
USB Stack OTG Reference Manual

NXP Semiconductors

Document Number: MCUXSDKUSBOAPIRM
Rev. 0
Mar 2017



Contents

Chapter Definitions and structures

1.1	Overview	1
1.2	Data Structure Documentation	2
1.2.1	struct usb_version_t	2
1.3	Typedef Documentation	3
1.3.1	usb_device_handle	3
1.4	Enumeration Type Documentation	3
1.4.1	usb_status_t	3
1.4.2	usb_controller_index_t	3

Chapter USB OTG driver

2.1	Overview	5
2.2	Data Structure Documentation	7
2.2.1	struct usb_otg_descriptor_t	7
2.2.2	struct usb_otg_instance_t	8
2.3	Typedef Documentation	8
2.3.1	usb_otg_callback_t	8
2.4	Enumeration Type Documentation	9
2.4.1	usb_otg_status_type_t	9
2.4.2	usb_otg_device_state_t	9
2.4.3	usb_otg_stack_init_type_t	10
2.4.4	usb_otg_event_type_t	10
2.5	Function Documentation	10
2.5.1	USB_OtgInit	10
2.5.2	USB_OtgDeinit	11
2.5.3	USB_OtgTaskFunction	11
2.5.4	USB_OtgKhciIsrFunction	12
2.5.5	USB_OtgBusDrop	12
2.5.6	USB_OtgBusRequest	12
2.5.7	USB_OtgBusRelease	13

Contents

Section Number	Title	Page Number
2.5.8	USB_OtgClearError	13
2.5.9	USB_OtgNotifyChange	14
2.6	USB OTG Controller driver	15
2.6.1	Overview	15
2.6.2	Data Structure Documentation	16
2.6.3	Macro Definition Documentation	16
2.6.4	Enumeration Type Documentation	16
2.7	USB OTG Peripheral driver	18
2.7.1	Overview	18
2.7.2	Function Documentation	18
Chapter	USB OS Adapter	
3.1	Overview	21
3.2	Enumeration Type Documentation	23
3.2.1	usb_osa_status_t	23
3.2.2	usb_osa_event_mode_t	23
3.3	Function Documentation	23
3.3.1	USB_OsaMemoryAllocate	23
3.3.2	USB_OsaMemoryFree	23
3.3.3	USB_OsaEventCreate	24
3.3.4	USB_OsaEventDestroy	24
3.3.5	USB_OsaEventSet	25
3.3.6	USB_OsaEventWait	26
3.3.7	USB_OsaEventCheck	27
3.3.8	USB_OsaEventClear	28
3.3.9	USB_OsaSemCreate	28
3.3.10	USB_OsaSemDestroy	29
3.3.11	USB_OsaSemPost	29
3.3.12	USB_OsaSemWait	30
3.3.13	USB_OsaMutexCreate	30
3.3.14	USB_OsaMutexDestroy	31
3.3.15	USB_OsaMutexLock	31
3.3.16	USB_OsaMutexUnlock	32
3.3.17	USB_OsaMsgqCreate	32
3.3.18	USB_OsaMsgqDestroy	33
3.3.19	USB_OsaMsgqSend	33
3.3.20	USB_OsaMsgqRecv	34
3.3.21	USB_OsaMsgqCheck	34

Chapter 1

Definitions and structures

1.1 Overview

This lists the common definitions and structures for USB stack.

Data Structures

- struct [usb_version_t](#)
USB stack version fields. [More...](#)

Macros

- #define [USB_STACK_VERSION_MAJOR](#) (0x01U)
Defines USB stack major version.
- #define [USB_STACK_VERSION_MINOR](#) (0x00U)
Defines USB stack minor version.
- #define [USB_STACK_VERSION_BUGFIX](#) (0x00U)
Defines USB stack bugfix version.
- #define [USB_MAKE_VERSION](#)(major, minor, bugfix) (((major) << 16) | ((minor) << 8) | (bugfix))
USB stack version definition.

Typedefs

- typedef void * [usb_host_handle](#)
USB host handle type define.
- typedef void * [usb_device_handle](#)
USB device handle type define.
- typedef void * [usb_otg_handle](#)
USB OTG handle type define.

Enumerations

- enum `usb_status_t` {
 `kStatus_USB_Success` = 0x00U,
 `kStatus_USB_Error`,
 `kStatus_USB_Busy`,
 `kStatus_USB_InvalidHandle`,
 `kStatus_USB_InvalidParameter`,
 `kStatus_USB_InvalidRequest`,
 `kStatus_USB_ControllerNotFound`,
 `kStatus_USB_InvalidControllerInterface`,
 `kStatus_USB_NotSupported`,
 `kStatus_USB_Retry`,
 `kStatus_USB_TransferStall`,
 `kStatus_USB_TransferFailed`,
 `kStatus_USB_AllocFail`,
 `kStatus_USB_LackSwapBuffer`,
 `kStatus_USB_TransferCancel`,
 `kStatus_USB_BandwidthFail`,
 `kStatus_USB_MSDDStatusFail` }

USB error code.

- enum `usb_controller_index_t` {
 `kUSB_ControllerKhci0` = 0U,
 `kUSB_ControllerKhci1` = 1U,
 `kUSB_ControllerEhci0` = 2U,
 `kUSB_ControllerEhci1` = 3U,
 `kUSB_ControllerLpcIp3511Fs0` = 4U,
 `kUSB_ControllerLpcIp3511Fs1`,
 `kUSB_ControllerLpcIp3511Hs0` = 6U,
 `kUSB_ControllerLpcIp3511Hs1`,
 `kUSB_ControllerOhci0` = 8U,
 `kUSB_ControllerOhci1` = 9U,
 `kUSB_ControllerIp3516Hs0` = 10U,
 `kUSB_ControllerIp3516Hs1` }

USB controller ID.

1.2 Data Structure Documentation

1.2.1 struct `usb_version_t`

Data Fields

- uint8_t `major`
Major.
- uint8_t `minor`
Minor.

- uint8_t [bugfix](#)
Bug fix.

1.3 Typedef Documentation

1.3.1 typedef void* usb_device_handle

For device stack it is the whole device handle; for host stack it is the attached device instance handle

1.4 Enumeration Type Documentation

1.4.1 enum usb_status_t

Enumerator

kStatus_USB_Success Success.
kStatus_USB_Error Failed.
kStatus_USB_Busy Busy.
kStatus_USB_InvalidHandle Invalid handle.
kStatus_USB_InvalidParameter Invalid parameter.
kStatus_USB_InvalidRequest Invalid request.
kStatus_USB_ControllerNotFound Controller cannot be found.
kStatus_USB_InvalidControllerInterface Invalid controller interface.
kStatus_USB_NotSupported Configuration is not supported.
kStatus_USB_Retry Enumeration get configuration retry.
kStatus_USB_TransferStall Transfer stalled.
kStatus_USB_TransferFailed Transfer failed.
kStatus_USB_AllocFail Allocation failed.
kStatus_USB_LackSwapBuffer Insufficient swap buffer for KHCI.
kStatus_USB_TransferCancel The transfer cancelled.
kStatus_USB_BandwidthFail Allocate bandwidth failed.
kStatus_USB_MSDStatusFail For MSD, the CSW status means fail.

1.4.2 enum usb_controller_index_t

Enumerator

kUSB_ControllerKhci0 KHCI 0U.
kUSB_ControllerKhci1 KHCI 1U, Currently, there are no platforms which have two KHCI IPs, this is reserved to be used in the future.
kUSB_ControllerEhci0 EHCI 0U.
kUSB_ControllerEhci1 EHCI 1U, Currently, there are no platforms which have two EHCI IPs, this is reserved to be used in the future.
kUSB_ControllerLpcIp3511Fs0 LPC USB IP3511 FS controller 0.

Enumeration Type Documentation

kUSB_ControllerLpcIp3511Fs1 LPC USB IP3511 FS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

kUSB_ControllerLpcIp3511Hs0 LPC USB IP3511 HS controller 0.

kUSB_ControllerLpcIp3511Hs1 LPC USB IP3511 HS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

kUSB_ControllerOhci0 OHCI 0U.

kUSB_ControllerOhci1 OHCI 1U, Currently, there are no platforms which have two OHCI IPs, this is reserved to be used in the future.

kUSB_ControllerIp3516Hs0 IP3516HS 0U.

kUSB_ControllerIp3516Hs1 IP3516HS 1U, Currently, there are no platforms which have two IP3516HS IPs, this is reserved to be used in the future.

Chapter 2

USB OTG driver

2.1 Overview

Modules

- [USB OTG Controller driver](#)
- [USB OTG Peripheral driver](#)

Data Structures

- struct [usb_otg_descriptor_t](#)
usb otg descriptor [More...](#)
- struct [usb_otg_instance_t](#)
USB OTG instance structure. [More...](#)

Macros

- #define [USB_OTG_MSG_COUNT](#) (8)
USB otg task message queue count.
- #define [USB_OTG_STATUS_HOST_REQUEST_FLAG](#) (0x01U)
USB otg host request flag.

Typedefs

- typedef void * [usb_otg_controller_handle](#)
USB otg controller handle type define.
- typedef void(* [usb_otg_callback_t](#))(void *param, uint8_t eventType, uint32_t eventValue)
OTG callback function typedef.

Enumerations

- enum `usb_otg_status_type_t` { ,
 `kOtg_StatusAdpChange` = 0x0002U,
 `kOtg_StatusSrpDet` = 0x0004U,
 `kOtg_StatusVbusVld` = 0x0008U,
 `kOtg_StatusAConn` = 0x0010U,
 `kOtg_StatusBusResume` = 0x0020U,
 `kOtg_StatusBusSuspend` = 0x0040U,
 `kOtg_StatusSe0Srp` = 0x0080U,
 `kOtg_StatusSsendSrp` = 0x0100U,
 `kOtg_StatusSessVld` = 0x0200U,
 `kOtg_StatusBusDrop` = 0x0400U,
 `kOtg_StatusBusReq` = 0x0800U,
 `kOtg_StatusPowerUp` = 0x1000U,
 `kOtg_StatusTimeOut` = 0x2000U,
 `kOtg_StatusBConn` = 0x4000U,
 `kOtg_StatusClrErr` = 0x8000U,
 `kOtg_StatusBSrpDone` = 0x10000U,
 `kOtg_StatusADisconn` = 0x20000U,
 `kOtg_StatusBDisconn` = 0x40000U,
 `kOtg_StatusVbusInvld` = 0x80000U,
 `kOtg_StatusSessInvld` = 0x100000U,
 `kOtg_StatusCheckIdleInAPeripheral` = 0x200000U,
 `kOtg_StatusBHNPFeature` = 0x40000000U,
 `kOtg_StatusChange` = 0x80000000U }
 please reference to 7.4 in otg spec
- enum `usb_otg_device_state_t` { ,
 `kOtg_State_AIdle`,
 `kOtg_State_AWaitVrise`,
 `kOtg_State_AWaitBcon`,
 `kOtg_State_AHost`,
 `kOtg_State_AWaitVfall`,
 `kOtg_State_ASuspend`,
 `kOtg_State_APeripheral`,
 `kOtg_State_AVbusErr`,
 `kOtg_State_BIdleEh`,
 `kOtg_State_BIdle`,
 `kOtg_State_BSrpInit`,
 `kOtg_State_BPeripheral`,
 `kOtg_State_BWaitAcon`,
 `kOtg_State_BHost` }
 Please reference to chapter 7 in otg spec.
- enum `usb_otg_stack_init_type_t` { ,

```

kOtg_StackHostInit,
kOtg_StackHostDeinit,
kOtg_StackDeviceInit,
kOtg_StackDeviceDeinit }

```

The event value for callback to application when event type is kOtg_EventStackInit.

- enum `usb_otg_event_type_t` {
`kOtg_EventStateChange` = 0U,
`kOtg_EventStackInit` }

The event types for callback to application.

USB OTG APIs

- `usb_status_t` `USB_OtgInit` (`uint8_t` controllerId, `usb_otg_handle` *otgHandle, `usb_otg_callback_t` otgCallbackFn, void *callbackParameter)
Initializes the USB OTG stack.
- `usb_status_t` `USB_OtgDeinit` (`usb_otg_handle` otgHandle)
Deinitializes the USB otg stack.
- void `USB_OtgTaskFunction` (`usb_otg_handle` otgHandle)
OTG stack task function.
- void `USB_OtgKhciIsrFunction` (`usb_otg_handle` otgHandle)
OTG KHCI ISR function.
- `usb_status_t` `USB_OtgBusDrop` (`usb_otg_handle` otgHandle, `uint8_t` drop)
A-device drop bus.
- `usb_status_t` `USB_OtgBusRequest` (`usb_otg_handle` otgHandle)
bus request.
- `usb_status_t` `USB_OtgBusRelease` (`usb_otg_handle` otgHandle)
bus request.
- `usb_status_t` `USB_OtgClearError` (`usb_otg_handle` otgHandle)
clear error.
- `usb_status_t` `USB_OtgNotifyChange` (`usb_otg_handle` otgHandle, `uint32_t` statusType, `uint32_t` statusValue)
Notify otg stack the status changes.

2.2 Data Structure Documentation

2.2.1 struct `usb_otg_descriptor_t`

Data Fields

- `uint8_t` `bLength`
Size of Descriptor.
- `uint8_t` `bDescriptorType`
OTG type = 9.
- `uint8_t` `bmAttributes`
Attribute Fields.
- `uint8_t` `bcdOTG` [2]
OTG and EH supplement release number in binary-coded decimal.

Typedef Documentation

2.2.1.0.0.1 Field Documentation

2.2.1.0.0.1.1 uint8_t usb_otg_descriptor_t::bmAttributes

D7..3: Reserved (reset to zero) D2: ADP support D1: HNP support D0: SRP support

2.2.2 struct usb_otg_instance_t

Data Fields

- [usb_otg_controller_handle](#) controllerHandle
The low level controller handle.
- [usb_otg_callback_t](#) otgCallback
OTG callback function.
- void * [otgCallbackParameter](#)
OTG callback function parameter.
- [usb_osa_msgq_handle](#) otgMsgHandle
OTG task message queue handle.
- const
usb_otg_controller_interface_t * [controllerInterface](#)
controller interface APIs
- uint32_t [otgControllerStatus](#)
please reference to [usb_otg_status_type_t](#)
- uint8_t [otgDeviceState](#)
please reference to [usb_otg_device_state_t](#)
- volatile uint8_t [hasTimeOutMsg](#)
There is timer out message in the message queue.
- volatile uint8_t [hasUpdateMsg](#)
There is update message in the message queue.
- uint8_t [cancelTime](#)
Don't process the timer out message.
- uint8_t [waitInit](#)
Waiting the opposite side board's device stack or host stack initializing.

2.3 Typedef Documentation

2.3.1 typedef void(* usb_otg_callback_t)(void *param, uint8_t eventType, uint32_t eventValue)

This callback function is used to notify application events, the events include [usb_otg_event_type_t](#). This callback pointer is passed when initializing OTG.

Parameters

<i>param</i>	The assigned parameter when initializing OTG.
<i>eventType</i>	Please reference to usb_otg_event_type_t .
<i>event_code</i>	Please reference to usb_otg_device_state_t and usb_otg_stack_init_type_t .

2.4 Enumeration Type Documentation

2.4.1 enum usb_otg_status_type_t

Enumerator

kOtg_StatusAdpChange id
kOtg_StatusSrpDet adp_change
kOtg_StatusVbusVld a_srp_det
kOtg_StatusAConn a_vbus_vld
kOtg_StatusBusResume a_conn
kOtg_StatusBusSuspend a_bus_resume
kOtg_StatusSe0Srp a_bus_suspend
kOtg_StatusSsendSrp b_se0_srp
kOtg_StatusSessVld b_ssend_srp
kOtg_StatusBusDrop b_sess_vld
kOtg_StatusBusReq a_bus_drop
kOtg_StatusPowerUp a_bus_req and b_bus_req
kOtg_StatusTimeOut power_up
kOtg_StatusBConn all the timeout in the state machine
kOtg_StatusClrErr b_conn
kOtg_StatusBSrpDone a_clr_err
kOtg_StatusADisconn b_srp_done
kOtg_StatusBDisconn a_conn(non)
kOtg_StatusVbusInvld b_conn(non)
kOtg_StatusSessInvld a_vbus_vld(non)
kOtg_StatusCheckIdleInAPeripheral b_sess_vld(non)
kOtg_StatusBHNPFeature check the idle timeout when in a_peripheral state
kOtg_StatusChange This status is valid when (1) b_hnp_enable feature is sent when A-device works as host; Or (2) b_hnp_enable feature is received when B-device works as device.

2.4.2 enum usb_otg_device_state_t

Enumerator

kOtg_State_AIdle state state
kOtg_State_AWaitVrise a_idle state

Function Documentation

kOtg_State_AWaitBcon a_wait_vrise state
kOtg_State_AHost a_wait_bcon state
kOtg_State_AWaitVfall a_host state
kOtg_State_ASuspend a_wait_vfall state
kOtg_State_APeripheral a_suspend state
kOtg_State_AVbusErr a_peripheral state
kOtg_State_BIdleEh a_vbus_err state
kOtg_State_BIdle b_idle_eh state
kOtg_State_BSrpInit b_idle or bp_idle state, when the device is peripheral-only B-device it means bp_idle
kOtg_State_BPeripheral b_srp_init or bp_srp_init state, when the device is peripheral-only B-device it means bp_srp_init
kOtg_State_BWaitAcon b_peripheral or bp_peripheral state, when the device is peripheral-only B-device it means bp_peripheral
kOtg_State_BHost b_wait_acon state

2.4.3 enum usb_otg_stack_init_type_t

Enumerator

kOtg_StackHostInit default state
kOtg_StackHostDeinit notify application to initialize host stack
kOtg_StackDeviceInit notify application to de-initialize host stack
kOtg_StackDeviceDeinit notify application to initialize device stack

2.4.4 enum usb_otg_event_type_t

Enumerator

kOtg_EventStateChange otg state change event, the event values are [usb_otg_device_state_t](#)
kOtg_EventStackInit host/device stack handle event, the event values are [usb_otg_stack_init_type_t](#)

2.5 Function Documentation

2.5.1 usb_status_t USB_OtgInit (uint8_t controllerId, usb_otg_handle * otgHandle, usb_otg_callback_t otgCallbackFn, void * callbackParameter)

This function initializes the USB otg module specified by the controllerId.

Parameters

in	<i>controllerId</i>	The controller ID of the USB IP. See the enumeration usb_controller_index_t .
out	<i>otgHandle</i>	Return the otg handle.
in	<i>otgCallbackFn</i>	Otg callback function, it is usb_otg_callback_t .
in	<i>callback-Parameter</i>	The callback parameter.

Return values

<i>kStatus_USB_Success</i>	The otg is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The otgHandle is a NULL pointer.
<i>kStatus_USB_AllocFail</i>	Allocation memory fail.
<i>kStatus_USB_Error</i>	message queue create fail, controller is not found, controller initialize fail.

2.5.2 **usb_status_t USB_OtgDeinit (usb_otg_handle *otgHandle*)**

This function deinitializes the USB otg module specified by the otgHandle.

Parameters

in	<i>otgHandle</i>	the otg handle.
----	------------------	-----------------

Return values

<i>kStatus_USB_Success</i>	The OTG is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The otgHandle is a NULL pointer.
<i>kStatus_USB_Error</i>	Controller deinitialization fail.

2.5.3 **void USB_OtgTaskFunction (usb_otg_handle *otgHandle*)**

The function implement the OTG stack state machine. In bare metal environment, this function should be called periodically in the main function. In the RTOS environment, this function should be used as a function entry to create a task.

Function Documentation

Parameters

in	<i>otgHandle</i>	The otg handle.
----	------------------	-----------------

2.5.4 void USB_OtgKhcIsrFunction (usb_otg_handle *otgHandle*)

The function is the KHCI interrupt service routine.

Parameters

in	<i>otgHandle</i>	The otg handle.
----	------------------	-----------------

2.5.5 usb_status_t USB_OtgBusDrop (usb_otg_handle *otgHandle*, uint8_t *drop*)

This function drop the bus.

Parameters

in	<i>otgHandle</i>	the otg handle.
in	<i>drop</i>	1 or 0.

Return values

<i>kStatus_USB_Success</i>	Success.
<i>kStatus_USB_Invalid-Handle</i>	The otgHandle is a NULL pointer.
<i>kStatus_USB_Error</i>	The device is not A-device or Send message error.

2.5.6 usb_status_t USB_OtgBusRequest (usb_otg_handle *otgHandle*)

This function can be called in the follow situations:

1. A-device request bus, change from a_idle to a_wait_vrise.
2. HNP, B-device is in the b_peripheral and request the bus.
3. A-device is in the a_peripheral and request the bus.
4. B-device request bus (SRP), change from b_idle to b_srp_init
5. Poll device status, "host request flag" is set.

Parameters

in	<i>otgHandle</i>	the otg handle.
----	------------------	-----------------

Return values

<i>kStatus_USB_Success</i>	Success.
<i>kStatus_USB_Invalid-Handle</i>	The otgHandle is a NULL pointer.
<i>kStatus_USB_Error</i>	Send message error.

2.5.7 usb_status_t USB_OtgBusRelease (usb_otg_handle *otgHandle*)

This function can be called in the follow situations:

1. A-device set the bus request false when in a_idle.
2. A-device release bus when A-device is host (a_host).
3. B-device release bus when B-device is host (b_host).

Parameters

in	<i>otgHandle</i>	the otg handle.
----	------------------	-----------------

Return values

<i>kStatus_USB_Success</i>	Success.
<i>kStatus_USB_Invalid-Handle</i>	The otgHandle is a NULL pointer.
<i>kStatus_USB_Error</i>	Send message error.

2.5.8 usb_status_t USB_OtgClearError (usb_otg_handle *otgHandle*)

This function clear the error.

Parameters

in	<i>otgHandle</i>	the otg handle.
----	------------------	-----------------

Function Documentation

Return values

<i>kStatus_USB_Success</i>	Success.
<i>kStatus_USB_Invalid-Handle</i>	The otgHandle is a NULL pointer.
<i>kStatus_USB_Error</i>	The device is not in error state or send message error.

2.5.9 **usb_status_t USB_OtgNotifyChange (usb_otg_handle *otgHandle*, uint32_t *statusType*, uint32_t *statusValue*)**

This function notify the [usb_otg_status_type_t](#) and values.

Parameters

in	<i>otgHandle</i>	the otg handle.
in	<i>statusType</i>	please reference to usb_otg_status_type_t
in	<i>statusValue</i>	the value is 1 or 0

Return values

<i>kStatus_USB_Success</i>	Success.
<i>kStatus_USB_Invalid-Handle</i>	The otgHandle is a NULL pointer.
<i>kStatus_USB_Error</i>	Send message error.

2.6 USB OTG Controller driver

2.6.1 Overview

Data Structures

- struct `usb_otg_msg_t`
OTG stack task message. [More...](#)

Macros

- #define `USB_OTG_TIMER_A_WAIT_VRISE_TMR` (100U)
a_wait_vrise_tmr in OTG spec, VBUS Rise Time, 100ms
- #define `USB_OTG_TIMER_A_WAIT_VFALL_TMR` (1000U)
a_wait_vfall_tmr in OTG spec, Session end to VOTG_VBUS_LKG, 1sec
- #define `USB_OTG_TIMER_A_WAIT_BCON_TMR` (2000U)
a_wait_bcon_tmr in OTG spec, Wait for B-Connect, 1.1sec ~ 30^15sec
- #define `USB_OTG_TIMER_A_AIDL_BDIS_TMR` (500)
a_aidl_bdis_tmr in OTG spec, A-Idle to B-Disconnect, 200ms ~ infinity
- #define `USB_OTG_TIMER_B_ASE0_BRST_TMR` (155)
b_ase0_brst_tmr in OTG spec, A-SE0 to B-Reset, 155ms ~ 200ms
- #define `USB_OTG_TIME_B_DATA_PLS` (7)
TB_DATA_PLS in OTG spec, Data-Line Pulse Time, 5ms ~ 10ms.
- #define `USB_OTG_TIME_B_DATA_PLS_MIN` (5)
TB_DATA_PLS in OTG spec, Data-Line Pulse Time's minimum value.
- #define `USB_OTG_TIME_B_DATA_PLS_MAX` (10)
TB_DATA_PLS in OTG spec, Data-Line Pulse Time's maximum value.
- #define `USB_OTG_TIME_A_BCON_LDB` (100)
TA_BCON_LDB in OTG spec, B-Connect Long Debounce, 100ms ~ infinity.
- #define `USB_OTG_TIME_A_BCON_SDB` (1)
TA_BCON_SDB in OTG spec, B-Connect Short Debounce, 2.5us ~ infinity.
- #define `USB_OTG_TIME_B_SSEND_SRP` (1500)
TB_SSEND_SRP in OTG spec, Session end to SRP init, 1.5sec ~ infinity.
- #define `USB_OTG_TIME_B_SE0_SRP` (1000)
TB_SE0_SRP in OTG spec, SE0 Time Before SRP, 1sec ~ infinity.
- #define `USB_OTG_TIME_B_AIDL_BDIS` (100)
TB_AIDL_BDIS in OTG spec, A-Idle to B-Disconnect, 4ms ~ 150ms.
- #define `USB_OTG_TIME_A_BIDL_ADIS` (190)
TA_BIDL_ADIS in OTG spec, B-Idle to A-Disconnect, Used by an A-device to determine when the B-device has finished being host, 155ms ~ 200ms.
- #define `USB_OTG_TIME_WAIT_DEVICE_INIT` (200U)
wait another device initialize device stack before initializing the host stack
- #define `USB_OTG_TIME_WAIT_BHOST` (1000U)
delay this time before check idle in a_peripheral state, wait another device initialize host stack

Enumerations

- enum `usb_otg_control_t` { ,
 `kOtg_ControlPullUp`,
 `kOtg_ControlPullDown`,
 `kOtg_ControlResume`,
 `kOtg_ControlAdpPrb`,
 `kOtg_ControlDataPulse`,
 `kOtg_ControlHNPCheckEnable`,
 `kOtg_ControlSetTimer`,
 `kOtg_ControlCancelTimer`,
 `kOtg_ControlRequestStatus`,
 `kOtg_ControlUpdateStatus` }
 The control types.
- enum `usb_otg_pull_control_t` {
 `kOtg_PullDp` = 0x01U,
 `kOtg_PullDm` = 0x02U }
 Pull up/down parameters.

2.6.2 Data Structure Documentation

2.6.2.1 struct `usb_otg_msg_t`

Data Fields

- uint32_t `otgStatusType`
 The status types please reference to [usb_otg_status_type_t](#).
- uint32_t `otgStatusValue`
 The status values.

2.6.3 Macro Definition Documentation

2.6.3.1 #define `USB_OTG_TIME_B_DATA_PLS` (7)

generate the data pulse using this time value.

2.6.4 Enumeration Type Documentation

2.6.4.1 enum `usb_otg_control_t`

Enumerator

`kOtg_ControlPullUp` control vbus

kOtg_ControlPullDown pull dp/dm up
kOtg_ControlResume pull dp/dm down
kOtg_ControlAdpPrb do resume
kOtg_ControlDataPulse probe adp
kOtg_ControlHNPCheckEnable generate data pulse
kOtg_ControlSetTimer start to check HNP
kOtg_ControlCancelTimer start timer
kOtg_ControlRequestStatus cancel timer
kOtg_ControlUpdateStatus request the status values [usb_otg_status_type_t](#)

2.6.4.2 enum usb_otg_pull_control_t

Enumerator

kOtg_PullDp pull DP line
kOtg_PullDm pull DM line

2.7 USB OTG Peripheral driver

2.7.1 Overview

Functions

- [usb_status_t USB_OtgPeripheralEnable](#) (void)
Enable OTG peripheral.
- [usb_status_t USB_OtgPeripheralDisable](#) (void)
Disable OTG peripheral.
- [usb_status_t USB_OtgPeripheralGetStatus](#) (uint32_t statusType, uint32_t *statusValue)
Get the peripheral status.
- [usb_status_t USB_OtgPeripheralControl](#) ([usb_otg_controller_handle](#) controllerHandle, uint32_t controlType, uint32_t controlValue1, uint32_t controlValue2)
Control the peripheral.

2.7.2 Function Documentation

2.7.2.1 [usb_status_t USB_OtgPeripheralEnable](#) (void)

This function enable OTG peripheral function.

Return values

<i>kStatus_USB_Success</i>	success.
<i>other</i>	values Fail.

2.7.2.2 [usb_status_t USB_OtgPeripheralDisable](#) (void)

This function disable OTG peripheral function.

Return values

<i>kStatus_USB_Success</i>	success.
<i>other</i>	values Fail.

2.7.2.3 [usb_status_t USB_OtgPeripheralGetStatus](#) (uint32_t *statusType*, uint32_t **statusValue*)

This function is nonblocking, return the result immediately.

Parameters

in	<i>statusType</i>	Please reference to usb_otg_status_type_t .
out	<i>statusValue</i>	The status value.

Return values

<i>kStatus_USB_Success</i>	success.
<i>other</i>	values Fail.

2.7.2.4 **usb_status_t USB_OtgPeripheralControl (usb_otg_controller_handle controllerHandle, uint32_t controlType, uint32_t controlValue1, uint32_t controlValue2)**

This function control the peripheral to implement the different functions.

Parameters

<i>controller-Handle</i>	The controller instance handle.
<i>controlType</i>	The control type, please reference to usb_otg_control_t .
<i>controlValue1</i>	The control value, it is 0 or 1 usually.
<i>controlValue2</i>	It only be used in the kOtg_ControlRequestStatus control now.

Return values

<i>kStatus_USB_Success</i>	success.
<i>other</i>	values Fail.

Chapter 3

USB OS Adapter

3.1 Overview

The OS adapter (OSA) is used to wrap the differences between RTOSes and make the USB stack with the same code base and behavior.

Note

OSA should not be used in the USB application. Therefore, from the USB application viewpoint, OSA is invisible.

Macros

- #define **USB_BIG_ENDIAN** (0U)
Define big endian.
- #define **USB_LITTLE_ENDIAN** (1U)
Define little endian.
- #define **ENDIANNESS USB_LITTLE_ENDIAN**
Define current endian.

Typedefs

- typedef void * **usb_osa_event_handle**
Define USB OSA event handle.
- typedef void * **usb_osa_sem_handle**
Define USB OSA semaphore handle.
- typedef void * **usb_osa_mutex_handle**
Define USB OSA mutex handle.
- typedef void * **usb_osa_msgq_handle**
Define USB OSA message queue handle.

Enumerations

- enum **usb_osa_status_t** {
 kStatus_USB_OSA_Success = 0x00U,
 kStatus_USB_OSA_Error,
 kStatus_USB_OSA_TimeOut }
USB OSA error code.
- enum **usb_osa_event_mode_t** {
 kUSB_OsaEventManualClear = 0U,
 kUSB_OsaEventAutoClear = 1U }
The event flags are cleared automatically or manually.

USB OSA Memory Management

- void * [USB_OsaMemoryAllocate](#) (uint32_t length)
Reserves the requested amount of memory in bytes.
- void [USB_OsaMemoryFree](#) (void *p)
Frees the memory previously reserved.

USB OSA Event

- [usb_osa_status_t USB_OsaEventCreate](#) ([usb_osa_event_handle](#) *handle, uint32_t flag)
Creates an event object with all flags cleared.
- [usb_osa_status_t USB_OsaEventDestroy](#) ([usb_osa_event_handle](#) handle)
Destroys a created event object.
- [usb_osa_status_t USB_OsaEventSet](#) ([usb_osa_event_handle](#) handle, uint32_t bitMask)
Sets an event flag.
- [usb_osa_status_t USB_OsaEventWait](#) ([usb_osa_event_handle](#) handle, uint32_t bitMask, uint32_t flag, uint32_t timeout, uint32_t *bitSet)
Waits for an event flag.
- [usb_osa_status_t USB_OsaEventCheck](#) ([usb_osa_event_handle](#) handle, uint32_t bitMask, uint32_t *bitSet)
Checks an event flag.
- [usb_osa_status_t USB_OsaEventClear](#) ([usb_osa_event_handle](#) handle, uint32_t bitMask)
Clears an event flag.

USB OSA Semaphore

- [usb_osa_status_t USB_OsaSemCreate](#) ([usb_osa_sem_handle](#) *handle, uint32_t count)
Creates a semaphore with a given value.
- [usb_osa_status_t USB_OsaSemDestroy](#) ([usb_osa_sem_handle](#) handle)
Destroys a semaphore object.
- [usb_osa_status_t USB_OsaSemPost](#) ([usb_osa_sem_handle](#) handle)
Posts a semaphore.
- [usb_osa_status_t USB_OsaSemWait](#) ([usb_osa_sem_handle](#) handle, uint32_t timeout)
Waits on a semaphore.

USB OSA Mutex

- [usb_osa_status_t USB_OsaMutexCreate](#) ([usb_osa_mutex_handle](#) *handle)
Creates a mutex.
- [usb_osa_status_t USB_OsaMutexDestroy](#) ([usb_osa_mutex_handle](#) handle)
Destroys a mutex.
- [usb_osa_status_t USB_OsaMutexLock](#) ([usb_osa_mutex_handle](#) handle)
Waits for a mutex and locks it.
- [usb_osa_status_t USB_OsaMutexUnlock](#) ([usb_osa_mutex_handle](#) handle)
Unlocks a mutex.

USB OSA Message Queue

- [usb_osa_status_t USB_OsaMsgqCreate](#) ([usb_osa_msgq_handle](#) *handle, uint32_t count, uint32_t size)

- Creates a message queue.*
- `usb_osa_status_t USB_OsaMsgqDestroy (usb_osa_msgq_handle handle)`
- Destroys a message queue.*
- `usb_osa_status_t USB_OsaMsgqSend (usb_osa_msgq_handle handle, void *msg)`
- Sends a message.*
- `usb_osa_status_t USB_OsaMsgqRecv (usb_osa_msgq_handle handle, void *msg, uint32_t timeout)`
- Receives a message.*
- `usb_osa_status_t USB_OsaMsgqCheck (usb_osa_msgq_handle handle, void *msg)`
- Checks a message queue and receives a message if the queue is not empty.*

3.2 Enumeration Type Documentation

3.2.1 enum usb_osa_status_t

Enumerator

kStatus_USB_OSA_Success Success.
kStatus_USB_OSA_Error Failed.
kStatus_USB_OSA_TimeOut Timeout occurs while waiting.

3.2.2 enum usb_osa_event_mode_t

Enumerator

kUSB_OsaEventManualClear The flags of the event is cleared manually.
kUSB_OsaEventAutoClear The flags of the event is cleared automatically.

3.3 Function Documentation

3.3.1 void* USB_OsaMemoryAllocate (uint32_t length)

The function is used to reserve the requested amount of memory in bytes and initializes it to 0.

Parameters

<i>length</i>	Amount of bytes to reserve.
---------------	-----------------------------

Returns

Pointer to the reserved memory. NULL if memory can't be allocated.

3.3.2 void USB_OsaMemoryFree (void * p)

The function is used to free the memory block previously reserved.

Function Documentation

Parameters

<i>p</i>	Pointer to the start of the memory block previously reserved.
----------	---

3.3.3 `usb_osa_status_t USB_OsaEventCreate (usb_osa_event_handle * handle, uint32_t flag)`

This function creates an event object and sets its clear mode. If the clear mode is `kUSB_OsaEventAutoClear`, when a task gets the event flags, these flags are cleared automatically. If the clear mode is `kUSB_OsaEventManualClear`, the flags must be cleared manually.

Parameters

<i>handle</i>	It is an out parameter, which is used to return the pointer of the event object.
<i>flag</i>	The event is auto-clear or manual-clear. See the enumeration usb_osa_event_mode_t .

Returns

A USB OSA error code or `kStatus_OSA_Success`.

Example:

```
usb_osa_event_handle eventHandle;  
usb_osa_status_t      usbOsaStatus;  
usbOsaStatus = USB_OsaEventCreate(&eventHandle,  
    kUSB_OsaEventManualClear);
```

3.3.4 `usb_osa_status_t USB_OsaEventDestroy (usb_osa_event_handle handle)`

Parameters

<i>handle</i>	Pointer to the event object.
---------------	------------------------------

Returns

A USB OSA error code or `kStatus_OSA_Success`.

Example:

```
usb_osa_status_t      usbOsaStatus;  
...  
usbOsaStatus = USB_OsaEventDestroy(eventHandle);
```

3.3.5 `usb_osa_status_t` `USB_OsaEventSet` (`usb_osa_event_handle` *handle*, `uint32_t` *bitMask*)

Sets specified flags for an event object.

Function Documentation

Parameters

<i>handle</i>	Pointer to the event object.
<i>bitMask</i>	Event flags to be set.

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_status_t    usbOsaStatus;  
...  
usbOsaStatus = USB_OsaEventSet(eventHandle, 0x01U);
```

3.3.6 usb_osa_status_t USB_OsaEventWait (usb_osa_event_handle *handle*, uint32_t *bitMask*, uint32_t *flag*, uint32_t *timeout*, uint32_t * *bitSet*)

This function waits for a combination of flags to be set in an event object. An applications can wait for any/all bits to be set. This function can get the flags that wake up the waiting task.

Parameters

<i>handle</i>	Pointer to the event object.
<i>bitMask</i>	Event flags to wait.
<i>flag</i>	Wait all flags or any flag to be set. 0U - wait any flag, others, wait all flags.
<i>timeout</i>	The maximum number of milliseconds to wait for the event. If the wait condition is not met, passing 0U waits indefinitely when the environment is an RTOS and returns the kStatus_OSA_Timeout immediately. Pass any value for the bare metal.
<i>bitSet</i>	Flags that wake up the waiting task are obtained by this parameter.

Returns

An USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_status_t    usbOsaStatus;  
uint32_t            bitSet;  
...  
usbOsaStatus = USB_OsaEventWait(eventHandle, 0x01U, 0U, 0U, &bitSet);
```

3.3.7 `usb_osa_status_t` `USB_OsaEventCheck` (`usb_osa_event_handle` *handle*, `uint32_t` *bitMask*, `uint32_t` * *bitSet*)

This function checks for a combination of flags to be set in an event object.

Function Documentation

Parameters

<i>handle</i>	Pointer to the event object.
<i>bitMask</i>	Event flags to check.
<i>bitSet</i>	Flags have been set.

Returns

An USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_status_t    usbOsaStatus;  
uint32_t            bitSet;  
...  
usbOsaStatus = USB_OsaEventCheck(eventHandle, 0x01U, &bitSet);
```

3.3.8 usb_osa_status_t USB_OsaEventClear (usb_osa_event_handle *handle*, uint32_t *bitMask*)

This function clears flags of an event object.

Parameters

<i>handle</i>	Pointer to the event object
<i>bitMask</i>	Event flags to be cleared.

Returns

An USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_status_t    usbOsaStatus;  
...  
usbOsaStatus = USB_OsaEventClear(eventHandle, 0x01U);
```

3.3.9 usb_osa_status_t USB_OsaSemCreate (usb_osa_sem_handle * *handle*, uint32_t *count*)

This function creates a semaphore and sets the default count.

Parameters

<i>handle</i>	It is an out parameter, which is used to return pointer of the semaphore object.
<i>count</i>	Initializes a value of the semaphore.

Returns

An USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_sem_handle    semHandle;
usb_osa_status_t      usbOsaStatus;
usbOsaStatus = USB_OsaSemCreate(&semHandle, 1U);
```

3.3.10 usb_osa_status_t USB_OsaSemDestroy (usb_osa_sem_handle *handle*)

This function destroys a semaphore object.

Parameters

<i>handle</i>	Pointer to the semaphore.
---------------	---------------------------

Returns

An USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_sem_handle    semHandle;
usb_osa_status_t      usbOsaStatus;
...
usbOsaStatus = USB_OsaSemDestroy(semHandle);
```

3.3.11 usb_osa_status_t USB_OsaSemPost (usb_osa_sem_handle *handle*)

This function wakes up a task waiting on the semaphore. If a task is not pending, increases the semaphore's value.

Function Documentation

Parameters

<i>handle</i>	Pointer to the semaphore.
---------------	---------------------------

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_sem_handle    semHandle;
usb_osa_status_t      usbOsaStatus;
...
usbOsaStatus = USB_OsaSemPost(semHandle);
```

3.3.12 usb_osa_status_t USB_OsaSemWait (usb_osa_sem_handle *handle*, uint32_t *timeout*)

This function checks the semaphore's value. If it is positive, it decreases the semaphore's value and return kStatus_OSA_Success.

Parameters

<i>handle</i>	Pointer to the semaphore.
<i>timeout</i>	The maximum number of milliseconds to wait for the semaphore. If the wait condition is not met, passing 0U waits indefinitely when environment is RTOS. And return kStatus_OSA_Timeout immediately for bare metal no matter what value has been passed.

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_sem_handle    semHandle;
usb_osa_status_t      usbOsaStatus;
...
usbOsaStatus = USB_OsaSemWait(semHandle, 0U);
```

3.3.13 usb_osa_status_t USB_OsaMutexCreate (usb_osa_mutex_handle * *handle*)

This function creates a mutex and sets it to an unlocked status.

Parameters

<i>handle</i>	It is out parameter, which is used to return the pointer of the mutex object.
---------------	---

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_mutex_handle mutexHandle;
usb_osa_status_t      usbOsaStatus;
usbOsaStatus = USB_OsaMutexCreate(&mutexHandle);
```

3.3.14 usb_osa_status_t USB_OsaMutexDestroy (usb_osa_mutex_handle *handle*)

This function destroys a mutex and sets it to an unlocked status.

Parameters

<i>handle</i>	Pointer to the mutex.
---------------	-----------------------

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_mutex_handle mutexHandle;
usb_osa_status_t      usbOsaStatus;
...
usbOsaStatus = USB_OsaMutexDestroy(mutexHandle);
```

3.3.15 usb_osa_status_t USB_OsaMutexLock (usb_osa_mutex_handle *handle*)

This function checks the mutex status. If it is unlocked, it locks it and returns the kStatus_OSA_Success. Otherwise, it waits forever to lock in RTOS and returns the kStatus_OSA_Success immediately for bare metal.

Function Documentation

Parameters

<i>handle</i>	Pointer to the mutex.
---------------	-----------------------

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_mutex_handle mutexHandle;  
usb_osa_status_t      usbOsaStatus;  
...  
usbOsaStatus = USB_OsaMutexLock(mutexHandle);
```

3.3.16 usb_osa_status_t USB_OsaMutexUnlock (usb_osa_mutex_handle *handle*)

This function unlocks a mutex.

Parameters

<i>handle</i>	Pointer to the mutex.
---------------	-----------------------

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_mutex_handle mutexHandle;  
usb_osa_status_t      usbOsaStatus;  
...  
usbOsaStatus = USB_OsaMutexUnlock(mutexHandle);
```

3.3.17 usb_osa_status_t USB_OsaMsgqCreate (usb_osa_msgq_handle * *handle*, uint32_t *count*, uint32_t *size*)

This function creates a message queue.

Parameters

<i>handle</i>	It is an out parameter, which is used to return a pointer of the message queue object.
<i>count</i>	The count of elements in the queue.
<i>size</i>	Size of every elements in words.

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_msgq_handle msgqHandle;
usb_osa_status_t    usbOsaStatus;
usbOsaStatus = USB_OsaMsgqCreate(msgqHandle, 8U, 4U);
```

3.3.18 usb_osa_status_t USB_OsaMsgqDestroy (usb_osa_msgq_handle *handle*)

This function destroys a message queue.

Parameters

<i>handle</i>	Pointer to a message queue.
---------------	-----------------------------

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_msgq_handle msgqHandle;
usb_osa_status_t    usbOsaStatus;
...
usbOsaStatus = USB_OsaMsgqDestroy(msgqHandle);
```

3.3.19 usb_osa_status_t USB_OsaMsgqSend (usb_osa_msgq_handle *handle*, void * *msg*)

This function sends a message to the tail of the message queue.

Function Documentation

Parameters

<i>handle</i>	Pointer to a message queue.
<i>msg</i>	The pointer to a message to be put into the queue.

Returns

A USB OSA error code or `kStatus_OSA_Success`.

Example:

```
usb_osa_msgq_handle    msgqHandle;
message_struct_t       message;
usb_osa_status_t       usbOsaStatus;
...
usbOsaStatus = USB_OsaMsgqSend(msgqHandle, &message);
```

3.3.20 `usb_osa_status_t USB_OsaMsgqRecv (usb_osa_msgq_handle handle, void * msg, uint32_t timeout)`

This function receives a message from the head of the message queue.

Parameters

<i>handle</i>	Pointer to a message queue.
<i>msg</i>	The pointer to save a received message.
<i>timeout</i>	The maximum number of milliseconds to wait for a message. If the wait condition is not met, passing 0U waits indefinitely when an environment is RTOS and returns the <code>kStatus_OSA_Timeout</code> immediately for bare metal.

Returns

A USB OSA error code or `kStatus_OSA_Success`.

Example:

```
usb_osa_msgq_handle    msgqHandle;
message_struct_t       message;
usb_osa_status_t       usbOsaStatus;
...
usbOsaStatus = USB_OsaMsgqRecv(msgqHandle, &message, 0U);
```

3.3.21 `usb_osa_status_t USB_OsaMsgqCheck (usb_osa_msgq_handle handle, void * msg)`

This function checks a message queue and receives a message if the queue is not empty.

Parameters

<i>handle</i>	Pointer to a message queue.
<i>msg</i>	The pointer to save a received message.

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_msgq_handle    msgqHandle;  
message_struct_t       message;  
usb_osa_status_t       usbOsaStatus;  
...  
usbOsaStatus = USB_OsaMsgqCheck(msgqHandle, &message);
```


How to Reach Us:**Home Page:**

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

NXP reserves the right to make changes without further notice to any products herein. NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages.

“Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address:

nxp.com/SalesTermsandConditions.

NXP, the NXP logo, Freescale, the Freescale logo, Kinetis, Processor Expert are trademarks of NXP B.V. Tower is a trademark of NXP B.V. All other product or service names are the property of their respective owners. ARM, ARM Powered logo, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2017 NXP B.V.

Document Number: MCUXSDKUSBOAPIRM

Rev. 0

Mar 2017

