



USB Super Speed Host software driver user guide

Product Version 2.0.7

August 2020

© 1996-2020 Cadence Design Systems, Inc. All rights reserved.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522. All other trademarks are the property of their respective holders.

Restricted Permission: This document is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this document, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this document may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. This document contains the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only in accordance with, a written agreement between Cadence and its customer.

Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this document subject to the following conditions:

1. This document may not be modified in any way.
2. Any authorized copy of this document or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
3. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

Disclaimer: INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT NOTICE AND DOES NOT REPRESENT A COMMITMENT ON THE PART OF CADENCE. EXCEPT AS MAY BE EXPLICITLY SET FORTH IN A WRITTEN AGREEMENT BETWEEN CADENCE AND ITS CUSTOMER, CADENCE DOES NOT MAKE, AND EXPRESSLY DISCLAIMS, ANY REPRESENTATIONS OR WARRANTIES AS TO THE COMPLETENESS, ACCURACY OR USEFULNESS OF THE INFORMATION CONTAINED IN THIS DOCUMENT. CADENCE DOES NOT WARRANT THAT USE OF SUCH INFORMATION WILL NOT INFRINGE ANY THIRD PARTY RIGHTS, AND CADENCE DISCLAIMS ALL IMPLIED WARRANTIES, INCLUDING MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. CADENCE DOES NOT ASSUME ANY LIABILITY FOR DAMAGES OR COSTS OF ANY KIND THAT MAY RESULT FROM USE OF SUCH INFORMATION. CADENCE CUSTOMER HAS COMPLETE CONTROL AND FINAL DECISION-MAKING AUTHORITY OVER ALL ASPECTS OF THE DEVELOPMENT, MANUFACTURE, SALE AND USE OF CUSTOMER'S PRODUCT, INCLUDING, BUT NOT LIMITED TO, ALL DECISIONS WITH REGARD TO DESIGN, PRODUCTION, TESTING, ASSEMBLY, QUALIFICATION, CERTIFICATION, INTEGRATION OF CADENCE PRODUCTS, INSTRUCTIONS FOR USE, LABELING AND DISTRIBUTION, AND CADENCE EXPRESSLY DISAVOWS ANY RESPONSIBILITY WITH REGARD TO ANY SUCH DECISIONS REGARDING CUSTOMER'S PRODUCT.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227- 14 and DFAR252.227-7013 et seq. or its successor.

USB Super Speed Host software driver user guide

Cadence Design Systems

Hardware Identifier:1bffd26f756c8a5110b236b2b4af3c1c

Table of Contents

1. Introduction	1
1.1. Acronyms	1
1.2. Document Purpose	1
1.3. Naming conventions	1
2. Configuration and Hardware Operation Information	2
2.1. Introduction	2
3. Data Structures	11
3.1. Introduction	11
3.2. USBSSP_RingElementT_s	11
3.3. USBSSP_InputContextT_s	11
3.4. USBSSP_OutputContextT_s	11
3.5. USBSSP_XferDescT_s	12
3.6. USBSSP_ProducerQueueT_s	12
3.7. USBSSP_DescT_s	13
3.8. USBSSP_DcbaaT_s	14
3.9. USBSSP_XhciResourcesT_s	14
3.10. USBSSP_IpIdVerRegs_s	14
3.11. USBSSP_DriverResourcesT_s	15
3.12. USBSSP_DriverContextT_s	17
3.13. USBSSP_ForceHdrParams_s	17
3.14. USBSSP_XferBufferDesc_s	18
3.15. CUSBD_EpAuxBufferConfig_s	18
3.16. CUSBD_EpAuxBuffer_s	18
3.17. CUSBD_EpConfig_s	19
3.18. CUSBD_Config_s	19
3.19. CUSBD_SysReq_s	20
3.20. CUSBD_SgList_s	20
3.21. CUSBD_Req_s	21
3.22. CUSBD_EpOps_s	21
3.23. CUSBD_Ep_s	22
3.24. CUSBD_Dev_s	23
3.25. CUSBD_Callbacks_s	23
3.26. CUSBD_EpPrivate_s	24
3.27. CUSBD_PrivateData_s	24
3.28. CUSBDMA_DmaTrb_s	25
3.29. CUSBDMA_MemResources_s	25
3.30. CUSBDMA_Config_s	26
3.31. CUSBDMA_SysReq_s	26
3.32. CUSBDMA_ChannelParams_s	27
3.33. CUSBDMA_DmaTrbChainDesc_s	27
3.34. CUSBDMA_DmaChannel_s	28
3.35. CUSBDMA_DmaController_s	29
3.36. CUSBDMA_DmaTransferParam_s	29
3.37. LIST_ListHead_s	29
3.38. USB_SSP_DRD_Config_s	30
3.39. USB_SSP_DRD_Callbacks_s	30
3.40. USB_SSP_DRD_State_s	30
3.41. USB_SSP_DRD_PrivData_s	31
3.42. USB_SSP_DRD_SysReq_s	31

3.43. USBSSP_CapabilityT_s	32
3.44. USBSSP_QuickAccessRegs_s	32
3.45. USBSSP_PortControlT_s	32
3.46. USBSSP_OperationalT_s	33
3.47. USBSSP_InterrupterT_s	33
3.48. USBSSP_RuntimeT_s	34
3.49. USBSSP_ExtCapElemT_s	34
3.50. USBSSP_ExtCapSetT_s	34
3.51. USBSSP_SfrT_s	35
3.52. USBSSP_Ep0StateEnum	35
3.53. USBSSP_ExtraFlagsEnumT	36
3.54. CUSBD_DMAInterfaceWidth	36
3.55. CUSBD_epState	37
3.56. CUSBDMA_Status	37
3.57. USB_SSP_DRD_Mode	38
3.58. USBSSP_EpContextEpTypeT	38
3.59. USBSSP_EpContextEpState	39
3.60. USBSSP_SlotContextState	40
3.61. USBSSP_PortControlRegIdx	40
3.62. USBSSP_DmaAddrT	41
3.63. USBSSP_Complete	41
3.64. USBSSP_NopComplete	41
3.65. USBSSP_ForceHeaderComplete	42
3.66. USBSSP_GenericCallback	42
3.67. USBSSP_PostCallback	42
3.68. USBSSP_PreportChangeDetectCallback	43
3.69. USBSSP_InputContextCallback	43
3.70. USBSSP_USB2PhySoftReset	43
3.71. USBSSP_USB2ResetCallback	43
3.72. USBSSP_get_phys_from_log_ptr_proc_t	44
3.73. USBSSP_get_log_from_phys_ptr_proc_t	44
3.74. CUSBD_CbReqComplete	44
3.75. CUSBD_CbEpEnable	45
3.76. CUSBD_CbEpDisable	45
3.77. CUSBD_CbEpSetHalt	45
3.78. CUSBD_CbEpSetWedge	46
3.79. CUSBD_CbEpFifoStatus	46
3.80. CUSBD_CbEpFifoFlush	46
3.81. CUSBD_CbReqQueue	47
3.82. CUSBD_CbReqDequeue	47
3.83. CUSBD_CbDisconnect	47
3.84. CUSBD_CbConnect	48
3.85. CUSBD_CbSetup	48
3.86. CUSBD_CdSuspend	48
3.87. CUSBD_CbResume	48
3.88. CUSBD_CbbusInterval	49
3.89. CUSBD_CbDescMissing	49
3.90. CUSBD_CbUSB2PhySoftReset	49
3.91. USB_SSP_DRD_GenericCallback	50
3.92. USB_SSP_DRD_InterruptHandler	50
3.93. USB_SSP_DRD_EventIdUp	50

3.94. USB_SSP_DRD_EventIdDown	50
3.95. USB_SSP_DRD_EventIdUpVBusDown	51
3.96. USB_SSP_DRD_EventIdUpVBusUp	51
3.97. USB_SSP_DRD_EventIdDownVBusDown	51
3.98. USB_SSP_DRD_EventIdDownVBusUp	52
3.99. USB_SSP_DRD_EventEnableDrd	52
3.100. USB_SSP_DRD_EventEnableHost	52
3.101. USB_SSP_DRD_EventEnableDev	52
3.102. USB_SSP_DRD_EventDisableHost	53
3.103. USB_SSP_DRD_EventDisableDev	53
3.104. __attribute__	53
3.105. __attribute__	53
3.106. __attribute__	53
3.107. __attribute__	53
3.108. __attribute__	54
4. Driver API Object	55
4.1. Introduction	55
4.2. USBSSP_GetInstance	55
4.3. CUSBD_GetInstance	55
4.4. CUSBDMA_GetInstance	55
4.5. USB_SSP_DRD_GetInstance	56
4.6. transferData	56
4.7. transferVectorData	57
4.8. stopEndpoint	58
4.9. resetEndpoint	58
4.10. resetDevice	59
4.11. isr	59
4.12. SetMemRes	60
4.13. init	60
4.14. getDescriptor	61
4.15. setAddress	62
4.16. resetRootHubPort	62
4.17. issueGenericCommand	63
4.18. endpointSetFeature	63
4.19. setConfiguration	64
4.20. nBControlTransfer	65
4.21. noOpTest	65
4.22. calcFsLsEPIntrptInterval	66
4.23. enableSlot	66
4.24. disableSlot	67
4.25. enableEndpoint	67
4.26. disableEndpoint	68
4.27. getMicroFrameIndex	68
4.28. setEndpointExtraFlag	69
4.29. cleanEndpointExtraFlag	69
4.30. getEndpointExtraFlag	70
4.31. setFrameID	71
4.32. addEventDataTRB	71
4.33. forceHeader	72
4.34. setPortOverrideReg	72
4.35. getPortOverrideReg	73

4.36. setPortControlReg	73
4.37. getPortControlReg	74
4.38. SaveState	74
4.39. RestoreState	75
4.40. getPortConnected	75
4.41. getDevAddressState	76
4.42. probe	76
4.43. init	77
4.44. destroy	77
4.45. start	78
4.46. stop	78
4.47. isr	79
4.48. epEnable	79
4.49. epDisable	80
4.50. epSetHalt	81
4.51. epSetWedge	81
4.52. epFifoStatus	82
4.53. epFifoFlush	82
4.54. reqQueue	83
4.55. reqDequeue	84
4.56. getDevInstance	84
4.57. dGetFrame	85
4.58. dSetSelfpowered	85
4.59. dClearSelfpowered	86
4.60. dGetConfigParams	86
4.61. probe	87
4.62. init	87
4.63. destroy	88
4.64. channelAlloc	88
4.65. channelReset	89
4.66. channelRelease	89
4.67. channelProgram	90
4.68. channelTrigger	91
4.69. channelUpdateState	91
4.70. channelSetMaxPktSz	92
4.71. channelHandleStall	92
4.72. channelFreeHeadTrbChain	93
4.73. Probe	94
4.74. Init	94
4.75. Isr	95
4.76. Start	95
4.77. Stop	96
4.78. Destroy	96
4.79. CheckIfReady	97
4.80. CheckStrapMode	97
4.81. CheckOperationMode	98
4.82. SetOperationMode	99
4.83. CheckIrq	99
4.84. ProcessIrq	100
5. Driver Function API	101
5.1. Introduction	101

5.2. USBSSP_TransferData	101
5.3. USBSSP_TransferVectorData	102
5.4. USBSSP_StopEndpoint	103
5.5. USBSSP_ResetEndpoint	104
5.6. USBSSP_ResetDevice	105
5.7. USBSSP_Isr	106
5.8. USBSSP_SetMemRes	106
5.9. USBSSP_Init	107
5.10. USBSSP_GetDescriptor	108
5.11. USBSSP_SetAddress	109
5.12. USBSSP_ResetRootHubPort	109
5.13. USBSSP_IssueGenericCommand	110
5.14. USBSSP_EndpointSetFeature	111
5.15. USBSSP_SetConfiguration	112
5.16. USBSSP_NBControlTransfer	113
5.17. USBSSP_NoOpTest	114
5.18. USBSSP_CalcFsLsEPIntrptInterval	115
5.19. USBSSP_EnableSlot	115
5.20. USBSSP_DisableSlot	116
5.21. USBSSP_EnableEndpoint	116
5.22. USBSSP_DisableEndpoint	117
5.23. USBSSP_GetMicroFrameIndex	118
5.24. USBSSP_SetEndpointExtraFlag	119
5.25. USBSSP_CleanEndpointExtraFlag	119
5.26. USBSSP_GetEndpointExtraFlag	120
5.27. USBSSP_SetFrameID	121
5.28. USBSSP_AddEventDataTRB	122
5.29. USBSSP_ForceHeader	123
5.30. USBSSP_SetPortOverrideReg	124
5.31. USBSSP_GetPortOverrideReg	125
5.32. USBSSP_SetPortControlReg	126
5.33. USBSSP_GetPortControlReg	127
5.34. USBSSP_SaveState	128
5.35. USBSSP_RestoreState	128
5.36. USBSSP_GetPortConnected	129
5.37. USBSSP_GetDevAddressState	130
5.38. CUSBD_Probe	130
5.39. CUSBD_Init	131
5.40. CUSBD_Destroy	132
5.41. CUSBD_Start	133
5.42. CUSBD_Stop	133
5.43. CUSBD_Isr	134
5.44. CUSBD_EpEnable	134
5.45. CUSBD_EpDisable	135
5.46. CUSBD_EpSetHalt	135
5.47. CUSBD_EpSetWedge	136
5.48. CUSBD_EpFifoStatus	137
5.49. CUSBD_EpFifoFlush	137
5.50. CUSBD_ReqQueue	138
5.51. CUSBD_ReqDequeue	139
5.52. CUSBD_GetDevInstance	139

5.53. CUSBD_DGetFrame	140
5.54. CUSBD_DSetSelfpowered	140
5.55. CUSBD_DClearSelfpowered	141
5.56. CUSBD_DGetConfigParams	141
5.57. CUSBDMA_Probe	142
5.58. CUSBDMA_Init	143
5.59. CUSBDMA_Destroy	143
5.60. CUSBDMA_ChannelAlloc	144
5.61. CUSBDMA_ChannelReset	145
5.62. CUSBDMA_ChannelRelease	146
5.63. CUSBDMA_ChannelProgram	146
5.64. CUSBDMA_ChannelTrigger	147
5.65. CUSBDMA_ChannelUpdateState	148
5.66. CUSBDMA_ChannelSetMaxPktSz	149
5.67. CUSBDMA_ChannelHandleStall	150
5.68. CUSBDMA_ChannelFreeHeadTrbChain	151
5.69. USB_SSP_DRD_Probe	151
5.70. USB_SSP_DRD_Init	152
5.71. USB_SSP_DRD_Isr	153
5.72. USB_SSP_DRD_Start	153
5.73. USB_SSP_DRD_Stop	154
5.74. USB_SSP_DRD_Destroy	154
5.75. USB_SSP_DRD_CheckIfReady	155
5.76. USB_SSP_DRD_CheckStrapMode	155
5.77. USB_SSP_DRD_CheckOperationMode	156
5.78. USB_SSP_DRD_SetOperationMode	156
5.79. USB_SSP_DRD_CheckIrq	157
5.80. USB_SSP_DRD_ProcessIrq	158

List of Tables

2.1. Configuration and Hardware Operation Information 2

Chapter 1. Introduction

1.1. Acronyms

API Application Programming Interface.

The driver Cadence USB Super Speed Host software driver.

1.2. Document Purpose

This document is an overview and reference of user accessible functions (ie. API) of the Cadence USB Super Speed Host software driver intended to be run with Cadence USB Super Speed Host hardware controller.

1.3. Naming conventions

The driver is using a prefix *USBSSP_* for user accessible functions, constants and data objects definitions.

Chapter 2. Configuration and Hardware Operation Information

2.1. Introduction

The following definitions specify the driver operation environment that is defined by hardware configuration or client code.

These defines are located in the header file of the core driver.

Table 2.1. Configuration and Hardware Operation Information

#define	value	description
USBSSP_DID_VAL	0x0004024EU	DID value.
USBSSP_RID_VAL	0x0200U	RID value.
USBSSP_EXTENDED_CONTEXT	0U	extended context
USBSSP_CONTEXT_WIDTH	8U	context width
USBSSP_DBG_DRV	0x000000010U	Debug driver.
USBSSP_HOST_OFFSET	0x0000U	host offset
USBSSP_DEVICE_OFFSET	0x4000U	device offset
USBSSP_OTG_OFFSET	0x8000U	OTG offset.
USBSSP_MAGIC_NUMBER	0x0004034EU	magic number
USBSSP_RING_ALIGNMENT	64U	ring alignment
USBSSP_RING_BOUNDARY	65536U	ring boundary
USBSSP_ERST_ALIGNMENT	64U	event ring segment table alignment
USBSSP_ERST_BOUNDARY	0U	event ring segment table boundary
USBSSP_ERST_ENTRY_SIZE	8U	event ring segment table entry size
USBSSP_CONTEXT_ALIGNMENT	64U	context alignment
USBSSP_DCBAAL_ALIGNMENT	64U	Device context base address array alignment.
USBSSP_INTERRUPTER_COUNT	4U	Number of interrupters supported by software driver.
USBSSP_MAX_NUM_INTERRUPTERS	8U	Max number of interrupters supported by the hardware.
USBSSP_DBG_TEST	0x000000040U	Debug test value.
USBSSP_DBG_EXTERNAL_STACK	0x000000080U	Debug external stack.
USBSSP_EVENT_QUEUE_SIZE	64U	Event queue size.
USBSSP_MAX_DEVICE_SLOT_NUM	64U	Maximum device slot supported by the hardware.
USBSSP_SCRATCHPAD_BUFF_NUM	((USBSSP_MAX_DEVICE_SLOT_NUM + 1U) >> 1U)	Scratchpad buffer
USBSSP_PAGE_SIZE	4096U	page size

#define	value	description
USBSSP_PRODUCER_QUEUE_SIZE	64U	producer queue size
USBSSP_XFER_DESC_QUEUE_SIZE	$((\text{USBSSP_PRODUCER_QUEUE_SIZE} + 7\text{U}) \gg 3\text{U}) + 1\text{U}$	Transfer Descriptor queue size.
USBSSP_MAX_SPEED	6U	max speed
USBSSP_MAX_STRING_NUM	5U	max string number
USBSSP_MAX_EP_CONTEXT_NUM	30U	max endpoint context
USBSSP_MAX_EP_NUM_STRM_EN	30U	max endpoint number of streams
USBSSP_MAX_STREMS_PER_EP	2U	given according to XHCI register value: 1 = 4 streams, 2 = 8 streams, 3 = 16 streams and so on
USBSSP_STREAM_ARRAY_SIZE	8U	Should be calculated according to formula: $\text{STREAM_ARRAY_SIZE} = 2^{\text{exp}(\text{MAX_STREMS_PER_EP} + 1)}$
USBSSP_DEV_MODE_2_PORT	0U	Device mode ports settings.
USBSSP_DEV_MODE_3_PORT	1U	Device mode ports settings.
USBSSP_EP0_CONT_OFFSET	1U	Endpoint0 container offset value.
USBSSP_EP_CONT_OFFSET	2U	Endpoint container offset value.
USBSSP_EP_CONT_MAX	$(\text{USBSSP_MAX_EP_CONTEXT_NUM} + \text{USBSSP_EP_CONT_OFFSET})$	Endpoint container offset max value.
USBSSP_EP0_DATA_BUFF_SIZE	1024U	Endpoint 0 data buffer size - used in enumeration.
USBSSP_RX_DETECT_LINK_STATE	5U	Link is in the RxDetect State.
USBSSP_EALIGN	130U	Value returned by API functions to report align error.
USBSSP_EPAGE	131U	Value returned by API functions to report memory out of page error.
USBSSP_EEVENT	132U	Value returned by USBSSP_Isr to report occurrence of unexpectedly large number of consecutive XHCI events.
USBSSP_ESTALL	133U	Status code reported by callback when transfer stalls.
USBSSP_DELAY_T3_DEBOUNCE	1U	Delay in ms for delta-T3 debounce interval before reset.
USBSSP_DELAY_T6_RECOVERY	1U	Delay in ms for delta-T6 reset recovery.
CUSBD_DID_VAL	0x0004024EU	DID value.
CUSBD_RID_VAL	0x0200U	RID value.
CUSBD_NUM_EP_OUT	15U	Number of out endpoints.
CUSBD_NUM_EP_IN	15U	Number of in endpoints.

Configuration and Hardware Operation Information

#define	value	description
CUSBD_INACTIVITY_TMOU	0x7FFFU	Inactivity timeout value corresponding to max value of LFPS_POLLING_MAX_TREPEAT.
CUSBD_PRECISE_BURST_0	0x0000000U	Precise burst length 0 (valid for OCP, AHB, AXI3)
CUSBD_PRECISE_BURST_1	0x0100000U	Precise burst length 1 (valid for OCP, AHB, AXI3)
CUSBD_PRECISE_BURST_2	0x0200000U	Precise burst length 2 (valid for OCP, AXI3)
CUSBD_PRECISE_BURST_4	0x0400000U	Precise burst length 4 (valid for OCP, AHB, AXI3)
CUSBD_PRECISE_BURST_8	0x0800000U	Precise burst length 8 (valid for OCP, AHB, AXI3)
CUSBD_PRECISE_BURST_16	0x1000000U	Precise burst length 16 (valid for OCP, AHB, AXI3)
CUSBD_PRECISE_BURST_32	0x2000000U	Precise burst length 32 (valid for OCP, AXI3)
CUSBD_PRECISE_BURST_64	0x4000000U	Precise burst length 64 (valid for OCP, AXI3)
CUSBD_PRECISE_BURST_128	0x8000000U	Precise burst length 128 (valid for OCP, AXI3)
CUSBD_ENDIANESS_CONV_FLAG	0U	Value is 0 to disable it and 1 to enable it
CUSBDMA_TRB_CHAIN_FREE	0U	The TRB chain descriptor is unused and free.
CUSBDMA_TRB_CHAIN_QUEUED	1U	The TRB chain descriptor is queued for processing.
CUSBDMA_TRB_CHAIN_PARTIAL	2U	The TRB chain descriptor is partially processed.
CUSBDMA_TRB_CHAIN_COMPLETE	3U	The TRB chain descriptor is completely processed.
CUSBDMA_MAX_DMA_CHANNELS	16U	This is used internally by the driver.
CUSBDMA_TRB_SIZE_OF_DMA_CHAIN(CUSBDMA_MAX_DMA_CHANNELS * 16U)		It means that for every channel for IN and OUT direction we can allocate average one TRBS chain.
CUSBDMA_NUM_OF_IN_ENDPOINTS	16U	Number of IN endpoints.
CUSBDMA_OUT_ENDPOINT_OFFSET	16U	OUT endpoint offset.
USB_SSP_DRD_NRDY_TIMEOUT	20U	Timeout value for waiting for DRD Ready.
USB_SSP_DRD_DID_VALUE	0x0004024EU	DID value of compatible hardware.
USB_SSP_DRD_RID_VALUE	0x200U	RID value of compatible hardware.

#define	value	description
USB_SSP_DRD_ALL_INTERRUPTS	0xFFFFFFFFU	Macro for disabling/enabling all interrupts.
USBSSP_MAX_EXT_CAP_ELEM_DWORDS	256U	capability
USBSSP_MAX_EXT_CAPS_COUNT	3U	USB legacy support are not taken under consideration
USBSSP_TRB_NORMAL	(uint32_t)1U	Normal TRB.
USBSSP_TRB_SETUP_STAGE	(uint32_t)2U	Setup stage TRB.
USBSSP_TRB_DATA_STAGE	(uint32_t)3U	Data stage TRB.
USBSSP_TRB_STATUS_STAGE	(uint32_t)4U	Status stage TRB.
USBSSP_TRB_ISOCH	(uint32_t)5U	Isoch TRB.
USBSSP_TRB_LINK	(uint32_t)6U	Link TRB.
USBSSP_TRB_EVENT_DATA	(uint32_t)7U	Event data TRB.
USBSSP_TRB_NO_OP	(uint32_t)8U	No Op TRB.
USBSSP_TRB_ENABLE_SLOT_COMMAND	(uint32_t)9U	Enable slot command TRB.
USBSSP_TRB_DISABLE_SLOT_COMMAND	(uint32_t)10U	Disable slot command TRB.
USBSSP_TRB_ADDR_DEV_CMD	(uint32_t)11U	Address device command TRB.
USBSSP_TRB_CONF_EP_CMD	(uint32_t)12U	Configure endpoint command TRB.
USBSSP_TRB_EVALUATE_CXT_CMD	(uint32_t)13U	Evaluate context command TRB.
USBSSP_TRB_RESET_EP_CMD	(uint32_t)14U	Reset endpoint command TRB.
USBSSP_TRB_STOP_EP_CMD	(uint32_t)15U	Stop endpoint command TRB.
USBSSP_TRB_SET_TR_DQ_PTR_CMD	(uint32_t)16U	Set TR Dequeue pointer command TRB.
USBSSP_TRB_RESET_DEVICE_COMMAND	(uint32_t)17U	Reset device command TRB.
USBSSP_TRB_FORCE_EVENT_COMMAND	(uint32_t)18U	Force event command TRB.
USBSSP_TRB_NEGOTIATE_BANDWIDTH	(uint32_t)19U	Negotiate bandwidth TRB.
USBSSP_TRB_SET_LAT_TOL_VAL_CMD	(uint32_t)20U	Set Latency Tolerance Value Command TRB.
USBSSP_TRB_GET_PORT_BNDWTH_CMD	(uint32_t)21U	Get port bandwidth command TRB.
USBSSP_TRB_FORCE_HEADER_COMMAND	(uint32_t)22U	Force header command TRB.
USBSSP_TRB_NO_OP_COMMAND	(uint32_t)23U	No Op command TRB.
USBSSP_TRB_TRANSFER_EVENT	(uint32_t)32U	Transfer event TRB.
USBSSP_TRB_CMD_CMPL_EVT	(uint32_t)33U	Command completion event TRB.
USBSSP_TRB_PORT_ST_CHG_EVT	(uint32_t)34U	Port status change event TRB.
USBSSP_TRB_BNDWTH_RQ_EVT	(uint32_t)35U	Bandwidth request event TRB.
USBSSP_TRB_DOORBELL_EVENT	(uint32_t)36U	Doorbell event TRB.
USBSSP_TRB_HOST_CTRL_EVT	(uint32_t)37U	Host controller event TRB.
USBSSP_TRB_DEV_NOTIFCN_EVT	(uint32_t)38U	Device notification event TRB.
USBSSP_TRB_MFINDEX_WRAP_EVENT	(uint32_t)39U	MFINDEX Wrap event TRB.

#define	value	description
USBSSP_TRB_NRDY_EVT	(uint32_t)48U	Vendor-defined TRBs (for USBSSP)
USBSSP_TRB_SETUP_PROTO_ENDP_CMD	(uint32_t)49U	Setup proto endpoint command TRB.
USBSSP_TRB_GET_PROTO_ENDP_CMD	(uint32_t)50U	Get proto endpoint command TRB.
USBSSP_TRB_SET_ENDPS_ENA_CMD	(uint32_t)51U	Set endpoint enable command TRB.
USBSSP_TRB_GET_ENDPS_ENA_CMD	(uint32_t)52U	Get endpoint enable command TRB.
USBSSP_TRB_ADD_TDL_CMD	(uint32_t)53U	Add TDL command TRB.
USBSSP_TRB_HALT_ENDP_CMD	(uint32_t)54U	Halt endpoint command TRB.
USBSSP_TRB_SETUP_STAGE1	(uint32_t)55U	Setup stage TRB.
USBSSP_TRB_HALT_ENDP_CMD1	(uint32_t)56U	Halt endpoint command TRB.
USBSSP_TRB_DRBL_OVERFLOW_EVENT	(uint32_t)57U	Doorbell overflow event TRB.
USBSSP_TRB_FLUSH_EP_CMD	(uint32_t)58U	Flush endpoint command TRB.
USBSSP_TRB_VF_SEC_VIOLN_EVT	(uint32_t)59U	VF security violation event TRB.
USBSSP_TRB_TBC_TBSTS_POS	(uint32_t)7U	Position of Transfer Burst Count of a TRB depending on ETE value.
USBSSP_TRB_TBC_TBSTS_MASK	(uint32_t)0x180U	Mask of Transfer Burst Count of a TRB depending on ETE value.
USBSSP_TRB_TDSIZE_TBC_POS	(uint32_t)17U	Position of TD size of TBC of a TRB depending on ETE value.
USBSSP_TRB_TDSIZE_TBC_MASK	(uint32_t)0x3E0000U	Mask of TD size of TBC of a TRB depending on ETE value.
USBSSP_TRB_INTR_TRGT_POS	(uint32_t)22U	Interrupter target position in TRB dword2.
USBSSP_TRB_TYPE_POS	(uint32_t)10U	Position offset for TRB Type.
USBSSP_BSR_POS	9U	Position offset for BSR.
USBSSP_TRB_MAX_TRANSFER_LENGTH	(uint32_t)0x10000U	TRB max transfer length.
USBSSP_SYSTEM_MEMORY_PAGE_SIZE	(uint32_t)0x10000U	System memory page size.
USBSSP_TRB_NORMAL_ISP_MASK	(uint32_t)0x4U	Mask for TRB normal ISP field.
USBSSP_TRB_NORMAL_CH_MASK	(uint32_t)0x10U	Mask for TRB normal CH field.
USBSSP_TRB_BMREQUESTTYPE_POS	0U	Position offset for bmrequesttype field.
USBSSP_TRB_BREQUEST_POS	8U	Position offset for brequest field.
USBSSP_TRB_WVALUE_POS	16U	Position offset for wvalue field.
USBSSP_TRB_WINDEX_POS	0U	Position offset for windex field.
USBSSP_TRB_FORCEEV_VF_ID_POS	16U	Position offset for VF ID field in force event command TRB.
USBSSP_TRB_FRCEVT_VFINTTGT_POS	22U	Position offset for VF Interrupter target in force event command TRB.
USBSSP_TRB_WLENGTH_POS	16U	Position offset for wlength field.
USBSSP_TRB_SETUPID_POS	8U	Position offset for setup ID field.

#define	value	description
USBSSP_TRB_SETUPID_MASK	(uint32_t)0x300U	Mask for setup ID field.
USBSSP_TRB_STS_STG_STAT_POS	6U	Position offset for setup ID field.
USBSSP_TRB_SPEED_ID_2	0x00U	TRB speed ID 2.
USBSSP_TRB_SPEED_ID_3	0x80U	TRB speed ID 3.
USBSSP_TRB_STS_STG_STAT_ACK	1U	Setup Data acknowledgment.
USBSSP_TRB_STS_STG_STAT_STALL	0U	Setup data stall.
USBSSP_TRB_NORMAL_IOC_MASK	(uint32_t)0x20U	IOC mask for normal TRB.
USBSSP_TRB_NORMAL_IDT_MASK	(uint32_t)0x40U	IDT mask for normal TRB.
USBSSP_TRB_NORMAL_ENT_MASK	(uint32_t)0x02U	ENT mask for normal TRB.
USBSSP_TRB_NORM_TRFR_LEN_MSK	(uint32_t)0x1FFFFU	Transfer length mask for normal TRB.
USBSSP_TRB_LNK_TGLE_CYC_MSK	(uint32_t)0x02U	Toggle cycle mask for Link TRB.
USBSSP_TRB_EVT_RESIDL_LEN_MSK	(uint32_t)0xFFFFFU	Residual length mask for event transfer TRB.
USBSSP_TRB_COMPLETE_INVALID	0U	TRB completion code invalid.
USBSSP_TRB_COMPLETE_SUCCESS	1U	TRB completion code success.
USBSSP_TRB_CMPL_DATA_BUFF_ER	2U	TRB completion code data buffer error.
USBSSP_TRB_CMPL_BBL_DETECT_ER	3U	TRB completion code babble detected error.
USBSSP_TRB_CMPL_USB_TRANSCN_ER	4U	TRB completion code USB transaction error.
USBSSP_TRB_COMPLETE_TRB_ERROR	5U	TRB completion code TRB error.
USBSSP_TRB_COMPLETE_STALL_ERROR	6U	TRB completion code stall error.
USBSSP_TRB_CMPL_RSRC_ER	7U	TRB completion code resource error.
USBSSP_TRB_CMPL_BDWTH_ER	8U	TRB completion code bandwidth error.
USBSSP_TRB_CMPL_NO_SLTS_AVL_ER	9U	TRB completion code no slots available error.
USBSSP_TRB_CMPL_INVSTRM_TYP_ER	10U	TRB completion code invalid stream type error.
USBSSP_TRB_CMPL_SLT_NOT_EN_ER	11U	TRB completion code slot not enabled error.
USBSSP_TRB_CMPL_EP_NOT_EN_ER	12U	TRB completion code endpoint not enabled error.
USBSSP_TRB_CMPL_SHORT_PKT	13U	TRB completion code short packet error.
USBSSP_TRB_CMPL_RING_UNDERRUN	14U	TRB completion code ring underrun error.
USBSSP_TRB_CMPL_RING_OVERRUN	15U	TRB completion code ring overrun error.
USBSSP_TRB_CMPL_VF_EVTRNGFL_ER	16U	TRB completion code VF event ring full error.

#define	value	description
USBSSP_TRB_CMPL_PARAMETER_ER	17U	TRB completion code parameter error.
USBSSP_TRB_CMPL_BDWT_OVRN_ER	18U	TRB completion code bandwidth overrun error.
USBSSP_TRB_CMPL_CXT_ST_ER	19U	TRB completion code context state error.
USBSSP_TRB_CMPL_NO_PNG_RSP_ER	20U	TRB completion code no ping response error.
USBSSP_TRB_CMPL_EVT_RNG_FL_ER	21U	TRB completion code event ring full error.
USBSSP_TRB_CMPL_INCMPT_DEV_ER	22U	TRB completion code incompatible device error.
USBSSP_TRB_CMPL_MISSED_SRV_ER	23U	TRB completion code missed service error.
USBSSP_TRB_CMPL_CMD_RNG_STOPPED	24U	TRB completion code command ring stopped error.
USBSSP_TRB_CMPL_CMD_ABORTED	25U	TRB completion code command aborted error.
USBSSP_TRB_COMPLETE_STOPPED	26U	TRB completion code stopped error.
USBSSP_TRB_CMPL_STOP_LEN_INV	27U	TRB completion code stopped - length invalid error.
USBSSP_TRB_CMPL_STOP_SHORT_PKT	28U	TRB completion code stopped - short packet error.
USBSSP_TRB_CMPL_MAXEXTLT_LG_ER	29U	TRB completion code max exit latency too large error.
USBSSP_TRB_CMPL_ISO_BUFF_OVRN	31U	TRB completion code isoch buffer overrun error.
USBSSP_TRB_CMPL_EVT_LOST_ER	32U	TRB completion code event lost error.
USBSSP_TRB_CMPL_UNDEFINED_ER	33U	TRB completion code undefined error.
USBSSP_TRB_CMPL_INV_STRM_ID_ER	34U	TRB completion code invalid stream ID error.
USBSSP_TRB_CMPL_SEC_BDWT_ER	35U	TRB completion code secondary bandwidth error.
USBSSP_TRB_CMPL_SPLT_TRNSCN_ER	36U	TRB completion code split transaction error.
USBSSP_TRB_CMPL_CDNSDEF_ERCODES	192U	TRB completion code vendor defined error.
USBSSP_TRB_SETUP_TRT_POS	16U	Position offset for TRT field in setup stage TRB.
USBSSP_TRB_SETUP_TRT_NO_DATA	(uint32_t)0x0U	NO_DATA in TRT field of setup stage TRB.

#define	value	description
USBSSP_TRB_SETUP_TRT_OUT_DATA	(uint32_t)0x2U	OUT_DATA in TRT field of setup stage TRB.
USBSSP_TRB_SETUP_TRT_IN_DATA	(uint32_t)0x3U	IN_DATA in TRT field of setup stage TRB.
USBSSP_TRB_ISOCH_FRAME_ID_POS	20U	Position offset for Frame ID bitfield in Isoch TRB.
USBSSP_TRB_ISOCH_FRAME_ID_MASK	(uint32_t)0x7FF00000U	Mask of Frame ID bitfield in Isoch TRB.
USBSSP_TRB_ISOCH_SIA_POS	31U	Position offset for SIA bitfield in Isoch TRB.
USBSSP_TRB_ISOCH_SIA_MASK	(uint32_t)0x80000000U	Mask for SIA bitfield in Isoch TRB.
USBSSP_TRB_TRANSFER_LENGTH_MASK	(uint32_t)0x1FFFFFFU	TRB transfer length mask.
USBSSP_COMPLETION_CODE_POS	24U	Position offset for completion code.
USBSSP_SLOT_ID_POS	24U	Position offset for Slot ID.
USBSSP_ENDPOINT_POS	16U	Position offset for endpoint.
USBSSP_INTERRUPTER_TARGET_POS	22U	Position offset for interrupter target.
USBSSP_TRANSFER_DIR_POS	16U	position offset for transfer direction
USBSSP_PORTSCUSB_PLS__RXDETECT	5U	Port Link state is RxDetect.
USBSSP_EP_CONTEXT_INTERVAL_POS	16U	Position offset for endpoint context interval.
USBSSP_EP_CXT_MAXESITPLD_HI_POS	24U	The MAX ESIT Payload represent the total number of bytes this endpoint will transfer during an ESIT.
USBSSP_EP_CXT_MAX_PKT_SZ_POS	16U	Position offset for max packet size in endpoint context data structure.
USBSSP_EP_CXT_MAX_BURST_SZ_POS	8U	Position offset for max burst size in endpoint context data structure.
USBSSP_EP_CXT_MAX_BURST_SZ_MASK	(uint32_t)0xFF00U	Mask for max burst size in endpoint context data structure.
USBSSP_EP_CONTEXT_MULT_POS	8U	The value of MULT is dependent on LEC bit
USBSSP_EP_CONTEXT_MULT_MASK	(uint32_t)0x300U	Mask for MULT bitfield in endpoint context data structure.
USBSSP_EP_CONTEXT_CERR_POS	1U	This bitfield identifies the number of consecutive USB Bus Errors allowed while executing a TD
USBSSP_EP_CONTEXT_CERR_MASK	(uint32_t)0x6U	Mask for CErr bitfield in endpoint context data structure.
USBSSP_EP_CXT_PMAXSTREAMS_POS	10U	Position offset for max primary stream IDs in endpoint context data structure.

#define	value	description
USBSSP_EP_CONTEXT_3ERR	3U	CErr bitfield in endpoint context data structure is set to '3' in normal operations.
USBSSP_EP_CXT_EP_DIR_IN	4U	Value to be set depending on endpoint direction.
USBSSP_EP_CXT_EP_DIR_OUT	0U	Value to be set depending on endpoint direction.
USBSSP_EP_CXT_EP_CTL_AVGTRB_LEN	8U	The xHCI Spec says that for a control ep the average trb length must be set by SW to 8.
USBSSP_EP_CXT_EP_INT_AVGTRB_LEN	1024U	The xHCI Spec says that for a int ep the average trb length must be set by SW to 1024.
USBSSP_EP_CXT_EP_ISO_AVGTRB_LEN	3072U	The xHCI Spec says that for a ISOC ep the average trb length must be set by SW to 3072.
USBSSP_EP_CXT_EP_BLK_AVGTRB_LEN	3072U	The xHCI Spec says that for a BULK ep the average trb length must be set by SW to 3072.
USBSSP_EP_CXT_EP_AVGTRBLEN_POS	0U	Position offset for average TRB length.
USBSSP_EP_CXT_MAXESITPLD_LO_POS	16U	The MAX ESIT Payload represent the total number of bytes this endpoint will transfer during an ESIT.
USBSSP_EP_CONTEXT_EP_TYPE_POS	3U	Position offset for endpoint type.
USBSSP_EP_CONTEXT_EP_TYPE_MASK	(uint32_t)0x38U	Mask for endpoint type.
USBSSP_EP_CONTEXT_STATE_MASK	7U	Mask for endpoint context state.
USBSSP_EP0_CONTEXT_OFFSET	1U	Endpoint 0 context offset.
USBSSP_SLOT_CXT_CXT_ENT_POS	27U	Position offset for the index of the last valid endpoint context.
USBSSP_SLOT_CONTEXT_SPEED_POS	20U	Position offset for speed of the device.
USBSSP_SLOT_CXT_NUM_PORTS_POS	24U	Position offset for the numver of ports.
USBSSP_SLOT_CXT_PORT_NUM_POS	16U	Position offset for Root Hub Port Number.
USBSSP_SLOT_CONTEXT_STATE_POS	27U	This field is updated when a device slot transitions from one state to another

Chapter 3. Data Structures

3.1. Introduction

This section defines the data structures used by the driver to provide hardware information, modification and dynamic operation of the driver.

These data structures are defined in the header file of the core driver and utilized by the API.

3.2. USBSSP_RingElementT_s

Type: struct

Description

Transfer request block structure, spec 4.11.1.

Fields

dword0 TRB parameter MSB.

dword1 TRB parameter LSB.

dword2 TRB status.

dword3 TRB control.

3.3. USBSSP_InputContextT_s

Type: struct

Description

Input context structure.

Fields

inputControlContext Input control context structure.

slot Slot context structure.

ep0Context Endpoint 0 context structure.

epContext Endpoints context structures.

3.4. USBSSP_OutputContextT_s

Type: struct

Description

Output context structure.

Fields

slot Slot context structure.

ep0Context Endpoint 0 context structure.

epContext Endpoints context structures.

3.5. USBSSP_XferDescT_s

Type: struct

Description

Transfer descriptor structure.

Fields

complete Pointer to the complete callback.

startTRBPhyAddr Phy address of the first TRB.

nextTRBPhyAddr Phy address for next empty TRB.

3.6. USBSSP_ProducerQueueT_s

Type: struct

Description

Structure describes element of producer queue in conjunction to associated endpoint.

Note that the same structure is used in command ring - in this case, fields referred to endpoint are not used

Fields

ring Memory buffer for ring used by hardware.

enqueuePtr Pointer to enqueued element.

dequeuePtr Pointer to dequeued element.

completePtr Pointer to last completed queue element.

firstQueuedTRB used for testing purposes

lastQueuedTRB last queued TRB

frameID frame ID

stream Streams container.

hwContext	Points to hardware endpoint context according to spec 6.2.3.
parent	Points to this object owner.
xferDesc	Transfer descriptor array.
complete	Callback function called on TRB complete event.
aggregatedComplete	Callback function called on an aggregated transfer completion.
actualSID	Keeps actually selected stream ID.
toggleBit	Keeps actual value of Cycle bit inserted to TRB.
contextIndex	Keeps context entry value of endpoint associated with this object.
isRunningFlag	Auxiliary flag, set to 1 when any TD associated with this object is issued to DMA and flag is set to 0 on complete event.
isDisabledFlag	Flag is active when endpoint is in stopped state.
xferStallError	Flag is indicates that last transfer stalled.
epDesc	Keeps copy of endpoint descriptor of associated endpoint.
completionCode	Completion code of last transfer.
interrupterIdx	Interrupter Index of the target interrupter.
extraFlags	extra flags
ignoreShortPacket	Blocks calling complete callback when set to 1.
xferDescReadIdx	Transfer Descriptor read index.
xferDescWriteIdx	Transfer Descriptor write index.
reserved_0	reserved

3.7. USBSSP_DescT_s

Type: struct

Description

structure keeps all descriptors pointers for USBSSP operating in device mode index of array correspond to USB speed: devDesc[1] - low speed, devDesc[2] - full speed arrays elements indexed 0 should be set to NULL

Fields

devDesc	Table keeps pointers to device descriptors.
confDesc	Table keeps pointers to configuration descriptors.
bosDesc	Table keeps pointers to BOS descriptors.

`string` Table keeps pointers to string descriptors.

3.8. USBSSP_DcbaaT_s

Type: struct

Description

Device context base array pointer structure.

Fields

`scratchPadPtr` Address of scratch pad pointers container.

`deviceSlot` Address of device slot 1.

3.9. USBSSP_XhciResourcesT_s

Type: struct

Description

Structure represents USB SSP memory required by XHCI specification.

That structure has to be provided by application and must be aligned to `USBSSP_PAGE_SIZE`.

Fields

`epRingPool` EP transfer ring memory pointer.

`eventPool` Event Ring.

`dcbaa` Device context base array structure.

`inputContext` Input context structure.

`outputContext` Output context structure.

`scratchpad` Scratchpad buffers (extra element for last pointer = NULL)

`eventRingSegmentEntry` event ring segment entry

`streamMemoryPool` allocated memory for stream objects

`streamRing` allocation memory for stream's rings

`scratchpadPool` Scratchpad buffers pool.

`ep0Buffer` Pointer to EP0 buffer of size `USBSSP_EP0_DATA_BUFF_SIZE`.

3.10. USBSSP_IpldVerRegs_s

Type: struct

Description

Structure that holds addresses to RID and DID registers.

Fields

`didRegPtr` Pointer to DID register.

`ridRegPtr` Pointer to RID register.

3.11. USBSSP_DriverResourcesT_s

Type: struct

Description

Structure represents USB SSP controller resources.

That structure must be aligned to `USBSSP_PAGE_SIZE` because of `xhciResources`.

Fields

<code>xhciMemRes</code>	Structure represents USB SSP memory required by XHCI specification.
<code>inputContext</code>	Input context structure.
<code>inputContextCopy</code>	Input context copy structure.
<code>eventPtrBuffer</code>	Pointers to actual event ring element for all interrupters.
<code>eventPtr</code>	Pointer to actual event ring element.
<code>commandQ</code>	Command queue object.
<code>ep0</code>	Default endpoint queue object.
<code>ep</code>	Container of non default endpoint objects.
<code>actualSpeed</code>	Keeps actual speed the port operate in.
<code>ep0Buff</code>	Internal buffer of size <code>USBSSP_EP0_DATA_BUFF_SIZE</code> , for control transfer.
<code>devDesc</code>	Pointer to device descriptors container.
<code>contextEntries</code>	Stores actual value of context Entry.
<code>qaRegs</code>	Local copy of some registers - for quick access.
<code>regs</code>	Keeps addresses of all SFR's.
<code>nopComplete</code>	NOP complete callback function, diagnostic function.
<code>forceHeaderComplete</code>	NOP complete callback function, diagnostic function.

Data Structures

enabledEndpsMask	Mask with enabled endpoints for last SET_CONFIGURATION request.
maxDeviceSlot	The maximum number of Device Context Structures and Doorbell Array entries this controller can support.
ep0State	flag active when setup packet received and enable_slot command isn't completed yet, it delegates setup handling to enable slot command completion
ipIdVerRegs	Pointers to IP RID and DID registers.
portOverrideRegs	Pointer to port override register.
genericCallback	Used for testing purposes.
postCallback	Used for testing purposes.
preportChangeDetectCallback	Used for testing purposes.
inputContextCallback	Used for testing purposes.
usb2PhySoftReset	Callback function to perform USB 2.0 PHY soft reset.
usb2ResetCallback	Reset USB-2 interface.
eventToggleBit	Keeps actual value of cycle bit for event ring.
epInterrupterIdx	InterrupterIdx corresponding to each endpoint.
actualPort	Keeps port ID.
actualdeviceSlot	Keeps actual device slot, when USBSSP works in device mode it is 1.
enableSlotPending	Indicates whether ENABLE_SLOT_COMMAND was sent but slot ID has not been set yet.
devConfigFlag	Flag is active when USB SSP is in configured state.
deviceModeFlag	1 - USB SSP works in device mode, 0 - host mode
usbModeFlag	Indicates USB mode (2 - forced USB2 mode, 3 - forced USB3 mode, others - default)
noControllerSetup	When set to 1 indicates initializing will be performed only on connected device, not on SSP controller itself.
extendedTBCMode	Extended TBC enable mode if ETC enabled (0: ETE always disabled, 1:ETE enabled for all speeds)
connected	Reflects actual state of Current Connect Status of PORTSC register, 0 - disconnected, 1 - connected.
devAddress	Keep device address sent during SET_ADDRESS setup request - is active only in device mode.

setupID	Keeps setupId value of actually handled setup packet.
lastEpIntIndex	Keeps index of non zero endpoint which generated latest interrupt.
controlXferEpIndex	Endpoint index corresponding to the last set/clear feature request.
instanceNo	Keeps index of instance associated with this resource.

3.12. USBSSP_DriverContextT_s

Type: struct

Description

This Structure holds the saved USB context, which can be used for restore.

Fields

usbcmd	USB Command Register.
dnctrl	Device Notification Control Register.
dcbaap	Device Context Base Address Array Pointer Register.
config	Configure Register.
iman	Interrupter Management.
imod	Interrupter Moderation.
erstsz	Event Ring Segment Table Size.
erstba	Event Ring Segment Table Base Address.
erdp	Event Ring Dequeue Pointer.

3.13. USBSSP_ForceHdrParams_s

Type: struct

Description

Force header params structure.

Fields

dword0	Header info low and type.
dword1	Header info mid.
dword2	Header info high.
dword3	Root hub port number and TRB type.

3.14. USBSSP_XferBufferDesc_s

Type: struct

Description

Transfer buffer descriptor structure.

Fields

buffVec buffer vector

sizeVec size vector

3.15. CUSBD_EpAuxBufferConfig_s

Type: struct

Description

Configuration of Auxiliary-overflow buffers for EP-OUT endpoints.

This structure is software related.

Fields

bufferAddr Virtual address of auxiliary buffer for this endpoint.

bufferSize Size of the auxiliary buffer in 32-bit-words.

3.16. CUSBD_EpAuxBuffer_s

Type: struct

Description

Auxiliary-overflow buffer structure for EP-OUT endpoints.

Fields

bufferAddr Virtual address of auxiliary buffer for this endpoint.

bufferSize Size of the auxiliary buffer in 32-bit-words.

readIdx Read index.

writeIdx write index

updateIdx update index

maxPacketSize Max packet size for this endpoint.

3.17. CUSBD_EpConfig_s

Type: struct

Description

Configuration structure for single endpoint.

Single endpoint configuration structure object is a part of CUSBD_Config object used for configuration of whole USB device

Fields

bufferingValue	Size of hardware (on-chip) memory buffer that is assigned to an endpoint for queuing USB packets. Setting bufferingValue to 0 means that endpoint is disabled. Setting bufferingValue to value larger then 0 will assign hardware (on-chip) buffering memory to an endpoint. Please refer SuperSpeed USB 3.0 Device Controller IP Design Specification section 2.9.6 for details about endpoint buffering.
supportStream	Bulk endpoint support streams, used only in Super Speed mode.

3.18. CUSBD_Config_s

Type: struct

Description

Configuration of device.

Object of this type is used for probe and init functions when checking memory requirements of device object and for hardware controller configuration. Size of device object in bytes is returned on probe function call in CUSBD_SysReq.privDataSize field. Remember that privDataSize returns size of device object only - it doesn't endpoints vector object size. It means that whole memory requirements = device size + (number of endpoints x single endpoint size)

Fields

regBase	base address of device controller registers
epIN	table of endpoint configuration objects for IN direction
epOUT	table of endpoint configuration objects for OUT direction
epOutAuxBufferCfg	array of endpoint aux buffer configuration objects for OUT direction
extendedTBCMode	Extended TBC enable mode if ETC enabled (0: ETE always disabled, 1:ETE enabled for all speeds)
dmultEnabled	hardware feature enabling DMA operation in DMULT mode 1 - enabled, 0 disabled For Device V1: A non-zero value of dmultEnabled enables DMULT mode for all channels For Device V3: Bits[15:0]: Control DMULT enable for EP-Out[15:0] Bits[31:16]: Control DMULT enable for EP-In[15:0]

<code>dmaInterfaceWidth</code>	DMA interface width, available values: <code>DMA_32_WIDTH</code> , <code>DMA_64_WIDTH</code> .
<code>preciseBurstLength</code>	Set configurable burst length in DMA, available values: <code>PRECISE_BURST_0</code> , <code>PRECISE_BURST_1</code> , <code>PRECISE_BURST_2</code> , <code>PRECISE_BURST_4</code> , <code>PRECISE_BURST_8</code> , <code>PRECISE_BURST_16</code> , <code>PRECISE_BURST_32</code> , <code>PRECISE_BURST_64</code> , <code>PRECISE_BURST_128</code> .
<code>epMemRes</code>	Keeps addresses of resources of all endpoints.
<code>forcedUsbMode</code>	Forced USB mode (0 - none (default), 2 - USB2/HS, 3 - USB3/SS, other - reserved)
<code>didRegPtr</code>	Pointer to DID register.
<code>ridRegPtr</code>	Pointer to RID register.
<code>setupPacket</code>	Pointer (virtual) to the setup packet.
<code>setupPacketDma</code>	Physical address of setup packet for DMA.

3.19. CUSBD_SysReq_s

Type: struct

Description

System requirements returned by probe.

Fields

<code>privDataSize</code>	size of memory required for driver's private data
<code>epObjSize</code>	size of memory required for endpoint object
<code>trbMemSize</code>	size of memory required for USB Transfer Request Descriptors.

This memory will be used by DMA controller, so it should be suitable for DMA operation.

3.20. CUSBD_SgList_s

Type: struct

Description

structure contains information about memory slices for scatter/gather transfer

Fields

<code>link</code>	link to the next element
<code>offset</code>	offset within memory page
<code>length</code>	data length of this transfer slice

`dmaAddress` physical address of buffer to read/write

3.21. CUSBD_Req_s

Type: struct

Description

Transfer request structure.

I/O transfers of higher layers are realized with the help of CUSBD_Req objects. CUSBD_Req is associated with endpoints.

Fields

<code>list</code>	Head of the list.
<code>prevReq</code>	pointer to the previous request in the linked list
<code>nextReq</code>	pointer to the next request in the linked list
<code>buf</code>	buffer with data to transfer - virtual address
<code>length</code>	This transfer data length.
<code>dma</code>	physical address of buf buffer
<code>streamId</code>	stream id - used only in Super Speed mode
<code>noInterrupt</code>	Setting this flag to 1 disables generating of transfer completion interrupt.
<code>zero</code>	Setting this flag to 1 causes adding zero length data packet at the end of data transfer when data transfer length is a multiple of MaxPacketSize for endpoint.
<code>shortNotOk</code>	Setting this flag causes all reading transfers with short packet as erroneous.
<code>requestPending</code>	Internal flag used by driver to mark request as pending.
<code>complete</code>	Callback function called on the transfer completion event.
<code>context</code>	general-use data hook driver doesn't make use of this field. May be used in complete callback function
<code>status</code>	keeps actual status of request
<code>actual</code>	number of actually processed bytes

3.22. CUSBD_EpOps_s

Type: struct

Description

Set of functions allowing to operate on endpoints.

Fields

<code>epEnable</code>	Function for endpoint enable.
<code>epDisable</code>	Function for endpoint disable.
<code>epSetHalt</code>	Function for endpoint sethalt.
<code>epSetWedge</code>	Function for endpoint setwedge.
<code>epFifoStatus</code>	Function for endpoint FIFO status.
<code>epFifoFlush</code>	Function for endpoint FIFO flush.
<code>reqQueue</code>	Function for reqQueue.
<code>reqDequeue</code>	Function for reqDequeue.

3.23. CUSBD_Ep_s

Type: struct

Description

Structure represents device USB endpoint.

Fields

<code>epList</code>	enables organization of endpoints in linked list
<code>name</code>	keeps name of endpoint
<code>epPrivate</code>	pointer to the private data of this endpoint
<code>ops</code>	endpoint driver, contains methods for all operation required on endpoint
<code>maxPacket</code>	Maximal packet size for endpoint.
<code>maxStreams</code>	Maximal number of streams used by endpoint, useful only in SS mode.
<code>mult</code>	packet multiplication, it enables to multiply number of packet per service interval, useful in HS and SS mode only for isochronous endpoint, mult value range from 0 to 2 - it multiplies packets by 1 to 3
<code>maxburst</code>	Maximal number of packets in burst transfer, useful only in SS mode. Maxburst value is in range from 0 to 15 giving number of packets 1 to 16. Refers to USB transfer burst.
<code>address</code>	Endpoint address, the oldest bit refers to endpoint direction. For example: address = 0x82 refers to endpoint 2 IN address = 0x03 refers to endpoint 3 OUT
<code>desc</code>	pointer to user defined endpoint descriptor, <code>CH9_UsbEndpointDescriptor</code> is defined in <code>usb_ch9_if.h</code>

`compDesc` pointer to user defined endpoint companion descriptor, useful only in SS mode, `CH9_UsbSSEndpointCompanionDescriptor` is defined in `cusb_ch9_if.h`

3.24. CUSBD_Dev_s

Type: struct

Description

Structure represents USB device.

Fields

<code>epList</code>	allows for manipulation on endpoints organized as linked list
<code>ep0</code>	pointer to bidirectional default endpoint, this endpoint should be always available after driver start
<code>speed</code>	speed value in which device actually works
<code>maxSpeed</code>	maximal speed the device is able to work
<code>state</code>	actual state in which device actually works
<code>sgSupported</code>	Flags informs if device is scatter/gather transfer capable.
<code>name</code>	Full name of USB device controller.

3.25. CUSBD_Callbacks_s

Type: struct

Description

struct containing function pointers for event notification callbacks issued by `isr()`.

Each call passes the driver's `privateData` pointer for instance identification if necessary, and may also pass data related to the event.

Fields

<code>disconnect</code>	callback for disconnect
<code>connect</code>	callback for connect
<code>setup</code>	callback for setup
<code>suspend</code>	callback for suspend
<code>resume</code>	callback for resume
<code>busInterval</code>	callback for businterval
<code>descMissing</code>	callback for missing descriptor

usb2PhySoftReset callback for USB to PHY soft reset

3.26. CUSBD_EpPrivate_s

Type: struct

Fields

reqList	Request list.
ep	Endpoint.
transferPending	Flag for pending transfer.
reqListHead	Pointer for request list head.
actualReq	Pointer for actual request.
channel	Pointer to DMA channel.
ep_state	Endpoint state.
wedgeFlag	Wedge flag.

3.27. CUSBD_PrivateData_s

Type: struct

Description

CUSBD private data.

Fields

device	this 'class must implement' CUSBD class, this class is an extension of CUSBD
config	copy of user configuration
callbacks	copy of user callback
u2_value	these variables reflect device state
	keeps U2 value
u1_value	keeps U1 value
suspend	keeps suspend value
isoch_delay	Iso delay.
deviceVersion	HW version of this device.
ep0	bidirectional endpoint 0

ep0NextState	Endpoint 0 state.
ep0DataDirFlag	Flag for endpoint 0 data direction.
ep_in_container	Endpoint IN container.
ep_out_container	Endpoint OUT container.
epOutAuxBuffer	Endpoint auxilliary buffer for OUT direction.
setupPacket	Virtual address of setup packet.
setupPacketDma	Physical address of setup packet.
status_value	status value
reg	Pointer to the controller registers.
dmaDrv	Driver object.
dmaController	DMA controller.
ep0DmaChannelIn	DMA Channel for device to host.
ep0DmaChannelOut	DMA channel for host to device.
request	Pointer to request object.
reqAlloc	Allocated request object.

3.28. CUSBDMA_DmaTrb_s

Type: struct

Description

Describes One trb.

Fields

dmaAddr	DMA address.
dmaSize	DMA size.
ctrl	DMA control.

3.29. CUSBDMA_MemResources_s

Type: struct

Description

Structure keeps endpoints resources pointers.

Fields

trbAddr	Virtual address of transfer ring base.
trbBufferSize	Size of the TRB buffer.
trbDmaAddr	Physical address of transfer ring.
memPageIndex	Physical address extension to 48 bits, this variable is a 16 bit width page index.

3.30. CUSBDMA_Config_s

Type: struct

Description

Configuration of DMA.

Object of this type is used for probe and init functions when checking memory requirements of device object and for hardware controller configuration. Size of device object in bytes is returned on probe function call in CUSBDMA_SysReq.privDataSize field.

Fields

regBase Base address of DMA special function register.

dmaModeTx Field holds information about DMA mode.

DMA mode can be programmed globally for all channels or individually for particular channels. Each bit of dmaModeTx field is responsible for one DMA channel. If this field will be set to 0xFFFF or 0xFFFF for both dmaModeTx and dmaModeRx then DMA mode will be programmed globally, elsewhere each DMA channel will be programmed individually in channelAlloc function. Meanings of bits: 0 DMA Single Mode, 1 DMA DMULT Mode.

dmaModeRx Field holds information about DMA mode.

DMA mode can be programmed globally for all channels or individually for particular channels. Each bit of dmaModeTx field is responsible for one DMA channel. If this field will be set to 0xFFFF or 0xFFFF for both dmaModeTx and dmaModeRx then DMA mode will be programmed globally, elsewhere each DMA channel will be programmed individually in channelAlloc function. Meanings of bits: 0 DMA Single Modem, 1 DMA DMULT Mode.

epMemRes Pointer to array containing endpoints resources addresses.

3.31. CUSBDMA_SysReq_s

Type: struct

Description

System requirements returned by probe.

Fields

`privDataSize` Size of memory required for driver's private data.

`trbMemSize` Size of memory required for USB Transfer Request Descriptors.

This memory will be used by DMA controller, so it should be suitable for DMA operation.

3.32. CUSB_DMA_ChannelParams_s

Type: struct

Description

Channel parameters used during channel allocation.

Fields

`isDirTx` direction for allocated channel

`hwEpNum` USB endpoint number for allocated channel.

`wMaxPacketSize` Max packet size for this channel.

`epConfig` endpoint configuration for this channel - EP_CFG

`epIrqConfig` IRQ settings for this channel - EP_STS_EN.

3.33. CUSB_DMA_DmaTrbChainDesc_s

Type: struct

Description

Structure describing TRB Chain corresponding to a request.

Fields

`len` Length.

`actualLen` Actual length.

`start` index of oldest element

`end` Index of the newest element.

`shortPacketFlag` Indicates that a short packet was received for this chain.

`trbChainState` Indicates the state of this TRB descriptor chain.

`requestQueued` Indicates whether the request for the transfer has been queued by cusbd.

`requestOverflowed` Indicates whether the buffer for the transfer is held in aux-buffer.

3.34. CUSBDMA_DmaChannel_s

Type: struct

Description

DMA Channel structure.

Fields

controller	Pointer to DMA controller.
wMaxPacketSize	Max packet size for this channel.
hwUsbEppNum	USB EP number. It can be different then channel number
isDirTx	Flag indicating if transfer direction is transmit.
maxTdLen	max TD length
maxTrbLen	max TRB length
status	status
priv	for private use
dmultEnabled	field holds dmult configuration for channel
channelStalled	Flag indicated whether this channel is stalled.
trbChainDescListHead	Pointer to the TRB chain desc list head for this channel.
epConfig	EP_CFG register for this channel.
currentStreamID	Stream ID of the current stream.
trbCycleBit	Cycle bit for the next write TRB.
trbBufferBaseAddr	Virtual address of transfer ring base.
trbBufferBaseAddrPhy	Physical address of transfer ring base.
trbBufferSize	Size of the TRB buffer.
trbBufferEnqueueIdx	Index where the next TRB will be written by SW.
trbBufferDequeueIdx	Last known Index where the next TRB would be read by HW.
trbBufferDmultDelinkIdx	Index where the TRB chain is intentionally De-linked in DMULT mode.
trbChainDescBufferSz	Size of the TRB chain descriptor buffer size.
trbChainDescWriteIdx	Index where the next TRB would be written by SW.

trbChainDescReadIdx Index where the next TRB would be read by HW.

3.35. CUSB_DMA_DmaController_s

Type: struct

Description

DMA controller structure.

Fields

regs	Pointer to the Device registers.
cfg	DMA configuration.
rx	DMA Receive channels.
tx	DMA transmit channels.
trbChainDesc	DMA TRB chain descriptor.
epMemRes	Pointer to array containing endpoints resources addresses.

3.36. CUSB_DMA_DmaTransferParam_s

Type: struct

Description

DMA Transfer Param structure.

Fields

dmaAddr	DMA address.
len	length
sid	streamID
requestQueued	Indicated whether the request for the transfer has been queued by cusbd.
requestOverflowed	Indicates whether the buffer for the transfer is held in aux-buffer.

3.37. LIST_ListHead_s

Type: struct

Fields

next	pointer to next element in list
prev	pointer to previous element in list

3.38. USB_SSP_DRD_Config_s

Type: struct

Description

Configuration parameters passed to init function.

Fields

regsBase	Base address of the register space.
cusbdPrivateData	USB Dev private data.
xhciPrivateData	USB Host private data.

3.39. USB_SSP_DRD_Callbacks_s

Type: struct

Description

Structure with callbacks to userspace.

Fields

interruptDrd	This function will be called when things are done.
enableHost	This callback is for enabling HOST driver.
enableAHost	This callback is for enabling HOST driver in A_HOST state.
enableDev	This callback is for enabling DEV driver.
enableBIdleDev	This callback is for enabling DEV driver in B_IDLE mode.
enableBPeriDev	This callback is for enabling DEV driver in B_PERIPHERAL mode.
interruptHost	This callback is for handling HOST interrupt.
interruptDev	This callback is for handling DEV interrupt.
disableHost	This callback is for disabling HOST driver.
disableDev	This callback is for disabling DEV driver.

3.40. USB_SSP_DRD_State_s

Type: struct

Description

This structure represents state of USBSSP DRD FSM.

Each member corresponds to a event that can occur during FSM's operation. Upon entering the state, members of this structure are set based on events supported by this state.

Fields

<code>isr</code>	Interrupt handler specific for current state.
<code>idUp</code>	Transition function for an event of ID pin level change to high.
<code>idUpVBusDown</code>	Transition function for an event of ID pin level change to high and VBUS level change to low.
<code>idUpVBusUp</code>	Transition function for an event of ID pin level change to high and VBUS level change to high.
<code>idDownVBusDown</code>	Transition function for an event of ID pin level change to low and VBUS level change to low.
<code>idDownVBusUp</code>	Transition function for an event of ID pin level change to low and VBUS level change to high.
<code>enableDrd</code>	Transition function for an event of forcing DRD mode.
<code>enableHost</code>	Transition function for an event of forcing host mode.
<code>enableDev</code>	Transition function for an event of forcing device mode.

3.41. USB_SSP_DRD_PrivData_s

Type: struct

Description

Driver's Private Data.

Fields

<code>regsBase</code>	Base address of the register space.
<code>callbacks</code>	Callback handlers.
<code>state</code>	State of DRD FSM.
<code>cusbdPrivateData</code>	USB Dev private data.
<code>xhciPrivateData</code>	USB Host private data.
<code>interruptVect</code>	Interrupt vector value of USB SSP DRD.
<code>defaultTransitionCalls</code>	Number of times the default transition of FSM was called.

3.42. USB_SSP_DRD_SysReq_s

Type: struct

Description

System requirements returned by probe.

Fields

`memReq` Size of memory required for driver's private data.

3.43. USBSSP_CapabilityT_s

Type: struct

Description

capability register structure

Fields

`caplength_hciver` Capability Register Length and Interface Version Number.

`hcsparams1` Structural Parameters 1.

`hcsparams2` Structural Parameters 2.

`hcsparams3` Structural Parameters 3.

`hccparams1` Capability Parameters 1.

`dboff` Doorbell Offset 1.

`rtsoff` Runtime Registers Space Offset.

`hccparams2` Capability Parameters 2.

3.44. USBSSP_QuickAccessRegs_s

Type: struct

Description

Quickaccess register structure.

Fields

`xHCCaps` Copy of xhci capability register for quick access.

3.45. USBSSP_PortControlT_s

Type: struct

Description

Port control register structure.

Fields

<code>portsc</code>	Port Status and Control.
<code>portpmc</code>	Port Power Management Status and Control.
<code>portli</code>	Port Link Info.
<code>portlpmc</code>	Port Hardware LPM Control Register.

3.46. USBSSP_OperationalT_s**Type:** struct**Description**

operational register structure

Fields

<code>usbcmd</code>	USB Command.
<code>usbsts</code>	USB Status.
<code>pagesize</code>	Page Size.
<code>reserved_0</code>	reserved
<code>dnctrl</code>	Device Notification Control.
<code>crcr</code>	Command Ring Control.
<code>reserved_1</code>	reserved
<code>dcbaap</code>	Device Context Base Address Array Pointer.
<code>config</code>	Configure.
<code>reserved_2</code>	reserved
<code>portControl</code>	Port Control Registers.

3.47. USBSSP_InterrupterT_s**Type:** struct**Description**

interrupter register structure

Fields

<code>iman</code>	Interrupter Management.
-------------------	-------------------------

<code>imod</code>	Interrupter Moderation.
<code>erstsz</code>	Event Ring Segment Table Size.
<code>reserved</code>	Reserved.
<code>erstba</code>	Event Ring Segment Table Base Address.
<code>erdp</code>	Event Ring Dequeue Pointer.

3.48. USBSSP_RuntimeT_s

Type: struct

Description

runtime register structure

Fields

<code>mfindex</code>	Microframe Index.
<code>reserved</code>	Reserved.
<code>interrupters</code>	Interrupter Register Sets.

3.49. USBSSP_ExtCapElemT_s

Type: struct

Description

Structure that describes single Extended Capability.

Fields

<code>firstDwordVal</code>	Value of first 32bit word for this Ext. Cap. (DWORD[0])
<code>capId</code>	Capability ID (DWORD[0].CapabilityID)
<code>firstCapSfrPtr</code>	Pointer to first SFR belonging to this capability.

3.50. USBSSP_ExtCapSetT_s

Type: struct

Description

extCapSet structure

Fields

<code>extCapsBaseAddr</code>	Address of first Extended Capabilities' SFR (XEC_USBLEGSUP)
<code>usbLegSup</code>	Contents of USBLEGSUP SFR.
<code>usbLegCtlSts</code>	Contents of USBLEGCTLSTS SFR.
<code>extCaps</code>	Array with Extended Capabilities.
<code>extCapsCount</code>	Number of Extended Capabilities recognized.

3.51. USBSSP_SfrT_s

Type: struct

Description

sfr structure

Fields

<code>xhciCapability</code>	XHCI capability register.
<code>xhciOperational</code>	XHCI operational register.
<code>xhciPortControl</code>	XHCI port control register.
<code>xhciRuntime</code>	XHCI runtime register.
<code>xhciInterrupter</code>	XHCI interrupter register.
<code>xhciDoorbell</code>	XHCI doorbell register.
<code>xhciExtCaps</code>	xHCI capabilities are not handled as ordinary SFRs

3.52. USBSSP_Ep0StateEnum

Type: enum

Description

Endpoint 0 states.

Values

<code>USBSSP_EP0_UNCONNECTED</code>	Endpoint 0 not connected. Value: = 0U
<code>USBSSP_EP0_SETUP_PHASE</code>	Endpoint 0 setup phase.

Value: = 1U

USBSSP_EP0_DATA_PHASE Endpoint 0 data phase.

Value: = 2U

USBSSP_EP0_STATUS_PHASE Endpoint 0 status phase.

Value: = 3U

3.53. USBSSP_ExtraFlagsEnumT

Type: enum

Description

Extra flags types.

Values

USBSSP_EXTRAFLAGSENUMT_UNDEFINED

Extraflags undefined.

Value: = 0U

USBSSP_EXTRAFLAGSENUMT_NODORBELL

Extraflags no doorbell.

Value: = 1U

USBSSP_EXTRAFLAGSENUMT_FORCELINKTRB

Extraflags forcmlinkTRB.

Value: = 2U

3.54. CUSBD_DMAInterfaceWidth

Type: enum

Description

DMA BUS width available values.

Values

CUSBD_DMA_64_WIDTH DMA bus width 64.

Value: = 8U

3.55. CUSBD_epState

Type: enum

Description

EndPoint state available values.

Values

CUSBD_EP_DISABLED	Endpoint disabled.
	Value: = 0U
CUSBD_EP_ENABLED	Endpoint enabled.
	Value: = 1U
CUSBD_EP_STALLED	Endpoint stalled.
	Value: = 2U

3.56. CUSBDMA_Status

Type: enum

Description

DMA Channel Status available values.

Values

CUSBDMA_STATUS_UNKNOW	channel unallocated
	Value: = 0U
CUSBDMA_STATUS_FREE	channel allocated but not busy, no errors
	Value: = 1U
CUSBDMA_STATUS_STALLED	DMA channel is stalled.
	Value: = 2U
CUSBDMA_STATUS_BUSY	channel busy - transfer in progress
	Value: = 3U
CUSBDMA_STATUS_ARMED	DMA has been armed and Doorbell or DMA BUSY bit is still set.
	If those bits are set it indicate that DMA is armed and data has not been transfered yet

Value: = 4U

3.57. USB_SSP_DRD_Mode

Type: enum

Description

Current mode of controller.

Regards hardware strap mode and software enabled operation

Values

USB_SSP_DRD_DRD	Controller strapped or works in DRD. Value: = 0U
USB_SSP_DRD_HOST	Controller strapped or works as HOST. Value: = 1U
USB_SSP_DRD_DEV	Controller strapped or works as DEVICE. Value: = 2U
USB_SSP_DRD_ILLEGAL	Controller strap error. Value: = 3U

3.58. USBSSP_EpContextEpTypeT

Type: enum

Description

Endpoint types.

Values

USBSSP_EP_CXT_EPTYP_ISO_OUT	Endpoint ISO Out. Value: = 1U
USBSSP_EP_CXT_EPTYP_BLK_OUT	Endpoint Bulk Out. Value: = 2U
USBSSP_EP_CXT_EPTYP_INT_OUT	

Endpoint INT Out.

Value: = 3U

USBSSP_EP_CXT_EPTYP_CTL_BI

Endpoint control bidirectional.

Value: = 4U

USBSSP_EP_CXT_EPTYP_ISO_IN

Endpoint ISO In.

Value: = 5U

USBSSP_EP_CXT_EPTYP_BLK_IN

Endpoint Bulk In.

Value: = 6U

USBSSP_EP_CXT_EPTYP_INT_IN

Endpoint Int In.

Value: = 7U

3.59. USBSSP_EpContexEpState

Type: enum

Description

Endpoint states.

Values

USBSSP_EP_CONTEXT_EP_STATE_DISABLED

Endpoint state disabled.

Value: = 0U

USBSSP_EP_CONTEXT_EP_STATE_RUNNING

Endpoint state running.

Value: = 1U

USBSSP_EP_CONTEXT_EP_STATE_HALTED

Endpoint state halted.

Value: = 2U

USBSSP_EP_CONTEXT_EP_STATE_STOPPED

Endpoint state stopped.

Value: = 3U

USBSSP_EP_CONTEXT_EP_STATE_ERROR

Endpoint state error.

Value: = 4U

3.60. USBSSP_SlotContextState

Type: enum

Description

Slot context states.

Values

USBSSP_SLOT_CONTEXT_STATE_DISABLED_ENABLED

Slot state disabled/enabled.

Value: = 0U

USBSSP_SLOT_CONTEXT_STATE_DEFAULT

Slot state default.

Value: = 1U

USBSSP_SLOT_CONTEXT_STATE_ADDRESSED

Slot state addressed.

Value: = 2U

USBSSP_SLOT_CONTEXT_STATE_CONFIGURED

Slot state configured.

Value: = 3U

USBSSP_SLOT_CONTEXT_STATE_RESERVED

Slot state reserved.

Value: = 4U

3.61. USBSSP_PortControlRegIdx

Type: enum

Description

Port Control Register ID.

Values

USBSSP_PORTSC_REG_IDX	Port status change register. Value: = 0U
USBSSP_PORTPMSC_REG_IDX	Port Power Management Status and Control register. Value: = 1U
USBSSP_PORTLI_REG_IDX	Port Link Info register. Value: = 2U
USBSSP_PORHLPMC_REG_IDX	Port hardware LPM control register. Value: = 3U

3.62. USBSSP_DmaAddrT

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t USBSSP_DmaAddrT
```

3.63. USBSSP_Complete

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* USBSSP_Complete )(USBSSP_DriverResourcesT *arg, uint32_t status, const
USBSSP_RingElementT *eventPtr)
```

Description

Completion callback.

3.64. USBSSP_NopComplete

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* USBSSP_NopComplete )(USBSSP_DriverResourcesT *arg, uint32_t status, const
USBSSP_RingElementT *eventPtr)
```

Description

No-Op completion callback.

3.65. USBSSP_ForceHeaderComplete

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* USBSSP_ForceHeaderComplete )(USBSSP_DriverResourcesT *arg, uint32_t
status, const USBSSP_RingElementT *eventPtr)
```

Description

Force Header complete callback function.

3.66. USBSSP_GenericCallback

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint8_t(* USBSSP_GenericCallback )(USBSSP_RingElementT *eventPtr)
```

Description

Used for testing purposes.

3.67. USBSSP_PostCallback

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* USBSSP_PostCallback )(USBSSP_RingElementT *eventPtr)
```

Description

Used for testing purposes.

3.68. USBSSP_PreportChangeDetectCallback

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* USBSSP_PreportChangeDetectCallback )(uint32_t portsc_value, uint32_t
port_id)
```

Description

Used for testing purposes.

3.69. USBSSP_InputContextCallback

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* USBSSP_InputContextCallback )(USBSSP_DriverResourceT *arg)
```

Description

Used for virtualization.

3.70. USBSSP_USB2PhySoftReset

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* USBSSP_USB2PhySoftReset )(USBSSP_DriverResourceT *res)
```

Description

Pointer to function to perform soft reset of USB 2.0 PHY.

3.71. USBSSP_USB2ResetCallback

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* USBSSP_USB2ResetCallback )(USBSSP_DriverResourcesT *res)
```

Description

Callback to reset USB-2 interface.

3.72. USBSSP_get_phys_from_log_ptr_proc_t

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uintptr_t(* USBSSP_get_phys_from_log_ptr_proc_t )(void *log_ptr, int32_t
byte_size)
```

Description

Pointer to function that maps logical pointer to physical one.

3.73. USBSSP_get_log_from_phys_ptr_proc_t

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void *(* USBSSP_get_log_from_phys_ptr_proc_t )(uintptr_t phys_ptr, int32_t
byte_size)
```

Description

Pointer to function that maps physical pointer to logical one.

3.74. CUSBD_CbReqComplete

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* CUSBD_CbReqComplete )(CUSBD_Ep *ep, CUSBD_Req *req)
```

Description

Reports request complete callback.

Params: ep - endpoint associated with the request req - request which has been completed Note that request received in completion routine may be used again

3.75. CUSBD_CbEpEnable

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* CUSBD_CbEpEnable )(CUSBD_PrivateData *pD, CUSBD_Ep *ep, const uint8_t *desc)
```

Description

Enable Endpoint.

This function should be called within SET_CONFIGURATION(configNum > 0) request context. It configures required hardware controller endpoint with parameters given in desc.

3.76. CUSBD_CbEpDisable

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* CUSBD_CbEpDisable )(CUSBD_PrivateData *pD, CUSBD_Ep *ep)
```

Description

Disable Endpoint.

Functions unconfigures hardware endpoint. Endpoint will not respond to any host packets. This function should be called within SET_CONFIGURATION(configNum = 0) request context or on disconnect event. All queued requests on endpoint are completed with `CDN_ECANCELED` status.

3.77. CUSBD_CbEpSetHalt

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* CUSBD_CbEpSetHalt )(CUSBD_PrivateData *pD, const CUSBD_Ep *ep, uint8_t value)
```

Description

Set halt or clear halt state on endpoint.

When setting halt, device will respond with STALL packet to each host packet. When clearing halt, endpoint returns to normal operating.

3.78. CUSBD_CbEpSetWedge

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* CUSBD_CbEpSetWedge )(CUSBD_PrivateData *pD, const CUSBD_Ep *ep)
```

Description

Set halt on endpoint.

Function sets endpoint to permanent halt state. State can not be changed even with `epSetHalt(pD, ep, 0)` function. Endpoint returns automatically to its normal state on disconnect event.

3.79. CUSBD_CbEpFifoStatus

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* CUSBD_CbEpFifoStatus )(const CUSBD_PrivateData *pD, const CUSBD_Ep *ep)
```

Description

Returns number of bytes in hardware endpoint FIFO.

Function useful in application where exact number of data bytes is required. In some situation software higher layers must be aware of number of data bytes issued but not transferred by hardware because of aborted transfers, for example on disconnect event.

3.80. CUSBD_CbEpFifoFlush

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* CUSBD_CbEpFifoFlush )(CUSBD_PrivateData *pD, const CUSBD_Ep *ep)
```


Description

Flush hardware endpoint FIFO.

3.81. CUSBD_CbReqQueue

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* CUSBD_CbReqQueue )(CUSBD_PrivateData *pD, const CUSBD_Ep *ep,
CUSBD_Req *req)
```

Description

Submits IN/OUT transfer request to an endpoint.

3.82. CUSBD_CbReqDequeue

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* CUSBD_CbReqDequeue )(CUSBD_PrivateData *pD, CUSBD_Ep *ep, CUSBD_Req
*req)
```

Description

Dequeues IN/OUT transfer request from an endpoint.

Function completes all queued request with `CDN_ECANCELED` status.

3.83. CUSBD_CbDisconnect

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* CUSBD_CbDisconnect )(CUSBD_PrivateData *pD)
```

Description

Callback called on disconnect from host event.

All queued transfer on all active endpoints are completed by driver with `CDN_ECANCELED` status.

3.84. CUSBD_CbConnect

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* CUSBD_CbConnect )(CUSBD_PrivateData *pD)
```

Description

Callback called on connection to host event.

To check speed at which device has connected to host read speed flag of CUSBD object

3.85. CUSBD_CbSetup

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* CUSBD_CbSetup )(CUSBD_PrivateData *pD, CH9_UsbSetup *ctrl)
```

Description

Callback called on setup request received event Params: ctrl contains usb setup request in little endian order.

3.86. CUSBD_CdSuspend

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* CUSBD_CdSuspend )(CUSBD_PrivateData *pD)
```

Description

Called when device controller goes into suspend state.

3.87. CUSBD_CbResume

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* CUSBD_CbResume )(CUSBD_PrivateData *pD)
```

Description

Called when device returns from suspend state.

3.88. CUSBD_CbbusInterval

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* CUSBD_CbbusInterval )(CUSBD_PrivateData *pD)
```

Description

Called on bus interval event only if any isochronous endpoint used.

3.89. CUSBD_CbDescMissing

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* CUSBD_CbDescMissing )(CUSBD_PrivateData *pD, uint8_t epAddr)
```

Description

Called when descriptor missing event received.

3.90. CUSBD_CbUSB2PhySoftReset

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* CUSBD_CbUSB2PhySoftReset )(CUSBD_PrivateData *pD)
```

Description

Called to perform soft reset of USB 2 PHY.

3.91. USB_SSP_DRD_GenericCallback

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* USB_SSP_DRD_GenericCallback )(USB_SSP_DRD_PrivData *privData)
```

Description

This is a generic callback.

It is used to controll depended host and device drivers.

3.92. USB_SSP_DRD_InterruptHandler

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* USB_SSP_DRD_InterruptHandler )(USB_SSP_DRD_PrivData *privData)
```

Description

Type definition for interrupt handler specific for current state.

3.93. USB_SSP_DRD_EventIdUp

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* USB_SSP_DRD_EventIdUp )(USB_SSP_DRD_PrivData *privData)
```

Description

Type definition for transition function for an event of ID pin level change to high.

3.94. USB_SSP_DRD_EventIdDown

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* USB_SSP_DRD_EventIdDown )(USB_SSP_DRD_PrivData *privData)
```

Description

Type definition for transition function for an event of ID pin level change to low.

3.95. USB_SSP_DRD_EventIdUpVBusDown

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* USB_SSP_DRD_EventIdUpVBusDown )(USB_SSP_DRD_PrivData *privData)
```

Description

Type definition for transition function for an event of ID pin level change to high and VBUS level change to low.

3.96. USB_SSP_DRD_EventIdUpVBusUp

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* USB_SSP_DRD_EventIdUpVBusUp )(USB_SSP_DRD_PrivData *privData)
```

Description

Type definition for transition function for an event of ID pin level change to high and VBUS level change to high.

3.97. USB_SSP_DRD_EventIdDownVBusDown

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* USB_SSP_DRD_EventIdDownVBusDown )(USB_SSP_DRD_PrivData *privData)
```

Description

Type definition for transition function for an event of ID pin level change to low and VBUS level change to low.

3.98. USB_SSP_DRD_EventIdDownVBusUp

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* USB_SSP_DRD_EventIdDownVBusUp )(USB_SSP_DRD_PrivData *privData)
```

Description

Type definition for transition function for an event of ID pin level change to low and VBUS level change to high.

3.99. USB_SSP_DRD_EventEnableDrd

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* USB_SSP_DRD_EventEnableDrd )(USB_SSP_DRD_PrivData *privData)
```

Description

Type definition for transition function for an event of forcing DRD mode.

3.100. USB_SSP_DRD_EventEnableHost

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* USB_SSP_DRD_EventEnableHost )(USB_SSP_DRD_PrivData *privData)
```

Description

Type definition for transition function for an event of forcing host mode.

3.101. USB_SSP_DRD_EventEnableDev

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* USB_SSP_DRD_EventEnableDev )(USB_SSP_DRD_PrivData *privData)
```

Description

Type definition for transition function for an event of forcing device mode.

3.102. USB_SSP_DRD_EventDisableHost

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* USB_SSP_DRD_EventDisableHost )(USB_SSP_DRD_PrivData *privData)
```

Description

Type definition for transition function for an event of disabling host mode.

3.103. USB_SSP_DRD_EventDisableDev

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint32_t(* USB_SSP_DRD_EventDisableDev )(USB_SSP_DRD_PrivData *privData)
```

Description

Type definition for transition function for an event of disabling device mode.

3.104. __attribute__

Type: variable

3.105. __attribute__

Type: variable

3.106. __attribute__

Type: variable

3.107. __attribute__

Type: variable

3.108. `__attribute__`

Type: function

Chapter 4. Driver API Object

4.1. Introduction

API listing for the driver.

The API is contained in the object as function pointers in the object structure. As the actual functions resides in the Driver Object, the client software must first use the global `GetInstance` function to obtain the Driver Object Pointer. The actual APIs then can be invoked using `obj->(api_name)()` syntax. These functions are defined in the header file of the core driver and utilized by the API.

4.2. USBSSP_GetInstance

Function Declaration

```
USBSSP_OBJ* USBSSP_GetInstance();
```

```
void
```

Description

In order to access the USBSSP APIs, the upper layer software must call this global function to obtain the pointer to the driver object.

Return Value

USBSSP_OBJ* Driver Object Pointer

4.3. CUSBD_GetInstance

Function Declaration

```
CUSBD_OBJ* CUSBD_GetInstance();
```

```
void
```

Description

In order to access the CUSBD APIs, the upper layer software must call this global function to obtain the pointer to the driver object.

Return Value

CUSBD_OBJ* Driver Object Pointer

4.4. CUSBDMA_GetInstance

Function Declaration

```
CUSBDMA_OBJ* CUSBDMA_GetInstance();
```

```
void
```

Description

In order to access the CUSBDMA APIs, the upper layer software must call this global function to obtain the pointer to the driver object.

Return Value

CUSBDMA_OBJ* Driver Object Pointer

4.5. USB_SSP_DRD_GetInstance

Function Declaration

```
USB_SSP_DRD_OBJ* USB_SSP_DRD_GetInstance();
```

```
void
```

Description

In order to access the USB_SSP_DRD APIs, the upper layer software must call this global function to obtain the pointer to the driver object.

Return Value

USB_SSP_DRD_OBJ* Driver Object Pointer

USBSSP_OBJ

4.6. transferData

Function Declaration

```
uint32_t ();
```

```
;
```

Parameter List

res [in]

Driver resources

epIndex [in]

index of endpoint according to xhci spec e.g for eplout epIndex=2, for eplin epIndex=3, for ep2out epIndex=4 end so on \$RANGE \$FROM USBSSP_EP_CONT_OFFSET \$TO USBSSP_EP_CONT_MAX - (uint8_t)1 \$

buff [in]

Buffer for data to send or to receive

size [in]

Size of data in bytes

complete [in]

pointer to function which will be returned in callback in input parameter, can be set to NULL when no extra parameter used

Description

Transfer data on given endpoint.

This function is non-blocking type. The XHCI operation result should be checked in complete callback function.

Return Value

CDN_EINVAL if selected endpoint index is out of available range

CDN_EOK if selected endpoint is within available endpoint range

4.7. transferVectorData

Function Declaration

```
uint32_t ();  
;
```

Parameter List

res [in]

Driver resources

epIndex [in]

index of endpoint according to xhci spec e.g for eplout epIndex=2, for eplin epIndex=3, for ep2out epIndex=4 end so on \$RANGE \$FROM USBSSP_EP_CONT_OFFSET \$TO USBSSP_EP_CONT_MAX - (uint8_t)1 \$

bufferDesc [in]

Pointer to an array of buffer descriptors

bufferCount [in]

Buffer descriptor count

complete [in]

pointer to function which will be returned in callback in input parameter, can be set to NULL when no extra parameter used

Description

Gather and Transfer Vector data.

Return Value

CDN_EINVAL if selected endpoint index is out of available range

CDN_EOK if selected endpoint is within available endpoint range

4.8. stopEndpoint

Function Declaration

```
uint32_t ();
```

```
;
```

Parameter List

res [in]

Driver resources

endpoint [in]

Index of endpoint to stop \$RANGE \$FROM USBSSP_EP0_CONT_OFFSET \$TO
USBSSP_EP_CONT_MAX - (uint8_t)1 \$

Description

Stop endpoint.

Function sends STOP_ENDPOINT_COMMAND command to USBSSP controller

Return Value

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if no errors

4.9. resetEndpoint

Function Declaration

```
uint32_t ();
```

```
;
```

Parameter List

res [in]

Driver resources

endpoint [in]

Index of endpoint to reset \$RANGE \$FROM USBSSP_EP0_CONT_OFFSET \$TO
USBSSP_EP_CONT_MAX - (uint8_t)1 \$

Description

Endpoint reset.

Function sends RESET_ENDPOINT_COMMAND to USBSSP controller

Return Value

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if no errors

4.10. resetDevice

Function Declaration

```
uint32_t ();  
;
```

Parameter List

res [in]

Driver resources

Description

Reset of connected device.

Function sends RESET_DEVICE_COMMAND to USBSSP controller in order to issue reset state on USB bus.

Return Value

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK selected endpoint is within available endpoint range

4.11. isr

Function Declaration

```
uint32_t ();  
;
```

Parameter List

res [in]
Driver resources

Description

Handling of USBSSP controller interrupt.

Function is called from USBSSP interrupt context.

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

4.12. SetMemRes

Function Declaration

```
uint32_t ( );  
;
```

Parameter List

res [in]
Driver resources

memRes [in]
User defined memory resources.

Description

Function sets memory resources used by driver.

Return Value

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if no errors

4.13. init

Function Declaration

```
uint32_t ( );
```

;

Parameter List

res [in]

Driver resources

base [in]

Physical address of USBSSP controller where is mapped in system

Description

Initialization of USBSSP_DriverResourcesT object.

USBSSP_DriverResourcesT object keeps all resources required by USBSSP controller. It represents USBSSP hardware controller.

Return Value

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if no errors

4.14. getDescriptor

Function Declaration

uint32_t ();

;

Parameter List

res [in]

Driver resources

descType [in]

Type of descriptor to get (CH9_USB_DT_DEVICE, CH9_USB_DT_CONFIGURATION,...)

complete [in]

Complete callback function

Description

Get descriptor.

Function gets descriptor from connected device, used in host mode and stores it in internal res->ep0Buff buffer. Maximal descriptor length is limited to 255. Function is blocking type and must not be called from interrupt context.

Return Value

CDN_EOK on success

complete_code XHCI transfer complete status code

4.15. setAddress**Function Declaration**

```
uint32_t ();  
;
```

Parameter List

```
res [in]  
    Driver resources
```

Description

Set address.

Function executes set address request on connected device.

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

4.16. resetRootHubPort**Function Declaration**

```
uint32_t ();  
;
```

Parameter List

```
res [in]  
    Driver resources
```

Description

Reset port.

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

4.17. issueGenericCommand

Function Declaration

```
uint32_t ();  
;
```

Parameter List

res	[in]	Driver resources
dword0	[in]	word 0 of command
dword1	[in]	word 1 of command
dword2	[in]	word 2 of command
dword3	[in]	word 3 of command

Description

Issue generic command to SSP controller.

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

4.18. endpointSetFeature

Function Declaration

```
uint32_t ();  
;
```

Parameter List

res	[in]
-----	--------

Driver resources

epIndex [in]

Index of endpoint to set/clear feature on \$RANGE \$FROM USBSSP_EP_CONT_OFFSET \$TO USBSSP_EP_CONT_MAX - (uint8_t)1 \$

feature [in]

When 1 sets stall, when 0 clears stall

Description

Set feature on device's endpoint.

Functions sends setup requested to device with set/cleared endpoint feature

Return Value

CDN_EOK on success

complete_code XHCI transfer complete status code

4.19. setConfiguration

Function Declaration

```
uint32_t ( );
;
```

Parameter List

res [in]

Driver resources

configValue [in]

USB device's configuration selector

Description

Set configuration.

Function configures USBSSP controller as well as device connected to this USBSSP controller. This is a blocking function. This function must not be called from an interrupt context.

Return Value

CDN_EOK on success

complete_code XHCI transfer complete status code

4.20. nBControlTransfer

Function Declaration

```
uint32_t ( );
```

```
;
```

Parameter List

res [in]

Driver resources

setup [in]

Keeps setup packet

pdata [in]

Pointer for data to send/receive \$RANGE \$NULLABLE \$

complete [in]

Complete callback function

Description

No blocking control transfer.

Function executes control transfer. Information about transfer like: data direction, data length, wIndex, wValue etc. are passed in 'setup' parameter.

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

4.21. noOpTest

Function Declaration

```
uint32_t ( );
```

```
;
```

Parameter List

res [in]

Driver resources

complete [in]

Callback

Description

No Operation test.

Function used for testing purposes: NO_OP_COMMAND is send to USBSSP controller. When event ring receives NO_OP_COMMAND complete it calls complete callback

Return Value

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if no errors

4.22. calcFsLsEPIntrptInterval**Function Declaration**

```
uint8_t ();
;
```

Parameter List

```
bInterval  [ in ]
           bInterval
```

Description

Calculate full/low speed endpoint interval based on bInterval See xHCI spec Section 6.2.3.6 for more details.

Return Value

value valid endpoint context interval value

4.23. enableSlot**Function Declaration**

```
uint32_t ();
;
```

Parameter List

```
res  [ in ]
     Driver resources
```

Description

Function enables slot for new connected device.

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

4.24. disableSlot

Function Declaration

```
uint32_t ();
;
```

Parameter List

```
res [in]
    Driver resources
```

Description

Function disables slot of connected device.

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

4.25. enableEndpoint

Function Declaration

```
uint32_t ();
;
```

Parameter List

```
res [in]
    Driver resources

desc [in]
    pointer to endpoint descriptor
```

Description

Enable endpoint.

Function used in device context.

Return Value

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if no errors

4.26. disableEndpoint

Function Declaration

```
uint32_t ();

;
```

Parameter List

res	[in]
	Driver resources
epAddress	[in]
	address of endpoint to be disabled

Description

Disable endpoint.

Function used in device context.

Return Value

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if no errors

4.27. getMicroFrameIndex

Function Declaration

```
uint32_t ();

;
```

Parameter List

res	[in]
-----	--------

driver resources

index [out]

Micro Frame Index returned by function.

Description

Get actual frame number.

Function returns actual frame number on USB bus. Remember that maximal frame number can be 0x7FF. Next frame increments returned value from 0.

4.28. setEndpointExtraFlag

Function Declaration

```
uint32_t ( );
;
```

Parameter List

res [in]

driver resources

epIndex [in]

endpoint index

flags [in]

Endpoint Extra Flag

Description

set end point extra flag

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

4.29. cleanEndpointExtraFlag

Function Declaration

```
uint32_t ( );
```

;

Parameter List

res [in]
 driver resources

epIndex [in]
 end point index

flags [in]
 Endpoint Extra Flag

Description

clean endpoint extra flag

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

4.30. getEndpointExtraFlag**Function Declaration**

```
uint32_t ( );
```

;

Parameter List

res [in]
 driver resources

epIndex [in]
 endpoint index

flag [out]
 Endpoint Extra Flag returned by pointer

Description

get endpoint extra flag

4.31. setFrameID

Function Declaration

```
uint32_t ( );
```

```
;
```

Parameter List

res	[in]
Driver resources	
epIndex	[in]
endpoint index	
frameID	[in]
Frame ID	

Description

set Frame ID

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

4.32. addEventDataTRB

Function Declaration

```
uint32_t ( );
```

```
;
```

Parameter List

res	[in]
Driver resources	
epIndex	[in]
endpoint index	
eventDataLo	[in]
event data Low	

```
eventDataHi  [ in ]
              event data High
flags        [ in ]
              Flags
```

Description

Add event data TRB.

Return Value

value TBC

4.33. forceHeader

Function Declaration

```
uint32_t ( );
;
```

Parameter List

```
res          [ in ]
              driver resources
trbDwords    [ in ]
              trb words
complete     [ in ]
              complete callback
```

Description

Force header.

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

4.34. setPortOverrideReg

Function Declaration

```
uint32_t ( );
```

;

Parameter List

res [in]
 driver resources

regValue [in]
 Register value

Description

set port override register

4.35. getPortOverrideReg**Function Declaration**

```
uint32_t ( );
;
```

Parameter List

res [in]
 Driver resources

regValue [out]
 Register value

Description

Get port override register.

4.36. setPortControlReg**Function Declaration**

```
uint32_t ( );
;
```

Parameter List

res [in]
 driver resources

portId	[in]
	port ID
portRegIdx	[in]
	port control register ID
regValue	[in]
	Register value

Description

set port control register

4.37. getPortControlReg

Function Declaration

```
uint32_t ();
;
```

Parameter List

res	[in]
	Driver resources
portId	[in]
	port ID
portRegIdx	[in]
	port control register ID
regValue	[out]
	Register value

Description

Get port control register.

4.38. SaveState

Function Declaration

```
uint32_t ();
;
```

Parameter List

res [in]
Driver resources

drvContext [in]
Pointer to driver context struct

Description

Save state and stop xHC.

4.39. RestoreState**Function Declaration**

```
uint32_t ();  
;
```

Parameter List

res [in]
Driver resources

drvContext [in]
Pointer to driver context struct

Description

Restore state and start xHC.

4.40. getPortConnected**Function Declaration**

```
uint32_t ();  
;
```

Parameter List

res [in]
Driver resources

connected [out]

connected status

Description

Get connected status.

4.41. getDevAddressState

Function Declaration

```
uint32_t ();
;
```

Parameter List

res	[in]
	Driver resources
addrState	[out]
	dev address state

Description

Get dev address state.

CUSBD_OBJ

4.42. probe

Function Declaration

```
uint32_t ();
;
```

Parameter List

config	[in]
	driver/hardware configuration required
sysReqCusbd	[out]
	returns the size of memory allocations required

Description

Obtain the private memory size required by the driver.

Return Value

0 on success (requirements structure filled)

CDN_ENOTSUP if configuration cannot be supported due to driver/hardware constraints

4.43. init

Function Declaration

```
uint32_t ();
```

```
;
```

Parameter List

`pD` [out]

driver state info specific to this instance

`config` [inout]

specifies driver/hardware configuration

`callbacks` [in]

client-supplied callback functions

Description

Initialize the driver instance and state, configure the USB device as specified in the 'config' settings, initialize locks used by the driver.

Return Value

CDN_EOK on success

CDN_ENOTSUP if hardware has an inconsistent configuration or doesn't support feature(s) required by 'config' parameters

CDN_ENOENT if insufficient locks were available (i.e. something allocated locks between probe and init)

CDN_EIO if driver encountered an error accessing hardware

CDN_EINVAL if illegal/inconsistent values in 'config'

4.44. destroy

Function Declaration

```
uint32_t ();
```

;

Parameter List

pD [in]

driver state info specific to this instance

Description

Destroy the driver (automatically performs a stop)

Return Value

CDN_EOK on success

CDN_EINVAL if illegal/inconsistent values in 'config'

4.45. start

Function Declaration

```
uint32_t ();
```

;

Parameter List

pD [in]

driver state info specific to this instance

Description

Start the USB driver.

Return Value

CDN_EOK on success

CDN_EINVAL if illegal/inconsistent values in 'config'

4.46. stop

Function Declaration

```
uint32_t ();
```

;

Parameter List

pD [in]

driver state info specific to this instance

Description

Stop the driver.

This should disable the hardware, including its interrupt at the source, and make a best-effort to cancel any pending transactions.

Return Value

CDN_EOK on success

CDN_EINVAL if any input parameter is NULL

4.47. isr

Function Declaration

```
uint32_t ();  
;
```

Parameter List

pD [in]

driver state info specific to this instance

Description

Driver ISR.

Platform-specific code is responsible for ensuring this gets called when the corresponding hardware's interrupt is asserted. Registering the ISR should be done after calling init, and before calling start. The driver's ISR will not attempt to lock any locks, but will perform client callbacks. If the client wishes to defer processing to non-interrupt time, it is responsible for doing so.

4.48. epEnable

Function Declaration

```
uint32_t ();  
;
```

Parameter List

pD [in]

driver state info specific to this instance

ep [in]

endpoint being configured

desc [in]

endpoint descriptor

Description

Enable Endpoint.

This function should be called within SET_CONFIGURATION(configNum > 0) request context. It configures required hardware controller endpoint with parameters given in desc.

Return Value

0 on success

CDN_EINVAL if pD, ep or desc is NULL or desc is not a endpoint descriptor

4.49. epDisable

Function Declaration

```
uint32_t ();  
;
```

Parameter List

pD [in]

driver state info specific to this instance

ep [in]

endpoint being unconfigured

Description

Disable Endpoint.

Functions unconfigures hardware endpoint. Endpoint will not respond to any host packets. This function should be called within SET_CONFIGURATION(configNum = 0) request context or on disconnect event. All queued requests on endpoint are completed with CDN_ECANCELED status.

Return Value

0 on success

CDN_EINVAL if pD or ep is NULL

4.50. epSetHalt

Function Declaration

```
uint32_t ( );  
;
```

Parameter List

pD [in]
driver state info specific to this instance

ep [in]
endpoint being setting or clearing halt state

value [in]
if 1 sets halt, if 0 clears halt on endpoint

Description

Set halt or clear halt state on endpoint.

When setting halt, device will respond with STALL packet to each host packet. When clearing halt, endpoint returns to normal operating.

Return Value

0 on success

CDN_EPERM if endpoint is disabled (has not been configured yet)

CDN_EINVAL if pD or ep is NULL

4.51. epSetWedge

Function Declaration

```
uint32_t ( );  
;
```

Parameter List

pD [in]
driver state info specific to this instance

ep [in]
endpoint being setting halt state

Description

Set halt on endpoint.

Function sets endpoint to permanent halt state. State can not be changed even with epSetHalt(pD, ep, 0) function. Endpoint returns automatically to its normal state on disconnect event.

Return Value

0 on success

CDN_EPERM if endpoint is disabled (has not been configured yet)

CDN_EINVAL if pD or ep is NULL

4.52. epFifoStatus

Function Declaration

```
uint32_t ();
;
```

Parameter List

pD [in]

driver state info specific to this instance

ep [in]

endpoint which status is to be returned

Description

Returns number of bytes in hardware endpoint FIFO.

Function useful in application where exact number of data bytes is required. In some situation software higher layers must be aware of number of data bytes issued but not transfered by hardware because of aborted transfers, for example on disconnect event. *

Return Value

number of bytes in hardware endpoint FIFO

CDN_EINVAL if pD or ep is NULL

4.53. epFifoFlush

Function Declaration

```
uint32_t ();
```

;

Parameter List

pD [in]

driver state info specific to this instance

ep [in]

endpoint which FIFO is to be flushed

Description

Flush hardware endpoint FIFO.

4.54. reqQueue**Function Declaration**

uint32_t ();

;

Parameter List

pD [in]

driver state info specific to this instance

ep [in]

endpoint associated with the request

req [in]

request being submitted

Description

Submits IN/OUT transfer request to an endpoint.

Return Value

0 on success

CDN_EPROTO only on default endpoint if endpoint is not in data stage phase

CDN_EPERM if endpoint is not enabled

CDN_EINVAL if pD, ep, or req is NULL

4.55. reqDequeue

Function Declaration

```
uint32_t ();  
;
```

Parameter List

pD [in]
driver state info specific to this instance

ep [in]
endpoint associated with the request

req [in]
request being dequeued

Description

Dequeues IN/OUT transfer request from an endpoint.

Function completes all queued request with CDN_ECANCELED status.

Return Value

0 on success

CDN_EINVAL if pD or ep is NULL

4.56. getDevInstance

Function Declaration

```
void ();  
;
```

Parameter List

pD [in]
driver state info specific to this instance

dev [out]
returns pointer to CUSBD instance

Description

Returns pointer to CUSBD object.

CUSBD object is a logical representation of USB device. CUSBD contains endpoint collection accessed through epList field. Endpoints collection is organized as double linked list.

4.57. dGetFrame

Function Declaration

```
uint32_t ();  
;
```

Parameter List

pD	[in]	driver state info specific to this instance
numOfFrame	[out]	returns number of USB frame

Description

Returns number of frame.

Some controllers have counter of SOF packets or ITP packets in the Super Speed case. Function returns value of this counter. This counter can be used for time measurement. Single FS frame is 1 ms measured, for HS and SS is 125us.

Return Value

CDN_EOK on success
CDN_EOPNOTSUPP if feature is not supported
CDN_EINVAL if pD or numOfFrame is NULL

4.58. dSetSelfpowered

Function Declaration

```
uint32_t ();  
;
```

Parameter List

pD	[in]	driver state info specific to this instance
----	--------	---

Description

Sets the device self powered feature.

Return Value

CDN_EOK on success

CDN_EOPNOTSUPP if feature is not supported

CDN_EINVAL if pD is NULL

4.59. dClearSelfpowered**Function Declaration**

```
uint32_t ();

;
```

Parameter List

pD [in]

driver state info specific to this instance

Description

Clear the device self powered feature.

Return Value

CDN_EOK on success

CDN_EOPNOTSUPP if feature is not supported

CDN_EINVAL if pD is NULL

4.60. dGetConfigParams**Function Declaration**

```
uint32_t ();

;
```

Parameter List

pD [in]

driver state info specific to this instance

configParams [out]

pointer to CH9_ConfigParams structure

Description

Returns configuration parameters: U1 exit latency and U2 exit latency Function useful only in Super Speed mode.

Return Value

CDN_EOK on success

CDN_EINVAL if illegal/inconsistent values in 'config'

CUSBDMA_OBJ

4.61. probe

Function Declaration

```
uint32_t ();
;
```

Parameter List

`config` [in]
driver/hardware configuration required

`sysReq` [out]
sysReq returns the size of memory allocations required

Description

Obtain the private memory size required by the driver.

Return Value

CDN_EOK on success

CDN_EINVAL if function parameters are not valid

CDN_ENOTSUP if configuration cannot be supported due to driver/hardware constraints

4.62. init

Function Declaration

```
uint32_t ();
;
```

Parameter List

`pD` [in]

driver state info specific to this instance

`config` [in]

specifies driver/hardware configuration

Description

Initialize the driver instance and state, configure the USB device as specified in the 'config' settings, initialize locks used by the driver.

Return Value

CDN_EOK on success

CDN_EINVAL if function parameters are not valid

CDN_ENOTSUP if configuration cannot be supported due to driver/hardware constraints

CDN_EIO if driver encountered an error accessing hardware

CDN_ENOENT insufficient locks were available (i.e. something allocated locks between probe and init)

4.63. destroy

Function Declaration

```
uint32_t ();
;
```

Parameter List

`pD` [in]

driver state info specific to this instance

Description

Destroy the driver (automatically performs a stop).

4.64. channelAlloc

Function Declaration

```
uint32_t ();
;
```

Parameter List

`pD` [in]

	driver state info specific to this instance
channelPtr	[out]
	address of channel pointer
channelParams	[in]
	Channel parameters

Description

Allocation the DMA channel.

4.65. channelReset

Function Declaration

```
uint32_t ();
;
```

Parameter List

pD	[in]
	driver state info specific to this instance
channel	[in]
	channel pointer to DMA channel

Description

Reset the DMA channel.

Return Value

CDN_EOK on success

CDN_EINVAL if function parameters are not valid

CDN_EPERM if the DMA is active and channel cannot be reset

4.66. channelRelease

Function Declaration

```
uint32_t ();
;
```

Parameter List

`pD` [in]
driver state info specific to this instance

`channel` [in]
channel pointer to DMA channel

Description

Release the DMA channel.

Return Value

CDN_EOK on success

CDN_EINVAL if function parameters are not valid

4.67. channelProgram

Function Declaration

```
uint32_t ();  
;
```

Parameter List

`pD` [in]
driver state info specific to this instance

`channel` [in]
channel pointer to DMA channel for which transfer will be started

`params` [in]
transfer parameters container

Description

Prepares transfer and starts it.

Return Value

CDN_EOK on success

CDN_EINVAL if function parameters are not valid

CDN_ENOMEM if DMA channel has no free memory for TRB

4.68. channelTrigger

Function Declaration

```
uint32_t ();  
;
```

Parameter List

`pD` [in]
driver state info specific to this instance

`channel` [in]
pointer to DMA channel which needs to be triggered

Description

Triggers DMA transfer for given DMA channel if TRBs are queued.

Return Value

CDN_EOK on successful trigger or if the DMA is already active

CDN_EINVAL if function parameters are not valid

CDN_ENOTSUP if TRBs are not queued

4.69. channelUpdateState

Function Declaration

```
uint32_t ();  
;
```

Parameter List

`pD` [in]
driver state info specific to this instance

`channel` [in]
pointer to DMA channel whose state needs to be updated

Description

Updates the data transfer status of a channel.

Return Value

CDN_EOK on successful trigger or if the DMA is already active

CDN_EINVAL if function parameters are not valid

4.70. channelSetMaxPktSz

Function Declaration

```
uint32_t ();
```

```
;
```

Parameter List

pD	[in]	
		driver state info specific to this instance
channel	[in]	
		pointer to DMA channel whose state needs to be updated
maxPacketSize	[in]	
		Value of max packet size

Description

Updates the max packet size for a channel.

Return Value

CDN_EOK on successful successful update of max packet size

CDN_EINVAL if function parameters are not valid

4.71. channelHandleStall

Function Declaration

```
uint32_t ();
```

```
;
```

Parameter List

pD	[in]	
		driver state info specific to this instance

`channel` [in]
 pointer to DMA channel whose stall state is handled

`value` [in]
 Clear stall if 0, else set stall

`timeout` [in]
 Timeout for waiting for flush operation while stalling

Description

Set or Clear channel stall.

Return Value

CDN_EOK on successful trigger or if the DMA is already active

CDN_EINVAL if function parameters are not valid

4.72. channelFreeHeadTrbChain

Function Declaration

```
uint32_t ( );  
;
```

Parameter List

`pD` [in]
 driver state info specific to this instance

`channel` [in]
 pointer to DMA channel whose descriptor needs to be freed

Description

Free the head(oldest) TRB chain descriptor for this channel.

Return Value

CDN_EOK on success

CDN_EINVAL if function parameters are not valid

USB_SSP_DRD_OBJ

4.73. Probe

Function Declaration

```
uint32_t ();  
;
```

Parameter List

sysReq [out]

Returns the memory requirements for given configuration.

Description

Returns the memory requirements for a driver instance.

Return Value

CDN_EOK On success.

CDN_EINVAL If config contains invalid values or not supported.

4.74. Init

Function Declaration

```
uint32_t ();  
;
```

Parameter List

privData [in]

Pointer to driver's private data object.

config [in]

Specifies driver/hardware configuration.

callbacks [in]

Event Handlers and Callbacks.

Description

Instantiates the USB_SSP_DRD Core Driver, given the required blocks of memory (this includes initializing the instance and the underlying hardware).

If a client configuration is required (likely to always be true), it is passed in also. Returns an instance pointer, which the client must maintain and pass to all other driver functions. (except probe).

Return Value

CDN_EOK On success

CDN_EINVAL If illegal/inconsistent values in 'config' doesn't support feature(s) required by 'config' parameters.

CDN_EIO if operation failed

4.75. Isr

Function Declaration

```
uint32_t ();
;
```

Parameter List

privData [in]

Pointer to driver's private data object filled by init.

Description

USB_SSP_DRD Core Driver's ISR.

Platform-specific code is responsible for ensuring this gets called when the corresponding hardware's interrupt is asserted. Registering the ISR should be done after calling init, and before calling start. The driver's ISR will not attempt to lock any locks, but will perform client callbacks. If the client wishes to defer processing to non- interrupt time, it is responsible for doing so. This function must not be called after calling destroy and releasing private data memory.

Return Value

CDN_EOK on success

CDN_EINVAL if input parameters are invalid

4.76. Start

Function Declaration

```
uint32_t ();
;
```

Parameter List

privData [in]

Pointer to driver's private data object.

Description

Start the USB_SSP_DRD driver, enabling interrupts.

This is called after the client has successfully initialized the driver and hooked the driver's ISR (the `isr` member of this struct) to the IRQ.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid

4.77. Stop

Function Declaration

```
uint32_t ();  
;
```

Parameter List

`privData` [in]

Pointer to driver's private data object.

Description

The client may call this to disable the hardware (disabling its IRQ at the source and disconnecting it if applicable).

Also, a best-effort is made to cancel any pending transactions.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid

4.78. Destroy

Function Declaration

```
uint32_t ();  
;
```

Parameter List

`privData` [in]

Pointer to driver's private data object.

Description

This performs an automatic stop and then de-initializes the driver.

The client may not make further requests on this instance.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid

4.79. CheckIfReady

Function Declaration

```
uint32_t ();  
;
```

Parameter List

privData [in]

Pointer to driver's private data object.

isReady [out]

Will tell if controller is ready.

Description

Checks if controller is ready.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid

4.80. CheckStrapMode

Function Declaration

```
uint32_t ();  
;
```

Parameter List

`privData` [in]

Pointer to driver's private data object.

`strapMode` [out]

Will tell how controller is strapped.

Description

Checks strap mode of controller.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid

4.81. CheckOperationMode

Function Declaration

```
uint32_t ( );

;
```

Parameter List

`privData` [in]

Pointer to driver's private data object.

`operationMode` [out]

Will tell operation mode of controller.

Description

Checks current operation mode of controller.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid

4.82. SetOperationMode

Function Declaration

```
uint32_t ( );
```

```
;
```

Parameter List

privData [in]

Pointer to driver's private data object.

operationMode [in]

Mode of operation that should be set for controller.

Description

Sets operation mode of controller.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid

4.83. CheckIrq

Function Declaration

```
uint32_t ( );
```

```
;
```

Parameter List

privData [in]

Pointer to driver's private data object.

interruptVect [out]

Interrupt Vector value.

Description

Checks if there is an unhandled interrupt pending.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid

4.84. ProcessIrq

Function Declaration

```
uint32_t ();  
  
;
```

Parameter List

privData [in]

Pointer to driver's private data object.

Description

This function will process an interrupt that is pending.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid

Chapter 5. Driver Function API

5.1. Introduction

Prototypes for the driver API functions.

The user application can link statically to the necessary API functions and call them directly.

5.2. USBSSP_TransferData

Function Declaration

```
uint32_t USBSSP_TransferData(res, epIndex, uintptr_t buff, size, complete);
```

```
USBSSP_DriverResourceT *res
```

```
uint8_t epIndex
```

```
const uintptr_t buff
```

```
uint32_t size
```

```
USBSSP_Complete complete
```

Parameter List

res [in]

Driver resources

epIndex [in]

index of endpoint according to xhci spec e.g for ep1out epIndex=2, for ep1in epIndex=3, for ep2out epIndex=4 end so on \$RANGE \$FROM USBSSP_EP_CONT_OFFSET \$TO USBSSP_EP_CONT_MAX - (uint8_t)1 \$

buff [in]

Buffer for data to send or to receive

size [in]

Size of data in bytes

complete [in]

pointer to function which will be returned in callback in input parameter, can be set to NULL when no extra parameter used

res [in]

driver resources

ep_index [in]

index of endpoint according to xhci spec e.g for ep1out ep_index=2, for ep1in ep_index=3, for ep2out ep_index=4 end so on

buff [in]

buffer for data to send or to receive

size [in]

size of data in bytes

complete [in]

pointer to complete callback function

Description

Transfer data on given endpoint.

This function is non-blocking type. The XHCI operation result should be checked in complete callback function.

Return Value

CDN_EINVAL if selected endpoint index is out of available range

CDN_EOK if selected endpoint is within available endpoint range

CDN_EINVAL if selected endpoint index is out of available range

CDN_EOK if selected endpoint is within available endpoint range

5.3. USBSSP_TransferVectorData

Function Declaration

```
uint32_t USBSSP_TransferVectorData(res, epIndex, bufferDesc, bufferCount, complete);
```

```
USBSSP_DriverResourceT *res
```

```
uint8_t epIndex
```

```
const USBSSP_XferBufferDesc *bufferDesc
```

```
uint32_t bufferCount
```

```
USBSSP_Complete complete
```

Parameter List

res [in]

Driver resources

epIndex [in]

index of endpoint according to xhci spec e.g for eplout epIndex=2, for eplin epIndex=3, for ep2out epIndex=4 end so on \$RANGE \$FROM USBSSP_EP_CONT_OFFSET \$TO USBSSP_EP_CONT_MAX - (uint8_t)1 \$

bufferDesc	[in]	Pointer to an array of buffer descriptors
bufferCount	[in]	Buffer descriptor count
complete	[in]	pointer to function which will be returned in callback in input parameter, can be set to NULL when no extra parameter used
res	[]	driver resources
epIndex	[]	endpoint index
bufferDesc	[]	buffer for user data buffers
bufferCount	[]	number of user buffers
complete	[]	completion callback

Description

Gather and Transfer Vector data.

Return Value

CDN_EINVAL if selected endpoint index is out of available range

CDN_EOK if selected endpoint is within available endpoint range

CDN_EOK if success, error code elsewhere

5.4. USBSSP_StopEndpoint

Function Declaration

```
uint32_t USBSSP_StopEndpoint(res, endpoint);
```

```
USBSSP_DriverResourceT *res
```

```
uint8_t endpoint
```

Parameter List

res [in]

Driver resources

endpoint [in]

Index of endpoint to stop \$RANGE \$FROM USBSSP_EP0_CONT_OFFSET \$TO
USBSSP_EP_CONT_MAX - (uint8_t)1 \$

res [in]

driver resources

endpoint [in]

index of endpoint to stop

Description

Stop endpoint.

Function sends STOP_ENDPOINT_COMMAND command to USBSSP controller

Return Value

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if no errors

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if no errors

5.5. USBSSP_ResetEndpoint

Function Declaration

```
uint32_t USBSSP_ResetEndpoint(res, endpoint);
```

```
USBSSP_DriverResourceT *res
```

```
uint8_t endpoint
```

Parameter List

res [in]

Driver resources

endpoint [in]

Index of endpoint to reset \$RANGE \$FROM USBSSP_EP0_CONT_OFFSET \$TO
 USBSSP_EP_CONT_MAX - (uint8_t)1 \$

res [in]

driver resources

endpoint [in]

index of endpoint to reset

Description

Endpoint reset.

Function sends RESET_ENDPOINT_COMMAND to USBSSP controller

Return Value

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if no errors

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if no errors

5.6. USBSSP_ResetDevice

Function Declaration

```
uint32_t USBSSP_ResetDevice(res);
```

```
USBSSP_DriverResourcesT *res
```

Parameter List

res [in]

Driver resources

res [in]

driver resources

Description

Reset of connected device.

Function sends RESET_DEVICE_COMMAND to USBSSP controller in order to issue reset state on USB bus.

Return Value

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK selected endpoint is within available endpoint range

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if selected endpoint is within available endpoint range

5.7. USBSSP_Isr

Function Declaration

```
uint32_t USBSSP_Isr(res);
```

```
USBSSP_DriverResourceT *res
```

Parameter List

```
res [ in ]
```

Driver resources

```
res [ in ]
```

driver resources

Description

Handling of USBSSP controller interrupt.

Function is called from USBSSP interrupt context.

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

CDN_EOK on success, error otherwise

5.8. USBSSP_SetMemRes

Function Declaration

```
uint32_t USBSSP_SetMemRes(res, memRes);
```

```
USBSSP_DriverResourceT *res
```

```
USBSSP_XhciResourceT *memRes
```

Parameter List

```
res [ in ]
```

Driver resources

```
memRes [ in ]
```

User defined memory resources.

`res` `[]`

driver resources

`memRes` `[]`

user's memory resources

Description

Function sets memory resources used by driver.

Return Value

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if no errors

CDN_EOK on success, error code elsewhere

5.9. USBSSP_Init

Function Declaration

```
uint32_t USBSSP_Init(res, base);
```

USBSSP_DriverResourcesT *res

uintptr_t base

Parameter List

`res` `[in]`

Driver resources

`base` `[in]`

Physical address of USBSSP controller where is mapped in system

`res` `[in]`

driver resources

`base` `[in]`

physical address of SSP controller where it is mapped in system

Description

Initialization of USBSSP_DriverResourcesT object.

USBSSP_DriverResourcesT object keeps all resources required by USBSSP controller. It represents USBSSP hardware controller.

Return Value

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if no errors

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if no errors

5.10. USBSSP_GetDescriptor

Function Declaration

```
uint32_t USBSSP_GetDescriptor(res, descType, complete);
```

```
USBSSP_DriverResourcesT *res
```

```
uint8_t descType
```

```
USBSSP_Complete complete
```

Parameter List

res [in]

Driver resources

descType [in]

Type of descriptor to get (CH9_USB_DT_DEVICE, CH9_USB_DT_CONFIGURATION,...)

complete [in]

Complete callback function

res [in]

driver resources

descType [in]

type of descriptor to get (CH9_USB_DT_DEVICE, CH9_USB_DT_CONFIGURATION,...)

complete [in]

Complete callback function

Description

Get descriptor.

Function gets descriptor from connected device, used in host mode and stores it in internal `res->ep0Buff` buffer. Maximal descriptor length is limited to 255. Function is blocking type and must not be called from interrupt context.

Return Value

CDN_EOK on success

complete_code XHCI transfer complete status code

CDN_EOK on success

complete_code XHCI transfer complete status code

5.11. USBSSP_SetAddress

Function Declaration

```
uint32_t USBSSP_SetAddress(res);
```

```
USBSSP_DriverResourcesT *res
```

Parameter List

res [in]

Driver resources

res []

driver resources

Description

Set address.

Function executes set address request on connected device.

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

5.12. USBSSP_ResetRootHubPort

Function Declaration

```
uint32_t USBSSP_ResetRootHubPort(res);
```

```
const USBSSP_DriverResourcesT *res
```

Parameter List

res [in]

Driver resources

`res` [in]

driver resources

Description

Reset port.

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

CDN_EOK if success, error code elsewhere

5.13. USBSSP_IssueGenericCommand

Function Declaration

```
uint32_t USBSSP_IssueGenericCommand(res, dword0, dword1, dword2, dword3);
```

USBSSP_DriverResourcesT **res*

uint32_t *dword0*

uint32_t *dword1*

uint32_t *dword2*

uint32_t *dword3*

Parameter List

`res` [in]

Driver resources

`dword0` [in]

word 0 of command

`dword1` [in]

word 1 of command

`dword2` [in]

word 2 of command

`dword3` [in]

word 3 of command

res [in]

Driver resources

dword0 [in]

word 0 of command

dword1 [in]

word 1 of command

dword2 [in]

word 2 of command

dword3 [in]

word 3 of command

Description

Issue generic command to SSP controller.

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

CDN_EOK for success, error code elsewhere

5.14. USBSSP_EndpointSetFeature

Function Declaration

```
uint32_t USBSSP_EndpointSetFeature(res, epIndex, feature);
```

USBSSP_DriverResourcesT *res

uint8_t epIndex

uint8_t feature

Parameter List

res [in]

Driver resources

epIndex [in]

Index of endpoint to set/clear feature on \$RANGE \$FROM USBSSP_EP_CONT_OFFSET \$TO USBSSP_EP_CONT_MAX - (uint8_t)1 \$

feature [in]

When 1 sets stall, when 0 clears stall

res [in]

driver resources

epIndex [in]

index of endpoint to set/clear feature on

feature [in]

when 1 sets stall, when 0 clears stall

Description

Set feature on device's endpoint.

Functions sends setup requested to device with set/cleared endpoint feature

Return Value

CDN_EOK on success

complete_code XHCI transfer complete status code

CDN_EOK on success

complete_code XHCI transfer complete status code

5.15. USBSSP_SetConfiguration

Function Declaration

```
uint32_t USBSSP_SetConfiguration(res, configValue);
```

USBSSP_DriverResourcesT *res

uint32_t configValue

Parameter List

res [in]

Driver resources

configValue [in]

USB device's configuration selector

`res` [in]
 driver resources

`configValue` [in]
 USB device's configuration selector

Description

Set configuration.

Function configures USBSSP controller as well as device connected to this USBSSP controller. This is a blocking function. This function must not be called from an interrupt context.

Return Value

CDN_EOK on success

`complete_code` XHCI transfer complete status code

CDN_EOK on success

`complete_code` XHCI transfer complete status code

5.16. USBSSP_NBControlTransfer

Function Declaration

```
uint32_t USBSSP_NBControlTransfer(res, setup, pdata, complete);

USBSSP_DriverResourceT *res

const CH9_UsbSetup *setup

const uint8_t *pdata

USBSSP_Complete complete
```

Parameter List

`res` [in]
 Driver resources

`setup` [in]
 Keeps setup packet

`pdata` [in]
 Pointer for data to send/receive \$RANGE \$NULLABLE \$

`complete` [in]

	Complete callback function
<code>res</code>	[in]
	driver resources
<code>setup</code>	[in]
	keeps setup packet
<code>pdata</code>	[in]
	pointer for data to send/receive
<code>complete_code</code>	[in]
	XHCI transfer complete status code

Description

No blocking control transfer.

Function executes control transfer. Information about transfer like: data direction, data length, wIndex, wValue etc. are passed in 'setup' parameter.

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

CDN_EOK on success

5.17. USBSSP_NoOpTest

Function Declaration

```
uint32_t USBSSP_NoOpTest(res, complete);
```

```
USBSSP_DriverResourceT *res
```

```
USBSSP_Complete complete
```

Parameter List

<code>res</code>	[in]
	Driver resources
<code>complete</code>	[in]
	Callback
<code>res</code>	[in]
	driver resources complete callback pointer

Description

No Operation test.

Function used for testing purposes: NO_OP_COMMAND is send to USBSSP controller. When event ring receives NO_OP_COMMAND complete it calls complete callback

Return Value

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if no errors

5.18. USBSSP_CalcFsLsEPIntrptInterval

Function Declaration

```
uint8_t USBSSP_CalcFsLsEPIntrptInterval(bInterval);
```

```
uint8_t bInterval
```

Parameter List

```
bInterval [in]
```

```
bInterval
```

```
bInterval [in]
```

Description

Calculate full/low speed endpoint interval based on bInterval See xHCI spec Section 6.2.3.6 for more details.

Return Value

value valid endpoint context interval value

valid endpoint context interval value

5.19. USBSSP_EnableSlot

Function Declaration

```
uint32_t USBSSP_EnableSlot(res);
```

```
USBSSP_DriverResourceT *res
```

Parameter List

```
res [in]
```

```
Driver resources
```

```
res [in]
```

driver resources

Description

Function enables slot for new connected device.

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

CDN_EOK if success, error code elsewhere

5.20. USBSSP_DisableSlot

Function Declaration

```
uint32_t USBSSP_DisableSlot(res);
```

```
USBSSP_DriverResourceT *res
```

Parameter List

```
res [in]
```

Driver resources

```
res [in]
```

driver resources

Description

Function disables slot of connected device.

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

CDN_EOK if success, error code elsewhere

5.21. USBSSP_EnableEndpoint

Function Declaration

```
uint32_t USBSSP_EnableEndpoint(res, desc);
```

```
USBSSP_DriverResourceT *res
```

```
const uint8_t *desc
```

Parameter List

`res` [in]
Driver resources

`desc` [in]
pointer to endpoint descriptor

`res` [in]
driver resources

`desc` [in]
endpoint descriptor

Description

Enable endpoint.

Function used in device context.

Return Value

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if no errors

CDN_EOK for success, error code elsewhere

5.22. USBSSP_DisableEndpoint

Function Declaration

```
uint32_t USBSSP_DisableEndpoint(res, epAddress);

USBSSP_DriverResourceT *res

uint8_t epAddress
```

Parameter List

`res` [in]
Driver resources

`epAddress` [in]
address of endpoint to be disabled

`res` [in]
driver resources

epAddress [in]

Endpoint address

Description

Disable endpoint.

Function used in device context.

Return Value

CDN_EINVAL when driver's settings doesn't suit to native platform settings

CDN_EOK if no errors

CDN_EOK for success, error code elsewhere

5.23. USBSSP_GetMicroFrameIndex

Function Declaration

```
uint32_t USBSSP_GetMicroFrameIndex(res, index);
```

USBSSP_DriverResourceT *res

uint32_t *index

Parameter List

res [in]

driver resources

index [out]

Micro Frame Index returned by function.

res []

driver resources

index []

pointer to memory where actual micro frame index will be stored

Description

Get actual frame number.

Function returns actual frame number on USB bus. Remember that maximal frame number can be 0x7FF. Next frame increments returned value from 0.

Return Value

CDN_EOK on success, CDN_EINVAL when any of input parameter is NULL

5.24. USBSSP_SetEndpointExtraFlag

Function Declaration

```
uint32_t USBSSP_SetEndpointExtraFlag(res, epIndex, flags);
```

```
USBSSP_DriverResourcesT *res
```

```
uint8_t epIndex
```

```
USBSSP_ExtraFlagsEnumT flags
```

Parameter List

<i>res</i>	[in]	driver resources
<i>epIndex</i>	[in]	endpoint index
<i>flags</i>	[in]	Endpoint Extra Flag
<i>res</i>	[in]	driver resources
<i>epIndex</i>	[in]	index of selected endpoint
<i>flag</i>	[out]	bitmap of flags to set, 1 on selected bit clears corresponding flag

Description

set end point extra flag

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

CDN_EOK if success, CDN_EINVAL when some input parameter is incorrect

5.25. USBSSP_CleanEndpointExtraFlag

Function Declaration

```
uint32_t USBSSP_CleanEndpointExtraFlag(res, epIndex, flags);
```

```
USBSSP_DriverResourceT *res
```

```
uint8_t epIndex
```

```
USBSSP_ExtraFlagsEnumT flags
```

Parameter List

```
res      [ in ]
```

driver resources

```
epIndex [ in ]
```

end point index

```
flags    [ in ]
```

Endpoint Extra Flag

```
res      [ in ]
```

driver resources

```
epIndex [ in ]
```

index of selected endpoint

```
flag     [ out ]
```

bitmap of flags to clear, 0 on selected bit clears corresponding flag

Description

clean endpoint extra flag

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

CDN_EOK if success, CDN_EINVAL when some input parameter is incorrect

5.26. USBSSP_GetEndpointExtraFlag

Function Declaration

```
uint32_t USBSSP_GetEndpointExtraFlag(res, epIndex, flag);
```

```
const USBSSP_DriverResourceT *res
```

```
uint8_t epIndex
```

```
uint8_t *flag
```

Parameter List

<code>res</code>	[in]	driver resources
<code>epIndex</code>	[in]	endpoint index
<code>flag</code>	[out]	Endpoint Extra Flag returned by pointer
<code>res</code>	[in]	driver resources
<code>epIndex</code>	[in]	index of selected endpoint
<code>flag</code>	[out]	pointer to memory where flags will be stored

Description

get endpoint extra flag

get endpoint extra flag

Return Value

CDN_EOK if no errors, CDN_EINVAL for incorrect input parameters

5.27. USBSSP_SetFrameID

Function Declaration

```
uint32_t USBSSP_SetFrameID(res, epIndex, frameID);
```

```
USBSSP_DriverResourcesT *res
```

```
uint8_t epIndex
```

```
uint32_t frameID
```

Parameter List

<code>res</code>	[in]	Driver resources
<code>epIndex</code>	[in]	

endpoint index
 frameID [in]
 Frame ID
 res [in]
 driver resources
 epIndex [in]
 index of selected endpoint
 frameID [in]
 value of frameID to set

Description

set Frame ID

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

5.28. USBSSP_AddEventDataTRB

Function Declaration

```
uint32_t USBSSP_AddEventDataTRB(res, epIndex, eventDataLo, eventDataHi, flags);
```

USBSSP_DriverResourcesT *res

uint8_t epIndex

uint32_t eventDataLo

uint32_t eventDataHi

uint32_t flags

Parameter List

res [in]
 Driver resources
 epIndex [in]
 endpoint index
 eventDataLo [in]

	event data Low
eventDataHi	[in]
	event data High
flags	[in]
	Flags
res	[]
	driver resources
epIndex	[]
	endpoint index
eventDataLo	[]
	event data low DWORD
eventDataHi	[]
	event data high DWORD
flags	[]
	extra flags for TRB

Description

Add event data TRB.

Return Value

value TBC

CDN_EOK for success, error code elsewhere

5.29. USBSSP_ForceHeader

Function Declaration

```
uint32_t USBSSP_ForceHeader(res, trbDwords, complete);
```

```
USBSSP_DriverResourcesT *res
```

```
const USBSSP_ForceHdrParams *trbDwords
```

```
USBSSP_Complete complete
```

Parameter List

res	[in]
-----	--------

	driver resources
trbDwords	[in]
	trb words
complete	[in]
	complete callback
res	[in]
	driver resources
trbDwords	[in]
complete	[]
	completion callback

Description

Force header.

Return Value

CDN_EINVAL Invalid Input parameters

CDN_EOK if no errors

CDN_EOK if success, error code elsewhere

5.30. USBSSP_SetPortOverrideReg

Function Declaration

```
uint32_t USBSSP_SetPortOverrideReg(res, regValue);
```

```
const USBSSP_DriverResourceT *res
```

```
uint32_t regValue
```

Parameter List

res	[in]
	driver resources
regValue	[in]
	Register value
res	[in]

driver resources

regValue [in]

memory place where port override register value will be stored

Description

set port override register

set port override register

Return Value

CDN_EOK if success, CDN_EINVAL when some input parameter is incorrect

5.31. USBSSP_GetPortOverrideReg

Function Declaration

```
uint32_t USBSSP_GetPortOverrideReg(res, regValue);
```

```
const USBSSP_DriverResourcesT *res
```

```
uint32_t *regValue
```

Parameter List

res [in]

Driver resources

regValue [out]

Register value

res [in]

driver resources

regValue [out]

memory place where port status/control register value will be stored

Description

Get port override register.

Get port override register.

Return Value

CDN_EOK if success, CDN_EINVAL when some input parameter is incorrect

5.32. USBSSP_SetPortControlReg

Function Declaration

```
uint32_t USBSSP_SetPortControlReg(res, portId, portRegIdx, regValue);
```

```
const USBSSP_DriverResourcesT *res
```

```
uint8_t portId
```

```
USBSSP_PortControlRegIdx portRegIdx
```

```
uint32_t regValue
```

Parameter List

res [in]

driver resources

portId [in]

port ID

portRegIdx [in]

port control register ID

regValue [in]

Register value

res [in]

driver resources

portId [in]

index of selected port

portRegIdx [in]

port register index

regValue [in]

memory place where port status/control register value will be stored

Description

set port control register

set port control register

Return Value

CDN_EOK if success, CDN_EINVAL when some input parameter is incorrect

5.33. USBSSP_GetPortControlReg

Function Declaration

```
uint32_t USBSSP_GetPortControlReg(res, portId, portRegIdx, regValue);
```

```
const USBSSP_DriverResourcesT *res
```

```
uint8_t portId
```

```
USBSSP_PortControlRegIdx portRegIdx
```

```
uint32_t *regValue
```

Parameter List

res [in]

Driver resources

portId [in]

port ID

portRegIdx [in]

port control register ID

regValue [out]

Register value

res [in]

driver resources

portId [in]

index of selected port

portRegIdx [in]

port register index

regValue [out]

memory place where port status/control register value will be stored

Description

Get port control register.

Get port control register.

Return Value

CDN_EOK if success, CDN_EINVAL when some input parameter is incorrect

5.34. USBSSP_SaveState

Function Declaration

```
uint32_t USBSSP_SaveState(res, drvContext);
```

```
USBSSP_DriverResourcesT *res
```

```
USBSSP_DriverContextT *drvContext
```

Parameter List

res	[in]
	Driver resources
drvContext	[in]
	Pointer to driver context struct
res	[]
	driver resources
drvContext	[]
	driver context

Description

Save state and stop xHC.

Save state and stop xHC.

Return Value

CDN_EOK on success, error code elsewhere

5.35. USBSSP_RestoreState

Function Declaration

```
uint32_t USBSSP_RestoreState(res, drvContext);
```

```
USBSSP_DriverResourcesT *res
```

```
const USBSSP_DriverContextT *drvContext
```

Parameter List

res	[in]
	Driver resources
drvContext	[in]

Pointer to driver context struct

`res` `[]`

driver resources

`drvContext` `[]`

driver context

Description

Restore state and start xHC.

Restore state and start xHC.

Return Value

CDN_EOK on success, error code elsewhere

5.36. USBSSP_GetPortConnected

Function Declaration

```
uint32_t USBSSP_GetPortConnected(res, connected);

const USBSSP_DriverResourcesT *res

uint8_t *connected
```

Parameter List

`res` `[in]`

Driver resources

`connected` `[out]`

connected status

`res` `[in]`

driver resources

`connected` `[out]`

status of port connection

Description

Get connected status.

Get connected status.

Return Value

CDN_EOK if success, CDN_EINVAL when some input parameter is incorrect

5.37. USBSSP_GetDevAddressState

Function Declaration

```
uint32_t USBSSP_GetDevAddressState(res, addrState);

const USBSSP_DriverResourcesT *res

uint8_t *addrState
```

Parameter List

res	[in]	Driver resources
addrState	[out]	dev address state
res	[in]	driver resources
addrState	[out]	dev address state

Description

Get dev address state.

Get dev address state.

Return Value

CDN_EOK if success, CDN_EINVAL when some input parameter is incorrect

5.38. CUSBD_Probe

Function Declaration

```
uint32_t CUSBD_Probe(config, sysReqCusbd);

const CUSBD_Config *config

CUSBD_SysReq *sysReqCusbd
```

Parameter List

config	[in]
--------	--------

driver/hardware configuration required

`sysReqCusbd` [out]

returns the size of memory allocations required

Description

Obtain the private memory size required by the driver.

Return Value

0 on success (requirements structure filled)

CDN_ENOTSUP if configuration cannot be supported due to driver/hardware constraints

5.39. CUSBD_Init

Function Declaration

```
uint32_t CUSBD_Init(pD, config, callbacks);
```

`CUSBD_PrivateData` **pD*

`const CUSBD_Config` **config*

`const CUSBD_Callbacks` **callbacks*

Parameter List

`pD` [out]

driver state info specific to this instance

`config` [inout]

specifies driver/hardware configuration

`callbacks` [in]

client-supplied callback functions

`pD` [in]

driver state info specific to this instance

`config` [inout]

specifies driver/hardware configuration

`callbacks` [in]

client-supplied callback functions

Description

Initialize the driver instance and state, configure the USB device as specified in the 'config' settings, initialize locks used by the driver.

Return Value

CDN_EOK on success

CDN_ENOTSUP if hardware has an inconsistent configuration or doesn't support feature(s) required by 'config' parameters

CDN_ENOENT if insufficient locks were available (i.e. something allocated locks between probe and init)

CDN_EIO if driver encountered an error accessing hardware

CDN_EINVAL if illegal/inconsistent values in 'config'

CDN_EOK on success

CDN_ENOTSUP if hardware has an inconsistent configuration or doesn't support feature(s) required by 'config' parameters

CDN_ENOENT if insufficient locks were available (i.e. something allocated locks between probe and init)

CDN_EIO if driver encountered an error accessing hardware

CDN_EINVAL if illegal/inconsistent values in 'config'

5.40. CUSBD_Destroy

Function Declaration

```
uint32_t CUSBD_Destroy(pD);

const CUSBD_PrivateData *pD
```

Parameter List

pD [in]

driver state info specific to this instance

Description

Destroy the driver (automatically performs a stop)

Return Value

CDN_EOK on success

CDN_EINVAL if illegal/inconsistent values in 'config'

5.41. CUSBD_Start

Function Declaration

```
uint32_t CUSBD_Start(pD);
```

```
CUSBD_PrivateData *pD
```

Parameter List

pD [in]

driver state info specific to this instance

Description

Start the USB driver.

Return Value

CDN_EOK on success

CDN_EINVAL if illegal/inconsistent values in 'config'

5.42. CUSBD_Stop

Function Declaration

```
uint32_t CUSBD_Stop(pD);
```

```
CUSBD_PrivateData *pD
```

Parameter List

pD [in]

driver state info specific to this instance

pD [in]

driver state info specific to this instance

Description

Stop the driver.

This should disable the hardware, including its interrupt at the source, and make a best-effort to cancel any pending transactions.

Return Value

CDN_EOK on success

CDN_EINVAL if any input parameter is NULL

CDN_EOK on success

CDN_EINVAL if illegal/inconsistent values in 'config'

5.43. CUSBD_Isr

Function Declaration

```
uint32_t CUSBD_Isr(pD);
```

```
CUSBD_PrivateData *pD
```

Parameter List

pD [in]

driver state info specific to this instance

pD [in]

driver state info specific to this instance

Description

Driver ISR.

Platform-specific code is responsible for ensuring this gets called when the corresponding hardware's interrupt is asserted. Registering the ISR should be done after calling init, and before calling start. The driver's ISR will not attempt to lock any locks, but will perform client callbacks. If the client wishes to defer processing to non-interrupt time, it is responsible for doing so.

Return Value

CDN_EOK on success, error otherwise

5.44. CUSBD_EpEnable

Function Declaration

```
uint32_t CUSBD_EpEnable(pD, ep, desc);
```

```
CUSBD_PrivateData *pD
```

```
CUSBD_Ep *ep
```

```
const uint8_t *desc
```

Parameter List

pD [in]

driver state info specific to this instance

ep [in]

endpoint being configured

desc [in]

endpoint descriptor

Description

Enable Endpoint.

This function should be called within SET_CONFIGURATION(configNum > 0) request context. It configures required hardware controller endpoint with parameters given in desc.

Return Value

0 on success

CDN_EINVAL if pD, ep or desc is NULL or desc is not a endpoint descriptor

5.45. CUSBD_EpDisable

Function Declaration

```
uint32_t CUSBD_EpDisable(pD, ep);
```

CUSBD_PrivateData *pD

CUSBD_Ep *ep

Parameter List

pD [in]

driver state info specific to this instance

ep [in]

endpoint being unconfigured

Description

Disable Endpoint.

Functions unconfigures hardware endpoint. Endpoint will not respond to any host packets. This function should be called within SET_CONFIGURATION(configNum = 0) request context or on disconnect event. All queued requests on endpoint are completed with CDN_ECANCELED status.

Return Value

0 on success

CDN_EINVAL if pD or ep is NULL

5.46. CUSBD_EpSetHalt

Function Declaration

```
uint32_t CUSBD_EpSetHalt(pD, ep, value);
```

```
CUSBD_PrivateData *pD
```

```
const CUSBD_Ep *ep
```

```
uint8_t value
```

Parameter List

`pD` [in]

driver state info specific to this instance

`ep` [in]

endpoint being setting or clearing halt state

`value` [in]

if 1 sets halt, if 0 clears halt on endpoint

Description

Set halt or clear halt state on endpoint.

When setting halt, device will respond with STALL packet to each host packet. When clearing halt, endpoint returns to normal operating.

Return Value

0 on success

CDN_EPERM if endpoint is disabled (has not been configured yet)

CDN_EINVAL if `pD` or `ep` is NULL

5.47. CUSBD_EpSetWedge

Function Declaration

```
uint32_t CUSBD_EpSetWedge(pD, ep);
```

```
CUSBD_PrivateData *pD
```

```
const CUSBD_Ep *ep
```

Parameter List

`pD` [in]

driver state info specific to this instance

`ep` [in]

endpoint being setting halt state

Description

Set halt on endpoint.

Function sets endpoint to permanent halt state. State can not be changed even with `epSetHalt(pD, ep, 0)` function. Endpoint returns automatically to its normal state on disconnect event.

Return Value

0 on success

CDN_EPERM if endpoint is disabled (has not been configured yet)

CDN_EINVAL if `pD` or `ep` is NULL

5.48. CUSBD_EpFifoStatus

Function Declaration

```
uint32_t CUSBD_EpFifoStatus(pD, ep);
```

```
const CUSBD_PrivateData *pD
```

```
const CUSBD_Ep *ep
```

Parameter List

`pD` [in]

driver state info specific to this instance

`ep` [in]

endpoint which status is to be returned

Description

Returns number of bytes in hardware endpoint FIFO.

Function useful in application where exact number of data bytes is required. In some situation software higher layers must be aware of number of data bytes issued but not transferred by hardware because of aborted transfers, for example on disconnect event. *

Return Value

number of bytes in hardware endpoint FIFO

CDN_EINVAL if `pD` or `ep` is NULL

5.49. CUSBD_EpFifoFlush

Function Declaration

```
uint32_t CUSBD_EpFifoFlush(pD, ep);
```

```
CUSBD_PrivateData *pD
```

```
const CUSBD_Ep *ep
```

Parameter List

`pD` [in]

driver state info specific to this instance

`ep` [in]

endpoint which FIFO is to be flushed

Description

Flush hardware endpoint FIFO.

5.50. CUSBD_ReqQueue

Function Declaration

```
uint32_t CUSBD_ReqQueue(pD, ep, req);
```

```
CUSBD_PrivateData *pD
```

```
const CUSBD_Ep *ep
```

```
CUSBD_Req *req
```

Parameter List

`pD` [in]

driver state info specific to this instance

`ep` [in]

endpoint associated with the request

`req` [in]

request being submitted

Description

Submits IN/OUT transfer request to an endpoint.

Return Value

0 on success

CDN_EPROTO only on default endpoint if endpoint is not in data stage phase

CDN_EPERM if endpoint is not enabled

CDN_EINVAL if pD, ep, or req is NULL

5.51. CUSBD_ReqDequeue

Function Declaration

```
uint32_t CUSBD_ReqDequeue(pD, ep, req);
```

CUSBD_PrivateData *pD

CUSBD_Ep *ep

CUSBD_Req *req

Parameter List

pD [in]

driver state info specific to this instance

ep [in]

endpoint associated with the request

req [in]

request being dequeued

Description

Dequeues IN/OUT transfer request from an endpoint.

Function completes all queued request with CDN_ECANCELED status.

Return Value

0 on success

CDN_EINVAL if pD or ep is NULL

5.52. CUSBD_GetDevInstance

Function Declaration

```
void CUSBD_GetDevInstance(pD, *dev);
```

CUSBD_PrivateData *pD

CUSBD_Dev **dev

Parameter List

pD [in]

driver state info specific to this instance

dev [out]

returns pointer to CUSBD instance

Description

Returns pointer to CUSBD object.

CUSBD object is a logical representation of USB device. CUSBD contains endpoint collection accessed through epList field. Endpoints collection is organized as double linked list.

5.53. CUSBD_DGetFrame

Function Declaration

```
uint32_t CUSBD_DGetFrame(pD, numOfFrame);
```

```
CUSBD_PrivateData *pD
```

```
uint32_t *numOfFrame
```

Parameter List

pD [in]

driver state info specific to this instance

numOfFrame [out]

returns number of USB frame

Description

Returns number of frame.

Some controllers have counter of SOF packets or ITP packets in the Super Speed case. Function returns value of this counter. This counter can be used for time measurement. Single FS frame is 1 ms measured, for HS and SS is 125us.

Return Value

CDN_EOK on success

CDN_EOPNOTSUPP if feature is not supported

CDN_EINVAL if pD or numOfFrame is NULL

5.54. CUSBD_DSetSelfpowered

Function Declaration

```
uint32_t CUSBD_DSetSelfpowered(pD);
```

```
const CUSBD_PrivateData *pD
```

Parameter List

pD [in]

driver state info specific to this instance

Description

Sets the device self powered feature.

Return Value

CDN_EOK on success

CDN_EOPNOTSUPP if feature is not supported

CDN_EINVAL if pD is NULL

5.55. CUSBD_DClearSelfpowered

Function Declaration

```
uint32_t CUSBD_DClearSelfpowered(pD);
```

```
const CUSBD_PrivateData *pD
```

Parameter List

pD [in]

driver state info specific to this instance

Description

Clear the device self powered feature.

Return Value

CDN_EOK on success

CDN_EOPNOTSUPP if feature is not supported

CDN_EINVAL if pD is NULL

5.56. CUSBD_DGetConfigParams

Function Declaration

```
uint32_t CUSBD_DGetConfigParams(pD, configParams);
```

```
const CUSBD_PrivateData *pD
```

```
CH9_ConfigParams *configParams
```

Parameter List

pD [in]

driver state info specific to this instance

`configParams` [out]

pointer to `CH9_ConfigParams` structure

Description

Returns configuration parameters: U1 exit latency and U2 exit latency Function useful only in Super Speed mode.

Return Value

`CDN_EOK` on success

`CDN_EINVAL` if illegal/inconsistent values in 'config'

5.57. CUSBDMA_Probe

Function Declaration

```
uint32_t CUSBDMA_Probe(config, sysReq);
```

```
const CUSBDMA_Config *config
```

```
CUSBDMA_SysReq *sysReq
```

Parameter List

`config` [in]

driver/hardware configuration required

`sysReq` [out]

`sysReq` returns the size of memory allocations required

`config` []

pointer to configuration structure

`sysReq` []

pointer to structure where DMA driver memory requirements will be stored

Description

Obtain the private memory size required by the driver.

Return Value

`CDN_EOK` on success

`CDN_EINVAL` if function parameters are not valid

CDN_ENOTSUP if configuration cannot be supported due to driver/hardware constraints

CDN_EOK on success, error code elsewhere

5.58. CUSBDMA_Init

Function Declaration

```
uint32_t CUSBDMA_Init(pD, config);
```

```
CUSBDMA_DmaController *pD
```

```
const CUSBDMA_Config *config
```

Parameter List

<i>pD</i>	[in]
	driver state info specific to this instance
<i>config</i>	[in]
	specifies driver/hardware configuration
<i>pD</i>	[]
	pointer to DMA controller object
<i>config</i>	[]
	pointer to configuration structure

Description

Initialize the driver instance and state, configure the USB device as specified in the 'config' settings, initialize locks used by the driver.

Return Value

CDN_EOK on success

CDN_EINVAL if function parameters are not valid

CDN_ENOTSUP if configuration cannot be supported due to driver/hardware constraints

CDN_EIO if driver encountered an error accessing hardware

CDN_ENOENT insufficient locks were available (i.e. something allocated locks between probe and init)

CDN_EOK on success, error code elsewhere

5.59. CUSBDMA_Destroy

Function Declaration

```
uint32_t CUSBDMA_Destroy(pD);
```

```
CUSBDMA_DmaController *pD
```

Parameter List

pD [in]

driver state info specific to this instance

pD []

pointer to DMA controller object

Description

Destroy the driver (automatically performs a stop).

Destroy the driver (automatically performs a stop).

Return Value

CDN_EOK on success, error code elsewhere

5.60. CUSBDMA_ChannelAlloc

Function Declaration

```
uint32_t CUSBDMA_ChannelAlloc(pD, *channelPtr, channelParams);
```

```
CUSBDMA_DmaController *pD
```

```
CUSBDMA_DmaChannel **channelPtr
```

```
CUSBDMA_ChannelParams *channelParams
```

Parameter List

pD [in]

driver state info specific to this instance

channelPtr [out]

address of channel pointer

channelParams [in]

Channel parameters

pD []

pointer to DMA controller object

channelPtr []

pointer to memory where pointer to channel is allocated

channelParams []
 channel parameters

Description

Allocation the DMA channel.

Allocation the DMA channel.

Return Value

CDN_EOK on success, error code elsewhere

5.61. CUSBDMA_ChannelReset

Function Declaration

```
uint32_t CUSBDMA_ChannelReset(pD, channel);
```

CUSBDMA_DmaController *pD

CUSBDMA_DmaChannel *channel

Parameter List

pD [in]
 driver state info specific to this instance

channel [in]
 channel pointer to DMA channel

pD []
 pointer to DMA controller object

channel []
 Pointer to DMA channel

Description

Reset the DMA channel.

Return Value

CDN_EOK on success

CDN_EINVAL if function parameters are not valid

CDN_EPERM if the DMA is active and channel cannot be reset

CDN_EOK on success, error code elsewhere

5.62. CUSBDMA_ChannelRelease

Function Declaration

```
uint32_t CUSBDMA_ChannelRelease(pD, channel);
```

```
CUSBDMA_DmaController *pD
```

```
CUSBDMA_DmaChannel *channel
```

Parameter List

<i>pD</i>	[in]	driver state info specific to this instance
<i>channel</i>	[in]	channel pointer to DMA channel
<i>pD</i>	[]	Pointer to private data of DMA controller
<i>channel</i>	[]	Pointer to DMA channel

Description

Release the DMA channel.

Return Value

CDN_EOK on success

CDN_EINVAL if function parameters are not valid

CDN_EOK on success, error code elsewhere

5.63. CUSBDMA_ChannelProgram

Function Declaration

```
uint32_t CUSBDMA_ChannelProgram(pD, channel, params);
```

```
CUSBDMA_DmaController *pD
```

```
CUSBDMA_DmaChannel *channel
```

```
const CUSBDMA_DmaTransferParam *params
```

Parameter List

<i>pD</i>	[in]
-----------	--------

driver state info specific to this instance

`channel` [in]
channel pointer to DMA channel for which transfer will be started

`params` [in]
transfer parameters container

`pD` []
Pointer to private data of DMA controller

`channel` []
Pointer to DMA channel

`params` []
transfer parameters

Description

Prepares transfer and starts it.

Return Value

CDN_EOK on success

CDN_EINVAL if function parameters are not valid

CDN_ENOMEM if DMA channel has no free memory for TRB

CDN_EOK on success, error code elsewhere

5.64. CUSBDMA_ChannelTrigger

Function Declaration

```
uint32_t CUSBDMA_ChannelTrigger(pD, channel);
```

CUSBDMA_DmaController **pD*

CUSBDMA_DmaChannel **channel*

Parameter List

`pD` [in]
driver state info specific to this instance

`channel` [in]
pointer to DMA channel which needs to be triggered

`pD` `[]`
 Pointer to private data of DMA controller

`channel` `[]`
 Pointer to DMA channel

Description

Triggers DMA transfer for given DMA channel if TRBs are queued.

Return Value

CDN_EOK on successful trigger or if the DMA is already active

CDN_EINVAL if function parameters are not valid

CDN_ENOTSUP if TRBs are not queued

CDN_EOK on success, error code elsewhere

5.65. CUSBDMA_ChannelUpdateState

Function Declaration

```
uint32_t CUSBDMA_ChannelUpdateState(pD, channel);
```

CUSBDMA_DmaController **pD*

CUSBDMA_DmaChannel **channel*

Parameter List

`pD` `[in]`
 driver state info specific to this instance

`channel` `[in]`
 pointer to DMA channel whose state needs to be updated

`pD` `[]`
 Pointer to private data of DMA controller

`channel` `[]`
 Pointer to DMA channel

Description

Updates the data transfer status of a channel.

Return Value

CDN_EOK on successful trigger or if the DMA is already active

CDN_EINVAL if function parameters are not valid

CDN_EOK on success, error code elsewhere

5.66. CUSBDMA_ChannelSetMaxPktSz

Function Declaration

```
uint32_t CUSBDMA_ChannelSetMaxPktSz(pD, channel, maxPacketSize);
```

CUSBDMA_DmaController **pD*

CUSBDMA_DmaChannel **channel*

uint16_t *maxPacketSize*

Parameter List

<i>pD</i>	[in]	driver state info specific to this instance
<i>channel</i>	[in]	pointer to DMA channel whose state needs to be updated
<i>maxPacketSize</i>	[in]	Value of max packet size
<i>pD</i>	[]	Pointer to private data of DMA controller
<i>channel</i>	[]	Pointer to DMA channel
<i>maxPacketSize</i>	[]	max packet size

Description

Updates the max packet size for a channel.

Return Value

CDN_EOK on successful successful update of max packet size

CDN_EINVAL if function parameters are not valid

CDN_EOK on success, error code elsewhere

5.67. CUSBDMA_ChannelHandleStall

Function Declaration

```
uint32_t CUSBDMA_ChannelHandleStall(pD, channel, value, timeout);
```

CUSBDMA_DmaController **pD*

CUSBDMA_DmaChannel **channel*

uint32_t *value*

uint32_t *timeout*

Parameter List

<i>pD</i>	[in]	driver state info specific to this instance
<i>channel</i>	[in]	pointer to DMA channel whose stall state is handled
<i>value</i>	[in]	Clear stall if 0, else set stall
<i>timeout</i>	[in]	Timeout for waiting for flush operation while stalling
<i>pD</i>	[]	Pointer to private data of DMA controller
<i>channel</i>	[]	Pointer to DMA channel
<i>value</i>	[]	
<i>timeout</i>	[]	

Description

Set or Clear channel stall.

Return Value

CDN_EOK on successful trigger or if the DMA is already active

CDN_EINVAL if function parameters are not valid

CDN_EOK on success, error code elsewhere

5.68. CUSBDMA_ChannelFreeHeadTrbChain

Function Declaration

```
uint32_t CUSBDMA_ChannelFreeHeadTrbChain(pD, channel);
```

```
const CUSBDMA_DmaController *pD
```

```
CUSBDMA_DmaChannel *channel
```

Parameter List

pD [in]

driver state info specific to this instance

channel [in]

pointer to DMA channel whose descriptor needs to be freed

pD []

pointer to DMA controller object

channel []

pointer to channel

Description

Free the head(oldest) TRB chain descriptor for this channel.

Return Value

CDN_EOK on success

CDN_EINVAL if function parameters are not valid

CDN_EOK on success, error code elsewhere

5.69. USB_SSP_DRD_Probe

Function Declaration

```
uint32_t USB_SSP_DRD_Probe(sysReq);
```

```
USB_SSP_DRD_SysReq *sysReq
```

Parameter List

sysReq [out]

Returns the memory requirements for given configuration.

Description

Returns the memory requirements for a driver instance.

Return Value

CDN_EOK On success.

CDN_EINVAL If config contains invalid values or not supported.

5.70. USB_SSP_DRD_Init

Function Declaration

```
uint32_t USB_SSP_DRD_Init(privData, config, callbacks);
```

```
USB_SSP_DRD_PrivData *privData
```

```
const USB_SSP_DRD_Config *config
```

```
const USB_SSP_DRD_Callbacks *callbacks
```

Parameter List

privData [in]

Pointer to driver's private data object.

config [in]

Specifies driver/hardware configuration.

callbacks [in]

Event Handlers and Callbacks.

Description

Instantiates the USB_SSP_DRD Core Driver, given the required blocks of memory (this includes initializing the instance and the underlying hardware).

If a client configuration is required (likely to always be true), it is passed in also. Returns an instance pointer, which the client must maintain and pass to all other driver functions. (except probe).

Return Value

CDN_EOK On success

CDN_EINVAL If illegal/inconsistent values in 'config' doesn't support feature(s) required by 'config' parameters.

CDN_EIO if operation failed

5.71. USB_SSP_DRD_Isr

Function Declaration

```
uint32_t USB_SSP_DRD_Isr(privData);
```

```
USB_SSP_DRD_PrivData *privData
```

Parameter List

```
privData [in]
```

Pointer to driver's private data object filled by init.

Description

USB_SSP_DRD Core Driver's ISR.

Platform-specific code is responsible for ensuring this gets called when the corresponding hardware's interrupt is asserted. Registering the ISR should be done after calling init, and before calling start. The driver's ISR will not attempt to lock any locks, but will perform client callbacks. If the client wishes to defer processing to non- interrupt time, it is responsible for doing so. This function must not be called after calling destroy and releasing private data memory.

Return Value

CDN_EOK on success

CDN_EINVAL if input parameters are invalid

5.72. USB_SSP_DRD_Start

Function Declaration

```
uint32_t USB_SSP_DRD_Start(privData);
```

```
USB_SSP_DRD_PrivData *privData
```

Parameter List

```
privData [in]
```

Pointer to driver's private data object.

Description

Start the USB_SSP_DRD driver, enabling interrupts.

This is called after the client has successfully initialized the driver and hooked the driver's ISR (the isr member of this struct) to the IRQ.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid

5.73. USB_SSP_DRD_Stop

Function Declaration

```
uint32_t USB_SSP_DRD_Stop(privData);

const USB_SSP_DRD_PrivData *privData
```

Parameter List

privData [in]

Pointer to driver's private data object.

Description

The client may call this to disable the hardware (disabling its IRQ at the source and disconnecting it if applicable).

Also, a best- effort is made to cancel any pending transactions.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid

5.74. USB_SSP_DRD_Destroy

Function Declaration

```
uint32_t USB_SSP_DRD_Destroy(privData);

const USB_SSP_DRD_PrivData *privData
```

Parameter List

privData [in]

Pointer to driver's private data object.

Description

This performs an automatic stop and then de-initializes the driver.

The client may not make further requests on this instance.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid

5.75. USB_SSP_DRD_CheckIfReady

Function Declaration

```
uint32_t USB_SSP_DRD_CheckIfReady(privData, isReady);
```

```
USB_SSP_DRD_PrivData *privData
```

```
bool *isReady
```

Parameter List

```
privData [ in ]
```

Pointer to driver's private data object.

```
isReady [ out ]
```

Will tell if controller is ready.

Description

Checks if controller is ready.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid

5.76. USB_SSP_DRD_CheckStrapMode

Function Declaration

```
uint32_t USB_SSP_DRD_CheckStrapMode(privData, strapMode);
```

```
USB_SSP_DRD_PrivData *privData
```

```
USB_SSP_DRD_Mode *strapMode
```

Parameter List

```
privData [ in ]
```

Pointer to driver's private data object.

```
strapMode [ out ]
```

Will tell how controller is strapped.

Description

Checks strap mode of controller.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid

5.77. USB_SSP_DRD_CheckOperationMode

Function Declaration

```
uint32_t USB_SSP_DRD_CheckOperationMode(privData, operationMode);
```

```
const USB_SSP_DRD_PrivData *privData
```

```
USB_SSP_DRD_Mode *operationMode
```

Parameter List

privData [in]

Pointer to driver's private data object.

operationMode [out]

Will tell operation mode of controller.

Description

Checks current operation mode of controller.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid

5.78. USB_SSP_DRD_SetOperationMode

Function Declaration

```
uint32_t USB_SSP_DRD_SetOperationMode(privData, USB_SSP_DRD_Mode operationMode);
```

```
USB_SSP_DRD_PrivData *privData
```

```
const USB_SSP_DRD_Mode operationMode
```

Parameter List

<code>privData</code>	[in]
Pointer to driver's private data object.	
<code>operationMode</code>	[in]
Mode of operation that should be set for controller.	

Description

Sets operation mode of controller.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid

5.79. USB_SSP_DRD_CheckIrq

Function Declaration

```
uint32_t USB_SSP_DRD_CheckIrq(privData, interruptVect);

const USB_SSP_DRD_PrivData *privData

uint32_t *interruptVect
```

Parameter List

<code>privData</code>	[in]
Pointer to driver's private data object.	
<code>interruptVect</code>	[out]
Interrupt Vector value.	

Description

Checks if there is an unhandled interrupt pending.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid

5.80. USB_SSP_DRD_ProcessIrq

Function Declaration

```
uint32_t USB_SSP_DRD_ProcessIrq(privData);
```

```
USB_SSP_DRD_PrivData *privData
```

Parameter List

```
privData [in]
```

Pointer to driver's private data object.

Description

This function will process an interrupt that is pending.

Return Value

CDN_EOK on success

CDN_EIO if operation failed

CDN_EINVAL if input parameters are invalid