data can be transferred on every bus interval, every second bus interval, every fourth bus interval or every eighth bus interval. Knowing these two parameters allows the calculation of the transfer speed by multiplying the data size transferred in each frame (1ms) by 1000, which is the number of frames in a second.

To calculate the speed the following formula can be used (assuming that a packet is sent on each bus interval):

Speed(MiB/s) = PacketSizeKB * 8 * 1000 / 1024

where PacketSizeKB is the packet size configured in the endpoint descriptors. Each millisecond (one USB frame) has 8 bus intervals, with 1000 USB frames each second.

Example for a 24KiB transmitted each bus interval:

Speed = 24 * 8 * 1000 / 1024 = 187.5 MiB/s

MDK provides an application that shows how to use isochronous transfers with UVC:

mdk/examples/Evaluation/UsbVideoIso

In order to prove the maximum throughput available when isochronous transfers are used, the application is sending two DDR preloaded images over USB. For testing purposes, this is a simple way to send data at the highest rate possible. Since the maximum transfer rate is different between USB 2 and USB 3, a different image resolution is used for each case: 640x480 for USB 2 and 2104x1560 for USB3. In both cases the format of the images is YUV422.

The results obtained using the application can be seen in the table below:

Frame size	Frame rate (frames/s)	Transfer speed (MiB/s)
HighSpeed		
640x480	40	23.4
SuperSpeed		
2104x1560	50	328.1

NOTE: Some players may show frame tearing when displaying USB 3 streamed video. The recommended players in order to avoid this are VLC for Windows and luvcview for Linux.

4.3 Vendor specific class performance

The benchmark application used to generate the table below was sending data to the host (IN transfers) and receiving data from the host (OUT transfers) in blocks of the size from the first column. The results are an average measured over multiple repetition of transfers of the same size.

The host controller used for testing is an Intel Corporation Sunrise Point-H USB 3.0 xHCI Controller on a HP ZBook laptop.

Buffer size (bytes)	Transfer speed OUT endpoint (MiB/s)	Transfer speed IN endpoint (MiB/s)
512	5.79	5.59
1024	15.01	14.51
2048	27.37	27.22
4096	67.58	67.32
8192	123.95	121.32
16384	195.68	197.41
32768	288.18	291.22
65536	333.11	335.84
131072	356.37	358.79
262144	367.63	375.86
524288	376.87	378.36
1048576	393.02	394.11

4.4 Host driver performance for mass storage

The test were performed with the Myriad chip configured as host using two different USB SSDs:

- Chipfancier WINDOWS TO GO Solid State USB3.0 SSD Drive 128MB

- Corsair Flash Voyager GTX 128GB USB 3.0 Flash Drive

Both SSDs were formatted as FAT32 before being tested. On the Myriad side, RTEMS dosfs was used for file system operations.

The internal buffer used for transfer in each case was 1MiB.

Drive type	Read speed	Write speed
Chipfancier	74.6 MiB/s	90.20 MiB/s
Corsair GTX	74.26 MiB/s	65.53 MiB/s

NOTE: The speeds measured in the table above does not represent the raw USB speeds. The test example was using RTEMS dosfs for file system operations. The transfer speed was limited by dosfs implementation. Using other FAT file system implementations may give different results.

5 MDK USB examples

Example	USB class	Description
Demo/UsbVideo208	UVC	Video streaming from IMX208 sensor using bulk endpoint for transfers.
Evaluation/ma2x5x/UsbVideoIso	UVC	Video streaming images placed in DDR using isochronous endpoint for transfers.
Demo/UsbSerialPortRtemsShell	CDC	Example application that shows how to connect RTEMS shell to a virtual serial port over USB.
Demo/UsbMscDemo	MSC	64MiB DDR ram disk example using USB mass storage class.
HowTo/UsbHidMouse	HID	Example showing how to send HID reports to the USB mouse connected to the host.

Each of the examples above contain a Readme.txt file that provides additional information on how to run the application and additional tools or actions that are needed on the host side.

6 USB classes

6.1 Video class (UVC)

USB video class is a class that describes devices capable of streaming video. Video streaming using UVC can use bulk or isochronous endpoints for transfers. An example for each case is available in MDK.

6.1.1 UVC isochronous API

Starting with version 3.14.11, DataPump provides a new API for bulk and isochronous video transfers. This API takes advantage of the scatter/gather feature of USB DMA, to remove the need of having the packet header placed in memory just before the data that needs to be sent. Because of this, there is no need to copy each isochronous packet in a separate buffer, leading to a much better performance. The old functions VideoClientLib_StreamReadIsoch and VideoClientLib_StreamWriteIsoch are still available, but it is recommended to use the new ones.

The prototype for the new functions is similar to the existing ones. The only addition is the buffer handle parameter.

VideoClientLib_StreamReadIsochV2 – sends data to the host using an isochronous pipe	
pVideo	Pointer to a UPROTO_VIDEO instance.
pCallback	Callback function called at the API completion.
pCallbackCtx	Callback context data.
hVideoStream	Video stream handle associated to the out endpoint.
pBuffer	The buffer where the data is read.
nBuffer	The size of the read data.
hBuffer	A buffer handle containing the information about headers and payloads location and size.
pIsochDescr	Structure that gives the packet-by-packet layout of the data buffers for isochronous transfers.
IsochDescrSize	Size of the isochronous descriptor list.

VideoClientLib_StreamWriteIsochV2 – sends data to the host using an isochronous pipe	
pVideo	Pointer to a UPROTO_VIDEO instance.
pCallback	Callback function called at the API completion.
pCallbackCtx	Callback context data.

VideoClientLib_StreamWriteIsochV2 – sends data to the host using an isochronous pipe	
hVideoStream	Video stream handle associated to the in endpoint.
pBuffer	The buffer where the data is read.
nBuffer	The size of the data read.
hBuffer	A buffer handle containing the information about headers and payloads location and size.
pIsochDescr	Structure that gives the packet-by-packet layout of the data buffers for isochronous transfers.
IsochDescrSize	Size of the isochronous descriptor list.

The buffer handle needed for the above functions is created using `USBPUMP_UHILAUX_CREATE_HBUFFER_WITH_SEGMENT_FN`. This is an interface function of the Hardware Interface Layer (HIL). The function has the following prototype defined in `usbpump_uhilaux_types.h`:

USBPUMP_UHILAUX_CREATE_HBUFFER_WITH_SEGMENT_FN – creates a buffer handle	
hSession	Session handle – the session handle is a member of pDevice.
phBuffer	An output parameter which returns a pointer to the newly created handle.
numSegments	The number of data packets that need to be sent.
vaBuffer1	The address of the buffer containing header data.
nSegBuffer1	The size of one header.
vaBuffer2	The address of the buffer containing the actual data that need to be sent.
nSegBuffer2	The size of a data packet.
Access	The access type: - USBPUMP_UHILAUX_BUFFER_ACCESS_READ – needed for VideoClientLib_StreamWriteIsochV2 - USBPUMP_UHILAUX_BUFFER_ACCESS_WRITE – needed for VideoClientLib_StreamReadIsochV2

An example on how to use the above API functions is available in MDK:

```
mdk/examples/Evaluation/UsbVideoIso
```

6.1.2 UVC documentation

- UVC class specifications – contains general description of USB video devices and documentation

for specific video formats supported by UVC protocol:
http://www.usb.org/developers/docs/devclass_docs/USB_Video_Class_1_5.zip

- Video protocol user's guide for DataPump: 950000736c_(MCCI-USB-DataPump-Video-Protocol-Users-Guide).pdf document available in MDK release

6.2 Mass storage class (MSC)

USB mass storage class is a set of protocols that enables file transfers between a host and USB device.

The MCCI MSC Protocol Library, in conjunction with the MCCI USB DataPump, provides an environment for implementing ATAPI compliant mass storage devices over USB using the USB Mass Storage BOT 1.0 protocol. The MCCI MSC Protocol Library can be used to create a stand-alone device, or can be combined with other user-provided protocols to create multi-function devices.

6.2.1 MSC Documentation

- MSC Bulk-only transport (BOT) specification:
  http://www.usb.org/developers/docs/devclass_docs/usbmassbulk_10.pdf
- MCCI MSC Protocol Library documentation: 950000250i_(MCCI-USB-DataPump-Mass-Storage-Protocol-Users-Guide).pdf provided with the MDK release

6.3 Human interface device class (HID)

6.3.1 HID device characteristics

- all the data is exchanged using a fixed length structure named report
- an interrupt IN endpoint is mandatory; it is used for sending Input reports
- an additional interrupt OUT endpoint can be added, but it is not mandatory
- HID device transmit data asynchronously to the host using the interrupt IN endpoint
- the transfer rate is limited:
  - 800 B/s maximum for low speed
  - 64 KB/s maximum for full speed
  - high speed support faster rates, but to comply with the USB 2.0 specifications, the endpoints in the default interface should request no more than 64 KB/s.

6.3.2 HID class Documentation

- HID class specification:
  http://www.usb.org/developers/hidpage#Class_Definitions
- HID class protocol implementation and API documentation for DataPump is described in 950000387d_(MCCI-USB-DataPump-HID-Protocol-Users-Guide).pdf document provided with the MDK release

6.4 Communication Device Class (CDC)

USB communication device class is a composite device class. It is usually used for modem devices, but may contain multiple other interfaces like data interface, audio, mass storage.

DataPump WMC (Wireless Mobile Communications) Protocol Library provides a generic implementation of the USB Device Communication Device Class (CDC) - Advanced Control Model (ACM) protocol.

The WMC Protocol Library can be used to create a standalone device, or can be combined with other USB protocols to create multi-function devices.

The library can be use to implement CDC-ACM and CDC WMC ACM compliant AT-compatible modem devices and modem-like devices over USB using the following protocols:

- USB CDC 1.1 Abstract Control Model (ACM) protocol USBCDC and the USB WMC 1.0 ACM
- USB WMC 1.0 Object Exchange (OBEX) and
- USB WMC 1.0 Device Management protocols (USBWMC)

DataPump also provides a higher level api: USBSERI, whose purpose is to simplify the WMC protocol client interface and to lower the integration effort

6.4.1 CDC devices in Windows

Windows requires an .inf file to be provided whenever a new virtual serial device is attached. The .inf file describes the device and provides some information about it. The inf file should be selected when Windows device manager asks for a driver for the serial device.

UsbSerialPortRtemsShell example from MDK contains an .inf file that can be used to install the proper driver for the serial device, which in this case is seen as a virtual serial port.

6.4.2 CDC documentation

- WMC protocol implementation details and some guidelines on how to use the WMC API: 950000255d_(MCCI-USB-DataPump-WMC-Protocol-Users-Guide).pdf document provided with the MDK release
- USBSERI API documentation: 950386a_(MCCI-USBSERI-API).pdf provided with MDK release
- Class definitions for communication devices: http://www.usb.org/developers/docs/devclass_docs/CDC1.2_WMC1.1_012011.zip

7 Reference Documentation

References can be found bellow to the official documentation of the MCCI DataPump USB Device Driver stack available in MDK.

Keep in mind that MCCI USB DataPump is a broad USB solution, so some parts of the MCCI documents may contain some specific DataPump sections which are not applicable for the Myriad™ 2 MDK integration of the SW stack.

Document	Description
950000066m_(MCCI-USB-DataPump-User-Guide).pdf	MCCI USB DataPump User's Guide
971001018a_(MCCI-USB-DataPump-Technical-Overview).pdf	MCCI USB DataPump Technical Overview
950000736c_(MCCI-USB-DataPump-Video-Protocol-Users-Guide).pdf	MCCI USB DataPump Video Class (UVC) Protocol User's Guide
950000387d_(MCCI-USB-DataPump-HID-Protocol-Users-Guide).pdf	MCCI USB DataPump Human Interface Device (HID) Protocol User's Guide
950000250i_(MCCI-USB-DataPump-Mass-Storage-Protocol-Users-Guide).pdf	MCCI USB DataPump Mass Storage (MSC) Protocol User's Guide
950000255d_(MCCI-USB-DataPump-WMC-Protocol-Users-Guide).pdf	MCCI USB DataPump WMC Protocol User's guide
950386a_(MCCI-USBSERI-API).pdf	MCCI USB DataPump USBSERI API documentation
950000298c_(DataPump-New-DFU-Protocol-Users-Guide).pdf	MCCI USB DataPump Device Firmware Upgrade (DFU) Protocol User's Guide
950001297a_(MCCI-USB-DataPump-VSC-V2-Protocol-Users-Guide).pdf	MCCI USB DataPump Vendor Specific (VSC) Protocol User's Guide
950000061r_(USBRC-User-Guide).pdf	MCCI USB DataPump USB Resource Compiler (USBRC) USBRC simplifies the development of USB peripherals by generating data for USB descriptors via a high-level easy-readable configuration text file.
950000819d_(MCCI-USB-DataPump-Footprint-	MCCI USB DataPump Describes the code and memory footprint statistics for the USB

Document	Description
Information).pdf	DataPump stack
950519a_(DataPump-Dynamic-Sizing).pdf	MCCI USB DataPump Describes method how to gather DataPump RAM usage information
950000325e_(MCCI-USB-DataPump-Embedded-USBDI).pdf	USB driver interface documentation
950000324f_(MCCI-USB-DataPump-HCD-API).pdf	Host controller driver interface
950000684b_(MCCI-Composite-Device-Users-Guide).pdf	Composite class driver documentation
950000548c_(MCCI-Transaction-Translator-Users-Guide).pdf	Transaction translator documentation
950000761c_(MCCI-USB-DataPump-Embedded-Host-Class-Driver-Development-Guide).pdf	Guide on building new USB class drivers
950001350a_(MCCI-Embedded-Mass-Storage-Class-Driver-V2-Users-Guide).pdf	Mass storage class driver documentation
950000692b_(MCCI-DataPump-Embedded-Host-Generic-Class-Driver-Users-Guide).pdf	Generic class driver documentation
950001473a_(MCCI-Embedded-UVC-Class-Driver-Users-Guide).pdf	Video class driver documentation