C-Media CM65xx USB 1.0 Controller

Firmware Programmers' Guide

C-MEDIA Electronics Inc.

Revision History

Revision	Date	Description
1.00	2013/01/24	Draft
1.01	2013/05/20	Redefine the file structure of Framework.
2.01	2014/03/12	The new library CM65xxB-1.lib has been released, and three code templates have been released, includes headset, speaker, and microphone.

Contents

1. Introduction

CM65xx is a family of R8051XC (1T)-based chip with on-chip memory for USB audio device. It complies with USB 1.1 and 2.0 specifications for full speed. Besides control endpoint 0, CM65xx provides one isochronous-in endpoints for ADC, one isochronous-out endpoints for DAC, one interrupt-in endpoints, two bulk-in endpoint, two bulk-out endpoint and one feedback-in endpoint.

The CM65xx firmware framework provides an easy-to-use software platform to simplify the development process of USB audio 1.0 peripherals with CM65xx family – CM65xx. This document acts as a programmer's reference manual for developers who need to design firmware based on CM65xx firmware framework.

There are sample applications of USB audio device in the framework package. The sample applications can be run on hardware platform of C-Media. Users can modify the application to develop different USB devices.

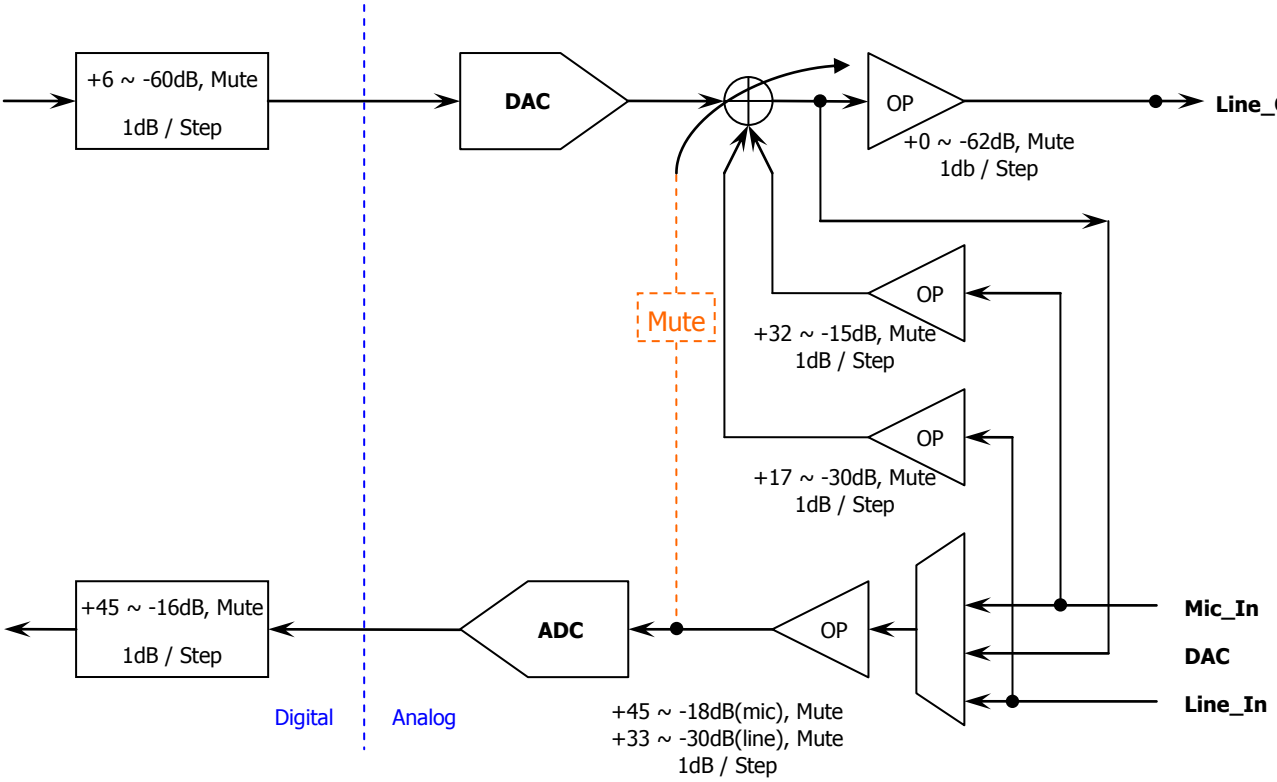
2. Features

The features of CM65xx sample application are listed below.

- Supports USB 1.1 and 2.0 full-speed operation.
- Compliance
  - USB HID class 1.1.
  - USB audio class 1.0.
- User can design a user-defined USB device.
- User can easily control peripherals of CM65xx including GPIO, UART, …
- Programmable MCU clock Speed (3/6/12/24MHz/48MHz)(default = 12MHz)
- Supports buttons "play-mute", "rec-mute", "vol-up", "vol-down", "play/pause", "stop", "next", and "previous".
- Supports LEDs "config/play/record", "play-mute", "rec-mute", "play-rec", and "rec-clipping".
- Output capability
  - Support 2-channel speaker output via internal DAC codec.
  - Support 2-channel SPDIF output.
- Input capability
  - Support 2-channel analog recording via internal ADC codec.
  - Support 2-channel SPDIF input
  - Support A/A paths for monitoring microphone and line inputs.

- Customized volume range
  - Analog
    - DAC gain: -62dB~0dB/step 1dB.
    - Mic ADC gain: 0dB~30dB/step 1dB.
    - Line ADC gain: -30dB~12dB/step 1dB.
    - Mic A/A gain: -15dB~22dB/step 1dB.
    - Line A/A gain: -30dB~12dB/step 1dB.
  - Digital (I2S)
    - DAC gain: -60dB~0dB/step 1dB.
    - ADC gain: -16dB~12dB/step 1dB.
- Three topologies are supported. They are headset, speaker only and microphone only. The PID and VID can be customized by the users no matter what the bonding option is.

3. Internal Codec Block Diagram



4. Packages

The directory “Framework” includes the CM65xx’s library which supports three topologies and sample applications of USB audio device. There are four sub-directories in the directory “Framework” that is shown as below.

Framework\
Tools\ - Hex2Rom.exe which is used to transfer hex files.
Doc\ - Documents
CM65xxB-1\ - CM65xx framework
inc\ - Header files
CM65XXB-1.LIB - The library of CM65xx framework
CM65xxB-1_xxx\ - Sample application source codes, xxx could be Headset, Speaker, or Microphone.
inc\ - Header files for customization
output\ - the output files after build
**.c, *.a51* - the source code for sample template
make.bat, build.bat - Batch files for building the project in the console mode

The source files, header files, and the library file of CM65xx firmware framework are listed in the following table:

Directory	Files	Description
Framework\ **CM65xxB-1**\	CM65xxB-1.LIB	The library file of framework.
Framework\ **CM65xxB-1**\ *inc*\	audio.h cm65xxlib.h cm65xx.h registers.h types.h usb.h	Header files of USB audio class 1.0 based on CM65xx ROM firmware.
Framework\ **CM65xxB-1_xxx**\ *inc*\	config.h	Header files for customization.
Framework\ **CM65xxB-1_xxx**\	main.c int.c usb.c io.c request.c	A sample application files of USB audio 1.0 device.

	audio.c dscr.a51	
Framework\ **CM65xxB-1_xxx**\	cm65xx.lin cm65xx_cmd.lin	Link script files, cm65xx.lin is for KeilC environment, and cm65xx_cmd.lin is for the console mode.
Framework\ **CM65xxB-1_xxx**\	CM65xx.Uv2 make.bat build.bat	CM65xx.Uv2 is the KeilC setting file, and batch files is for building in the console mode.
Framework\ **CM65xxB-1_xxx**\ *output*\		The output files after building.

5. Build and Update

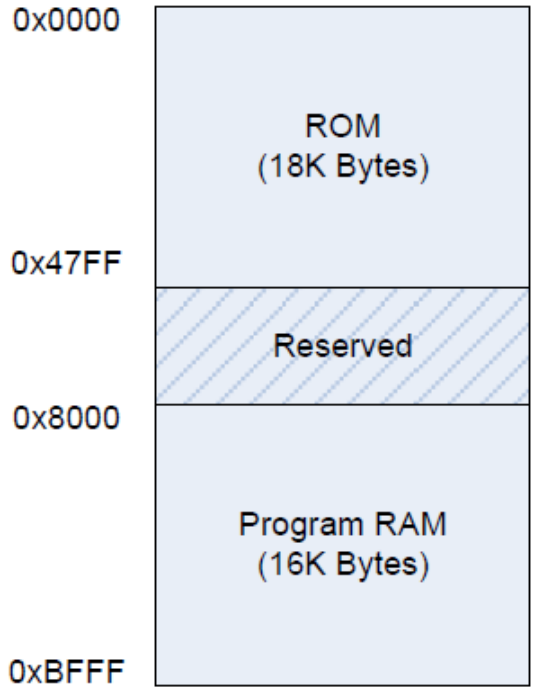
CM65xx firmware framework and sample applications are built by tool chain of Keil uVision3 V3.33. They have been built successfully by complier C51 V8.05a, assembler A51 V8.00b, and linker BL51 V6.02.

In the directory of sample application, there is a batch file “build.bat” for building the sample application. Users can build the firmware easily by going into directory “CM65xxB-1_xxx” and executing “build xxxx” in Windows’ console. Currently, one file “cm65xxfw.hex” will be created in the directory “CM65xxB-1_xxx”. Users can modify these batch files according to users’ requirement.

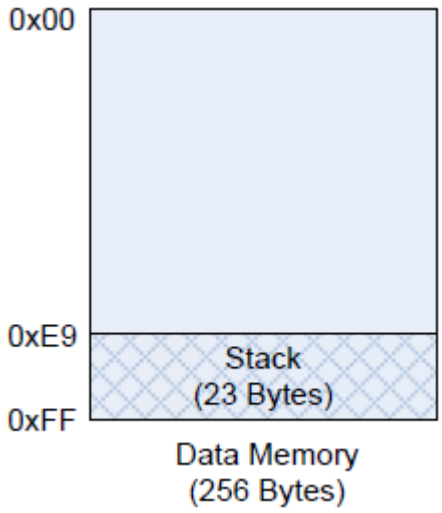
After building firmware, which the tool *Hex2Rom.exe* is also executed, two different bin files are made which are *Cm65xxCode.bin* and *CM65xxB-1_ROM.bin*; the *Cm65xxCode.bin* file is translated directly from *cm65xxfw.hex*, and *CM65xxB-1_ROM.bin* includes header settings so it can be written into EEPROM by using C-Media’s PC download tool.

6. Memory Usage

Program memory layout is shown below. The size of ROM memory is 18k bytes. One purpose of ROM is to be a boot loader that transmit the customization code, if it exists, from EEPROM to internal RAM sector (0x8000~0xBFFF); and the other purpose is to used as the default USB audio devices that include headset/speaker/microphone/docking/Lync_headset topologies.

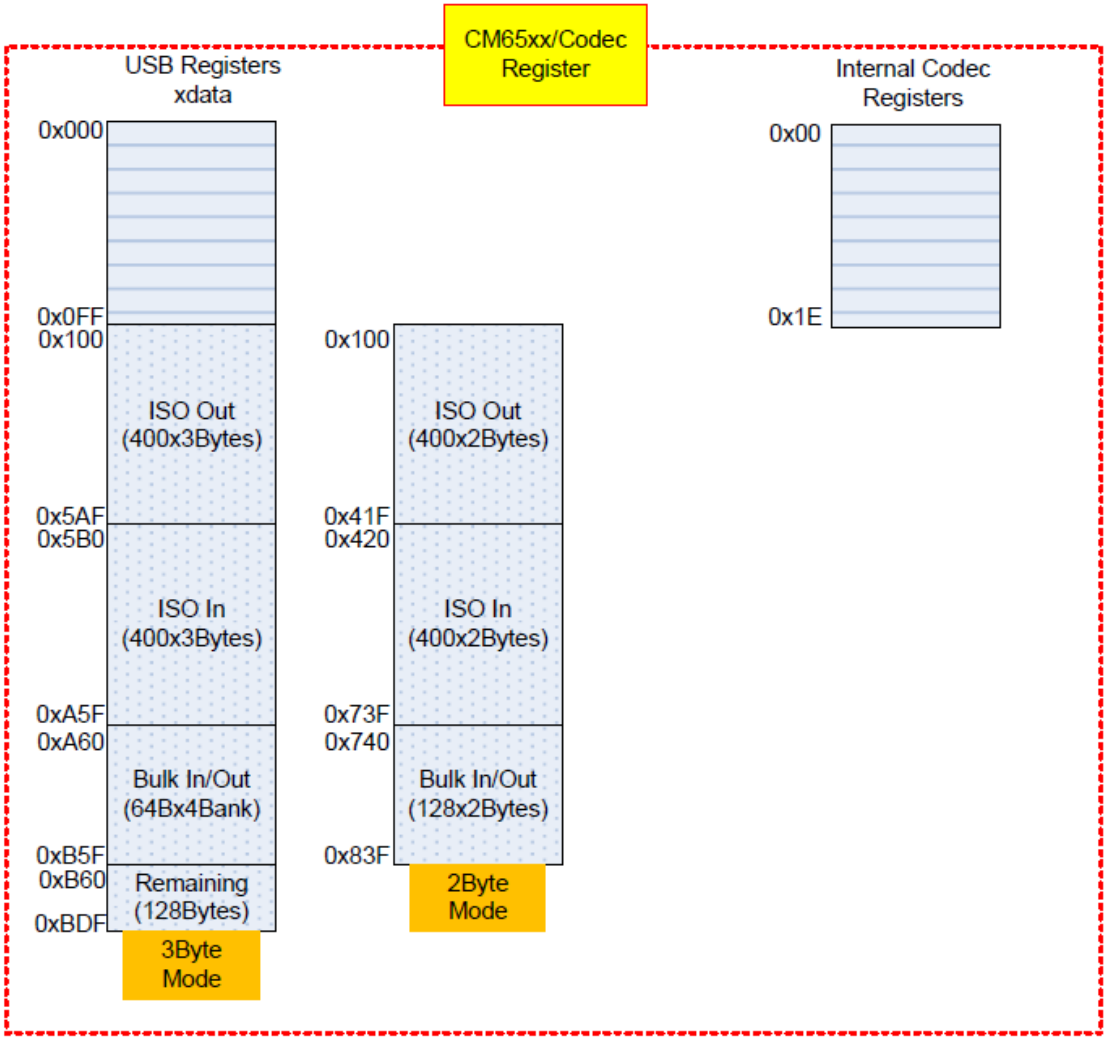


As general MCU 8051, the total data memory size is 256 bytes. Section DATA (0~0x7F) is accessible using direct addressing, the other section IDATA (0x80~0xFF) is accessible using indirect addressing. Currently, section 0xE9~0xFF is reserved for stack. This memory allocation can be changed by firmware developers.



The memory map of XDATA of CM65xx and internal codec registers are shown below.

Several memory areas can be used for customization if the internal data memory is not enough for development. For example, memory section 0xA60~0xBDF can be used for customized usage in 3Byte mode.



7. Interface of Standard CM65xx Library

In “Framework\CM65xxB-1\inc\”, major header files and source files define structures, macros and functions for CM65xx’s USB and peripheral control.

Standard functions are implemented in CM65xx sample firmware. CM65xx sample firmware will call them when corresponding events happen. “RefreshWdt” function is a must function which is used to refresh the watchdog timer because the watchdog timer is always on to avoid the system crash. Below functions with required mark are necessary to be used if the standard USB audio device is declared.

7.1 Structures

- **Descriptor structure**

Device descriptor structure as follows:

```
typedef struct _USB_DEVICE_DSCR_STRUCT
{
    BYTE bLength;              // Descriptor length ( = sizeof(DEVICEDSCR) )
    BYTE bDescriptorType;      // Decriptor type (Device = 1)
    WORD bcdUSB;               // Specification Version (BCD)
    BYTE bDeviceClass;         // Device class
    BYTE bDeviceSubClass;      // Device sub-class
    BYTE bDeviceProtocol;      // Device sub-sub-class
    BYTE bMaxPacketSize0;      // Maximum packet size for endpoint zero
    WORD idVendor;             // Vendor ID
    WORD idProduct;            // Product ID
    WORD bcdDevice;            // Product version ID
    BYTE iManufacture;         // Manufacturer string index
    BYTE iProduct;             // Product string index
    BYTE iSerialNumber;        // Serial number string index
    BYTE bNumConfigurations;   // Number of configurations
} USB_DEVICE_DSCR_STRUCT;
```

- **HID descriptor structure**

```
typedef struct _USB_HID_DSCR_STRUCT
{
    BYTE bLength;              // Descriptor length ( = sizeof(DEVICEDSCR) )
    BYTE bDescriptorType;      // Decriptor type (Device = 1)
```

```
    WORD bcdHID;
    BYTE bCountryCode;
    BYTE bNumDescriptors;
    BYTE bDescriptorType1;
    WORD wDescriptorLength;
} USB_HID_DSCR_STRUCT;
```

- **USB request**

```
typedef struct _USB_CONTROL_COMMAND_STRUCT
{
    BYTE bmRequestType;
    BYTE bRequest;
    WORD wValue;
    WORD wIndex;
    WORD wLength;
} USB_CONTROL_COMMAND_STRUCT;
```

USB_CONTROL_COMMAND_STRUCT defines the structure of an USB request from host. It complies with USB specification.

```
typedef enum
{
    NONE_STAGE,
    SETUP_STAGE,
    DATA_IN_STAGE,
    DATA_OUT_STAGE,
    STATUS_STAGE,
    STALL_STAGE
}USB_CONTROL_STATE;
```

This stage enumeration defines several stages of a control transfer that firmware flow handles.

- **Audio feature control**

There are five audio feature units dedicated for the control of volume and mute status mostly, defined as follows:

```
typedef enum
{
    FEATURE_DAC = 0,
    FEATURE_ADC_LINE,
```

```
    FEATURE_ADC_MIC,
    FEATURE_MIXER,
    FEATURE_MONITOR_LINE,
    FEATURE_MONITOR_MIC,
    FEATURE_SPDIF
} FEATURE_UNIT_NO;
```

FEATURE_DAC indicates the feature unit of DAC.
FEATURE_ADC_LINE indicates the feature unit of LINE-IN.
FEATURE_ADC_MIC indicates the feature unit of MIC-In.
FEATURE_MIXER indicates the feature unit of MIXER. (DAC path + AA path from MIC IN and LINE IN).
FEATURE_MONITOR_LINE indicates the feature unit of LINE-IN AA path.
FEATURE_MONITOR_MIC indicates the feature unit of MIC-IN AA path.
FEATURE_SPDIF indicates the feature unit of SPDIF-IN.

There are four ADC input sources for the selector unit, defined as follows.

```
typedef enum
{
    SELECTOR_MIC = 1,
    SELECTOR_LINE_IN,
    SELECTOR_SPDIF_IN,
    SELECTOR_SMIX_IN
} SELECTOR_SELECT;
```

SELECTOR_MIC can be used to select MIC-IN source.
SELECTOR_LINE_IN can be used to select LIN-IN source.
SELECTOR_SPDIF_IN can be used to enable SPDIF-IN source.
SELECTOR_SMIX_IN can be used to select the stereo mixer input (DAC + AA path from MIC IN and LINE IN).

Below functions are used to control the audio features like volume, mute, AGC, etc.

```
typedef struct _AUDIO_CONTROL_STRUCT
{
    void            (*featureVolume)();
    void            (*featureMute)();
    void            (*featureAgc)();
    void            (*setSelector)();
    void            (*recordMute)();
    BYTEcode        *pAcUnitTable;
```

```
    AS_CONTROL_STRUCT    asControl[2];
}AUDIO_CONTROL_STRUCT;
```

“featureVolume” function is used to control volume feature unit.

“featureMute” function is used to control mute feature unit.

“featureAgc” function is used to control AGC which can avoid the clipping from recording.

“setSelector” function is used to control the ADC input source.

“recordMute” function is used to mute the recording path.

“pAcUnitTable”is assigned to an audio control table.

“asControl[2]” is assigned to an audio stream interface.

7.2 Public Variables

Necessary variables are defined in the library and will be used in the application firmware, these are listed as below:

Variable	Description
BOOL g_bmUsbPuReset	This bit indicates whether the pull-up resistance of D+ is connected.
BYTE idata g_bHidI2CSInfo[2]	This is used for saving the information comes from I2C slave interface.
BYTE xdata I2CM_DEV_ADDR; BYTE xdata I2CM_MAP_ADDR[2]; BYTE xdata I2CM_DATA_BUF[16]; BYTE xdata I2CM_DATA_LEN; BYTE xdata I2CM_16BIT_MODE;	These are defined in the xdata memory and offered a access interface for the users to control I2C master communication.
WORD g_wIrTemp	The is used to represent the latest IR code data.
WORD code BaudRateTbl[6][5]	This is used to set UART baud rate at different MCU clocks.

Some variables of USB module are declared as global variables. Customized application can access them directly. These public variables of the sample application are listed below:

```
#define MAX_FEATURE_VOL_NUM          6
#define MAX_HID_REPORT_SIZE          16
#define MAX_BUFFER_SIZE              16
```

Variable	Description
BYTE g_bExtIntEnable	**DO NOT change the memory address which is always at 0x0067h**

BYTE code ManufactureStringDscr	Manufacture string descriptor
BYTE code LangIdStringDscr	Language ID string descriptor
BOOL g_bmRemoteWakeupEn	This is used to enable/disable USB remote wakeup function 1: enable, 0: disable
BOOL g_bmSelfPower	This is used for self power. 1: self power, 0: bus power
BOOL g_bmAdcHPSEn	This is used to enable/disable adc high pass filter. 1: disable, 0:enable
BOOL g_bmSpdifOutEn	This is used to enable/disable SPDIF output. 1: disable, 0: enable
BOOL g_bmGpiRequest	This represents if GPI event is received.
BOOL g_bmUsbResume	This represents if USB resume is received.
BYTE i_timercount	Timer0 count per 1ms
WORD code BaudRateTbl[6][5]	This is used to represent the supported baud-rate table. The baud rate selection is related to MCU clock.
BYTE idata g_bCurrentVolume[MAX_FEATURE_VOL_NUM][2]	This is used to represent the current volume value depending on different feature unit
BYTE g_bCurrentMute	This is used to represent the current mute state for different feature unit.
BYTE g_bConfiguration	This is set to 1 when the device is configured by the host.
BYTE g_bTimer1Count	This is used to indicate the counting status of the timer 1.
BYTE *g_pbConfigDscr	It's a generic pointer points to one of the defined USB configuration descriptors.
BYTE *g_pProductStringDscr	It's a generic pointer points to one of the defined USB product string descriptors.
BYTE *g_pHidDscr	This is a generic pointer to point to the HID interface descriptor.
BYTE *g_pHidReportDscr	It's a generic pointer points to the USB HID report descriptor.
USB_CONTROL_COMMAND_STRUCT g_UsbCtrlCmnd	It's a data buffer used to store each USB setup packet data.
BYTE g_bInputReport[MAX_HID_REPORT_SIZE]	This is used to report the HID input data. Fixed size = 16.
BYTE g_bOutputReport[MAX_HID_REPORT_SIZE]	This is used to store USB HID output report data. Fixed size = 16.
BYTE g_bDataBuffer[MAX_BUFFER_SIZE]	This can be used for data buffer.

BYTE *g_pbDataBuffer	This is a generic data buffer pointer points to any data buffer in the data stage of the USB control transfer.
BYTE xdata *g_pbXdataStart	This can be used to access xdata memory area.
BYTE g_bAttribute	This can be used to represent the audio setting.
BYTE g_bTemp	Temporary variable
BYTE g_bTempA	Temporary variable
BYTE g_bTempB	Temporary variable
BYTE g_bIndex	Temporary variable
BYTE g_bIndex2	Temporary variable
BYTE i_bTemp	This is a temporary variable used in ISR.
BYTE i_timercount	This is used to record the counting status of the timer 0.
WORD g_wTemp	Temporary variable
WORD g_wTempA	Temporary variable
WORD g_wDataLength	This indicates the data length when in USB data stage.
WORD g_wDataOffset	This indicates the offset of the data which has been consumed.
WORD g_wGpioDirMask	This is used to store directions of all GPIOs.
WORD g_wGpioIntMask	This is used to indicate interrupt enable status of GPIOs.
WORD g_wGpiData	This is used to store the current status of all GPIOs.
WORD g_wOldGpiData	This is used to keep the old GPI data value.
BYTE idata g_bRecordMuteGpio	This is used to represent the GPIO pin of the record mute button.

7.3 Macros

Below macros are used for USB status check and set.

Macros	Required?	Description
UsbEventRst()	■	Check the end of USB reset event
UsbClrEventRst()	■	Clear the flag of the end of USB reset event
UsbEventResume()	■	Check the USB resume event
UsbClrEventResume()	■	Clear the flag of USB resume event
UsbEventEpCtrl()	■	Check the endpoint 0(control) event
UsbClrEventEpCtrl()	■	Clear the endpoint 0(control) event
UsbEventSuspend()	■	Check the USB suspend event
UsbClrEventSuspend()	■	Clear the flag of the USB suspend event
UsbEventEpInt()	■	Check the endpoint 3(interrupt) event
UsbClrEventEpInt()	■	Clear the flag of the endpoint 3 event
UsbSelectEp(ep)	■	Select USB endpoint number
UsbSetTxReady()	■	Set TX Packet Ready Control Bit

7.4 Functions

Below functions are used to handle the USB protocol communication.

Functions	Required?	Description
Standard USB functions		
void HandleUsbReset()	■	The USB reset handler This function handles the initialization of global variables, USB EP reset and the codec reset, etc.
void HandleUsbSuspend()	■	The USB suspend handler This function handlers the configuration of peripheral and codec, and the anti-pop noise flow is processed. Finally, the USB device will go into a low-power mode.
void HandleUsbResume()	■	The USB resume handler This function handles the anti-pop noise control and the codec reset.
void HubInSuspend()	■	This function is used to resolve the suspend current issue when the USB device is plugged into a hub and goes into the suspend state. The system goes into a low-power mode when the USB device enters the suspend state.
void HandleUsbCtrlTransfer()	■	The handler of USB control transfer
void HandleUsbIntTransfer()	■	The handler of USB interrupt transfer Once the interrupt transfer is done, the function is used to handler the complete event.
void InputReportDataReady()	■	Pack the HID report ID 0 Data (fixed size: 16bytes) The variable “g_bmHidEn” can be used to enable/disable the HID report submission. The HID report data format is defined. Please refer to the appendix B.
void SubmitUsbIntTransfer()	■	Submit the USB interrupt data
void usbEpReset()		Enable/Disable Endpoint (ISO IN, ISO OUT, INT IN, ISO Feedback IN)
Audio functions		
void OriginFeatureVolume()		Volume feature unit control
void OriginFeatureMute()		Mute feature unit control

void OriginSetSelector()		ADC input source selection 1. MIC In 2. Line In 3. SPDIF In 4. Stereo Mixer In
void OriginFeatureAgc()		Enable/Disable REC AGC and clipping LED
Peripheral functions		
void RefreshWdt()	■/Must	Periodically update the watchdog timer count to avoid the system reset
void CodecReset()	■	Internal codec reset
void PeriClkReset()		Peripheral clock reset and gating (optional) 1. IR clock gated 2. SPDIF IN clock gated 3. SPDIF OUT clock gated 4. Playback or Record logic clock gated
void ModIrReset(BYTE type)		IR decoder mode reset Parameter 0 "type" can be: 0. NEC 1. RC5 2. RC6
void OriginIr()		Specific NEC IR code detection handler. Below codes are supported: 1. volume up 2. volume down 3. play mute 4. play pause 5. stop 6. next 7. Previous
void DelayAnalogMute()	■	Un-mute the codec mixer/playback paths after a period of time

NOTE1: “■/Must” means the function is a required function which needs to be added into the customized application.
NOTE2: “■” means the function is required function for the development of an USB audio device. However, if a standalone device is developed, the function is not required.

8. Interface of Customized Application

In “Framework\CM65xxB-1\”, major header files are inside “inc” folder, which define structures, macros and functions for standard CM65xx’s USB and peripheral control.

8.1 Entry-point Functions

After the ROM program finishes downloading the firmware from EEPROM to internal program memory, the ROM program will jump to this function for customized code flow. The function must be implemented.

Functions	Required ?	Description
void main()	■/Must	The entry-point function. The ROM program will jump to this function for the customized FW.

8.2 Interrupt Handler Functions

The 8051 and its derivatives provide a number of hardware interrupts that may be used for counting, timing, detecting external events, and sending and receiving data using the serial interface. The standard interrupts found on an 8051 are listed in the following table:

Interrupt number	Name	Description	Address
0	?INT0_ISR	External int 0	8000h
1	?TIMER0_ISR	Timer/counter 0	8003h
2	?INT1_ISR	External int 1	8006h
3	?TIMER1_ISR	Timer/counter 1	8009h
4	?UART_ISR	Serial port	800ch

Once the hardware interrupt event happens, the corresponding function will be called and handle the related tasks.

Functions	Required?	Description
void INT0_ISR ()		Customized external INT0 interrupt handler
void TIMER0_ISR ()		Customized Timer0 interrupt handler
void TIMER1_ISR ()		Customized Timer1 interrupt handler
void INT1_ISR ()		Customized external INT1 interrupt handler
void UART_ISR ()		Customized Uart interrupt handler

8.3 Customized Functions

Some functions need to be implemented in customized program.

Functions	Required?	Description
Peripheral functions		
void PowerOnReset()	■	The function for power-on initialization: 1. System clock configuration 2. Interrupt configuration 3. GPIO configuration 4. Global variable initialization
USB Handler Functions		
BOOL tackleClassCommand(BOOL dataStage)		This function handles the class request command.
BOOL tackleHidGetReport()		This function handles HID_GET_REPORT.
BOOL tackleHidSetReport()		This function handles HID_SET_REPORT.
BOOL ExTackleHidSetReportData()		This function handles HID_SET_REPORT.
void ExHandleUsbCtrlTransfer()		This function handles the USB control transfer.
BOOL tackleControlRequest(BOOL dataStage)		This function handles the USB control request.
BOOL tackleGetDescriptor(BOOL dataStage)		This function handles the string descriptor information.

9. Configuration definition

In “Framework\CM65xxB-1_xxx\inc\config.h”, there are macros for configuring the device. They are listed below.

Macros	Description
VENDOR_ID	Define vendor ID of the USB device.
PRODUCT_ID	Define product ID of the USB device.
VERSION_ID	Define version ID of the USB device.

10. Link file

The link file “cm65xx_cmd.lin” is a command file that may contain an *inputlist*, *outputfile*, and *directives*. The LX51 Linker/Locator uses this file to output the absolute object module.

Below definitions with red mark is for customization. The customized FW is located in the memory area.

```
.\output\audio.c.obj,.\output\dscr.a51.obj,.\output\int.c.obj,
.\output\io.c.obj,.\output\main.c.obj,.\output\request.c.obj,.\output\usb.c.obj,
..\CM65xxB-1\CM65xxB-1.LIB to .\output\cm65xxfw
CLASSES
(
    CODE(C:0x8000-C:0xBFFF),
    CONST(C:0x8000-C:0xBFFF),
    XDATA_EXTMEM(X:0x0A60-X:0x0BDF)
)

&

SEGMENTS
(
    ?STACK(D:0xE9),
    ?PR?INT0_ISR?INT(C:0x8000),
    ?PR?TIMER0_ISR?INT(C:0x8003),
    ?PR?INT1_ISR?INT(C:0x8006),
    ?PR?TIMER1_ISR?INT(C:0x8009),
    ?PR?UART_ISR?INT(C:0x800C),
    ?PR?MAIN?MAIN(C:0x8012)
)
```

---

11. Programming Notes

1. Stack memory usage
   - Key factors
     i. Stack size initialization can be defined by customers. Customers should be aware of this to avoid memory overlap.
     ii. Function call level: More function call level affects the stack memory usage.
   - Stack overflow check
     i. Review the compiler log and map file.
     ii. Use KeilC simulator to check max stack pointer (sp_max) in specific functions.
     iii. Fill known characters in the bottom of stack, and run real-time check if the context is polluted.
     iv. Put the stack pointer in a global variable inside ISRs to check if stack overflow happens.
     v. Print out the stack pointer information in some critical functions using HID or Uart.
2. Interrupt handler usage
   - Do **NOT** enable the event of endpoint interrupt. Please use the polling mechanism to check the endpoint status instead.
   - There are five hardware interrupts supported. They are open for customization. However, customers can define their own tasks inside the interrupt handler functions. Suggest not to change the “INT0” and “INT1” handlers.
3. Standalone mode
   - If no USB audio design is required, customers can use CM65xx as a standalone device without USB functions.
4. Anti-pop noise
   - The standard anti-pop noise control function is included in CM65xx standard library.
5. USB reset impact
   - Most internal registers are reset once USB reset event happens. The codec registers are not affected. Remember to set some critical registers after USB reset.
   - USB reset will impact the GPIO temporary state. Please disable GPIO debounce function if the GPI state check is required to during USB reset event.

6. C51 lib code
    - The C51 library “C51S.LIB” is required for ANSI C function calls. Therefore, if some ANSI C functions are included, the code size of the object codes of C51 library will increase.
7. Extra xdata memory usage
    - In addition to the internal 256-byte data memory, extra xdata memory area can be used. Please refer to the memory mapping. For example, the memory for bulk transfer which start address is 0xA60 can be used if no bulk transfer is required.
8. Extended GPIO usage
    - GPIO 16~23 is controlled by MCU Port1. GPIO 24~31 is controlled by MCU Port2.
    - The IO pin needs to be pulled high if high-level drive is required.

12. Design an USB Audio Device

With CM65xx firmware platform, it is easy to build up a new USB audio device. Firmware engineers have to perform only several main tasks:

(1) Building up all USB descriptors for full-speed. The descriptor file “dscr.a51” in “Framework\CM65xxB-1_xxx\” can be as a reference.

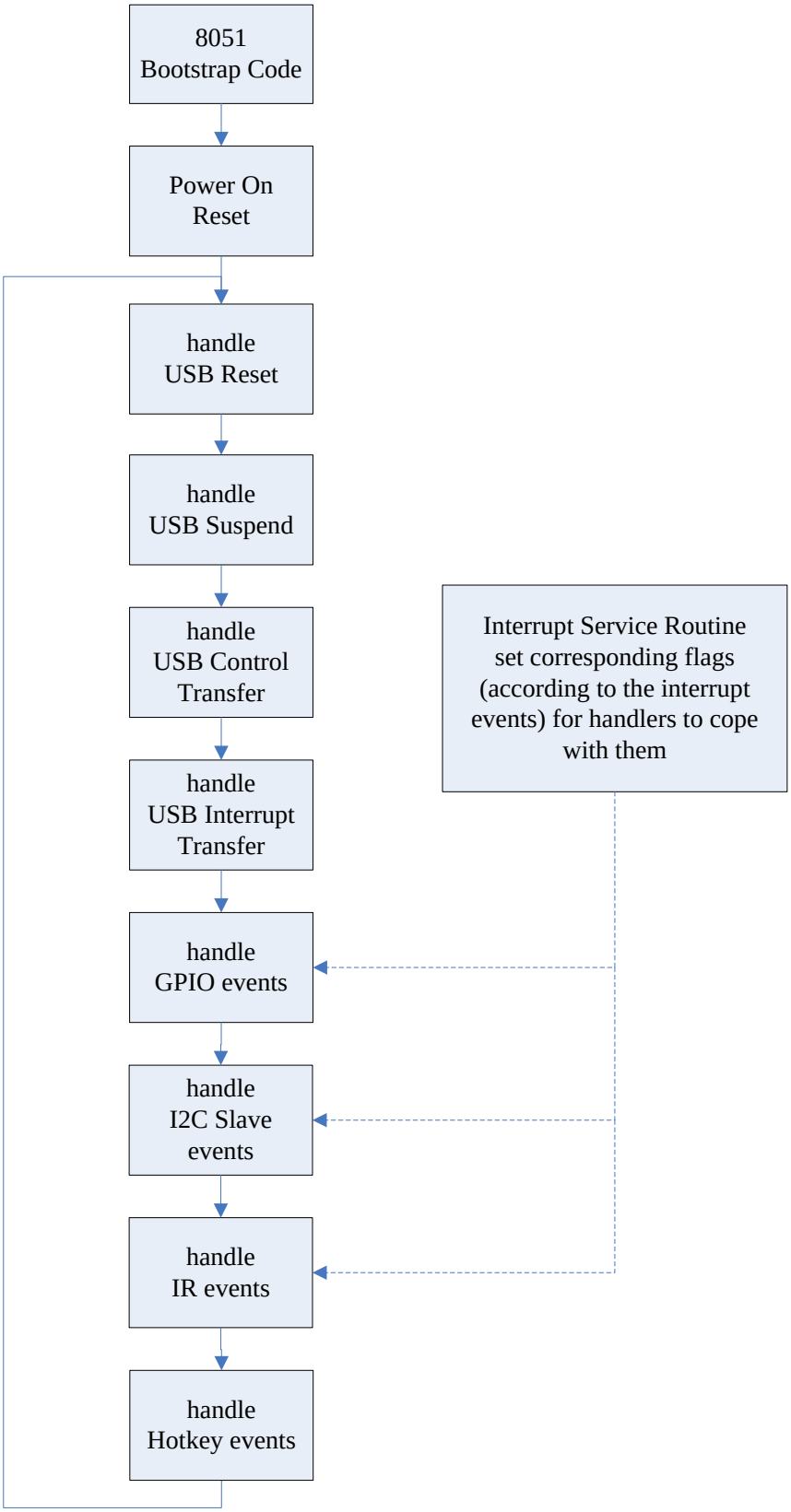
(2) Creating an instance of USB_DEVICE structure corresponding to the USB descriptors. "main.c” is an entry file of how USB_DEVICE is created.

(3) Implementing all customized functions that CM65xx framework needs. Firmware engineers can modify source files in sample application.

Then designer can get hex files for loading and testing by executing the batch file of building firmware.

In “Framework\CM65xxB-1_xxx\”, the files “audio.c” and “request.c” implement protocols of USB audio class 1.0 and internal codec control. If the topology or the configuration of the audio device is changed, these files are necessary to be modified to meet the USB descriptors. If CM65xx cooperates with different I2S codec, firmware engineers have to modify “audio.c” for the new codec.

13. The Example of Customized Firmware Flow

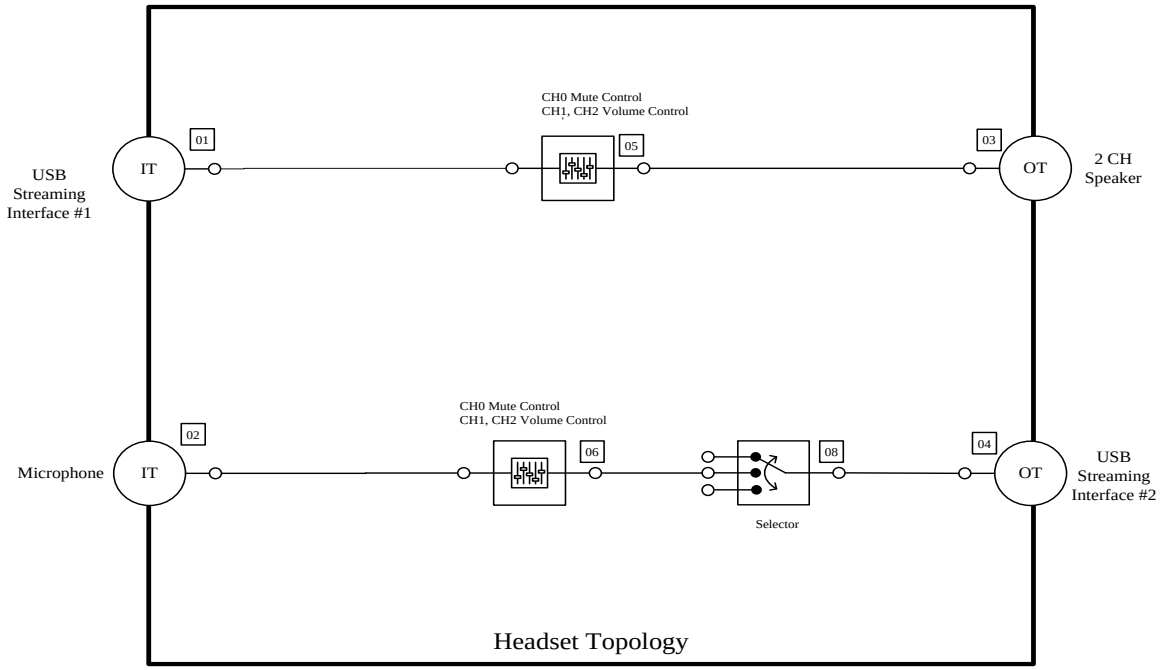


Appendix

Appendix describes specifications of the USB device implemented by sample applications.

A. Headset Configuration

- **Audio Topology**



Headset Topology

- **USB Interfaces List**

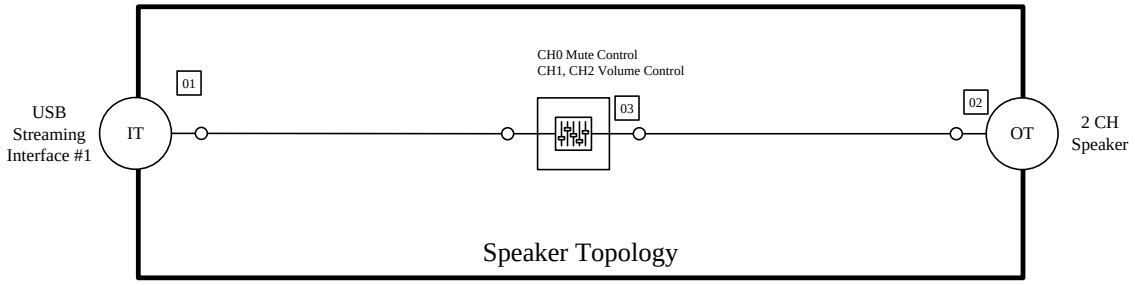
Interface #	Interface Description	Endpoint
Interface 0	Audio Control Interface of Audio Device	0x00 (Control)
Interface 1	Audio Stream Interface for playback	0x01 (Iso. Out)
Interface 2	Audio Stream Interface for recording	0x82 (Iso. In)
Interface 3	HID Interface	0x87 (Interrupt In)

- **Audio Stream Interfaces' Alternate Setting List for Full-Speed**

Interface #	Alternate Setting	Data Format	Sampling Rate (Hz)
Interface 1	Alt 1	2CH, 16Bits PCM	8000, 11025, 16000, 22050, 32000, 44100, 48000
Interface 2	Alt 1	2CH, 16Bits PCM	8000, 11025, 16000, 22050, 32000, 44100, 48000

B. Speaker Configuration

- **Audio Topology**



Speaker Topology

- **USB Interfaces List**

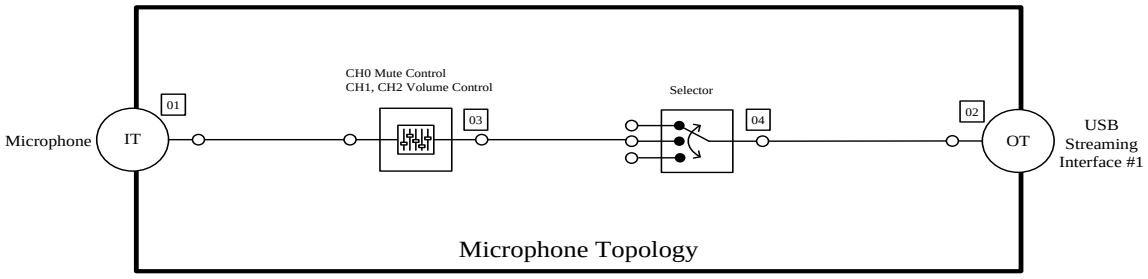
Interface #	Interface Description	Endpoint
Interface 0	Audio Control Interface of Audio Device	0x00 (Control)
Interface 1	Audio Stream Interface for playback	0x01 (Iso. Out)
Interface 2	HID Interface	0x87 (Interrupt In)

- **Audio Stream Interfaces' Alternate Setting List for Full-Speed**

Interface #	Alternate Setting	Data Format	Sampling Rate (Hz)
Interface 1	Alt 1	2CH, 16Bits PCM	8000 11025 16000 22050 32000 44100 48000

C. Microphone Configuration

- **Audio Topology**



Microphone Topology

- **USB Interfaces List**

Interface #	Interface Description	Endpoint
Interface 0	Audio Control Interface of Audio Device	0x00 (Control)
Interface 1	Audio Stream Interface for recording	0x81 (Iso. In)
Interface 2	HID Interface	0x87 (Interrupt In)

- **Audio Stream Interfaces' Alternate Setting List for Full-Speed**

Interface #	Alternate Setting	Data Format	Sampling Rate (Hz)
Interface 1	Alt 1	2CH, 16Bits PCM	8000 11025 16000 22050 32000 44100 48000

D. HID Interface

This section describes the details of HID interface.

- **HID Report Descriptor**

```
db    05H, 0CH                ;;Usage Page(Consumer)
db    09H, 01H                ;;Usage(Consumer Control)
db    0A1H, 01H               ;;Collection(Application)
db    85H, 01H                ;;Report ID(1)
db    15H, 00H                ;;Logical Min.(0)
db    25H, 01H                ;;Logical Max.(1)
db    09H, 0E9H               ;;Usage(Vol. Increment)
db    09H, 0EAH               ;;Usage(Vol. Decrement)
db    75H, 01H                ;;Report Size(1)
db    95H, 02H                ;;Report Count(2)
db    81H, 42H                ;;Input(Data, Variable, Absolute,
Null state)
db    09H, 0E2H               ;;Usage(Mute)
db    95H, 01H                ;;Report Count(1)
db    81H, 06H                ;;Input(Data, Variable, Relative)
db    06H, 01H, 0FFH          ;;Usage Page(Vendor Defined)
db    09H, 01H                ;;Usage(Vendor1 ??)
db    95H, 09H                ;;Report Count (9)
db    81H, 06H                ;;Input(Data, Variable, Relative)
db    05H, 0CH                ;;Usage Page(Consumer)
db    09H, 0CDH               ;;Usage(Play/Pause)
db    09H, 0B7H               ;;Usage(Stop)
db    09H, 0B5H               ;;Usage(Scan Next Track)
db    09H, 0B6H               ;;Usage(Scan Previous Track)
db    95H, 04H                ;;Report Count(4)
db    81H, 06H                ;;Input(Data, Variable, Relative)
db    06H, 07H, 0FFH          ;;Usage Page(Vendor Defined)
db    09H, 01H                ;;Usage(Vendor1 ??)
db    75H, 08H                ;;Report Size(8)
db    95H, 0DH                ;;Report Count(13)
db    81H, 06H                ;;Input(Data, Variable, Relative)
db    09H, 00H                ;;Usage(Undefined)
db    95H, 0FH                ;;Report Count(15)
```

```
db      91H, 02H                    ;;Output(Data, Variable, Absolute)
db      0C0H                        ;;End Collection
```

- **HID Input Report**

The 16-bytes input report is defined as the following table. Host will be notified by an input report via interrupt pipe. Host can also get input report with class request “Get Report” via control pipe.

Host can read registers of CM65xx by HID input report’s byte 6~byte 15. Start address and length of registers that host reads can be set by sending output report.

	Description	**Size**
Byte 0	Report ID (Always 1)	1
Byte 1~Byte 2	For defined HID event, and each event occupies one bit (this depends on HID report descriptor)	2
Byte 3	start address of returned data (H-start_addr)	1
Byte 4	start address of returned data (L-start_addr)	1
Byte 5	Interrupt source. Bit 7: Reserved Bit 6: UART_INT Bit 5: GPI_INT Bit 4: SPIS_INT Bit 3: SPIM_INT Bit 2: I2CS_INT Bit 1: I2CM_INT Bit 0: IR_INT	1
Byte 6~Byte15	Register content	10

- **HID Output Report**

HID output report is designed for writing registers to CM65xx. It is also used for setting start address and length of registers sent to host in input report.

	Description	Size
Byte 0	Report ID (Always 1)	5
Byte 1	1. 0x00: Set register read in input report 2. start address of returned data (H-start_addr)	1
Byte 2	1. 0xFE: Set register read in input report 2. start address of returned data (L-start_addr)	1

Byte 3	Effective write data length (<= 12) Effective read data length (<= 10)	1
Byte 4	1. If Byte1 is 0x00 and Byte2 is 0xFE, this byte is the value set to “Register Address(H)” in input report. 2. If Byte2 is not 0xFE, this byte is data written to register.	1
Byte 5	1. If Byte1 is 0x00 and Byte2 is 0xFE, this byte is the value set to “Register Address(L)” in input report. 2. If Byte2 is not 0xFE, this byte is data written to register.	1
Byte 6~Byte15	Byte 7~Byte15 are data written to register.	10

E. Vendor Requests

The vendor requests implemented in the demo application are listed in the following table.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
01000000b	Write Register 01h	Address offset	0	Byte count	Content of register
11000000b	Read Register 02h	Address offset	0	Byte count	Content of register