RE01 1500KB Group

R01AN4944EJ0100
Rev.1.00
Sep 30, 2019

USB Basic Peripheral Driver Specification

Introduction

This documentation describes the specification of USB Basic Periphral Driver of CMSIS software package.

Target Device

RE01 1500KB Group

Contents

1. Overview

This driver performs USB hardware control and operates in combination with device class drivers.

This driver supports the following functions.

- Supporting CMSIS USB Driver specification.
- Supporting USB Peripheral.
- Device connect/disconnect, suspend/resume, and USB bus reset processing.
- Control transfer on pipe 0.
- Data transfer on pipes 1 to 9. (Bulk or Interrupt transfer)
- In peripheral mode, enumeration as USB Host of USB1.1/2.0/3.0.

1.1 Note

This application note is not guaranteed to provide USB communication operations. The customer should verify operations when utilizing the USB device module in a system and confirm the ability to connect to a variety of different types of devices.

1.2 Limitations

This driver is subject to the following limitations.

1. Multiple configurations are not supported.
2. Multiple interfaces are not supported.
3. This USB driver does not support the error processing when the out of specification values are specified to the arguments of each function in the driver.
4. This driver does not support the CPU transfer using D0FIFO/D1FIFO register.

1.3 Terms and Abbreviations

APL	:	Application program
H/W	:	Renesas USB device
Non-OS	:	USB Driver for OS-less
PCD	:	Peripheral Control Driver of USB-BASIC-FW
PDCD	:	Peripheral Device Class Driver (Device driver and USB class driver)
USB-BASIC-FW	:	USB Basic Peripheral Driver

1.4 Software Configuration

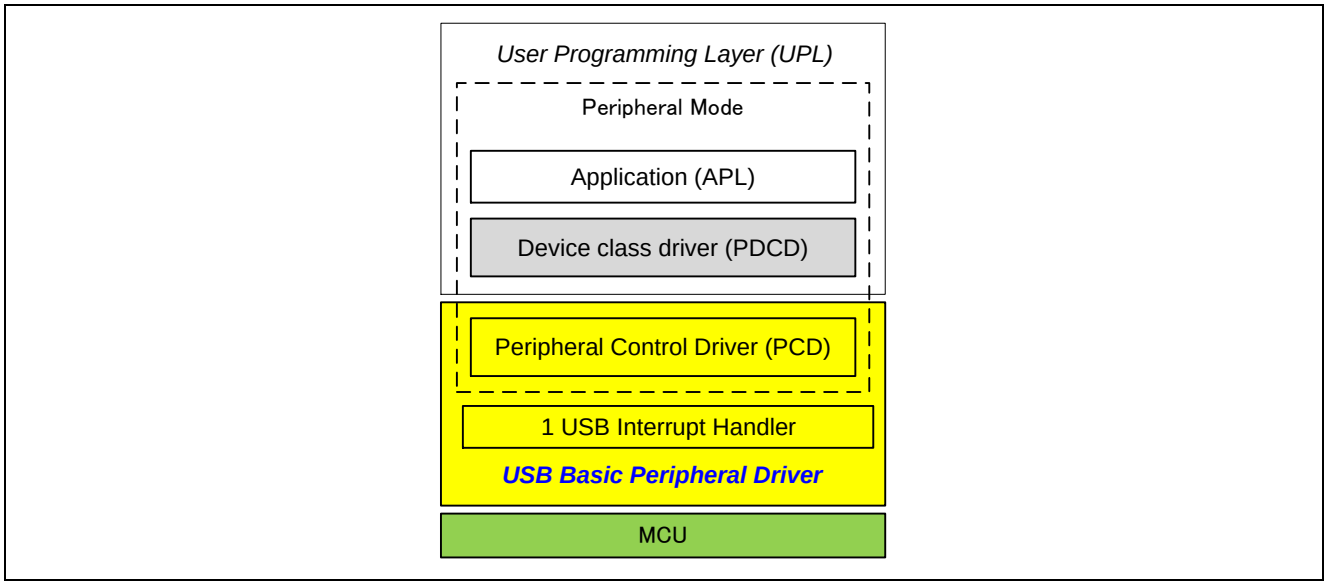

Figure 1-1 USB Driver Software Structure

Table 1-1 Software function overview

No	Module Name	Function
1	H/W Access Layer	Hardware control
2	USB Interrupt Handler	USB interrupt handler (USB packet transmit/receive end and special signal detection)
3	Peripheral Control Driver (PCD)	Hardware control in peripheral mode Peripheral transaction management
4	Device Class Driver	Provided by the customer as appropriate for the system.
5	Device Driver	Provided by the customer as appropriate for the system.
6	Application	Provided by the customer as appropriate for the system.

1.5 Pin Setting

To use the USB driver, input/output signals of the peripheral function has to be allocated to pins with the multi-function pin controller (MPC).

2. Peripheral

2.1 Peripheral Control Driver (PCD)

2.1.1 Basic functions

PCD is a program for controlling the hardware. PCD analyzes requests from PDCD and controls the hardware accordingly. It also sends notification of control results using a user provided call-back function. PCD also analyzes requests from hardware and notifies PDCD accordingly.

PCD accomplishes the following:

1. Control transfers. (Control Read, Control Write, and control commands without data stage.)
2. Data transfers. (Bulk, interrupt) and result notification.
3. Data transfer suspensions. (All pipes.)
4. USB bus reset signal detection and reset handshake result notifications.
5. Suspend/resume detections.
6. Attach/detach detection using the VBUS interrupt.

2.1.2 Issuing requests to PCD

API functions are used when hardware control requests are issued to the PCD and when performing data transfers.

Refer to chapter **3, API**.

2.2 API (Application Programming Interface)

For the API function, refer to chapter **3, API**.

2.3 Request Processing

For the USB request processisng procedure, refer to chapter **8.3, Enumeration (USB Request) Processing**.

2.4 Descriptor

2.4.1 String Descriptor

This USB driver requires each string descriptor that is constructed to be registered in the string descriptor table. The following describes how to register a string descriptor.

1. First construct each string descriptor. Then, define the variable of each string descriptor in uint8_t* type.

Example descriptor construction)

```
uint8_t smp_str_descriptor0[] {
      0x04, /* Length */
      0x03, /* Descriptor type */
      0x09, 0x04 /* Language ID */
};
uint8_t smp_str_descriptor1[] =
{
      0x10, /* Length */
      0x03, /* Descriptor type */
      'R', 0x00,
      'E', 0x00,
      'N', 0x00,
      'E', 0x00,
      'S', 0x00,
      'A', 0x00,
      'S', 0x00
};
uint8_t smp_str_descriptor2[] =
{
```

```
        0x12, /* Length */
        0x03, /* Descriptor type */
        'C', 0x00,
        'D', 0x00,
        'C', 0x00,
        '_', 0x00,
        'D', 0x00,
        'E', 0x00,
        'M', 0x00,
        'O', 0x00
};
```

2. Set the top address of each string descriptor constructed above in the string descriptor table. Define the variables of the string descriptor table as uint8_t* type.

Note:

The position set for each string descriptor in the string descriptor table is determined by the index values set in the descriptor itself (iManufacturer, iConfiguration, etc.).
For example, in the table below, the manufacturer is described in smp_str_descriptor1 and the value of iManufacturer in the device descriptor is “1”. Therefore, the top address "smp_str_descriptor1" is set at Index “1” in the string descriptor table.

```
/* String Descriptor table */
uint8_t *smp_str_table[] =
{
    smp_str_descriptor0,    /* Index: 0 */
    smp_str_descriptor1,    /* Index: 1 */
    smp_str_descriptor2,    /* Index: 2 */
};
```

2.4.2 Other Descriptors

Please construct the device descriptor, configuration descriptor, and qualifier descriptor based on instructions provided in the **Universal Serial Bus Revision 2.0 specification**(http://www.usb.org/developers/docs/) Each descriptor variable should be defined as uint8_t* type.

2.4.3 Descriptor Transmission

When the *GetDescriptor* request sent from USB Host is received, the application program gets the descriptor tyep after analyzing the *GetDescriptor* request. Please send USB Host the descriptor correspoing to the descriptor type using *ARM_USBD_EndpointTransfer* function (API).

Example) Case of Device Descriptor

```
ARM_USBD_EndpointTransfer ((USB_EP_IN | USB_EP0), g_device_descriptor, 18);
```

1. Specify the endpoint number 0 (zero) to the 1st argument.
2. Specify the start address of the area stored the descriptor to the 2nd argument. If the descriptor is the string descriptor, specify the start address of the string descriptor table.
3. Specify the byte count of the descriptor to the 3rd argument.

3. API

3.1 API List

Table 3-1 provides a list of API functions. Use the APIs below in application programs.

Table 3-1 List of API Functions

API	Description
ARM_USBD_Initialize	Initialize USB Device Interface.
ARM_USBD_Uninitialize	Uninitialize USB Device Interface.
ARM_USBD_PowerControl	Power ON or Power OFF Setting of USB Module.
ARM_USBD_DeviceConnect	Pull up enable setting of D+ Line.
ARM_USBD_DeviceDisConnect	Pull up disable setting of D+ Line.
ARM_USBD_DeviceGetState	Get USB Device State.
ARM_USBD_DeviceRemoteWakeup	Set Remote Wakeup.
ARM_USBD_ReadSetupPacket	Read Setup Packet.
ARM_USBD_EndpointConfigure	Configure the specifed endpoint.
ARM_USBD_EndpointUnconfigure	Unconfigure the specified endpoint.
ARM_USBD_EndpointStall	Set or Clear stall for the specified endpoint.
ARM_USBD_EndpointTransfer	Data transfer reuqest to the specified endpoint.
ARM_USBD_EndpointTransferGetResult	Get the reception data size
ARM_USBD_EndpointTransferAbort	Data transfer abort request to the specified endpoint.
ARM_USBD_GetFrameNumber	Get USB Frame Number.
ARM_USBD_GetCapabilities	Get the setting information to *ARM_USBD_CAPABILITIES* structure.
ARM_USBD_GetVersion	Get the USB driver version
R_USB_ControlDriver	Execution of USB driver processing

Note:

1. This driver does not support *ARM_USBD_DeviceSetAddress* function since USB module supports the auto response function for *SetAddress* request.
2. For *ARM_USBD_STATE/ARM_USBD_CAPABILITIES* structure and the API other than *R_USB_ControlDriver* function, refer to the follwoing URL.

   https://arm-software.github.io/CMSIS_5/Driver/html/group__usbd__interface__gr.html

3.2 ARM_USBD_Initialize

Table 3-2 ARM_USBD_Initialize Function Specification

Outline	int32_t ARM_USBD_Initialize(ARM_USBD_SignalDeviceEvent_t cb_device_event, ARM_USBD_SignalEndpointEvent_ cb_endpoint_event)
Description	USB Driver initialization (RAM initialization, Register setting, Callback function registration etc)
Argument	ARM_USBD_SignalDeviceEvent_t : Pointer to the callback function ARM_USBD_SignalEndpointEvent_t : Pointer to the callback function
Retrun Value	ARM_DRIVER_OK : Success

Remark	[Note]
	Do not call this API in the interrupt function.
	
	[Function calling example from the instance]
	static void pmsc_signal_device_event(uint32_t event);
	static void pmsc_signal_endpoint_event (uint8_t ep_addr, uint32_t ep_event);
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->Initialize (&pmsc_signal_device_event,
	&pmsc_signal_endpoint_event);
	}

3.3 ARM_USBD_Uninitialize

Table 3-3 ARM_USBD_Uninitialize Function Specification

Outline	int32_t ARM_USBD_UnInitialize(void)
Description	Release USB driver
Argument	None
Retrun Value	ARM_DRIVER_OK : Success

Remark	[Note]
	Do not call this API in the interrupt function.
	
	[Function calling example from the instance]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->Uninitialize ();
	}

3.4 ARM_USBD_PowerControl

Table 3-4 ARM_USBD_PowerControl Function Specification

Outline	int32_t ARM_USBD_PowerControl(ARM_POWER_STATE state)		
Description	Set or clear USB module stop bit.		
Argument	ARM_POWER_FULL ARM_POWER_OFF	: :	Clear USB module stop bit Set USB module stop bit
Return Value	ARM_DRIVER_OK ARM_DRIVER_ERROR_UNSUPPORTED	: :	Success Unsuppoted error

Remark	[Note]
	Do not call this API in the interrupt function.
	
	[Function calling example from the instance]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->PowerControl(ARM_POWER_FULL);
	}		

3.5 ARM_USBD_DeviceConnect

Table 3-5 ARM_USBD_DeviceConnect Function Specification

Outline	int32_t ARM_USBD_DeviceConnect(void)
Description	Pull up enable setting for D+ line
Argument	None
Return Value	ARM_DRIVER_OK : Success

Remark	[Note]
	Do not call this API in the interrupt function.
	
	[Function calling example from the instance]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->DeviceConnect ();
	}

3.6 ARM_USBD_DeviceDisconnect

Table 3-6 ARM_USBD_DeviceDisconnect Function Specification

Outline	int32_t ARM_USBD_DeviceDisconnect(void)
Description	Pull up disable setting for D+ line
Argument	None
Return Value	ARM_DRIVER_OK : Success

Remark	[Note]
	Do not call this API in the interrupt function.
	
	[Function calling example from the instance]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->DeviceDisconnect ();
	}

3.7 ARM_USBD_DeviceGetState

Table 3-7 ARM_USBD_DeviceGetState Function Specification

Outline	ARM_USBD_STATE ARM_USBD_DeviceGetState(void)
Description	Get USB device state
Argument	None
Return Value	USB device state (*ARM_USBD_STATE* structure)

Remark	[Function calling example from the instance]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	ARM_USBD_STATE usb_state;
	usb_state = gp_cmsis_usb_driver->DeviceGetState ();
	}

3.8 ARM_USBD_DeviceRemoteWakeup

Table 3-8 ARM_USBD_DeviceRemoteWakeup Function Specification

Outline	int32_t ARM_USBD_DeviceRemoteWakeup(void)
Description	Remote Wakeup process
Argument	None
Return Value	ARM_DRIVER_OK : Success ARM_DRIVER_ERROR_BUSY : Remote Wakeup process in progress ARM_DRIVER_ERROR : Other error

Remark	[Note]
	Do not call this API in the interrupt function.
	
	[Function calling example from the instance]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->DeviceRemoteWakeup ();
	}

3.9 ARM_USBD_ReadSetupPacket

Table 3-9 ARM_USBD_ReadSetupPacket Function Specification

Outline	int32_t ARM_USBD_ReadSetupPacket(uint8_t *setup)
Description	Read Setup packet
Argument	Pointer to the area for storing Setup packet
Return Value	ARM_DRIVER_OK : Success

Remark	[Note]
	Do not call this API in the interrupt function.
	
	[Function calling example from the instance]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->ReadSetupPacket (&usb_setup);
	}

3.10 ARM_USBD_EndpointConfigure

Table 3-10 ARM_USBD_EndpointConfigure Function Specification

Outline	int32_t ARM_USBD_EndpointConfigure(uint8_t ep_addr, uint8_t ep_type, uint16_t ep_maxpacket_size)
Description	Configure USB endpoint.
Argument	Endpoint Address b7: Transfer Direction(1: IN, 0: OUT), b6-b0: Endpoint Number
	Transfer Type ARM_USB_ENDPOINT_BULK / ARM_USB_ENDPOINT_INTERRUPT
	MaxPacketSize
Return Value	ARM_DRIVER_OK : Success ARM_DRIVER_ERROR_PARAMETER : Parameter error

Remark	[Note]
	Do not call this API in the interrupt function.
	
	[Function calling example from the instance]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->EndpointConfigure (
	USB_EP_IN|USB_EP2,
	ARM_USB_ENDPOINT_BULK,
	64
	);
	}

3.11 ARM_USBD_EndpointUnconfigure

Table 3-11 ARM_USBD_EndpointUnconfigure Function Specification

Outline	int32_t ARM_USBD_EndpointUnconfigure(uint8_t ep_addr)
Description	Unconfigure USB endpoint
Argument	Endpoint Address b7: Transfer Direction(1: IN, 0: OUT), b6-b0: Endpoint Number
Return Value	ARM_DRIVER_OK : Success ARM_DRIVER_ERROR_PARAMETER : Parameter error

Remark	[Note]
	Do not call this API in the interrupt function.
	
	[Function calling example from the instance]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->EndpointUnonfigure (USB_EP_IN|USB_EP2);
	}

3.12 ARM_USBD_EndpointStall

Table 3-12 ARM_USBD_EndpointStall Function Specification

Outline	int32_t ARM_USBD_EndpointStall(uint8_t ep_addr, bool stall)
Description	Set or Clear USB endpoint
Argument	Endpoint Address b7: Transfer Direction(1: IN, 0: OUT), b6-b0: Endpoint Number false (Clear) / true (Set)
Return Value	ARM_DRIVER_OK : Success ARM_DRIVER_ERROR_PARAMETER : Parameter error

Remark	[Note]
	Do not call this API in the interrupt function.
	
	[Function calling example from the instance]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->EndpointStall (USB_EP_IN|USB_EP2, true);
	}

3.13 ARM_USBD_EndpointTransfer

Table 3-13 ARM_USBD_EndpointTransfer Function Specification

Outline	int32_t ARM_USBD_EndpointTransfer(uint8_t ep_addr, uint8_t *data, uint32_t num)
Description	Data transfer request to USB endpoint
Argument	Endpoint Address b7: Transfer Direction(1: IN, 0: OUT), b6-b0: Endpoint Number Pointer to the area for storing the data Data size
Return Value	ARM_DRIVER_OK : Success ARM_DRIVER_ERROR_BUSY : Data tranfer request in progress ARM_DRIVER_ERROR_PARAMETER : Parameter error ARM_DRIVER_ERROR_BUSY : Other error

Remark	[Note]
	Do not call this API in the interrupt function.
	
	[Function calling example from the instance]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	uint8_t g_buf[64];
	
	void main(void)
	{
	gp_cmsis_usb_driver->EndpointTransfer (USB_EP_IN|USB_EP2, g_buf, 64);
	}

3.14 ARM_USBD_EndpointTransferGetResult

Table 3-14 ARM_USBD_EndpointTransferGetResult Function Specification

Outline	uint32_t ARM_USBD_EndpointTransferGetResult(uint8_t ep_addr)
Description	Get the data transfer size of the specified endpoint
Argument	Endpoint Address b7: Transfer Direction(1: IN, 0: OUT), b6-b0: Endpoint Number
Return Value	ARM_DRIVER_OK : Success ARM_DRIVER_ERROR_PARAMETER : Parameter error

Remark	[Note]
	1. This API can not get the data transfer size when the data transfer was aborted by *ARM_USBD_EndpointTransferAbort* function.
	2. Do not call this API in the interrupt function.
	
	[Function calling example from the instance]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	uint32_t usb_size;
	usb_size = gp_cmsis_usb_driver->EndpointTransferGetResult
	(USB_EP_OUT|USB_EP1);
	}

3.15 ARM_USBD_EndpointTransferAbort

Table 3-15 ARM_USBD_EndpointTransferAbort Function Specification

Outline	int32_t ARM_USBD_EndpointTransferAbort(uint8_t ep_addr)
Description	Data transfer abort request to USB endpoint
Argument	Endpoint Address b7: Transfer Direction(1: IN, 0: OUT), b6-b0: Endpoint Number
Return Value	ARM_DRIVER_OK : Success ARM_DRIVER_ERROR_PARAMETER : Parameter error

Remark	[Note]
	Do not call this API in the interrupt function.
	[Function calling example from the instance]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->EndpointTransferAbort(USB_EP_IN|USB_EP2);
	}

3.16 ARM_USBD_GetFrameNumber

Table 3-16 ARM_USBD_GetFrameNumber Function Specification

Outline	uint16_t ARM_USBD_GetFrameNumber(void)
Description	Get Frame Number
Argument	None
Return Value	Frame Number

Remark	[Function calling example from the instance]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	uint16_t frame_number;
	frame_number = gp_cmsis_usb_driver->FrameNumber();
	}

3.17 ARM_USBD_GetCapabilities

Table 3-17 ARM_USBD_GetCapabilities Function Specification

Outline	ARM_USBD_CAPABILITIES ARM_USBD_GetCapabilities(void)
Description	Get USB driver function
Argument	None
Return Value	USB driver function (*ARM_USBD_CAPABILITIES* structure)

Remark	[Function calling example from the instance]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	ARM_USBD_CAPABILITIES capabilities;
	capabilities = gp_cmsis_usb_driver->GetCapabilities();
	}

3.18 ARM_USBD_GetVersion

Table 3-18 ARM_USBD_GetVersion Function Specification

Outline	ARM_DRIVER_VERSION ARM_USBD_GetVersion(void)
Description	Get USB driver version
Argument	None
Return Value	USB driver version number (*ARM_DRIVER_VERSION* structure)

Remark	[Function calling example from the instance]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	ARM_DRIVER_VERSION version;
	version = gp_cmsis_usb_driver->GetVersion();
	}

3.19 R_USB_ControlDriver

Table 3-19 R_USB_ControlDriver Function Specification

Outline	void R_USB_ControlDriver(void)
Description	Execute USB driver
Argument	None
Return Value	None

Remark	[Note]
	1. Call this API in the main loop of the application program
	2. Do not call this API in the interrupt function.
	
	[Example of calling function]
	void main(void)
	{
	while(1)
	{
	R_USB_ControlDriver();
	:
	}
	}

4. Callback Function

USB driver calls the following callback function when USB event completes. Fo USB event, refer to chpater **4.3, USB Completion Event**.

4.1 ARM_USBD_SignalDeviceEvent

Outline	void	ARM_USBD_SignalDeviceEvent(uint32_t event)
Description	Callback function called when USB event completed	
Argument	event	USB completion event ARM_USBD_EVENT_VBUS_ON ARM_USBD_EVENT_VBUS_OFF ARM_USBD_EVENT_RESET ARM_USBD_EVENT_SUSPEND ARM_USBD_EVENT_RESUME
Return Value	None	

4.2 ARM_USBD_SignalEndpointEvent

Outline	void	ARM_USBD_SignalDeviceEvent(uint8_t ep_addr, uint32_t event)
Description	Callback function called when USB event completed	
Argument	ep_addr	Endpoint Address where USB evet completed b7: Transfer Direction(1: IN, 0: OUT), b6-b0: Endpoint Numbe
	event	USB completion event ARM_USBD_EVENT_SETUP ARM_USBD_EVENT_OUT ARM_USBD_EVENT_IN
Return Value	None	

4.3 USB Completion Event

When the USB event completes, USB dirver calls *ARM_USBD_SignalDeviceEvent* function (Callback function) or *ARM_USBD_SignalEndpointEvent* function (Callback function). The USB event information is set in the argument (*event*) of these callback functions.

4.3.1 ARM_USBD_SignalDeviceEvent

When the following event is detected, USB driver calls *ARM_USBD_SignalDeviceEvent* fucntion.

Argument (*event*)	Description
ARM_USBD_EVENT_VBUS_ON	VBUS(Hi) Detection
ARM_USBD_EVENT_VBUS_OFF	VBUS(Low) Detection
ARM_USBD_EVENT_RESET	USB Reset Detection
ARM_USBD_EVENT_SUSPEND	SUSPEND Detection
ARM_USBD_EVENT_RESUME	RESUME Detection

4.3.2 ARM_USBD_SignalEndpointEvent

When the following event completes, USB driver calls *ARM_USBD_SignalEndpointEvent* fucntion.

Argument (*event*)	Description
ARM_USBD_EVENT_SETUP	Setup packet reception complete
ARM_USBD_EVENT_OUT	OUT data reception complete
ARM_USBD_EVENT_IN	IN data transmisssion complete

Note:

This callback funcion is not called when the endpoint number 0 is specified to the argument of *ARM_USB_EndpointTransferAbort* function.

4.4 Callback Function Registration

The application program must define the following 2 callback functions and register these callback functions using *ARM_USBD_Initialize* function to USB driver.

[Registration Example]

```
ARM_DRIVER_USBD    *gp_cmsis_usb_driver= &Driver_USBD0;

/* Callback Function (ARM_USBD_SignalDeviceEvent Function) */
void usb_apl_signal_device_event(uint32_t event)
{
      :
}
/* Callback Function (ARM_USBD_SignalEndpointEvent Function) */
void usb_apl_signal_endpoint_event(uint8_t ep_addr, uint32_t event)
{
      :
}

void main(void)
{
                        :
      /* Callback function registration */
       gp_cmsis_usb_driver->Initialize ( usb_apl_signal_device_event, usb_apl_signal_endpoint_event);
                        :
}
```

5. Structure

5.1 *ARM_DRIVER_USB* Structure

Type	Member
ARM_DRIVER_VERSION	(*GetVersion) (void)
ARM_USBD_CAPABILITIES	(*GetCapabilities) (void)
int32_t	(*Initialize) (ARM_USBD_SignalDeviceEvent_t cb_device_event ,ARM_USBD_SignalEndpointEvent_t cb_endpoint_event)
int32_t	(*Uninitialize) (void);
int32_t	(*PowerControl) (ARM_POWER_STATE state)
int32_t	(*DeviceConnect) (void)
int32_t	(*DeviceDisconnect) (void)
ARM_USBD_STATE	(*DeviceGetState) (void)
int32_t	(*DeviceRemoteWakeup) (void)
int32_t	(*DeviceSetAddress) (void)
int32_t	(*ReadSetupPacket) (uint8_t *setup)
int32_t	(*EndpointConfigure) (uint8_t ep_addr, uint8_t ep_type, uint16_t ep_max_packet_size)
int32_t	(*EndpointUnconfigure) (uint8_t ep_addr)
int32_t	(*EndpointStall) (uint8_t ep_addr)
int32_t	(*EndpointTransfer) (uint8_t ep_addr, uint8_t *data, uint32_t num)
uint32_t	(*EndpointTransferGetResult) (uint8_t ep_addr)
int32_t	(*EndpointTranferAbort) (uint8_t ep_addr)
uint16_t	(*GetFrameNumber) (void)

5.2 *ARM_USBD_STATE* Structure

Type	Member		Description
uint32_t	vbus:	1	VBUS state (ON: 1, OFF:0)
uint32_t	speed:	2	USB device speed
uint32_t	active:	1	USB device state (Configured State: 1, Other State: 0)
uint32_t	reserved:	28	--

5.3 *ARM_USBD_CAPABILITIES* Structure

Type	Member		Description
uint32_t	vbus_detection:	1	VBUS Detection Functon (Support:1, Not Support: 0)
uint32_t	event_vbus_on:	1	Attach Event Notification Function (Support:1, Not Support: 0)
uint32_t	event_vbus_off:	1	Detach Event Notification Function (Support:1, Not Support: 0)
uint32_t	reserved:	29	--

6. Macro Definition

Definition	Value
ARM_USBD_EVENT_VBUS_ON	(1UL << 0)
ARM_USBD_EVENT_VBUS_OFF	(1UL << 1)
ARM_USBD_EVENT_VBUS_RESET	(1UL << 2)
ARM_USBD_EVENT_HIGH_SPEED	(1UL << 3)
ARM_USBD_EVENT_SUSPEND	(1UL << 4)
ARM_USBD_EVENT_RESUME	(1UL << 5)
ARM_USBD_EVENT_SETUP	(1UL << 0)
ARM_USBD_EVENT_OUT	(1UL << 1)
ARM_USBD_EVENT_IN	(1UL << 2)
ARM_USB_SPEED_LOW	0
ARM_USB_SPEED_FULL	1
ARM_USB_SPEED_HIGH	2
ARM_USB_ENDPOINT_CONTROL	0
ARM_USB_ENDPOINT_ISOCHRONOUS	1
ARM_USB_ENDPOINT_BULK	2
ARM_USB_ENDPOINT_INTERRUPT	3

7. Configuration (r_usb_basic_mini_cfg.h)

1. USB operating mode setting

   Set *USB_CFG_PERI* for the definition of *USB_CFG_MODE*.

   ```
   #define    USB_CFG_MODE    USB_CFG_PERI
   ```

2. Device class setting

   Defines *USB_CFG_PMSC_USE* definition.

   ```
   #define    USB_CFG_PMSC_USE    // Peripheral Mass Storage Class
   ```

3. DTC use setting

   Specify whether to use the DTC.

   ```
   #define    USB_CFG_DTC    USB_CFG_ENABLE     // Uses DTC
   #define    USB_CFG_DTC    USB_CFG_DISABLE    // Does not use DTC
   ```

   Note:

   If *USB_CFG_ENABLE* is set for the definition of *USB_CFG_DTC*, be sure to set *USB_CFG_DISABLE* for the definition of *USB_CFG_DMA* in 4 below.

4. DMA use setting

   Specify whether to use the DMA.

   ```
   #define    USB_CFG_DMA    USB_CFG_ENABLE     // Uses DMA.
   #define    USB_CFG_DMA    USB_CFG_DISABLE    // Does not use DMA.
   ```

   Note:

   (1). If *USB_CFG_ENABLE* is set for the definition of *USB_CFG_DMA*, be sure to set *USB_CFG_DISABLE* for the definition of *USB_CFG_DTC* in 3 above.

   (2). If *USB_CFG_ENABLE* is set for the definition of *USB_CFG_DMA*, set the DMA Channel number for the definition in 5 below.

5. DMA Channel setting

   If *USB_CFG_ENABLE* is set in 4 above, set the DMA Channel number to be used.

   ```
   #define    USB_CFG_USB0_DMA_TX    DMA Channel number    // Transmission setting for USB0 module
   #define    USB_CFG_USB0_DMA_RX    DMA Channel number    // Transmission setting for USB0 module
   ```

   Note:

   (1). Set one of the DMA channel numbers from *USB_CFG_CH0* to *USB_CFG_CH3*. Do not set the same DMA Channel number.

   (2). If DMA transfer is not used, set *USB_CFG_NOUSE* as the DMA Channel number.

6. Interrupt Priority Level setting

   Assign the interrupt priority level of the interrupt related to USB for *USB_CFG_INTERRUPT_PRIORITY* definition.

   ```
   #define    USB_CFG_INTERRUPT_PRIORITY    3    // 1(low) – 15(high)
   ```

7. DBLB bit setting

   Set or clear the DBLB bit in the pipe configuration register (*PIPECFG*) of the USB module using the following definition. For details on the pipe configuration register (*PIPECFG*), refer to the MCU hardware manual.

   ```
   #define    USB_CFG_DBLB    USB_CFG_DBLBON     // DBLB bit set.
   #define    USB_CFG_DBLB    USB_CFG_DBLBOFF    // DBLB bit cleared.
   ```

8. RAM Allocation of Function

Setting to execute USB driver function in RAM.

```
#define   USB_CFG_FUNC     SYSTEM_SECTION_CODE          // Does not allocate function in RAM
#define   USB_CFG_FUNC     SYSTEM_SECTION_RAM_FUNC      // Allocate function in RAM
```

Note:

When the macro *SYSTEM_SECTION_RAM_FUNC* is set to the definition *USB_CFG_FUNC*, All functions of USB driver are allocated in RAM.

8. USB Operatioin Processing Overview AND Related API

8.1 Attach Processing

When a USB device is attached to the USB host, the USB driver calls the *ARM_USBD_SignalDeviceEvent* function (callback function). The macro (*ARM_USBD_EVENT_VBUS_ON*) is set to the argument (*event*) at this time. The application program can detect the attach to the USB host by this callback function.

The application program needs to call *ARM_USBD_DeviceConnect* function to pull up the D+ line after detcting the attach.

When the D+ line is pulled up, USB Host recognizes that USB device was attached and start the USB enumeration after sending the USB Reset signal to USB device.

Related API:

ARM_USBD_SignalDeviceEvent (event: ARM_USBD_EVENT_VBUS_ON)
ARM_USBD_DeviceConnet

8.2 Default State Detection

When the D+ line is pulled up, USB Host sends USB Reset signal. USB module shifts to the default state by receiving the USB Reset signal.

The application program needs to configure the endpoint 0 after detecting the default state. The *ARM_USBD_EndpointConfigure* function is used to configure the endpoint.

Note:

When USB device state shifts to the default state, USB driver calls *ARM_USBD_SignalDeviceEvent* function (Callback function). The macro (*ARM_USBD_EVENT_RESET*) is set to the argument at this time. The application program can detect the default state by this callback function.

Related API:

ARM_USBD_SignalDeviceEvent (event: ARM_USBD_EVENT_RESET)
ARM_USBD_EndpointConfigure

8.3 Enumeration (USB Request) Processing

The application program must perform the enumeration processing with USB Host by the following processing.

8.3.1 Setup Stage Processing (USB Request Reception)

When this driver receives the USB request sent from USB Host, *ARM_USBD_SignalEndpointEvent* function (Callback function) is called. USB request (Setup:8bytes) is gotten by using *ARM_USB_ReadSetupPacket* function. The application program needs to analyze USB request and perform processing corresponding to the USB request.

Related API:

ARM_USBD_SignalEndpointEvent (event: ARM_USBD_EVENT_SETUP)
ARM_USBD_ReadSetupPacket

8.3.2 Data Stage Processing

ARM_USBD_EndpointTransfer function is used to trasfer the data on the date stage.

1. Specify the endpoint 0 to the 1st argument of *ARM_USBD_EndpointTransfer* function.
2. Specify the start address of the area to store the transfer data to 2nd argument of *ARM_USBD_EndpointTransfer* function.
3. Specify the transfer data size to the 3rd argument of *ARM_USBD_EndpointTransfer* function.
4. Call *ARM_USBD_EndpointTransfer* function.

Note:

When completing the above data transer, USB driver calls the callback function (*ARM_USBD_SignalEndpointEvent*).

Related API:

ARM_USBD_EndpointTransfer
ARM_USBD_SignalEndpointEvent(event: ARM_USBD_EVENT_IN)

8.3.3 Status Stage Processing

In the following case, this driver does not process to the status stage. The user need to process the status stage in the application program. The user needs to process the status stage in the application program.

1. Case of responsing ACK to a USB request that supports no-data control status stage

Use *ARM_USBD_EndpointTransfer* function.

Example)

```
ARM_USBD_EndpointTransfer ((USB_EP_IN | USB_EP0), USB_NULL, USB_NULL);
```

a. Specify the endpoint number 0 to the 1st argument.
b. Specify *USB_NULL* to the 2nd argumet and 3rd argument.

2. Case of responsing STALL to a USB request

Use *ARM_USBD_EndpointStall* function. Specify the endpoint number 0 to the 1st argumet and *true* to the 2nd argument.

Example)

```
ARM_USBD_EndpointStall ((USB_EP_IN | USB_EP0), true);
```

Note:

When receiving a USB request that support the data stage, this USB driver process the status stage after processing the data stage.

Related API:

ARM_USBD_EndpointStall
ARM_USBD_EndpointTransfer
ARM_USBD_SignalEndpointEvent(event: ARM_USBD_EVENT_IN)

8.4 Enumeration Completion

USB device state shifts to the configued state when the enumration completes properly, USB data transfer by Bulk/Interrupt transfer can perform. The application program must configure each endpoint to perform USB data transfer after detecting the configured state. The function *ARM_USBD_EndpointConfigue* is used to configure the endpoint.

Note:

When USB module receives *SetConfiguration* request, USB device state shifts to the configured state.The application can judge whether the USB device state shifted to the configured state or not by checking the request. For how to receive USB request, refer to chapter **8.3.1, Setup Stage Processing (USB Request Reception)**.

Related API:

ARM_USBD_SignalEndpointEvent (event: ARM_USBD_EVENT_SETUP)
ARM_USBD_ReadSetupPacket
ARM_USBD_EndpointConfigure

8.5 USB Data Transfer

When USB device state shifts to the configured state and the configuration for each endpoint is completed, the application program can perform the USB data transfer by Bulk transfer or Interrupt transfer. Please use *ARM_USBD_EndpointTransfer* function when performing the data transfer by Bulk transfer or Interupt transfer.

Note:

1. USB driver calls the callback function (*ARM_USBD_SignalEndpointEvent* function) when completing the data transfer. The application program can detect the data transfer completion by this callback function.
2. Please use *ARM_USBD_EndpointTransferGetResult* function when getting the transfer data size.

Related API:

ARM_USBD_EndpointTransfer
ARM_USBD_SignalEndpointEvent (event: ARM_USBD_EVENT_OUT)
ARM_USBD_SignalEndpointEvent (event: ARM_USBD_EVENT_IN)
ARM_USBD_EndpointTransferGetResult

8.6 Suspend

When USB module detects Suspend signal, USB driver calls *ARM_USBD_SignalDeviceEvent* function (Callback function). The macro (*ARM_USBD_EVENT_SUSPEND*) is set to the argument (*event*) of this callback function. The application program can detect by this callback function.

Related API:

ARM_USBD_SignalDeviceEvent (event: ARM_USBD_EVENT_SUSPEND)

8.7 Resume

When USB module detects Resume signal, USB driver calls *ARM_USBD_SignalDeviceEvent* function (Callback function). The macro (*ARM_USBD_EVENT_RESUME*) is set to the argument (*event*) of this callback function. The application program can detect Resume signal by this callback function.

Related API:

ARM_USBD_SignalDeviceEvent (event: ARM_USBD_EVENT_RESUME)

8.8 RemoteWakeup

Please call *ARM_USBD_DeviceRemoteWakeup* function when sending RemoteWakeup signal to USB Host.

Related API:

ARM_USBD_DeviceRemoteWakeup

8.9 Detach Detection

USB driver calls ARM_USBD_SignalDeviceEvent function when detaching USB device from USB Host. The macro (*ARM_USBD_EVENT_VBUS_OFF*) is set to the argument (*event*) of this callback function. The application program can detect the detach by this callback funcion.

Related API:

ARM_USBD_SignalDeviceEvent (event: ARM_USBD_EVENT_VBUS_OFF)
ARM_USBD_EndpointUnconfigure

8.10 API Calling Sequence

The following shows API calling sequence.

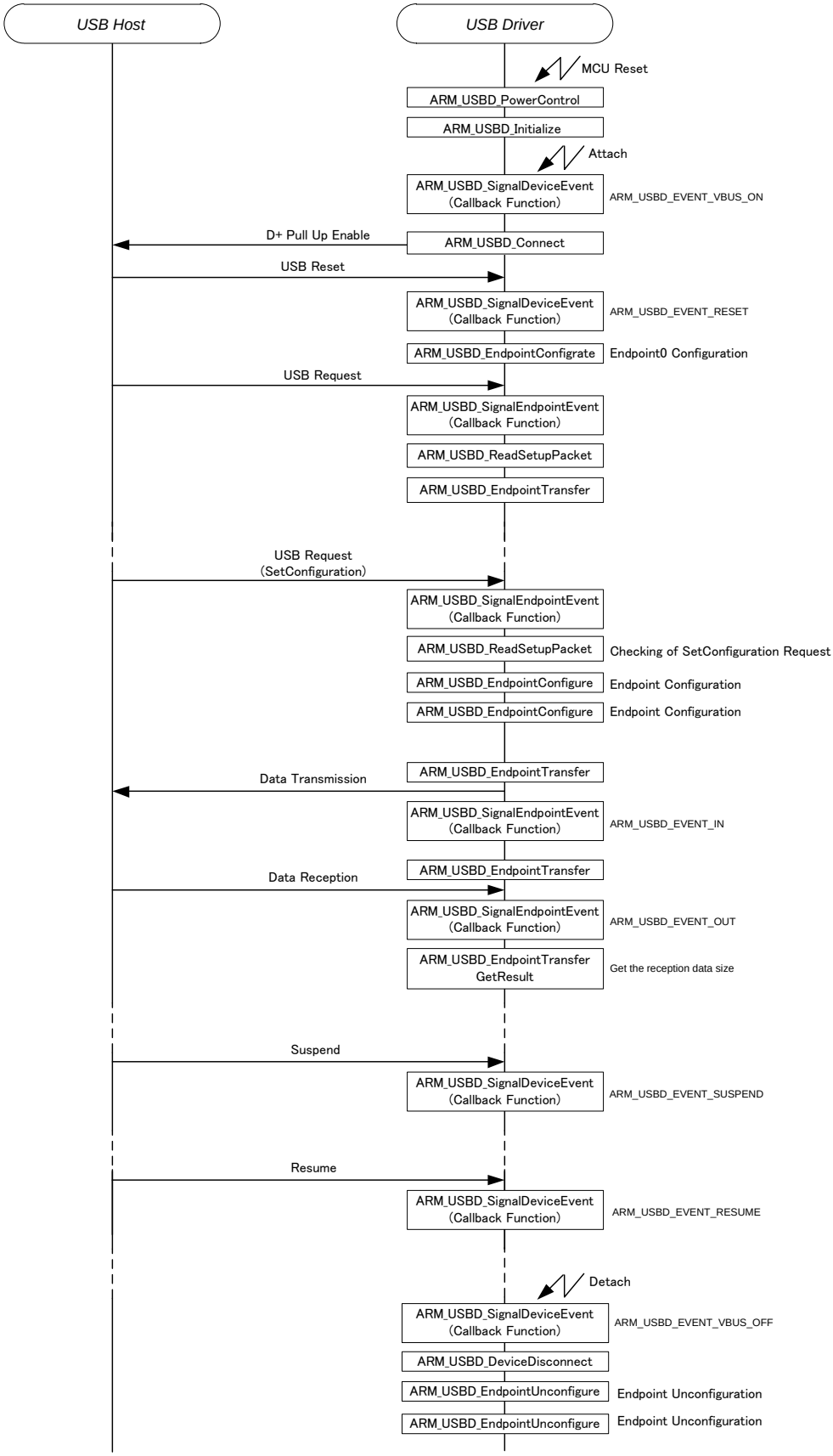

Figure 8-1 API Calling Sequence

9. USB Class Requests

For the class request processisng procedure, refer to chapter **8.3, Enumeration (USB Request) Processing**.

Note:

If an unsupported class request is received, respond STALL to the USB host. For responding STALL, refer to chapter **8.3.3, Status Stage Processing**.

10. Zero-Length Packet

Please use *ARM_USBD_EndpointTransfer* function (API) when sending the zero-length packet to USB Host.

Example) Case of sending the zero-length packet to the endpoint 1.

```
ARM_USBD_EndpointTransfer ((USB_EP_IN | USB_EP1), USB_NULL, USB_NULL);
```

1. Specify the endpoint number for sending the zero-length packet to the 1st argument.
2. Specify *USB_NULL* to the 2nd argument and the 3rd argument.

11. DTC/DMA Transfer

11.1 Basic Specification

The specifications of the DTC/DMA transfer program code used in this driver are listed below.
USB Pipe 1 and Pipe2 can used DTC/DMA access.
Table11-1 shows DTC/DMA Setting Specifications.

Table11-1 DTC/DMA Setting Specifications

Setting	Description
FIFO port used	D0FIFO and D1FIFO port
Transfer mode	Block transfer mode
Chain transfer	Disabled
Address mode	Full address mode
Read skip	Disabled
Access bit width (MBW)	2-byte transfer: 16-bit width
USB transfer type	BULK transfer
Transfer end	Receive direction: BRDY interrupt Transmit direction: D0FIFO/D1FIFO interrupt, BEMP interrupt

Note:

This driver does not support to use DMA transfer and DTC transfer at the same time.

11.2 Notes

11.2.1 Data Reception Buffer Size

The user needs to allocate the area n times the max packet size to store the receiving data.

11.2.2 Initialization Function for DMA/DTC transfer

Call the following DMA/DTC transfer initialization function in the user application program.

Transfer Type	Initialization Function
DTC	R_DTC_Open
DMA	R_DMAC_Open

Note:

Specify the following definition to the argument in *R_DMAC_Open* function. For the defail of the following definition, refer to chapter **7, Configuration (r_usb_basic_mini_cfg.h)**.

USB_CFG_USB0_DMA_TX, USB_CFG_USB0_DMA_RX

Example)

```
R_DMAC_Open(USB_CFG_USB0_DMA_TX);
R_DMAC_Open(USB_CFG_USB0_DMA_RX);
```

12. Additional Notes

12.1 USB Interrupt Registration to NVIC

Please make registrating of the following USB interrupt to NVIC by r_system_cfg.h file.

1. USBI Interrupt
2. USBR Interrupt
3. D0FIFO Interrupt
4. D1FIFO Interrupt

The following is the registraion example.

```
#define    SYSTEM_CFG_EVENT_NUMBER_USBFS_D0FIFO    SYSTEM_IRQ_EVENT_NUMBER8
#define    SYSTEM_CFG_EVENT_NUMBER_USBFS_D1FIFO    SYSTEM_IRQ_EVENT_NUMBER9
#define    SYSTEM_CFG_EVENT_NUMBER_USBFS_USBI      SYSTEM_IRQ_EVENT_NUMBER10
#define    SYSTEM_CFG_EVENT_NUMBER_USBFS_USBR      SYSTEM_IRQ_EVENT_NUMBER11
```

12.2 Clock Setting

When using USB module, 48MHz must be supplied for *UCLK*. In order to supply 48MHz to *UCLK*, please make setting for the following in the *r_core_cfg.h* file.

1.	SYSTEM_CFG_MOSC_ENABLE	Be sure to specify the numeric value 1.
2.	SYSTEM_CFG_LOCO_ENABLE	Be sure to specify the numeric value 0.
3.	SYSTEM_CFG_PLL_ENABLE	Be sure to specify the numeric value 1.
4.	SYSTEM_CFG_PLL_DIV	
5.	SYSTEM_CFG_PLL_MUL	
6.	SYSTEM_CFG_CLOCK_SOURCE	Be sure to specify the numeric value 5.
7.	SYSTEM_CFG_PCKB_DIV	
8.	SYSTEM_CFG_POWER_CONTROL_MODE	Be sure to specify the numeric value 0.

12.3 Vendor ID

Be sure to use the user's own Vendor ID for the one to be provided in the Device Descriptor.

13. Creating an Application Program

This chapter explains how to create an application program using the API functions described throughout this document. Please make sure you use the API functions described here when developing your application program.

13.1 Configuration

Set each configuration file (header file) in the r_config folder to meet the specifications and requirements of your system. Please refer to chapter **7, Configuration** about setting of the configuration file.

13.2 Descriptor Creation

For USB peripheral operations, your will need to create descriptors to meet your system specifications. Register the created descriptors in the *usb_descriptor_t* function members. USB host operations do not require creation of special descriptors.

13.3 Application Program Creation

13.3.1 Include

Make sure you include the following files in your application program.

1. Driver_USBD.h
2. r_usb_basic_if.h
3. r_usb_xxxxx_if.h (I/F file provided for the USB device class to be used)
4. Include any other driver-related header files that are used within the application program.

13.3.2 Initialization

1. MCU pin settings

USB input/output pin settings are necessary to use the USB controller. The following is a list of USB pins that need to be set. Set the following pins as necessary.

Table13-1 USB I/O Pin Settings

Pin Name	I/O	Function
USB_VBUS	input	VBUS pin for USB communication

Note:

(1). Please refer to the corresponding MCU user's manual for the pin settings in ports used for your application program.

(2). The function *R_USB_Pinset()/R_USB_Pinclr()* for USB pin setting is prepared.

2. DTC/DMA-related initialization

Call the DTC/DMA initialization fucntion when using the DTC/DMA transfer.

Tranfer	Function
DTC	R_DTC_Open
DMA	R_DMAC_Open

Note:

The setting for DTC/DMA transfer is needed when using DTC/DMA transfer. Refer to chpater 7, Configuration (r_usb_basic_mini_cfg.h).

Initialization Example)

(1). DTC

```
e_dma_err_t     ret;

ret = Driver_DTC.Open();
if (DMA_OK != ret)
{
```

```
        /* do something */
    }
```

(2). DMA

```
    ret = Driver_DMAC0.Open();
    if (DMA_OK != ret)
    {
        /* do something */
    }

    ret = Driver_DMAC1.Open();
    if (DMA_OK != ret)
    {
        /* do something */
    }
```

3. USB-related initialization

Call the *ARM_USBD_Initialize* function to initialize the USB module (hardware), USB driver software and the registration of the callback function (*ARM_USBD_SignalDeviceEvent* fucntion and *ARM_USBD_ SignalEndpointEvent* function) used for your application program.

13.3.3 Descriptor Creation

Please create descriptors to meet your system specifications. Refer to chapter **2.4, Descriptor** for more details about descriptors.

13.3.4 Creation of Main Routine

Please describe the main routine in the main loop format. Make sure you call the *R_USB_ControlDriver* function in the main loop. The USB completed events are set to the argument of *ARM_USBD_SignalDeviceEvent* function and *ARM_USBD_SignalEndpointEvent* function (Callback function). Please descibe the application program the correspoing the USB completed event.

Note:

Carry out USB data communication using the *ARM_USBD_EndpointTransfer* function after checking of the detection of shifting to USB configured state.

Website and Support

Renesas Electronics Website

http://www.renesas.com/

Inquiries

http://www.renesas.com/inquiry

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep 30, 2019	—	First edition issued

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.