



**EZ-USB™ FX3 Firmware Library  
API Reference Guide  
Version 1.3.4**

Cypress Semiconductor

198 Champion Court

San Jose, CA 95134-1709

Phone (USA): 800.858.1810

Phone (Intl): 408.943.2600

<http://www.cypress.com>

Copyright © 2010-2018 Cypress Semiconductor Corporation. All rights reserved.

EZ-USB™ , FX3, FX3S, SD3, CX3 and GPIF are trademarks of Cypress Semiconductor. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

The information in this document is subject to change without notice and should not be construed as a commitment by Cypress. While reasonable precautions have been taken, Cypress assumes no responsibility for any errors that may appear in this document. No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Cypress. Made in the U.S.A.

#### **Disclaimer**

CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

#### **License Agreement**

Please read the license agreement during SDK installation.

# Contents

<b>1 EZ-USB FX3 Api Reference Guide</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.2 EZ-USB FX3 SDK . . . . .	1
1.2.1 FX3 API Libraries . . . . .	2
1.2.2 FX3 Boot API Library . . . . .	2
1.2.3 FX3 Application Examples . . . . .	3
1.3 Embedded Real Time Operating System . . . . .	3
1.4 DMA Management . . . . .	3
1.4.1 DMA Sockets . . . . .	3
1.4.2 DMA Buffers . . . . .	4
1.4.3 DMA Descriptors . . . . .	4
1.4.4 DMA Channels . . . . .	4
1.5 Logging Support . . . . .	6
1.6 USB Driver and API . . . . .	7
1.6.1 USB Peripheral (Device) Mode . . . . .	7
1.6.2 USB 2.0 Host Mode . . . . .	8
1.6.3 USB On-The-Go (OTG) Mode . . . . .	8
1.7 Serial Peripheral Interfaces . . . . .	8
1.7.1 UART Interface . . . . .	9
1.7.2 I2C interface . . . . .	9
1.7.3 SPI interface . . . . .	9
1.7.4 I2S interface . . . . .	10
1.7.5 General Purpose IO (GPIO) Support . . . . .	10
1.8 GPIF II Driver and API . . . . .	10
1.8.1 GPIF Configuration . . . . .	11
1.8.2 GPIF-II Resources . . . . .	11
1.8.3 GPIF State Machine Control . . . . .	12
1.9 Storage Driver and API . . . . .	12
1.10 MIPI-CSI2 and Fixed Function GPIF Interface for CX3 . . . . .	13
1.11 Memory Allocation . . . . .	13
1.11.1 Driver Heap . . . . .	13

1.11.2 Buffer Heap . . . . .	14
1.11.3 Memory Leak Checking . . . . .	14
1.11.4 Memory Corruption Detection . . . . .	15
<b>2 Data Structure Index</b>	<b>17</b>
2.1 Data Structures . . . . .	17
<b>3 File Index</b>	<b>21</b>
3.1 File List . . . . .	21
<b>4 Data Structure Documentation</b>	<b>25</b>
4.1 CyFx3BootDmaDescriptor_t Struct Reference . . . . .	25
4.1.1 Detailed Description . . . . .	25
4.1.2 Field Documentation . . . . .	25
4.1.2.1 buffer . . . . .	25
4.1.2.2 chain . . . . .	25
4.1.2.3 size . . . . .	25
4.1.2.4 sync . . . . .	25
4.2 CyFx3BootDmaSocket_t Struct Reference . . . . .	26
4.2.1 Detailed Description . . . . .	26
4.2.2 Field Documentation . . . . .	26
4.2.2.1 dscrChain . . . . .	26
4.2.2.2 intr . . . . .	26
4.2.2.3 intrMask . . . . .	26
4.2.2.4 status . . . . .	26
4.2.2.5 xferCount . . . . .	26
4.2.2.6 xferSize . . . . .	27
4.3 CyFx3BootDmaSockRegs_t Struct Reference . . . . .	27
4.3.1 Detailed Description . . . . .	27
4.3.2 Field Documentation . . . . .	27
4.3.2.1 actBuffer . . . . .	27
4.3.2.2 actChain . . . . .	27
4.3.2.3 actSize . . . . .	27
4.3.2.4 actSync . . . . .	28
4.3.2.5 dscrChain . . . . .	28
4.3.2.6 intr . . . . .	28
4.3.2.7 intrMask . . . . .	28
4.3.2.8 sckEvent . . . . .	28
4.3.2.9 status . . . . .	28
4.3.2.10 unused19 . . . . .	28
4.3.2.11 unused2 . . . . .	28

4.3.2.12	xferCount . . . . .	28
4.3.2.13	xferSize . . . . .	28
4.4	CyFx3BootGpioSimpleConfig_t Struct Reference . . . . .	28
4.4.1	Detailed Description . . . . .	29
4.4.2	Field Documentation . . . . .	29
4.4.2.1	driveHighEn . . . . .	29
4.4.2.2	driveLowEn . . . . .	29
4.4.2.3	inputEn . . . . .	29
4.4.2.4	intrMode . . . . .	29
4.4.2.5	outValue . . . . .	29
4.5	CyFx3BootI2cConfig_t Struct Reference . . . . .	29
4.5.1	Detailed Description . . . . .	30
4.5.2	Field Documentation . . . . .	30
4.5.2.1	bitRate . . . . .	30
4.5.2.2	busTimeout . . . . .	30
4.5.2.3	dmaTimeout . . . . .	30
4.5.2.4	isDma . . . . .	30
4.6	CyFx3BootI2cPreamble_t Struct Reference . . . . .	30
4.6.1	Detailed Description . . . . .	31
4.6.2	Field Documentation . . . . .	31
4.6.2.1	buffer . . . . .	31
4.6.2.2	ctrlMask . . . . .	31
4.6.2.3	length . . . . .	32
4.7	CyFx3BootIoMatrixConfig_t Struct Reference . . . . .	32
4.7.1	Detailed Description . . . . .	32
4.7.2	Field Documentation . . . . .	32
4.7.2.1	gpioSimpleEn . . . . .	32
4.7.2.2	isDQ32Bit . . . . .	32
4.7.2.3	useI2C . . . . .	32
4.7.2.4	useI2S . . . . .	33
4.7.2.5	useSpi . . . . .	33
4.7.2.6	useUart . . . . .	33
4.8	CyFx3BootPibClock_t Struct Reference . . . . .	33
4.8.1	Detailed Description . . . . .	33
4.8.2	Field Documentation . . . . .	33
4.8.2.1	clkDiv . . . . .	33
4.8.2.2	clkSrc . . . . .	33
4.8.2.3	isDIIEnable . . . . .	34
4.8.2.4	isHalfDiv . . . . .	34
4.9	CyFx3BootSpiConfig_t Struct Reference . . . . .	34

---

4.9.1	Detailed Description	34
4.9.2	Field Documentation	34
4.9.2.1	clock	34
4.9.2.2	cpha	34
4.9.2.3	cpol	35
4.9.2.4	isLsbFirst	35
4.9.2.5	lagTime	35
4.9.2.6	leadTime	35
4.9.2.7	ssnCtrl	35
4.9.2.8	ssnPol	35
4.9.2.9	wordLen	35
4.10	CyFx3BootUartConfig_t Struct Reference	35
4.10.1	Detailed Description	36
4.10.2	Field Documentation	36
4.10.2.1	baudRate	36
4.10.2.2	flowCtrl	36
4.10.2.3	isDma	36
4.10.2.4	parity	36
4.10.2.5	rxEnable	36
4.10.2.6	stopBit	36
4.10.2.7	txEnable	36
4.11	CyFx3BootUsbEp0Pkt_t Struct Reference	37
4.11.1	Detailed Description	37
4.11.2	Field Documentation	37
4.11.2.1	bIdx0	37
4.11.2.2	bIdx1	37
4.11.2.3	bmReqType	37
4.11.2.4	bReq	37
4.11.2.5	bVal0	37
4.11.2.6	bVal1	37
4.11.2.7	pData	38
4.11.2.8	wLen	38
4.12	CyFx3BootUsbEpConfig_t Struct Reference	38
4.12.1	Detailed Description	38
4.12.2	Field Documentation	38
4.12.2.1	burstLen	38
4.12.2.2	enable	38
4.12.2.3	epType	38
4.12.2.4	isoPkts	38
4.12.2.5	pcktSize	39

4.12.2.6	streams	39
4.13	CyU3PCardCtxt Struct Reference	39
4.13.1	Detailed Description	39
4.13.2	Field Documentation	39
4.13.2.1	blkLen	39
4.13.2.2	busWidth	40
4.13.2.3	cardRCA	40
4.13.2.4	cardSpeed	40
4.13.2.5	cardType	40
4.13.2.6	cardVer	40
4.13.2.7	ccc	40
4.13.2.8	cidRegData	40
4.13.2.9	clkRate	40
4.13.2.10	csdRegData	40
4.13.2.11	dataTimeOut	40
4.13.2.12	ddrMode	40
4.13.2.13	eraseSize	40
4.13.2.14	highCapacity	41
4.13.2.15	locked	41
4.13.2.16	numBlks	41
4.13.2.17	ocrRegData	41
4.13.2.18	opVoltage	41
4.13.2.19	removable	41
4.13.2.20	uhslOpMode	41
4.13.2.21	writable	41
4.14	CyU3PDebugLog_t Struct Reference	41
4.14.1	Detailed Description	42
4.14.2	Field Documentation	42
4.14.2.1	msg	42
4.14.2.2	param	42
4.14.2.3	priority	42
4.14.2.4	threadId	42
4.15	CyU3PDmaBuffer_t Struct Reference	42
4.15.1	Detailed Description	42
4.15.2	Field Documentation	43
4.15.2.1	buffer	43
4.15.2.2	count	43
4.15.2.3	size	43
4.15.2.4	status	43
4.16	CyU3PDmaCBInput_t Union Reference	43

4.16.1	Detailed Description	43
4.16.2	Field Documentation	44
4.16.2.1	buffer_p	44
4.17	CyU3PDmaChannel Struct Reference	44
4.17.1	Detailed Description	45
4.17.2	Field Documentation	45
4.17.2.1	activeConsIndex	45
4.17.2.2	activeProdIndex	45
4.17.2.3	cb	45
4.17.2.4	commitConsIndex	45
4.17.2.5	commitProdIndex	46
4.17.2.6	consHeader	46
4.17.2.7	consSckId	46
4.17.2.8	consSusp	46
4.17.2.9	count	46
4.17.2.10	currentConsIndex	46
4.17.2.11	currentProdIndex	46
4.17.2.12	discardCount	46
4.17.2.13	dmaMode	46
4.17.2.14	endSig	46
4.17.2.15	firstConsIndex	46
4.17.2.16	firstProdIndex	46
4.17.2.17	flags	47
4.17.2.18	isDmaHandleDCache	47
4.17.2.19	lock	47
4.17.2.20	notification	47
4.17.2.21	overrideDscrIndex	47
4.17.2.22	prodAvailCount	47
4.17.2.23	prodFooter	47
4.17.2.24	prodHeader	47
4.17.2.25	prodSckId	47
4.17.2.26	prodSusp	47
4.17.2.27	size	47
4.17.2.28	startSig	47
4.17.2.29	state	48
4.17.2.30	type	48
4.17.2.31	usbConsSusp	48
4.17.2.32	xferSize	48
4.18	CyU3PDmaChannelConfig_t Struct Reference	48
4.18.1	Detailed Description	48

4.18.2 Field Documentation . . . . .	49
4.18.2.1 cb . . . . .	49
4.18.2.2 consHeader . . . . .	49
4.18.2.3 consSckId . . . . .	49
4.18.2.4 count . . . . .	49
4.18.2.5 dmaMode . . . . .	49
4.18.2.6 notification . . . . .	49
4.18.2.7 prodAvailCount . . . . .	50
4.18.2.8 prodFooter . . . . .	50
4.18.2.9 prodHeader . . . . .	50
4.18.2.10 prodSckId . . . . .	50
4.18.2.11 size . . . . .	50
4.19 CyU3PDmaDescriptor_t Struct Reference . . . . .	50
4.19.1 Detailed Description . . . . .	50
4.19.2 Field Documentation . . . . .	51
4.19.2.1 buffer . . . . .	51
4.19.2.2 chain . . . . .	51
4.19.2.3 size . . . . .	51
4.19.2.4 sync . . . . .	51
4.20 CyU3PDmaMultiChannel Struct Reference . . . . .	51
4.20.1 Detailed Description . . . . .	52
4.20.2 Field Documentation . . . . .	53
4.20.2.1 activeConsIndex . . . . .	53
4.20.2.2 activeProdIndex . . . . .	53
4.20.2.3 bufferCount . . . . .	53
4.20.2.4 cb . . . . .	53
4.20.2.5 commitConsIndex . . . . .	53
4.20.2.6 commitProdIndex . . . . .	53
4.20.2.7 consDisabled . . . . .	53
4.20.2.8 consHeader . . . . .	53
4.20.2.9 consSckId . . . . .	53
4.20.2.10 consSusp . . . . .	53
4.20.2.11 count . . . . .	53
4.20.2.12 currentConsIndex . . . . .	53
4.20.2.13 currentProdIndex . . . . .	54
4.20.2.14 discardCount . . . . .	54
4.20.2.15 dmaMode . . . . .	54
4.20.2.16 endSig . . . . .	54
4.20.2.17 firstConsIndex . . . . .	54
4.20.2.18 firstProdIndex . . . . .	54

4.20.2.19 flags . . . . .	54
4.20.2.20 isDmaHandleDCache . . . . .	54
4.20.2.21 lock . . . . .	54
4.20.2.22 notification . . . . .	54
4.20.2.23 overrideDscrIndex . . . . .	54
4.20.2.24 prodAvailCount . . . . .	54
4.20.2.25 prodFooter . . . . .	55
4.20.2.26 prodHeader . . . . .	55
4.20.2.27 prodSckId . . . . .	55
4.20.2.28 prodSusp . . . . .	55
4.20.2.29 size . . . . .	55
4.20.2.30 startSig . . . . .	55
4.20.2.31 state . . . . .	55
4.20.2.32 type . . . . .	55
4.20.2.33 usbConsSusp . . . . .	55
4.20.2.34 validSckCount . . . . .	55
4.20.2.35 xferSize . . . . .	55
4.21 CyU3PDmaMultiChannelConfig_t Struct Reference . . . . .	56
4.21.1 Detailed Description . . . . .	56
4.21.2 Field Documentation . . . . .	56
4.21.2.1 cb . . . . .	56
4.21.2.2 consHeader . . . . .	56
4.21.2.3 consSckId . . . . .	57
4.21.2.4 count . . . . .	57
4.21.2.5 dmaMode . . . . .	57
4.21.2.6 notification . . . . .	57
4.21.2.7 prodAvailCount . . . . .	57
4.21.2.8 prodFooter . . . . .	57
4.21.2.9 prodHeader . . . . .	57
4.21.2.10 prodSckId . . . . .	57
4.21.2.11 size . . . . .	57
4.21.2.12 validSckCount . . . . .	57
4.22 CyU3PDmaSocket_t Struct Reference . . . . .	58
4.22.1 Detailed Description . . . . .	58
4.22.2 Field Documentation . . . . .	58
4.22.2.1 activeDscr . . . . .	58
4.22.2.2 dscrChain . . . . .	58
4.22.2.3 intr . . . . .	58
4.22.2.4 intrMask . . . . .	58
4.22.2.5 sckEvent . . . . .	59

---

4.22.2.6	status	59
4.22.2.7	unused19	59
4.22.2.8	unused2	59
4.22.2.9	xferCount	59
4.22.2.10	xferSize	59
4.23	CyU3PDmaSocketConfig_t Struct Reference	59
4.23.1	Detailed Description	59
4.23.2	Field Documentation	60
4.23.2.1	dscrChain	60
4.23.2.2	intr	60
4.23.2.3	intrMask	60
4.23.2.4	status	60
4.23.2.5	xferCount	60
4.23.2.6	xferSize	60
4.24	CyU3PEpConfig_t Struct Reference	60
4.24.1	Detailed Description	61
4.24.2	Field Documentation	61
4.24.2.1	burstLen	61
4.24.2.2	enable	61
4.24.2.3	epType	61
4.24.2.4	isoPkts	61
4.24.2.5	pcktSize	61
4.24.2.6	streams	61
4.25	CyU3PGpifConfig_t Struct Reference	61
4.25.1	Detailed Description	62
4.25.2	Field Documentation	62
4.25.2.1	functionCount	62
4.25.2.2	functionData	62
4.25.2.3	regCount	62
4.25.2.4	regData	62
4.25.2.5	stateCount	63
4.25.2.6	stateData	63
4.25.2.7	statePosition	63
4.26	CyU3PGpifWaveData Struct Reference	63
4.26.1	Detailed Description	63
4.26.2	Field Documentation	63
4.26.2.1	leftData	63
4.26.2.2	rightData	64
4.27	CyU3PGpioClock_t Struct Reference	64
4.27.1	Detailed Description	64

4.27.2 Field Documentation . . . . .	64
4.27.2.1 clkSrc . . . . .	64
4.27.2.2 fastClkDiv . . . . .	64
4.27.2.3 halfDiv . . . . .	64
4.27.2.4 simpleDiv . . . . .	65
4.27.2.5 slowClkDiv . . . . .	65
4.28 CyU3PGpioComplexConfig_t Struct Reference . . . . .	65
4.28.1 Detailed Description . . . . .	65
4.28.2 Field Documentation . . . . .	66
4.28.2.1 driveHighEn . . . . .	66
4.28.2.2 driveLowEn . . . . .	66
4.28.2.3 inputEn . . . . .	66
4.28.2.4 intrMode . . . . .	66
4.28.2.5 outValue . . . . .	66
4.28.2.6 period . . . . .	66
4.28.2.7 pinMode . . . . .	66
4.28.2.8 threshold . . . . .	66
4.28.2.9 timer . . . . .	66
4.28.2.10 timerMode . . . . .	66
4.29 CyU3PGpioSimpleConfig_t Struct Reference . . . . .	66
4.29.1 Detailed Description . . . . .	67
4.29.2 Field Documentation . . . . .	67
4.29.2.1 driveHighEn . . . . .	67
4.29.2.2 driveLowEn . . . . .	67
4.29.2.3 inputEn . . . . .	67
4.29.2.4 intrMode . . . . .	67
4.29.2.5 outValue . . . . .	67
4.30 CyU3PI2cConfig_t Struct Reference . . . . .	68
4.30.1 Detailed Description . . . . .	68
4.30.2 Field Documentation . . . . .	68
4.30.2.1 bitRate . . . . .	68
4.30.2.2 busTimeout . . . . .	68
4.30.2.3 dmaTimeout . . . . .	68
4.30.2.4 isDma . . . . .	68
4.31 CyU3PI2cPreamble_t Struct Reference . . . . .	69
4.31.1 Detailed Description . . . . .	69
4.31.2 Field Documentation . . . . .	69
4.31.2.1 buffer . . . . .	70
4.31.2.2 ctrlMask . . . . .	70
4.31.2.3 length . . . . .	70

---

4.32 CyU3PI2sConfig_t Struct Reference . . . . .	70
4.32.1 Detailed Description . . . . .	70
4.32.2 Field Documentation . . . . .	70
4.32.2.1 isDma . . . . .	70
4.32.2.2 isLsbFirst . . . . .	70
4.32.2.3 isMono . . . . .	71
4.32.2.4 padMode . . . . .	71
4.32.2.5 sampleRate . . . . .	71
4.32.2.6 sampleWidth . . . . .	71
4.33 CyU3PIOMatrixConfig_t Struct Reference . . . . .	71
4.33.1 Detailed Description . . . . .	71
4.33.2 Field Documentation . . . . .	72
4.33.2.1 gpioComplexEn . . . . .	72
4.33.2.2 gpioSimpleEn . . . . .	72
4.33.2.3 isDQ32Bit . . . . .	72
4.33.2.4 lppMode . . . . .	72
4.33.2.5 s0Mode . . . . .	72
4.33.2.6 s1Mode . . . . .	72
4.33.2.7 uselI2C . . . . .	72
4.33.2.8 uselI2S . . . . .	72
4.33.2.9 useSpi . . . . .	72
4.33.2.10 useUart . . . . .	72
4.34 CyU3PMbox Struct Reference . . . . .	73
4.34.1 Detailed Description . . . . .	73
4.34.2 Field Documentation . . . . .	73
4.34.2.1 w0 . . . . .	73
4.34.2.2 w1 . . . . .	73
4.35 CyU3PMipicsiCfg_t Struct Reference . . . . .	73
4.35.1 Detailed Description . . . . .	74
4.35.2 Field Documentation . . . . .	74
4.35.2.1 csiRxClkDiv . . . . .	74
4.35.2.2 dataFormat . . . . .	74
4.35.2.3 fifoDelay . . . . .	74
4.35.2.4 hResolution . . . . .	74
4.35.2.5 mClkCtl . . . . .	74
4.35.2.6 mClkRefDiv . . . . .	74
4.35.2.7 numDataLanes . . . . .	75
4.35.2.8 parClkDiv . . . . .	75
4.35.2.9 pllFbd . . . . .	75
4.35.2.10 pllFr . . . . .	75

4.35.2.11	plIPrd	75
4.36	CyU3PMipicsiErrorCounts_t Struct Reference	75
4.36.1	Detailed Description	75
4.36.2	Field Documentation	76
4.36.2.1	crcErrCnt	76
4.36.2.2	ctlErrCnt	76
4.36.2.3	eidErrCnt	76
4.36.2.4	frmErrCnt	76
4.36.2.5	mdlErrCnt	76
4.36.2.6	recrErrCnt	76
4.36.2.7	recSyncErrCnt	76
4.36.2.8	unrcErrCnt	76
4.36.2.9	unrSyncErrCnt	76
4.37	CyU3POtgConfig_t Struct Reference	76
4.37.1	Detailed Description	77
4.37.2	Field Documentation	77
4.37.2.1	cb	77
4.37.2.2	chargerMode	77
4.37.2.3	otgMode	77
4.38	CyU3PPibClock_t Struct Reference	77
4.38.1	Detailed Description	78
4.38.2	Field Documentation	78
4.38.2.1	clkDiv	78
4.38.2.2	clkSrc	78
4.38.2.3	isDIIEnable	78
4.38.2.4	isHalfDiv	78
4.39	CyU3PSdioCardRegs Struct Reference	78
4.39.1	Detailed Description	79
4.39.2	Field Documentation	79
4.39.2.1	addrCIS	79
4.39.2.2	cardCapability	79
4.39.2.3	cardSpeed	79
4.39.2.4	CCCRVersion	79
4.39.2.5	fn0BlockSize	79
4.39.2.6	isMemoryPresent	79
4.39.2.7	manufacturerId	79
4.39.2.8	manufacturerInfo	79
4.39.2.9	numberOfFunctions	79
4.39.2.10	sdioVersion	80
4.39.2.11	supportsAsyncIntr	80

4.39.2.12 uhsSupport . . . . .	80
4.40 CyU3PSibCtx Struct Reference . . . . .	80
4.40.1 Detailed Description . . . . .	80
4.40.2 Field Documentation . . . . .	80
4.40.2.1 activeUnitId . . . . .	80
4.40.2.2 inUse . . . . .	80
4.40.2.3 isRead . . . . .	81
4.40.2.4 mutexLock . . . . .	81
4.40.2.5 numBootLuns . . . . .	81
4.40.2.6 numUserLuns . . . . .	81
4.40.2.7 partition . . . . .	81
4.40.2.8 status . . . . .	81
4.40.2.9 writeTimer . . . . .	81
4.40.2.10 writeTimerCbk . . . . .	81
4.41 CyU3PSibDevInfo Struct Reference . . . . .	81
4.41.1 Detailed Description . . . . .	82
4.41.2 Field Documentation . . . . .	82
4.41.2.1 blkLen . . . . .	82
4.41.2.2 busWidth . . . . .	82
4.41.2.3 cardType . . . . .	82
4.41.2.4 ccc . . . . .	82
4.41.2.5 clkRate . . . . .	82
4.41.2.6 ddrMode . . . . .	82
4.41.2.7 eraseSize . . . . .	82
4.41.2.8 locked . . . . .	83
4.41.2.9 numBlks . . . . .	83
4.41.2.10 numUnits . . . . .	83
4.41.2.11 opVoltage . . . . .	83
4.41.2.12 removable . . . . .	83
4.41.2.13 writeable . . . . .	83
4.42 CyU3PSibGlobalData Struct Reference . . . . .	83
4.42.1 Detailed Description . . . . .	84
4.42.2 Field Documentation . . . . .	84
4.42.2.1 activePartition . . . . .	84
4.42.2.2 isActive . . . . .	84
4.42.2.3 nextWrAddress . . . . .	84
4.42.2.4 openWrSize . . . . .	84
4.42.2.5 partConfig . . . . .	84
4.42.2.6 s0Enabled . . . . .	84
4.42.2.7 s1Enabled . . . . .	84

4.42.2.8 sibDmaChannel . . . . .	84
4.42.2.9 sibEvtCbk . . . . .	84
4.42.2.10 wrCommitPending . . . . .	84
4.42.2.11 wrCommitSize . . . . .	85
4.43 CyU3PSibIntfParams Struct Reference . . . . .	85
4.43.1 Detailed Description . . . . .	85
4.43.2 Field Documentation . . . . .	85
4.43.2.1 cardDetType . . . . .	85
4.43.2.2 cardInitDelay . . . . .	85
4.43.2.3 lowVoltage . . . . .	85
4.43.2.4 lvGpioState . . . . .	85
4.43.2.5 maxFreq . . . . .	86
4.43.2.6 resetGpio . . . . .	86
4.43.2.7 rstActHigh . . . . .	86
4.43.2.8 useDdr . . . . .	86
4.43.2.9 voltageSwGpio . . . . .	86
4.43.2.10 writeProtEnable . . . . .	86
4.44 CyU3PSibLunInfo Struct Reference . . . . .	86
4.44.1 Detailed Description . . . . .	86
4.44.2 Field Documentation . . . . .	87
4.44.2.1 blockSize . . . . .	87
4.44.2.2 location . . . . .	87
4.44.2.3 numBlocks . . . . .	87
4.44.2.4 startAddr . . . . .	87
4.44.2.5 type . . . . .	87
4.44.2.6 valid . . . . .	87
4.44.2.7 writeable . . . . .	87
4.45 CyU3PSpiConfig_t Struct Reference . . . . .	87
4.45.1 Detailed Description . . . . .	88
4.45.2 Field Documentation . . . . .	88
4.45.2.1 clock . . . . .	88
4.45.2.2 cpha . . . . .	88
4.45.2.3 cpol . . . . .	88
4.45.2.4 isLsbFirst . . . . .	88
4.45.2.5 lagTime . . . . .	88
4.45.2.6 leadTime . . . . .	88
4.45.2.7 ssnCtrl . . . . .	88
4.45.2.8 ssnPol . . . . .	89
4.45.2.9 wordLen . . . . .	89
4.46 CyU3PSysClockConfig_t Struct Reference . . . . .	89

4.46.1	Detailed Description	89
4.46.2	Field Documentation	90
4.46.2.1	clkSrc	90
4.46.2.2	cpuClkDiv	90
4.46.2.3	dmaClkDiv	90
4.46.2.4	mmioClkDiv	90
4.46.2.5	setSysClk400	90
4.46.2.6	useStandbyClk	90
4.47	CyU3PUuartConfig_t Struct Reference	90
4.47.1	Detailed Description	91
4.47.2	Field Documentation	91
4.47.2.1	baudRate	91
4.47.2.2	flowCtrl	91
4.47.2.3	isDma	91
4.47.2.4	parity	91
4.47.2.5	rxEnable	91
4.47.2.6	stopBit	91
4.47.2.7	txEnable	91
4.48	CyU3PUsbDescrPtrs Struct Reference	92
4.48.1	Detailed Description	92
4.48.2	Field Documentation	92
4.48.2.1	usbConfigDesc_p	92
4.48.2.2	usbDevDesc_p	92
4.48.2.3	usbDevQualDesc_p	92
4.48.2.4	usbFSConfigDesc_p	92
4.48.2.5	usbHSConfigDesc_p	92
4.48.2.6	usbOtherSpeedConfigDesc_p	92
4.48.2.7	usbSSBOSDesc_p	93
4.48.2.8	usbSSConfigDesc_p	93
4.48.2.9	usbSSDevDesc_p	93
4.48.2.10	usbStringDesc_p	93
4.49	CyU3PUsbHostConfig_t Struct Reference	93
4.49.1	Detailed Description	93
4.49.2	Field Documentation	93
4.49.2.1	ep0LowLevelControl	93
4.49.2.2	eventCb	94
4.49.2.3	xferCb	94
4.50	CyU3PUsbHostEpConfig_t Struct Reference	94
4.50.1	Detailed Description	94
4.50.2	Field Documentation	94

4.50.2.1	fullPktSize . . . . .	94
4.50.2.2	isStreamMode . . . . .	94
4.50.2.3	maxPktSize . . . . .	95
4.50.2.4	mult . . . . .	95
4.50.2.5	pollingRate . . . . .	95
4.50.2.6	type . . . . .	95
4.51	MemBlockInfo Struct Reference . . . . .	95
4.51.1	Detailed Description . . . . .	95
4.51.2	Field Documentation . . . . .	95
4.51.2.1	alloc_id . . . . .	95
4.51.2.2	alloc_size . . . . .	96
4.51.2.3	next_blk . . . . .	96
4.51.2.4	pad . . . . .	96
4.51.2.5	prev_blk . . . . .	96
4.51.2.6	start_sig . . . . .	96
<b>5</b>	<b>File Documentation</b> . . . . .	<b>97</b>
5.1	firmware/boot-fw/include/cyfx3device.h File Reference . . . . .	97
5.1.1	Detailed Description . . . . .	98
5.1.2	FX3 Memory Regions . . . . .	99
5.1.3	Typedef Documentation . . . . .	99
5.1.3.1	CyFx3BootIoMatrixConfig_t . . . . .	99
5.1.3.2	CyFx3BootSysClockSrc_t . . . . .	100
5.1.3.3	CyFx3PartNumber_t . . . . .	100
5.1.4	Enumeration Type Documentation . . . . .	100
5.1.4.1	CyFx3BootSysClockSrc_t . . . . .	100
5.1.5	Function Documentation . . . . .	100
5.1.5.1	CyFx3BootDeviceConfigureIOMatrix(CyFx3BootIoMatrixConfig_t *cfg_p) . . . . .	100
5.1.5.2	CyFx3BootDeviceInit(CyBool_t setFastSysClk) . . . . .	101
5.1.5.3	CyFx3BootDeviceReset(void) . . . . .	101
5.1.5.4	CyFx3BootGetPartNumber(void) . . . . .	101
5.1.5.5	CyFx3BootGpioOverride(uint8_t pinNumber) . . . . .	102
5.1.5.6	CyFx3BootGpioRestore(uint8_t pinNumber) . . . . .	102
5.1.5.7	CyFx3BootJumpToProgramEntry(uint32_t address) . . . . .	102
5.1.5.8	CyFx3BootRetainGpioState(void) . . . . .	103
5.1.5.9	CyFx3BootWatchdogClear(void) . . . . .	103
5.1.5.10	CyFx3BootWatchdogConfigure(CyBool_t enable, uint32_t period) . . . . .	103
5.2	firmware/boot-fw/include/cyfx3dma.h File Reference . . . . .	103
5.2.1	Detailed Description . . . . .	106
5.2.2	Macro Definition Documentation . . . . .	106

5.2.2.1	CY_FX3_DMA_GETDSCR_BY_INDEX . . . . .	106
5.2.2.2	CY_FX3_DMA_LPP_SOCKCNT . . . . .	106
5.2.2.3	CY_FX3_DMA_MAX_DSCR_INDEX . . . . .	106
5.2.2.4	CY_FX3_DMA_MIN_DSCR_INDEX . . . . .	106
5.2.2.5	CY_FX3_DMA_PIB_SOCKCNT . . . . .	106
5.2.2.6	CY_FX3_DMA_USB_IN_SOCKCNT . . . . .	106
5.2.2.7	CY_FX3_DMA_USB_OUT_SOCKCNT . . . . .	106
5.2.2.8	CY_FX3_LPP_DMA_DSCR_INDEX . . . . .	106
5.2.2.9	CY_FX3_PIB_DMA_DSCR_INDEX . . . . .	107
5.2.2.10	CY_FX3_USB_DMA_DSCR_INDEX . . . . .	107
5.2.3	Typedef Documentation . . . . .	107
5.2.3.1	CyFx3BootDmaCallback_t . . . . .	107
5.2.3.2	CyFx3BootDmaDescriptor_t . . . . .	107
5.2.3.3	CyFx3BootDmaSocket_t . . . . .	107
5.2.3.4	CyFx3BootDmaSockId_t . . . . .	107
5.2.3.5	CyFx3BootDmaSockRegs_t . . . . .	107
5.2.4	Enumeration Type Documentation . . . . .	108
5.2.4.1	CyFx3BootDmaSockId_t . . . . .	108
5.2.5	Function Documentation . . . . .	110
5.2.5.1	CyFx3BootDmaClearSockInterrupts(CyFx3BootDmaSockId_t sockfd, uint32_t intrVal) . . . . .	110
5.2.5.2	CyFx3BootDmaDisableSocket(CyFx3BootDmaSockId_t sockfd) . . . . .	110
5.2.5.3	CyFx3BootDmaEnableSocket(CyFx3BootDmaSockId_t sockfd) . . . . .	110
5.2.5.4	CyFx3BootDmaGetDscrConfig(uint16_t dscrIndex, CyFx3BootDmaDescriptor_t *dscr_p) . . . . .	111
5.2.5.5	CyFx3BootDmaGetSocketConfig(CyFx3BootDmaSockId_t sockfd, CyFx3BootDmaSocket_t *sock_p) . . . . .	111
5.2.5.6	CyFx3BootDmaGetSockInterrupts(CyFx3BootDmaSockId_t sockfd) . . . . .	112
5.2.5.7	CyFx3BootDmaRegisterCallback(CyFx3BootDmaCallback_t cbFunc, CyBool_t usblntrEn, CyBool_t piblntrEn, CyBool_t serialIntrEn) . . . . .	112
5.2.5.8	CyFx3BootDmaSendSocketEvent(CyFx3BootDmaSockId_t sockfd, uint16_t dscrIndex, CyBool_t isOccupied) . . . . .	113
5.2.5.9	CyFx3BootDmaSetDscrConfig(uint16_t dscrIndex, CyFx3BootDmaDescriptor_t *dscr_p) . . . . .	113
5.2.5.10	CyFx3BootDmaSetSocketConfig(CyFx3BootDmaSockId_t sockfd, CyFx3BootDmaSocket_t *sock_p) . . . . .	113
5.2.5.11	CyFx3BootDmaWrapSocket(CyFx3BootDmaSockId_t sockfd) . . . . .	114
5.3	firmware/boot_fw/include/cyfx3error.h File Reference . . . . .	114
5.3.1	Detailed Description . . . . .	115
5.3.2	Enumeration Type Documentation . . . . .	115
5.3.2.1	CyFx3BootErrorCode_t . . . . .	115
5.4	firmware/boot_fw/include/cyfx3gpio.h File Reference . . . . .	116

5.4.1	Detailed Description . . . . .	117
5.4.2	Typedef Documentation . . . . .	117
5.4.2.1	CyFx3BootGpioIntrMode_t . . . . .	117
5.4.2.2	CyFx3BootGpioloMode_t . . . . .	117
5.4.2.3	CyFx3BootGpioSimpleConfig_t . . . . .	117
5.4.3	Enumeration Type Documentation . . . . .	118
5.4.3.1	CyFx3BootGpioIntrMode_t . . . . .	118
5.4.3.2	CyFx3BootGpioloMode_t . . . . .	118
5.4.4	Function Documentation . . . . .	118
5.4.4.1	CyFx3BootGpioDeInit(void) . . . . .	118
5.4.4.2	CyFx3BootGpioDisable(uint8_t gpioid) . . . . .	119
5.4.4.3	CyFx3BootGpioGetValue(uint8_t gpioid, CyBool_t *value_p) . . . . .	119
5.4.4.4	CyFx3BootGpioInit(void) . . . . .	119
5.4.4.5	CyFx3BootGpioSetIoMode(uint8_t gpioid, CyFx3BootGpioloMode_t ioMode) . . . . .	120
5.4.4.6	CyFx3BootGpioSetSimpleConfig(uint8_t gpioid, CyFx3BootGpioSimpleConfig_t *cfg_p) . . . . .	120
5.4.4.7	CyFx3BootGpioSetValue(uint8_t gpioid, CyBool_t value) . . . . .	121
5.5	firmware/boot_fw/include/cyfx3i2c.h File Reference . . . . .	121
5.5.1	Detailed Description . . . . .	122
5.5.2	Typedef Documentation . . . . .	122
5.5.2.1	CyFx3BootI2cConfig_t . . . . .	122
5.5.2.2	CyFx3BootI2cPreamble_t . . . . .	122
5.5.3	Function Documentation . . . . .	123
5.5.3.1	CyFx3BootI2cDeInit(void) . . . . .	123
5.5.3.2	CyFx3BootI2cDmaXferData(CyBool_t isRead, uint32_t address, uint32_t length, uint32_t timeout) . . . . .	123
5.5.3.3	CyFx3BootI2cInit(void) . . . . .	124
5.5.3.4	CyFx3BootI2cReceiveBytes(CyFx3BootI2cPreamble_t *preamble, uint8_t *data, uint32_t byteCount, uint32_t retryCount) . . . . .	124
5.5.3.5	CyFx3BootI2cSendCommand(CyFx3BootI2cPreamble_t *preamble, uint32_t byteCount, CyBool_t isRead) . . . . .	125
5.5.3.6	CyFx3BootI2cSetConfig(CyFx3BootI2cConfig_t *config) . . . . .	125
5.5.3.7	CyFx3BootI2cSetTimeout(uint32_t timeout) . . . . .	126
5.5.3.8	CyFx3BootI2cTransmitBytes(CyFx3BootI2cPreamble_t *preamble, uint8_t *data, uint32_t byteCount, uint32_t retryCount) . . . . .	126
5.5.3.9	CyFx3BootI2cWaitForAck(CyFx3BootI2cPreamble_t *preamble, uint32_t retryCount) . . . . .	126
5.6	firmware/boot_fw/include/cyfx3pib.h File Reference . . . . .	127
5.6.1	Detailed Description . . . . .	129
5.6.2	Typedef Documentation . . . . .	129
5.6.2.1	CyFx3BootGpiflIntrCb_t . . . . .	129
5.6.2.2	CyFx3BootPibClock_t . . . . .	129

---

5.6.2.3	CyFx3BootPMMCEvent_t . . . . .	129
5.6.2.4	CyFx3BootPMMCIntrCb_t . . . . .	130
5.6.2.5	CyFx3GpifCounterType . . . . .	130
5.6.2.6	CyU3PGpifConfig_t . . . . .	130
5.6.2.7	CyU3PGpifWaveData . . . . .	130
5.6.3	Enumeration Type Documentation . . . . .	131
5.6.3.1	CyFx3BootPMMCEvent_t . . . . .	131
5.6.3.2	CyFx3GpifCounterType . . . . .	131
5.6.4	Function Documentation . . . . .	131
5.6.4.1	CyFx3BootGpifControlSWInput(CyBool_t set) . . . . .	131
5.6.4.2	CyFx3BootGpifDisable(CyBool_t forceReload) . . . . .	132
5.6.4.3	CyFx3BootGpifGetState(void) . . . . .	132
5.6.4.4	CyFx3BootGpifInitCounter(CyFx3GpifCounterType counter, uint32_t initialValue, uint32_t limit, CyBool_t reload, int8_t increment, uint8_t outputbit) . . . . .	132
5.6.4.5	CyFx3BootGpifLoad(const CyU3PGpifConfig_t *conf) . . . . .	133
5.6.4.6	CyFx3BootGpifRegisterCallback(CyFx3BootGpifIntrCb_t cbFunc) . . . . .	133
5.6.4.7	CyFx3BootGpifSMStart(uint8_t startState, uint8_t initialAlpha) . . . . .	134
5.6.4.8	CyFx3BootGpifSMSwitch(uint16_t fromState, uint16_t toState, uint8_t initialAlpha) . . . . .	134
5.6.4.9	CyFx3BootGpifSocketConfigure(uint8_t threadIndex, uint8_t socketNum, uint16_t watermark, CyBool_t flagOnData, uint8_t burst) . . . . .	134
5.6.4.10	CyFx3BootPibDeinit(void) . . . . .	135
5.6.4.11	CyFx3BootPibDmaXferData(uint8_t sockNum, CyBool_t isRead, uint32_t address, uint32_t length, uint32_t timeout) . . . . .	135
5.6.4.12	CyFx3BootPibHandleEvents(void) . . . . .	136
5.6.4.13	CyFx3BootPibInit(CyFx3BootPibClock_t *clkInfo_p, CyBool_t isMmcMode) . . . . .	136
5.6.4.14	CyFx3BootPibRegisterMmcCallback(CyFx3BootPMMCIntrCb_t cb) . . . . .	136
5.7	firmware/boot_fw/include/cyfx3spi.h File Reference . . . . .	137
5.7.1	Detailed Description . . . . .	138
5.7.2	Typedef Documentation . . . . .	138
5.7.2.1	CyFx3BootSpiConfig_t . . . . .	138
5.7.2.2	CyFx3BootSpiSsnCtrl_t . . . . .	138
5.7.2.3	CyFx3BootSpiSsnLagLead_t . . . . .	139
5.7.3	Enumeration Type Documentation . . . . .	139
5.7.3.1	CyFx3BootSpiSsnCtrl_t . . . . .	139
5.7.3.2	CyFx3BootSpiSsnLagLead_t . . . . .	140
5.7.4	Function Documentation . . . . .	140
5.7.4.1	CyFx3BootSpiDeinit(void) . . . . .	140
5.7.4.2	CyFx3BootSpiDisableBlockXfer(void) . . . . .	140
5.7.4.3	CyFx3BootSpiDmaXferData(CyBool_t isRead, uint32_t address, uint32_t length, uint32_t timeout) . . . . .	141
5.7.4.4	CyFx3BootSpiInit(void) . . . . .	141

5.7.4.5	CyFx3BootSpiReceiveWords(uint8_t *data, uint32_t byteCount) . . . . .	141
5.7.4.6	CyFx3BootSpiSetBlockXfer(uint32_t txSize, uint32_t rxSize) . . . . .	142
5.7.4.7	CyFx3BootSpiSetConfig(CyFx3BootSpiConfig_t *config) . . . . .	142
5.7.4.8	CyFx3BootSpiSetSsnLine(CyBool_t isHigh) . . . . .	143
5.7.4.9	CyFx3BootSpiSetTimeout(uint32_t timeout) . . . . .	143
5.7.4.10	CyFx3BootSpiTransmitWords(uint8_t *data, uint32_t byteCount) . . . . .	143
5.8	firmware/boot_fw/include/cyfx3uart.h File Reference . . . . .	144
5.8.1	Detailed Description . . . . .	145
5.8.2	Typedef Documentation . . . . .	145
5.8.2.1	CyFx3BootUartBaudrate_t . . . . .	145
5.8.2.2	CyFx3BootUartConfig_t . . . . .	146
5.8.2.3	CyFx3BootUartParity_t . . . . .	146
5.8.2.4	CyFx3BootUartStopBit_t . . . . .	146
5.8.3	Enumeration Type Documentation . . . . .	146
5.8.3.1	CyFx3BootUartBaudrate_t . . . . .	146
5.8.3.2	CyFx3BootUartParity_t . . . . .	147
5.8.3.3	CyFx3BootUartStopBit_t . . . . .	147
5.8.4	Function Documentation . . . . .	147
5.8.4.1	CyFx3BootUartDelInit(void) . . . . .	147
5.8.4.2	CyFx3BootUartDmaXferData(CyBool_t isRead, uint32_t address, uint32_t length, uint32_t timeout) . . . . .	148
5.8.4.3	CyFx3BootUartInit(void) . . . . .	148
5.8.4.4	CyFx3BootUartPrintMessage(uint8_t *outBuf_p, uint16_t outBufLen, char *fmt,...) . . . . .	148
5.8.4.5	CyFx3BootUartReceiveBytes(uint8_t *data_p, uint32_t count) . . . . .	149
5.8.4.6	CyFx3BootUartRxSetBlockXfer(uint32_t rxSize) . . . . .	149
5.8.4.7	CyFx3BootUartSetConfig(CyFx3BootUartConfig_t *config) . . . . .	150
5.8.4.8	CyFx3BootUartSetTimeout(uint32_t timeout) . . . . .	150
5.8.4.9	CyFx3BootUartTransmitBytes(uint8_t *data_p, uint32_t count) . . . . .	150
5.8.4.10	CyFx3BootUartTxSetBlockXfer(uint32_t txSize) . . . . .	151
5.9	firmware/boot_fw/include/cyfx3usb.h File Reference . . . . .	151
5.9.1	Detailed Description . . . . .	154
5.9.2	Macro Definition Documentation . . . . .	154
5.9.2.1	CY_FX3_USB_MAX_STRING_DESC_INDEX . . . . .	154
5.9.3	Typedef Documentation . . . . .	154
5.9.3.1	CyFx3BootUsbEp0Pkt_t . . . . .	154
5.9.3.2	CyFx3BootUsbEpConfig_t . . . . .	154
5.9.3.3	CyFx3BootUSBEVENTCb_t . . . . .	154
5.9.3.4	CyFx3BootUSBSetupCb_t . . . . .	154
5.9.3.5	CyU3PUsbDescrPtrs . . . . .	155
5.9.3.6	CyU3PUsbDescType . . . . .	155

5.9.3.7	CyU3PUsbLinkPowerMode . . . . .	155
5.9.3.8	CyU3PUSBSetDescType_t . . . . .	155
5.9.4	Enumeration Type Documentation . . . . .	155
5.9.4.1	CyFx3BootUsbEpType_t . . . . .	155
5.9.4.2	CyFx3BootUsbEventType_t . . . . .	155
5.9.4.3	CyFx3BootUsbSpeed_t . . . . .	156
5.9.4.4	CyU3PUsbDescType . . . . .	156
5.9.4.5	CyU3PUsbLinkPowerMode . . . . .	157
5.9.4.6	CyU3PUSBSetDescType_t . . . . .	157
5.9.5	Function Documentation . . . . .	158
5.9.5.1	CyFx3BootRegisterSetupCallback(CyFx3BootUSBSetupCb_t callback) . . . . .	158
5.9.5.2	CyFx3BootUsbAckSetup(void) . . . . .	158
5.9.5.3	CyFx3BootUsbCheckUsb3Disconnect(void) . . . . .	158
5.9.5.4	CyFx3BootUsbConnect(CyBool_t connect, CyBool_t ssEnable) . . . . .	159
5.9.5.5	CyFx3BootUsbDmaXferData(uint8_t epNum, uint32_t address, uint32_t length, uint32_t timeout) . . . . .	159
5.9.5.6	CyFx3BootUsbEp0StatusCheck(void) . . . . .	159
5.9.5.7	CyFx3BootUsbGetDesc(void) . . . . .	160
5.9.5.8	CyFx3BootUsbGetEpCfg(uint8_t ep, CyBool_t *isNak, CyBool_t *isStall) . . . . .	160
5.9.5.9	CyFx3BootUsbGetLinkPowerState(void) . . . . .	160
5.9.5.10	CyFx3BootUsbGetSpeed(void) . . . . .	161
5.9.5.11	CyFx3BootUsbHandleEvents(void) . . . . .	161
5.9.5.12	CyFx3BootUsbLPMDisable(void) . . . . .	161
5.9.5.13	CyFx3BootUsbLPMEnable(void) . . . . .	161
5.9.5.14	CyFx3BootUsbSendCompliancePatterns(void) . . . . .	161
5.9.5.15	CyFx3BootUsbSetClrFeature(uint32_t sc, CyBool_t isConfigured, CyFx3BootUsbEp0Pkt_t *pEp0) . . . . .	162
5.9.5.16	CyFx3BootUsbSetDesc(CyU3PUSBSetDescType_t descType, uint8_t desc_index, uint8_t *desc) . . . . .	162
5.9.5.17	CyFx3BootUsbSetEpConfig(uint8_t ep, CyFx3BootUsbEpConfig_t *epinfo) . . . . .	163
5.9.5.18	CyFx3BootUsbSetLinkPowerState(CyU3PUsbLinkPowerMode mode) . . . . .	163
5.9.5.19	CyFx3BootUsbSigResume(void) . . . . .	163
5.9.5.20	CyFx3BootUsbStall(uint8_t ep, CyBool_t stall, CyBool_t toggle) . . . . .	164
5.9.5.21	CyFx3BootUsbStart(CyBool_t noReEnum, CyFx3BootUSBEVENTCb_t cb) . . . . .	164
5.9.5.22	CyFx3BootUsbVBattEnable(CyBool_t enable) . . . . .	164
5.10	firmware/boot_firmware/include/cyfx3utils.h File Reference . . . . .	165
5.10.1	Detailed Description . . . . .	165
5.10.2	Function Documentation . . . . .	165
5.10.2.1	CyFx3BootBusyWait(uint16_t usWait) . . . . .	165
5.10.2.2	CyFx3BootMemCopy(uint8_t *dest, uint8_t *src, uint32_t count) . . . . .	165
5.10.2.3	CyFx3BootMemSet(uint8_t *buf, uint8_t value, uint32_t count) . . . . .	166

5.10.2.4	CyFx3BootSNPrintf(uint8_t *buffer, uint16_t maxLength, char *fmt,...) . . . . .	166
5.11	firmware/u3p_firmware/inc/cyfx3_api.h File Reference . . . . .	166
5.11.1	Detailed Description . . . . .	169
5.11.2	Typedef Documentation . . . . .	169
5.11.2.1	CyU3PloMatrixConfig_t . . . . .	169
5.11.2.2	CyU3PloMatrixLppMode_t . . . . .	170
5.11.2.3	CyU3PPartNumber_t . . . . .	170
5.11.2.4	CyU3PSPortMode_t . . . . .	170
5.11.3	Enumeration Type Documentation . . . . .	170
5.11.3.1	CyU3PloMatrixLppMode_t . . . . .	170
5.11.3.2	CyU3PPartNumber_t . . . . .	171
5.11.3.3	CyU3PSPortMode_t . . . . .	172
5.11.4	Function Documentation . . . . .	173
5.11.4.1	CyFx3BusyWait(uint16_t usWait) . . . . .	173
5.11.4.2	CyFx3DevClearSwInterrupt(void) . . . . .	173
5.11.4.3	CyFx3DevGetMipiLaneCount(void) . . . . .	173
5.11.4.4	CyFx3DevIdentifyPart(void) . . . . .	174
5.11.4.5	CyFx3DevInitPageTables(void) . . . . .	174
5.11.4.6	CyFx3DevIOConfigure(CyU3PloMatrixConfig_t *cfg_p) . . . . .	174
5.11.4.7	CyFx3DevIODisableSib0(void) . . . . .	174
5.11.4.8	CyFx3DevIODisableSib1(void) . . . . .	174
5.11.4.9	CyFx3DevIOIsGpio(uint32_t gpiod, CyBool_t isSimple) . . . . .	175
5.11.4.10	CyFx3DevIOIsI2cConfigured(void) . . . . .	175
5.11.4.11	CyFx3DevIOIsI2sConfigured(void) . . . . .	175
5.11.4.12	CyFx3DevIOIsSib0Configured(void) . . . . .	175
5.11.4.13	CyFx3DevIOIsSib1BitWide(uint8_t portId) . . . . .	175
5.11.4.14	CyFx3DevIOIsSib1Configured(void) . . . . .	176
5.11.4.15	CyFx3DevIOIsSib8BitWide(uint8_t portId) . . . . .	176
5.11.4.16	CyFx3DevIOIsSpiConfigured(void) . . . . .	176
5.11.4.17	CyFx3DevIOIsUartConfigured(void) . . . . .	176
5.11.4.18	CyFx3DevIOSelectGpio(uint8_t gpiod, CyBool_t enable, CyBool_t isSimple) . . . . .	176
5.11.4.19	CyFx3DevIsGpif32Supported(void) . . . . .	177
5.11.4.20	CyFx3DevIsGpifSupported(void) . . . . .	177
5.11.4.21	CyFx3DevIsI2sSupported(void) . . . . .	177
5.11.4.22	CyFx3DevIsMipicsiSupported(void) . . . . .	177
5.11.4.23	CyFx3DevIsOtgSupported(void) . . . . .	178
5.11.4.24	CyFx3DevIsRam512Supported(void) . . . . .	178
5.11.4.25	CyFx3DevIsSib0Supported(void) . . . . .	178
5.11.4.26	CyFx3DevIsSib1Supported(void) . . . . .	178
5.11.4.27	CyFx3DevIsUsb3Supported(void) . . . . .	178

5.11.4.28 CyFx3LppIsOn(void) . . . . .	179
5.11.4.29 CyFx3PibDIIEnable(void) . . . . .	179
5.11.4.30 CyFx3PibGetDIIStatus(void) . . . . .	179
5.11.4.31 CyFx3PibIsOn(void) . . . . .	179
5.11.4.32 CyFx3PibPowerOff(void) . . . . .	179
5.11.4.33 CyFx3PibPowerOn(void) . . . . .	179
5.11.4.34 CyFx3SetCpuFreq(uint32_t cpuFreq) . . . . .	180
5.11.4.35 CyFx3SibPowerOff(void) . . . . .	180
5.11.4.36 CyFx3SibPowerOn(void) . . . . .	180
5.11.4.37 CyFx3Usb2PhySetup(void) . . . . .	180
5.11.4.38 CyFx3Usb3LnkRelaxHpTimeout(void) . . . . .	181
5.11.4.39 CyFx3Usb3LnkSetup(void) . . . . .	181
5.11.4.40 CyFx3Usb3SendTP(uint32_t *tpData) . . . . .	181
5.11.4.41 CyFx3UsbDmaPrefetchEnable(CyBool_t streamEnable) . . . . .	181
5.11.4.42 CyFx3UsbIsOn(void) . . . . .	182
5.11.4.43 CyFx3UsbPowerOn(void) . . . . .	182
5.11.4.44 CyFx3UsbWritePhyReg(uint16_t phyAddr, uint16_t phyVal) . . . . .	182
5.12 firmware/u3p_firmware/inc/cyfxapidesc.h File Reference . . . . .	182
5.12.1 Detailed Description . . . . .	182
5.13 firmware/u3p_firmware/inc/cyfxversion.h File Reference . . . . .	182
5.13.1 Detailed Description . . . . .	183
5.14 firmware/u3p_firmware/inc/cyu3cardmgr.h File Reference . . . . .	183
5.14.1 Detailed Description . . . . .	186
5.14.2 Macro Definition Documentation . . . . .	186
5.14.2.1 CY_U3P_MMC_SW_PARTCFG_BOOT1_PARAM . . . . .	186
5.14.2.2 CY_U3P_MMC_SW_PARTCFG_BOOT2_PARAM . . . . .	186
5.14.2.3 CY_U3P_MMC_SW_PARTCFG_USER_PARAM . . . . .	187
5.14.2.4 CY_U3P_SD_SW_HIGHSP_PARAM . . . . .	187
5.14.2.5 CY_U3P_SD_SW_QUERY_FUNCTIONS . . . . .	187
5.14.2.6 CY_U3P_SD_SW_UHS1_PARAM . . . . .	187
5.14.2.7 CyU3PSibClearIntr . . . . .	187
5.14.2.8 CyU3PSibConvertAddr . . . . .	187
5.14.2.9 CyU3PSibResetSibCtrlr . . . . .	187
5.14.2.10 CyU3PSibSetActiveSocket . . . . .	188
5.14.3 Typedef Documentation . . . . .	188
5.14.3.1 CyU3PCardCtx_t . . . . .	188
5.14.4 Enumeration Type Documentation . . . . .	188
5.14.4.1 CyU3PCardBusWidth_t . . . . .	188
5.14.4.2 CyU3PCardOpMode_t . . . . .	188
5.14.4.3 CyU3PSDCardVer_t . . . . .	189

5.14.4.4 CyU3PSdMmcStates_t . . . . .	189
5.14.4.5 CyU3PSibSDRegs_t . . . . .	189
5.14.5 Function Documentation . . . . .	189
5.14.5.1 CyU3PCardMgrCheckStatus(uint8_t portId) . . . . .	189
5.14.5.2 CyU3PCardMgrCompleteSDInit(uint8_t portId) . . . . .	190
5.14.5.3 CyU3PCardMgrContinueReadWrite(uint8_t isRead, uint8_t portId, uint8_t socket, uint16_t numBlks) . . . . .	190
5.14.5.4 CyU3PCardMgrDeInit(uint8_t portId) . . . . .	190
5.14.5.5 CyU3PCardMgrGetCSD(uint8_t portId, uint8_t *csd_buffer) . . . . .	191
5.14.5.6 CyU3PCardMgrInit(uint8_t portId) . . . . .	191
5.14.5.7 CyU3PCardMgrReadExtCsd(uint8_t portId, uint8_t *buffer_p) . . . . .	191
5.14.5.8 CyU3PCardMgrSendCmd(uint8_t portId, uint8_t cmd, uint8_t respLen, uint32_t cmdArg, uint32_t flags) . . . . .	192
5.14.5.9 CyU3PCardMgrSetClockFreq(uint8_t portId, uint16_t clkDivider, CyBool_t halfDiv) . . . . .	192
5.14.5.10 CyU3PCardMgrSetupRead(uint8_t portId, uint8_t unit, uint8_t socket, uint16_t numReadBlks, uint32_t blkAddr) . . . . .	192
5.14.5.11 CyU3PCardMgrSetupWrite(uint8_t portId, uint8_t unit, uint8_t socket, uint16_t numWriteBlks, uint32_t blkAddr) . . . . .	193
5.14.5.12 CyU3PCardMgrStopTransfer(uint8_t portId) . . . . .	193
5.14.5.13 CyU3PCardMgrWaitForInterrupt(uint8_t portId, uint32_t intr, uint32_t timeout) . . . . .	193
5.15 firmware/u3p_firmware/inc/cyu3cardmgr_fx3s.h File Reference . . . . .	194
5.15.1 Detailed Description . . . . .	196
5.15.2 Macro Definition Documentation . . . . .	196
5.15.2.1 CY_U3P_SDIO_CARD_CAPABILITY_4BLS . . . . .	196
5.15.2.2 CY_U3P_SDIO_CARD_CAPABILITY_E4MI . . . . .	196
5.15.2.3 CY_U3P_SDIO_CARD_CAPABILITY_LSC . . . . .	196
5.15.2.4 CY_U3P_SDIO_CARD_CAPABILITY_S4MI . . . . .	197
5.15.2.5 CY_U3P_SDIO_CARD_CAPABILITY_SBS . . . . .	197
5.15.2.6 CY_U3P_SDIO_CARD_CAPABILITY_SDC . . . . .	197
5.15.2.7 CY_U3P_SDIO_CARD_CAPABILITY_SMB . . . . .	197
5.15.2.8 CY_U3P_SDIO_CARD_CAPABILITY_SRW . . . . .	197
5.15.2.9 CY_U3P_SDIO_CCCR_Version_1_00 . . . . .	197
5.15.2.10 CY_U3P_SDIO_CCCR_Version_1_10 . . . . .	197
5.15.2.11 CY_U3P_SDIO_CCCR_Version_2_00 . . . . .	197
5.15.2.12 CY_U3P_SDIO_CCCR_Version_3_00 . . . . .	197
5.15.2.13 CY_U3P_SDIO_CHECK_INT_ENABLE_REG . . . . .	197
5.15.2.14 CY_U3P_SDIO_CIA_FUNCTION . . . . .	197
5.15.2.15 CY_U3P_SDIO_CISTPL_END . . . . .	197
5.15.2.16 CY_U3P_SDIO_CISTPL_FUNCE . . . . .	198
5.15.2.17 CY_U3P_SDIO_CISTPL_MANFID . . . . .	198
5.15.2.18 CY_U3P_SDIO_CISTPL_NULL . . . . .	198

5.15.2.19 CY_U3P_SDIO_DISABLE_INT . . . . .	198
5.15.2.20 CY_U3P_SDIO_EAI . . . . .	198
5.15.2.21 CY_U3P_SDIO_ENABLE_ASYNC_INT . . . . .	198
5.15.2.22 CY_U3P_SDIO_ENABLE_HIGH_SPEED . . . . .	198
5.15.2.23 CY_U3P_SDIO_ENABLE_INT . . . . .	198
5.15.2.24 CY_U3P_SDIO_FULL_SPEED . . . . .	198
5.15.2.25 CY_U3P_SDIO_HIGH_SPEED . . . . .	198
5.15.2.26 CY_U3P_SDIO_INT_MASTER . . . . .	198
5.15.2.27 CY_U3P_SDIO_INTFC_BT_A . . . . .	198
5.15.2.28 CY_U3P_SDIO_INTFC_BT_A_AMP . . . . .	199
5.15.2.29 CY_U3P_SDIO_INTFC_BT_B . . . . .	199
5.15.2.30 CY_U3P_SDIO_INTFC_CAM . . . . .	199
5.15.2.31 CY_U3P_SDIO_INTFC_EMBD_ATA . . . . .	199
5.15.2.32 CY_U3P_SDIO_INTFC_GPS . . . . .	199
5.15.2.33 CY_U3P_SDIO_INTFC_NONE . . . . .	199
5.15.2.34 CY_U3P_SDIO_INTFC_PHS . . . . .	199
5.15.2.35 CY_U3P_SDIO_INTFC_UART . . . . .	199
5.15.2.36 CY_U3P_SDIO_INTFC_WLAN . . . . .	199
5.15.2.37 CY_U3P_SDIO_LOW_SPEED . . . . .	199
5.15.2.38 CY_U3P_SDIO_READ_AFTER_WRITE . . . . .	199
5.15.2.39 CY_U3P_SDIO_REG_BUS_INTERFACE_CONTROL . . . . .	199
5.15.2.40 CY_U3P_SDIO_REG_BUS_SUSPEND . . . . .	200
5.15.2.41 CY_U3P_SDIO_REG_CARD_CAPABILITY . . . . .	200
5.15.2.42 CY_U3P_SDIO_REG_CCCR_HIGH_SPEED . . . . .	200
5.15.2.43 CY_U3P_SDIO_REG_CCCR_REVISION . . . . .	200
5.15.2.44 CY_U3P_SDIO_REG_CIS_PTR_D0 . . . . .	200
5.15.2.45 CY_U3P_SDIO_REG_CIS_PTR_D1 . . . . .	200
5.15.2.46 CY_U3P_SDIO_REG_CIS_PTR_D2 . . . . .	200
5.15.2.47 CY_U3P_SDIO_REG_DRIVER_STRENGTH . . . . .	200
5.15.2.48 CY_U3P_SDIO_REG_EXEC_FLAGS . . . . .	200
5.15.2.49 CY_U3P_SDIO_REG_FBR_CIS_PTR_D1 . . . . .	200
5.15.2.50 CY_U3P_SDIO_REG_FBR_CIS_PTR_D2 . . . . .	200
5.15.2.51 CY_U3P_SDIO_REG_FBR_CIS_PTR_DO . . . . .	200
5.15.2.52 CY_U3P_SDIO_REG_FBR_CSA_PTR_D1 . . . . .	201
5.15.2.53 CY_U3P_SDIO_REG_FBR_CSA_PTR_D2 . . . . .	201
5.15.2.54 CY_U3P_SDIO_REG_FBR_CSA_PTR_DO . . . . .	201
5.15.2.55 CY_U3P_SDIO_REG_FBR_DATA_ACCESS_WINDOW . . . . .	201
5.15.2.56 CY_U3P_SDIO_REG_FBR_EXT_INTERFACE_CODE . . . . .	201
5.15.2.57 CY_U3P_SDIO_REG_FBR_INTERFACE_CODE . . . . .	201
5.15.2.58 CY_U3P_SDIO_REG_FBR_IO_BLOCKSIZE_D0 . . . . .	201

5.15.2.59 CY_U3P_SDIO_REG_FBR_IO_BLOCKSIZE_D1 . . . . .	201
5.15.2.60 CY_U3P_SDIO_REG_FBR_POWER_SELECT . . . . .	201
5.15.2.61 CY_U3P_SDIO_REG_FUNCTION_BLOCKSIZE_D0 . . . . .	201
5.15.2.62 CY_U3P_SDIO_REG_FUNCTION_BLOCKSIZE_D1 . . . . .	201
5.15.2.63 CY_U3P_SDIO_REG_FUNCTION_SELECT . . . . .	201
5.15.2.64 CY_U3P_SDIO_REG_INTERRUPT_EXTENSION . . . . .	202
5.15.2.65 CY_U3P_SDIO_REG_IO_ABORT . . . . .	202
5.15.2.66 CY_U3P_SDIO_REG_IO_ENABLE . . . . .	202
5.15.2.67 CY_U3P_SDIO_REG_IO_INTR_ENABLE . . . . .	202
5.15.2.68 CY_U3P_SDIO_REG_IO_INTR_PENDING . . . . .	202
5.15.2.69 CY_U3P_SDIO_REG_IO_READY . . . . .	202
5.15.2.70 CY_U3P_SDIO_REG_POWER_CONTROL . . . . .	202
5.15.2.71 CY_U3P_SDIO_REG_READY_FLAGS . . . . .	202
5.15.2.72 CY_U3P_SDIO_REG_SD_SPEC_REVISION . . . . .	202
5.15.2.73 CY_U3P_SDIO_REG_UHS_I_SUPPORT . . . . .	202
5.15.2.74 CY_U3P_SDIO_RESET . . . . .	202
5.15.2.75 CY_U3P_SDIO_SAI . . . . .	202
5.15.2.76 CY_U3P_SDIO_SD_Version_1_00 . . . . .	203
5.15.2.77 CY_U3P_SDIO_SD_Version_1_10 . . . . .	203
5.15.2.78 CY_U3P_SDIO_SD_Version_2_00 . . . . .	203
5.15.2.79 CY_U3P_SDIO_SD_Version_3_00 . . . . .	203
5.15.2.80 CY_U3P_SDIO_SDDR50_SPEED . . . . .	203
5.15.2.81 CY_U3P_SDIO_SDDR104_SPEED . . . . .	203
5.15.2.82 CY_U3P_SDIO_SDDR12_SPEED . . . . .	203
5.15.2.83 CY_U3P_SDIO_SDDR25_SPEED . . . . .	203
5.15.2.84 CY_U3P_SDIO_SDDR50_SPEED . . . . .	203
5.15.2.85 CY_U3P_SDIO_SUPPORT_HIGH_SPEED . . . . .	203
5.15.2.86 CY_U3P_SDIO_UHS_SDDR50 . . . . .	203
5.15.2.87 CY_U3P_SDIO_UHS_SDDR104 . . . . .	203
5.15.2.88 CY_U3P_SDIO_UHS_SDDR50 . . . . .	204
5.15.2.89 CY_U3P_SDIO_Version_1_00 . . . . .	204
5.15.2.90 CY_U3P_SDIO_Version_1_10 . . . . .	204
5.15.2.91 CY_U3P_SDIO_Version_1_20 . . . . .	204
5.15.2.92 CY_U3P_SDIO_Version_2_00 . . . . .	204
5.15.2.93 CY_U3P_SDIO_Version_3_00 . . . . .	204
5.16 firmware/u3p_firmware/inc/cyu3descriptor.h File Reference . . . . .	204
5.16.1 Detailed Description . . . . .	205
5.16.2 Descriptor Functions . . . . .	205
5.16.3 Typedef Documentation . . . . .	206
5.16.3.1 CyU3PDmaDescriptor_t . . . . .	206

---

5.16.4 Function Documentation . . . . .	206
5.16.4.1 CyU3PDmaDscrChainCreate(uint16_t *dscrIndex_p, uint16_t count, uint16_t bufferSize, uint32_t dscrSync) . . . . .	206
5.16.4.2 CyU3PDmaDscrChainDestroy(uint16_t dscrIndex, uint16_t count, CyBool_t isProdChain, CyBool_t freeBuffer) . . . . .	207
5.16.4.3 CyU3PDmaDscrGet(uint16_t *index_p) . . . . .	207
5.16.4.4 CyU3PDmaDscrGetConfig(uint16_t index, CyU3PDmaDescriptor_t *dscr_p) . . . . .	207
5.16.4.5 CyU3PDmaDscrGetFreeCount(void) . . . . .	208
5.16.4.6 CyU3PDmaDscrListCreate(void) . . . . .	208
5.16.4.7 CyU3PDmaDscrListDestroy(void) . . . . .	208
5.16.4.8 CyU3PDmaDscrPut(uint16_t index) . . . . .	209
5.16.4.9 CyU3PDmaDscrSetConfig(uint16_t index, CyU3PDmaDescriptor_t *dscr_p) . . . . .	209
5.16.5 Variable Documentation . . . . .	210
5.16.5.1 gIDmaDescriptor . . . . .	210
5.17 firmware/u3p_firmware/inc/cyu3dma.h File Reference . . . . .	210
5.17.1 Detailed Description . . . . .	216
5.17.2 Typedef Documentation . . . . .	216
5.17.2.1 CyU3PDmaBuffer_t . . . . .	216
5.17.2.2 CyU3PDmaCallback_t . . . . .	216
5.17.2.3 CyU3PDmaCBInput_t . . . . .	217
5.17.2.4 CyU3PDmaCbType_t . . . . .	217
5.17.2.5 CyU3PDmaChannelConfig_t . . . . .	217
5.17.2.6 CyU3PDmaMode_t . . . . .	218
5.17.2.7 CyU3PDmaMultiCallback_t . . . . .	218
5.17.2.8 CyU3PDmaMultiChannelConfig_t . . . . .	218
5.17.2.9 CyU3PDmaMultiType_t . . . . .	219
5.17.2.10 CyU3PDmaSckSuspType_t . . . . .	219
5.17.2.11 CyU3PDmaSocketId_t . . . . .	220
5.17.2.12 CyU3PDmaState_t . . . . .	220
5.17.2.13 CyU3PDmaType_t . . . . .	220
5.17.3 Enumeration Type Documentation . . . . .	220
5.17.3.1 CyU3PDmaCbType_t . . . . .	220
5.17.3.2 CyU3PDmaMode_t . . . . .	221
5.17.3.3 CyU3PDmaMultiType_t . . . . .	221
5.17.3.4 CyU3PDmaSckSuspType_t . . . . .	222
5.17.3.5 CyU3PDmaSocketId_t . . . . .	223
5.17.3.6 CyU3PDmaState_t . . . . .	225
5.17.3.7 CyU3PDmaType_t . . . . .	226
5.17.4 Function Documentation . . . . .	226
5.17.4.1 CyU3PDmaChannelAbort(CyU3PDmaChannel *handle) . . . . .	226

5.17.4.2 CyU3PDmaChannelCacheControl(CyU3PDmaChannel *handle, CyBool_t isDmaHandleDCache) . . . . .	227
5.17.4.3 CyU3PDmaChannelCommitBuffer(CyU3PDmaChannel *handle, uint16_t count, uint16_t bufStatus) . . . . .	227
5.17.4.4 CyU3PDmaChannelCreate(CyU3PDmaChannel *handle, CyU3PDmaType_t type, CyU3PDmaChannelConfig_t *config) . . . . .	228
5.17.4.5 CyU3PDmaChannelDestroy(CyU3PDmaChannel *handle) . . . . .	229
5.17.4.6 CyU3PDmaChannelDiscardBuffer(CyU3PDmaChannel *handle) . . . . .	229
5.17.4.7 CyU3PDmaChannelGetBuffer(CyU3PDmaChannel *handle, CyU3PDmaBuffer_t *buffer_p, uint32_t waitOption) . . . . .	230
5.17.4.8 CyU3PDmaChannelGetHandle(CyU3PDmaSocketId_t sckId) . . . . .	231
5.17.4.9 CyU3PDmaChannelGetStatus(CyU3PDmaChannel *handle, CyU3PDmaState_t *state, uint32_t *prodXferCount, uint32_t *consXferCount) . . . . .	231
5.17.4.10 CyU3PDmaChannelsValid(CyU3PDmaChannel *handle) . . . . .	232
5.17.4.11 CyU3PDmaChannelReset(CyU3PDmaChannel *handle) . . . . .	232
5.17.4.12 CyU3PDmaChannelResume(CyU3PDmaChannel *handle, CyBool_t isProdResume, CyBool_t isConsResume) . . . . .	232
5.17.4.13 CyU3PDmaChannelResumeUsbConsumer(CyU3PDmaChannel *handle) . . . . .	233
5.17.4.14 CyU3PDmaChannelSetSuspend(CyU3PDmaChannel *handle, CyU3PDmaSckSuspType_t prodSusp, CyU3PDmaSckSuspType_t consSusp) . . . . .	233
5.17.4.15 CyU3PDmaChannelSetupRecvBuffer(CyU3PDmaChannel *handle, CyU3PDmaBuffer_t *buffer_p) . . . . .	234
5.17.4.16 CyU3PDmaChannelSetupSendBuffer(CyU3PDmaChannel *handle, CyU3PDmaBuffer_t *buffer_p) . . . . .	235
5.17.4.17 CyU3PDmaChannelSetWrapUp(CyU3PDmaChannel *handle) . . . . .	235
5.17.4.18 CyU3PDmaChannelSetXfer(CyU3PDmaChannel *handle, uint32_t count) . . . . .	236
5.17.4.19 CyU3PDmaChannelSuspendUsbConsumer(CyU3PDmaChannel *handle, uint32_t waitOption) . . . . .	237
5.17.4.20 CyU3PDmaChannelUpdateMode(CyU3PDmaChannel *handle, CyU3PDmaMode_t dmaMode) . . . . .	237
5.17.4.21 CyU3PDmaChannelWaitForCompletion(CyU3PDmaChannel *handle, uint32_t waitOption) . . . . .	238
5.17.4.22 CyU3PDmaChannelWaitForRecvBuffer(CyU3PDmaChannel *handle, CyU3PDmaBuffer_t *buffer_p, uint32_t waitOption) . . . . .	238
5.17.4.23 CyU3PDmaEnableMulticast(void) . . . . .	239
5.17.4.24 CyU3PDmaMulticastDisableConsumer(CyU3PDmaMultiChannel *chHandle, uint16_t consIndex) . . . . .	239
5.17.4.25 CyU3PDmaMulticastSocketSelect(CyU3PDmaMultiChannel *chHandle, uint32_t consMask) . . . . .	239
5.17.4.26 CyU3PDmaMultiChannelAbort(CyU3PDmaMultiChannel *handle) . . . . .	240
5.17.4.27 CyU3PDmaMultiChannelCacheControl(CyU3PDmaMultiChannel *handle, CyBool_t isDmaHandleDCache) . . . . .	240
5.17.4.28 CyU3PDmaMultiChannelCommitBuffer(CyU3PDmaMultiChannel *handle, uint16_t count, uint16_t bufStatus) . . . . .	241

5.17.4.29 CyU3PDmaMultiChannelCreate(CyU3PDmaMultiChannel *handle, CyU3P ← DmaMultiType_t type, CyU3PDmaMultiChannelConfig_t *config) . . . . .	242
5.17.4.30 CyU3PDmaMultiChannelDestroy(CyU3PDmaMultiChannel *handle) . . . . .	242
5.17.4.31 CyU3PDmaMultiChannelDiscardBuffer(CyU3PDmaMultiChannel *handle) . . . . .	242
5.17.4.32 CyU3PDmaMultiChannelGetBuffer(CyU3PDmaMultiChannel *handle, CyU3P ← DmaBuffer_t *buffer_p, uint32_t waitOption) . . . . .	243
5.17.4.33 CyU3PDmaMultiChannelGetHandle(CyU3PDmaSocketId_t sckId) . . . . .	244
5.17.4.34 CyU3PDmaMultiChannelGetStatus(CyU3PDmaMultiChannel *handle, CyU3P ← DmaState_t *state, uint32_t *prodXferCount, uint32_t *consXferCount, uint8_t sckIndex) . . . . .	244
5.17.4.35 CyU3PDmaMultiChannelIsValid(CyU3PDmaMultiChannel *handle) . . . . .	245
5.17.4.36 CyU3PDmaMultiChannelReset(CyU3PDmaMultiChannel *handle) . . . . .	245
5.17.4.37 CyU3PDmaMultiChannelResume(CyU3PDmaMultiChannel *handle, CyBool_t isProdResume, CyBool_t isConsResume) . . . . .	246
5.17.4.38 CyU3PDmaMultiChannelResumeUsbConsumer(CyU3PDmaMultiChannel *handle) . . . . .	246
5.17.4.39 CyU3PDmaMultiChannelSetSuspend(CyU3PDmaMultiChannel *handle, Cy ← U3PDmaSckSuspType_t prodSusp, CyU3PDmaSckSuspType_t consSusp) . . . . .	247
5.17.4.40 CyU3PDmaMultiChannelSetupRecvBuffer(CyU3PDmaMultiChannel *handle, CyU3PDmaBuffer_t *buffer_p, uint16_t multiSckOffset) . . . . .	247
5.17.4.41 CyU3PDmaMultiChannelSetupSendBuffer(CyU3PDmaMultiChannel *handle, CyU3PDmaBuffer_t *buffer_p, uint16_t multiSckOffset) . . . . .	248
5.17.4.42 CyU3PDmaMultiChannelSetWrapUp(CyU3PDmaMultiChannel *handle, uint16 ← _t multiSckOffset) . . . . .	249
5.17.4.43 CyU3PDmaMultiChannelSetXfer(CyU3PDmaMultiChannel *handle, uint32_t count, uint16_t multiSckOffset) . . . . .	249
5.17.4.44 CyU3PDmaMultiChannelSuspendUsbConsumer(CyU3PDmaMultiChannel *handle, uint32_t waitOption) . . . . .	250
5.17.4.45 CyU3PDmaMultiChannelUpdateMode(CyU3PDmaMultiChannel *handle, Cy ← U3PDmaMode_t dmaMode) . . . . .	250
5.17.4.46 CyU3PDmaMultiChannelWaitForCompletion(CyU3PDmaMultiChannel *handle, uint32_t waitOption) . . . . .	251
5.17.4.47 CyU3PDmaMultiChannelWaitForRecvBuffer(CyU3PDmaMultiChannel *handle, CyU3PDmaBuffer_t *buffer_p, uint32_t waitOption) . . . . .	251
5.17.4.48 CyU3PDmaUsbInEpGetChannel(uint8_t ep) . . . . .	252
5.17.4.49 CyU3PDmaUsbInEpGetMultiChannel(uint8_t ep) . . . . .	252
5.18 firmware/u3p_firmware/inc/cyu3error.h File Reference . . . . .	253
5.18.1 Detailed Description . . . . .	254
5.18.2 Error Codes . . . . .	254
5.18.3 Typedef Documentation . . . . .	254
5.18.3.1 CyU3PErrorCode_t . . . . .	254
5.18.4 Enumeration Type Documentation . . . . .	254
5.18.4.1 CyU3PErrorCode_t . . . . .	254
5.19 firmware/u3p_firmware/inc/cyu3gpif.h File Reference . . . . .	256

5.19.1	Detailed Description	259
5.19.2	Typedef Documentation	259
5.19.2.1	CyU3PGpifComparatorType	259
5.19.2.2	CyU3PGpifConfig_t	259
5.19.2.3	CyU3PGpifEventCb_t	260
5.19.2.4	CyU3PGpifEventType	260
5.19.2.5	CyU3PGpifOutput_t	260
5.19.2.6	CyU3PGpifSMIntrCb_t	260
5.19.2.7	CyU3PGpifWaveData	261
5.19.3	Enumeration Type Documentation	261
5.19.3.1	CyU3PGpifComparatorType	261
5.19.3.2	CyU3PGpifEventType	261
5.19.3.3	CyU3PGpifOutput_t	262
5.19.4	Function Documentation	262
5.19.4.1	CyU3PGpifConfigure(uint8_t numRegs, const uint32_t *regData)	262
5.19.4.2	CyU3PGpifControlSWInput(CyBool_t set)	263
5.19.4.3	CyU3PGpifDisable(CyBool_t forceReload)	263
5.19.4.4	CyU3PGpifGetSMState(uint8_t *curState_p)	264
5.19.4.5	CyU3PGpifInitAddrCounter(uint32_t initialValue, uint32_t limit, CyBool_t reload, CyBool_t upCount, uint8_t increment)	264
5.19.4.6	CyU3PGpifInitComparator(CyU3PGpifComparatorType type, uint32_t value, uint32_t mask)	264
5.19.4.7	CyU3PGpifInitCtrlCounter(uint16_t initialValue, uint16_t limit, CyBool_t reload, CyBool_t upCount, uint8_t outputBit)	265
5.19.4.8	CyU3PGpifInitDataCounter(uint32_t initialValue, uint32_t limit, CyBool_t reload, CyBool_t upCount, uint8_t increment)	265
5.19.4.9	CyU3PGpifInitTransFunctions(uint16_t *fnTable)	266
5.19.4.10	CyU3PGpifLoad(const CyU3PGpifConfig_t *conf)	266
5.19.4.11	CyU3PGpifOutputConfigure(uint8_t ctrlPin, CyU3PGpifOutput_t opFlag, CyBool_t isActiveLow)	266
5.19.4.12	CyU3PGpifReadDataWords(uint32_t threadIndex, CyBool_t selectThread, uint32_t numWords, uint32_t *buffer_p, uint32_t waitOption)	267
5.19.4.13	CyU3PGpifRegisterCallback(CyU3PGpifEventCb_t cbFunc)	267
5.19.4.14	CyU3PGpifRegisterConfig(uint16_t numRegs, uint32_t regData[][2])	268
5.19.4.15	CyU3PGpifRegisterSMIntrCallback(CyU3PGpifSMIntrCb_t cb)	268
5.19.4.16	CyU3PGpifSMControl(CyBool_t pause)	269
5.19.4.17	CyU3PGpifSMStart(uint8_t stateIndex, uint8_t initialAlpha)	269
5.19.4.18	CyU3PGpifSMSwitch(uint16_t fromState, uint16_t toState, uint16_t endState, uint8_t initialAlpha, uint32_t switchTimeout)	269
5.19.4.19	CyU3PGpifSocketConfigure(uint8_t threadIndex, CyU3PDmaSocketId_t socketNum, uint16_t watermark, CyBool_t flagOnData, uint8_t burst)	270
5.19.4.20	CyU3PGpifWaveformLoad(uint8_t firstState, uint16_t stateCnt, uint8_t *stateDataMap, CyU3PGpifWaveData *transitionData)	271

5.19.4.21 CyU3PGpifWriteDataWords(uint32_t threadIndex, CyBool_t selectThread, uint32_t numWords, uint32_t *buffer_p, uint32_t waitOption) . . . . .	271
5.20 firmware/u3p_firmware/inc/cyu3gpio.h File Reference . . . . .	272
5.20.1 Detailed Description . . . . .	274
5.20.2 Typedef Documentation . . . . .	274
5.20.2.1 CyU3PGpioComplexConfig_t . . . . .	274
5.20.2.2 CyU3PGpioComplexMode_t . . . . .	274
5.20.2.3 CyU3PGpioIntrCb_t . . . . .	275
5.20.2.4 CyU3PGpioIntrMode_t . . . . .	275
5.20.2.5 CyU3PGpioSimpleConfig_t . . . . .	275
5.20.2.6 CyU3PGpioTimerMode_t . . . . .	275
5.20.3 Enumeration Type Documentation . . . . .	276
5.20.3.1 CyU3PGpioComplexMode_t . . . . .	276
5.20.3.2 CyU3PGpioIntrMode_t . . . . .	277
5.20.3.3 CyU3PGpioTimerMode_t . . . . .	277
5.20.4 Function Documentation . . . . .	278
5.20.4.1 CyU3PGpioComplexGetThreshold(uint8_t gpioid, uint32_t *threshold_p) . . . . .	278
5.20.4.2 CyU3PGpioComplexMeasureOnce(uint8_t gpioid, CyU3PGpioComplexMode_t pinMode) . . . . .	278
5.20.4.3 CyU3PGpioComplexPulse(uint8_t gpioid, uint32_t threshold) . . . . .	279
5.20.4.4 CyU3PGpioComplexPulseNow(uint8_t gpioid, uint32_t threshold) . . . . .	280
5.20.4.5 CyU3PGpioComplexSampleNow(uint8_t gpioid, uint32_t *value_p) . . . . .	280
5.20.4.6 CyU3PGpioComplexUpdate(uint8_t gpioid, uint32_t threshold, uint32_t period) . . . . .	281
5.20.4.7 CyU3PGpioComplexWaitForCompletion(uint8_t gpioid, uint32_t *threshold_p, CyBool_t isWait) . . . . .	281
5.20.4.8 CyU3PGpioDeInit(void) . . . . .	282
5.20.4.9 CyU3PGpioDisable(uint8_t gpioid) . . . . .	282
5.20.4.10 CyU3PGpioGetIOValues(uint32_t *gpioVal0_p, uint32_t *gpioVal1_p) . . . . .	282
5.20.4.11 CyU3PGpioGetValue(uint8_t gpioid, CyBool_t *value_p) . . . . .	283
5.20.4.12 CyU3PGpioInit(CyU3PGpioClock_t *clk_p, CyU3PGpioIntrCb_t irq) . . . . .	283
5.20.4.13 CyU3PGpioSetComplexConfig(uint8_t gpioid, CyU3PGpioComplexConfig_t *cfg_p) . . . . .	284
5.20.4.14 CyU3PGpioSetSimpleConfig(uint8_t gpioid, CyU3PGpioSimpleConfig_t *cfg_p) . . . . .	285
5.20.4.15 CyU3PGpioSetValue(uint8_t gpioid, CyBool_t value) . . . . .	285
5.20.4.16 CyU3PGpioSimpleGetValue(uint8_t gpioid, CyBool_t *value_p) . . . . .	286
5.20.4.17 CyU3PGpioSimpleSetValue(uint8_t gpioid, CyBool_t value) . . . . .	286
5.20.4.18 CyU3PRegisterGpioCallBack(CyU3PGpioIntrCb_t gpiointrCb) . . . . .	287
5.21 firmware/u3p_firmware/inc/cyu3i2c.h File Reference . . . . .	287
5.21.1 Detailed Description . . . . .	289
5.21.2 Typedef Documentation . . . . .	289
5.21.2.1 CyU3PI2cConfig_t . . . . .	289

5.21.2.2	CyU3PI2cError_t . . . . .	289
5.21.2.3	CyU3PI2cEvt_t . . . . .	290
5.21.2.4	CyU3PI2cIntrCb_t . . . . .	290
5.21.2.5	CyU3PI2cPreamble_t . . . . .	290
5.21.3	Enumeration Type Documentation . . . . .	291
5.21.3.1	CyU3PI2cError_t . . . . .	291
5.21.3.2	CyU3PI2cEvt_t . . . . .	292
5.21.4	Function Documentation . . . . .	292
5.21.4.1	CyU3PI2cDeInit(void) . . . . .	292
5.21.4.2	CyU3PI2cGetErrorCode(CyU3PI2cError_t *error_p) . . . . .	292
5.21.4.3	CyU3PI2cInit(void) . . . . .	293
5.21.4.4	CyU3PI2cReceiveBytes(CyU3PI2cPreamble_t *preamble, uint8_t *data, uint32_t byteCount, uint32_t retryCount) . . . . .	293
5.21.4.5	CyU3PI2cSendCommand(CyU3PI2cPreamble_t *preamble, uint32_t byteCount, CyBool_t isRead) . . . . .	294
5.21.4.6	CyU3PI2cSendStopCondition(void) . . . . .	295
5.21.4.7	CyU3PI2cSetConfig(CyU3PI2cConfig_t *config, CyU3PI2cIntrCb_t cb) . . . . .	295
5.21.4.8	CyU3PI2cSetTimeout(uint32_t readLoopCnt, uint32_t writeLoopCnt, uint32_t flushLoopCnt) . . . . .	295
5.21.4.9	CyU3PI2cTransmitBytes(CyU3PI2cPreamble_t *preamble, uint8_t *data, uint32_t byteCount, uint32_t retryCount) . . . . .	296
5.21.4.10	CyU3PI2cWaitForAck(CyU3PI2cPreamble_t *preamble, uint32_t retryCount) . . . . .	297
5.21.4.11	CyU3PI2cWaitForBlockXfer(CyBool_t isRead) . . . . .	297
5.21.4.12	CyU3PRegisterI2cCallBack(CyU3PI2cIntrCb_t i2cIntrCb) . . . . .	298
5.22	firmware/u3p_firmware/inc/cyu3i2s.h File Reference . . . . .	298
5.22.1	Detailed Description . . . . .	300
5.22.2	Typedef Documentation . . . . .	300
5.22.2.1	CyU3PI2sConfig_t . . . . .	300
5.22.2.2	CyU3PI2sError_t . . . . .	300
5.22.2.3	CyU3PI2sEvt_t . . . . .	301
5.22.2.4	CyU3PI2sIntrCb_t . . . . .	301
5.22.2.5	CyU3PI2sPadMode_t . . . . .	301
5.22.2.6	CyU3PI2sSampleRate_t . . . . .	301
5.22.2.7	CyU3PI2sSampleWidth_t . . . . .	301
5.22.3	Enumeration Type Documentation . . . . .	302
5.22.3.1	CyU3PI2sError_t . . . . .	302
5.22.3.2	CyU3PI2sEvt_t . . . . .	302
5.22.3.3	CyU3PI2sPadMode_t . . . . .	302
5.22.3.4	CyU3PI2sSampleRate_t . . . . .	303
5.22.3.5	CyU3PI2sSampleWidth_t . . . . .	303
5.22.4	Function Documentation . . . . .	303

---

5.22.4.1	CyU3PI2sDeInit(void) . . . . .	303
5.22.4.2	CyU3PI2sEnableExternalMclk(void) . . . . .	304
5.22.4.3	CyU3PI2sInit(void) . . . . .	304
5.22.4.4	CyU3PI2sSetConfig(CyU3PI2sConfig_t *config, CyU3PI2sIntrCb_t cb) . . . . .	305
5.22.4.5	CyU3PI2sSetMute(CyBool_t isMute) . . . . .	305
5.22.4.6	CyU3PI2sSetPause(CyBool_t isPause) . . . . .	306
5.22.4.7	CyU3PI2sTransmitBytes(uint8_t *lData, uint8_t *rData, uint8_t lByteCount, uint8_t rByteCount) . . . . .	306
5.22.4.8	CyU3PRegisterI2sCallBack(CyU3PI2sIntrCb_t i2sIntrCb) . . . . .	306
5.23	firmware/u3p_firmware/inc/cyu3lpp.h File Reference . . . . .	307
5.23.1	Detailed Description . . . . .	308
5.23.2	Typedef Documentation . . . . .	309
5.23.2.1	CyU3PGpioClock_t . . . . .	309
5.23.2.2	CyU3PGpioloMode_t . . . . .	309
5.23.2.3	CyU3PGpioSimpleClkDiv_t . . . . .	309
5.23.2.4	CyU3PLppInterruptHandler . . . . .	309
5.23.3	Enumeration Type Documentation . . . . .	310
5.23.3.1	CyU3PGpioloMode_t . . . . .	310
5.23.3.2	CyU3PGpioSimpleClkDiv_t . . . . .	310
5.23.4	Function Documentation . . . . .	310
5.23.4.1	CyU3PGpioSetClock(CyU3PGpioClock_t *clk_p) . . . . .	310
5.23.4.2	CyU3PGpioSetIoMode(uint8_t gpioid, CyU3PGpioloMode_t ioMode) . . . . .	312
5.23.4.3	CyU3PGpioStopClock(void) . . . . .	312
5.23.4.4	CyU3PI2cSetClock(uint32_t bitRate) . . . . .	312
5.23.4.5	CyU3PI2cStopClock(void) . . . . .	313
5.23.4.6	CyU3PI2sSetClock(uint32_t clkRate) . . . . .	313
5.23.4.7	CyU3PI2sStopClock(void) . . . . .	313
5.23.4.8	CyU3PLppDeInit(CyU3PLppModule_t lppModule) . . . . .	314
5.23.4.9	CyU3PLppGpioBlockIsOn(void) . . . . .	314
5.23.4.10	CyU3PLppInit(CyU3PLppModule_t lppModule, CyU3PLppInterruptHandler intrHandler) . . . . .	314
5.23.4.11	CyU3PSetGpioDriveStrength(CyU3PDriveStrengthState_t gpioDriveStrength) . . . . .	315
5.23.4.12	CyU3PSetI2cDriveStrength(CyU3PDriveStrengthState_t i2cDriveStrength) . . . . .	315
5.23.4.13	CyU3PSpiSetClock(uint32_t clock) . . . . .	316
5.23.4.14	CyU3PSpiStopClock(void) . . . . .	316
5.23.4.15	CyU3PUartSetClock(uint32_t baudRate) . . . . .	316
5.23.4.16	CyU3PUartStopClock(void) . . . . .	317
5.24	firmware/u3p_firmware/inc/cyu3mbox.h File Reference . . . . .	317
5.24.1	Detailed Description . . . . .	318
5.24.2	Typedef Documentation . . . . .	318

5.24.2.1	CyU3PMbox . . . . .	318
5.24.2.2	CyU3PMboxCb_t . . . . .	318
5.24.3	Function Documentation . . . . .	319
5.24.3.1	CyU3PMboxDeInit(void) . . . . .	319
5.24.3.2	CyU3PMboxInit(CyU3PMboxCb_t callback) . . . . .	319
5.24.3.3	CyU3PMboxRead(CyU3PMbox *mbox) . . . . .	319
5.24.3.4	CyU3PMboxReset(void) . . . . .	319
5.24.3.5	CyU3PMboxWait(void) . . . . .	320
5.24.3.6	CyU3PMboxWrite(CyU3PMbox *mbox) . . . . .	320
5.24.4	Variable Documentation . . . . .	320
5.24.4.1	gIMboxCb . . . . .	320
5.25	firmware/u3p_firmware/inc/cyu3mipicsi.h File Reference . . . . .	320
5.25.1	Detailed Description . . . . .	323
5.25.2	Macro Definition Documentation . . . . .	323
5.25.2.1	ALPHA_CX3_START_SCK0 . . . . .	323
5.25.2.2	ALPHA_CX3_START_SCK1 . . . . .	323
5.25.2.3	CX3_ALPHA_START . . . . .	324
5.25.2.4	CX3_FULL_BUFFER_IN_SCK0 . . . . .	324
5.25.2.5	CX3_FULL_BUFFER_IN_SCK1 . . . . .	324
5.25.2.6	CX3_IDLE . . . . .	324
5.25.2.7	CX3_IDLE_SCK0 . . . . .	324
5.25.2.8	CX3_IDLE_SCK1 . . . . .	324
5.25.2.9	CX3_NUMBER_OF_STATES . . . . .	324
5.25.2.10	CX3_PARTIAL_BUFFER_IN_SCK0 . . . . .	324
5.25.2.11	CX3_PARTIAL_BUFFER_IN_SCK1 . . . . .	324
5.25.2.12	CX3_PUSH_DATA_SCK0 . . . . .	324
5.25.2.13	CX3_PUSH_DATA_SCK1 . . . . .	324
5.25.2.14	CX3_PUSH_DATA_TO_SCK0 . . . . .	324
5.25.2.15	CX3_PUSH_DATA_TO_SCK1 . . . . .	325
5.25.2.16	CX3_START . . . . .	325
5.25.2.17	CX3_START_SCK0 . . . . .	325
5.25.3	Fixed Function GPIF States . . . . .	325
5.25.3.1	CX3_START_SCK1 . . . . .	325
5.25.3.2	CX3_WAIT_FOR_FRAME_START . . . . .	325
5.25.3.3	CX3_WAIT_FOR_FRAME_START_SCK0 . . . . .	325
5.25.3.4	CX3_WAIT_FOR_FRAME_START_SCK1 . . . . .	325
5.25.3.5	CX3_WAIT_FULL_SCK0_NEXT_SCK1 . . . . .	325
5.25.3.6	CX3_WAIT_FULL_SCK1_NEXT_SCK0 . . . . .	325
5.25.3.7	CX3_WAIT_TO_FILL_SCK0 . . . . .	325
5.25.3.8	CX3_WAIT_TO_FILL_SCK1 . . . . .	325

5.25.3.9	FW_WAIT_SCK0	325
5.25.3.10	FW_WAIT_SCK1	326
5.25.4	Typedef Documentation	326
5.25.4.1	CyU3PMipicsiBusWidth_t	326
5.25.4.2	CyU3PMipicsiCfg_t	326
5.25.4.3	CyU3PMipicsiDataFormat_t	326
5.25.4.4	CyU3PMipicsiErrorCounts_t	327
5.25.4.5	CyU3PMipicsiI2cFreq_t	327
5.25.4.6	CyU3PMipicsiPllClkDiv_t	327
5.25.4.7	CyU3PMipicsiPllClkFrs_t	327
5.25.4.8	CyU3PMipicsiReset_t	328
5.25.4.9	CyU3PMipicsiSensorIo_t	328
5.25.5	Enumeration Type Documentation	328
5.25.5.1	CyU3PMipicsiBusWidth_t	328
5.25.5.2	CyU3PMipicsiDataFormat_t	328
5.25.5.3	CyU3PMipicsiI2cFreq_t	330
5.25.5.4	CyU3PMipicsiPllClkDiv_t	330
5.25.5.5	CyU3PMipicsiPllClkFrs_t	330
5.25.5.6	CyU3PMipicsiReset_t	331
5.25.5.7	CyU3PMipicsiSensorIo_t	331
5.25.6	Function Documentation	331
5.25.6.1	CyU3PCx3DeviceReset(CyBool_t isWarmReset, CyBool_t sensorResetHigh)	331
5.25.6.2	CyU3PMipicsiCheckBlockActive(void)	332
5.25.6.3	CyU3PMipicsiDeInit(void)	332
5.25.6.4	CyU3PMipicsiGetErrors(CyBool_t clrErrCnts, CyU3PMipicsiErrorCounts_t *errorCounts)	332
5.25.6.5	CyU3PMipicsiGpifLoad(CyU3PMipicsiBusWidth_t busWidth, uint32_t bufferSize)	333
5.25.6.6	CyU3PMipicsiInit(void)	333
5.25.6.7	CyU3PMipicsiInitializeGPIO(void)	334
5.25.6.8	CyU3PMipicsiInitializeI2c(CyU3PMipicsiI2cFreq_t freq)	334
5.25.6.9	CyU3PMipicsiInitializePIB(void)	335
5.25.6.10	CyU3PMipicsiQueryIntfParams(CyU3PMipicsiCfg_t *csiCfg)	335
5.25.6.11	CyU3PMipicsiReset(CyU3PMipicsiReset_t resetType)	336
5.25.6.12	CyU3PMipicsiSetIntfParams(CyU3PMipicsiCfg_t *csiCfg, CyBool_t wakeOnConfigure)	336
5.25.6.13	CyU3PMipicsiSetPhyTimeDelay(uint8_t tdTerm, uint8_t thsSettleDelay)	337
5.25.6.14	CyU3PMipicsiSetSensorControl(CyU3PMipicsiSensorIo_t io, CyBool_t value)	338
5.25.6.15	CyU3PMipicsiSleep(void)	338
5.25.6.16	CyU3PMipicsiWakeup(void)	338
5.26	firmware/u3p_firmware/inc/cyu3mmu.h File Reference	339

5.26.1	Detailed Description	341
5.26.2	Macro Definition Documentation	341
5.26.2.1	CYU3P_CACHE_LINE_SZ	341
5.26.2.2	CYU3P_CACHE_MMU_EN_MASK	341
5.26.2.3	CYU3P_CACHE_NWAYS	341
5.26.2.4	CYU3P_CACHE_REPLACEMENT_MASK	341
5.26.2.5	CYU3P_CACHE_SIZE	341
5.26.2.6	CYU3P_DCACHE_EN_MASK	341
5.26.2.7	CYU3P_DTCM_BASE_ADDR	341
5.26.2.8	CYU3P_DTCM_SIZE	341
5.26.2.9	CYU3P_DTCM_SZ_EN	341
5.26.2.10	CYU3P_GCTL_PAGE_TABLE_ADDR	342
5.26.2.11	CYU3P_ICACHE_EN_MASK	342
5.26.2.12	CYU3P_ITCM_BASE_ADDR	342
5.26.2.13	CYU3P_ITCM_SIZE	342
5.26.2.14	CYU3P_ITCM_SZ_EN	342
5.26.2.15	CYU3P_MMIO_BASE_ADDR	342
5.26.2.16	CYU3P_MMIO_SIZE	342
5.26.2.17	CYU3P_MMU_EN_MASK	342
5.26.2.18	CYU3P_ROM_BASE_ADDR	342
5.26.2.19	CYU3P_ROM_SIZE	342
5.26.2.20	CYU3P_SYSMEM_BASE_ADDR	342
5.26.2.21	CYU3P_SYSMEM_SIZE	342
5.26.2.22	CYU3P_TCMREG_ADDRESS_MASK	343
5.26.2.23	CYU3P_VIC_BASE_ADDR	343
5.26.2.24	CYU3P_VIC_SIZE	343
5.26.3	Function Documentation	343
5.26.3.1	CyU3PInitPageTable(void)	343
5.26.3.2	CyU3PSysBarrierSync(void)	343
5.26.3.3	CyU3PSysCacheDRegion(uint32_t *addr, uint32_t len)	343
5.26.3.4	CyU3PSysCacheIRegion(uint32_t *addr, uint32_t len)	344
5.26.3.5	CyU3PSysCleanDCache(void)	344
5.26.3.6	CyU3PSysCleanDRegion(uint32_t *addr, uint32_t len)	344
5.26.3.7	CyU3PSysClearDCache(void)	344
5.26.3.8	CyU3PSysClearDRegion(uint32_t *addr, uint32_t len)	344
5.26.3.9	CyU3PSysDisableCacheMMU(void)	345
5.26.3.10	CyU3PSysDisableDCache(void)	345
5.26.3.11	CyU3PSysDisableICache(void)	345
5.26.3.12	CyU3PSysDisableMMU(void)	346
5.26.3.13	CyU3PSysEnableCacheMMU(void)	346

5.26.3.14 CyU3PSysEnableDCache(void) . . . . .	346
5.26.3.15 CyU3PSysEnableICache(void) . . . . .	347
5.26.3.16 CyU3PSysEnableMMU(void) . . . . .	347
5.26.3.17 CyU3PSysFlushCaches(void) . . . . .	347
5.26.3.18 CyU3PSysFlushDCache(void) . . . . .	347
5.26.3.19 CyU3PSysFlushDRegion(uint32_t *addr, uint32_t len) . . . . .	347
5.26.3.20 CyU3PSysFlushICache(void) . . . . .	348
5.26.3.21 CyU3PSysFlushIRegion(uint32_t *addr, uint32_t len) . . . . .	348
5.26.3.22 CyU3PSysFlushTLBEntry(uint32_t *addr) . . . . .	348
5.26.3.23 CyU3PSysInitTCMs(void) . . . . .	349
5.26.3.24 CyU3PSysLoadTLB(void) . . . . .	349
5.26.3.25 CyU3PSysLockTLBEntry(uint32_t *addr) . . . . .	349
5.27 firmware/u3p_firmware/inc/cyu3os.h File Reference . . . . .	349
5.27.1 Detailed Description . . . . .	354
5.27.2 RTOS Constants . . . . .	354
5.27.3 Exceptions and Interrupts . . . . .	355
5.27.3.1 Exception Handler Functions . . . . .	356
5.27.4 RTOS Memory Functions . . . . .	356
5.27.5 Thread Functions . . . . .	356
5.27.6 Message Queue Functions . . . . .	356
5.27.7 Mutex functions . . . . .	357
5.27.8 Semaphore Functions . . . . .	357
5.27.9 Event Flag Functions . . . . .	357
5.27.10 Timer Functions . . . . .	357
5.27.11 Typedef Documentation . . . . .	357
5.27.11.1 CyU3PBlockPool . . . . .	357
5.27.11.2 CyU3PBytePool . . . . .	358
5.27.11.3 CyU3PEvent . . . . .	358
5.27.11.4 CyU3PMemCorruptCallback . . . . .	358
5.27.11.5 CyU3PMutex . . . . .	359
5.27.11.6 CyU3PQueue . . . . .	359
5.27.11.7 CyU3PSemaphore . . . . .	359
5.27.11.8 CyU3PThread . . . . .	359
5.27.11.9 CyU3PThreadEntry_t . . . . .	360
5.27.11.10 CyU3PTimer . . . . .	360
5.27.11.11 CyU3PTimerCb_t . . . . .	360
5.27.11.12 MemBlockInfo . . . . .	361
5.27.12 Function Documentation . . . . .	361
5.27.12.1 CyU3PAbortHandler(void) . . . . .	361
5.27.12.2 CyU3PApplicationDefine(void) . . . . .	361

5.27.12.3 CyU3PBlockAlloc(CyU3PBlockPool *pool_p, void **block_p, uint32_t waitOption)	361
5.27.12.4 CyU3PBlockFree(void *block_p) . . . . .	362
5.27.12.5 CyU3PBlockPoolCreate(CyU3PBlockPool *pool_p, uint32_t blockSize, void *poolStart, uint32_t poolSize) . . . . .	362
5.27.12.6 CyU3PBlockPoolDestroy(CyU3PBlockPool *pool_p) . . . . .	363
5.27.12.7 CyU3PBufCorruptionCheck(void) . . . . .	363
5.27.12.8 CyU3PBufEnableChecks(CyBool_t enable, CyU3PMemCorruptCallback cb) . . .	363
5.27.12.9 CyU3PBufGetActiveList(void) . . . . .	364
5.27.12.10 CyU3PBufGetCounts(uint32_t *allocCnt_p, uint32_t *freeCnt_p) . . . . .	364
5.27.12.11 CyU3PByteAlloc(CyU3PBytePool *pool_p, void **mem_p, uint32_t memSize, uint32_t waitOption) . . . . .	364
5.27.12.12 CyU3PByteFree(void *mem_p) . . . . .	365
5.27.12.13 CyU3PBytePoolCreate(CyU3PBytePool *pool_p, void *poolStart, uint32_t poolSize) . . . . .	365
5.27.12.14 CyU3PBytePoolDestroy(CyU3PBytePool *pool_p) . . . . .	365
5.27.12.15 CyU3PDmaBufferAlloc(uint16_t size) . . . . .	366
5.27.12.16 CyU3PDmaBufferDeInit(void) . . . . .	366
5.27.12.17 CyU3PDmaBufferFree(void *buffer) . . . . .	366
5.27.12.18 CyU3PDmaBufferInit(void) . . . . .	367
5.27.12.19 CyU3PEventCreate(CyU3PEvent *event_p) . . . . .	367
5.27.12.20 CyU3PEventDestroy(CyU3PEvent *event_p) . . . . .	368
5.27.12.21 CyU3PEventGet(CyU3PEvent *event_p, uint32_t rqtFlag, uint32_t getOption, uint32_t *flag_p, uint32_t waitOption) . . . . .	368
5.27.12.22 CyU3PEventPerfGet(CyU3PEvent *event_p, uint32_t *setCnt_p, uint32_t *getCnt_p, uint32_t *suspCnt_p, uint32_t *timeoutCnt_p) . . . . .	369
5.27.12.23 CyU3PEventSet(CyU3PEvent *event_p, uint32_t rqtFlag, uint32_t setOption) . . . . .	369
5.27.12.24 CyU3PEventSetActivityGpio(CyU3PEvent *event_p, uint32_t gpio_id) . . . . .	369
5.27.12.25 CyU3PEventSystemPerfGet(uint32_t *setCnt_p, uint32_t *getCnt_p, uint32_t *suspCnt_p, uint32_t *timeoutCnt_p) . . . . .	370
5.27.12.26 CyU3PFIqContextRestore(void) . . . . .	370
5.27.12.27 CyU3PFIqContextSave(void) . . . . .	370
5.27.12.28 CyU3PFreeHeaps(void) . . . . .	371
5.27.12.29 CyU3PGetTime(void) . . . . .	371
5.27.12.30 CyU3PIrqContextRestore(void) . . . . .	371
5.27.12.31 CyU3PIrqContextSave(void) . . . . .	372
5.27.12.32 CyU3PIrqNestingStart(void) . . . . .	372
5.27.12.33 CyU3PIrqNestingStop(void) . . . . .	372
5.27.12.34 CyU3PIrqVectoredContextSave(void) . . . . .	372
5.27.12.35 CyU3PKernelEntry(void) . . . . .	373
5.27.12.36 CyU3PMemAlloc(uint32_t size) . . . . .	373
5.27.12.37 CyU3PMemCmp(const void *s1, const void *s2, uint32_t n) . . . . .	373
5.27.12.38 CyU3PMemCopy(uint8_t *dest, uint8_t *src, uint32_t count) . . . . .	374

5.27.12.39CyU3PMemCorruptionCheck(void) . . . . .	374
5.27.12.40CyU3PMemEnableChecks(CyBool_t enable, CyU3PMemCorruptCallback cb) . . . . .	374
5.27.12.41CyU3PMemFree(void *mem_p) . . . . .	375
5.27.12.42CyU3PMemGetActiveList(void) . . . . .	375
5.27.12.43CyU3PMemGetCounts(uint32_t *allocCnt_p, uint32_t *freeCnt_p) . . . . .	375
5.27.12.44CyU3PMemInit(void) . . . . .	376
5.27.12.45CyU3PMemSet(uint8_t *ptr, uint8_t data, uint32_t count) . . . . .	376
5.27.12.46CyU3PMutexCreate(CyU3PMutex *mutex_p, uint32_t priorityInherit) . . . . .	376
5.27.12.47CyU3PMutexDestroy(CyU3PMutex *mutex_p) . . . . .	377
5.27.12.48CyU3PMutexGet(CyU3PMutex *mutex_p, uint32_t waitOption) . . . . .	377
5.27.12.49CyU3PMutexPerfGet(CyU3PMutex *mutex_p, uint32_t *putCnt_p, uint32_t *getCnt_p, uint32_t *suspCnt_p, uint32_t *timeoutCnt_p, uint32_t *invertCnt_p, uint32_t *inheritCnt_p) . . . . .	377
5.27.12.50CyU3PMutexPut(CyU3PMutex *mutex_p) . . . . .	378
5.27.12.51CyU3PMutexSetActivityGpio(CyU3PMutex *mutex_p, uint32_t gpio_id) . . . . .	378
5.27.12.52CyU3PMutexSystemPerfGet(uint32_t *putCnt_p, uint32_t *getCnt_p, uint32_t *suspCnt_p, uint32_t *timeoutCnt_p, uint32_t *invertCnt_p, uint32_t *inheritCnt_p) . . . . .	379
5.27.12.53CyU3POsTimerHandler(void) . . . . .	379
5.27.12.54CyU3POsTimerInit(uint16_t intervalMs) . . . . .	379
5.27.12.55CyU3PPrefetchHandler(void) . . . . .	380
5.27.12.56CyU3PQueueCreate(CyU3PQueue *queue_p, uint32_t messageSize, void *queueStart, uint32_t queueSize) . . . . .	380
5.27.12.57CyU3PQueueDestroy(CyU3PQueue *queue_p) . . . . .	380
5.27.12.58CyU3PQueueFlush(CyU3PQueue *queue_p) . . . . .	381
5.27.12.59CyU3PQueuePerfGet(CyU3PQueue *queue_p, uint32_t *sentCnt_p, uint32_t *recvCnt_p, uint32_t *emptyCnt_p, uint32_t *fullCnt_p, uint32_t *fullErrCnt_p, uint32_t *timeoutCnt_p) . . . . .	381
5.27.12.60CyU3PQueuePrioritySend(CyU3PQueue *queue_p, void *src_p, uint32_t waitOption) . . . . .	382
5.27.12.61CyU3PQueueReceive(CyU3PQueue *queue_p, void *dest_p, uint32_t waitOption) . . . . .	382
5.27.12.62CyU3PQueueSend(CyU3PQueue *queue_p, void *src_p, uint32_t waitOption) . . . . .	383
5.27.12.63CyU3PQueueSystemPerfGet(uint32_t *sentCnt_p, uint32_t *recvCnt_p, uint32_t *emptyCnt_p, uint32_t *fullCnt_p, uint32_t *fullErrCnt_p, uint32_t *timeoutCnt_p) . . . . .	383
5.27.12.64CyU3PSemaphoreCreate(CyU3PSemaphore *semaphore_p, uint32_t initialCount) . . . . .	383
5.27.12.65CyU3PSemaphoreDestroy(CyU3PSemaphore *semaphore_p) . . . . .	384
5.27.12.66CyU3PSemaphoreGet(CyU3PSemaphore *semaphore_p, uint32_t waitOption) . . . . .	384
5.27.12.67CyU3PSemaphorePut(CyU3PSemaphore *semaphore_p) . . . . .	385
5.27.12.68CyU3PSemaphoreSetActivityGpio(CyU3PSemaphore *semaphore_p, uint32_t gpio_id) . . . . .	385
5.27.12.69CyU3PSetTime(uint32_t newTime) . . . . .	386

5.27.12.70CyU3PThreadCreate(CyU3PThread *thread_p, char *threadName, CyU3PThreadEntry_t entryFn, uint32_t entryInput, void *stackStart, uint32_t stackSize, uint32_t priority, uint32_t preemptThreshold, uint32_t timeSlice, uint32_t autoStart)	386
5.27.12.71CyU3PThreadDestroy(CyU3PThread *thread_ptr)	387
5.27.12.72CyU3PThreadIdentify(void)	387
5.27.12.73CyU3PThreadInfoGet(CyU3PThread *thread_p, uint8_t **name_p, uint32_t *priority, uint32_t *preemptionThreshold, uint32_t *timeSlice)	387
5.27.12.74CyU3PThreadPerfGet(CyU3PThread *thread_p, uint32_t *resumeCnt_p, uint32_t *suspendCnd_p, uint32_t *reqPreemptionCnt_p, uint32_t *intrPreemptionCnt_p, uint32_t *priInvertCnt_p, uint32_t *relinquishCnt_p, uint32_t *timeoutCnt_p)	388
5.27.12.75CyU3PThreadPriorityChange(CyU3PThread *thread_p, uint32_t newPriority, uint32_t *oldPriority)	388
5.27.12.76CyU3PThreadRelinquish(void)	389
5.27.12.77CyU3PThreadResume(CyU3PThread *thread_p)	389
5.27.12.78CyU3PThreadSetActivityGpio(CyU3PThread *thread_p, uint32_t gpio_id)	390
5.27.12.79CyU3PThreadSleep(uint32_t timerTicks)	390
5.27.12.80CyU3PThreadSuspend(CyU3PThread *thread_p)	390
5.27.12.81CyU3PThreadSystemPerfGet(uint32_t *resumeCnt_p, uint32_t *suspendCnd_p, uint32_t *reqPreemptionCnt_p, uint32_t *intrPreemptionCnt_p, uint32_t *prioInvertCnt_p, uint32_t *relinquishCnt_p, uint32_t *timeoutCnt_p, uint32_t *busyReturnCnt_p, uint32_t *idleReturnCnt_p)	391
5.27.12.82CyU3PThreadWaitAbort(CyU3PThread *thread_p)	391
5.27.12.83CyU3PTimerCreate(CyU3PTimer *timer_p, CyU3PTimerCb_t expirationFunction, uint32_t expirationInput, uint32_t initialTicks, uint32_t rescheduleTicks, uint32_t timerOption)	392
5.27.12.84CyU3PTimerDestroy(CyU3PTimer *timer_p)	392
5.27.12.85CyU3PTimerModify(CyU3PTimer *timer_p, uint32_t initialTicks, uint32_t rescheduleTicks)	393
5.27.12.86CyU3PTimerPerfGet(CyU3PTimer *timer_p, uint32_t *activateCnt_p, uint32_t *reactivateCnt_p, uint32_t *deactivateCnt_p, uint32_t *expiryCnt_p)	393
5.27.12.87CyU3PTimerStart(CyU3PTimer *timer_p)	394
5.27.12.88CyU3PTimerStop(CyU3PTimer *timer_p)	394
5.27.12.89CyU3PTimerSystemPerfGet(uint32_t *activateCnt_p, uint32_t *reactivateCnt_p, uint32_t *deactivateCnt_p, uint32_t *expiryCnt_p)	394
5.27.12.90CyU3PUndefinedHandler(void)	395
<b>5.28 firmware/u3p_firmware/inc/cyu3pib.h File Reference</b>	395
<b>5.28.1 Detailed Description</b>	398
<b>5.28.2 Macro Definition Documentation</b>	398
<b>5.28.2.1 CYU3P_GET_GPIF_ERROR_TYPE</b>	398
<b>5.28.2.2 CYU3P_GET_PIB_ERROR_TYPE</b>	398
<b>5.28.3 Typedef Documentation</b>	398
<b>5.28.3.1 CyU3PGpifErrorType</b>	398
<b>5.28.3.2 CyU3PPibClock_t</b>	399

5.28.3.3	CyU3PPibClockPhase_t . . . . .	399
5.28.3.4	CyU3PPibDllMode_t . . . . .	399
5.28.3.5	CyU3PPibErrorType . . . . .	400
5.28.3.6	CyU3PPibIntrCb_t . . . . .	400
5.28.3.7	CyU3PPibIntrType . . . . .	400
5.28.3.8	CyU3PPmmcEventType . . . . .	400
5.28.3.9	CyU3PPmmcIntrCb_t . . . . .	401
5.28.3.10	CyU3PPmmcState . . . . .	401
5.28.4	Enumeration Type Documentation . . . . .	401
5.28.4.1	CyU3PGpifErrorType . . . . .	401
5.28.4.2	CyU3PPibClockPhase_t . . . . .	402
5.28.4.3	CyU3PPibDllMode_t . . . . .	402
5.28.4.4	CyU3PPibErrorType . . . . .	403
5.28.4.5	CyU3PPibIntrType . . . . .	404
5.28.4.6	CyU3PPmmcEventType . . . . .	405
5.28.4.7	CyU3PPmmcState . . . . .	406
5.28.5	Function Documentation . . . . .	406
5.28.5.1	CyU3PPibDeInit(void) . . . . .	406
5.28.5.2	CyU3PPibDllConfigure(CyU3PPibDllMode_t dll_mode, uint16_t slave_delay, CyU3PPibClockPhase_t core_phase, CyU3PPibClockPhase_t sync_phase, CyU3PPibClockPhase_t output_phase, CyBool_t should_lock) . . . . .	406
5.28.5.3	CyU3PPibInit(CyBool_t doInit, CyU3PPibClock_t *pibClock) . . . . .	407
5.28.5.4	CyU3PPibRegisterCallback(CyU3PPibIntrCb_t cbFunc, uint32_t intMask) . . . . .	408
5.28.5.5	CyU3PPibSelectIntSources(CyBool_t pibSockEn, CyBool_t gpifIntEn, CyBool_t pibErrEn, CyBool_t mboxIntEn, CyBool_t wakeupEn) . . . . .	408
5.28.5.6	CyU3PPibSelectMmcSlaveMode(void) . . . . .	408
5.28.5.7	CyU3PPibSetInterruptPriority(CyBool_t isHigh) . . . . .	409
5.28.5.8	CyU3PPibSetPmmcBusyMode(uint32_t waitBuffer, uint32_t waitCount) . . . . .	409
5.28.5.9	CyU3PPibSetSocketEop(uint32_t eopMask) . . . . .	409
5.28.5.10	CyU3PPmmcEnableDirectAccess(CyBool_t enable, uint8_t wrSock, uint8_t rdSock) . . . . .	410
5.28.5.11	CyU3PPmmcRegisterCallback(CyU3PPmmcIntrCb_t cbFunc) . . . . .	410
5.28.5.12	CyU3PSetPportDriveStrength(CyU3PDriveStrengthState_t pportDriveStrength) . . . . .	411
5.29	firmware/u3p_firmware/inc/cyu3sib.h File Reference . . . . .	411
5.29.1	Detailed Description . . . . .	415
5.29.2	Typedef Documentation . . . . .	415
5.29.2.1	CyU3PSdioCardRegs_t . . . . .	415
5.29.2.2	CyU3PSibCardDetect . . . . .	415
5.29.2.3	CyU3PSibDevInfo_t . . . . .	415
5.29.2.4	CyU3PSibDevRegType . . . . .	416
5.29.2.5	CyU3PSibDevType . . . . .	416
5.29.2.6	CyU3PSibEraseMode . . . . .	416

5.29.2.7	CyU3PSibEventType . . . . .	416
5.29.2.8	CyU3PSibEvtCbk_t . . . . .	416
5.29.2.9	CyU3PSibIntfParams_t . . . . .	416
5.29.2.10	CyU3PSibIntfVoltage . . . . .	417
5.29.2.11	CyU3PSibLunInfo_t . . . . .	417
5.29.2.12	CyU3PSibLunLocation . . . . .	417
5.29.2.13	CyU3PSibLunType . . . . .	417
5.29.2.14	CyU3PSibOpFreq . . . . .	417
5.29.2.15	CyU3PSibPortId . . . . .	417
5.29.3	Enumeration Type Documentation . . . . .	418
5.29.3.1	CyU3PSibCardDetect . . . . .	418
5.29.3.2	CyU3PSibDevRegType . . . . .	418
5.29.3.3	CyU3PSibDevType . . . . .	418
5.29.3.4	CyU3PSibEraseMode . . . . .	418
5.29.3.5	CyU3PSibEventType . . . . .	419
5.29.3.6	CyU3PSibIntfVoltage . . . . .	419
5.29.3.7	CyU3PSibLunLocation . . . . .	419
5.29.3.8	CyU3PSibLunType . . . . .	420
5.29.3.9	CyU3PSibOpFreq . . . . .	420
5.29.3.10	CyU3PSibPortId . . . . .	420
5.29.4	Function Documentation . . . . .	420
5.29.4.1	CyU3PSdioAbortFunctionIO(uint8_t portId, uint8_t functionNumber) . . . . .	420
5.29.4.2	CyU3PSdioByteReadWrite(uint8_t portId, uint8_t functionNumber, uint8_t is←Write, uint8_t readAfterWriteFlag, uint32_t registerAddr, uint8_t *data) . . . . .	421
5.29.4.3	CyU3PSdioCardReset(uint8_t portId) . . . . .	421
5.29.4.4	CyU3PSdioDirectReadWrite(uint8_t portId, uint8_t functionNumber, uint8_t is←Write, uint8_t opCode, uint8_t *data_p, uint32_t registerAddr, uint16_t transfer←Count) . . . . .	422
5.29.4.5	CyU3PSdioExtendedReadWrite(uint8_t portId, uint8_t functionNumber, uint8_t isWrite, uint8_t blockMode, uint8_t opCode, uint32_t registerAddr, uint16_t transferCount, uint16_t sckId) . . . . .	422
5.29.4.6	CyU3PSdioGetBlockSize(uint8_t portId, uint8_t funcNo, uint16_t *blockSize) . . . . .	423
5.29.4.7	CyU3PSdioGetCISAddress(uint8_t portId, uint8_t functionNumber, uint32_t *addressCIS) . . . . .	424
5.29.4.8	CyU3PSdioGetTuples(uint8_t portId, uint8_t funcNo, uint8_t tupleId, uint32_t addrCIS, uint8_t *buffer, uint8_t *tupleSize) . . . . .	424
5.29.4.9	CyU3PSdioInterruptControl(uint8_t portId, uint8_t functionNumber, uint8_t operation, uint8_t *intEnReg) . . . . .	425
5.29.4.10	CyU3PSdioQueryCard(uint8_t portId, CyU3PSdioCardRegs_t *data) . . . . .	425
5.29.4.11	CyU3PSdioReadWaitEnable(uint8_t portId, uint8_t isRdWtEnable) . . . . .	426
5.29.4.12	CyU3PSdioSetBlockSize(uint8_t portId, uint8_t funcNo, uint16_t blockSize) . . . . .	426
5.29.4.13	CyU3PSdioSuspendResumeFunction(uint8_t portId, uint8_t functionNumber, uint8_t operation, uint8_t *isDataAvailable) . . . . .	427

5.29.4.14 CyU3PSibAbortRequest(uint8_t portId) . . . . .	427
5.29.4.15 CyU3PSibCommitReadWrite(uint8_t portId) . . . . .	427
5.29.4.16 CyU3PSibDeInit(uint8_t portId) . . . . .	428
5.29.4.17 CyU3PSibEraseBlocks(uint8_t portId, uint16_t numEraseUnits, uint32_t startAddr, CyU3PSibEraseMode mode) . . . . .	428
5.29.4.18 CyU3PSibForceErase(uint8_t portId) . . . . .	429
5.29.4.19 CyU3PSibGetCardStatus(uint8_t portId, uint32_t *status_p) . . . . .	429
5.29.4.20 CyU3PSibGetCSD(uint8_t portId, uint8_t *csd_buffer) . . . . .	430
5.29.4.21 CyU3PSibGetMMCExtCsd(uint8_t portId, uint8_t *buffer_p) . . . . .	430
5.29.4.22 CyU3PSibInit(uint8_t portId) . . . . .	430
5.29.4.23 CyU3PSibLockUnlockCard(uint8_t portId, CyBool_t lockCard, CyBool_t clrPasswd, uint8_t *passwd, uint8_t passwdLen) . . . . .	431
5.29.4.24 CyU3PSibPartitionStorage(uint8_t portId, uint8_t partCount, uint32_t *partSizes_p, uint8_t *partType_p) . . . . .	431
5.29.4.25 CyU3PSibQueryDevice(uint8_t portId, CyU3PSibDevInfo_t *devInfo_p) . . . . .	432
5.29.4.26 CyU3PSibQueryUnit(uint8_t portId, uint8_t unitId, CyU3PSibLunInfo_t *unitInfo_p) . . . . .	432
5.29.4.27 CyU3PSibReadRegister(uint8_t portId, CyU3PSibDevRegType regType, uint8_t *dataBuffer, uint8_t *dataLen) . . . . .	433
5.29.4.28 CyU3PSibReadWriteRequest(CyBool_t isRead, uint8_t portId, uint8_t unitId, uint16_t numBlks, uint32_t blkAddr, uint8_t socket) . . . . .	433
5.29.4.29 CyU3PSibRegisterCbk(CyU3PSibEvtCbk_t sibEvtCbk) . . . . .	434
5.29.4.30 CyU3PSibRemovePartitions(uint8_t portId) . . . . .	434
5.29.4.31 CyU3PSibRemovePasswd(uint8_t portId, uint8_t *passwd, uint8_t passwdLen) . . . . .	435
5.29.4.32 CyU3PSibSendSwitchCommand(uint8_t portId, uint32_t argument, uint32_t *resp_p) . . . . .	435
5.29.4.33 CyU3PSibSetBlockLen(uint8_t portId, uint16_t blkLen) . . . . .	435
5.29.4.34 CyU3PSibSetIntfParams(uint8_t portId, CyU3PSibIntfParams_t *intfParams) . . . . .	436
5.29.4.35 CyU3PSibSetPasswd(uint8_t portId, CyBool_t lockCard, uint8_t *passwd, uint8_t passwdLen, uint8_t *newPasswd, uint8_t newPasswdLen) . . . . .	436
5.29.4.36 CyU3PSibSetWriteCommitSize(uint8_t portId, uint32_t numSectors) . . . . .	437
5.29.4.37 CyU3PSibStart(void) . . . . .	437
5.29.4.38 CyU3PSibStop(void) . . . . .	437
5.29.4.39 CyU3PSibVendorAccess(uint8_t portId, uint8_t cmd, uint32_t cmdArg, uint8_t respLen, uint8_t *respData, uint16_t numBlks, CyBool_t isRead, uint8_t socket) . . . . .	438
5.30 firmware/u3p_firmware/inc/cyu3sibpp.h File Reference . . . . .	439
5.30.1 Detailed Description . . . . .	440
5.30.2 Macro Definition Documentation . . . . .	440
5.30.2.1 CY_U3P_BOOT_PARTITION_NUM_BLKS . . . . .	440
5.30.2.2 CY_U3P_PARTITION_TYPE_AP_BOOT . . . . .	440
5.30.2.3 CY_U3P_PARTITION_TYPE_BENICIA_BOOT . . . . .	440
5.30.2.4 CY_U3P_PARTITION_TYPE_DATA_AREA . . . . .	441

5.30.2.5 CY_U3P_SIB_EVT_ABORT . . . . .	441
5.30.2.6 CY_U3P_SIB_EVT_DRVENTRY . . . . .	441
5.30.2.7 CY_U3P_SIB_EVT_PORT_0 . . . . .	441
5.30.2.8 CY_U3P_SIB_EVT_PORT_1 . . . . .	441
5.30.2.9 CY_U3P_SIB_EVT_PORT_POS . . . . .	441
5.30.2.10 CY_U3P_SIB_STACK_SIZE . . . . .	441
5.30.2.11 CY_U3P_SIB_THREAD_PRIORITY . . . . .	441
5.30.2.12 CY_U3P_SIB_TIMER0_EVT . . . . .	441
5.30.2.13 CY_U3P_SIB_TIMER1_EVT . . . . .	441
5.30.2.14 CYU3P_SIB_INT_READ_SOCKET . . . . .	441
5.30.2.15 CYU3P_SIB_INT_WRITE_SOCKET . . . . .	441
5.30.2.16 CyU3PSibDisableCoreIntr . . . . .	442
5.30.2.17 CyU3PSibEnableCoreIntr . . . . .	442
5.30.3 Typedef Documentation . . . . .	442
5.30.3.1 CyU3PSibGlobalData . . . . .	442
5.30.4 Enumeration Type Documentation . . . . .	442
5.30.4.1 CyU3PSibDevPartition . . . . .	442
5.30.4.2 CyU3PSibSocketId_t . . . . .	443
5.30.5 Function Documentation . . . . .	443
5.30.5.1 CyU3PSibInitCard(uint8_t portId) . . . . .	443
5.30.5.2 CyU3PSibUpdateLunInfo(uint8_t portId, uint8_t partNum, uint8_t partType, uint8_t location, uint32_t startAddr, uint32_t partSize) . . . . .	443
5.30.6 Variable Documentation . . . . .	443
5.30.6.1 glSibCtxt . . . . .	443
5.30.6.2 glSibDevInfo . . . . .	444
5.30.6.3 glSibEvent . . . . .	444
5.30.6.4 glSibEvtCbk . . . . .	444
5.30.6.5 glSibIntfParams . . . . .	444
5.30.6.6 glSibLunInfo . . . . .	444
5.30.6.7 glSibThread . . . . .	444
5.31 firmware/u3p_firmware/inc/cyu3socket.h File Reference . . . . .	444
5.31.1 Detailed Description . . . . .	446
5.31.2 Macro Definition Documentation . . . . .	446
5.31.2.1 CyU3PDmaGetSckId . . . . .	446
5.31.3 Typedef Documentation . . . . .	446
5.31.3.1 CyU3PBlockId_t . . . . .	446
5.31.3.2 CyU3PDmaSocket_t . . . . .	446
5.31.3.3 CyU3PDmaSocketCallback_t . . . . .	447
5.31.3.4 CyU3PDmaSocketConfig_t . . . . .	447
5.31.4 Enumeration Type Documentation . . . . .	447

5.31.4.1	CyU3PBlockId_t . . . . .	447
5.31.5	Function Documentation . . . . .	447
5.31.5.1	CyU3PDmaSocketClearInterrupts(uint16_t sckId, uint32_t intVal) . . . . .	447
5.31.5.2	CyU3PDmaSocketDisable(uint16_t sckId) . . . . .	448
5.31.5.3	CyU3PDmaSocketEnable(uint16_t sckId) . . . . .	448
5.31.5.4	CyU3PDmaSocketGetConfig(uint16_t sckId, CyU3PDmaSocketConfig_t *sck_p) . . . . .	449
5.31.5.5	CyU3PDmaSocketIsValid(uint16_t sckId) . . . . .	449
5.31.5.6	CyU3PDmaSocketIsValidConsumer(uint16_t sckId) . . . . .	449
5.31.5.7	CyU3PDmaSocketIsValidProducer(uint16_t sckId) . . . . .	450
5.31.5.8	CyU3PDmaSocketRegisterCallback(CyU3PDmaSocketCallback_t cb) . . . . .	450
5.31.5.9	CyU3PDmaSocketSendEvent(uint16_t sckId, uint16_t dscrIndex, CyBool_t isOccupied) . . . . .	450
5.31.5.10	CyU3PDmaSocketSetConfig(uint16_t sckId, CyU3PDmaSocketConfig_t *sck_p) . . . . .	451
5.31.5.11	CyU3PDmaSocketSetWrapUp(uint16_t sckId) . . . . .	451
5.31.5.12	CyU3PDmaUpdateSocketResume(uint16_t sckId, uint16_t suspendOption) . . . . .	452
5.31.5.13	CyU3PDmaUpdateSocketSuspendOption(uint16_t sckId, uint16_t suspendOption) . . . . .	452
5.32	firmware/u3p_firmware/inc/cyu3spi.h File Reference . . . . .	453
5.32.1	Detailed Description . . . . .	454
5.32.2	Typedef Documentation . . . . .	454
5.32.2.1	CyU3PSpiConfig_t . . . . .	454
5.32.2.2	CyU3PSpiError_t . . . . .	455
5.32.2.3	CyU3PSpiEvt_t . . . . .	455
5.32.2.4	CyU3PSpiIntrCb_t . . . . .	455
5.32.2.5	CyU3PSpiSsnCtrl_t . . . . .	455
5.32.2.6	CyU3PSpiSsnLagLead_t . . . . .	456
5.32.3	Enumeration Type Documentation . . . . .	456
5.32.3.1	CyU3PSpiError_t . . . . .	456
5.32.3.2	CyU3PSpiEvt_t . . . . .	456
5.32.3.3	CyU3PSpiSsnCtrl_t . . . . .	457
5.32.3.4	CyU3PSpiSsnLagLead_t . . . . .	457
5.32.4	Function Documentation . . . . .	458
5.32.4.1	CyU3PRegisterSpiCallBack(CyU3PSpiIntrCb_t spiIntrCb) . . . . .	458
5.32.4.2	CyU3PSpiDeInit(void) . . . . .	458
5.32.4.3	CyU3PSpiDisableBlockXfer(CyBool_t rxDisable, CyBool_t txDisable) . . . . .	458
5.32.4.4	CyU3PSpiInit(void) . . . . .	459
5.32.4.5	CyU3PSpiReceiveWords(uint8_t *data, uint32_t byteCount) . . . . .	459
5.32.4.6	CyU3PSpiSetBlockXfer(uint32_t txSize, uint32_t rxSize) . . . . .	460
5.32.4.7	CyU3PSpiSetConfig(CyU3PSpiConfig_t *config, CyU3PSpiIntrCb_t cb) . . . . .	460
5.32.4.8	CyU3PSpiSetSsnLine(CyBool_t isHigh) . . . . .	461
5.32.4.9	CyU3PSpiSetTimeout(uint32_t readLoopCnt, uint32_t writeLoopCnt) . . . . .	461

5.32.4.10 CyU3PSpiTransferWords(uint8_t *txBuf, uint32_t txByteCount, uint8_t *rxBuf, uint32_t rxByteCount) . . . . .	462
5.32.4.11 CyU3PSpiTransmitWords(uint8_t *data, uint32_t byteCount) . . . . .	462
5.32.4.12 CyU3PSpiWaitForBlockXfer(CyBool_t isRead) . . . . .	463
5.33 firmware/u3p_firmware/inc/cyu3system.h File Reference . . . . .	463
5.33.1 Detailed Description . . . . .	467
5.33.2 Typedef Documentation . . . . .	467
5.33.2.1 CyU3PDebugLog_t . . . . .	467
5.33.2.2 CyU3PDriveStrengthState_t . . . . .	467
5.33.2.3 CyU3PLppModule_t . . . . .	467
5.33.2.4 CyU3PSysClockConfig_t . . . . .	468
5.33.2.5 CyU3PSysClockSrc_t . . . . .	468
5.33.2.6 CyU3PSysThreadId_t . . . . .	468
5.33.3 Enumeration Type Documentation . . . . .	469
5.33.3.1 CyU3PDriveStrengthState_t . . . . .	469
5.33.3.2 CyU3PLppModule_t . . . . .	469
5.33.3.3 CyU3PSysClockSrc_t . . . . .	470
5.33.3.4 CyU3PSysThreadId_t . . . . .	470
5.33.4 Function Documentation . . . . .	471
5.33.4.1 CyFxApplicationDefine(void) . . . . .	471
5.33.4.2 CyU3PDebugDelInit(void) . . . . .	471
5.33.4.3 CyU3PDebugDisable(void) . . . . .	471
5.33.4.4 CyU3PDebugEnable(uint16_t threadMask) . . . . .	471
5.33.4.5 CyU3PDebugInit(CyU3PDmaSocketId_t destSckId, uint8_t traceLevel) . . . . .	472
5.33.4.6 CyU3PDebugLog(uint8_t priority, uint16_t message, uint32_t parameter) . . . . .	472
5.33.4.7 CyU3PDebugLogClear(void) . . . . .	473
5.33.4.8 CyU3PDebugLogFlush(void) . . . . .	473
5.33.4.9 CyU3PDebugPreamble(CyBool_t sendPreamble) . . . . .	473
5.33.4.10 CyU3PDebugPrint(uint8_t priority, char *message,...) . . . . .	474
5.33.4.11 CyU3PDebugSetTimeout(uint32_t timeout) . . . . .	474
5.33.4.12 CyU3PDebugSetTraceLevel(uint8_t level) . . . . .	475
5.33.4.13 CyU3PDebugStringPrint(uint8_t *buffer, uint16_t maxLength, char *fmt,...) . . . . .	475
5.33.4.14 CyU3PDebugSysMemDeInit(void) . . . . .	476
5.33.4.15 CyU3PDebugSysMemInit(uint8_t *buffer, uint16_t size, CyBool_t isWrapAround, uint8_t traceLevel) . . . . .	476
5.33.4.16 CyU3PDeviceCacheControl(CyBool_t isICacheEnable, CyBool_t isDCacheEnable, CyBool_t isDmaHandleDCache) . . . . .	476
5.33.4.17 CyU3PDeviceConfigureIOMatrix(CyU3PloMatrixConfig_t *cfg_p) . . . . .	477
5.33.4.18 CyU3PDeviceGetCpuLoad(void) . . . . .	478
5.33.4.19 CyU3PDeviceGetDriverLoad(void) . . . . .	478
5.33.4.20 CyU3PDeviceGetPartNumber(void) . . . . .	478

5.33.4.21 CyU3PDeviceGetSysClkFreq(uint32_t *freq) . . . . .	479
5.33.4.22 CyU3PDeviceGetThreadLoad(CyU3PThread *thread_p) . . . . .	479
5.33.4.23 CyU3PDeviceGpioOverride(uint8_t gpioid, CyBool_t isSimple) . . . . .	479
5.33.4.24 CyU3PDeviceGpioRestore(uint8_t gpioid) . . . . .	480
5.33.4.25 CyU3PDeviceInit(CyU3PSysClockConfig_t *clkCfg) . . . . .	480
5.33.4.26 CyU3PDeviceReset(CyBool_t isWarmReset) . . . . .	480
5.33.4.27 CyU3PFirmwareEntry(void) . . . . .	481
5.33.4.28 CyU3PIsGpioComplexIOConfigured(uint32_t gpioid) . . . . .	481
5.33.4.29 CyU3PIsGpioSimpleIOConfigured(uint32_t gpioid) . . . . .	482
5.33.4.30 CyU3PIsGpioValid(uint8_t gpioid) . . . . .	482
5.33.4.31 CyU3PIsLppIOConfigured(CyU3PLppModule_t lppModule) . . . . .	482
5.33.4.32 CyU3PJumpToAddress(uint32_t address) . . . . .	483
5.33.4.33 CyU3PSetSerialIoDriveStrength(CyU3PDriveStrengthState_t driveStrength) . . . . .	483
5.33.4.34 CyU3PSysCheckStandbyParam(uint16_t wakeupFlags, uint16_t polarity, uint8_t *bkp_buff_p) . . . . .	483
5.33.4.35 CyU3PSysCheckSuspendParams(uint16_t wakeupFlags, uint16_t polarity) . . . . .	484
5.33.4.36 CyU3PSysEnterStandbyMode(uint16_t wakeupFlags, uint16_t polarity, uint8_t *bkp_buff_p) . . . . .	484
5.33.4.37 CyU3PSysEnterSuspendMode(uint16_t wakeupFlags, uint16_t polarity, uint16_t *wakeupSource) . . . . .	485
5.33.4.38 CyU3PSysGetApiVersion(uint16_t *majorVersion, uint16_t *minorVersion, uint16_t *patchNumer, uint16_t *buildNumer) . . . . .	485
5.33.4.39 CyU3PSysSendEnterSuspendStatus(void) . . . . .	486
5.33.4.40 CyU3PSystemRegisterDriver(CyU3PThread *thread_p) . . . . .	486
5.33.4.41 CyU3PSysWatchDogClear(void) . . . . .	486
5.33.4.42 CyU3PSysWatchDogConfigure(CyBool_t enable, uint32_t period) . . . . .	486
5.33.4.43 CyU3PToolChainInit(void) . . . . .	487
5.34 firmware/u3p_firmware/inc/cyu3types.h File Reference . . . . .	487
5.34.1 Detailed Description . . . . .	488
5.34.2 Macro Definition Documentation . . . . .	488
5.34.2.1 CyFalse . . . . .	488
5.34.2.2 CyTrue . . . . .	488
5.34.3 Typedef Documentation . . . . .	488
5.34.3.1 CyBool_t . . . . .	488
5.34.3.2 CyU3PReturnStatus_t . . . . .	488
5.34.3.3 uvint16_t . . . . .	488
5.34.3.4 uvint32_t . . . . .	488
5.34.3.5 uvint8_t . . . . .	489
5.35 firmware/u3p_firmware/inc/cyu3uart.h File Reference . . . . .	489
5.35.1 Detailed Description . . . . .	490
5.35.2 Typedef Documentation . . . . .	491

5.35.2.1	CyU3PUartBaudrate_t . . . . .	491
5.35.2.2	CyU3PUartConfig_t . . . . .	491
5.35.2.3	CyU3PUartError_t . . . . .	491
5.35.2.4	CyU3PUartEvt_t . . . . .	491
5.35.2.5	CyU3PUartIntrCb_t . . . . .	492
5.35.2.6	CyU3PUartParity_t . . . . .	492
5.35.2.7	CyU3PUartStopBit_t . . . . .	492
5.35.3	Enumeration Type Documentation . . . . .	492
5.35.3.1	CyU3PUartBaudrate_t . . . . .	492
5.35.3.2	CyU3PUartError_t . . . . .	494
5.35.3.3	CyU3PUartEvt_t . . . . .	494
5.35.3.4	CyU3PUartParity_t . . . . .	494
5.35.3.5	CyU3PUartStopBit_t . . . . .	495
5.35.4	Function Documentation . . . . .	495
5.35.4.1	CyU3PRegisterUartCallBack(CyU3PUartIntrCb_t uartIntrCb) . . . . .	495
5.35.4.2	CyU3PUartDelInit(void) . . . . .	495
5.35.4.3	CyU3PUartInit(void) . . . . .	495
5.35.4.4	CyU3PUartReceiveBytes(uint8_t *data_p, uint32_t count, CyU3PReturnStatus_t *status) . . . . .	496
5.35.4.5	CyU3PUartRxSetBlockXfer(uint32_t rxSize) . . . . .	496
5.35.4.6	CyU3PUartSetConfig(CyU3PUartConfig_t *config, CyU3PUartIntrCb_t cb) . . . . .	497
5.35.4.7	CyU3PUartSetTimeout(uint32_t readLoopCnt, uint32_t writeLoopCnt) . . . . .	497
5.35.4.8	CyU3PUartTransmitBytes(uint8_t *data_p, uint32_t count, CyU3PReturnStatus_t *status) . . . . .	498
5.35.4.9	CyU3PUartTxSetBlockXfer(uint32_t txSize) . . . . .	498
5.36	firmware/u3p_firmware/inc/cyu3usb.h File Reference . . . . .	499
5.36.1	Detailed Description . . . . .	504
5.36.2	Enumeration Modes . . . . .	504
5.36.3	USB Device Mode Functions . . . . .	504
5.36.4	Macro Definition Documentation . . . . .	504
5.36.4.1	CY_U3P_MAX_STRING_DESC_INDEX . . . . .	505
5.36.5	Typedef Documentation . . . . .	505
5.36.5.1	CyU3PEpConfig_t . . . . .	505
5.36.5.2	CyU3PUsbDevProperty . . . . .	505
5.36.5.3	CyU3PUsbEpEvtCb_t . . . . .	505
5.36.5.4	CyU3PUsbEpEvtType . . . . .	505
5.36.5.5	CyU3PUSBEventCb_t . . . . .	506
5.36.5.6	CyU3PUsbEventType_t . . . . .	506
5.36.5.7	CyU3PUsbLinkPowerMode . . . . .	506
5.36.5.8	CyU3PUsbLPMReqCb_t . . . . .	506

5.36.5.9 CyU3PUsbMgrStates_t . . . . .	507
5.36.5.10 CyU3PUSBSetDescType_t . . . . .	507
5.36.5.11 CyU3PUSBSetupCb_t . . . . .	507
5.36.6 Enumeration Type Documentation . . . . .	508
5.36.6.1 CyU3PUsbDevProperty . . . . .	508
5.36.6.2 CyU3PUsbEpEvtType . . . . .	508
5.36.6.3 CyU3PUsbEventType_t . . . . .	509
5.36.6.4 CyU3PUsbLinkPowerMode . . . . .	510
5.36.6.5 CyU3PUsbMgrStates_t . . . . .	511
5.36.6.6 CyU3PUSBSetDescType_t . . . . .	511
5.36.6.7 CyU3PUSBSpeed_t . . . . .	512
5.36.7 Function Documentation . . . . .	512
5.36.7.1 CyU3PConnectState(CyBool_t connect, CyBool_t ssEnable) . . . . .	512
5.36.7.2 CyU3PGetConnectState(void) . . . . .	512
5.36.7.3 CyU3PSetEpConfig(uint8_t ep, CyU3PEpConfig_t *epinfo) . . . . .	513
5.36.7.4 CyU3PSetEpPacketSize(uint8_t ep, uint16_t maxPktSize) . . . . .	513
5.36.7.5 CyU3PUibCheckConnection(void) . . . . .	514
5.36.7.6 CyU3PUsb2Resume(void) . . . . .	514
5.36.7.7 CyU3PUsbAckSetup(void) . . . . .	514
5.36.7.8 CyU3PUsbChangeMapping(uint8_t ep, uint8_t socketNum, CyBool_t remap, uint16_t newstreamId, uint8_t newep) . . . . .	515
5.36.7.9 CyU3PUsbControlUsb2Support(CyBool_t enable) . . . . .	515
5.36.7.10 CyU3PUsbControlVBusDetect(CyBool_t enable, CyBool_t useVbatt) . . . . .	516
5.36.7.11 CyU3PUsbDoRemoteWakeup(void) . . . . .	516
5.36.7.12 CyU3PUsbEnableEPPPrefetch(void) . . . . .	516
5.36.7.13 CyU3PUsbEnableITPEvent55 Td [(uint16_t)-278(ne)20(wstreamId,)-278(u44(5.36.6.6)-940(CyU3PUSB8	

5.36.7.27 CyU3PUsbGetLinkPowerState(CyU3PUsbLinkPowerMode *mode_p) . . . . .	523
5.36.7.28 CyU3PUsbGetSpeed(void) . . . . .	524
5.36.7.29 CyU3PUsbInitEventLog(uint8_t *buffer, uint32_t bufSize) . . . . .	524
5.36.7.30 CyU3PUsbIsStarted(void) . . . . .	524
5.36.7.31 CyU3PUsbJumpBackToBooter(uint32_t address) . . . . .	525
5.36.7.32 CyU3PUsbLinkComplianceControl(CyBool_t isEnabled) . . . . .	525
5.36.7.33 CyU3PUsbLPMDisable(void) . . . . .	526
5.36.7.34 CyU3PUsbLPMEnable(void) . . . . .	526
5.36.7.35 CyU3PUsbMapStream(uint8_t ep, uint8_t socketNum, uint16_t streamId) . . . . .	526
5.36.7.36 CyU3PUsbRegisterEpEvtCallback(CyU3PUsbEpEvtCb_t cbFunc, uint32_t eventMask, uint16_t outEpMask, uint16_t inEpMask) . . . . .	527
5.36.7.37 CyU3PUsbRegisterEventCallback(CyU3PUSBEventCb_t callback) . . . . .	527
5.36.7.38 CyU3PUsbRegisterLPMRequestCallback(CyU3PUsbLPMReqCb_t cb) . . . . .	528
5.36.7.39 CyU3PUsbRegisterSetupCallback(CyU3PUSBSetupCb_t callback, CyBool_t fastEnum) . . . . .	528
5.36.7.40 CyU3PUsbResetEndpointMemories(void) . . . . .	529
5.36.7.41 CyU3PUsbResetEp(uint8_t ep) . . . . .	529
5.36.7.42 CyU3PUsbSendAckTP(uint8_t ep, uint8_t nump, uint16_t bulkStream) . . . . .	529
5.36.7.43 CyU3PUsbSendDevNotification(uint8_t notificationType, uint32_t param0, uint32_t param1) . . . . .	530
5.36.7.44 CyU3PUsbSendEP0Data(uint16_t count, uint8_t *buffer) . . . . .	530
5.36.7.45 CyU3PUsbSendErdy(uint8_t ep, uint16_t bulkStream) . . . . .	531
5.36.7.46 CyU3PUsbSendNrdy(uint8_t ep, uint16_t bulkStream) . . . . .	531
5.36.7.47 CyU3PUsbSetBooterSwitch(CyBool_t enable) . . . . .	532
5.36.7.48 CyU3PUsbSetDesc(CyU3PUSBSetDescType_t desc_type, uint8_t desc_index, uint8_t *desc) . . . . .	532
5.36.7.49 CyU3PUsbSetEpNak(uint8_t ep, CyBool_t nak) . . . . .	533
5.36.7.50 CyU3PUsbSetEpPktMode(uint8_t ep, CyBool_t pktMode) . . . . .	533
5.36.7.51 CyU3PUsbSetEpSeqNum(uint8_t ep, uint8_t seqnum) . . . . .	533
5.36.7.52 CyU3PUsbSetEpSuspDisableMask(uint16_t noSuspMask) . . . . .	534
5.36.7.53 CyU3PUsbSetLinkPowerState(CyU3PUsbLinkPowerMode link_mode) . . . . .	534
5.36.7.54 CyU3PUsbSetTxDeemphasis(uint32_t value) . . . . .	535
5.36.7.55 CyU3PUsbSetTxSwing(uint32_t swing) . . . . .	535
5.36.7.56 CyU3PUsbSetXfer(uint8_t ep, uint16_t count) . . . . .	535
5.36.7.57 CyU3PUsbSSCDisable(void) . . . . .	536
5.36.7.58 CyU3PUsbStall(uint8_t ep, CyBool_t stall, CyBool_t toggle) . . . . .	536
5.36.7.59 CyU3PUsbStart(void) . . . . .	536
5.36.7.60 CyU3PUsbStop(void) . . . . .	537
5.36.7.61 CyU3PUsbVBattEnable(CyBool_t enable) . . . . .	537
5.37 firmware/u3p_firmware/inc/cyu3usbconst.h File Reference . . . . .	538
5.37.1 Detailed Description . . . . .	540

5.37.2	Typedef Documentation . . . . .	540
5.37.2.1	CyU3PUsb2TestModes . . . . .	540
5.37.2.2	CyU3PUsb3PacketType . . . . .	540
5.37.2.3	CyU3PUsb3TpSubType . . . . .	540
5.37.2.4	CyU3PUsbDescType . . . . .	540
5.37.2.5	CyU3PUsbDevCapType . . . . .	540
5.37.2.6	CyU3PUsbEpType_t . . . . .	540
5.37.2.7	CyU3PUsbFeatureSelector . . . . .	540
5.37.2.8	CyU3PUsbLinkState_t . . . . .	541
5.37.2.9	CyU3PUsbSetupCmds . . . . .	541
5.37.3	Enumeration Type Documentation . . . . .	541
5.37.3.1	CyU3PUsb2TestModes . . . . .	541
5.37.3.2	CyU3PUsb3PacketType . . . . .	541
5.37.3.3	CyU3PUsb3TpSubType . . . . .	542
5.37.3.4	CyU3PUsbDescType . . . . .	542
5.37.3.5	CyU3PUsbDevCapType . . . . .	543
5.37.3.6	CyU3PUsbEpType_t . . . . .	543
5.37.3.7	CyU3PUsbFeatureSelector . . . . .	543
5.37.3.8	CyU3PUsbLinkState_t . . . . .	544
5.37.3.9	CyU3PUsbSetupCmds . . . . .	544
5.38	firmware/u3p_firmware/inc/cyu3usbhost.h File Reference . . . . .	545
5.38.1	Detailed Description . . . . .	547
5.38.2	Typedef Documentation . . . . .	548
5.38.2.1	CyU3PUsbHostConfig_t . . . . .	548
5.38.2.2	CyU3PUsbHostEpConfig_t . . . . .	548
5.38.2.3	CyU3PUsbHostEpStatus_t . . . . .	548
5.38.2.4	CyU3PUsbHostEpXferType_t . . . . .	548
5.38.2.5	CyU3PUsbHostEventCb_t . . . . .	548
5.38.2.6	CyU3PUsbHostEventType_t . . . . .	549
5.38.2.7	CyU3PUsbHostOpSpeed_t . . . . .	549
5.38.2.8	CyU3PUsbHostPortStatus_t . . . . .	549
5.38.2.9	CyU3PUsbHostXferCb_t . . . . .	549
5.38.3	Enumeration Type Documentation . . . . .	549
5.38.3.1	CyU3PUsbHostEpXferType_t . . . . .	549
5.38.3.2	CyU3PUsbHostEventType_t . . . . .	550
5.38.3.3	CyU3PUsbHostOpSpeed_t . . . . .	550
5.38.4	Function Documentation . . . . .	550
5.38.4.1	CyU3PUsbHostEp0BeginXfer(void) . . . . .	550
5.38.4.2	CyU3PUsbHostEpAbort(uint8_t ep) . . . . .	551
5.38.4.3	CyU3PUsbHostEpAdd(uint8_t ep, CyU3PUsbHostEpConfig_t *cfg) . . . . .	551

5.38.4.4 CyU3PUsbHostEpRemove(uint8_t ep) . . . . .	552
5.38.4.5 CyU3PUsbHostEpReset(uint8_t ep) . . . . .	552
5.38.4.6 CyU3PUsbHostEpSetXfer(uint8_t ep, CyU3PUsbHostEpXferType_t type, uint32_t count) . . . . .	553
5.38.4.7 CyU3PUsbHostEpWaitForCompletion(uint8_t ep, CyU3PUsbHostEpStatus_t *epStatus, uint32_t waitOption) . . . . .	553
5.38.4.8 CyU3PUsbHostGetDeviceAddress(uint8_t *devAddr) . . . . .	554
5.38.4.9 CyU3PUsbHostGetFrameNumber(uint32_t *frameNumber) . . . . .	554
5.38.4.10 CyU3PUsbHostGetPortStatus(CyU3PUsbHostPortStatus_t *portStatus, CyU3PUsbHostOpSpeed_t *portSpeed) . . . . .	554
5.38.4.11 CyU3PUsbHostIsStarted(void) . . . . .	555
5.38.4.12 CyU3PUsbHostPortDisable(void) . . . . .	555
5.38.4.13 CyU3PUsbHostPortEnable(void) . . . . .	555
5.38.4.14 CyU3PUsbHostPortReset(void) . . . . .	556
5.38.4.15 CyU3PUsbHostPortResume(void) . . . . .	556
5.38.4.16 CyU3PUsbHostPortSuspend(void) . . . . .	556
5.38.4.17 CyU3PUsbHostSendSetupRqt(uint8_t *setupPkt, uint8_t *buf_p) . . . . .	557
5.38.4.18 CyU3PUsbHostSetDeviceAddress(uint8_t devAddr) . . . . .	557
5.38.4.19 CyU3PUsbHostStart(CyU3PUsbHostConfig_t *hostCfg) . . . . .	558
5.38.4.20 CyU3PUsbHostStop(void) . . . . .	558
5.39 firmware/u3p_firmware/inc/cyu3usbott.h File Reference . . . . .	558
5.39.1 Detailed Description . . . . .	560
5.39.2 Typedef Documentation . . . . .	560
5.39.2.1 CyU3POtgChargerDetectMode_t . . . . .	560
5.39.2.2 CyU3POtgConfig_t . . . . .	561
5.39.2.3 CyU3POtgEvent_t . . . . .	561
5.39.2.4 CyU3POtgEventCallback_t . . . . .	561
5.39.2.5 CyU3POtgMode_t . . . . .	561
5.39.2.6 CyU3POtgPeripheralType_t . . . . .	562
5.39.3 Enumeration Type Documentation . . . . .	562
5.39.3.1 CyU3POtgChargerDetectMode_t . . . . .	562
5.39.3.2 CyU3POtgEvent_t . . . . .	562
5.39.3.3 CyU3POtgMode_t . . . . .	563
5.39.3.4 CyU3POtgPeripheralType_t . . . . .	563
5.39.4 Function Documentation . . . . .	564
5.39.4.1 CyU3POtgGetMode(void) . . . . .	564
5.39.4.2 CyU3POtgGetPeripheralType(void) . . . . .	564
5.39.4.3 CyU3POtgHnpEnable(CyBool_t isEnabled) . . . . .	564
5.39.4.4 CyU3POtgIsDeviceMode(void) . . . . .	566
5.39.4.5 CyU3POtgIsHnpEnabled(void) . . . . .	566
5.39.4.6 CyU3POtgIsHostMode(void) . . . . .	566

5.39.4.7	CyU3POtgIsStarted(void) . . . . .	567
5.39.4.8	CyU3POtgIsVBusValid(void) . . . . .	567
5.39.4.9	CyU3POtgRequestHnp(CyBool_t isEnabled) . . . . .	567
5.39.4.10	CyU3POtgSrpAbort(void) . . . . .	568
5.39.4.11	CyU3POtgSrpStart(uint32_t repeatInterval) . . . . .	568
5.39.4.12	CyU3POtgStart(CyU3POtgConfig_t *cfg) . . . . .	568
5.39.4.13	CyU3POtgStop(void) . . . . .	569
5.40	firmware/u3p_firmware/inc/cyu3utils.h File Reference . . . . .	569
5.40.1	Detailed Description . . . . .	571
5.40.2	Macro Definition Documentation . . . . .	571
5.40.2.1	CY_U3P_MAKEDWORD . . . . .	571
5.40.3	Function Documentation . . . . .	571
5.40.3.1	CyU3PBusyWait(uint16_t usWait) . . . . .	571
5.40.3.2	CyU3PComputeChecksum(uint32_t *buffer, uint32_t length, uint32_t *chkSum) . . . . .	571
5.40.3.3	CyU3PMemCopy32(uint32_t *dest, uint32_t *src, uint32_t count) . . . . .	572
5.40.3.4	CyU3PReadDeviceRegisters(uint32_t *regAddr, uint8_t numRegs, uint32_t *dataBuf) . . . . .	572
5.40.3.5	CyU3PWriteDeviceRegisters(uint32_t *regAddr, uint8_t numRegs, uint32_t *dataBuf) . . . . .	573
5.41	firmware/u3p_firmware/inc/cyu3vic.h File Reference . . . . .	573
5.41.1	Detailed Description . . . . .	574
5.41.2	Enumeration Type Documentation . . . . .	574
5.41.2.1	CyU3PVicVector_t . . . . .	574
5.41.3	Function Documentation . . . . .	575
5.41.3.1	CyU3PVicClearInt(void) . . . . .	575
5.41.3.2	CyU3PVicDisableAllInterrupts(void) . . . . .	575
5.41.3.3	CyU3PVicDisableInt(uint32_t vectorNum) . . . . .	575
5.41.3.4	CyU3PVicEnableInt(uint32_t vectorNum) . . . . .	576
5.41.3.5	CyU3PVicEnableInterrupts(uint32_t mask) . . . . .	576
5.41.3.6	CyU3PVicInit(void) . . . . .	576
5.41.3.7	CyU3PVicIntGetPriority(uint32_t vectorNum) . . . . .	577
5.41.3.8	CyU3PVicIntGetStatus(void) . . . . .	577
5.41.3.9	CyU3PVicIntSetPriority(uint32_t vectorNum, uint32_t priority) . . . . .	577
5.41.3.10	CyU3PVicIRQGetStatus(void) . . . . .	578
5.41.3.11	CyU3PVicSetupIntVectors(void) . . . . .	578
Index		579



# Chapter 1

## EZ-USB FX3 Api Reference Guide

The EZ-USB family includes multiple general purpose USB peripheral controllers that provide the advantages of high performance and flexible usage models to system designers. The FX3, FX3S and CX3 devices are the latest additions to this family, and provide the high performance USB 3.0 SuperSpeed capability in addition to a variety of industry standard peripheral interfaces.

### 1.1 Introduction

#### Description

Cypress EZ-USB FX3 is the next generation USB 3.0 peripheral controller providing highly integrated and flexible features that enable developers to add USB 3.0 functionality to any system.

It has a fully configurable, parallel, General Programmable Interface called GPIF II, which can connect to any processor, ASIC, DSP, or FPGA. It has an integrated USB PHY and controller along with a 32-bit micro-controller (ARM926EJ-S) for powerful data processing and for building custom applications.

The FX3S device is an extension to the FX3 device that adds the capability to control SD/eMMC/SDIO peripherals to the feature set supported by the FX3.

The EZ-USB CX3 controller is an extension to the EZ-USB FX3 device which adds the ability to interface with and perform uncompressed video transfers from Image Sensors implementing the MIPI CSI-2 interface.

These devices have an integrated inter-port DMA architecture which enables data transfers at speeds greater than 400 MBps (Upto 300MBps for CX3). An integrated USB 2.0 OTG controller (on FX3/FX3S) enables applications that need dual role usage scenarios. There is up to 512 KB of on-chip SRAM for code and data usage.

There are also serial peripherals such as UART, SPI, I2C, I2S and GPIO for communicating with on board peripherals.

### 1.2 EZ-USB FX3 SDK

The EZ-USB FX3 SDK is a full featured firmware solution that allows customers to leverage the features of the FX3/FX3S/CX3 devices, and build a full featured design in a short time.

#### Description

The FX3 SDK provides a firmware framework that allows FX3/FX3S/CX3 customers to quickly complete the firmware part of their system design. The firmware framework provides a set of drivers and convenience APIs for the various hardware blocks on these devices. The drivers abstract the complexity of the FX3 device architecture and provide easy-to-use API interfaces for customers to configure and control the FX3 device operation.

## 1.2.1 FX3 API Libraries

The FX3 SDK provides a set of libraries which provide APIs to configure and control the operation of the FX3 device.

### Description

The FX3 SDK provides a firmware framework which includes an embedded RTOS, drivers for each of the FX3 device blocks and a set of APIs that configure and control the operation of the device. These APIs abstract out the complexity of the device hardware and allows users to focus on the core application logic.

The drivers and API are structured in the form of multiple static libraries that adhere to the ARM EABI conventions; and can be linked in to application built using compiler toolchains such as gcc that support the EABI standard.

The following libraries are part of the SDK release and can be found under the firmware/u3p\_firmware/lib folder in the installation:

- cyu3threadx.a : This library provides the ThreadX Operating System from Express Logic. All of the Operating System functionality is enabled and the licensing terms allow users to use the OS services free of cost.
- cyfxapi.a : This library provides the drivers and APIs for the core FX3 device functionality including memory, cache and interrupt control, DMA, USB and GPIF blocks.
- cyu3lpp.a : This library provides the drivers and APIs for the serial peripheral blocks (UART, SPI, I2C, I2S and GPIO) on the FX3 device. The sources for this library are also provided and can be customized as needed.
- cyu3sport.a : This library provides the drivers and APIs for the Storage (SD/MMC/SDIO) interface on the FX3S device. This library only needs to be used when building storage enabled applications using the FX3S device.
- cyu3mipicsi.a : This library provides the APIs for the MIPI-CSI2 interface on the CX3 device. This library only needs to be used when building Image Sensor applications on the CX3 which use the CX3 MIPI-CSI2 interface.

The API definitions for all of these libraries is provided in the header files under the firmware/u3p\_firmware/inc folder in the SDK installation.

## 1.2.2 FX3 Boot API Library

The FX3 SDK also provides a low footprint API library which supports USB, SPI, I2C, UART and GPIO interfaces. This library can be used to built custom boot-loaders or minimal FX3 applications.

### Description

Some FX3 based systems may need to use a custom boot-loader due to one of the following reasons:

1. The system needs a custom set of USB descriptors (with customer specific VID/PID) when booting from the USB host.
2. The full FX3 application is too large to fit on the EEPROM/Flash device used for booting.

The SDK provides a separate API library to facilitate such applications. This library only supports USB device mode, I2C, SPI, UART and GPIO (simple only) operations. DMA support is limited and covers only transfer of single data buffers into or out of the device.

These APIs are provided as a separate static library which has no connection with the libraries described above. It is not possible to use both sets of libraries in a single FX3 application.

When using the Boot API to implement a USB based boot-loader application, it is possible to transfer control from the boot application to the full application and back without going through USB re-enumeration. This method allows seamless firmware upgrades without the need to go through multiple driver binding steps on the host side.

The library and header files for the boot API are found under the firmware/boot\_fw/lib and firmware/boot\_fw/include folders in the SDK installation.

### 1.2.3 FX3 Application Examples

The FX3 SDK includes a set of application examples which use the FX3 API to demonstrate various types of data transfer.

#### Description

In addition to these driver and API libraries, the FX3 SDK also provides a set of application examples. These examples demonstrate how the APIs can be used to put together applications. The main focus of these examples is the USB and DMA configuration which forms the core of most FX3 applications. Specific examples demonstrate the use of the GPIF, Serial Peripheral and Storage APIs.

Please refer to the FX3 Programmer's Manual for details on the various application examples.

## 1.3 Embedded Real Time Operating System

The FX3 SDK provides a full-featured embedded Real-Time Operating System (RTOS) which can be used to create complex multi-threaded applications.

#### Description

The FX3 firmware framework makes use of a Embedded Real-Time Operating System. The drivers for various peripheral blocks in the platform are implemented as separate threads and other OS services such as Semaphores, Message Queues, Mutexes and Timers are used for inter-thread communication and task synchronization.

All of the RTOS services are made available to the user application through a set of wrapper functions that abstract out the actual RTOS API calls. This allows users to implement their applications using multiple communicating threads with a full set of IPC mechanisms for synchronization and data protection.

The ThreadX operating system from Express Logic is used as the embedded RTOS in the FX3 device. All of the functionality supported by the ThreadX OS is made available for use by the application logic. Some constraints on their use are placed to ensure the smooth functioning of all of the drivers.

#### See also

[CyU3PThread](#)  
[CyU3PQueue](#)  
[CyU3PMutex](#)  
[CyU3PSemaphore](#)  
[CyU3PEvent](#)  
[CyU3PTimer](#)  
[CyU3PBytePool](#)  
[CyU3PBlockPool](#)

## 1.4 DMA Management

The DMA manager in the firmware library provides functions to create, configure, control and operate DMA channels within the FX3 device.

#### Description

The FX3 device architecture includes a DMA fabric that is used to steer data between various interfaces of the device. The DMA manager provides a set of functions that allow the user application to create data flows between any pair of interfaces, to configure its behavior and to steer or monitor the actual flow of data packets on this path.

### 1.4.1 DMA Sockets

A DMA socket is a FX3 device construct that maps to one end of a data path into or out of the device.

### Description

Each interface of the FX3 device (such as USB or Processor port) can support multiple independent data flows into or out of the device through that port. Each data flow on a port is handled by a hardware block called a DMA socket. Each port supports a device defined number of sockets all of which can function independently of each other.

For example, different sockets on the Processor port will be used to handle the incoming data going to the storage device and to handle the outgoing data that originated from a USB endpoint.

A socket that takes data coming in from an external interface and directs it into the FX3 system memory is called a Producer or Ingress socket. A socket through which data from the system memory goes out of the FX3 device is called a Consumer or Egress socket.

See also

[CyU3PDmaSocket\\_t](#)

### 1.4.2 DMA Buffers

A DMA buffer is a memory buffer in the FX3 system memory that is assigned to hold one or more packets of data in a data flow.

### Description

All data flowing through the FX3 device passes through the system memory. i.e., All data packets being streamed from a USB endpoint to the external Processor on the P-port pass through designated buffers in the FX3 system memory. Depending on the type of data path and the bandwidth requirements, each data flow might require different sizes and number of buffers in the device memory.

### 1.4.3 DMA Descriptors

A DMA descriptor is a data structure that binds the buffers and sockets corresponding to a data flow.

### Description

The DMA fabric in the FX3 device makes use of data structures called as DMA descriptors to bind the DMA buffers used for a data flow with the DMA sockets that form its ends. Each descriptor contains information on the location, size and status of one DMA buffer along with the ids of its producer and consumer sockets.

See also

[CyU3PDmaDescriptor\\_t](#)

### 1.4.4 DMA Channels

A DMA channel is a software construct that encapsulates all of the hardware elements that are used in a single data flow.

### Description

The DMA manager in the FX3 firmware library introduces the notion of a DMA channel that encapsulates the hardware resources such as sockets, buffers and descriptors used for handling a data flow through the device. The channel concept is used to hide the complexity of configuring all of these resources in a consistent manner. The DMA manager provides API functions that can be used to create data flows between any two interfaces on the FX3 device.

The relationship between the DMA sockets, descriptors and buffers in a DMA channel is shown below.

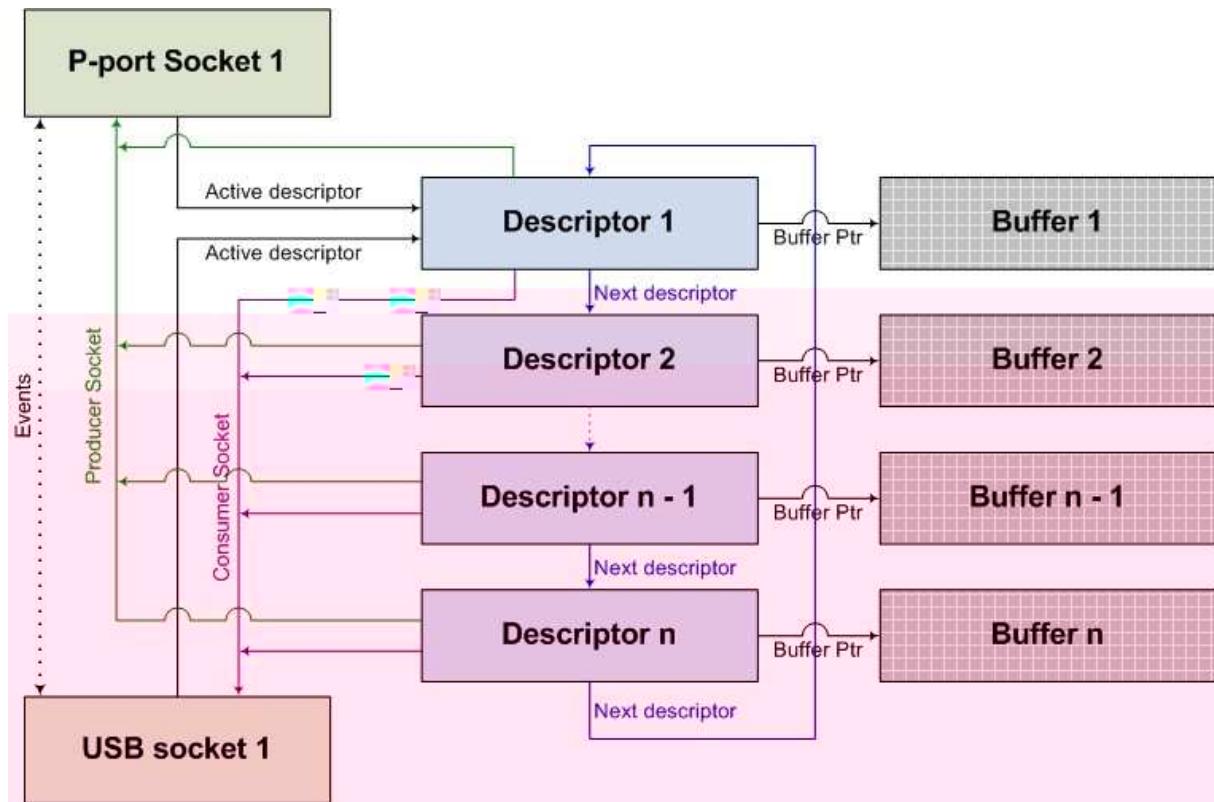


Figure 1.1: Resource linkage in a DMA channel

Since different applications can require different levels of firmware involvement in the processing of the data flow, the DMA manager supports multiple channel types that require different levels of software intervention.

The following channel types are supported:

- Simple Channels

These are DMA channels that bind one producer socket with one consumer socket. That is, the data flows from one socket to another in the case of Simple DMA channels.

1. Auto Channel

An Auto channel is one where there is no software involvement in the processing of individual data packets. A channel of this type can be created, configured to transfer a specified amount of data and then left alone. The DMA manager will come back with a notification once the specified data transfer has been completed. It is possible to set the transfer length as infinite, if these notifications are not required.

2. Auto Channel with Signalling

This is a special case of the Auto channel where the DMA manager provides notifications for every data packet that is handled. The notifications cannot be used to control the data flow because the packets would already have been forwarded to the consumer, but can be used for statistics collection or for error checks.

3. Manual Channel

A manual channel is one which requires software intervention for the handling of each individual data packet. The user application will be notified on arrival of each data packet from the producer. The application can make desired changes to the data packet (content as well as size changes) before sending it on to the consumer. The throughput obtained on a manual channel will depend on the amount of processing done by the application on each packet.

4. Manual IN Channel

This channel type is used for data flows from an external interface to the application running on the FX3 CPU. The application gets notified on the arrival of each data packet and can use this to read and analyze the packet contents.

## 5. Manual OUT Channel

This channel type is used when the application running on the FX3 CPU needs to send data out to an external device. The application can generate the data packets one at a time, and call a function to have it sent out through the specified socket. It is possible for the application to create a synthetic Manual channel by combining a Manual IN channel and a Manual OUT channel with suitable processing.

- **Multi-Channels**

Multi-channels are specialized versions of DMA channels that involve either multiple producers or multiple consumers. Based on whether there are multiple producers or multiple consumers, these can be designated as many-to-one or one-to-many channels. Data transfer will be performed in an interleaved fashion. Multi-channels can be auto or manual channels based on the operating model.

1. **One-to-Many Auto Channel**

This Multi-Channel type has one producer and two or more consumers connected in an interleaved manner. The first buffer of data received by the producer is sent to the first consumer, the second buffer to the second consumer; and so on. As this is an AUTO channel, the user sets up the transfer size and gets notified when the specified amount of data has been transferred by the producer.

2. **One-to-Many Manual Channel**

This is similar to the One-to-Many AUTO channel, but operates in manual mode. Produce event notifications are provided whenever the producer has received a buffer of data. A commit operation sends the data out through the next consumer socket in the interleaved sequence.

3. **Many-to-One Auto Channel**

The Many-to-One channel type connects multiple producer sockets to a single consumer socket. In this case, data received through multiple sockets is aggregated into a single flow and sent out through a consumer socket. The channel operating model is AUTO and event notifications are provided only when the desired amount of data has been transferred by the consumer socket.

4. **Many-to-One Manual Channel**

This is a MANUAL version of the Many-to-One channel, where the user application is notified when data is received by any one of the producer sockets. Even if data is received out of order by the various sockets, it will only be sent out in the correct sequence which is defined when creating the DMA channel.

5. **Multicast Channel**

This is a special one to many channel where there is one producer and a maximum of four consumers. The data received on the producer is sent to each of the consumers. The channel type only supports a manual mode of operation.

### See also

[CyU3PDmaType\\_t](#)  
[CyU3PDmaChannel](#)  
[CyU3PDmaMultiType\\_t](#)  
[CyU3PDmaMultiChannel](#)

## 1.5 Logging Support

This section documents the APIs that are provided to facilitate logging of firmware activity for debug purposes.

### Description

The FX3 API library includes a provision to log firmware actions and send them out to a selected target such as the UART console. The logs can be generated by the API library or the drivers themselves, or be messages sent by the application logic.

The log messages are classified into two types:

1. Codified messages: These messages contain only a two byte message ID and a four byte parameter. This kind of messages are used to compress the log output; and require an external decoder to interpret the logs

based on the message ID.

2. Verbose messages: These are free-form string messages. A function with a signature similar to printf is provided for generating these messages.

The log messages are further classified based on priority levels ranging from 0 to 9. 0 is the highest priority

## 1.6.2 USB 2.0 Host Mode

The FX3 device supports a programmable USB 2.0 host implementation that can control a single USB Hi-Speed, Full speed or Low speed peripheral. The USB host API in the FX3 SDK allow the user to control the host port and to perform transfers on the control and data pipes.

### Description

When functioning as a USB 2.0 host, the FX3 device supports a single root port which can be connected to any USB 2.0 device (hubs are not supported). The USB host mode API requires that the USB driver be placed into a specific host mode of operation. If a dual role device implementation is required, refer to the OTG section of the USB API.

The host API provides functions to control the root port as well as to configure/control the connected peripheral device. Both periodic (isochronous and interrupt) as well as asynchronous (control and bulk) transfers are supported. The APIs provided help implementation of raw data transfers from/to the device endpoints, and does not provide an USB host stack or class driver support.

### See also

[CyU3PUsbHostStart](#)

## 1.6.3 USB On-The-Go (OTG) Mode

The USB API supports the use of FX3 as a OTG A or B device, and supports the standard OTG Session Request (SRP) and Host Negotiation (HNP) protocols.

### Description

If FX3 is to be used as a dual-role USB On-The-Go (OTG) device, the USB driver needs to be initialized for the OTG mode of operation. In this case, the driver monitors the state of the ID pin and performs the following functions:

- Performs ACA charger detection as per the USB Batter Charging Specification 1.1.
- Detects connection type (A device or B device) and sets up the device accordingly.

The OTG APIs also provide functions to perform the Session Request Protocol (SRP) for requesting a VBus supply; and the Host Negotiation Protocol (HNP) to initiate a device role change.

### See also

[CyU3POtgMode\\_t](#)

[CyU3POtgStart](#)

## 1.7 Serial Peripheral Interfaces

The serial peripheral interfaces and the corresponding API in the FX3 library support data exchange between the FX3 device and external peripherals or controllers that are connected through standard peripheral interfaces.

### Description

The FX3 device supports a set of serial peripheral interfaces that can be used to connect a variety of slave or peer devices to FX3. The peripheral interfaces supported by the device are:

1. UART
2. I2C
3. SPI

4. I2S
5. GPIOs

The I2C, SPI and I2S interface implementations on the FX3 device are master mode only and can talk to a variety of slave devices. The I2C interface is also capable of functioning in multi-master mode where other I2C master devices are present on the bus.

### 1.7.1 UART Interface

The UART API allows users to configure the UART interface parameters and to setup UART data transfers.

#### Description

The UART interface on the FX3 device can communicate with other devices at baud rates ranging from 300 to 4608000. The UART data width is always 8 bit, but the number of stop bits and the parity configuration can be selected by the user. The UART interface also supports flow control, and can perform CPU based (using a register interface) or DMA based data transfers.

The UART API provides functions to power up the UART interface, configure the interface parameters and to initiate data transfers.

#### See also

[CyU3PUartInit](#)  
[CyU3PUartSetConfig](#)

### 1.7.2 I2C interface

the I2C API allows users to configure the I2C interface on the FX3 device and to perform I2C data transfers.

#### Description

The I2C interface on the FX3 device is a master-only implementation that can work with multiple slave devices in a single-master or multi-master configuration. The interface can be configured to function at all of the standard I2C frequencies.

The I2C API in the FX3 SDK allows the user to power up the I2C interface, configure the I2C interface and perform data transfers from/to a variety of slave devices.

#### See also

[CyU3PI2cInit](#)  
[CyU3PI2cSetConfig](#)

### 1.7.3 SPI interface

The SPI API allows users to configure the SPI interface and perform transfers from/to one or more SPI slave devices.

#### Description

The SPI interface on the FX3 device is a master-only implementation that can work with one or more slave devices at a variety of frequencies. The interface provides a single SPI Slave Select (SSN) pin that is directly controlled by the SPI block. FX3 GPIOs need to be used as the SSN pin when talking to additional slave devices.

The SPI API allows the user to configure the SPI interface parameters and to perform data transfers. Both CPU based and DMA based data transfers are supported; and read and write transfers can be performed individually or together.

See also

[CyU3PSpiInit](#)  
[CyU3PSpiSetConfig](#)

### 1.7.4 I2S interface

The I2S API in the FX3 SDK allows users to configure the I2S interface and perform data transfers out on one or two audio channels.

#### Description

The Inter-IC-Sound (I2S) interface is a serial wire interface used for audio streams. The FX3 device provides an I2S master interface that can output stereo or mono audio streams at different bit rates.

The I2S API allows users to configure the I2S interface parameters, and to perform write transfers to the external I2S device. Even though CPU based transfers are supported, it is expected that DMA transfers will typically be used when talking to I2S devices.

See also

[CyU3PI2sInit](#)  
[CyU3PI2sSetConfig](#)

### 1.7.5 General Purpose IO (GPIO) Support

Almost all pins on the FX3 device which are unused for other functions can be used as GPIOs. The GPIO API supports configuration of the GPIO functionality.

#### Description

Most of the IOs on the FX3 device can be multiplexed to perform one of many different functions. Any of these IOs that are otherwise unused can be configured as a General Purpose IO (GPIO) pin.

The FX3 device supports two classes of GPIO pins:

1. Simple GPIO: The simple GPIOs are standard IO pins that also support a programmable interrupt function.
2. Complex GPIO: A few of the GPIOs can be configured to perform more complex operations such as PWM output, input pulse width measurement etc. These GPIOs have an associated timer which can be used to measure elapsed time during CPU operations as well.

FX3 also provides software controlled pull-up or pull-down resistors internally on all the digital I/O pins. These can be enabled/disabled individually for any of the GPIO pins.

See also

[CyU3PGpioInit](#)  
[CyU3PGpioSetSimpleConfig](#)  
[CyU3PGpioSetComplexConfig](#)

## 1.8 GPIF II Driver and API

The GPIF II Driver and API allows users to configure the GPIF II interface functionality and to control data transfers across the GPIF II interface.

**Description**

The FX3 device features a GPIF II interface that allows it to be connected to other controllers, ASICs or FPGAs. The programmability of the GPIF II interface allows it to meet the connectivity and timing requirements of the peer device without any glue logic. FX3 can act as a master or as a slave device for transfers across the GPIF II interface.

The GPIF II Designer utility in the FX3 SDK allows users to define the signal and timing parameters for the required interface, and generates the corresponding configuration values in the form of a C header file. This can then be integrated into the FX3 firmware application using the GPIF API.

The GPIF API provides functions to initialize the interface and perform control operations.

### 1.8.1 GPIF Configuration

The GPIF configuration for a specific protocol is typically generated in the form of a set of data structures by the GPIF II Designer tool and programmed using a set of GPIF APIs that take these data structures as parameters.

**Description**

The GPIF block is configured by writing to a set of device registers and configuration memories. The GPIF II Designer utility can be used to generate the configuration data corresponding to the communication protocol to be implemented. This tool generates the GPIF configuration information in the form of a C header file that defines a set of data structures. The FX3 SDK and the GPIF II Designer installation also include a set of header files that have the configuration data for a number of commonly used protocols.

The [CyU3PGpifLoad\(\)](#) API can be used by the firmware application to load the configuration generated by the GPIF II Designer tool. This API in turn internally uses the [CyU3PGpifWaveformLoad\(\)](#), [CyU3PGpifInitTransFunctions\(\)](#) and [CyU3PGpifConfigure\(\)](#) APIs to complete the configuration. These APIs can be used directly as a replacement to the [CyU3PGpifLoad\(\)](#) call with the constraint that the [CyU3PGpifConfigure\(\)](#) call should be made after the [CyU3PGpifWaveformLoad\(\)](#) and [CyU3PGpifInitTransFunctions\(\)](#) calls.

**See also**

[CyU3PGpifLoad](#)  
[CyU3PGpifWaveformLoad](#)  
[CyU3PGpifInitTransFunctions](#)  
[CyU3PGpifConfigure](#)

### 1.8.2 GPIF-II Resources

The GPIF block implements multiple counter and comparator elements that can be initialized and controlled using the GPIF API.

**Description**

The GPIF II hardware block also implements a set of hardware resources that supplement the GPIF state machine operation. These resources include three counters and comparators that can be used to compare data, address and control signal values against preset threshold values.

The counter resources include a 16 bit control counter, a 32 bit address counter and a 32 bit data counter. These counters are initialized through the [CyU3PGpifLoad\(\)](#) API as part of the configuration load or at other points through the [CyU3PGpifInitCtrlCounter\(\)](#), [CyU3PGpifInitAddrCounter\(\)](#) and [CyU3PGpifInitDataCounter\(\)](#) APIs. The counters can be reset explicitly through these APIs or by the GPIF state machine. The counters can only be updated (increment/decrement) through the GPIF state machine. When the count value reaches the programmed limit, a GPIF callback function can be generated. The counter match condition can also be used as a transition trigger variable in the state machine.

There are three comparators: a control comparator, an address comparator and a data comparator that can be used to continuously check the current values on the control, address and data signal groups against a user configured pattern. The patterns for comparison can be programmed initially through the [CyU3PGpifLoad\(\)](#) API and at later points through the [CyU3PGpifInitComparator\(\)](#) API. When any of the comparators detect a match, a GPIF callback

function can be generated. The comparator match can also be used as a transition trigger variable in the state machine.

#### See also

[CyU3PGpifInitComparator](#)  
[CyU3PGpifInitCtrlCounter](#)  
[CyU3PGpifInitDataCounter](#)  
[CyU3PGpifInitAddrCounter](#)

### 1.8.3 GPIF State Machine Control

A set of GPIF APIs are provided to start and control the operation of the GPIF state machine.

#### Description

The firmware application may require the GPIF state machine to be started and stopped multiple times during runtime.

In most cases where the FX3 device is functioning as a slave device, there will be a single GPIF state machine and the application needs to start it as soon as the configuration has been loaded. The [CyU3PGpifSMStart\(\)](#) API can be used to start the state machine operation in this case.

If the application is using the FX3 device as a master on the GPIF interface, there may be multiple disjoint state machines that implement different parts of the communication protocol. The [CyU3PGpifSMSwitch\(\)](#) API can be used to switch between different state machines in this case.

There may be cases where the GPIF state machine operation is to be suspended and later resumed. The [CyU3PGpifSMControl\(\)](#) API can be used to pause or resume state machine operation.

If the state machine operation is to be stopped at some point and restarted from the reset state later, the [CyU3PGpifDisable\(\)](#) API can be used with the forceReload parameter set to CyFalse. The [CyU3PGpifSMStart\(\)](#) API can then be used to restart the state machine.

If the active GPIF configuration is to be changed, the [CyU3PGpifDisable\(\)](#) API can be used with the forceReload parameter set to CyTrue. The [CyU3PGpifLoad\(\)](#) API can then be used to load a fresh configuration.

#### See also

[CyU3PGpifSMStart](#)  
[CyU3PGpifSMSwitch](#)  
[CyU3PGpifDisable](#)

### 1.9 Storage Driver and API

The Storage driver and API for FX3S supports initialization of and data transfers from/to SD/MMC/SDIO peripherals.

#### Description

The FX3S device has two storage ports which can be connected to SD cards, eMMC devices or SDIO cards. The device and the storage driver support SD cards upto version 3.0, eMMC devices upto version 4.41 and SDIO cards upto version 3.0.

The storage driver identifies the type of storage device connected on each of the enabled storage ports, and initializes it with the appropriate command sequence. The storage API provides functions to query the storage device properties like device type, memory capacity, number of SDIO functions etc. APIs are provided to perform read/write transfers to SD/MMC memory devices as well as to SDIO cards.

The storage API also provides functions to send individual commands to the storage device and obtain the corresponding responses. The device initialization will be performed by the storage driver in this case as well, and the

command mode can be used only after device initialization.

#### See also

[CyU3PSibStart](#)  
[CyU3PSibQueryDevice](#)  
[CyU3PSibReadWriteRequest](#)  
[CyU3PSibVendorAccess](#)  
[CyU3PSdioByteReadWrite](#)  
[CyU3PSdioExtendedReadWrite](#)

## 1.10 MIPI-CSI2 and Fixed Function GPIF Interface for CX3

The CX3 device provides the ability to interface with and perform uncompressed video transfers from Image Sensors implementing the MIPI CSI-2 interface.

#### Description

The CX3 device provides a MIPI-CSI2 interface block which can be interfaced with Image Sensors implementing the MIPI-CSI2 interface and is capable of performing uncompressed high-definition video transfers from such sensors via a fixed-function GPIF II interface.

This SDK release provides API support for configuring and utilizing the MIPI-CSI2 interface block and configuration APIs for setting the GPIF II interface width for the fixed function GPIF II interface on the CX3 part.

Additionally, example projects which demonstrate the usage of CX3 MIPI-CSI2 APIs are provided. The example projects implement high-definition (1920x1080 pixels, 16/24Bits per pixel) uncompressed video transfers at upto 30 frames per second from an Aptina AS0260 image sensor over USB 3.0. An example project demonstrating 5Megapixel (2592x1944 pixels, 16 bits per pixel) uncompressed video transfers at upto 15 frames per second from an Omnivision OV5640 image sensor over USB 3.0 is also provided.

#### See also

[CyU3PMipicsiInit](#)  
[CyU3PMipicsiDeInit](#)  
[CyU3PMipicsiSetIntfParams](#)  
[CyU3PMipicsiQueryIntfParams](#)  
[CyU3PMipicsiReset](#)  
[CyU3PMipicsiSleep](#)  
[CyU3PMipicsiWakeups](#)  
[CyU3PMipicsiGetErrors](#)  
[CyU3PMipicsiGpifLoad](#)

## 1.11 Memory Allocation

The FX3 firmware makes use of dynamic memory allocation to obtain memory for the RTOS objects (thread stacks, message queues etc.) and for the DMA buffers required during channel creation. A set of functions are made available to perform this dynamic memory allocation.

### 1.11.1 Driver Heap

This is a memory allocator that is used for allocating memory required for the FX3 driver threads. This allocator can also be used to allocate memory required by RTOS objects created in the user application code.

**Description**

The drivers for various blocks in the FX3 device make use of RTOS threads for execution, so that the driver operation need not be scheduled into the application code flow. The SDK obtains memory for the thread specific stacks and other OS objects (message queues) using the CyU3PMemAlloc function. This function is not provided within the SDK libraries, but is provided in source form in the cyfxtx.c file embedded in all of the firmware examples.

The default implementation of this function (in cyfxtx.c) makes use of the ThreadX byte pool services. If required, users can re-implement the following functions to use a different memory allocation scheme.

Buffers allocated through these functions do not have any specific alignment requirements. As with all memory allocators, the buffers will be aligned to 4-byte (DWORD) boundaries in terms of address and size.

**See also**

[CyU3PMemInit](#)  
[CyU3PMemAlloc](#)  
[CyU3PMemFree](#)  
[CyU3PFreeHeaps](#)

### 1.11.2 Buffer Heap

This is a memory allocator that is used to allocate memory required for the buffers used in DMA transfers through the FX3 device.

**Description**

This is a memory allocator used to get memory required for the buffers used in DMA transfers through the FX3 device. This allocator is also provided in source form as part of the cyfxtx.c file, and can be modified if required by the application.

The DMA APIs for FX3 internally make use of these functions during DMA channel creation and destruction. These functions can also be called from the user application code itself.

The CyU3PDmaBufferInit function responsible for setting up the allocator, and the memory is obtained and freed using the CyU3PDmaBufferAlloc and CyU3PDmaBufferFree functions. It is required that DMA buffers on FX3 be aligned with cache lines (32 bytes) in terms of address and size. The default implementation provided in the cyfxtx.c satisfies the requirement. This is a custom implementation which imposes a fixed sized memory overhead instead of a per-buffer allocation overhead.

**See also**

[CyU3PDmaBufferInit](#)  
[CyU3PDmaBufferDlInit](#)  
[CyU3PDmaBufferAlloc](#)  
[CyU3PDmaBufferFree](#)

### 1.11.3 Memory Leak Checking

The driver and buffer heap implementations provided by Cypress have provision for run-time memory leak detection.

**Description**

The driver and buffer heap implementations provided by Cypress have provision for run-time memory leak detection. The memory leak detection is enabled using the CyU3PMemEnableChecks and CyU3PBufEnableChecks functions respectively.

If the memory leak detection is enabled, the allocator adds a 32 byte header and a 4 byte footer around each memory block that is allocated. Since the checks change the behavior of the allocator, it is required that the enable function be called before the allocator is set up using the CyU3PMemInit or CyU3PDmaBufferInit function.

The [MemBlockInfo](#) structure defined in [cyu3os.h](#) defines the format of the 32 byte header that preceded each memory block. This structure is used to maintain a doubly linked list of all allocated memory blocks. Memory leaks can be detected by calling the [CyU3PMemGetActiveList](#) or [CyU3PBufGetActiveList](#) function. These functions will return a pointer to the head of the list of active memory blocks. The list can be traversed using the `prev_blk` pointer to identify all active (unfree) memory blocks and their allocation ids and sizes.

Please note that the active list will also contain memory blocks that are allocated internally by the drivers in the SDK. These blocks can be identified by getting the active list before allocating any memory from the application code; and then stopping the list traversal when this node has been reached.

#### See also

[CyU3PMemEnableChecks](#)  
[CyU3PMemGetActiveList](#)  
[CyU3PBufEnableChecks](#)  
[CyU3PBufGetActiveList](#)

### 1.11.4 Memory Corruption Detection

The driver and buffer heap implementations from Cypress support detection of memory corruption around the buffers allocated.

#### Description

As the FX3 system RAM is being shared between DMA buffers, user code and data; it is possible that small programming errors cause failures due to memory corruption. The driver and buffer memory allocators support detection of memory corruption around the allocated buffers.

The corruption detection is performed by placing signatures around the allocated memory blocks and checking for valid signatures on user request. As with the memory leak detection, the memory corruption detection is also enabled using the [CyU3PMemEnableChecks](#) and [CyU3PBufEnableChecks](#) functions.

The actual corruption check is done when any memory is being freed or on explicit user request. When memory is being freed using [CyU3PMemFree](#) or [CyU3PDmaBufferFree](#), the signatures around the buffer being freed are checked. The signatures around all active (un-free) memory blocks can be checked by calling the [CyU3PMemCorruptionCheck](#) or [CyU3PBufCorruptionCheck](#) functions.

The user provided bad memory callback function is called if any corrupted signature is found during the check. Please note the check is stopped at the first instance of corruption, as it is likely that the linked list pointers in the [MemBlockInfo](#) header have also been corrupted when the signature is bad.

#### See also

[CyU3PMemEnableChecks](#)  
[CyU3PMemCorruptionCheck](#)  
[CyU3PBufEnableChecks](#)  
[CyU3PBufCorruptionCheck](#)



## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">CyFx3BootDmaDescriptor_t</a>	Descriptor data structure . . . . .	25
<a href="#">CyFx3BootDmaSocket_t</a>	DMA socket configuration structure . . . . .	26
<a href="#">CyFx3BootDmaSockRegs_t</a>	Actual socket specific register structure . . . . .	27
<a href="#">CyFx3BootGpioSimpleConfig_t</a>	Configuration information for simple GPIOs . . . . .	28
<a href="#">CyFx3BootI2cConfig_t</a>	Structure defining the configuration of the I2C interface . . . . .	29
<a href="#">CyFx3BootI2cPreamble_t</a>	Structure defining the preamble to be sent on the I2C interface . . . . .	30
<a href="#">CyFx3BootIoMatrixConfig_t</a>	Defines the IO matrix configuration parameters . . . . .	32
<a href="#">CyFx3BootPibClock_t</a>	Clock configuration information for the PIB block . . . . .	33
<a href="#">CyFx3BootSpiConfig_t</a>	Structure defining the configuration of SPI interface . . . . .	34
<a href="#">CyFx3BootUartConfig_t</a>	Configuration parameters for the UART interface . . . . .	35
<a href="#">CyFx3BootUsbEp0Pkt_t</a>	USB Setup Packet data structure . . . . .	37
<a href="#">CyFx3BootUsbEpConfig_t</a>	Endpoint configuration information . . . . .	38
<a href="#">CyU3PCardCtx_t</a>	Structure that holds properties of an SD/MMC peripheral . . . . .	39
<a href="#">CyU3PDebugLog_t</a>	FX3 debug logger data type . . . . .	41
<a href="#">CyU3PDmaBuffer_t</a>	DMA buffer data structure . . . . .	42
<a href="#">CyU3PDmaCBIInput_t</a>	DMA channel callback input . . . . .	43
<a href="#">CyU3PDmaChannel</a>	DMA Channel structure . . . . .	44
<a href="#">CyU3PDmaChannelConfig_t</a>	DMA channel parameters . . . . .	48
<a href="#">CyU3PDmaDescriptor_t</a>	Descriptor data structure . . . . .	50

CyU3PDmaMultiChannel	DMA multi-channel structure . . . . .	51
CyU3PDmaMultiChannelConfig_t	DMA multi-channel parameter structure . . . . .	56
CyU3PDmaSocket_t	DMA socket register structure . . . . .	58
CyU3PDmaSocketConfig_t	DMA socket configuration structure . . . . .	59
CyU3PEpConfig_t	Endpoint configuration information . . . . .	60
CyU3PGpifConfig_t	Structure that holds all configuration inputs for the GPIF hardware . . . . .	61
CyU3PGpifWaveData	Information on a single GPIF transition from one state to another . . . . .	63
CyU3PGpioClock_t	Clock configuration information for the GPIO block . . . . .	64
CyU3PGpioComplexConfig_t	Configuration information for complex GPIO pins . . . . .	65
CyU3PGpioSimpleConfig_t	Configuration information for simple GPIOs . . . . .	66
CyU3PI2cConfig_t	Structure defining the I2C interface configuration . . . . .	68
CyU3PI2cPreamble_t	Structure defining the preamble to be sent on the I2C interface . . . . .	69
CyU3PI2sConfig_t	I2S interface configuration parameters . . . . .	70
CyU3PIoMatrixConfig_t	IO matrix configuration parameters . . . . .	71
CyU3PMbox	Structure that holds a packet of mailbox data . . . . .	73
CyU3PMipicsiCfg_t	Structure defining the MIPI-CSI block interface configuration . . . . .	73
CyU3PMipicsiErrorCounts_t	Structure defining MIPI-CSI block Phy errors . . . . .	75
CyU3POtgConfig_t	OTG configuration information . . . . .	76
CyU3PPibClock_t	Clock configuration information for the PIB block . . . . .	77
CyU3PSdioCardRegs	SDIO Card Generic Information and CCCR registers . . . . .	78
CyU3PSibCtx	Structure that holds transfer state corresponding to each storage port . . . . .	80
CyU3PSibDevInfo	Storage (SD/MMC) device properties . . . . .	81
CyU3PSibGlobalData	Data structure that holds the storage driver state . . . . .	83
CyU3PSibIntfParams	Storage interface control parameters . . . . .	85
CyU3PSibLunInfo	Logical unit properties . . . . .	86
CyU3PSpiConfig_t	Structure defining the configuration of SPI interface . . . . .	87
CyU3PSysClockConfig_t	Clock configuration for FX3 CPU, DMA and register access . . . . .	89
CyU3PUartConfig_t	Configuration parameters for the UART interface . . . . .	90
CyU3PUsbDescrPtrs	Pointer to the various descriptors . . . . .	92

<a href="#">CyU3PUsbHostConfig_t</a>		
USB host mode configuration information	.....	93
<a href="#">CyU3PUsbHostEpConfig_t</a>		
Host mode endpoint configuration structure	.....	94
<a href="#">MemBlockInfo</a>		
Structure representing a block of memory that has been dynamically allocated and is in use	..	95



# Chapter 3

## File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

firmware/boot-fw/include/cyfx3device.h	The Boot APIs for FX3 provide a low footprint API library that can be used to put together simple FX3 applications, primarily for the purpose of building custom boot-loaders . . . . .	97
firmware/boot-fw/include/cyfx3dma.h	The DMA driver module in the FX3 boot firmware provides a set of APIs to manage data transfers, and provides DMA interrupt handlers . . . . .	103
firmware/boot-fw/include/cyfx3error.h	This file lists the various error codes that can be returned by the APIs in the FX3 boot library . . . . .	114
firmware/boot-fw/include/cyfx3gpio.h	The file defines the data structures and APIs that are provided for making use of FX3 GPIOs . . . . .	116
firmware/boot-fw/include/cyfx3i2c.h	The I2C interface driver in the FX3 Boot Firmware provides a set of APIs that allow the configuration of the I2C interface properties and data exchange with one or more I2C slave devices . . . . .	121
firmware/boot-fw/include/cyfx3pib.h	The PIB driver module in the FX3 boot firmware enables the PMMC interface on the FX3 device and allows PMMC data transfers . . . . .	127
firmware/boot-fw/include/cyfx3spi.h	SPI (Serial Peripheral Interface) is a serial interface defined for inter-device communication. The FX3 device includes a SPI master that can connect to a variety of SPI slave devices and function at various speeds and modes. This file defines the data structures and APIs that enable SPI slave access from the FX3 boot API library . . . . .	137
firmware/boot-fw/include/cyfx3uart.h	The UART interface manager module is responsible for handling the transfer of data through the UART interface on the device. This file defines the data structures and APIs for UART interface management . . . . .	144
firmware/boot-fw/include/cyfx3usb.h	The FX3 boot library implements a USB driver that supports the device mode of USB operation (no host or OTG support). This file defines the data structures and APIs provided by the USB driver to control the USB operation . . . . .	151
firmware/boot-fw/include/cyfx3utils.h	This file provides some generic utility functions for the use of the FX3 boot library . . . . .	165
firmware/u3p_firmware/inc/cyfx3_api.h	Defines the APIs provided by the FX3 core library . . . . .	166
firmware/u3p_firmware/inc/cyfxapidesc.h	Provides a brief description of the FX3 APIs . . . . .	182
firmware/u3p_firmware/inc/cyfxversion.h	Version information for the FX3 API library . . . . .	182

<a href="#">firmware/u3p_firmware/inc/cyu3cardmgr.h</a>	This file defines the data types and APIs provided by the low level SD/MMC/SDIO driver for FX3S	<a href="#">183</a>
<a href="#">firmware/u3p_firmware/inc/cyu3cardmgr_fx3s.h</a>	This file contains the provides the low level SD/MMC/SDIO driver related macros and definitions	<a href="#">194</a>
<a href="#">firmware/u3p_firmware/inc/cyu3descriptor.h</a>	DMA descriptor management	<a href="#">204</a>
<a href="#">firmware/u3p_firmware/inc/cyu3dma.h</a>	DMA driver and API definitions for EZ-USB FX3	<a href="#">210</a>
<a href="#">firmware/u3p_firmware/inc/cyu3error.h</a>	This file lists the error codes that are returned by the firmware library functions	<a href="#">253</a>
<a href="#">firmware/u3p_firmware/inc/cyu3gpi.h</a>	This file defines the GPIF related data structures and APIs in the FX3 SDK	<a href="#">256</a>
<a href="#">firmware/u3p_firmware/inc/cyu3gpio.h</a>	The GPIO manager module is responsible for handling general purpose IO pins on the FX3 device	<a href="#">272</a>
<a href="#">firmware/u3p_firmware/inc/cyu3i2c.h</a>	The I2C driver in the FX3 firmware provides a set of APIs to configure the I2C interface properties and to perform I2C data transfers from/to one or more slave devices	<a href="#">287</a>
<a href="#">firmware/u3p_firmware/inc/cyu3i2s.h</a>	The I2S (Inter-IC Sound) interface is a serial interface defined for communication of stereophonic audio data between devices. The FX3 device includes a I2S master interface which can be connected to an I2S peripheral. The I2S driver module provides functions to configure the I2S interface and to send mono or stereo audio output on the I2S link	<a href="#">298</a>
<a href="#">firmware/u3p_firmware/inc/cyu3lpp.h</a>	All of the serial peripherals and GPIOs on the FX3 device share a set of common hardware resources that need to be initialized before any of these interfaces can be used. Further, the drivers for all of the serial peripherals share a single RTOS thread; because the CPU load due to each of these is expected to be minimal. This file defines the data types and APIs that are used for the central management of all these serial peripheral blocks	<a href="#">307</a>
<a href="#">firmware/u3p_firmware/inc/cyu3mbox.h</a>	The mailbox handler is responsible for sending/receiving short messages from an external processor through the mailbox registers	<a href="#">317</a>
<a href="#">firmware/u3p_firmware/inc/cyu3mipicsi.h</a>	This file contains the MIPI-CSI interface management data structures, functions and macros for CX3 devices	<a href="#">320</a>
<a href="#">firmware/u3p_firmware/inc/cyu3mmu.h</a>	Cache and Memory Management for the FX3 firmware	<a href="#">339</a>
<a href="#">firmware/u3p_firmware/inc/cyu3os.h</a>	This file defines the RTOS services and wrappers that are provided as part of the FX3 firmware solution. Most of these services are used by the drivers in the FX3 library and need to be provided when porting to a new OS environment	<a href="#">349</a>
<a href="#">firmware/u3p_firmware/inc/cyu3pib.h</a>	The Processor Interface Block (PIB) on the FX3 device contains the GPIF-II controller and the associated DMA sockets and configuration registers. The PIB driver in FX3 firmware is responsible for managing PIB bound data transfers and interrupts. This file contains the data types and API definitions for the general P-port functionality that is independent of the electrical protocol implemented by GPIF-II	<a href="#">395</a>
<a href="#">firmware/u3p_firmware/inc/cyu3sib.h</a>	This file defines the data types and APIs that support SD/MMC/SDIO peripheral access on the EZ-USB FX3S device	<a href="#">411</a>
<a href="#">firmware/u3p_firmware/inc/cyu3sibpp.h</a>	This is an FX3S internal header file that defines some global data structures that are used by the storage driver module	<a href="#">439</a>

firmware/u3p_firmware/inc/cyu3socket.h	All block based data transfers IN or OUT of the FX3 device are performed through hardware blocks called sockets. An end-to-end data flow through the FX3 device (DMA channel) is created by tying two or more sockets together using a shared set of data buffers for intermediate storage. This file provides a set of low level APIs that operate on these sockets, and allow custom data pipes to be created . . . . .	444
firmware/u3p_firmware/inc/cyu3spi.h	SPI (Serial Peripheral Interface) is a serial interface defined for inter-device communication. The FX3 device includes a SPI master that can connect to a variety of SPI slave devices and function at various speeds and modes. The SPI driver module provides functions to configure the SPI interface parameters and to perform data read/writes from/to the slave devices . . . . .	453
firmware/u3p_firmware/inc/cyu3system.h	This file defines the device initialization and configuration functions associated with the FX3 SDK . . . . .	463
firmware/u3p_firmware/inc/cyu3types.h	This file contains definitions for the basic data types used by the FX3 firmware library. If a compiler provided version is available, this can be used instead of the custom ones, by disabling the CYU3_DEFINE_BASIC_TYPES definition . . . . .	487
firmware/u3p_firmware/inc/cyu3uart.h	The UART driver in FX3 firmware is responsible for handling data transfers through the USART interface on the device. This file defines the data structures and APIs for USART interface management . . . . .	489
firmware/u3p_firmware/inc/cyu3usb.h	The USB device mode driver on the FX3 device is responsible for handling USB 3.0 and 2.0 connections, and providing the API hooks for configuring device operations . . . . .	499
firmware/u3p_firmware/inc/cyu3usbconst.h	This file defines constants that are derived from the USB specifications, for the use of the USB driver and user firmware . . . . .	538
firmware/u3p_firmware/inc/cyu3usbhost.h	The FX3 device supports programmable USB host implementation for a single USB host port at USB-HS, USB-FS and USB-LS speeds. The control pipe as well as the data pipes to be used can be configured through a set of USB host mode APIs. The USB host mode APIs also provide the capability to manage the host port . . . . .	545
firmware/u3p_firmware/inc/cyu3usbotg.h	The FX3 device supports USB 2.0 On-The-Go (OTG) functionality which allows the device to identify whether it should function as a USB host or a peripheral. This file defines the data types and APIs provided by the USB driver on FX3 for OTG management . . . . .	558
firmware/u3p_firmware/inc/cyu3utils.h	Utility functions provided by the FX3 library . . . . .	569
firmware/u3p_firmware/inc/cyu3vic.h	Internal functions for managing the VIC and interrupts on the FX3 device . . . . .	573



## Chapter 4

# Data Structure Documentation

## 4.1 CyFx3BootDmaDescriptor\_t Struct Reference

Descriptor data structure.

```
#include <cyfx3dma.h>
```

### Data Fields

- `uint8_t * buffer`
- `uint32_t sync`
- `uint32_t chain`
- `uint32_t size`

### 4.1.1 Detailed Description

Descriptor data structure.

#### Description

This data structure contains the fields that make up a DMA descriptor on the FX3 device.

### 4.1.2 Field Documentation

#### 4.1.2.1 `uint8_t* buffer`

Pointer to buffer used.

#### 4.1.2.2 `uint32_t chain`

Next descriptor links.

#### 4.1.2.3 `uint32_t size`

Current and maximum sizes of buffer.

#### 4.1.2.4 `uint32_t sync`

Consumer, Producer binding.

The documentation for this struct was generated from the following file:

- firmware/boot-fw/include/cyfx3dma.h

## 4.2 CyFx3BootDmaSocket\_t Struct Reference

DMA socket configuration structure.

```
#include <cyfx3dma.h>
```

### Data Fields

- uint32\_t `dscrChain`
- uint32\_t `xferSize`
- uint32\_t `xferCount`
- uint32\_t `status`
- uint32\_t `intr`
- uint32\_t `intrMask`

### 4.2.1 Detailed Description

DMA socket configuration structure.

#### Description

This structure holds all the configuration fields that can be directly updated on a DMA socket. Refer to the `sock_regs.h` file for the detailed break up into bit-fields for each of the structure members.

### 4.2.2 Field Documentation

#### 4.2.2.1 uint32\_t `dscrChain`

The descriptor chain associated with the socket.

#### 4.2.2.2 uint32\_t `intr`

Interrupt status.

#### 4.2.2.3 uint32\_t `intrMask`

Interrupt mask.

#### 4.2.2.4 uint32\_t `status`

Socket status register.

#### 4.2.2.5 uint32\_t `xferCount`

Transfer status for the socket.

#### 4.2.2.6 uint32\_t xferSize

Transfer size for the socket.

The documentation for this struct was generated from the following file:

- firmware/boot\_fw/include/cyfx3dma.h

## 4.3 CyFx3BootDmaSockRegs\_t Struct Reference

Actual socket specific register structure.

```
#include <cyfx3dma.h>
```

### Data Fields

- uvint32\_t dscrChain
- uvint32\_t xferSize
- uvint32\_t xferCount
- uvint32\_t status
- uvint32\_t intr
- uvint32\_t intrMask
- uint32\_t unused2 [2]
- uvint32\_t actBuffer
- uvint32\_t actSync
- uvint32\_t actChain
- uvint32\_t actSize
- uint32\_t unused19 [19]
- uvint32\_t sckEvent

### 4.3.1 Detailed Description

Actual socket specific register structure.

#### Description

This structure represents the actual socket status register space on the FX3 device. This includes a number of reserved and hardware-access only registers which are skipped in the [CyFx3BootDmaSocket\\_t](#) structure.

### 4.3.2 Field Documentation

#### 4.3.2.1 uvint32\_t actBuffer

Buffer pointer from current DMA descriptor.

#### 4.3.2.2 uvint32\_t actChain

Chain field from current DMA descriptor.

#### 4.3.2.3 uvint32\_t actSize

Size field from current DMA descriptor.

**4.3.2.4 uvint32\_t actSync**

Sync field from current DMA descriptor.

**4.3.2.5 uvint32\_t dscrChain**

The descriptor chain associated with the socket

**4.3.2.6 uvint32\_t intr**

Interrupt status register.

**4.3.2.7 uvint32\_t intrMask**

Interrupt mask register.

**4.3.2.8 uvint32\_t sckEvent**

Generate event register.

**4.3.2.9 uvint32\_t status**

Socket configuration and status register.

**4.3.2.10 uint32\_t unused19[19]**

Reserved register space.

**4.3.2.11 uint32\_t unused2[2]**

Reserved register space.

**4.3.2.12 uvint32\_t xferCount**

The completed transfer count for this socket.

**4.3.2.13 uvint32\_t xferSize**

The transfer size requested for this socket. The size can be specified in bytes or in terms of number of buffers, depending on the UNIT field in the status value.

The documentation for this struct was generated from the following file:

- firmware/boot-fw/include/[cyfx3dma.h](#)

## 4.4 CyFx3BootGpioSimpleConfig\_t Struct Reference

Configuration information for simple GPIOs.

```
#include <cyfx3gpio.h>
```

## Data Fields

- `CyBool_t outValue`
- `CyBool_t driveLowEn`
- `CyBool_t driveHighEn`
- `CyBool_t inputEn`
- `CyFx3BootGpioIntrMode_t intrMode`

### 4.4.1 Detailed Description

Configuration information for simple GPIOs.

#### Description

This structure encapsulates all of the configuration information for a simple GPIO on the FX3 device.

If the pin is configured as input, then the fields `driveLowEn` and `driveHighEn` should be `CyFalse`. The `outValue` field is a don't care in this case.

If the pin is configured as an output, the `inputEn` field should be `CyFalse`. The `driveLowEn` and `driveHighEn` fields can be used to select whether the device should drive the pin to low/high levels or just tri-state it.

#### See also

[CyFx3BootGpioIntrMode\\_t](#)

### 4.4.2 Field Documentation

#### 4.4.2.1 CyBool\_t driveHighEn

When set true, the output driver is enabled for `outValue` = `CyTrue` (1), otherwise tristated.

#### 4.4.2.2 CyBool\_t driveLowEn

When set true, the output driver is enabled for `outValue` = `CyFalse` (0), otherwise tristated.

#### 4.4.2.3 CyBool\_t inputEn

When set true, the input state is enabled.

#### 4.4.2.4 CyFx3BootGpioIntrMode\_t intrMode

Interrupt mode for the GPIO. Should be set to `CY_FX3_BOOT_GPIO_NO_INTR`.

#### 4.4.2.5 CyBool\_t outValue

Initial value of the GPIO if configured as output: `CyFalse` = 0, `CyTrue` = 1.

The documentation for this struct was generated from the following file:

- `firmware/boot-fw/include/cyfx3gpio.h`

## 4.5 CyFx3BootI2cConfig\_t Struct Reference

Structure defining the configuration of the I2C interface.

```
#include <cyfx3i2c.h>
```

## Data Fields

- `uint32_t bitRate`
- `CyBool_t isDma`
- `uint32_t busTimeout`
- `uint16_t dmaTimeout`

### 4.5.1 Detailed Description

Structure defining the configuration of the I2C interface.

#### Description

This structure encapsulates all of the configurable parameters that can be selected for the I2C interface. The [CyFx3BootI2cSetConfig\(\)](#) function accepts a pointer to this structure, and updates all of the interface parameters.

The I2C block can function in the bit rate range of 100 KHz to 1MHz. In default mode of operation, the timeouts need to be kept disabled.

#### See also

[CyFx3BootI2cSetConfig](#)

### 4.5.2 Field Documentation

#### 4.5.2.1 `uint32_t bitRate`

Bit rate for the interface (e.g.: 100000 for 100KHz).

#### 4.5.2.2 `uint32_t busTimeout`

Number of core clocks that SCK can be held low for by the slave byte transmission before triggering a timeout error. 0xFFFFFFFFU means no timeout.

#### 4.5.2.3 `uint16_t dmaTimeout`

Number of core clocks DMA can remain not ready before flagging an error. 0xFFFF means no timeout.

#### 4.5.2.4 `CyBool_t isDma`

CyFalse: Register transfer mode, CyTrue: DMA transfer mode

The documentation for this struct was generated from the following file:

- firmware/boot\_fw/include/[cyfx3i2c.h](#)

## 4.6 CyFx3BootI2cPreamble\_t Struct Reference

Structure defining the preamble to be sent on the I2C interface.

```
#include <cyfx3i2c.h>
```

## Data Fields

- `uint8_t buffer [8]`
- `uint8_t length`
- `uint16_t ctrlMask`

### 4.6.1 Detailed Description

Structure defining the preamble to be sent on the I2C interface.

#### Description

All I2C data transfers start with a preamble where the first byte contains the slave address and the direction of transfer. The preamble can optionally contain other bytes where device specific address values or other commands are sent to the slave device.

The FX3 device supports associating a preamble with a maximum length of 8 bytes to any I2C data transfer. This allows the user to specify a multi-byte preamble which covers the slave address, device specific address fields and then initiate the data transfer.

The `ctrlMask` indicate the start / stop bit conditions after each byte of the preamble.

For example, an I2C EEPROM read requires the byte address for the read to be written first. These two I2C operations can be combined into one I2C API call using the parameters of the structure.

Typical I2C EEPROM page Read operation:

Byte 0:

Bit 7 - 1: Slave address.

Bit 0 : 0 - Indicating this is a write from master.

Byte 1, 2: Address to which the data has to be written.

Byte 3:

Bit 7 - 1: Slave address.

Bit 0 : 1 - Indicating this is a read operation.

The buffer field shall hold the above four bytes, the length field shall be four; and `ctrlMask` field will be 0x0004 as a start bit is required after the third byte (third bit is set).

Typical I2C EEPROM page Write operation:

Byte 0:

Bit 7 - 1: Slave address.

Bit 0 : 0 - Indicating this is a write from master.

Byte 1, 2: Address to which the data has to be written.

The buffer field shall hold the above three bytes, the length field shall be three, and the `ctrlMask` field is zero as no additional start/stop conditions are needed.

#### See also

[CyFx3BootI2cSetConfig](#)

### 4.6.2 Field Documentation

#### 4.6.2.1 `uint8_t buffer[8]`

The extended preamble information.

#### 4.6.2.2 `uint16_t ctrlMask`

This field controls the start stop condition after every byte of preamble data. Bits 0 - 7 represent a bit mask for start condition and Bits 8 - 15 represent a bit mask for stop condition. If both bits are set for an index, then the stop condition takes priority.

#### 4.6.2.3 uint8\_t length

The length of the preamble to be sent. Should be between 1 and 8.

The documentation for this struct was generated from the following file:

- firmware/boot\_fw/include/cyfx3i2c.h

## 4.7 CyFx3BootIoMatrixConfig\_t Struct Reference

Defines the IO matrix configuration parameters.

```
#include <cyfx3device.h>
```

### Data Fields

- CyBool\_t isDQ32Bit
- CyBool\_t useUart
- CyBool\_t useI2C
- CyBool\_t useI2S
- CyBool\_t useSpi
- uint32\_t gpioSimpleEn [2]

### 4.7.1 Detailed Description

Defines the IO matrix configuration parameters.

#### Description

Most of the IOs on the FX3 device are multi-purpose; and the specific function that each pin should serve need to be selected during device initialization. This structure defines all of the parameters that are required to configure the FX3 IOs.

Please note that the DQ[15:0] pins, CTL[3:0] pins and the PMODE[2:0] pins are reserved and cannot be enabled as GPIOs through this structure. The GPIO Override APIs need to be used for this purpose.

#### See also

- [CyFx3BootDeviceConfigureIMatrix](#)
- [CyFx3BootGpioOverride](#)

### 4.7.2 Field Documentation

#### 4.7.2.1 uint32\_t gpioSimpleEn[2]

Bitmap variable that identifies pins that should be configured as simple GPIOs.

#### 4.7.2.2 CyBool\_t isDQ32Bit

CyTrue: The GPIF bus width is 32 bit; CyFalse: The GPIF bus width is 16 bit

#### 4.7.2.3 CyBool\_t useI2C

CyTrue: The I2C interface is to be used; CyFalse: The I2C interface is not to be used

#### 4.7.2.4 CyBool\_t useI2S

CyTrue: The I2S interface is to be used; CyFalse: The I2S interface is not to be used.

#### 4.7.2.5 CyBool\_t useSpi

CyTrue: The SPI interface is to be used; CyFalse: The SPI interface is not to be used.

#### 4.7.2.6 CyBool\_t useUart

CyTrue: The UART interface is to be used; CyFalse: The UART interface is not to be used

The documentation for this struct was generated from the following file:

- firmware/boot\_fw/include/cyfx3device.h

## 4.8 CyFx3BootPibClock\_t Struct Reference

Clock configuration information for the PIB block.

```
#include <cyfx3pib.h>
```

### Data Fields

- [CyFx3BootSysClockSrc\\_t clkSrc](#)
- [uint16\\_t clkDiv](#)
- [CyBool\\_t isHalfDiv](#)
- [CyBool\\_t isDIIEnable](#)

### 4.8.1 Detailed Description

Clock configuration information for the PIB block.

#### Description

The clock for the PIB block is derived from the SYS\_CLK. The base clock and frequency divider values are selected through this structure.

#### See also

[CyFx3BootSysClockSrc\\_t](#)  
[CyFx3BootPibInit](#)

### 4.8.2 Field Documentation

#### 4.8.2.1 uint16\_t clkDiv

Divider for the PIB clock. The min value is 2 and max value is 1024.

#### 4.8.2.2 CyFx3BootSysClockSrc\_t clkSrc

The clock source to be used.

#### 4.8.2.3 CyBool\_t isDIIEnable

Whether the DLL should be enabled or not. The DLL in the PIB block should be enabled when implementing Asynchronous GPIF protocols, or Master mode GPIF protocols. It should be left turned off when implementing synchronous slave mode GPIF protocols.

#### 4.8.2.4 CyBool\_t isHalfDiv

If set to true, adds 0.5 to the divider value selected by clkDiv.

The documentation for this struct was generated from the following file:

- firmware/boot\_fw/include/cyfx3pib.h

## 4.9 CyFx3BootSpiConfig\_t Struct Reference

Structure defining the configuration of SPI interface.

```
#include <cyfx3spi.h>
```

### Data Fields

- CyBool\_t isLsbFirst
- CyBool\_t cpol
- CyBool\_t cpha
- CyBool\_t ssnPol
- CyFx3BootSpiSsnCtrl\_t ssnCtrl
- CyFx3BootSpiSsnLagLead\_t leadTime
- CyFx3BootSpiSsnLagLead\_t lagTime
- uint32\_t clock
- uint8\_t wordLen

### 4.9.1 Detailed Description

Structure defining the configuration of SPI interface.

#### Description

This structure encapsulates all of the configurable parameters that can be selected for the SPI interface. The [CyFx3BootSpiSetConfig\(\)](#) function accepts a pointer to this structure, and updates all of the interface parameters.

#### See also

- [CyFx3BootSpiSetConfig](#)
- [CyFx3BootSpiSsnCtrl\\_t](#)
- [CyFx3BootSpiSsnLagLead\\_t](#)

### 4.9.2 Field Documentation

#### 4.9.2.1 uint32\_t clock

SPI interface clock frequency in Hz.

#### 4.9.2.2 CyBool\_t cpha

Clock phase - CyFalse: Slave samples at idle-active edge; CyTrue: Slave samples at active-idle edge

#### 4.9.2.3 CyBool\_t cpol

Clock polarity - CyFalse: SCK idles low; CyTrue: SCK idles high

#### 4.9.2.4 CyBool\_t isLsbFirst

Data shift mode - CyFalse: MSB first; CyTrue: LSB first

#### 4.9.2.5 CyFx3BootSpiSsnLagLead\_t lagTime

Time between the last SCK edge to SSN de-assertion. This is at the end of a transfer and is valid only for hardware controlled SSN.

#### 4.9.2.6 CyFx3BootSpiSsnLagLead\_t leadTime

Time between SSN assertion and first SCLK edge. This is at the beginning of a transfer and is valid only for hardware controlled SSN. Zero lead time is not supported.

#### 4.9.2.7 CyFx3BootSpiSsnCtrl\_t ssnCtrl

Mode of SSN control.

#### 4.9.2.8 CyBool\_t ssnPol

Polarity of SSN line. CyFalse: SSN is active low; CyTrue: SSN is active high.

#### 4.9.2.9 uint8\_t wordLen

Word length in bits. Valid values are 4 - 32.

The documentation for this struct was generated from the following file:

- firmware/boot-fw/include/cyfx3spi.h

## 4.10 CyFx3BootUartConfig\_t Struct Reference

Configuration parameters for the UART interface.

```
#include <cyfx3uart.h>
```

### Data Fields

- [CyBool\\_t txEnable](#)
- [CyBool\\_t rxEnable](#)
- [CyBool\\_t flowCtrl](#)
- [CyBool\\_t isDma](#)
- [CyFx3BootUartBaudrate\\_t baudRate](#)
- [CyFx3BootUartStopBit\\_t stopBit](#)
- [CyFx3BootUartParity\\_t parity](#)

#### 4.10.1 Detailed Description

Configuration parameters for the UART interface.

##### Description

This structure defines all of the configurable parameters for the UART interface such as baud rate, stop and parity bits etc. A pointer to this structure is passed in to the CyFx3BootUartSetConfig function to configure the UART interface.

##### See also

[CyFx3BootUartBaudrate\\_t](#)  
[CyFx3BootUartStopBit\\_t](#)  
[CyFx3BootUartParity\\_t](#)  
[CyFx3BootUartSetConfig](#)

#### 4.10.2 Field Documentation

##### 4.10.2.1 CyFx3BootUartBaudrate\_t baudRate

Baud rate for data transfer.

##### 4.10.2.2 CyBool\_t flowCtrl

Enable Flow control for both RX and TX.

##### 4.10.2.3 CyBool\_t isDma

CyFalse: Byte-by-byte transfer, CyTrue: Block based transfer.

##### 4.10.2.4 CyFx3BootUartParity\_t parity

Parity configuration.

##### 4.10.2.5 CyBool\_t rxEnable

Enable the receiver.

##### 4.10.2.6 CyFx3BootUartStopBit\_t stopBit

The number of stop bits used.

##### 4.10.2.7 CyBool\_t txEnable

Enable the transmitter.

The documentation for this struct was generated from the following file:

- firmware/boot\_fw/include/[cyfx3uart.h](#)

## 4.11 CyFx3BootUsbEp0Pkt\_t Struct Reference

USB Setup Packet data structure.

```
#include <cyfx3usb.h>
```

### Data Fields

- uint8\_t bmReqType
- uint8\_t bReq
- uint8\_t bVal0
- uint8\_t bVal1
- uint8\_t bIdx0
- uint8\_t bIdx1
- uint16\_t wLen
- uint8\_t \* pData

#### 4.11.1 Detailed Description

USB Setup Packet data structure.

##### Description

Control Endpoint setup packet data structure.

#### 4.11.2 Field Documentation

##### 4.11.2.1 uint8\_t bIdx0

wIndex field, LSB.

##### 4.11.2.2 uint8\_t bIdx1

wIndex field, MSB.

##### 4.11.2.3 uint8\_t bmReqType

Direction, type of request and intended recipient

##### 4.11.2.4 uint8\_t bReq

Request being made

##### 4.11.2.5 uint8\_t bVal0

wValue field, LSB.

##### 4.11.2.6 uint8\_t bVal1

wValue field, MSB.

#### 4.11.2.7 `uint8_t* pData`

Pointer to the data

#### 4.11.2.8 `uint16_t wLen`

wLength field.

The documentation for this struct was generated from the following file:

- firmware/boot-fw/include/[cyfx3usb.h](#)

## 4.12 CyFx3BootUsbEpConfig\_t Struct Reference

Endpoint configuration information.

```
#include <cyfx3usb.h>
```

### Data Fields

- `CyBool_t enable`
- `CyFx3BootUsbEpType_t epType`
- `uint16_t streams`
- `uint16_t pcktSize`
- `uint8_t burstLen`
- `uint8_t isoPkts`

### 4.12.1 Detailed Description

Endpoint configuration information.

#### Description

This structure holds all the properties of a USB endpoint. This structure is used to provide information about the desired configuration of various endpoints in the system.

### 4.12.2 Field Documentation

#### 4.12.2.1 `uint8_t burstLen`

Maximum burst length in packets.

#### 4.12.2.2 `CyBool_t enable`

Endpoint status - enabled or not.

#### 4.12.2.3 `CyFx3BootUsbEpType_t epType`

The endpoint type

#### 4.12.2.4 `uint8_t isoPkts`

Number of packets per micro-frame for ISO endpoints.

#### 4.12.2.5 uint16\_t pcktSize

Maximum packet size for the endpoint. Valid range <1 - 1024>

#### 4.12.2.6 uint16\_t streams

Number of bulk streams used by the endpoint.

The documentation for this struct was generated from the following file:

- firmware/boot-fw/include/cyfx3usb.h

## 4.13 CyU3PCardCtx Struct Reference

Structure that holds properties of an SD/MMC peripheral.

```
#include <cyu3cardmgr.h>
```

### Data Fields

- uint8\_t cidRegData [CY\_U3P\_SD\_REG\_CID\_LEN]
- uint8\_t csdRegData [CY\_U3P\_SD\_REG\_CSD\_LEN]
- uint32\_t ocrRegData
- uint32\_t numBlks
- uint32\_t eraseSize
- uint32\_t dataTimeOut
- uint16\_t blkLen
- uint16\_t cardRCA
- uint16\_t ccc
- uint8\_t cardType
- uint8\_t removable
- uint8\_t writeable
- uint8\_t locked
- uint8\_t ddrMode
- uint8\_t clkRate
- uint8\_t opVoltage
- uint8\_t busWidth
- uint8\_t cardVer
- uint8\_t highCapacity
- uint8\_t uhslOpMode
- uint8\_t cardSpeed

### 4.13.1 Detailed Description

Structure that holds properties of an SD/MMC peripheral.

#### Description

This structure holds status and state information about an SD/MMC peripheral connected to the FX3S device.

### 4.13.2 Field Documentation

#### 4.13.2.1 uint16\_t blkLen

Current block size setting for the device.

**4.13.2.2 uint8\_t busWidth**

Current bus width setting for the device.

**4.13.2.3 uint16\_t cardRCA**

Relative card address

**4.13.2.4 uint8\_t cardSpeed**

Card speed information

**4.13.2.5 uint8\_t cardType**

Type of storage device connected on the S port. (Can be none if no device detected).

**4.13.2.6 uint8\_t cardVer**

Card version

**4.13.2.7 uint16\_t ccc**

Card command classes

**4.13.2.8 uint8\_t cidRegData[CY\_U3P\_SD\_REG\_CID\_LEN]**

CID Register data

**4.13.2.9 uint8\_t clkRate**

Current operating clock frequency for the device.

**4.13.2.10 uint8\_t csdRegData[CY\_U3P\_SD\_REG\_CSD\_LEN]**

CSD Register data

**4.13.2.11 uint32\_t dataTimeOut**

Timeout value for data transactions

**4.13.2.12 uint8\_t ddrMode**

Whether DDR clock mode is being used for this device.

**4.13.2.13 uint32\_t eraseSize**

The erase unit size in bytes for this device.

**4.13.2.14 uint8\_t highCapacity**

Indicates if the card is a high capacity card.

**4.13.2.15 uint8\_t locked**

Identifies whether the storage device is password locked.

**4.13.2.16 uint32\_t numBlks**

Current block size setting for the device.

**4.13.2.17 uint32\_t ocrRegData**

OCR Register data

**4.13.2.18 uint8\_t opVoltage**

Current operating voltage setting for the device.

**4.13.2.19 uint8\_t removable**

Indicates if the storage device is a removable device.

**4.13.2.20 uint8\_t uhsIOpMode**

SD 3.0 Operating modes

**4.13.2.21 uint8\_t writeable**

Whether the storage device is write enabled.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/[cyu3cardmgr.h](#)

## 4.14 CyU3PDebugLog\_t Struct Reference

FX3 debug logger data type.

```
#include <cyu3system.h>
```

### Data Fields

- `uint8_t priority`
- `uint8_t threadId`
- `uint16_t msg`
- `uint32_t param`

#### 4.14.1 Detailed Description

FX3 debug logger data type.

##### Description

The structure defines the header data type sent out by the debug logger function. For CyU3PDebugPrint function, the preamble will be the same with msg = 0xFFFF and param specifying the size of the message in bytes.

##### See also

[CyU3PDebugLog](#)  
[CyU3PDebugPrint](#)

#### 4.14.2 Field Documentation

##### 4.14.2.1 uint16\_t msg

Message Index. 0xFFFF in case of a CyU3PDebugPrint output.

##### 4.14.2.2 uint32\_t param

32 bit message parameter.

##### 4.14.2.3 uint8\_t priority

Priority of the message

##### 4.14.2.4 uint8\_t threadId

Id of thread sending the message.

The documentation for this struct was generated from the following file:

- [firmware/u3p\\_firmware/inc/cyu3system.h](#)

### 4.15 CyU3PDmaBuffer\_t Struct Reference

DMA buffer data structure.

```
#include <cyu3dma.h>
```

#### Data Fields

- `uint8_t * buffer`
- `uint16_t count`
- `uint16_t size`
- `uint16_t status`

#### 4.15.1 Detailed Description

DMA buffer data structure.

##### Description

The data structure is used to describe the status of a DMA buffer. It holds the address, size, valid data count, and

the status information. This type is used in DMA callbacks and APIs to identify the properties of the data buffer to be transferred.

See also

[CyU3PDmaCBInput\\_t](#)  
[CyU3PDmaChannelGetBuffer](#)  
[CyU3PDmaChannelSetupSendBuffer](#)  
[CyU3PDmaChannelSetupRecvBuffer](#)  
[CyU3PDmaChannelWaitForRecvBuffer](#)  
[CyU3PDmaMultiChannelGetBuffer](#)  
[CyU3PDmaMultiChannelSetupSendBuffer](#)  
[CyU3PDmaMultiChannelSetupRecvBuffer](#)  
[CyU3PDmaMultiChannelWaitForRecvBuffer](#)

## 4.15.2 Field Documentation

### 4.15.2.1 uint8\_t\* buffer

Pointer to the data buffer.

### 4.15.2.2 uint16\_t count

Byte count of valid data in buffer.

### 4.15.2.3 uint16\_t size

Actual size of the buffer in bytes. Should be a multiple of 16.

### 4.15.2.4 uint16\_t status

Buffer status. This is a four bit data field defined by CY\_U3P\_DMA\_BUFFER\_STATUS\_MASK. This holds information like whether the buffer is occupied, whether the buffer holds the end of packet and whether the buffer encountered a DMA error.

The documentation for this struct was generated from the following file:

- [firmware/u3p\\_firmware/inc/cyu3dma.h](#)

## 4.16 CyU3PDmaCBInput\_t Union Reference

DMA channel callback input.

```
#include <cyu3dma.h>
```

### Data Fields

- [CyU3PDmaBuffer\\_t buffer\\_p](#)

### 4.16.1 Detailed Description

DMA channel callback input.

### Description

This data structure is used to provide event specific information when a DMA callback is called. This structure is defined as a union to facilitate future updates to the DMA manager.

### See also

[CyU3PDmaBuffer\\_t](#)  
[CyU3PDmaCallback\\_t](#)

## 4.16.2 Field Documentation

### 4.16.2.1 CyU3PDmaBuffer\_t buffer\_p

Data about the DMA buffer that caused the callback to be called.

The documentation for this union was generated from the following file:

- firmware/u3p\_firmware/inc/[cyu3dma.h](#)

## 4.17 CyU3PDmaChannel Struct Reference

DMA Channel structure.

```
#include <cyu3dma.h>
```

### Data Fields

- uint32\_t [startSig](#)
- uint32\_t [state](#)
- uint16\_t [type](#)
- uint16\_t [size](#)
- uint16\_t [count](#)
- uint16\_t [prodAvailCount](#)
- uint16\_t [firstProdIndex](#)
- uint16\_t [firstConsIndex](#)
- uint16\_t [prodSckId](#)
- uint16\_t [consSckId](#)
- uint16\_t [overrideDscrIndex](#)
- uint16\_t [currentProdIndex](#)
- uint16\_t [currentConsIndex](#)
- uint16\_t [commitProdIndex](#)
- uint16\_t [commitConsIndex](#)
- uint16\_t [activeProdIndex](#)
- uint16\_t [activeConsIndex](#)
- uint16\_t [prodHeader](#)
- uint16\_t [prodFooter](#)
- uint16\_t [consHeader](#)
- uint16\_t [dmaMode](#)
- uint16\_t [prodSusp](#)
- uint16\_t [consSusp](#)
- uint16\_t [usbConsSusp](#)
- uint16\_t [discardCount](#)
- uint32\_t [notification](#)
- uint32\_t [xferSize](#)

- [CyBool\\_t isDmaHandleDCache](#)
- [CyU3PMutex lock](#)
- [CyU3PEvent flags](#)
- [CyU3PDmaCallback\\_t cb](#)
- [uint32\\_t endSig](#)

### 4.17.1 Detailed Description

DMA Channel structure.

#### Description

This structure keeps track of all the configuration and state information corresponding to a DMA channel. This structure is updated and maintained by the DMA driver and APIs. It is not recommended that the user application directly access any of this structure members.

#### See also

[CyU3PDmaChannelConfig\\_t](#)  
[CyU3PDmaChannelCreate](#)  
[CyU3PDmaChannelDestroy](#)  
[CyU3PDmaChannelUpdateMode](#)  
[CyU3PDmaChannelGetStatus](#)  
[CyU3PDmaChannelGetHandle](#)  
[CyU3PDmaChannelSetXfer](#)  
[CyU3PDmaChannelGetBuffer](#)  
[CyU3PDmaChannelCommitBuffer](#)  
[CyU3PDmaChannelDiscardBuffer](#)  
[CyU3PDmaChannelSetupSendBuffer](#)  
[CyU3PDmaChannelSetupRecvBuffer](#)  
[CyU3PDmaChannelWaitForCompletion](#)  
[CyU3PDmaChannelWaitForRecvBuffer](#)  
[CyU3PDmaChannelSetWrapUp](#)  
[CyU3PDmaChannelSetSuspend](#)  
[CyU3PDmaChannelResume](#)  
[CyU3PDmaChannelAbort](#)  
[CyU3PDmaChannelReset](#)  
[CyU3PDmaChannelCacheControl](#)

### 4.17.2 Field Documentation

#### 4.17.2.1 uint16\_t activeConsIndex

Active consumer descriptor.

#### 4.17.2.2 uint16\_t activeProdIndex

Active producer descriptor.

#### 4.17.2.3 CyU3PDmaCallback\_t cb

Callback function which gets invoked on DMA events

#### 4.17.2.4 uint16\_t commitConsIndex

Consumer descriptor for the buffer to be consumed.

**4.17.2.5 uint16\_t commitProdIndex**

Producer descriptor for the buffer to be consumed.

**4.17.2.6 uint16\_t consHeader**

The consumer socket header offset

**4.17.2.7 uint16\_t consSckId**

The consumer (egress) socket ID

**4.17.2.8 uint16\_t consSusp**

Consumer suspend option.

**4.17.2.9 uint16\_t count**

Number of buffers for the channel

**4.17.2.10 uint16\_t currentConsIndex**

Consumer descriptor for the latest buffer produced.

**4.17.2.11 uint16\_t currentProdIndex**

Producer descriptor for the latest buffer produced.

**4.17.2.12 uint16\_t discardCount**

Number of pending discards.

**4.17.2.13 uint16\_t dmaMode**

Mode of DMA operation

**4.17.2.14 uint32\_t endSig**

Channel structure end signature.

**4.17.2.15 uint16\_t firstConsIndex**

Head for the manual mode consumer descriptor chain list

**4.17.2.16 uint16\_t firstProdIndex**

Head for the normal descriptor chain list

**4.17.2.17 CyU3PEvent flags**

Event flags for the channel

**4.17.2.18 CyBool\_t isDmaHandleDCache**

Whether to do DMA cache handling for this channel.

**4.17.2.19 CyU3PMutex lock**

Lock for this channel structure.

**4.17.2.20 uint32\_t notification**

Registered notifications

**4.17.2.21 uint16\_t overrideDscrIndex**

Descriptor for override modes.

**4.17.2.22 uint16\_t prodAvailCount**

Minimum available buffers before producer is active.

**4.17.2.23 uint16\_t prodFooter**

The producer socket footer offset

**4.17.2.24 uint16\_t prodHeader**

The producer socket header offset

**4.17.2.25 uint16\_t prodSckId**

The producer (ingress) socket ID

**4.17.2.26 uint16\_t prodSusp**

Producer suspend option.

**4.17.2.27 uint16\_t size**

The buffer size associated with the channel

**4.17.2.28 uint32\_t startSig**

Channel structure start signature.

## 4.17.2.29 uint32\_t state

The current state of the DMA channel

## 4.17.2.30 uint16\_t type

The type of the DMA channel

## 4.17.2.31 uint16\_t usbConsSusp

USB consumer suspend mode for device work-around.

## 4.17.2.32 uint32\_t xferSize

Current transfer size

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/[cyu3dma.h](#)

## 4.18 CyU3PDmaChannelConfig\_t Struct Reference

DMA channel parameters.

```
#include <cyu3dma.h>
```

### Data Fields

- uint16\_t size
- uint16\_t count
- [CyU3PDmaSocketId\\_t](#) prodSckId
- [CyU3PDmaSocketId\\_t](#) consSckId
- uint16\_t prodAvailCount
- uint16\_t prodHeader
- uint16\_t prodFooter
- uint16\_t consHeader
- [CyU3PDmaMode\\_t](#) dmaMode
- uint32\_t notification
- [CyU3PDmaCallback\\_t](#) cb

### 4.18.1 Detailed Description

DMA channel parameters.

#### Description

This structure encapsulates the parameters that are provided at the time of DMA channel creation, and specifies the resources and events that are to be associated with the channel.

The size field specifies the size in bytes of each DMA buffer to be allocated for this DMA channel.

The offsets prodHeader, prodFooter and consHeader are used to do header addition and removal. These are valid only for manual channels and should be zero for auto channels.

The buffer address seen by the producer = (buffer + prodHeader).

The buffer size seen by the producer = (size - prodHeader - prodFooter).

The buffer address seen by the consumer = (buffer + consHeader).

The buffer size seen by the consumer = (buffer - consHeader).

For header addition to the buffer generated by the producer, the prodHeader should be the length of the header to be added and the other offsets should be zero. Once the buffer is generated, the header can be modified manually by the CPU and committed using the CyU3PDmaChannelCommitBuffer call.

For footer addition to the buffer generated by the producer, the prodFooter should be the length of the footer to be added and the other offsets should be zero. Once the buffer is generated, the footer can be added and committed using the CyU3PDmaChannelCommitBuffer call.

For header deletion from the buffer generated by the producer, the consHeader should be the length of the header to be removed and the other offsets should be zero. Once the buffer is generated, the buffer can be committed to the consumer socket with only change to the size of the data to be transmitted using the CommitBuffer call.

The size of the buffer as seen by the producer socket should always be a multiple of 16 bytes; ie, (size - prodHeader - prodFooter) must be a multiple of 16 bytes.

The prodAvailCount count should always be zero. This is used only for very specific use case where there should always be free buffers. Since there is no current use case for such a channel, this field should always be zero.

The count field specifies the number of buffers that should be allocated for this DMA channel. It is possible to obtain a large buffering depth by specifying a large count value, subject to availability of DMA buffer space and free descriptors.

#### See also

[CyU3PDmaChannelCreate](#)

## 4.18.2 Field Documentation

### 4.18.2.1 CyU3PDmaCallback\_t cb

Callback function which gets invoked on DMA events.

### 4.18.2.2 uint16\_t consHeader

The consumer socket header offset.

### 4.18.2.3 CyU3PDmaSocketId\_t consSckId

The consumer (egress) socket ID.

### 4.18.2.4 uint16\_t count

Number of buffers to be allocated for the channel. The count can be zero for MANUAL, MANUAL\_OUT and MAN↔UAL\_IN channels if the channel is intended to operate only in override mode and no buffer need to be allocated for the channel. The count cannot be zero for AUTO and AUTO\_SIGNAL channels.

### 4.18.2.5 CyU3PDmaMode\_t dmaMode

Mode of DMA operation.

### 4.18.2.6 uint32\_t notification

Registered notifications. This is a bit map based on CyU3PDmaCbType\_t. This defines the events for which the callback is triggered.

#### 4.18.2.7 uint16\_t prodAvailCount

Minimum available empty buffers before producer is active. By default, this should be zero. A non-zero value will activate this feature. The producer socket will not receive data into memory until the specified number of free buffers are available. This feature should be used only for very specific use cases where there is a requirement that there should always be free buffers during the transfer.

#### 4.18.2.8 uint16\_t prodFooter

The producer socket footer offset.

#### 4.18.2.9 uint16\_t prodHeader

The producer socket header offset.

#### 4.18.2.10 CyU3PDmaSocketId\_t prodSckId

The producer (ingress) socket ID.

#### 4.18.2.11 uint16\_t size

The buffer size associated with the channel.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/cyu3dma.h

## 4.19 CyU3PDmaDescriptor\_t Struct Reference

Descriptor data structure.

```
#include <cyu3descriptor.h>
```

### Data Fields

- uint8\_t \* **buffer**
- uint32\_t **sync**
- uint32\_t **chain**
- uint32\_t **size**

#### 4.19.1 Detailed Description

Descriptor data structure.

#### Description

This data structure contains the fields that make up a DMA descriptor on the FX3 device.

Each structure member is composed of multiple fields as shown below. Refer to the `sock_regs.h` header file for the definitions used.

**buffer:** (CY\_U3P\_BUFFER\_ADDR\_MASK)

**sync:** (CY\_U3P\_EN\_PROD\_INT | CY\_U3P\_EN\_PROD\_EVENT | CY\_U3P\_PROD\_IP\_MASK | CY\_U3P\_PROD\_SCK\_MASK | CY\_U3P\_EN\_CONS\_INT | CY\_U3P\_EN\_CONS\_EVENT | CY\_U3P\_CONS\_IP\_MASK | CY\_U3P\_CONS\_SCK\_MASK)

chain: (CY\_U3P\_WR\_NEXT\_DSCR\_MASK | CY\_U3P\_RD\_NEXT\_DSCR\_MASK)

size: (CY\_U3P\_BYTE\_COUNT\_MASK | CY\_U3P\_BUFFER\_SIZE\_MASK | CY\_U3P\_BUFFER\_OCCUPIED | CY\_U3P\_BUFFER\_ERROR | CY\_U3P\_EOP | CY\_U3P\_MARKER)

See also

[CyU3PDmaDscrGetConfig](#)  
[CyU3PDmaDscrSetConfig](#)

## 4.19.2 Field Documentation

### 4.19.2.1 uint8\_t\* buffer

Pointer to buffer used.

### 4.19.2.2 uint32\_t chain

Next descriptor links.

### 4.19.2.3 uint32\_t size

Current and maximum sizes of buffer.

### 4.19.2.4 uint32\_t sync

Consumer, Producer binding.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/[cyu3descriptor.h](#)

## 4.20 CyU3PDmaMultiChannel Struct Reference

DMA multi-channel structure.

```
#include <cyu3dma.h>
```

### Data Fields

- uint32\_t startSig
- uint32\_t state
- uint16\_t type
- uint16\_t size
- uint16\_t count
- uint16\_t validSckCount
- uint16\_t consDisabled [CY\_U3P\_DMA\_MAX\_MULTI\_SCK\_COUNT]
- uint16\_t firstProdIndex [CY\_U3P\_DMA\_MAX\_MULTI\_SCK\_COUNT]
- uint16\_t firstConsIndex [CY\_U3P\_DMA\_MAX\_MULTI\_SCK\_COUNT]
- uint16\_t prodSckId [CY\_U3P\_DMA\_MAX\_MULTI\_SCK\_COUNT]
- uint16\_t consSckId [CY\_U3P\_DMA\_MAX\_MULTI\_SCK\_COUNT]

- `uint16_t overrideDscrIndex`
- `uint16_t currentProdIndex`
- `uint16_t currentConsIndex`
- `uint16_t commitProdIndex`
- `uint16_t commitConsIndex`
- `uint16_t activeProdIndex [CY_U3P_DMA_MAX_MULTI_SCK_COUNT]`
- `uint16_t activeConsIndex [CY_U3P_DMA_MAX_MULTI_SCK_COUNT]`
- `uint16_t prodHeader`
- `uint16_t prodFooter`
- `uint16_t consHeader`
- `uint16_t prodAvailCount`
- `uint16_t dmaMode`
- `uint16_t prodSusp`
- `uint16_t consSusp`
- `uint16_t usbConsSusp`
- `uint16_t bufferCount [CY_U3P_DMA_MAX_MULTI_SCK_COUNT]`
- `uint16_t discardCount [CY_U3P_DMA_MAX_MULTI_SCK_COUNT]`
- `uint32_t notification`
- `uint32_t xferSize`
- `CyBool_t isDmaHandleDCache`
- `CyU3PMutex lock`
- `CyU3PEvent flags`
- `CyU3PDmaMultiCallback_t cb`
- `uint32_t endSig`

#### 4.20.1 Detailed Description

DMA multi-channel structure.

##### Description

This structure holds all configuration and state information about the corresponding DMA multi-channel. This structure is updated and maintained by the DMA driver and APIs, and it is recommended that user code does not access any of the structure members directly.

##### See also

- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelConfig\\_t](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelGetHandle](#)
- [CyU3PDmaMultiChannelUpdateMode](#)
- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelCommitBuffer](#)
- [CyU3PDmaMultiChannelDiscardBuffer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)
- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelSetWrapUp](#)
- [CyU3PDmaMultiChannelSetSuspend](#)
- [CyU3PDmaMultiChannelResume](#)
- [CyU3PDmaMultiChannelAbort](#)
- [CyU3PDmaMultiChannelReset](#)
- [CyU3PDmaMultiChannelGetStatus](#)
- [CyU3PDmaMultiChannelCacheControl](#)

## 4.20.2 Field Documentation

4.20.2.1 `uint16_t activeConsIndex[CY_U3P_DMA_MAX_MULTI_SCK_COUNT]`

Active consumer descriptor.

4.20.2.2 `uint16_t activeProdIndex[CY_U3P_DMA_MAX_MULTI_SCK_COUNT]`

Active producer descriptor.

4.20.2.3 `uint16_t bufferCount[CY_U3P_DMA_MAX_MULTI_SCK_COUNT]`

Number of active buffers available for consumer.

4.20.2.4 `CyU3PDmaMultiCallback_t cb`

Callback function which gets invoked on DMA events

4.20.2.5 `uint16_t commitConsIndex`

Consumer descriptor for the buffer to be consumed.

4.20.2.6 `uint16_t commitProdIndex`

Producer descriptor for the buffer to be consumed.

4.20.2.7 `uint16_t consDisabled[CY_U3P_DMA_MAX_MULTI_SCK_COUNT]`

Whether each consumer socket is disabled. Applies only to multicast channels.

4.20.2.8 `uint16_t consHeader`

The consumer socket header offset

4.20.2.9 `uint16_t consSckId[CY_U3P_DMA_MAX_MULTI_SCK_COUNT]`

The consumer (egress) socket ID

4.20.2.10 `uint16_t consSusp`

Consumer suspend option.

4.20.2.11 `uint16_t count`

Number of buffers for the channel

4.20.2.12 `uint16_t currentConsIndex`

Consumer descriptor for the latest buffer produced.

**4.20.2.13 uint16\_t currentProdIndex**

Producer descriptor for the latest buffer produced.

**4.20.2.14 uint16\_t discardCount[CY\_U3P\_DMA\_MAX\_MULTI\_SCK\_COUNT]**

Number of buffers to be discarded.

**4.20.2.15 uint16\_t dmaMode**

Mode of DMA operation

**4.20.2.16 uint32\_t endSig**

Channel structure end signature.

**4.20.2.17 uint16\_t firstConsIndex[CY\_U3P\_DMA\_MAX\_MULTI\_SCK\_COUNT]**

Head for the manual consumer descriptor chain

**4.20.2.18 uint16\_t firstProdIndex[CY\_U3P\_DMA\_MAX\_MULTI\_SCK\_COUNT]**

Head for the normal descriptor chain

**4.20.2.19 CyU3PEvent flags**

Event flags for the channel

**4.20.2.20 CyBool\_t isDmaHandleDCache**

Whether to do internal DMA cache handling.

**4.20.2.21 CyU3PMutex lock**

Lock for the channel structure

**4.20.2.22 uint32\_t notification**

Registered notifications.

**4.20.2.23 uint16\_t overrideDscrIndex**

Descriptor for override modes.

**4.20.2.24 uint16\_t prodAvailCount**

Minimum available buffers before producer is active.

4.20.2.25 uint16\_t prodFooter

The producer socket footer offset

4.20.2.26 uint16\_t prodHeader

The producer socket header offset

4.20.2.27 uint16\_t prodSckId[CY\_U3P\_DMA\_MAX\_MULTI\_SCK\_COUNT]

The producer (ingress) socket ID

4.20.2.28 uint16\_t prodSusp

Producer suspend option.

4.20.2.29 uint16\_t size

The buffer size associated with the channel

4.20.2.30 uint32\_t startSig

Channel structure start signature.

4.20.2.31 uint32\_t state

The current state of the DMA channel

4.20.2.32 uint16\_t type

The type of the DMA channel

4.20.2.33 uint16\_t usbConsSusp

USB consumer suspend mode for device work-around.

4.20.2.34 uint16\_t validSckCount

Number of sockets in the multi-socket operation.

4.20.2.35 uint32\_t xferSize

Current xfer size

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/[cyu3dma.h](#)

## 4.21 CyU3PDmaMultiChannelConfig\_t Struct Reference

DMA multi-channel parameter structure.

```
#include <cyu3dma.h>
```

### Data Fields

- `uint16_t size`
- `uint16_t count`
- `uint16_t validSckCount`
- `CyU3PDmaSocketId_t prodSckId [CY_U3P_DMA_MAX_MULTI_SCK_COUNT]`
- `CyU3PDmaSocketId_t consSckId [CY_U3P_DMA_MAX_MULTI_SCK_COUNT]`
- `uint16_t prodAvailCount`
- `uint16_t prodHeader`
- `uint16_t prodFooter`
- `uint16_t consHeader`
- `CyU3PDmaMode_t dmaMode`
- `uint32_t notification`
- `CyU3PDmaMultiCallback_t cb`

### 4.21.1 Detailed Description

DMA multi-channel parameter structure.

#### Description

This structure encapsulates all the parameters required to create a DMA multi-channel.

In the case of many to one channels, there shall be 'validSckCount' number of producer sockets and only one consumer socket. The producer sockets needs to be updated in the required order of operation. The first buffer shall be taken from the `prodSckId[0]`, second from `prodSckId[1]` and so on. If only two producer sockets are used, then only `prodSckId[0]`, `prodSckId[1]` and `consSckId[0]` shall be considered.

In the case of one to many operations, there shall be only one producer socket and 'validSckCount' number of consumer sockets.

The size field is the total buffer that needs to be allocated for DMA operations. This field has restrictions for DMA operations.

The size, count, prodHeader, prodFooter, consHeader and prodAvailCount fields are used in the same way as in the [CyU3PDmaChannelConfig\\_t](#) structure.

#### See also

[CyU3PDmaChannelConfig\\_t](#)  
[CyU3PDmaMultiChannelCreate](#)

### 4.21.2 Field Documentation

#### 4.21.2.1 CyU3PDmaMultiCallback\_t cb

Callback function which gets invoked on multi socket DMA events.

#### 4.21.2.2 uint16\_t consHeader

The consumer socket header offset.

**4.21.2.3 CyU3PDmaSocketId\_t consSckId[CY\_U3P\_DMA\_MAX\_MULTI\_SCK\_COUNT]**

The consumer (egress) socket ID.

**4.21.2.4 uint16\_t count**

Number of buffers to be allocated for each socket of the channel. For one to many and many to one channels, there will be twice the number of buffers as specified in the count and for multicast it will have the same number of buffers as specified in count. The count cannot be zero.

**4.21.2.5 CyU3PDmaMode\_t dmaMode**

Mode of DMA operation

**4.21.2.6 uint32\_t notification**

Registered notifications. This is a bit map based on CyU3PDmaCbType\_t. This defines the events for which the callback is triggered.

**4.21.2.7 uint16\_t prodAvailCount**

Minimum available empty buffers before producer is active. By default, this should be zero. A non-zero value will activate this feature. The producer socket will not receive data into memory until the specified number of free buffers are available. This feature should be used only for very specific use cases where there is a requirement that there should always be free buffers during the transfer.

**4.21.2.8 uint16\_t prodFooter**

The producer socket footer offset.

**4.21.2.9 uint16\_t prodHeader**

The producer socket header offset.

**4.21.2.10 CyU3PDmaSocketId\_t prodSckId[CY\_U3P\_DMA\_MAX\_MULTI\_SCK\_COUNT]**

The producer (ingress) socket ID.

**4.21.2.11 uint16\_t size**

The buffer size associated with the channel.

**4.21.2.12 uint16\_t validSckCount**

Number of sockets in the multi-socket operation.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/cyu3dma.h

## 4.22 CyU3PDmaSocket\_t Struct Reference

DMA socket register structure.

```
#include <cyu3socket.h>
```

### Data Fields

- `uvint32_t dscrChain`
- `uvint32_t xferSize`
- `uvint32_t xferCount`
- `uvint32_t status`
- `uvint32_t intr`
- `uvint32_t intrMask`
- `uvint32_t unused2 [2]`
- `CyU3PDmaDescriptor_t activeDscr`
- `uvint32_t unused19 [19]`
- `uvint32_t sckEvent`

### 4.22.1 Detailed Description

DMA socket register structure.

#### Description

Each hardware block on the FX3 device implements a number of DMA sockets through which it handles data transfers with the external world. Each DMA socket serves as an endpoint for an independent data stream going through the hardware block.

Each socket has a set of registers associated with it, that reflect the configuration and status information for that socket. The CyU3PDmaSocket structure is a replica of the config/status registers for a socket and is designed to perform socket configuration and status checks directly from firmware.

See the `sock_regs.h` header file for the definitions of the fields that make up each of these registers.

#### See also

[CyU3PDmaSocketConfig\\_t](#)

### 4.22.2 Field Documentation

#### 4.22.2.1 CyU3PDmaDescriptor\_t activeDscr

Active descriptor information. See [cyu3descriptor.h](#) for definition.

#### 4.22.2.2 uvint32\_t dscrChain

The descriptor chain associated with the socket

#### 4.22.2.3 uvint32\_t intr

Interrupt status register.

#### 4.22.2.4 uvint32\_t intrMask

Interrupt mask register.

**4.22.2.5 uvint32\_t sckEvent**

Generate event register.

**4.22.2.6 uvint32\_t status**

Socket configuration and status register.

**4.22.2.7 uvint32\_t unused19[19]**

Reserved register space.

**4.22.2.8 uvint32\_t unused2[2]**

Reserved register space.

**4.22.2.9 uvint32\_t xferCount**

The completed transfer count for this socket.

**4.22.2.10 uvint32\_t xferSize**

The transfer size requested for this socket. The size can be specified in bytes or in terms of number of buffers, depending on the UNIT field in the status value.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/cyu3socket.h

## 4.23 CyU3PDmaSocketConfig\_t Struct Reference

DMA socket configuration structure.

```
#include <cyu3socket.h>
```

### Data Fields

- uint32\_t [dscrChain](#)
- uint32\_t [xferSize](#)
- uint32\_t [xferCount](#)
- uint32\_t [status](#)
- uint32\_t [intr](#)
- uint32\_t [intrMask](#)

### 4.23.1 Detailed Description

DMA socket configuration structure.

#### Description

This structure holds all the configuration fields that can be directly updated on a DMA socket. Refer to the `sock_regs.h` file for the detailed break up into bit-fields for each of the structure members.

**See also**

[CyU3PDmaSocket\\_t](#)  
[CyU3PDmaSocketSetConfig](#)  
[CyU3PDmaSocketGetConfig](#)

## 4.23.2 Field Documentation

### 4.23.2.1 uint32\_t dscrChain

The descriptor chain associated with the socket.

### 4.23.2.2 uint32\_t intr

Interrupt status.

### 4.23.2.3 uint32\_t intrMask

Interrupt mask.

### 4.23.2.4 uint32\_t status

Socket status register.

### 4.23.2.5 uint32\_t xferCount

Transfer status for the socket.

### 4.23.2.6 uint32\_t xferSize

Transfer size for the socket.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/[cyu3socket.h](#)

## 4.24 CyU3PEpConfig\_t Struct Reference

Endpoint configuration information.

```
#include <cyu3usb.h>
```

### Data Fields

- [CyBool\\_t enable](#)
- [CyU3PUsbEpType\\_t epType](#)
- [uint16\\_t streams](#)
- [uint16\\_t pcktSize](#)
- [uint8\\_t burstLen](#)
- [uint8\\_t isoPkts](#)

#### 4.24.1 Detailed Description

Endpoint configuration information.

##### Description

This structure holds all the properties of a USB endpoint. This structure is used to provide information about the desired configuration of various endpoints in the system.

##### See also

[CyU3PSetEpConfig](#)

#### 4.24.2 Field Documentation

##### 4.24.2.1 uint8\_t burstLen

Maximum burst length in packets. This needs to be specified as the number of packets per burst and not as "burst length - 1" as in the case of the Super Speed Companion descriptor.

##### 4.24.2.2 CyBool\_t enable

Endpoint status - enabled or not.

##### 4.24.2.3 CyU3PUsbEpType\_t epType

The endpoint type - Isochronous = 1, Bulk = 2, Interrupt = 3. See USB specification for type values.

##### 4.24.2.4 uint8\_t isoPkts

Number of packets per micro-frame for ISO endpoints.

##### 4.24.2.5 uint16\_t pcktSize

Maximum packet size for the endpoint. Can be set to 0 if the maximum packet size should be set to the maximum value consistent with the USB specification.

##### 4.24.2.6 uint16\_t streams

Number of bulk streams used by the endpoint. This needs to be specified as the number of streams and not as "stream count - 1" as in the case of the Super Speed Companion descriptor.

The documentation for this struct was generated from the following file:

- [firmware/u3p\\_firmware/inc/cyu3usb.h](#)

## 4.25 CyU3PGpifConfig\_t Struct Reference

Structure that holds all configuration inputs for the GPIF hardware.

```
#include <cyfx3pib.h>
```

## Data Fields

- const uint16\_t `stateCount`
- const `CyU3PGpifWaveData` \* `stateData`
- const uint8\_t \* `statePosition`
- const uint16\_t `functionCount`
- const uint16\_t \* `functionData`
- const uint16\_t `regCount`
- const uint32\_t \* `regData`

### 4.25.1 Detailed Description

Structure that holds all configuration inputs for the GPIF hardware.

#### Description

The GPIF block on the FX3 device has a set of general configuration registers, transition function registers and state descriptors that need to be initialized to make the GPIF state machine functional. This structure encapsulates all the data that is required to program the GPIF block to load a user defined state machine.

The GPIF configuration data in the form of this structure is commonly generated by the GPIF II Designer tool.

#### See also

[CyU3PGpifWaveData](#)  
[CyFx3BootGpifLoad](#)

#### Description

The GPIF block on the FX3 device has a set of general configuration registers, transition function registers and state descriptors that need to be initialized to make the GPIF state machine functional. This structure encapsulates all the data that is required to program the GPIF block to load a user defined state machine.

#### See also

[CyU3PGpifLoad](#)  
[CyU3PGpifWaveData](#)

### 4.25.2 Field Documentation

#### 4.25.2.1 const uint16\_t `functionCount`

Number of transition functions to be initialized.

#### 4.25.2.2 const uint16\_t \* `functionData`

Pointer to array containing transition function data.

#### 4.25.2.3 const uint16\_t `regCount`

Number of GPIF config registers to be initialized.

#### 4.25.2.4 const uint32\_t \* `regData`

Pointer to array containing GPIF register values.

#### 4.25.2.5 const uint16\_t stateCount

Number of states to be initialized.

#### 4.25.2.6 const CyU3PGpifWaveData \* stateData

Pointer to array containing state descriptors.

#### 4.25.2.7 const uint8\_t \* statePosition

Pointer to array index -> state number mapping.

The documentation for this struct was generated from the following files:

- firmware/boot\_fw/include/cyfx3pib.h
- firmware/u3p\_firmware/inc/cyu3gpif.h

## 4.26 CyU3PGpifWaveData Struct Reference

Information on a single GPIF transition from one state to another.

```
#include <cyfx3pib.h>
```

### Data Fields

- uint32\_t [leftData](#) [3]
- uint32\_t [rightData](#) [3]

### 4.26.1 Detailed Description

Information on a single GPIF transition from one state to another.

#### Description

The GPIF state machine on the FX3 device is defined through a set of transition descriptors. These descriptors include fields for specifying the next state, the conditions for transition, and the output values.

This structure encapsulates all of the information that forms the left and right transition descriptors for a state.

#### See also

[CyU3PGpifConfig\\_t](#)

#### Description

The GPIF state machine on the FX3 device is defined through a set of transition descriptors. These descriptors include fields for specifying the next state, the conditions for transition, and the output values.

This structure encapsulates all of the information that forms the left and right transition descriptors for a state.

#### See also

[CyU3PGpifWaveformLoad](#)

### 4.26.2 Field Documentation

#### 4.26.2.1 uint32\_t leftData

12 byte left transition descriptor.

#### 4.26.2.2 uint32\_t rightData

12 byte right transition descriptor.

The documentation for this struct was generated from the following files:

- firmware/boot-fw/include/cyfx3pib.h
- firmware/u3p\_firmware/inc/cyu3gpif.h

## 4.27 CyU3PGpioClock\_t Struct Reference

Clock configuration information for the GPIO block.

```
#include <cyu3lpp.h>
```

### Data Fields

- uint8\_t fastClkDiv
- uint8\_t slowClkDiv
- [CyBool\\_t](#) halfDiv
- [CyU3PGpioSimpleClkDiv\\_t](#) simpleDiv
- [CyU3PSysClockSrc\\_t](#) clkSrc

### 4.27.1 Detailed Description

Clock configuration information for the GPIO block.

#### Description

The GPIO block on the FX3 makes use of three different clocks. The master (fast) clock for this block is directly divided down from the SYS\_CLK. The block also uses a slow clock which can be derived from this fast clock. The complex GPIOs on FX3 can be clocked on either the fast or the slow clock.

The simple GPIOs are clocked by another clock which is separately derived from the fast clock.

This structure encapsulates all of the clock parameters for the GPIO clock.

#### See also

- [CyU3PGpioSetClock](#)
- [CyU3PSysClockSrc\\_t](#)
- [CyU3PGpioSimpleClkDiv\\_t](#)

### 4.27.2 Field Documentation

#### 4.27.2.1 CyU3PSysClockSrc\_t clkSrc

The clock source to be used for this peripheral.

#### 4.27.2.2 uint8\_t fastClkDiv

Divider value for the GPIO fast clock. The min value is 2 and max value is 16.

#### 4.27.2.3 CyBool\_t halfDiv

This allows the fast clock to be divided by a non integral value. If set to true, this adds 0.5 to the fastClkDiv value. This should be used only if the slow clock is disabled.

#### 4.27.2.4 CyU3PGpioSimpleClkDiv\_t simpleDiv

Divider value from fast clock for sampling simple GPIOs.

#### 4.27.2.5 uint8\_t slowClkDiv

Divider value for the GPIO slow clock. This divisor applies on top of the fast clock and can be set to zero, to disable the slow clock. The min value is 0 (1 is not supported) and max value is 64.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/cyu3lpp.h

## 4.28 CyU3PGpioComplexConfig\_t Struct Reference

Configuration information for complex GPIO pins.

```
#include <cyu3gpio.h>
```

### Data Fields

- CyBool\_t outValue
- CyBool\_t driveLowEn
- CyBool\_t driveHighEn
- CyBool\_t inputEn
- CyU3PGpioComplexMode\_t pinMode
- CyU3PGpioIntrMode\_t intrMode
- CyU3PGpioTimerMode\_t timerMode
- uint32\_t timer
- uint32\_t period
- uint32\_t threshold

### 4.28.1 Detailed Description

Configuration information for complex GPIO pins.

#### Description

Complex GPIOs on FX3 can be configured to behave in different ways. All of the parameters that are used for configuring simple GPIOs are also applicable for complex GPIOs.

In addition to these parameters, the pinMode parameter is used to specify the specific complex GPIO functionality desired. The timerMode, timer, period and threshold parameters are used to configure the complex GPIO timer for this pin; in cases where the timer is required.

#### See also

- [CyU3PGpioSetComplexConfig](#)
- [CyU3PGpioSimpleConfig\\_t](#)
- [CyU3PGpioComplexMode\\_t](#)
- [CyU3PGpioTimerMode\\_t](#)
- [CyU3PGpioIntrMode\\_t](#)

## 4.28.2 Field Documentation

### 4.28.2.1 CyBool\_t driveHighEn

When set true, the output driver is enabled for outValue = [CyTrue\(1\)](#), otherwise tristated.

### 4.28.2.2 CyBool\_t driveLowEn

When set true, the output driver is enabled for outValue = [CyFalse\(0\)](#), otherwise tristated.

### 4.28.2.3 CyBool\_t inputEn

When set true, the input state is enabled.

### 4.28.2.4 CyU3PGpioIntrMode\_t intrMode

Interrupt mode for the GPIO.

### 4.28.2.5 CyBool\_t outValue

Initial value for the GPIO if configured as output: CyFalse = 0, CyTrue = 1.

### 4.28.2.6 uint32\_t period

Timer period.

### 4.28.2.7 CyU3PGpioComplexMode\_t pinMode

Complex GPIO operating mode.

### 4.28.2.8 uint32\_t threshold

Timer threshold value.

### 4.28.2.9 uint32\_t timer

Timer initial value.

### 4.28.2.10 CyU3PGpioTimerMode\_t timerMode

Timer mode.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/[cyu3gpio.h](#)

## 4.29 CyU3PGpioSimpleConfig\_t Struct Reference

Configuration information for simple GPIOs.

```
#include <cyu3gpio.h>
```

## Data Fields

- [CyBool\\_t outValue](#)
- [CyBool\\_t driveLowEn](#)
- [CyBool\\_t driveHighEn](#)
- [CyBool\\_t inputEn](#)
- [CyU3PGpioIntrMode\\_t intrMode](#)

### 4.29.1 Detailed Description

Configuration information for simple GPIOs.

#### Description

Simple GPIOs on the FX3 have configurable properties such as I/O direction, output value and interrupt mode. This structure holds all of the information required to configure a simple GPIO on the FX3 device.

The input and output stages are configured separately. Care should be taken that the output stage is only turned on where desired, so that external devices are not damaged.

For use as a normal output pin, both `driveLowEn` and `driveHighEn` need to be `CyTrue` and `inputEn` needs to be `CyFalse`. Similarly for use as a normal input pin, `inputEn` must be `CyTrue` and both `driveLowEn` and `driveHighEn` should be `CyFalse`.

When output stage is enabled, the `outValue` field contains the initial state of the pin. `CyTrue` means high and `CyFalse` means low.

#### See also

- [CyU3PGpioIntrMode\\_t](#)
- [CyU3PGpioSetSimpleConfig](#)

### 4.29.2 Field Documentation

#### 4.29.2.1 CyBool\_t driveHighEn

When set true, the output driver is enabled for `outValue` = `CyTrue`, otherwise tristated.

#### 4.29.2.2 CyBool\_t driveLowEn

When set true, the output driver is enabled for `outValue` = `CyFalse`, otherwise tristated.

#### 4.29.2.3 CyBool\_t inputEn

When set true, the input stage on the pin is enabled.

#### 4.29.2.4 CyU3PGpioIntrMode\_t intrMode

Interrupt mode for the GPIO.

#### 4.29.2.5 CyBool\_t outValue

Initial output on the GPIO if configured as output: `CyFalse` = 0, `CyTrue` = 1.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/[cyu3gpio.h](#)

## 4.30 CyU3PI2cConfig\_t Struct Reference

Structure defining the I2C interface configuration.

```
#include <cyu3i2c.h>
```

### Data Fields

- `uint32_t bitRate`
- `CyBool_t isDma`
- `uint32_t busTimeout`
- `uint16_t dmaTimeout`

#### 4.30.1 Detailed Description

Structure defining the I2C interface configuration.

##### Description

This structure encapsulates all of the configurable parameters that can be selected for the I2C interface. The [CyU3PI2cSetConfig\(\)](#) function accepts a pointer to this structure, and updates all of the interface parameters.

The I2C block can function in the bit rate range of 100 KHz to 1MHz. In the default mode of operation, the timeouts need to be kept disabled.

In the register mode of operation (isDma is false), the data transfer APIs are blocking and return only after the requested amount of data has been read or written. In such a case, the I2C specific callbacks are meaningless; and the CyU3PI2cSetConfig API expects that no callback is specified when the register mode is selected.

##### See also

[CyU3PI2cSetConfig](#)

#### 4.30.2 Field Documentation

##### 4.30.2.1 `uint32_t bitRate`

Bit rate for the interface. (Eg: 100000 for 100KHz)

##### 4.30.2.2 `uint32_t busTimeout`

Number of core clocks SCK can be held low by the slave byte transmission before triggering a timeout error. 0xF←FFFFFFFFFF means no timeout.

##### 4.30.2.3 `uint16_t dmaTimeout`

Number of core clocks DMA can remain not ready before flagging an error. 0xFFFF means no timeout.

##### 4.30.2.4 `CyBool_t isDma`

CyFalse: Register transfer mode, CyTrue: DMA transfer mode

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/[cyu3i2c.h](#)

## 4.31 CyU3PI2cPreamble\_t Struct Reference

Structure defining the preamble to be sent on the I2C interface.

```
#include <cyu3i2c.h>
```

### Data Fields

- `uint8_t buffer [8]`
- `uint8_t length`
- `uint16_t ctrlMask`

#### 4.31.1 Detailed Description

Structure defining the preamble to be sent on the I2C interface.

##### Description

All I2C data transfers start with a preamble where the first byte contains the slave address and the direction of transfer. The preamble can optionally contain other bytes where device specific address values or other commands are sent to the slave device.

The FX3 device supports associating a preamble with a maximum length of 8 bytes to any I2C data transfer. This allows the user to specify a multi-byte preamble which covers the slave address, device specific address fields and then initiate the data transfer.

The `ctrlMask` indicate the start / stop bit conditions after each byte of the preamble.

For example, an I2C EEPROM read requires the byte address for the read to be written first. These two I2C operations can be combined into one I2C API call using the parameters of the structure.

Typical I2C EEPROM page Read operation:

Byte 0:

Bit 7 - 1: Slave address.

Bit 0 : 0 - Indicating this is a write from master.

Byte 1, 2: Address to which the data has to be written.

Byte 3:

Bit 7 - 1: Slave address.

Bit 0 : 1 - Indicating this is a read operation.

The buffer field shall hold the above four bytes, the length field shall be four; and `ctrlMask` field will be 0x0004 as a start bit is required after the third byte (third bit is set).

Typical I2C EEPROM page Write operation:

Byte 0:

Bit 7 - 1: Slave address.

Bit 0 : 0 - Indicating this is a write from master.

Byte 1, 2: Address to which the data has to be written.

The buffer field shall hold the above three bytes, the length field shall be three, and the `ctrlMask` field is zero as no additional start/stop conditions are needed.

Please note that the user is expected to set the direction bit in the preamble bytes properly based on the type of transfer to be performed. The API does not check whether the preamble direction matches the type of transfer API being called.

##### See also

[CyU3PI2cTransmitBytes](#)  
[CyU3PI2cReceiveBytes](#)

#### 4.31.2 Field Documentation

#### 4.31.2.1 uint8\_t buffer[8]

The actual preamble bytes starting with slave address.

#### 4.31.2.2 uint16\_t ctrlMask

This field controls the start stop condition after every byte of preamble data. Bits 0 - 7 represent a bit mask for the start condition and Bits 8 - 15 represent a bit mask for stop condition. If both start and stop bits are set at one index; the stop condition takes priority.

#### 4.31.2.3 uint8\_t length

The length of the preamble to be sent. Should be between 1 and 8.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/cyu3i2c.h

## 4.32 CyU3PI2sConfig\_t Struct Reference

I2S interface configuration parameters.

```
#include <cyu3i2s.h>
```

### Data Fields

- CyBool\_t isMono
- CyBool\_t isLsbFirst
- CyBool\_t isDma
- CyU3PI2sPadMode\_t padMode
- CyU3PI2sSampleRate\_t sampleRate
- CyU3PI2sSampleWidth\_t sampleWidth

### 4.32.1 Detailed Description

I2S interface configuration parameters.

#### Description

This structure encapsulates all of the configurable parameters that can be selected for the I2S interface. The [CyU3PI2sSetConfig\(\)](#) function accepts a pointer to this structure, and updates all of the interface parameters.

#### See also

[CyU3PI2sSetConfig](#)

### 4.32.2 Field Documentation

#### 4.32.2.1 CyBool\_t isDma

CyTrue: DMA mode, CyFalse: Register mode

#### 4.32.2.2 CyBool\_t isLsbFirst

CyTrue: LSB First, CyFalse: MSB First

#### 4.32.2.3 CyBool\_t isMono

CyTrue: Mono, CyFalse: Stereo

#### 4.32.2.4 CyU3PI2sPadMode\_t padMode

Type of padding to be used

#### 4.32.2.5 CyU3PI2sSampleRate\_t sampleRate

Sample rate for this audio stream.

#### 4.32.2.6 CyU3PI2sSampleWidth\_t sampleWidth

Bit width for samples in this audio stream.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/cyu3i2s.h

## 4.33 CyU3PloMatrixConfig\_t Struct Reference

IO matrix configuration parameters.

```
#include <cyfx3_api.h>
```

### Data Fields

- [CyBool\\_t isDQ32Bit](#)
- [CyBool\\_t useUart](#)
- [CyBool\\_t useI2C](#)
- [CyBool\\_t useI2S](#)
- [CyBool\\_t useSpi](#)
- [CyU3PSPortMode\\_t s0Mode](#)
- [CyU3PSPortMode\\_t s1Mode](#)
- [CyU3PloMatrixLppMode\\_t lppMode](#)
- [uint32\\_t gpioSimpleEn \[2\]](#)
- [uint32\\_t gpioComplexEn \[2\]](#)

### 4.33.1 Detailed Description

IO matrix configuration parameters.

#### Description

The EZ-USB FX3 and FX3S devices have a flexible IO architecture that allows each IO Pin to serve multiple functions. The desired IO configuration for all of the pins needs to be specified before any of the FX3 internal blocks such as USB, GPIF or UART are powered on.

This structure captures the desired IO configuration for the FX3/FX3S device as a whole.

#### Note

A common structure including the storage port configuration is used for both FX3 and FX3S devices, in order to maintain a common API interface. The s0Mode and s1Mode fields should be set to CY\_U3P\_SPORT\_INACTIVE when using the EZ-USB FX3 device.

**See also**

[CyU3PloMatrixLppMode\\_t](#)  
[CyU3PSPortMode\\_t](#)  
[CyU3PDeviceConfigureIOMatrix](#)

### 4.33.2 Field Documentation

#### 4.33.2.1 `uint32_t gpioComplexEn[2]`

Bitmap that identifies pins that should be configured as complex GPIOs.

#### 4.33.2.2 `uint32_t gpioSimpleEn[2]`

Bitmap that identifies pins that should be configured as simple GPIOs.

#### 4.33.2.3 `CyBool_t isDQ32Bit`

Whether the GPIF data bus is 32 bits wide.

#### 4.33.2.4 `CyU3PloMatrixLppMode_t lppMode`

LPP IO configuration to be used.

#### 4.33.2.5 `CyU3PSPortMode_t s0Mode`

Interface mode for the S0 storage port (where available).

#### 4.33.2.6 `CyU3PSPortMode_t s1Mode`

Interface mode for the S1 storage port (where available).

#### 4.33.2.7 `CyBool_t useI2C`

Whether pins are to be reserved for the I2C interface.

#### 4.33.2.8 `CyBool_t useI2S`

Whether pins are to be reserved for the I2S interface.

#### 4.33.2.9 `CyBool_t useSpi`

Whether pins are to be reserved for the SPI interface.

#### 4.33.2.10 `CyBool_t useUart`

Whether pins are to be reserved for the UART interface.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/[cyfx3\\_api.h](#)

## 4.34 CyU3PMbox Struct Reference

Structure that holds a packet of mailbox data.

```
#include <cyu3mbox.h>
```

### Data Fields

- `uint32_t w0`
- `uint32_t w1`

#### 4.34.1 Detailed Description

Structure that holds a packet of mailbox data.

##### Description

The FX3 device has 8 byte mailbox registers that can be used when the P-port mode is enabled. This structure represents the eight bytes to be written to or read from the corresponding mailbox registers.

#### 4.34.2 Field Documentation

##### 4.34.2.1 `uint32_t w0`

Contents of the lower mailbox register.

##### 4.34.2.2 `uint32_t w1`

Contents of the upper mailbox register.

The documentation for this struct was generated from the following file:

- `firmware/u3p_firmware/inc/cyu3mbox.h`

## 4.35 CyU3PMipicsiCfg\_t Struct Reference

Structure defining the MIPI-CSI block interface configuration.

```
#include <cyu3mipicsi.h>
```

### Data Fields

- `CyU3PMipicsiDataFormat_t dataFormat`
- `uint8_t numDataLanes`
- `uint8_t pllPrd`
- `uint16_t pllFbd`
- `CyU3PMipicsiPllClkFrs_t pllFrs`
- `CyU3PMipicsiPllClkDiv_t csiRxClkDiv`
- `CyU3PMipicsiPllClkDiv_t parClkDiv`
- `uint16_t mClkCtl`
- `CyU3PMipicsiPllClkDiv_t mClkRefDiv`
- `uint16_t hResolution`
- `uint16_t fifoDelay`

### 4.35.1 Detailed Description

Structure defining the MIPI-CSI block interface configuration.

#### Description

This structure encapsulates all the configurable parameters that can be selected for the MIPI-CSI interface. The [CyU3PMipicsiSetIntfParams\(\)](#) function accepts a pointer to this structure and updates the interface parameters.

#### Note

This structure has changed from the 1.3 SDK release (CX3 BETA release). If you are using CX3 code from the 1.3 SDK release, please update your code to use the current version of this structure. The changes between the 1.3 release and 1.3.1 release are as follows: 1) The order of structure members has changed. 2) A new member fifoDelay has been added. 3) The names for some members has changed (ppiClkDiv is now csiRxClkDiv, and sClkDiv is now parClkDiv).

#### See also

[CyU3PMipicsiSetIntfParams](#)  
[CyU3PMipicsiQueryIntfParams](#)  
[CyU3PMipicsiPliClkFrs\\_t](#)  
[CyU3PMipicsiPliClkDiv\\_t](#)

### 4.35.2 Field Documentation

#### 4.35.2.1 CyU3PMipicsiPliClkDiv\_t csiRxClkDiv

Clock divider for generating clock used for detecting CSI Link LP<->HS transitions

#### 4.35.2.2 CyU3PMipicsiDataFormat\_t dataFormat

Data Format expected at the GPIF. The image sensor will also need to be separately set to transmit in this format.

#### 4.35.2.3 uint16\_t fifoDelay

Threshold at which the output from the parallel output buffer on the MIPI-CSI interface will be initiated. Range 0x000 - 0x1FF

#### 4.35.2.4 uint16\_t hResolution

Horizontal Resolution defined in number of pixels per line of the video frame.

#### 4.35.2.5 uint16\_t mClkCtl

Note: This field is no longer actively set by the CX3 configuration tool as we do not recommend using the MCLK as a clock source for image sensors due to issues with high jitter.

Settings used to generate MCLK output clock. The output clock is available on the CLKM\_CX3 line  $mClkOutput = (PLL\_Clock/mClkRefDiv)/[(CY\_U3P\_GET\_MSB(mClkCtl) + 1) + (CY\_U3P\_GET\_LSB(mClkCtl) + 1)]$

#### 4.35.2.6 CyU3PMipicsiPliClkDiv\_t mClkRefDiv

Clock divider used for generating the MCLK



See also

[CyU3PMipicsiGetErrors](#)

#### 4.36.2 Field Documentation

4.36.2.1 `uint8_t crcErrCnt`

CRC Error Count

4.36.2.2 `uint8_t ctlErrCnt`

Control Error (Incorrect Line State Sequence) Count

4.36.2.3 `uint8_t eidErrCnt`

Unsupported Packet ID Error Count

4.36.2.4 `uint8_t frmErrCnt`

Framing Error Count

4.36.2.5 `uint8_t mdIErrCnt`

Multi-Data Lane Sync Byte Error Count

4.36.2.6 `uint8_t recrErrCnt`

Recoverable Packet Header Error Count

4.36.2.7 `uint8_t recSyncErrCnt`

Recoverable Sync Byte Error Count

4.36.2.8 `uint8_t unrErrCnt`

Unrecoverable Packet Header Error Count

4.36.2.9 `uint8_t unrSyncErrCnt`

Unrecoverable Sync Byte Error Count

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/[cyu3mipicsi.h](#)

### 4.37 CyU3POtgConfig\_t Struct Reference

OTG configuration information.

```
#include <cyu3usb0tg.h>
```

## Data Fields

- [CyU3POtgMode\\_t otgMode](#)
- [CyU3POtgChargerDetectMode\\_t chargerMode](#)
- [CyU3POtgEventCallback\\_t cb](#)

### 4.37.1 Detailed Description

OTG configuration information.

#### Description

This structure encapsulates all of the OTG configuration information, and is taken in as an argument by the CyU3POtgStart function.

#### See also

[CyU3POtgMode\\_t](#)  
[CyU3POtgEventCallback\\_t](#)  
[CyU3POtgStart](#)

### 4.37.2 Field Documentation

#### 4.37.2.1 CyU3POtgEventCallback\_t cb

OTG event callback function.

#### 4.37.2.2 CyU3POtgChargerDetectMode\_t chargerMode

Charger detect mode.

#### 4.37.2.3 CyU3POtgMode\_t otgMode

USB port mode of operation.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/[cyu3usbott.h](#)

## 4.38 CyU3PPibClock\_t Struct Reference

Clock configuration information for the PIB block.

```
#include <cyu3pib.h>
```

## Data Fields

- `uint16_t clkDiv`
- [CyBool\\_t isHalfDiv](#)
- [CyBool\\_t isDIIEnable](#)
- [CyU3PSysClockSrc\\_t clkSrc](#)

### 4.38.1 Detailed Description

Clock configuration information for the PIB block.

#### Description

The clock for the PIB block is derived from the SYS\_CLK. The base clock and frequency divider values are selected through this structure.

The default values used by the driver are:

clkDiv = 2, isHalfDiv = CyFalse, isDIIEnable = CyFalse, and clkSrc = CY\_U3P\_SYS\_CLK.

#### See also

[CyU3PSysClockSrc\\_t](#)  
[CyU3PPibInit](#)

### 4.38.2 Field Documentation

#### 4.38.2.1 uint16\_t clkDiv

Divider value for the PIB clock. The min value is 2 and max value is 1024.

#### 4.38.2.2 CyU3PSysClockSrc\_t clkSrc

The clock source to be used.

#### 4.38.2.3 CyBool\_t isDIIEnable

Whether the DLL should be enabled or not. The DLL in the PIB block should be enabled when implementing Asynchronous GPIF protocols, or Master mode GPIF protocols. It should be left turned off when implementing synchronous slave mode GPIF protocols.

#### 4.38.2.4 CyBool\_t isHalfDiv

If set to true, adds 0.5 to the frequency divider value selected by clkDiv.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/[cyu3pib.h](#)

## 4.39 CyU3PSdioCardRegs Struct Reference

SDIO Card Generic Information and CCCR registers.

```
#include <cyu3sib.h>
```

### Data Fields

- uint8\_t isMemoryPresent
- uint8\_t numberOfFunctions
- uint8\_t CCCRVersion
- uint8\_t sdioVersion
- uint8\_t cardCapability
- uint8\_t cardSpeed

- `uint16_t fn0BlockSize`
- `uint32_t addrCIS`
- `uint16_t manufacturerId`
- `uint16_t manufacturerInfo`
- `uint8_t uhsSupport`
- `uint8_t supportsAsynclntr`

#### 4.39.1 Detailed Description

SDIO Card Generic Information and CCCR registers.

##### Description

The following structure stores information about the SDIO card attached to a storage port of the FX3S.

#### 4.39.2 Field Documentation

##### 4.39.2.1 `uint32_t addrCIS`

Pointer to card's common Card Information Structure (CIS)

##### 4.39.2.2 `uint8_t cardCapability`

Sdio Card Capability register from the CCCR

##### 4.39.2.3 `uint8_t cardSpeed`

Sdio card speed information (Low Speed, Full Speed or High Speed Card)

##### 4.39.2.4 `uint8_t CCCRVersion`

CCCR Format Version Number as defined by SDIO spec

##### 4.39.2.5 `uint16_t fn0BlockSize`

Function 0 Block Size

##### 4.39.2.6 `uint8_t isMemoryPresent`

Is Memory is present on the SDIO. 1 in case of a Combo card, 0 for I/O only cards.

##### 4.39.2.7 `uint16_t manufacturerId`

Manufacturer ID

##### 4.39.2.8 `uint16_t manufacturerInfo`

Manufacturer information

##### 4.39.2.9 `uint8_t numberOfFunctions`

Number of I/O Functions present on the card

#### 4.39.2.10 uint8\_t sdioVersion

Lower 4 bits define SDIO Version supported by the card. Upper 4 bits define SD Physical Layer Spec supported by the card.

#### 4.39.2.11 uint8\_t supportsAsyncIntr

Interrupt Extension byte from CCCR SDIO 3.0 Asynchronous Interrupt support information.

#### 4.39.2.12 uint8\_t uhsSupport

UHS-I support byte from CCCR for SDIO3.0 cards

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/cyu3sib.h

## 4.40 CyU3PSibCtx Struct Reference

Structure that holds transfer state corresponding to each storage port.

```
#include <cyu3sibpp.h>
```

### Data Fields

- uint8\_t isRead
- volatile uint8\_t inUse
- uint8\_t partition
- uint8\_t activeUnitId
- uint8\_t numBootLuns
- uint8\_t numUserLuns
- CyU3PReturnStatus\_t status
- CyU3PMutex mutexLock
- CyU3PTimer writeTimer
- CyU3PTimerCb\_t writeTimerCbk

#### 4.40.1 Detailed Description

Structure that holds transfer state corresponding to each storage port.

#### 4.40.2 Field Documentation

##### 4.40.2.1 uint8\_t activeUnitId

The current active logical unit of the card

##### 4.40.2.2 volatile uint8\_t inUse

Variable to indicate if the card is being used.

**4.40.2.3 uint8\_t isRead**

Indicates if its a read or a write operation.

**4.40.2.4 CyU3PMutex mutexLock**

Mutex lock for the port

**4.40.2.5 uint8\_t numBootLuns**

Number of Logical UNits containing boot code.

**4.40.2.6 uint8\_t numUserLuns**

Number of user accessible logical units on the device.

**4.40.2.7 uint8\_t partition**

Flag indicating if the device is partitioned.

**4.40.2.8 CyU3PReturnStatus\_t status**

Return status from internal API calls.

**4.40.2.9 CyU3PTimer writeTimer**

Sib Write Timeout Timer

**4.40.2.10 CyU3PTimerCb\_t writeTimerCbk**

SIB Timer Call back function

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/cyu3sibpp.h

## 4.41 CyU3PSibDevInfo Struct Reference

Storage (SD/MMC) device properties.

```
#include <cyu3sib.h>
```

### Data Fields

- CyU3PSibDevType cardType
- uint32\_t clkRate
- uint32\_t numBlks
- uint32\_t eraseSize
- uint16\_t blkLen
- uint16\_t ccc
- uint8\_t removable

- `uint8_t writeable`
- `uint8_t locked`
- `uint8_t ddrMode`
- `uint8_t opVoltage`
- `uint8_t busWidth`
- `uint8_t numUnits`

#### 4.41.1 Detailed Description

Storage (SD/MMC) device properties.

##### Description

The following structure stores information about the storage device attached to the FX3S's storage ports. Please note that most of these fields are relevant only for SD and MMC devices. SDIO specific query APIs should be used to get the properties of an SDIO device.

##### See also

[CyU3PSibDevType](#)

#### 4.41.2 Field Documentation

##### 4.41.2.1 `uint16_t blkLen`

Current block size setting for the device.

##### 4.41.2.2 `uint8_t busWidth`

Current bus width setting for the device.

##### 4.41.2.3 `CyU3PSibDevType cardType`

Type of storage device connected on the S port. (Can be none if no device detected).

##### 4.41.2.4 `uint16_t ccc`

Card command classes (CCC) from the CSD register.

##### 4.41.2.5 `uint32_t clkRate`

Current operating clock frequency for the device.

##### 4.41.2.6 `uint8_t ddrMode`

Whether DDR clock mode is being used for this device.

##### 4.41.2.7 `uint32_t eraseSize`

The erase unit size in bytes for this device.

**4.41.2.8 uint8\_t locked**

Identifies whether the storage device is password locked.

**4.41.2.9 uint32\_t numBlks**

Number of blocks of the storage device.

**4.41.2.10 uint8\_t numUnits**

Number of boot LUNs & User LUNs present on this device.

**4.41.2.11 uint8\_t opVoltage**

Current operating voltage setting for the device.

**4.41.2.12 uint8\_t removable**

Indicates if the storage device is a removable device.

**4.41.2.13 uint8\_t writeable**

Whether the storage device is write enabled.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/cyu3sib.h

## 4.42 CyU3PSibGlobalData Struct Reference

Data structure that holds the storage driver state.

```
#include <cyu3sibpp.h>
```

### Data Fields

- [CyBool\\_t isActive](#)
- [CyBool\\_t s0Enabled](#)
- [CyBool\\_t s1Enabled](#)
- [CyU3PSibEvtCbk\\_t sibEvtCbk](#)
- [uint32\\_t wrCommitSize \[CY\\_U3P\\_SIB\\_NUM\\_PORTS\]](#)
- [uint32\\_t nextWrAddress \[CY\\_U3P\\_SIB\\_NUM\\_PORTS\]](#)
- [uint32\\_t openWrSize \[CY\\_U3P\\_SIB\\_NUM\\_PORTS\]](#)
- [CyBool\\_t wrCommitPending \[CY\\_U3P\\_SIB\\_NUM\\_PORTS\]](#)
- [CyU3PSibLunLocation activePartition \[CY\\_U3P\\_SIB\\_NUM\\_PORTS\]](#)
- [uint32\\_t partConfig \[CY\\_U3P\\_SIB\\_NUM\\_PORTS\]](#)
- [CyU3PDmaChannel sibDmaChannel](#)

#### 4.42.1 Detailed Description

Data structure that holds the storage driver state.

##### Description

This type defines a global data structure that holds all state information corresponding to the FX3S storage driver.

#### 4.42.2 Field Documentation

##### 4.42.2.1 CyU3PSibLunLocation activePartition[CY\_U3P\_SIB\_NUM\_PORTS]

Partition on which transfer is happening.

##### 4.42.2.2 CyBool\_t isActive

Driver is active or not.

##### 4.42.2.3 uint32\_t nextWrAddress[CY\_U3P\_SIB\_NUM\_PORTS]

Address at which a write has been paused.

##### 4.42.2.4 uint32\_t openWrSize[CY\_U3P\_SIB\_NUM\_PORTS]

Size of currently open write operation.

##### 4.42.2.5 uint32\_t partConfig[CY\_U3P\_SIB\_NUM\_PORTS]

Partition config definition from device.

##### 4.42.2.6 CyBool\_t s0Enabled

Whether S0 port is enabled by user.

##### 4.42.2.7 CyBool\_t s1Enabled

Whether S1 port is enabled by user.

##### 4.42.2.8 CyU3PDmaChannel sibDmaChannel

DMA channel used for device init.

##### 4.42.2.9 CyU3PSibEvtCbk\_t sibEvtCbk

Callback used for event notifications.

##### 4.42.2.10 CyBool\_t wrCommitPending[CY\_U3P\_SIB\_NUM\_PORTS]

Whether write commit is pending.

#### 4.42.2.11 uint32\_t wrCommitSize[CY\_U3P\_SIB\_NUM\_PORTS]

Size at which writes should be committed.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/cyu3sibpp.h

## 4.43 CyU3PSibIntfParams Struct Reference

Storage interface control parameters.

```
#include <cyu3sib.h>
```

### Data Fields

- uint8\_t resetGpio
- CyBool\_t rstActHigh
- uint8\_t cardDetType
- uint8\_t writeProtEnable
- uint8\_t lowVoltage
- uint8\_t voltageSwGpio
- CyBool\_t lvGpioState
- uint8\_t useDdr
- CyU3PSibOpFreq maxFreq
- uint8\_t cardInitDelay

### 4.43.1 Detailed Description

Storage interface control parameters.

#### Description

This structure encapsulates the desired operating conditions for the storage ports.

### 4.43.2 Field Documentation

#### 4.43.2.1 uint8\_t cardDetType

Card detection method that should be used for this S port.

#### 4.43.2.2 uint8\_t cardInitDelay

Initialization delay (in ms) to allow card to stabilize. Some storage devices appear to need some time to start responding after initial power up.

#### 4.43.2.3 uint8\_t lowVoltage

Enable/disable low voltage operation on the port.

#### 4.43.2.4 CyBool\_t lvGpioState

State of the GPIO that sets the S-port to low voltage: CyTrue : GPIO=0 => 3.3 V, GPIO=1 => 1.8 V. CyFalse: GPIO=1 => 3.3 V, GPIO=0 => 1.8 V.

#### 4.43.2.5 CyU3PSibOpFreq maxFreq

Maximum operating frequency for the S port.

#### 4.43.2.6 uint8\_t resetGpio

GPIO to be used for controlling power/reset to the storage device. Set to 255 if no GPIO is to be used.

#### 4.43.2.7 CyBool\_t rstActHigh

Whether the reset GPIO (if present) is active high or active low.

#### 4.43.2.8 uint8\_t useDdr

Whether to enable DDR clock for the S port.

#### 4.43.2.9 uint8\_t voltageSwGpio

FX3 GPIO to be used for voltage switching. Set to 255 if no GPIO is to be used.

#### 4.43.2.10 uint8\_t writeProtEnable

Indicates whether device specified write protection is allowed for this S port.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/cyu3sib.h

## 4.44 CyU3PSibLunInfo Struct Reference

Logical unit properties.

```
#include <cyu3sib.h>
```

### Data Fields

- uint32\_t startAddr
- uint32\_t numBlocks
- uint32\_t blockSize
- uint8\_t valid
- uint8\_t writeable
- CyU3PSibLunLocation location
- CyU3PSibLunType type

### 4.44.1 Detailed Description

Logical unit properties.

#### Description

The following structure stores information about each logical unit (partition) on the storage devices connected on each storage port. Each boot partition is counted as a separate logical unit. The logical units can be virtual partitions managed by the firmware, in the case where the storage device used does not support native partitions. A maximum of four logical units can be supported on each storage device.

See also

[CyU3PSibLunType](#)  
[CyU3PSibLunLocation](#)

## 4.44.2 Field Documentation

### 4.44.2.1 uint32\_t blockSize

Block size in bytes for this logical unit. This will be the same as the device block size.

### 4.44.2.2 CyU3PSibLunLocation location

Location of the logical unit.

### 4.44.2.3 uint32\_t numBlocks

Size of the partition in blocks.

### 4.44.2.4 uint32\_t startAddr

Starting address for the logical unit within the device.

### 4.44.2.5 CyU3PSibLunType type

Identifies the type of the logical unit.

### 4.44.2.6 uint8\_t valid

Whether this partition exists on the storage device.

### 4.44.2.7 uint8\_t writeable

Whether the unit is write enabled. The application can define separate write permissions for each unit.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/[cyu3sib.h](#)

## 4.45 CyU3PSpiConfig\_t Struct Reference

Structure defining the configuration of SPI interface.

```
#include <cyu3spi.h>
```

### Data Fields

- [CyBool\\_t isLsbFirst](#)
- [CyBool\\_t cpol](#)
- [CyBool\\_t cpha](#)
- [CyBool\\_t ssnPol](#)

- [CyU3PSpiSsnCtrl\\_t ssnCtrl](#)
- [CyU3PSpiSsnLagLead\\_t leadTime](#)
- [CyU3PSpiSsnLagLead\\_t lagTime](#)
- [uint32\\_t clock](#)
- [uint8\\_t wordLen](#)

#### 4.45.1 Detailed Description

Structure defining the configuration of SPI interface.

##### Description

This structure encapsulates all of the configurable parameters that can be selected for the SPI interface. The [CyU3PSpiSetConfig\(\)](#) function accepts a pointer to this structure, and updates all of the interface parameters.

##### See also

[CyU3PSpiSsnCtrl\\_t](#)  
[CyU3PSpiSclkParam\\_t](#)  
[CyU3PSpiSetConfig](#)

#### 4.45.2 Field Documentation

##### 4.45.2.1 uint32\_t clock

SPI clock frequency in Hz.

##### 4.45.2.2 CyBool\_t cpha

Clock phase - [CyFalse\(0\)](#): Slave samples at idle-active edge, [CyTrue\(1\)](#): Slave samples at active-idle edge

##### 4.45.2.3 CyBool\_t cpol

Clock polarity - [CyFalse\(0\)](#): SCK idles low, [CyTrue\(1\)](#): SCK idles high

##### 4.45.2.4 CyBool\_t isLsbFirst

Data shift mode - CyFalse: MSB first, CyTrue: LSB first

##### 4.45.2.5 CyU3PSpiSsnLagLead\_t lagTime

Time between the last SCK edge to SSN de-assertion. This is at the end of a transfer and is valid only when the hardware controls the SSN. When CPHA = 1, lag time cannot be zero.

##### 4.45.2.6 CyU3PSpiSsnLagLead\_t leadTime

Time between SSN assertion and first SCLK edge. This is at the beginning of a transfer and is valid only when the hardware controls the SSN. A lead time of zero is not supported.

##### 4.45.2.7 CyU3PSpiSsnCtrl\_t ssnCtrl

SSN control method.

#### 4.45.2.8 CyBool\_t ssnPol

Polarity of SSN line - CyFalse (0): SSN is active low, CyTrue (1): SSN is active high.

#### 4.45.2.9 uint8\_t wordLen

Word length in bits. Valid values are in the range 4 - 32.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/cyu3spi.h

## 4.46 CyU3PSysClockConfig\_t Struct Reference

Clock configuration for FX3 CPU, DMA and register access.

```
#include <cyu3system.h>
```

### Data Fields

- [CyBool\\_t setSysClk400](#)
- [uint8\\_t cpuClkDiv](#)
- [uint8\\_t dmaClkDiv](#)
- [uint8\\_t mmioClkDiv](#)
- [CyBool\\_t useStandbyClk](#)
- [CyU3PSysClockSrc\\_t clkSrc](#)

### 4.46.1 Detailed Description

Clock configuration for FX3 CPU, DMA and register access.

#### Description

This structure holds information to set the clock divider for CPU, DMA and internal register access. The DMA and register (MMIO) clocks are derived from the CPU clock.

There is an additional condition: DMA clock = N \* MMIO clock, where N is a positive integer.

The useStandbyClk parameter specifies whether a 32KHz clock has been supplied on the CLKIN\_32 pin of the device. This clock is the standby clock for the device. If this pin is not connected, then the device internally generates a clock from the SYS\_CLK.

The setSysClk400 parameter specifies whether the FX3 device's master clock is to be set to a frequency greater than 400 MHz. By default, the FX3 master clock is set to 384 MHz when using a 19.2 MHz crystal or clock source. This frequency setting may lead to DMA overflow errors on the GPIF, if the GPIF is configured as 32-bit wide and is running at 100 MHz. Setting this parameter will switch the master clock frequency to 403.2 MHz during the CyU3PDeviceInit call.

This structure is passed as parameter to CyU3PDeviceInit call.

#### See also

- [CyU3PSysClockSrc\\_t](#)
- [CyU3PDeviceInit](#)
- [CyU3PDeviceGetSysClkFreq](#)

## 4.46.2 Field Documentation

### 4.46.2.1 CyU3PSysClockSrc\_t clkSrc

Clock source for CPU clocking.

### 4.46.2.2 uint8\_t cpuClkDiv

CPU clock divider from clkSrc. Valid value ranges from 2 - 16.

### 4.46.2.3 uint8\_t dmaClkDiv

DMA clock divider from CPU clock. Valid value ranges from 2 - 16.

### 4.46.2.4 uint8\_t mmioClkDiv

MMIO clock divider from CPU clock. Valid value ranges from 2 - 16.

### 4.46.2.5 CyBool\_t setSysClk400

Whether the FX3 master (System) clock is to be set to a frequency greater than 400 MHz. This is required to be set to True if the GPIF is running in 32-bit mode at 100 MHz.

### 4.46.2.6 CyBool\_t useStandbyClk

Whether the 32 KHz standby clock is supplied.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/[cyu3system.h](#)

## 4.47 CyU3PUartConfig\_t Struct Reference

Configuration parameters for the UART interface.

```
#include <cyu3uart.h>
```

### Data Fields

- [CyBool\\_t txEnable](#)
- [CyBool\\_t rxEnable](#)
- [CyBool\\_t flowCtrl](#)
- [CyBool\\_t isDma](#)
- [CyU3PUartBaudrate\\_t baudRate](#)
- [CyU3PUartStopBit\\_t stopBit](#)
- [CyU3PUartParity\\_t parity](#)

## 4.47.1 Detailed Description

Configuration parameters for the UART interface.

### Description

This structure defines all of the configurable parameters for the UART interface such as baud rate, stop bits, parity bits etc. A pointer to this structure is passed in to the CyU3PUartSetConfig function to configure the UART interface.

The isDma member specifies whether the UART should be configured to transfer data one byte at a time, or in terms of larger blocks.

All of the parameters can be changed dynamically by calling the CyU3PUartSetConfig function repeatedly.

### See also

[CyU3PUartBaudrate\\_t](#)  
[CyU3PUartStopBit\\_t](#)  
[CyU3PUartParity\\_t](#)  
[CyU3PUartSetConfig](#)

## 4.47.2 Field Documentation

### 4.47.2.1 CyU3PUartBaudrate\_t baudRate

Baud rate for data transfer.

### 4.47.2.2 CyBool\_t flowCtrl

Enable hardware flow control for both RX and TX.

### 4.47.2.3 CyBool\_t isDma

CyFalse: Byte by byte transfer; CyTrue: Block based transfer.

### 4.47.2.4 CyU3PUartParity\_t parity

Parity configuration.

### 4.47.2.5 CyBool\_t rxEnable

Enable the receiver.

### 4.47.2.6 CyU3PUartStopBit\_t stopBit

The number of stop bits appended.

### 4.47.2.7 CyBool\_t txEnable

Enable the transmitter.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/[cyu3uart.h](#)

## 4.48 CyU3PUsbDescrPtrs Struct Reference

Pointer to the various descriptors.

```
#include <cyfx3usb.h>
```

### Data Fields

- `uint8_t * usbDevDesc_p`
- `uint8_t * usbSSDevDesc_p`
- `uint8_t * usbDevQualDesc_p`
- `uint8_t * usbConfigDesc_p`
- `uint8_t * usbOtherSpeedConfigDesc_p`
- `uint8_t * usbHSConfigDesc_p`
- `uint8_t * usbFSConfigDesc_p`
- `uint8_t * usbSSConfigDesc_p`
- `uint8_t * usbStringDesc_p [CY_FX3_USB_MAX_STRING_DESC_INDEX]`
- `uint8_t * usbSSBOSDesc_p`

### 4.48.1 Detailed Description

Pointer to the various descriptors.

#### Description

This data structure stores pointers to the various USB descriptors. These pointers are set as part of the [CyFx3BootUsbSetDesc\(\)](#) function.

### 4.48.2 Field Documentation

#### 4.48.2.1 `uint8_t* usbConfigDesc_p`

Pointer to config desc of device

#### 4.48.2.2 `uint8_t* usbDevDesc_p`

Pointer to device desc of device

#### 4.48.2.3 `uint8_t* usbDevQualDesc_p`

Pointer to device qualifier desc of device

#### 4.48.2.4 `uint8_t* usbFSConfigDesc_p`

Pointer to FULL SPEED speed configuration desc of device

#### 4.48.2.5 `uint8_t* usbHSConfigDesc_p`

Pointer to HIGH SPEED speed configuration desc of device

#### 4.48.2.6 `uint8_t* usbOtherSpeedConfigDesc_p`

Pointer to other speed configuration desc of device

**4.48.2.7 uint8\_t\* usbSSBOSDesc\_p**

Pointer to Super speed BOS descriptor

**4.48.2.8 uint8\_t\* usbSSConfigDesc\_p**

Pointer to SUPER SPEED speed configuration desc of device

**4.48.2.9 uint8\_t\* usbSSDevDesc\_p**

Pointer to SS device desc of device

**4.48.2.10 uint8\_t\* usbStringDesc\_p[CY\_FX3\_USB\_MAX\_STRING\_DESC\_INDEX]**

Array of pointers to string descriptors.

The documentation for this struct was generated from the following file:

- firmware/boot-fw/include/cyfx3usb.h

## 4.49 CyU3PUsbHostConfig\_t Struct Reference

USB host mode configuration information.

```
#include <cyu3usbhost.h>
```

### Data Fields

- [CyBool\\_t ep0LowLevelControl](#)
- [CyU3PUsbHostEventCb\\_t eventCb](#)
- [CyU3PUsbHostXferCb\\_t xferCb](#)

### 4.49.1 Detailed Description

USB host mode configuration information.

#### Description

The FX3 host mode driver takes the following configuration parameters, which are set when starting the stack. These settings cannot be changed dynamically while the stack is active.

#### See also

[CyU3PUsbHostStart](#)

### 4.49.2 Field Documentation

#### 4.49.2.1 CyBool\_t ep0LowLevelControl

Whether to enable EP0 low level control. If CyFalse, the EP0 DMA is handled by firmware. Only the CyU3PUsbHostSendSetupRqt API needs to be called. If CyTrue, EP0 DMA transfers need to be handled by the application. This allows fine control over setup, data and status phase. This mode should be used only if the EP0 data needs to be routed to a different path out of the FX3 device. The maxPktSize and fullPktSize for EP0 should be the same of this setting is set to true.

#### 4.49.2.2 CyU3PUsbHostEventCb\_t eventCb

Event callback function for USB host stack.

#### 4.49.2.3 CyU3PUsbHostXferCb\_t xferCb

EP transfer completion callback for USB host stack.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/cyu3usbhost.h

## 4.50 CyU3PUsbHostEpConfig\_t Struct Reference

Host mode endpoint configuration structure.

```
#include <cyu3usbhost.h>
```

### Data Fields

- CyU3PUsbEpType\_t type
- uint8\_t mult
- uint16\_t maxPktSize
- uint8\_t pollingRate
- uint16\_t fullPktSize
- CyBool\_t isStreamMode

### 4.50.1 Detailed Description

Host mode endpoint configuration structure.

#### Description

The structure holds the information for configuring an endpoint when the FX3 is acting as a USB host.

See also

[CyU3PUsbHostEpAdd](#)

### 4.50.2 Field Documentation

#### 4.50.2.1 uint16\_t fullPktSize

This is used for DMA packet boundary identification. If the DMA buffer allocated is larger than the maxPktSize specified, this field determines when a buffer is wrapped up. A DMA buffer is wrapped up by the hardware when it sees a SLP or ZLP. So, as long as the data received is a multiple of fullPktSize, the buffer is not wrapped up. fullPktSize cannot be smaller than the maxPktSize.

#### 4.50.2.2 CyBool\_t isStreamMode

Enable stream mode. This means that the EP is always active. This setting is valid only for an IN EP, and should be CyFalse for EP0 and OUT endpoints. When the flag is set, an IN EP will send IN tokens and collect data whenever there is a free buffer on the DMA channel.

#### 4.50.2.3 uint16\_t maxPktSize

The maximum packet size for the endpoint. Valid range is as defined in the USB specification.

#### 4.50.2.4 uint8\_t mult

Number of packets to be transferred per micro-frame. This should be 1 for bulk, control and interrupt endpoints; and 1, 2, or 3 for isochronous endpoints.

#### 4.50.2.5 uint8\_t pollingRate

Rate at which the endpoint has to be polled in ms. Zero will indicate that polling will be done every micro-frame and any non-zero value will specify the polling rate. It should be noted that pollingRate is valid only when the request itself is larger than a single packet. This setting is valid for synchronous endpoints. For asynchronous endpoints, this should be set to zero.

#### 4.50.2.6 CyU3PUsbEpType\_t type

Type of endpoint to be created.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/cyu3usbhost.h

## 4.51 MemBlockInfo Struct Reference

Structure representing a block of memory that has been dynamically allocated and is in use.

```
#include <cyu3os.h>
```

### Data Fields

- uint32\_t alloc\_id
- uint32\_t alloc\_size
- struct MemBlockInfo \* prev\_blk
- struct MemBlockInfo \* next\_blk
- uint32\_t start\_sig
- uint32\_t pad [3]

### 4.51.1 Detailed Description

Structure representing a block of memory that has been dynamically allocated and is in use.

#### Description

This structure represents the header associated with a memory block allocated through the CyU3PMemAlloc and CyU3PDmaBufferAlloc functions. This header structure is used to check for memory leak and corruption occurrences.

### 4.51.2 Field Documentation

#### 4.51.2.1 uint32\_t alloc\_id

Allocation id.

**4.51.2.2 uint32\_t alloc\_size**

Actual memory size requested.

**4.51.2.3 struct MemBlockInfo\* next\_blk**

Next memory block.

**4.51.2.4 uint32\_t pad[3]**

Padding used to extend the header to 32 bytes.

**4.51.2.5 struct MemBlockInfo\* prev\_blk**

Previous memory block.

**4.51.2.6 uint32\_t start\_sig**

Block start signature.

The documentation for this struct was generated from the following file:

- firmware/u3p\_firmware/inc/[cyu3os.h](#)

# Chapter 5

## File Documentation

### 5.1 firmware/boot-fw/include/cyfx3device.h File Reference

The Boot APIs for FX3 provide a low footprint API library that can be used to put together simple FX3 applications, primarily for the purpose of building custom boot-loaders.

```
#include <cyu3types.h>
#include <cyfx3error.h>
#include <cyfx3_api.h>
#include <cyu3externcstart.h>
#include <cyu3externcend.h>
```

#### Data Structures

- struct [CyFx3BootIoMatrixConfig\\_t](#)  
*Defines the IO matrix configuration parameters.*

#### Macros

- #define [CY\\_FX3\\_BOOT\\_ITCM\\_BASE](#) (0x00000000)  
*Base address of the I-TCM region.*
- #define [CY\\_FX3\\_BOOT\\_ITCM\\_END](#) ([CY\\_FX3\\_BOOT\\_ITCM\\_BASE](#) + 0x4000)  
*End address of the I-TCM region.*
- #define [CY\\_FX3\\_BOOT\\_SYSMEM\\_BASE](#) (0x40000000)  
*Base address of the SYSMEM region.*
- #define [CY\\_FX3\\_BOOT\\_SYSMEM\\_END](#) ([CY\\_FX3\\_BOOT\\_SYSMEM\\_BASE](#) + 0x80000)  
*End address of the SYSMEM region. This value is valid for the CYUSB3014 part. Some other FX3 parts could have smaller SYSMEM regions.*
- #define [CY\\_FX3\\_BOOT\\_SYSMEM\\_BASE1](#) ([CY\\_FX3\\_BOOT\\_SYSMEM\\_BASE](#) + 0x2000)  
*Start address of the usable range of SYSMEM.*
- #define [CY\\_FX3\\_BOOT\\_NO\\_WAIT](#) (0x00)  
*Option that specifies that a polling API should return immediately after checking the status of the transfer.*
- #define [CY\\_FX3\\_BOOT\\_WAIT\\_FOREVER](#) (0xFFFFFFFF)  
*Option that specifies that a polling API should wait as long as required for a transfer to complete.*

## Typedefs

- `typedef enum CyFx3BootSysClockSrc_t CyFx3BootSysClockSrc_t`  
*Clock source for a peripheral block.*
- `typedef CyU3PPartNumber_t CyFx3PartNumber_t`  
*Enumeration of EZ-USB FX3 part numbers.*
- `typedef struct CyFx3BootIoMatrixConfig_t CyFx3BootIoMatrixConfig_t`  
*Defines the IO matrix configuration parameters.*

## Enumerations

- `enum CyFx3BootSysClockSrc_t {  
CY_FX3_BOOT_SYS_CLK_BY_16 = 0, CY_FX3_BOOT_SYS_CLK_BY_4, CY_FX3_BOOT_SYS_CLK_BY_2,  
CY_FX3_BOOT_SYS_CLK, CY_FX3_BOOT_NUM_CLK_SRC }`  
*Clock source for a peripheral block.*

## Functions

- `void CyFx3BootDeviceInit (CyBool_t setFastSysClk)`  
*This function initializes the FX3 device.*
- `void CyFx3BootJumpToProgramEntry (uint32_t address)`  
*Transfer control to the specified address.*
- `CyFx3BootErrorCode_t CyFx3BootDeviceConfigureIOMatrix (CyFx3BootIoMatrixConfig_t *cfg_p)`  
*Configure the IO matrix for the device.*
- `void CyFx3BootWatchdogConfigure (CyBool_t enable, uint32_t period)`  
*Enable/disable the watchdog timer.*
- `void CyFx3BootWatchdogClear (void)`  
*Clear the watchdog timer to prevent device reset.*
- `CyFx3PartNumber_t CyFx3BootGetPartNumber (void)`  
*Get the part number of the FX3 device in use.*
- `void CyFx3BootRetainGpioState (void)`  
*Request to keep the GPIO block powered ON across control transfer to the full firmware.*
- `CyFx3BootErrorCode_t CyFx3BootGpioOverride (uint8_t pinNumber)`  
*Override a specific FX3 pin as a GPIO.*
- `CyFx3BootErrorCode_t CyFx3BootGpioRestore (uint8_t pinNumber)`  
*Restore the standard function of a pin.*
- `void CyFx3BootDeviceReset (void)`  
*Reset the FX3 device.*

### 5.1.1 Detailed Description

The Boot APIs for FX3 provide a low footprint API library that can be used to put together simple FX3 applications, primarily for the purpose of building custom boot-loaders.

#### Description

The regular FX3 API library is designed for maximum flexibility and performance, and makes use of an embedded OS. This results in a higher memory footprint for simple applications. This is not always desirable, particularly in cases where a small boot-loader type application is required to improve the flexibility and robustness of the boot process.

The FX3 boot API library is provided to address this problem, and provides a low footprint set of APIs without any RTOS dependency. This library only supports a small set of features, all of which are targeted at building booting applications.

The software boot supports the following interfaces on the FX3 device:

1. USB device mode

2. SPI

3. I2C

4. UART

5. GPIO

This library does not make use of an RTOS and minimal use of interrupts. All of the serial peripheral APIs operate on a polling model, and it is expected that these will be called in the appropriate sequence from the firmware main.

The only interrupt that is enabled is for the USB block. This interrupt handler ensures that all of the USB 2.0 and USB 3.0 link and protocol requests are handled appropriately, and provides hooks in the form of callback functions.

### 5.1.2 FX3 Memory Regions

ITCM and SYSMEM memory address ranges on the FX3 device.

#### Description

The I-TCM is a 16 KB memory region which is tightly coupled to the ARM9 CPU and can be used to locate interrupt service routines. The first 256 bytes of the I-TCM are reserved for setting up the ARM exception vectors.

The SYSMEM region is the main code/data RAM on the FX3 device, and can be 512 KB or 256 KB in size depending on the part being used. The first 8 KB of the SYSMEM is reserved for holding DMA related data structures (descriptors).

### 5.1.3 Typedef Documentation

#### 5.1.3.1 `typedef struct CyFx3BootIoMatrixConfig_t CyFx3BootIoMatrixConfig_t`

Defines the IO matrix configuration parameters.

#### Description

Most of the IOs on the FX3 device are multi-purpose; and the specific function that each pin should serve need to be selected during device initialization. This structure defines all of the parameters that are required to configure the FX3 IOs.

Please note that the DQ[15:0] pins, CTL[3:0] pins and the PMODE[2:0] pins are reserved and cannot be enabled as GPIOs through this structure. The GPIO Override APIs need to be used for this purpose.

#### See also

[CyFx3BootDeviceConfigureIOMatrix](#)

[CyFx3BootGpioOverride](#)

### 5.1.3.2 `typedef enum CyFx3BootSysClockSrc_t CyFx3BootSysClockSrc_t`

Clock source for a peripheral block.

#### Description

The clocks for various hardware blocks on the FX3 device are derived using frequency dividers from the master system clock. The source clock for these frequency dividers can be the master system clock itself or some other clocks derived from it.

This type lists the various clocks that can be used as the source clock for deriving the peripheral clocks.

### 5.1.3.3 `typedef CyU3PPartNumber_t CyFx3PartNumber_t`

Enumeration of EZ-USB FX3 part numbers.

#### Description

There are multiple EZ-USB FX3 parts which support varying feature sets. Please refer to the device data sheets or the Cypress device catalogue for information on the features supported by each FX3 part.

This enumerated type lists the various valid part numbers in the EZ-USB FX3 family.

#### See also

[CyFx3BootGetPartNumber](#)

## 5.1.4 Enumeration Type Documentation

### 5.1.4.1 `enum CyFx3BootSysClockSrc_t`

Clock source for a peripheral block.

#### Description

The clocks for various hardware blocks on the FX3 device are derived using frequency dividers from the master system clock. The source clock for these frequency dividers can be the master system clock itself or some other clocks derived from it.

This type lists the various clocks that can be used as the source clock for deriving the peripheral clocks.

#### Enumerator

- `CY_FX3_BOOT_SYS_CLK_BY_16` SYS\_CLK divided by 16.
- `CY_FX3_BOOT_SYS_CLK_BY_4` SYS\_CLK divided by 4.
- `CY_FX3_BOOT_SYS_CLK_BY_2` SYS\_CLK divided by 2.
- `CY_FX3_BOOT_SYS_CLK` SYS\_CLK itself.
- `CY_FX3_BOOT_NUM_CLK_SRC` Number of clock source enumerations.

## 5.1.5 Function Documentation

### 5.1.5.1 `CyFx3BootErrorCode_t CyFx3BootDeviceConfigureIOMatrix ( CyFx3BootIoMatrixConfig_t * cfg_p )`

Configure the IO matrix for the device.

#### Description

This function configures the functionality for each of the FX3 IO pins. If the GPIF data bus is 32-bits wide, only one from among the SPI, UART and I2S peripherals can be used. If the GPIF data bus is not 32-bits wide; all of the SPI, UART and I2S interfaces are available for simultaneous use. The specific pins mapped to these serial interfaces can also be changed.

This API item the IO matrix user specified parameters. Any pins not used part of the GPIF or serial peripheral interfaces can be used as a GPIO.

The IO configuration is not expected to be changed dynamically, and it is recommended that this be setup as soon as the CyFx3BootDeviceInit API has been called.

#### Return value

CY\_FX3\_BOOT\_SUCCESS - When the IO configuration is successful.  
 CY\_FX3\_BOOT\_ERROR\_NOT\_SUPPORTED - the FX3 part in use does not support the desired configuration.  
 CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT - If some configuration value is invalid.

#### See also

[CyFx3BootIoMatrixConfig\\_t](#)

#### Parameters

<i>cfg_p</i>	Pointer to configuration structure.
--------------	-------------------------------------

### 5.1.5.2 void CyFx3BootDeviceInit ( CyBool\_t setFastSysClk )

This function initializes the FX3 device.

#### Description

The function is expected to be invoked as the first call from the main () function. This function should be called only once.

The setFastSysClk parameter is equivalent to the setSysClk400 parameter used in the main FX3 API library; and specifies that the SYSCLK frequency should be set to a value greater than 400 MHz.

#### Parameters

<i>setFastSysClk</i>	Indicates whether the FX3 system clock should be set to faster than 400 MHz or not. Should be set to CyTrue if the GPIO will be used in Synchronous 32-bit mode at 100 MHz.
----------------------	---

### 5.1.5.3 void CyFx3BootDeviceReset ( void )

Reset the FX3 device.

#### Description

Function to do a full FX3 device reset and return to ROM boot-loader.

#### Return value

None

### 5.1.5.4 CyFx3PartNumber\_t CyFx3BootGetPartNumber ( void )

Get the part number of the FX3 device in use.

#### Description

The EZ-USB FX3 family has multiple parts which support various sets of features. This function can be used to query the part number of the current device so as to check whether specific functionality is supported or not.

#### Return value

Part number of the FX3 device in use.

#### See also

[CyFx3PartNumber\\_t](#)

### 5.1.5.5 CyFx3BootErrorCode\_t CyFx3BootGpioOverride ( *uint8\_t pinNumber* )

Override a specific FX3 pin as a GPIO.

#### Description

Some of the FX3 device pins are reserved for standard functions and not allowed to be configured as GPIOs at the time of configuring the IO Matrix. It is possible that the customer design does not make use of the standard functionality of these pins, and needs to use them as GPIOs. This function is used to override a specified IO pin as a GPIO.

Please note that this API does not check whether the pin being overridden is currently in use by any of the other interfaces. Therefore, this API should be used with caution.

#### Return value

CY\_FX3\_BOOT\_SUCCESS if the pin override is successful.

CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT if the pin specified is not valid.

See also

[CyFx3BootDeviceConfigureIOMatrix](#)

[CyFx3BootGpioRestore](#)

#### Parameters

<i>pinNumber</i>	Pin number to be over-ridden as a simple GPIO.
------------------	--

### 5.1.5.6 CyFx3BootErrorCode\_t CyFx3BootGpioRestore ( *uint8\_t pinNumber* )

Restore the standard function of a pin.

#### Description

This function restores the standard function of a pin that was previously over-ridden as a simple GPIO.

#### Return value

CY\_FX3\_BOOT\_SUCCESS if the pin restore is successful.

CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT if the pin specified is not valid.

See also

[CyFx3BootGpioOverride](#)

#### Parameters

<i>pinNumber</i>	Pin number to be restored to default function.
------------------	--

### 5.1.5.7 void CyFx3BootJumpToProgramEntry ( *uint32\_t address* )

Transfer control to the specified address.

#### Description

This function is used to transfer the control to the next stage's program entry. All the Serial IOs except GPIO (I2C, SPI, UART) that have been initialized must be de-initialized prior to calling this function.

**Parameters**

<code>address</code>	The program entry address
----------------------	---------------------------

5.1.5.8 `void CyFx3BootRetainGpioState( void )`

Request to keep the GPIO block powered ON across control transfer to the full firmware.

**Description**

All serial peripheral blocks on the FX3 device are normally reset when control of execution is transferred to the full firmware. This API is used to specify that the GPIO block should be left ON while jumping to the full firmware.

**Return value**

None

5.1.5.9 `void CyFx3BootWatchdogClear( void )`

Clear the watchdog timer to prevent device reset.

**Description**

This function is used to clear the watchdog timer so as to prevent the timer from resetting the device. This function needs to be called more often than the period of the watchdog timer.

**Return value**

None

**See also**

[CyFx3BootWatchdogConfigure](#)

5.1.5.10 `void CyFx3BootWatchdogConfigure( CyBool_t enable, uint32_t period )`

Enable/disable the watchdog timer.

**Description**

The FX3 device implements a watchdog timer that can be used to reset the device when the CPU is not responsive. This function is used to enable the watchdog feature and to set the period for the timer.

**Return value**

None

**See also**

[CyFx3BootWatchdogClear](#)

**Parameters**

<code>enable</code>	Whether the watchdog timer is to be enabled or disabled.
<code>period</code>	Period for the timer in milliseconds. Used only for enable calls.

## 5.2 firmware/boot\_fw/include/cyfx3dma.h File Reference

The DMA driver module in the FX3 boot firmware provides a set of APIs to manage data transfers, and provides DMA interrupt handlers.

```
#include <cyfx3error.h>
#include <cyu3types.h>
#include <cyfx3device.h>
#include <cyu3externcstart.h>
#include <cyu3externcend.h>
```

## Data Structures

- struct [CyFx3BootDmaDescriptor\\_t](#)  
*Descriptor data structure.*
- struct [CyFx3BootDmaSocket\\_t](#)  
*DMA socket configuration structure.*
- struct [CyFx3BootDmaSockRegs\\_t](#)  
*Actual socket specific register structure.*

## Macros

- #define CY\_FX3\_DMA\_USB\_OUT SOCKCNT (16)
- #define CY\_FX3\_DMA\_USB\_IN SOCKCNT (16)
- #define CY\_FX3\_DMA\_PIB SOCKCNT (32)
- #define CY\_FX3\_DMA\_LPP SOCKCNT (8)
- #define CY\_FX3\_USB\_DMA\_DSCR\_INDEX (1)
- #define CY\_FX3\_LPP\_DMA\_DSCR\_INDEX (2)
- #define CY\_FX3\_PIB\_DMA\_DSCR\_INDEX (3)
- #define CY\_FX3\_DMA\_MIN\_DSCR\_INDEX (4)
- #define CY\_FX3\_DMA\_MAX\_DSCR\_INDEX (511)
- #define CY\_FX3\_DMA\_GETDSCR\_BY\_INDEX(i) ((CyFx3BootDmaDescriptor\_t \*) (0x40000000 + ((i) << 4)))

## Typedefs

- typedef enum [CyFx3BootDmaSockId\\_t](#) CyFx3BootDmaSockId\_t  
*DMA socket IDs for all sockets in the device.*
- typedef struct [CyFx3BootDmaDescriptor\\_t](#) CyFx3BootDmaDescriptor\_t  
*Descriptor data structure.*
- typedef struct [CyFx3BootDmaSocket\\_t](#) CyFx3BootDmaSocket\_t  
*DMA socket configuration structure.*
- typedef struct [CyFx3BootDmaSockRegs\\_t](#) CyFx3BootDmaSockRegs\_t  
*Actual socket specific register structure.*
- typedef void(\* [CyFx3BootDmaCallback\\_t](#)) (uint16\_t sckId, uint32\_t status)  
*DMA socket interrupt handler callback function.*

## Enumerations

- enum CyFx3BootDmaSockId\_t {
 CY\_DMA\_LPP\_SOCKET\_I2S\_LEFT = 0x0000, CY\_DMA\_LPP\_SOCKET\_I2S\_RIGHT, CY\_DMA\_LPP\_SOCKET\_I2C\_CONS, CY\_DMA\_LPP\_SOCKET\_UART\_CONS, CY\_DMA\_LPP\_SOCKET\_SPI\_CONS, CY\_DMA\_LPP\_SOCKET\_I2C\_PROD, CY\_DMA\_LPP\_SOCKET\_UART\_PROD, CY\_DMA\_LPP\_SOCKET\_SPI\_PROD, CY\_DMA\_PIB\_SOCKET\_0 = 0x0100, CY\_DMA\_PIB\_SOCKET\_1, CY\_DMA\_PIB\_SOCKET\_2, CY\_DMA\_PIB\_SOCKET\_3}

```

    _PIB_SOCKET_3,
    CY_DMA_PIB_SOCKET_4, CY_DMA_PIB_SOCKET_5, CY_DMA_PIB_SOCKET_6, CY_DMA_PIB_SO-
    CKET_7,
    CY_DMA_PIB_SOCKET_8, CY_DMA_PIB_SOCKET_9, CY_DMA_PIB_SOCKET_10, CY_DMA_PIB_SO-
    CKET_11,
    CY_DMA_PIB_SOCKET_12, CY_DMA_PIB_SOCKET_13, CY_DMA_PIB_SOCKET_14, CY_DMA_PIB_SO-
    CKET_15,
    CY_DMA_PIB_SOCKET_16, CY_DMA_PIB_SOCKET_17, CY_DMA_PIB_SOCKET_18, CY_DMA_PIB_SO-
    CKET_19,
    CY_DMA_PIB_SOCKET_20, CY_DMA_PIB_SOCKET_21, CY_DMA_PIB_SOCKET_22, CY_DMA_PIB_SO-
    CKET_23,
    CY_DMA_PIB_SOCKET_24, CY_DMA_PIB_SOCKET_25, CY_DMA_PIB_SOCKET_26, CY_DMA_PIB_SO-
    CKET_27,
    CY_DMA_PIB_SOCKET_28, CY_DMA_PIB_SOCKET_29, CY_DMA_PIB_SOCKET_30, CY_DMA_PIB_SO-
    CKET_31,
    CY_DMA_SIB_SOCKET_0 = 0x0200, CY_DMA_SIB_SOCKET_1, CY_DMA_SIB_SOCKET_2, CY_DMA-
    _SIB_SOCKET_3,
    CY_DMA_SIB_SOCKET_4, CY_DMA_SIB_SOCKET_5, CY_DMA_UIB_SOCKET_CONS_0 = 0x0300, C-
    Y_DMA_UIB_SOCKET_CONS_1,
    CY_DMA_UIB_SOCKET_CONS_2, CY_DMA_UIB_SOCKET_CONS_3, CY_DMA_UIB_SOCKET_CONS-
    _4, CY_DMA_UIB_SOCKET_CONS_5,
    CY_DMA_UIB_SOCKET_CONS_6, CY_DMA_UIB_SOCKET_CONS_7, CY_DMA_UIB_SOCKET_CONS-
    _8, CY_DMA_UIB_SOCKET_CONS_9,
    CY_DMA_UIB_SOCKET_CONS_10, CY_DMA_UIB_SOCKET_CONS_11, CY_DMA_UIB_SOCKET_CO-
    NS_12, CY_DMA_UIB_SOCKET_CONS_13,
    CY_DMA_UIB_SOCKET_CONS_14, CY_DMA_UIB_SOCKET_CONS_15, CY_DMA_UIB_SOCKET_PR-
    OD_0 = 0x400, CY_DMA_UIB_SOCKET_PROD_1,
    CY_DMA_UIB_SOCKET_PROD_2, CY_DMA_UIB_SOCKET_PROD_3, CY_DMA_UIB_SOCKET_PROD-
    _4, CY_DMA_UIB_SOCKET_PROD_5,
    CY_DMA_UIB_SOCKET_PROD_6, CY_DMA_UIB_SOCKET_PROD_7, CY_DMA_UIB_SOCKET_PROD-
    _8, CY_DMA_UIB_SOCKET_PROD_9,
    CY_DMA_UIB_SOCKET_PROD_10, CY_DMA_UIB_SOCKET_PROD_11, CY_DMA_UIB_SOCKET_PR-
    OD_12, CY_DMA_UIB_SOCKET_PROD_13,
    CY_DMA_UIB_SOCKET_PROD_14, CY_DMA_UIB_SOCKET_PROD_15, CY_DMA_CPU_SOCKET_C-
    ONS = 0x3F00, CY_DMA_CPU_SOCKET_PROD }

```

*DMA socket IDs for all sockets in the device.*

## Functions

- void [CyFx3BootDmaRegisterCallback](#) ([CyFx3BootDmaCallback\\_t](#) cbFunc, [CyBool\\_t](#) usblntrEn, [CyBool\\_t](#) piblntrEn, [CyBool\\_t](#) seriallntrEn)
 

*Register a callback for notification of DMA interrupts.*
- [CyFx3BootErrorCode\\_t CyFx3BootDmaGetDscrConfig](#) ([uint16\\_t](#) dscriIndex, [CyFx3BootDmaDescriptor\\_t](#) \*dscr\_p)
 

*Get the current values from the specified DMA descriptor index.*
- [CyFx3BootErrorCode\\_t CyFx3BootDmaSetDscrConfig](#) ([uint16\\_t](#) dscriIndex, [CyFx3BootDmaDescriptor\\_t](#) \*dscr\_p)
 

*Update the values of a DMA descriptor on the FX3 device.*
- [uint32\\_t CyFx3BootDmaGetSockInterrupts](#) ([CyFx3BootDmaSockId\\_t](#) sockId)
 

*Get the current interrupt status for a given socket.*
- [CyFx3BootErrorCode\\_t CyFx3BootDmaGetSocketConfig](#) ([CyFx3BootDmaSockId\\_t](#) sockId, [CyFx3BootDmaSocket\\_t](#) \*sock\_p)
 

*Get the current status of a DMA socket.*
- [CyFx3BootErrorCode\\_t CyFx3BootDmaSetSocketConfig](#) ([CyFx3BootDmaSockId\\_t](#) sockId, [CyFx3BootDmaSocket\\_t](#) \*sock\_p)
 

*Update the configuration of a DMA socket.*

- `CyFx3BootErrorCode_t CyFx3BootDmaWrapSocket (CyFx3BootDmaSockId_t sockId)`  
*Forcibly commit the currently available data in an ingress socket.*
- `CyFx3BootErrorCode_t CyFx3BootDmaDisableSocket (CyFx3BootDmaSockId_t sockId)`  
*Disable a DMA socket.*
- `CyFx3BootErrorCode_t CyFx3BootDmaEnableSocket (CyFx3BootDmaSockId_t sockId)`  
*Enable a DMA socket.*
- `CyFx3BootErrorCode_t CyFx3BootDmaSendSocketEvent (CyFx3BootDmaSockId_t sockId, uint16_t dscrIndex, CyBool_t isOccupied)`  
*Send an event to notify a socket that a descriptor that is waiting on has been modified.*
- `void CyFx3BootDmaClearSocklInterrupts (CyFx3BootDmaSockId_t sockId, uint32_t intrVal)`  
*Clear the specified socket interrupts.*

### 5.2.1 Detailed Description

The DMA driver module in the FX3 boot firmware provides a set of APIs to manage data transfers, and provides DMA interrupt handlers.

### 5.2.2 Macro Definition Documentation

5.2.2.1 `#define CY_FX3_DMA_GETDSCR_BY_INDEX( i ) ((CyFx3BootDmaDescriptor_t *) (0x40000000 + ((i) << 4)))`

Calculate DMA descriptor address corresponding to an index.

5.2.2.2 `#define CY_FX3_DMA_LPP_SOCKCNT (8)`

Number of LPP Sockets.

5.2.2.3 `#define CY_FX3_DMA_MAX_DSCR_INDEX (511)`

Maximum allowed descriptor index. This is based on an allocation of 8 KB for DMA descriptors.

5.2.2.4 `#define CY_FX3_DMA_MIN_DSCR_INDEX (4)`

Minimum allowed descriptor index. The first 4 descriptors are reserved for API internal usage.

5.2.2.5 `#define CY_FX3_DMA_PIB_SOCKCNT (32)`

Number of PIB Sockets.

5.2.2.6 `#define CY_FX3_DMA_USB_IN_SOCKCNT (16)`

Number of USB Egress Sockets.

5.2.2.7 `#define CY_FX3_DMA_USB_OUT_SOCKCNT (16)`

Number of USB Ingress Sockets.

5.2.2.8 `#define CY_FX3_LPP_DMA_DSCR_INDEX (2)`

DMA descriptor used for LPP transfers.

5.2.2.9 #define CY\_FX3\_PIB\_DMA\_DSCR\_INDEX (3)

DMA descriptor used for PIB transfers.

5.2.2.10 #define CY\_FX3\_USB\_DMA\_DSCR\_INDEX (1)

DMA descriptor used for USB transfers.

### 5.2.3 Typedef Documentation

5.2.3.1 `typedef void(* CyFx3BootDmaCallback_t) (uint16_t sckId, uint32_t status )`

DMA socket interrupt handler callback function.

#### Description

Callback function type used to notify user about DMA socket interrupts. This callback will be invoked from interrupt context with all other interrupts disabled. Adding delays in the callback implementation may adversely impact handling of USB block interrupts.

See also

[CyFx3BootDmaRegisterCallback](#)

5.2.3.2 `typedef struct CyFx3BootDmaDescriptor_t CyFx3BootDmaDescriptor_t`

Descriptor data structure.

#### Description

This data structure contains the fields that make up a DMA descriptor on the FX3 device.

5.2.3.3 `typedef struct CyFx3BootDmaSocket_t CyFx3BootDmaSocket_t`

DMA socket configuration structure.

#### Description

This structure holds all the configuration fields that can be directly updated on a DMA socket. Refer to the `sock_regs.h` file for the detailed break up into bit-fields for each of the structure members.

5.2.3.4 `typedef enum CyFx3BootDmaSockId_t CyFx3BootDmaSockId_t`

DMA socket IDs for all sockets in the device.

#### Description

This is a software representation of all sockets on the device. The socket ID has two parts: IP number and socket number. Each peripheral (IP) has a fixed ID. LPP is 0, PIB is 1, Storage port is 2, USB egress is 3 and USB ingress is 4.

Each peripheral has a number of sockets. The LPP sockets are fixed and have to be used as defined. The PIB sockets 0-15 can be used as both producer and consumer, but the PIB sockets 16-31 are strictly producer sockets. The UIB sockets are defined as 0-15 producer and 0-15 consumer sockets. The CPU sockets are virtual representations and have no backing hardware block.

5.2.3.5 `typedef struct CyFx3BootDmaSockRegs_t CyFx3BootDmaSockRegs_t`

Actual socket specific register structure.

**Description**

This structure represents the actual socket status register space on the FX3 device. This includes a number of reserved and hardware-access only registers which are skipped in the [CyFx3BootDmaSocket\\_t](#) structure.

## 5.2.4 Enumeration Type Documentation

### 5.2.4.1 enum CyFx3BootDmaSockId\_t

DMA socket IDs for all sockets in the device.

**Description**

This is a software representation of all sockets on the device. The socket ID has two parts: IP number and socket number. Each peripheral (IP) has a fixed ID. LPP is 0, PIB is 1, Storage port is 2, USB egress is 3 and USB ingress is 4.

Each peripheral has a number of sockets. The LPP sockets are fixed and have to be used as defined. The P $\leftarrow$ IB sockets 0-15 can be used as both producer and consumer, but the PIB sockets 16-31 are strictly producer sockets. The UIB sockets are defined as 0-15 producer and 0-15 consumer sockets. The CPU sockets are virtual representations and have no backing hardware block.

Enumerator

**CY\_DMA\_LPP\_SOCKET\_I2S\_LEFT** Left channel output to I2S port.  
**CY\_DMA\_LPP\_SOCKET\_I2S\_RIGHT** Right channel output to I2S port.  
**CY\_DMA\_LPP\_SOCKET\_I2C\_CONS** Outgoing data to I2C slave.  
**CY\_DMA\_LPP\_SOCKET\_UART\_CONS** Outgoing data to UART peer.  
**CY\_DMA\_LPP\_SOCKET\_SPI\_CONS** Outgoing data to SPI slave.  
**CY\_DMA\_LPP\_SOCKET\_I2C\_PROD** Incoming data from I2C slave.  
**CY\_DMA\_LPP\_SOCKET\_UART\_PROD** Incoming data from UART peer.  
**CY\_DMA\_LPP\_SOCKET\_SPI\_PROD** Incoming data from SPI slave.  
**CY\_DMA\_PIB\_SOCKET\_0** P-port socket number 0.  
**CY\_DMA\_PIB\_SOCKET\_1** P-port socket number 1.  
**CY\_DMA\_PIB\_SOCKET\_2** P-port socket number 2.  
**CY\_DMA\_PIB\_SOCKET\_3** P-port socket number 3.  
**CY\_DMA\_PIB\_SOCKET\_4** P-port socket number 4.  
**CY\_DMA\_PIB\_SOCKET\_5** P-port socket number 5.  
**CY\_DMA\_PIB\_SOCKET\_6** P-port socket number 6.  
**CY\_DMA\_PIB\_SOCKET\_7** P-port socket number 7.  
**CY\_DMA\_PIB\_SOCKET\_8** P-port socket number 8.  
**CY\_DMA\_PIB\_SOCKET\_9** P-port socket number 9.  
**CY\_DMA\_PIB\_SOCKET\_10** P-port socket number 10.  
**CY\_DMA\_PIB\_SOCKET\_11** P-port socket number 11.  
**CY\_DMA\_PIB\_SOCKET\_12** P-port socket number 12.  
**CY\_DMA\_PIB\_SOCKET\_13** P-port socket number 13.  
**CY\_DMA\_PIB\_SOCKET\_14** P-port socket number 14.  
**CY\_DMA\_PIB\_SOCKET\_15** P-port socket number 15.  
**CY\_DMA\_PIB\_SOCKET\_16** P-port socket number 16.  
**CY\_DMA\_PIB\_SOCKET\_17** P-port socket number 17.  
**CY\_DMA\_PIB\_SOCKET\_18** P-port socket number 18.  
**CY\_DMA\_PIB\_SOCKET\_19** P-port socket number 19.

**CY\_DMA\_PIB\_SOCKET\_20** P-port socket number 20.  
**CY\_DMA\_PIB\_SOCKET\_21** P-port socket number 21.  
**CY\_DMA\_PIB\_SOCKET\_22** P-port socket number 22.  
**CY\_DMA\_PIB\_SOCKET\_23** P-port socket number 23.  
**CY\_DMA\_PIB\_SOCKET\_24** P-port socket number 24.  
**CY\_DMA\_PIB\_SOCKET\_25** P-port socket number 25.  
**CY\_DMA\_PIB\_SOCKET\_26** P-port socket number 26.  
**CY\_DMA\_PIB\_SOCKET\_27** P-port socket number 27.  
**CY\_DMA\_PIB\_SOCKET\_28** P-port socket number 28.  
**CY\_DMA\_PIB\_SOCKET\_29** P-port socket number 29.  
**CY\_DMA\_PIB\_SOCKET\_30** P-port socket number 30.  
**CY\_DMA\_PIB\_SOCKET\_31** P-port socket number 31.  
**CY\_DMA\_SIB\_SOCKET\_0** S-port socket number 0.  
**CY\_DMA\_SIB\_SOCKET\_1** S-port socket number 1.  
**CY\_DMA\_SIB\_SOCKET\_2** S-port socket number 2.  
**CY\_DMA\_SIB\_SOCKET\_3** S-port socket number 3.  
**CY\_DMA\_SIB\_SOCKET\_4** S-port socket number 4.  
**CY\_DMA\_SIB\_SOCKET\_5** S-port socket number 5.  
**CY\_DMA\_UIB\_SOCKET\_CONS\_0** U-port output socket number 0.  
**CY\_DMA\_UIB\_SOCKET\_CONS\_1** U-port output socket number 1.  
**CY\_DMA\_UIB\_SOCKET\_CONS\_2** U-port output socket number 2.  
**CY\_DMA\_UIB\_SOCKET\_CONS\_3** U-port output socket number 3.  
**CY\_DMA\_UIB\_SOCKET\_CONS\_4** U-port output socket number 4.  
**CY\_DMA\_UIB\_SOCKET\_CONS\_5** U-port output socket number 5.  
**CY\_DMA\_UIB\_SOCKET\_CONS\_6** U-port output socket number 6.  
**CY\_DMA\_UIB\_SOCKET\_CONS\_7** U-port output socket number 7.  
**CY\_DMA\_UIB\_SOCKET\_CONS\_8** U-port output socket number 8.  
**CY\_DMA\_UIB\_SOCKET\_CONS\_9** U-port output socket number 9.  
**CY\_DMA\_UIB\_SOCKET\_CONS\_10** U-port output socket number 10.  
**CY\_DMA\_UIB\_SOCKET\_CONS\_11** U-port output socket number 11.  
**CY\_DMA\_UIB\_SOCKET\_CONS\_12** U-port output socket number 12.  
**CY\_DMA\_UIB\_SOCKET\_CONS\_13** U-port output socket number 13.  
**CY\_DMA\_UIB\_SOCKET\_CONS\_14** U-port output socket number 14.  
**CY\_DMA\_UIB\_SOCKET\_CONS\_15** U-port output socket number 15.  
**CY\_DMA\_UIB\_SOCKET\_PROD\_0** U-port input socket number 0.  
**CY\_DMA\_UIB\_SOCKET\_PROD\_1** U-port input socket number 1.  
**CY\_DMA\_UIB\_SOCKET\_PROD\_2** U-port input socket number 2.  
**CY\_DMA\_UIB\_SOCKET\_PROD\_3** U-port input socket number 3.  
**CY\_DMA\_UIB\_SOCKET\_PROD\_4** U-port input socket number 4.  
**CY\_DMA\_UIB\_SOCKET\_PROD\_5** U-port input socket number 5.  
**CY\_DMA\_UIB\_SOCKET\_PROD\_6** U-port input socket number 6.  
**CY\_DMA\_UIB\_SOCKET\_PROD\_7** U-port input socket number 7.  
**CY\_DMA\_UIB\_SOCKET\_PROD\_8** U-port input socket number 8.  
**CY\_DMA\_UIB\_SOCKET\_PROD\_9** U-port input socket number 9.  
**CY\_DMA\_UIB\_SOCKET\_PROD\_10** U-port input socket number 10.

**CY\_DMA\_UIB\_SOCKET\_PROD\_11** U-port input socket number 11.  
**CY\_DMA\_UIB\_SOCKET\_PROD\_12** U-port input socket number 12.  
**CY\_DMA\_UIB\_SOCKET\_PROD\_13** U-port input socket number 13.  
**CY\_DMA\_UIB\_SOCKET\_PROD\_14** U-port input socket number 14.  
**CY\_DMA\_UIB\_SOCKET\_PROD\_15** U-port input socket number 15.  
**CY\_DMA\_CPU\_SOCKET\_CONS** Socket through which the FX3 CPU receives data.  
**CY\_DMA\_CPU\_SOCKET\_PROD** Socket through which the FX3 CPU produces data.

## 5.2.5 Function Documentation

### 5.2.5.1 void CyFx3BootDmaClearSockInterruptions ( **CyFx3BootDmaSockId\_t sockId**, **uint32\_t intrVal** )

Clear the specified socket interrupts.

#### Description

This function clears the specified interrupt on a DMA socket.

#### Parameters

<i>sockId</i>	Socket on which interrupts are to be cleared.
<i>intrVal</i>	Bitmask representing interrupts to be cleared.

### 5.2.5.2 CyFx3BootErrorCode\_t CyFx3BootDmaDisableSocket ( **CyFx3BootDmaSockId\_t sockId** )

Disable a DMA socket.

#### Description

This API forcibly disables the specified DMA socket. This can be used to ensure that all sockets start from a clean state, or as part of error handling.

#### Returns

CY\_FX3\_BOOT\_SUCCESS if the socket is disabled. CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT if the socket id is invalid.

#### See also

[CyFx3BootDmaEnableSocket](#)

#### Parameters

<i>sockId</i>	Id of socket to be disabled.
---------------	------------------------------

### 5.2.5.3 CyFx3BootErrorCode\_t CyFx3BootDmaEnableSocket ( **CyFx3BootDmaSockId\_t sockId** )

Enable a DMA socket.

#### Description

Enables the specified DMA socket for operation. Data transfers through the socket can happen once it is enabled. The typical sequence for starting up a DMA transfer is to configure the sockets using CyFx3BootDmaGetSocketConfig and CyFx3BootDmaSetSocketConfig, and then to enable the sockets using CyFx3BootDmaEnableSocket.

**Returns**

CY\_FX3\_BOOT\_SUCCESS if the socket was enabled. CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT if the socket id is invalid. CY\_FX3\_BOOT\_ERROR\_ALREADY\_STARTED if the socket is already enabled.

**See also**

[CyFx3BootDmaDisableSocket](#)

**Parameters**

<i>sock← Id</i>	Id of socket to be enabled.
---------------------	-----------------------------

#### 5.2.5.4 CyFx3BootErrorCode\_t CyFx3BootDmaGetDscrConfig ( uint16\_t *dscrIndex*, CyFx3BootDmaDescriptor\_t \* *dscr\_p* )

Get the current values from the specified DMA descriptor index.

**Description**

A DMA descriptor structure maintains information about a single DMA buffer on the FX3 device. An area of the FX3 device SRAM is reserved for DMA descriptor usage, and the address for each descriptor is a function of the descriptor index. This function fetches the current values from the specified descriptor index. This is typically called as part of a read-modify-write sequence to setup the descriptors.

**Note\*\***

Descriptor indices 0-3 are reserved and should not be used. The valid range is from 4 to 511.

**Returns**

CY\_FX3\_BOOT\_SUCCESS if the descriptor fetch is successful.

CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT if the descriptor index or *dscr\_p* parameter is invalid.

**See also**

[CyFx3BootDmaDescriptor\\_t](#)  
[CyFx3BootDmaSetDscrConfig](#)

**Parameters**

<i>dscrIndex</i>	Index of descriptor to be updated.
<i>dscr_p</i>	Structure to be filled with descriptor configuration.

#### 5.2.5.5 CyFx3BootErrorCode\_t CyFx3BootDmaGetSocketConfig ( CyFx3BootDmaSockId\_t *sockId*, CyFx3BootDmaSocket\_t \* *sock\_p* )

Get the current status of a DMA socket.

**Description**

A DMA socket is a construct that manages a unique data pipe between FX3 and an external device. All configuration and status information about a socket are stored in the form of a set of registers, and can be queried using this API. This API is typically called as part of a read-modify-write sequence to update the socket configuration.

**Returns**

CY\_FX3\_BOOT\_SUCCESS if the socket fetch is successful.

CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT if the socket id or *sock\_p* parameter is invalid.

**See also**

[CyFx3BootDmaSockId\\_t](#)  
[CyFx3BootDmaSocket\\_t](#)  
[CyFx3BootDmaSetSocketConfig](#)

**Parameters**

<i>sockId</i>	Id of socket to be retrieved.
<i>sock</i> <i>_p</i>	Structure into which the configuration is queried.

**5.2.5.6 uint32\_t CyFx3BootDmaGetSockInterrupts ( CyFx3BootDmaSockId\_t *sockId* )**

Get the current interrupt status for a given socket.

**Description**

This function gets the current interrupt status value for a given socket.

**Returns**

Current interrupt status value. Will be 0 if the socket is invalid.

**Parameters**

<i>sock</i> <i>Id</i>	Socket on which interrupts are to be queried.
--------------------------	---

**5.2.5.7 void CyFx3BootDmaRegisterCallback ( CyFx3BootDmaCallback\_t *cbFunc*, CyBool\_t *usbIntrEn*, CyBool\_t *pibIntrEn*, CyBool\_t *serialIntrEn* )**

Register a callback for notification of DMA interrupts.

**Description**

Register a callback function for notification of DMA interrupts from various FX3 blocks. The callback function pointer a set of block level interrupt enable flags are provided as parameters. The driver will only enable interrupts at the VIC level for blocks that have their interrupt enable flags set.

Note\*\*

The callback is responsible for clearing the interrupt. The ISR does not clear any interrupts, as this may cause some interrupts to get lost.

**See also**

[CyFx3BootDmaCallback\\_t](#)

**Parameters**

<i>cbFunc</i>	Callback function pointer.
<i>usbIntrEn</i>	Enable DMA interrupt for USB endpoints.
<i>pibIntrEn</i>	Enable DMA interrupt for the GPIF-II / PMMC block.
<i>serialIntrEn</i>	Enable DMA interrupt for the serial (SPI, I2C, I2S and UART) interface blocks.

**5.2.5.8 CyFx3BootErrorCode\_t CyFx3BootDmaSendSocketEvent ( CyFx3BootDmaSockId\_t *sockId*, uint16\_t *dscrIndex*, CyBool\_t *isOccupied* )**

Send an event to notify a socket that a descriptor that is waiting on has been modified.

#### Description

A data path through the FX3 device involves a pair of ingress (producer) and egress (consumer) sockets that are bound together with a set of DMA descriptors and buffers. The two sockets at the ends of the data path communicate through the shared descriptor structures which are maintained in memory. Once a socket has updated a DMA descriptor in memory, it notifies its peer socket by writing into its EVENT register.

In cases where the communication between the two sockets is being controlled by firmware (manual DMA channels), the sockets are not configured to send these EVENTS directly to the peer socket. Instead, the CPU gets notified through an interrupt; and then updates the socket by writing to the EVENT register using this API.

#### Returns

CY\_FX3\_BOOT\_SUCCESS if the event sending is successful.

CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT if the parameters are invalid.

#### Parameters

<i>sockId</i>	Socket to which the event has to be sent.
<i>dscrIndex</i>	Index of descriptor that has been updated.
<i>isOccupied</i>	Whether the descriptor has been marked as occupied (=1) or empty (=0).

**5.2.5.9 CyFx3BootErrorCode\_t CyFx3BootDmaSetDscrConfig ( uint16\_t *dscrIndex*, CyFx3BootDmaDescriptor\_t \* *dscr\_p* )**

Update the values of a DMA descriptor on the FX3 device.

#### Description

This function updates the values of a DMA descriptor from the contents of the structure parameter passed. This is typically called as part of a descriptor read-modify-write sequence.

Note\*\*

Descriptor indices 0-3 are reserved and should not be used. The valid range is from 4 to 511.

#### Returns

CY\_FX3\_BOOT\_SUCCESS if the descriptor fetch is successful.

CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT if the descriptor index or *dscr\_p* parameter is invalid.

#### See also

[CyFx3BootDmaDescriptor\\_t](#)

[CyFx3BootDmaGetDscrConfig](#)

#### Parameters

<i>dscrIndex</i>	Index of descriptor to be updated.
<i>dscr_p</i>	Structure containing new descriptor configuration.

**5.2.5.10 CyFx3BootErrorCode\_t CyFx3BootDmaSetSocketConfig ( CyFx3BootDmaSockId\_t *sockId*, CyFx3BootDmaSocket\_t \* *sock\_p* )**

Update the configuration of a DMA socket.

**Description**

This API updates the configuration of a DMA socket based on the contents of the structure parameter. This is typically called as part of a read-modify-write sequence to update the socket configuration. It is also possible to set the ENABLE bit in the socket as part of this API and avoid calling CyFx3BootDmaEnableSocket.

Note\*\*

Please note that the `sock_p->intr` value is a write one to clear setting. Calling this API with a non-zero interrupt intr value will cause some interrupts to (unexpectedly) be cleared.

**Returns**

`CY_FX3_BOOT_SUCCESS` if the socket update is successful.

`CY_FX3_BOOT_ERROR_BAD_ARGUMENT` if the socket id or `sock_p` parameter is invalid.

`CY_FX3_BOOT_ERROR_ALREADY_STARTED` if a socket is being updated while it is in the active state.

**See also**

[CyFx3BootDmaSockId\\_t](#)

[CyFx3BootDmaSocket\\_t](#)

[CyFx3BootDmaGetSocketConfig](#)

**Parameters**

<code>sockId</code>	Id of socket to be updated.
<code>sock_p</code>	Structure containing new socket configuration.

**5.2.5.11 CyFx3BootErrorCode\_t CyFx3BootDmaWrapSocket ( `CyFx3BootDmaSockId_t sockId` )**

Forcibly commit the currently available data in an ingress socket.

**Description**

Data received into an ingress (data incoming) socket on FX3 will only be committed (made available to CPU or for sending out from the device) when the buffer has been filled, or when an end of packet (buffer) signal is received. In some cases, neither of these events may happen for a long time; leaving a buffer partially filled with data. This API can be used to forcibly wrap-up (commit) the available data on an ingress socket.

**Returns**

`CY_FX3_BOOT_SUCCESS` if the socket wrap-up is successful.

`CY_FX3_BOOT_ERROR_BAD_ARGUMENT` if the `sockId` parameter is invalid.

**Parameters**

<code>sockId</code>	Id of socket to be wrapped up.
---------------------	--------------------------------

**5.3 firmware/boot\_fw/include/cyfx3error.h File Reference**

This file lists the various error codes that can be returned by the APIs in the FX3 boot library.

```
#include <cyu3externcstart.h>
#include <cyu3externcend.h>
```

## Typedefs

- **typedef enum CyFx3BootErrorCode\_t CyFx3BootErrorCode\_t**  
*List of error codes returned by the boot APIs.*

## Enumerations

- **enum CyFx3BootErrorCode\_t {**  
**CY\_FX3\_BOOT\_SUCCESS** = 0x0, **CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT**, **CY\_FX3\_BOOT\_ERROR\_NULL\_POINTER**, **CY\_FX3\_BOOT\_ERROR\_TIMEOUT**,  
**CY\_FX3\_BOOT\_ERROR\_NOT\_SUPPORTED**, **CY\_FX3\_BOOT\_ERROR\_NOT\_CONFIGURED**, **CY\_FX3\_BOOT\_ERROR\_BAD\_DESCRIPTOR\_TYPE**, **CY\_FX3\_BOOT\_ERROR\_XFER\_FAILURE**,  
**CY\_FX3\_BOOT\_ERROR\_NO\_REENUM\_REQUIRED**, **CY\_FX3\_BOOT\_ERROR\_NOT\_STARTED**, **CY\_FX3\_BOOT\_ERROR\_MEMORY\_ERROR**, **CY\_FX3\_BOOT\_ERROR\_ABORTED**,  
**CY\_FX3\_BOOT\_ERROR\_INVALID\_DMA\_ADDR**, **CY\_FX3\_BOOT\_ERROR\_ALREADY\_STARTED**, **CY\_FX3\_BOOT\_ERROR\_FAILURE**, **CY\_FX3\_BOOT\_ERROR\_I2C** }

*List of error codes returned by the boot APIs.*

### 5.3.1 Detailed Description

This file lists the various error codes that can be returned by the APIs in the FX3 boot library.

### 5.3.2 Enumeration Type Documentation

#### 5.3.2.1 enum CyFx3BootErrorCode\_t

List of error codes returned by the boot APIs.

##### Enumerator

- CY\_FX3\_BOOT\_SUCCESS** Success.
- CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT** One or more parameters to a function are invalid.
- CY\_FX3\_BOOT\_ERROR\_NULL\_POINTER** A null pointer has been passed in unexpectedly.
- CY\_FX3\_BOOT\_ERROR\_TIMEOUT** Timeout on relevant operation.
- CY\_FX3\_BOOT\_ERROR\_NOT\_SUPPORTED** Operation requested is not supported in current mode.
- CY\_FX3\_BOOT\_ERROR\_NOT\_CONFIGURED** The peripheral block being accessed has not been configured.
- CY\_FX3\_BOOT\_ERROR\_BAD\_DESCRIPTOR\_TYPE** Invalid USB descriptor type.
- CY\_FX3\_BOOT\_ERROR\_XFER\_FAILURE** Data Transfer failed.
- CY\_FX3\_BOOT\_ERROR\_NO\_REENUM\_REQUIRED** Indicates that the booter has successfully configured the FX3 device after control was transferred from the full firmware application. The user need not go through the cycle of setting the descriptors and issuing connect call if this error code is returned by the CyFx3BootUsbStart () API.
- CY\_FX3\_BOOT\_ERROR\_NOT\_STARTED** Indicates that the block being configured has not been initialized.
- CY\_FX3\_BOOT\_ERROR\_MEMORY\_ERROR** Indicates that the API was unable to find enough memory to copy the descriptor set through the [CyFx3BootUsbSetDesc\(\)](#) API.
- CY\_FX3\_BOOT\_ERROR\_ABORTED** Indicates that the USB control request being processed has been aborted due to a USB reset or a new control request.
- CY\_FX3\_BOOT\_ERROR\_INVALID\_DMA\_ADDR** Indicates that the address passed into a DMA transfer API is invalid (could be a TCM address).

**CY\_FX3\_BOOT\_ERROR\_ALREADY\_STARTED** Indicates that a block that is already ON is being initialized again.

**CY\_FX3\_BOOT\_ERROR\_FAILURE** Generic error code indicating any other failure.

**CY\_FX3\_BOOT\_ERROR\_I2C** I2C specific error code.

## 5.4 firmware/boot-fw/include/cyfx3gpio.h File Reference

The file defines the data structures and APIs that are provided for making use of FX3 GPIOs.

```
#include <cyu3types.h>
#include <cyfx3error.h>
#include <cyfx3device.h>
#include <cyu3externcstart.h>
#include <cyu3externcend.h>
```

### Data Structures

- struct [CyFx3BootGpioSimpleConfig\\_t](#)  
*Configuration information for simple GPIOs.*

### Typedefs

- typedef enum [CyFx3BootGpioloMode\\_t](#) [CyFx3BootGpioloMode\\_t](#)  
*List of IO modes.*
- typedef enum [CyFx3BootGpioIntrMode\\_t](#) [CyFx3BootGpioIntrMode\\_t](#)  
*List of GPIO interrupt modes.*
- typedef struct [CyFx3BootGpioSimpleConfig\\_t](#) [CyFx3BootGpioSimpleConfig\\_t](#)  
*Configuration information for simple GPIOs.*

### Enumerations

- enum [CyFx3BootGpioloMode\\_t](#) { [CY\\_FX3\\_GPIO\\_IO\\_MODE\\_NONE](#) = 0, [CY\\_FX3\\_GPIO\\_IO\\_MODE\\_WPU](#), [CY\\_FX3\\_GPIO\\_IO\\_MODE\\_WPD](#) }  
*List of IO modes.*
- enum [CyFx3BootGpioIntrMode\\_t](#) {  
[CY\\_FX3\\_BOOT\\_GPIO\\_NO\\_INTR](#) = 0, [CY\\_FX3\\_BOOT\\_GPIO\\_INTR\\_POS\\_EDGE](#), [CY\\_FX3\\_BOOT\\_GPIO\\_INTR\\_NEG\\_EDGE](#), [CY\\_FX3\\_BOOT\\_GPIO\\_INTR\\_BOTH\\_EDGE](#),  
[CY\\_FX3\\_BOOT\\_GPIO\\_INTR\\_LOW\\_LEVEL](#), [CY\\_FX3\\_BOOT\\_GPIO\\_INTR\\_HIGH\\_LEVEL](#) }  
*List of GPIO interrupt modes.*

### Functions

- void [CyFx3BootGpioInit](#) (void)  
*Initializes the GPIO block.*
- void [CyFx3BootGpioDeInit](#) (void)  
*De-initializes the GPIO block.*
- [CyFx3BootErrorCode\\_t](#) [CyFx3BootGpioSetSimpleConfig](#) (uint8\_t gpioid, [CyFx3BootGpioSimpleConfig\\_t](#) \*cfg\_p)  
*Configures a simple GPIO.*
- void [CyFx3BootGpioDisable](#) (uint8\_t gpioid)

*Disables a GPIO pin.*

- [CyFx3BootErrorCode\\_t CyFx3BootGpioGetValue](#) (uint8\_t gpioid, [CyBool\\_t](#) \*value\_p)  
*Query the state of GPIO input.*
- [CyFx3BootErrorCode\\_t CyFx3BootGpioSetValue](#) (uint8\_t gpioid, [CyBool\\_t](#) value)  
*Set the state of GPIO pin output.*
- [CyFx3BootErrorCode\\_t CyFx3BootGpioSetIoMode](#) (uint8\_t gpioid, [CyFx3BootGpioIoMode\\_t](#) ioMode)  
*Set IO mode for the selected GPIO.*

## 5.4.1 Detailed Description

The file defines the data structures and APIs that are provided for making use of FX3 GPIOs.

### Description

Any unused pin on the FX3 device can be used as a GPIO. The Boot API library only allows for these pins to be used as standard input or output pins.

## 5.4.2 Typedef Documentation

### 5.4.2.1 [typedef enum CyFx3BootGpioIntrMode\\_t CyFx3BootGpioIntrMode\\_t](#)

List of GPIO interrupt modes.

### Description

FX3 GPIOs can be configured to trigger interrupts when a desired transition or signal level is detected on the input signal.

Note\*\*

Interrupts are not supported in this release of booter. The interrupt mode for all GPIOs need to be set to CY\_FX3\_BOOT\_GPIO\_NO\_INTR.

### 5.4.2.2 [typedef enum CyFx3BootGpioIoMode\\_t CyFx3BootGpioIoMode\\_t](#)

List of IO modes.

### Description

The FX3 device supports internal weak pull-ups or pull-downs on its GPIOs. These terminations can be used even on signals that are not used as GPIOs.

This type lists the possible internal IO modes that can be selected for a FX3 GPIO.

### See also

[CyU3PGpioSetIoMode](#)

### 5.4.2.3 [typedef struct CyFx3BootGpioSimpleConfig\\_t CyFx3BootGpioSimpleConfig\\_t](#)

Configuration information for simple GPIOs.

### Description

This structure encapsulates all of the configuration information for a simple GPIO on the FX3 device.

If the pin is configured as input, then the fields driveLowEn and driveHighEn should be CyFalse. The outValue field is a don't care in this case.

If the pin is configured as an output, the inputEn field should be CyFalse. The driveLowEn and driveHighEn fields can be used to select whether the device should drive the pin to low/high levels or just tri-state it.

See also

[CyFx3BootGpioIntrMode\\_t](#)

### 5.4.3 Enumeration Type Documentation

#### 5.4.3.1 enum CyFx3BootGpioIntrMode\_t

List of GPIO interrupt modes.

##### Description

FX3 GPIOs can be configured to trigger interrupts when a desired transition or signal level is detected on the input signal.

Note\*\*

Interrupts are not supported in this release of booter. The interrupt mode for all GPIOs need to be set to CY\_FX3\_BOOT\_GPIO\_NO\_INTR.

Enumerator

**CY\_FX3\_BOOT\_GPIO\_NO\_INTR** No Interrupt is triggered.

**CY\_FX3\_BOOT\_GPIO\_INTR\_POS\_EDGE** Interrupt is triggered on positive edge of input.

**CY\_FX3\_BOOT\_GPIO\_INTR\_NEG\_EDGE** Interrupt is triggered on negative edge of input.

**CY\_FX3\_BOOT\_GPIO\_INTR\_BOTH\_EDGE** Interrupt is triggered on both edges of input.

**CY\_FX3\_BOOT\_GPIO\_INTR\_LOW\_LEVEL** Interrupt is triggered when input is low.

**CY\_FX3\_BOOT\_GPIO\_INTR\_HIGH\_LEVEL** Interrupt is triggered when input is high.

#### 5.4.3.2 enum CyFx3BootGpioloMode\_t

List of IO modes.

##### Description

The FX3 device supports internal weak pull-ups or pull-downs on its GPIOs. These terminations can be used even on signals that are not used as GPIOs.

This type lists the possible internal IO modes that can be selected for a FX3 GPIO.

See also

[CyU3PGpioSetIoMode](#)

Enumerator

**CY\_FX3\_GPIO\_IO\_MODE\_NONE** No internal pull-up or pull-down. Default condition.

**CY\_FX3\_GPIO\_IO\_MODE\_WPU** A weak pull-up is provided on the IO.

**CY\_FX3\_GPIO\_IO\_MODE\_WPD** A weak pull-down is provided on the IO.

### 5.4.4 Function Documentation

#### 5.4.4.1 void CyFx3BootGpioDelInit( void )

De-initializes the GPIO block.

##### Description

This function powers off the GPIO block on the FX3 device.

##### Return value

None

See also

[CyFx3BootGpioInit](#)

#### 5.4.4.2 void CyFx3BootGpioDisable ( uint8\_t *gpioid* )

Disables a GPIO pin.

##### Description

This function is used to disable the use of a pin as a GPIO. Both input and output stages on the pin will be disabled after calling this API.

##### Return value

None

See also

[CyFx3BootGpioSetSimpleConfig](#)

##### Parameters

<i>gpiod</i> ←	GPIO ID to be disabled
<i>Id</i>	

#### 5.4.4.3 CyFx3BootErrorCode\_t CyFx3BootGpioGetValue ( uint8\_t *gpioid*, CyBool\_t \* *value\_p* )

Query the state of GPIO input.

##### Description

Get the current signal level on a GPIO input pin. This API can also be used to retrieve the current value being driven out on a output pin.

##### Return value

CY\_FX3\_BOOT\_SUCCESS - if the operation is successful.  
 CY\_FX3\_BOOT\_ERROR\_NULL\_POINTER - if *value\_p* is NULL.  
 CY\_FX3\_BOOT\_ERROR\_NOT\_CONFIGURED - if the pin is not selected as a GPIO in the IOMATRIX.

See also

[CyFx3BootGpioSetSimpleConfig](#)  
[CyFx3BootGpioSetValue](#)

##### Parameters

<i>gpiod</i>	GPIO id to be queried.
<i>value_p</i> ←	Output parameter that will be filled with the GPIO value.

#### 5.4.4.4 void CyFx3BootGpioInit ( void )

Initializes the GPIO block.

##### Description

This function should be called before any other GPIO APIs can be called. This powers up the GPIO block and sets up the API data structures.

**Return value**

None

**See also**

[CyFx3BootGpioDelInit](#)

**5.4.4.5 CyFx3BootErrorCode\_t CyFx3BootGpioSetIoMode ( uint8\_t gpioid, CyFx3BootGpioIoMode\_t ioMode )**

Set IO mode for the selected GPIO.

**Description**

This function enables or disables internal pull-ups/pull-downs on the selected FX3 IO pin. This IO mode change will take about 5 us to become active.

**Return value**

CY\_FX3\_BOOT\_SUCCESS - If the operation is successful.

CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT - If the gpioid or ioMode requested is invalid.

**See also**

[CyFx3BootGpioIoMode\\_t](#)

**Parameters**

<i>gpioid</i>	GPIO Pin to be updated.
<i>ioMode</i>	Desired pull-up/pull-down mode.

**5.4.4.6 CyFx3BootErrorCode\_t CyFx3BootGpioSetSimpleConfig ( uint8\_t gpioid, CyFx3BootGpioSimpleConfig\_t \* cfg\_p )**

Configures a simple GPIO.

**Description**

This function is used to configure and enable a simple GPIO pin. This function needs to be called before using the simple GPIO.

**Return value**

CY\_FX3\_BOOT\_SUCCESS - if the operation is successful.

CY\_FX3\_BOOT\_ERROR\_NULL\_POINTER - if cfg\_p is NULL.

CY\_FX3\_BOOT\_ERROR\_NOT\_CONFIGURED - if the pin has not been selected as a GPIO in the IOMATRIX.

**See also**

[CyFx3BootGpioSimpleConfig\\_t](#)

[CyFx3BootDeviceConfigureIOMatrix](#)

[CyFx3BootGpioDisable](#)

**Parameters**

<i>gpioid</i>	GPIO id to be configured.
<i>cfg_p</i>	Configuration parameters for the GPIO.

#### 5.4.4.7 CyFx3BootErrorCode\_t CyFx3BootGpioSetValue ( uint8\_t gpioid, CyBool\_t value )

Set the state of GPIO pin output.

##### Description

This function updates the output state of a GPIO pin, and is valid only for GPIOs configured as output. The output value is updated by this function, but the driveLowEn and driveHighEn configuration parameters will determine whether the pin is driven actively or tri-stated.

##### Return value

CY\_FX3\_BOOT\_SUCCESS - if the operation is successful.

CY\_FX3\_BOOT\_ERROR\_NOT\_CONFIGURED - if the pin is not selected as a GPIO in the IOMATRIX.

##### See also

[CyFx3BootGpioSetSimpleConfig](#)  
[CyFx3BootGpioGetValue](#)

##### Parameters

<i>gpiod</i>	GPIO id to be modified.
<i>value</i>	Value to set on the GPIO pin.

## 5.5 firmware/boot\_fw/include/cyfx3i2c.h File Reference

The I2C interface driver in the FX3 Boot Firmware provides a set of APIs that allow the configuration of the I2C interface properties and data exchange with one or more I2C slave devices.

```
#include <cyfx3error.h>
#include <cyu3types.h>
#include <cyu3externcstart.h>
#include <cyu3externcend.h>
```

## Data Structures

- struct [CyFx3BootI2cConfig\\_t](#)  
*Structure defining the configuration of the I2C interface.*
- struct [CyFx3BootI2cPreamble\\_t](#)  
*Structure defining the preamble to be sent on the I2C interface.*

## TypeDefs

- typedef struct [CyFx3BootI2cConfig\\_t](#) [CyFx3BootI2cConfig\\_t](#)  
*Structure defining the configuration of the I2C interface.*
- typedef struct [CyFx3BootI2cPreamble\\_t](#) [CyFx3BootI2cPreamble\\_t](#)  
*Structure defining the preamble to be sent on the I2C interface.*

## Functions

- [CyFx3BootErrorCode\\_t CyFx3BootI2cInit](#) (void)  
*Starts the I2C interface block.*

- void [CyFx3BootI2cDeInit](#) (void)  
*Stops the I2C module.*
- [CyFx3BootErrorCode\\_t CyFx3BootI2cSetConfig](#) ([CyFx3BootI2cConfig\\_t](#) \*config)  
*Sets the I2C interface parameters.*
- [CyFx3BootErrorCode\\_t CyFx3BootI2cSendCommand](#) ([CyFx3BootI2cPreamble\\_t](#) \*preamble, [uint32\\_t](#) byteCount, [CyBool\\_t](#) isRead)  
*Perform a read or write operation to the I2C slave.*
- [CyFx3BootErrorCode\\_t CyFx3BootI2cTransmitBytes](#) ([CyFx3BootI2cPreamble\\_t](#) \*preamble, [uint8\\_t](#) \*data, [uint32\\_t](#) byteCount, [uint32\\_t](#) retryCount)  
*Writes a small number of bytes to an I2C slave.*
- [CyFx3BootErrorCode\\_t CyFx3BootI2cReceiveBytes](#) ([CyFx3BootI2cPreamble\\_t](#) \*preamble, [uint8\\_t](#) \*data, [uint32\\_t](#) byteCount, [uint32\\_t](#) retryCount)  
*Reads a small number of bytes from an I2C slave.*
- [CyFx3BootErrorCode\\_t CyFx3BootI2cWaitForAck](#) ([CyFx3BootI2cPreamble\\_t](#) \*preamble, [uint32\\_t](#) retryCount)  
*Poll an I2C slave until all of the preamble is ACKed.*
- [CyFx3BootErrorCode\\_t CyFx3BootI2cDmaXferData](#) ([CyBool\\_t](#) isRead, [uint32\\_t](#) address, [uint32\\_t](#) length, [uint32\\_t](#) timeout)  
*This function is used to setup a dma from CPU to I2C or vice versa.*
- void [CyFx3BootI2cSetTimeout](#) ([uint32\\_t](#) timeout)  
*Set the timeout duration for I2C byte mode data transfers.*

### 5.5.1 Detailed Description

The I2C interface driver in the FX3 Boot Firmware provides a set of APIs that allow the configuration of the I2C interface properties and data exchange with one or more I2C slave devices.

### 5.5.2 Typedef Documentation

#### 5.5.2.1 [typedef struct CyFx3BootI2cConfig\\_t CyFx3BootI2cConfig\\_t](#)

Structure defining the configuration of the I2C interface.

##### Description

This structure encapsulates all of the configurable parameters that can be selected for the I2C interface. The [CyFx3BootI2cSetConfig\(\)](#) function accepts a pointer to this structure, and updates all of the interface parameters.

The I2C block can function in the bit rate range of 100 KHz to 1MHz. In default mode of operation, the timeouts need to be kept disabled.

##### See also

[CyFx3BootI2cSetConfig](#)

#### 5.5.2.2 [typedef struct CyFx3BootI2cPreamble\\_t CyFx3BootI2cPreamble\\_t](#)

Structure defining the preamble to be sent on the I2C interface.

##### Description

All I2C data transfers start with a preamble where the first byte contains the slave address and the direction of transfer. The preamble can optionally contain other bytes where device specific address values or other commands are sent to the slave device.

The FX3 device supports associating a preamble with a maximum length of 8 bytes to any I2C data transfer. This allows the user to specify a multi-byte preamble which covers the slave address, device specific address fields and then initiate the data transfer.

The ctrlMask indicate the start / stop bit conditions after each byte of the preamble.

For example, an I2C EEPROM read requires the byte address for the read to be written first. These two I2C operations can be combined into one I2C API call using the parameters of the structure.

Typical I2C EEPROM page Read operation:

Byte 0:

Bit 7 - 1: Slave address.

Bit 0 : 0 - Indicating this is a write from master.

Byte 1, 2: Address to which the data has to be written.

Byte 3:

Bit 7 - 1: Slave address.

Bit 0 : 1 - Indicating this is a read operation.

The buffer field shall hold the above four bytes, the length field shall be four; and ctrlMask field will be 0x0004 as a start bit is required after the third byte (third bit is set).

Typical I2C EEPROM page Write operation:

Byte 0:

Bit 7 - 1: Slave address.

Bit 0 : 0 - Indicating this is a write from master.

Byte 1, 2: Address to which the data has to be written.

The buffer field shall hold the above three bytes, the length field shall be three, and the ctrlMask field is zero as no additional start/stop conditions are needed.

See also

[CyFx3BootI2cSetConfig](#)

### 5.5.3 Function Documentation

#### 5.5.3.1 void CyFx3BootI2cDeInit ( void )

Stops the I2C module.

##### Description

This function disables and powers off the I2C interface. This function can be used to shut off the interface to save power when it is not in use.

##### Return value

None

See also

[CyFx3BootI2cInit](#)

#### 5.5.3.2 CyFx3BootErrorCode\_t CyFx3BootI2cDmaXferData ( CyBool\_t isRead, uint32\_t address, uint32\_t length, uint32\_t timeout )

This function is used to setup a dma from CPU to I2C or vice versa.

##### Description

This function is used to read/write data from/to an I2C slave when the I2C interface is configured in DMA mode. The CyFx3BootI2cSendCommand API should be used to address the slave and initiate the data transfer on the slave side.

This function is a blocking call which waits until the desired transfer is complete or the specified timeout duration has elapsed.

It is expected that the length of data being transferred is a multiple of 16 bytes.

**Return value**

CY\_FX3\_BOOT\_SUCCESS - if the data transfer is successful.  
CY\_FX3\_BOOT\_ERROR\_INVALID\_DMA\_ADDR - if the address specified is not in the SYSMEM area.  
CY\_FX3\_BOOT\_ERROR\_XFER\_FAILURE - if the data transfer encountered any error.  
CY\_FX3\_BOOT\_ERROR\_TIMEOUT - if the data transfer times out.

**See also**

[CyFx3BootI2cSetConfig](#)  
[CyFx3BootI2cSendCommand](#)

**Parameters**

<i>isRead</i>	isRead=CyTrue for read transfers, and isRead=CyFalse for write transfers.
<i>address</i>	Address of the buffer from/to which data is to be transferred
<i>length</i>	Length of the data to be transferred. Should be a multiple of 16 bytes and is limited by the amount of data that the slave can transfer without delays.
<i>timeout</i>	Timeout duration in multiples of 10 us. Can be set to CY_FX3_BOOT_WAIT_FOREVER to wait until the transfer is complete.

**5.5.3.3 CyFx3BootErrorCode\_t CyFx3BootI2cInit( void )**

Starts the I2C interface block.

**Description**

This function powers up the I2C interface block on the FX3 device and is expected to be the first I2C API function that is called by the application. IO configuration function is expected to have been called prior to this call.

This function also sets up the I2C interface at a default bit rate of 100KHz.

**Return value**

CY\_FX3\_BOOT\_SUCCESS - If the block is initialized successfully.

**See also**

[CyFx3BootI2cSetConfig](#)  
[CyFx3BootI2cDeinit](#)

**5.5.3.4 CyFx3BootErrorCode\_t CyFx3BootI2cReceiveBytes( CyFx3BootI2cPreamble\_t \* preamble, uint8\_t \* data, uint32\_t byteCount, uint32\_t retryCount )**

Reads a small number of bytes from an I2C slave.

**Description**

This function reads a few bytes one at a time from the I2C slave selected through the preamble parameter. The I2C interface should be configured in register (non-DMA) mode to make use of this function.

**Return value**

CY\_FX3\_BOOT\_SUCCESS - if the TransmitBytes is successful.  
CY\_FX3\_BOOT\_ERROR\_NULL\_POINTER - if the preamble or data is NULL.  
CY\_FX3\_BOOT\_ERROR\_XFER\_FAILURE - if the transfer fails due to a bus error.  
CY\_FX3\_BOOT\_ERROR\_TIMEOUT - if the transfer times out.

**See also**

[CyFx3BootI2cSetConfig](#)  
[CyFx3BootI2cTransmitBytes](#)

### Parameters

<i>preamble</i>	Preamble to be sent out before the data transfer.
<i>data</i>	Pointer to buffer where the data is to be placed.
<i>byteCount</i>	Size of the transfer in bytes.
<i>retryCount</i>	Number of times to retry request if preamble is NAKed or an error is encountered.

### 5.5.3.5 CyFx3BootErrorCode\_t CyFx3BootI2cSendCommand ( *CyFx3BootI2cPreamble\_t \*preamble*, *uint32\_t byteCount*, *CyBool\_t isRead* )

Perform a read or write operation to the I2C slave.

#### Description

This function is used to send the extended preamble over the I2C bus, and is used to initiate a transfer when the I2C block is configured for DMA mode of transfer.

This function is used internally by the CyFx3BootI2cReceiveBytes and CyFx3BootI2cTransmitBytes APIs, and should not be called directly when the register mode of transfer is selected.

#### Return value

CY\_FX3\_BOOT\_SUCCESS - When the SendCommand is successful.

CY\_FX3\_BOOT\_ERROR\_NULL\_POINTER - When the preamble is NULL.

CY\_FX3\_BOOT\_ERROR\_TIMEOUT - When the transfer times out.

#### See also

[CyFx3BootI2cReceiveBytes](#)  
[CyFx3BootI2cTransmitBytes](#)

### Parameters

<i>preamble</i>	Preamble information to be sent out.
-----------------	--------------------------------------

**Parameters**

<i>config</i>	I2C configuration settings.
---------------	-----------------------------

**5.5.3.7 void CyFx3BootI2cSetTimeout ( uint32\_t *timeout* )**

Set the timeout duration for I2C byte mode data transfers.

**Description**

The I2C byte mode transfer functions (CyFx3BootI2cSendCommand, CyFx3BootI2cTransmitBytes and CyFx3BootI2cReceiveBytes) are implemented using a poll loop which continually checks for the completion of the transfer. The delay for each of these loops will be approximately 2 us, and the default value of the maximum loop count is 4095. This API can be used to change the timeout loop count to a different value.

**Return value**

None

**5.5.3.8 CyFx3BootErrorCode\_t CyFx3BootI2cTransmitBytes ( CyFx3BootI2cPreamble\_t \* *preamble*, uint8\_t \* *data*, uint32\_t *byteCount*, uint32\_t *retryCount* )**

Writes a small number of bytes to an I2C slave.

**Description**

This function is used to write data one byte at a time to an I2C slave. This function requires that the I2C interface be configured in register (non-DMA) mode.

If a non-zero retryCount is specified, the API will retry a failing transfer until it succeeds or the specified count has been reached.

**Return value**

CY\_FX3\_BOOT\_SUCCESS - if the TransmitBytes is successful.  
 CY\_FX3\_BOOT\_ERROR\_NULL\_POINTER - if the preamble or data is NULL.  
 CY\_FX3\_BOOT\_ERROR\_XFER\_FAILURE - if the transfer fails due to a bus error.  
 CY\_FX3\_BOOT\_ERROR\_TIMEOUT - if the transfer times out.

**See also**

[CyFx3BootI2cSetConfig](#)  
[CyFx3BootI2cReceiveBytes](#)

**Parameters**

<i>preamble</i>	Preamble to be sent out before the data transfer.
<i>data</i>	Pointer to buffer containing data to be written.
<i>byteCount</i>	Size of the transfer in bytes.
<i>retryCount</i>	Number of times to retry request if a byte is NAKed by the slave.

**5.5.3.9 CyFx3BootErrorCode\_t CyFx3BootI2cWaitForAck ( CyFx3BootI2cPreamble\_t \* *preamble*, uint32\_t *retryCount* )**

Poll an I2C slave until all of the preamble is ACKed.

**Description**

This function waits for a ACK handshake from the slave, and can be used to ensure that the slave device has reached a desired state before issuing the next transaction or shutting the interface down. This function call returns

when the specified handshake has been received or when the wait has timed out. The function call can be repeated on CY\_FX3\_BOOT\_ERROR\_TIMEOUT without any error recovery.

#### Return value

CY\_FX3\_BOOT\_SUCCESS - if the device ACKs the preamble.  
 CY\_FX3\_BOOT\_ERROR\_NULL\_POINTER - if the preamble is NULL.  
 CY\_FX3\_BOOT\_ERROR\_TIMEOUT - if a timeout occurs.

#### Parameters

<i>preamble</i>	Preamble to be sent out for polling the slave.
<i>retryCount</i>	Number of times to retry request if preamble is NAKed or an error is encountered.

## 5.6 firmware/boot-fw/include/cyfx3pib.h File Reference

The PIB driver module in the FX3 boot firmware enables the PMMC interface on the FX3 device and allows PMMC data transfers.

```
#include <cyfx3error.h>
#include <cyu3types.h>
#include <cyfx3device.h>
#include <cyu3externcstart.h>
#include <cyu3externcend.h>
```

## Data Structures

- struct [CyFx3BootPibClock\\_t](#)  
*Clock configuration information for the PIB block.*
- struct [CyU3PGpifWaveData](#)  
*Information on a single GPIF transition from one state to another.*
- struct [CyU3PGpifConfig\\_t](#)  
*Structure that holds all configuration inputs for the GPIF hardware.*

## Macros

- #define [CYFX3\\_GPIF\\_NUM\\_STATES](#) (256)  
*Number of states supported by the GPIF hardware.*
- #define [CYFX3\\_GPIF\\_NUM\\_TRANS\\_FNS](#) (32)  
*Number of distinct transfer functions supported by the GPIF hardware.*
- #define [CYFX3\\_GPIF\\_INVALID\\_STATE](#) (0xFFFF)  
*Invalid state index for use in state machine control functions.*
- #define [CYFX3\\_PIB\\_THREAD\\_COUNT](#) (4)  
*Number of DMA threads on the GPIF (P-Port)*
- #define [CYFX3\\_PIB\\_SOCKET\\_COUNT](#) (32)  
*Number of DMA sockets on the GPIF (P-port)*
- #define [CYFX3\\_PIB\\_MAX\\_BURST\\_SETTING](#) (14)  
*Maximum burst size allowed for P-port sockets. The constant corresponds to Log(2) of size, which means that the max. size is 16 KB.*

## Typedefs

- **typedef struct CyFx3BootPibClock\_t CyFx3BootPibClock\_t**  
*Clock configuration information for the PIB block.*
- **typedef enum CyFx3BootPMMCEvent\_t CyFx3BootPMMCEvent\_t**  
*List of MMC-Slave Event notifications.*
- **typedef void(\* CyFx3BootPMMCIntrCb\_t) (CyFx3BootPMMCEvent\_t cbType, uint32\_t cbArg)**  
*Callback function type used for MMC-Slave event notifications.*
- **typedef void(\* CyFx3BootGpifIntrCb\_t) (uint8\_t currentState)**  
*Callback function type to be called when a GPIF state machine interrupt is triggered.*
- **typedef struct CyU3PGpifWaveData CyU3PGpifWaveData**  
*Information on a single GPIF transition from one state to another.*
- **typedef struct CyU3PGpifConfig\_t CyU3PGpifConfig\_t**  
*Structure that holds all configuration inputs for the GPIF hardware.*
- **typedef enum CyFx3GpifCounterType CyFx3GpifCounterType**  
*List of GPIF counters.*

## Enumerations

- **enum CyFx3BootPMMCEvent\_t {**  
**CYFX3\_PMMC\_GOIDLE\_CMD = 0, CYFX3\_PMMC\_CMD5\_SLEEP, CYFX3\_PMMC\_CMD5\_AWAKE, CYFX3\_PMMC\_CMD6\_SWITCH,**  
**CYFX3\_PMMC\_CMD12\_STOP, CYFX3\_PMMC\_CMD15\_INACTIVE, CYFX3\_PMMC\_SOCKET\_NOT\_READY, CYFX3\_PMMC\_CMD7\_SELECT }**  
*List of MMC-Slave Event notifications.*
- **enum CyFx3GpifCounterType { CYFX3\_GPIF\_COUNTER\_CONTROL = 0, CYFX3\_GPIF\_COUNTER\_ADDRESS, CYFX3\_GPIF\_COUNTER\_DATA }**  
*List of GPIF counters.*

## Functions

- **CyFx3BootErrorCode\_t CyFx3BootPibInit (CyFx3BootPibClock\_t \*clkInfo\_p, CyBool\_t isMmcMode)**  
*Starts the Processor Interface Block.*
- **CyFx3BootErrorCode\_t CyFx3BootPibDeinit (void)**  
*Stops the Processor Interface Block.*
- **void CyFx3BootPibRegisterMmcCallback (CyFx3BootPMMCIntrCb\_t cb)**  
*Register a callback for notification of PMMC interface events.*
- **void CyFx3BootPibHandleEvents (void)**  
*PIB event handler function.*
- **CyFx3BootErrorCode\_t CyFx3BootPibDmaXferData (uint8\_t sockNum, CyBool\_t isRead, uint32\_t address, uint32\_t length, uint32\_t timeout)**  
*This function is used to setup a dma from/to the external PIB master.*
- **void CyFx3BootGpifRegisterCallback (CyFx3BootGpifIntrCb\_t cbFunc)**  
*Register a callback function to be called on GPIF interrupt.*
- **CyFx3BootErrorCode\_t CyFx3BootGpifLoad (const CyU3PGpifConfig\_t \*conf)**  
*Configure the GPIF II hardware functionality.*
- **void CyFx3BootGpifDisable (CyBool\_t forceReload)**  
*Disable the GPIF II State Machine and Hardware.*
- **CyFx3BootErrorCode\_t CyFx3BootGpifSMStart (uint8\_t startState, uint8\_t initialAlpha)**  
*Start the GPIF II State Machine from the specified state.*
- **CyFx3BootErrorCode\_t CyFx3BootGpifSMSwitch (uint16\_t fromState, uint16\_t toState, uint8\_t initialAlpha)**

*This function is used to start GPIF state machine execution from a desired state.*

- [CyFx3BootErrorCode\\_t CyFx3BootGpifInitCounter](#) ([CyFx3GpifCounterType](#) counter, [uint32\\_t](#) initialValue, [uint32\\_t](#) limit, [CyBool\\_t](#) reload, [int8\\_t](#) increment, [uint8\\_t](#) outputbit)

*Function to initialize one of the GPIF II Counters.*

- [void CyFx3BootGpifControlSWInput](#) ([CyBool\\_t](#) set)

*Change the state of the firmware input to the GPIF II State Machine.*

- [uint8\\_t CyFx3BootGpifGetState](#) (void)

*Get the current GPIF state.*

- [CyFx3BootErrorCode\\_t CyFx3BootGpifSocketConfigure](#) ([uint8\\_t](#) threadIndex, [uint8\\_t](#) socketNum, [uint16\\_t](#) watermark, [CyBool\\_t](#) flagOnData, [uint8\\_t](#) burst)

*Function to select an active socket on the P-port and to configure it.*

## 5.6.1 Detailed Description

The PIB driver module in the FX3 boot firmware enables the PMMC interface on the FX3 device and allows PMMC data transfers.

## 5.6.2 Typedef Documentation

### 5.6.2.1 [typedef void\(\\* CyFx3BootGpifIntrCb\\_t\)](#) ([uint8\\_t](#) currentState )

Callback function type to be called when a GPIF state machine interrupt is triggered.

#### Description

This type defines the type of a callback function that will be called when the GPIF state machine triggers an interrupt.

#### See also

[CyFx3BootGpifRegisterCallback](#)

### 5.6.2.2 [typedef struct CyFx3BootPibClock\\_t CyFx3BootPibClock\\_t](#)

Clock configuration information for the PIB block.

#### Description

The clock for the PIB block is derived from the SYS\_CLK. The base clock and frequency divider values are selected through this structure.

#### See also

[CyFx3BootSysClockSrc\\_t](#)  
[CyFx3BootPibInit](#)

### 5.6.2.3 [typedef enum CyFx3BootPMMCEvent\\_t CyFx3BootPMMCEvent\\_t](#)

List of MMC-Slave Event notifications.

#### Description

This type lists the various event notifications that are provided by the FX3 device when configured in the MMC-Slave mode.

#### See also

[CyFx3BootPibRegisterMmcCallback](#)

**5.6.2.4 `typedef void(* CyFx3BootPMMCIntrCb_t) (CyFx3BootPMMCEvent_t cbType, uint32_t cbArg )`**

Callback function type used for MMC-Slave event notifications.

**Description**

This function pointer type defines the signature of the callback function that will be called to notify the user of MMC-Slave (PMMC) Mode events.

**See also**

[CyFx3BootPMMCEvent\\_t](#)  
[CyFx3BootPibRegisterMmcCallback](#)

**5.6.2.5 `typedef enum CyFx3GpifCounterType CyFx3GpifCounterType`**

List of GPIF counters.

**Description**

The GPIF II hardware provides a set of counters that can be used to control the state machine operation. This type lists the various types of counters supported by the GPIF hardware.

**See also**

[CyFx3BootGpifInitCounter](#)

**5.6.2.6 `typedef struct CyU3PGpifConfig_t CyU3PGpifConfig_t`**

Structure that holds all configuration inputs for the GPIF hardware.

**Description**

The GPIF block on the FX3 device has a set of general configuration registers, transition function registers and state descriptors that need to be initialized to make the GPIF state machine functional. This structure encapsulates all the data that is required to program the GPIF block to load a user defined state machine.

The GPIF configuration data in the form of this structure is commonly generated by the GPIF II Designer tool.

**See also**

[CyU3PGpifWaveData](#)  
[CyFx3BootGpifLoad](#)

**5.6.2.7 `typedef struct CyU3PGpifWaveData CyU3PGpifWaveData`**

Information on a single GPIF transition from one state to another.

**Description**

The GPIF state machine on the FX3 device is defined through a set of transition descriptors. These descriptors include fields for specifying the next state, the conditions for transition, and the output values.

This structure encapsulates all of the information that forms the left and right transition descriptors for a state.

**See also**

[CyU3PGpifConfig\\_t](#)

### 5.6.3 Enumeration Type Documentation

#### 5.6.3.1 enum CyFx3BootPMMCEvent\_t

List of MMC-Slave Event notifications.

##### Description

This type lists the various event notifications that are provided by the FX3 device when configured in the MMC-Slave mode.

##### See also

[CyFx3BootPibRegisterMmcCallback](#)

##### Enumerator

**CYFX3\_PMMC\_GOIDLE\_CMD** GO\_IDLE\_STATE (CMD0) command has been received.

**CYFX3\_PMMC\_CMD5\_SLEEP** CMD5(SLEEP) command has been used to place FX3 into suspend mode.

**CYFX3\_PMMC\_CMD5\_AWAKE** CMD5(AWAKE) command has been used to wake FX3 from suspend mode.

**CYFX3\_PMMC\_CMD6\_SWITCH** SWITCH (CMD6) command has been received.

**CYFX3\_PMMC\_CMD12\_STOP** STOP\_TRANSMISSION (CMD12) command has been received.

**CYFX3\_PMMC\_CMD15\_INACTIVE** GO\_INACTIVE\_STATE (CMD15) command has been received.

**CYFX3\_PMMC\_SOCKET\_NOT\_READY** Socket being read/written by the host is not ready.

**CYFX3\_PMMC\_CMD7\_SELECT** SELECT\_CARD (CMD7) command has been received.

#### 5.6.3.2 enum CyFx3GpifCounterType

List of GPIF counters.

##### Description

The GPIF II hardware provides a set of counters that can be used to control the state machine operation. This type lists the various types of counters supported by the GPIF hardware.

##### See also

[CyFx3BootGpifInitCounter](#)

##### Enumerator

**CYFX3\_GPIF\_COUNTER\_CONTROL** Control counter: 16 bit.

**CYFX3\_GPIF\_COUNTER\_ADDRESS** Address counter: 32 bit.

**CYFX3\_GPIF\_COUNTER\_DATA** Data counter: 32 bit.

### 5.6.4 Function Documentation

#### 5.6.4.1 void CyFx3BootGpifControlSWInput ( CyBool\_t set )

Change the state of the firmware input to the GPIF II State Machine.

##### Description

The GPIF II State Machine is capable of waiting for firmware readiness using an input signal that can be controlled by the firmware. This function updates the state of this input signal as desired.

##### Return value

None

**Parameters**

<code>set</code>	Whether to set (CyTrue) or clear (CyFalse) the input signal.
------------------	--

**5.6.4.2 void CyFx3BootGpifDisable ( `CyBool_t forceReload` )**

Disable the GPIF II State Machine and Hardware.

**Description**

This function disables the GPIF II State Machine and hardware. If the forceReload parameter is true, the current configuration is cleared completely, and the CyFx3BootGpifLoad function must be called again. If the forceReload parameter is not set, the GPIF configuration is retained in memory, and the state machine can be restarted using the CyFx3BootGpifSMStart API.

**Return value**

None

**See also**

[CyFx3BootGpifLoad](#)  
[CyFx3BootGpifSMStart](#)

**Parameters**

<code>forceReload</code>	Whether a GPIF re-configuration is to be forced.
--------------------------	--

**5.6.4.3 uint8\_t CyFx3BootGpifGetState ( `void` )**

Get the current GPIF state.

**Description**

This function gets the current GPIF II state machine state.

**Return value**

The current state. 0 if the state cannot be determined.

**5.6.4.4 CyFx3BootErrorCode\_t CyFx3BootGpifInitCounter ( `CyFx3GpifCounterType counter`, `uint32_t initialValue`, `uint32_t limit`, `CyBool_t reload`, `int8_t increment`, `uint8_t outputbit` )**

Function to initialize one of the GPIF II Counters.

**Description**

This function initializes one of the GPIF II counters with the specified parameters.

**Return value**

CY\_FX3\_BOOT\_SUCCESS - if the counter is configured properly.

CY\_FX3\_BOOT\_ERROR\_NOT\_STARTED - if the PIB block is not started.

CY\_FX3\_BOOT\_ERROR\_NOT\_CONFIGURED - if the PIB block is configured in MMC mode.

CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT - if the parameters are incorrect.

**See also**

[CyFx3GpifCounterType](#)

**Parameters**

<i>counter</i>	Type of counter to be initialized.
<i>initValue</i>	Initial value for the counter.
<i>limit</i>	Value at which the counter must be stopped.
<i>reload</i>	Whether the counter should be reloaded and restarted after reaching the limit.
<i>increment</i>	Value to update the counter by at each step. Can be negative.
<i>outputbit</i>	Selects the control counter bit to be connected to the CTRL[9] output.

**5.6.4.5 CyFx3BootErrorCode\_t CyFx3BootGpifLoad ( const CyU3PGpifConfig\_t \* *conf* )**

Configure the GPIF II hardware functionality.

**Description**

This function configures the GPIF II hardware on the FX3 device to implement the desired interface and functionality. The configuration data passed as an argument to this function is generated by the GPIF II Designer tool.

**Return value**

CY\_FX3\_BOOT\_SUCCESS - if the configuration load is successful.

CY\_FX3\_BOOT\_ERROR\_NOT\_SUPPORTED - if the configuration being loaded cannot be supported.

CY\_FX3\_BOOT\_ERROR\_ALREADY\_STARTED - if the GPIF hardware is already configured and running.

CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT - if the input structure is inconsistent.

**See also**

[CyU3PGpifConfig\\_t](#)

**Parameters**

<i>conf</i>	Pointer to GPIF configuration information.
-------------	--

**5.6.4.6 void CyFx3BootGpifRegisterCallback ( CyFx3BootGpifIntrCb\_t *cbFunc* )**

Register a callback function to be called on GPIF interrupt.

**Description**

This function registers a callback function that will be called when the GPIF state machine triggers an interrupt. No other GPIF interrupt types are supported.

**Return value**

None

**See also**

[CyFx3BootGpifIntrCb\\_t](#)

**Parameters**

<i>cbFunc</i>	Callback function pointer.
---------------	----------------------------

#### 5.6.4.7 CyFx3BootErrorCode\_t CyFx3BootGpifSMStart ( uint8\_t *startState*, uint8\_t *initialAlpha* )

Start the GPIF II State Machine from the specified state.

##### Description

This function starts the GPIF II State Machine from the specified state index. This can only be used to start the GPIF II State Machine from an idle state. The CyFx3BootGpifSMSwitch API should be used to switch the state machine from one state to another in mid-execution.

##### Return value

CY\_FX3\_BOOT\_SUCCESS - if the state machine is started successfully.

CY\_FX3\_BOOT\_ERROR\_NOT\_CONFIGURED - if the GPIF hardware has not been configured.

CY\_FX3\_BOOT\_ERROR\_ALREADY\_STARTED - if the state machine has already been started.

##### See also

[CyFx3BootGpifSMSwitch](#)

##### Parameters

<i>startState</i>	State from which to start state machine execution. This should be one of the start states in the GPIF state machine design.
<i>initialAlpha</i>	Initial values of alpha inputs to start the GPIF state machine with.

#### 5.6.4.8 CyFx3BootErrorCode\_t CyFx3BootGpifSMSwitch ( uint16\_t *fromState*, uint16\_t *toState*, uint8\_t *initialAlpha* )

This function is used to start GPIF state machine execution from a desired state.

##### Description

This function allows the caller to switch to a desired state and continue GPIF state machine execution from there. The toState parameter specifies the state to initiate operation from.

The fromState parameter can be used to ensure that the transition to toState happens only when the state machine is in a well defined idle state. If a valid state id (0 - 255) is passed as the fromState, the transition is only allowed from that state index. If not, the state machine is immediately switched to the toState.

##### Return value

CY\_FX3\_BOOT\_SUCCESS - if the switch request is successful.

CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT - if the parameters are invalid.

##### See also

[CyFx3BootGpifSMStart](#)

##### Parameters

<i>fromState</i>	State from which to do the switch. Can be set to an invalid value to force an immediate switch.
<i>toState</i>	State to switch operation into. Must be a valid state number.
<i>initialAlpha</i>	Initial values for the alpha inputs when switching states.

#### 5.6.4.9 CyFx3BootErrorCode\_t CyFx3BootGpifSocketConfigure ( uint8\_t *threadIndex*, uint8\_t *socketNum*, uint16\_t *watermark*, CyBool\_t *flagOnData*, uint8\_t *burst* )

Function to select an active socket on the P-port and to configure it.

**Description**

The GPIF hardware allows 4 different sockets on the P-port to be accessed at a time by supporting 4 independent DMA threads. The active socket for each thread and its properties can be selected by the user at run-time. This should be done in software only in the case where it is not being done through the PP registers or the state machine itself.

This function allows the user to select and configure the active socket in the case where software is responsible for these actions. The API will respond with an error if the hardware is taking care of socket configurations.

**Return value**

CY\_FX3\_BOOT\_SUCCESS - if successful.

CY\_FX3\_BOOT\_ERROR\_FAILURE - if the socket selection and configuration is being done automatically.

CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT - if one or more of the parameters are out of range.

**Parameters**

<i>threadIndex</i>	Thread index whose active socket is to be configured.
<i>socketNum</i>	The socket to be associated with this thread. shoulf be if the socket or more if the use 5-24339(r)k100(.)] TJ 0 -11.95

**Parameters**

<i>sockNum</i>	Socket through which to do the data transfer.
<i>isRead</i>	<i>isRead</i> =CyTrue for read transfers, and <i>isRead</i> =CyFalse for write transfers.
<i>address</i>	Address of the buffer from/to which data is to be transferred
<i>length</i>	Length of the data to be transferred. Should be a multiple of 16 bytes.
<i>timeout</i>	Timeout duration in multiples of 10 us. Can be set to CY_FX3_BOOT_WAIT_FOREVER to wait until the transfer is complete.

5.6.4.12 void CyFx3BootPibHandleEvents( void )

PIB event handler function.

**Description**

This function looks for PIB related interrupts and handles them. This function is expected to be called periodically from the application main, if the PIB block is being used.

**Return Value**

None

5.6.4.13 CyFx3BootErrorCode\_t CyFx3BootPibInit( CyFx3BootPibClock\_t \* *clkInfo\_p*, CyBool\_t *isMmcMode* )

Starts the Processor Interface Block.

**Description**

Initialize the Processor Interface Block on the FX3 device. The clock parameters and the interface mode (PMMC or GPIF-II) are to be specified as parameters.

**Return value**

CY\_FX3\_BOOT\_SUCCESS - If the block is initialized successfully.

CY\_FX3\_BOOT\_ERROR\_NOT\_SUPPORTED - If the mode selected is not supported.

**See also**

[CyFx3BootPibDeinit](#)

**Parameters**

<i>clkInfo_p</i>	Selected clock parameters.
<i>isMmcMode</i>	Whether to start in PMMC (CyTrue) or GPIF-II (CyFalse) mode.

5.6.4.14 void CyFx3BootPibRegisterMmcCallback( CyFx3BootPMMCIntrCb\_t *cb* )

Register a callback for notification of PMMC interface events.

**Description**

Set a callback function to receive notification of PMMC interface events. These are typically invoked when the MMC host sends down specific commands that require firmware intervention.

**See also**

[CyFx3BootPMMCEvent\\_t](#)  
[CyFx3BootPMMCIntrCb\\_t](#)

## Parameters

<code>cb</code>	Callback function pointer.
-----------------	----------------------------

## 5.7 firmware/boot-fw/include/cyfx3spi.h File Reference

SPI (Serial Peripheral Interface) is a serial interface defined for inter-device communication. The FX3 device includes a SPI master that can connect to a variety of SPI slave devices and function at various speeds and modes. This file defines the data structures and APIs that enable SPI slave access from the FX3 boot API library.

```
#include <cyu3types.h>
#include <cyfx3error.h>
#include <cyu3externcstart.h>
#include <cyu3externcend.h>
```

## Data Structures

- struct [CyFx3BootSpiConfig\\_t](#)  
*Structure defining the configuration of SPI interface.*

## TypeDefs

- typedef enum [CyFx3BootSpiSsnLagLead\\_t](#) [CyFx3BootSpiSsnLagLead\\_t](#)  
*Enumeration defining SSN lead and lag times with respect to SCK.*
- typedef enum [CyFx3BootSpiSsnCtrl\\_t](#) [CyFx3BootSpiSsnCtrl\\_t](#)  
*List of ways in which the SPI Slave Select (SSN) line can be controlled.*
- typedef struct [CyFx3BootSpiConfig\\_t](#) [CyFx3BootSpiConfig\\_t](#)  
*Structure defining the configuration of SPI interface.*

## Enumerations

- enum [CyFx3BootSpiSsnLagLead\\_t](#) {  
 CY\_FX3\_BOOT\_SPI\_SSN\_LAG\_LEAD\_ZERO\_CLK = 0, CY\_FX3\_BOOT\_SPI\_SSN\_LAG\_LEAD\_HALF\_CLK, CY\_FX3\_BOOT\_SPI\_SSN\_LAG\_LEAD\_ONE\_CLK, CY\_FX3\_BOOT\_SPI\_SSN\_LAG\_LEAD\_ONE\_HALF\_CLK,  
 CY\_FX3\_BOOT\_SPI\_NUM\_SSN\_LAG\_LEAD }  
*Enumeration defining SSN lead and lag times with respect to SCK.*
- enum [CyFx3BootSpiSsnCtrl\\_t](#) {  
 CY\_FX3\_BOOT\_SPI\_SSN\_CTRL\_FW = 0, CY\_FX3\_BOOT\_SPI\_SSN\_CTRL\_HW\_END\_OF\_XFER, CY\_FX3\_BOOT\_SPI\_SSN\_CTRL\_HW\_EACH\_WORD, CY\_FX3\_BOOT\_SPI\_SSN\_CTRL\_HW\_CPHA\_BASERD,  
 CY\_FX3\_BOOT\_SPI\_SSN\_CTRL\_NONE, CY\_FX3\_BOOT\_SPI\_NUM\_SSN\_CTRL }  
*List of ways in which the SPI Slave Select (SSN) line can be controlled.*

## Functions

- [CyFx3BootErrorCode\\_t CyFx3BootSpiInit](#) (void)  
*Starts the SPI interface block on the device.*
- void [CyFx3BootSpiDeInit](#) (void)

- *Stops the SPI block.*  
 • [CyFx3BootErrorCode\\_t CyFx3BootSpiSetConfig \(CyFx3BootSpiConfig\\_t \\*config\)](#)  
*Configures SPI interface parameters.*
- void [CyFx3BootSpiSetSsnLine \(CyBool\\_t isHigh\)](#)  
*Assert / Deassert the SSN Line.*
- [CyFx3BootErrorCode\\_t CyFx3BootSpiTransmitWords \(uint8\\_t \\*data, uint32\\_t byteCount\)](#)  
*Transmits data word by word over the SPI interface.*
- [CyFx3BootErrorCode\\_t CyFx3BootSpiReceiveWords \(uint8\\_t \\*data, uint32\\_t byteCount\)](#)  
*Receives data word by word over the SPI interface.*
- void [CyFx3BootSpiSetBlockXfer \(uint32\\_t txSize, uint32\\_t rxSize\)](#)  
*Enables DMA data transfers through the SPI interface.*
- void [CyFx3BootSpiDisableBlockXfer \(void\)](#)  
*Disable DMA data transfers through the SPI interface.*
- [CyFx3BootErrorCode\\_t CyFx3BootSpiDmaXferData \(CyBool\\_t isRead, uint32\\_t address, uint32\\_t length, uint32\\_t timeout\)](#)  
*This function is used to setup a DMA transfer from CPU to SPI or vice versa.*
- void [CyFx3BootSpiSetTimeout \(uint32\\_t timeout\)](#)  
*Set the timeout duration for SPI byte mode data transfers.*

### 5.7.1 Detailed Description

SPI (Serial Peripheral Interface) is a serial interface defined for inter-device communication. The FX3 device includes a SPI master that can connect to a variety of SPI slave devices and function at various speeds and modes. This file defines the data structures and APIs that enable SPI slave access from the FX3 boot API library.

### 5.7.2 Typedef Documentation

#### 5.7.2.1 `typedef struct CyFx3BootSpiConfig_t CyFx3BootSpiConfig_t`

Structure defining the configuration of SPI interface.

##### Description

This structure encapsulates all of the configurable parameters that can be selected for the SPI interface. The [CyFx3BootSpiSetConfig\(\)](#) function accepts a pointer to this structure, and updates all of the interface parameters.

##### See also

[CyFx3BootSpiSetConfig](#)  
[CyFx3BootSpiSsnCtrl\\_t](#)  
[CyFx3BootSpiSsnLagLead\\_t](#)

#### 5.7.2.2 `typedef enum CyFx3BootSpiSsnCtrl_t CyFx3BootSpiSsnCtrl_t`

List of ways in which the SPI Slave Select (SSN) line can be controlled.

##### Description

The SPI Slave Select (SSN) signal is a per-slave signal; and multiple SSN signals may be needed in the case where FX3 needs to talk to multiple SPI slaves. To satisfy this need, the FX3 device supports multiple ways in which the SSN signal can be controlled during a SPI data transfer.

The default SSN signal on FX3 can be automatically controlled by the FX3 hardware in sync with the SPI clock ( $S\leftarrow CK$ ). Another possibility is to have the SSN signal controlled as a GPIO by the SPI transfer APIs. This functionality is also supported only for the default SSN signal.

A third option is for the firmware application to directly assert/de-assert the SSN signal through the GPIO APIs. This mode of operation can be used with any FX3 GPIO.

See also

[CyFx3BootSpiConfig\\_t](#)  
[CyFx3BootSpiSetConfig](#)

### 5.7.2.3 `typedef enum CyFx3BootSpiSsnLagLead_t CyFx3BootSpiSsnLagLead_t`

Enumeration defining SSN lead and lag times with respect to SCK.

#### Description

The Slave Select (SSN) signal needs to lead the SCK at the beginning of a SPI transaction and lag the SCK at the end of the transfer. When the default SSN signal on the FX3 is used, the hardware provides various modes of automatically controlling the SSN assertion. This enumerated type lists the various SSN timings that are supported by the FX3 hardware.

See also

[CyFx3BootSpiConfig\\_t](#)  
[CyFx3BootSpiSetConfig](#)

## 5.7.3 Enumeration Type Documentation

### 5.7.3.1 `enum CyFx3BootSpiSsnCtrl_t`

List of ways in which the SPI Slave Select (SSN) line can be controlled.

#### Description

The SPI Slave Select (SSN) signal is a per-slave signal; and multiple SSN signals may be needed in the case where FX3 needs to talk to multiple SPI slaves. To satisfy this need, the FX3 device supports multiple ways in which the SSN signal can be controlled during a SPI data transfer.

The default SSN signal on FX3 can be automatically controlled by the FX3 hardware in sync with the SPI clock ( $S \leftarrow CK$ ). Another possibility is to have the SSN signal controlled as a GPIO by the SPI transfer APIs. This functionality is also supported only for the default SSN signal.

A third option is for the firmware application to directly assert/de-assert the SSN signal through the GPIO APIs. This mode of operation can be used with any FX3 GPIO.

See also

[CyFx3BootSpiConfig\\_t](#)  
[CyFx3BootSpiSetConfig](#)

#### Enumerator

**CY\_FX3\_BOOT\_SPI\_SSN\_CTRL\_FW** SSN is controlled by the API and is not synchronized to clock boundaries. It is asserted at the beginning of a transfer, and is de-asserted at the end of transfer.

**CY\_FX3\_BOOT\_SPI\_SSN\_CTRL\_HW\_END\_OF\_XFER** SSN is controlled by hardware and is toggled in sync with clock. The SSN is asserted at the beginning of a transfer, and is de-asserted at the end of a transfer or when no data is available to transmit (underrun).

**CY\_FX3\_BOOT\_SPI\_SSN\_CTRL\_HW\_EACH\_WORD** SSN is controlled by hardware and is toggled in sync with clock. The SSN is asserted at the beginning of transfer of every word, and de-asserted at the end of the transfer of that word.

**CY\_FX3\_BOOT\_SPI\_SSN\_CTRL\_HW\_CPHA\_BASED** If CPHA is 0, the SSN control is per word; and if CPHA is 1, then the SSN control is per transfer.

**CY\_FX3\_BOOT\_SPI\_SSN\_CTRL\_NONE** SSN control is done externally. The SSN lines are selected by the application and are ignored by the hardware / API.

**CY\_FX3\_BOOT\_SPI\_NUM\_SSN\_CTRL** Number of enumerations. Should not be used.

### 5.7.3.2 enum CyFx3BootSpiSsnLagLead\_t

Enumeration defining SSN lead and lag times with respect to SCK.

#### Description

The Slave Select (SSN) signal needs to lead the SCK at the beginning of a SPI transaction and lag the SCK at the end of the transfer. When the default SSN signal on the FX3 is used, the hardware provides various modes of automatically controlling the SSN assertion. This enumerated type lists the various SSN timings that are supported by the FX3 hardware.

#### See also

[CyFx3BootSpiConfig\\_t](#)  
[CyFx3BootSpiSetConfig](#)

#### Enumerator

**CY\_FX3\_BOOT\_SPI\_SSN\_LAG\_LEAD\_ZERO\_CLK** SSN will be asserted in sync with SCK.  
**CY\_FX3\_BOOT\_SPI\_SSN\_LAG\_LEAD\_HALF\_CLK** SSN leads / lags SCK by a half clock cycle.  
**CY\_FX3\_BOOT\_SPI\_SSN\_LAG\_LEAD\_ONE\_CLK** SSN leads / lags SCK by one clock cycle.  
**CY\_FX3\_BOOT\_SPI\_SSN\_LAG\_LEAD\_ONE\_HALF\_CLK** SSN leads / lags SCK by one and half clock cycles.  
**CY\_FX3\_BOOT\_SPI\_NUM\_SSN\_LAG\_LEAD** Number of enumerations. Should not be used.

## 5.7.4 Function Documentation

### 5.7.4.1 void CyFx3BootSpiDelInit( void )

Stops the SPI block.

#### Description

This function disables and powers off the SPI interface. This function can be used to shut off the interface to save power when it is not in use.

#### Return value

None

#### See also

[CyFx3BootSpiInit](#)

### 5.7.4.2 void CyFx3BootSpiDisableBlockXfer( void )

Disable DMA data transfers through the SPI interface.

#### Description

This function disables DMA data transfers through the SPI interface, so that register mode transfers can be performed again.

#### Return value

None

#### See also

[CyFx3BootSpiSetBlockXfer](#)

**5.7.4.3 CyFx3BootErrorCode\_t CyFx3BootSpiDmaXferData ( CyBool\_t isRead, uint32\_t address, uint32\_t length, uint32\_t timeout )**

This function is used to setup a DMA transfer from CPU to SPI or vice versa.

#### Description

This function is used to read/write data from/to a SPI slave in DMA mode. The contents of a single data buffer can be transferred through a one-shot DMA pipe that is configured by this API call.

Note\*\*

The CyFx3BootSpiSetBlockXfer API needs to be called to configure the SPI block for DMA bound data transfers.

#### Return value

CY\_FX3\_BOOT\_SUCCESS - if the data transfer is successful.  
 CY\_FX3\_BOOT\_ERROR\_INVALID\_DMA\_ADDR - if the address specified is not in the SYSMEM area.  
 CY\_FX3\_BOOT\_ERROR\_XFER\_FAILURE - if the data transfer encountered any error.  
 CY\_FX3\_BOOT\_ERROR\_TIMEOUT - if the data transfer times out.

#### See also

[CyFx3BootSpiDisableBlockXfer](#)  
[CyFx3BootSpiSetBlockXfer](#)

#### Parameters

<i>isRead</i>	Transfer direction - isRead=CyTrue for read operations; and isRead=CyFalse for write operations.
<i>address</i>	Address of the buffer from/to which data is to be transferred.
<i>length</i>	Length of the data to be transferred.
<i>timeout</i>	Timeout duration in multiples of 10 us. Can be set to CY_FX3_BOOT_WAIT_FOREVER to compulsorily wait for end of transfer.

**5.7.4.4 CyFx3BootErrorCode\_t CyFx3BootSpiInit ( void )**

Starts the SPI interface block on the device.

#### Description

This function powers up the SPI interface block on the device and is expected to be the first SPI API function that is called by the application.

#### Return value

CY\_FX3\_BOOT\_SUCCESS - if the SPI block was successfully initialized.

#### See also

[CyFx3BootSpiDeInit](#)  
[CyFx3BootSpiSetConfig](#)

**5.7.4.5 CyFx3BootErrorCode\_t CyFx3BootSpiReceiveWords ( uint8\_t \* data, uint32\_t byteCount )**

Receives data word by word over the SPI interface.

#### Description

Receives data from the SPI slave in word-by-word register mode of transfer. This can be used only when a DMA transfer on the SPI block has not been started using the CyFx3BootSpiSetBlockXfer API.

#### Return value

CY\_FX3\_BOOT\_SUCCESS - when the ReceiveWords is successful.

CY\_FX3\_BOOT\_ERROR\_NULL\_POINTER - when the data pointer is NULL.

CY\_FX3\_BOOT\_ERROR\_XFER\_FAILURE - when the SPI block encounters an error during the transfer.

CY\_FX3\_BOOT\_ERROR\_TIMEOUT - when the ReceiveWords times out.

#### See also

[CyFx3BootSpiTransmitWords](#)

#### Parameters

<i>data</i>	Destination buffer pointer.
<i>byteCount</i>	Amount of data to be received (in bytes). This needs to be a multiple of the word length after alignment to the next byte value.

#### 5.7.4.6 void CyFx3BootSpiSetBlockXfer ( *uint32\_t txSize, uint32\_t rxSize* )

Enables DMA data transfers through the SPI interface.

#### Description

This function switches the SPI block to DMA mode, and initiates the desired read/write transfers.

If the txSize parameter is non-zero, then TX is enabled; and if rxSize parameter is non-zero, then RX is enabled. If both TX and RX are enabled, transfers can only happen while both the SPI producer and SPI consumer sockets are ready for data transfer.

If the receive count is less than the transmit count, the data transmit continues after the scheduled reception is complete. However, if the transmit count is lesser, data reception is stalled at the end of the transmit operation. The SPI transmit transfer needs to be disabled before the receive can proceed.

The CyFx3BootSpiDmaXferData API needs to be used to set up the DMA data path to do the actual data transfer. A call to SetBlockXfer has to be followed by a call to DisableBlockXfer, before the SPI block can be used in Register mode.

#### Return value

None

#### See also

[CyFx3BootSpiDisableBlockXfer](#)  
[CyFx3BootSpiDmaXferData](#)

#### Parameters

<i>txSize</i>	Number of words to be transmitted.
<i>rxSize</i>	Number of words to be received.

#### 5.7.4.7 CyFx3BootErrorCode\_t CyFx3BootSpiSetConfig ( *CyFx3BootSpiConfig\_t \* config* )

Configures SPI interface parameters.

#### Description

This function is used to configure the SPI master interface based on the desired settings to talk to the desired slave. This function can be called repeatedly to change the settings if different settings are to be used to communicate with different slave devices.

This API resets the TX/RX FIFOs associated with the SPI block, and will result in the loss of any data that is in the FIFOs.

**Return value**

CY\_FX3\_BOOT\_SUCCESS - when the SetConfig is successful.  
CY\_FX3\_BOOT\_ERROR\_NOT\_STARTED - if the SPI block has not been initialized.  
CY\_FX3\_BOOT\_ERROR\_NULL\_POINTER - when the config pointer is NULL.  
CY\_FX3\_BOOT\_ERROR\_TIMEOUT - when the operation times out.

**See also**

[CyFx3BootSpiConfig\\_t](#)

**Parameters**

<i>config</i>	Pointer to the SPI config structure
---------------	-------------------------------------

**5.7.4.8 void CyFx3BootSpiSetSsnLine ( CyBool\_t *isHigh* )**

Assert / Deassert the SSN Line.

**Description**

Asserts/de-asserts the SSN Line for the default slave device. This is possible only if the SSN line control is configured for FW controlled mode.

**Return value**

None

**See also**

[CyFx3BootSpiSsnCtrl\\_t](#)

**Parameters**

<i>isHigh</i>	CyFalse: Pull down the SSN line, CyTrue: Pull up the SSN line.
---------------	--

**5.7.4.9 void CyFx3BootSpiSetTimeout ( uint32\_t *timeout* )**

Set the timeout duration for SPI byte mode data transfers.

**Description**

The SPI byte mode transfer functions (CyFx3BootSpiTransmitWords and CyFx3BootSpiReceiveWords) are implemented using a poll loop which continually checks for the completion of the transfer. The delay for each of these loops will be approximately 2 us, and the default value of the maximum loop count is 1048575. This API can be used to change the timeout loop count to a different value.

**Return value**

None

**5.7.4.10 CyFx3BootErrorCode\_t CyFx3BootSpiTransmitWords ( uint8\_t \* *data*, uint32\_t *byteCount* )**

Transmits data word by word over the SPI interface.

**Description**

This function is used to transmit data word by word. The function can be called only if there is no active DMA transfer on the bus. If CyFx3BootSpiSetBlockXfer has been called, the CyFx3BootSpiDisableBlockXfer API needs to be called before this API can be used.

**Return value**

`CY_FX3_BOOT_SUCCESS` - when the `TransmitWords` is successful.  
`CY_FX3_BOOT_ERROR_NULL_POINTER` - when the data pointer is NULL.  
`CY_FX3_BOOT_ERROR_XFER_FAILURE` - when the SPI block encounters an error during the transfer.  
`CY_FX3_BOOT_ERROR_TIMEOUT` - when the `TransmitWords` times out.

**See also**

[CyFx3BootSpiReceiveWords](#)

**Parameters**

<code>data</code>	Source data pointer. This needs to be padded to nearest byte if the word length is not byte aligned.
<code>byteCount</code>	This needs to be a multiple of the word length aligned to the next byte value.

## 5.8 firmware/boot-fw/include/cyfx3uart.h File Reference

The UART interface manager module is responsible for handling the transfer of data through the UART interface on the device. This file defines the data structures and APIs for UART interface management.

```
#include <cyu3types.h>
#include <cyfx3error.h>
#include <cyu3externcstart.h>
#include <cyu3externcend.h>
```

### Data Structures

- struct [CyFx3BootUartConfig\\_t](#)  
*Configuration parameters for the UART interface.*

### Typedefs

- typedef enum [CyFx3BootUartBaudrate\\_t](#) [CyFx3BootUartBaudrate\\_t](#)  
*List of baud rates supported by the UART.*
- typedef enum [CyFx3BootUartStopBit\\_t](#) [CyFx3BootUartStopBit\\_t](#)  
*List of number of stop bits to be used in UART communication.*
- typedef enum [CyFx3BootUartParity\\_t](#) [CyFx3BootUartParity\\_t](#)  
*List of parity settings supported by the UART interface.*
- typedef struct [CyFx3BootUartConfig\\_t](#) [CyFx3BootUartConfig\\_t](#)  
*Configuration parameters for the UART interface.*

### Enumerations

- enum [CyFx3BootUartBaudrate\\_t](#) {  
`CY_FX3_BOOT_UART_BAUDRATE_4800` = 4800, `CY_FX3_BOOT_UART_BAUDRATE_9600` = 9600, `CY_FX3_BOOT_UART_BAUDRATE_19200` = 19200, `CY_FX3_BOOT_UART_BAUDRATE_38400` = 38400,  
`CY_FX3_BOOT_UART_BAUDRATE_57600` = 57600, `CY_FX3_BOOT_UART_BAUDRATE_115200` = 115200 }

*List of baud rates supported by the UART.*

- enum [CyFx3BootUartStopBit\\_t](#) { CY\_FX3\_BOOT\_UART\_ONE\_STOP\_BIT = 1, CY\_FX3\_BOOT\_UART\_TWO\_STOP\_BIT = 2 }

*List of number of stop bits to be used in UART communication.*

- enum [CyFx3BootUartParity\\_t](#) { CY\_FX3\_BOOT\_UART\_NO\_PARITY = 0, CY\_FX3\_BOOT\_UART\_EVEN\_PARITY, CY\_FX3\_BOOT\_UART\_ODD\_PARITY, CY\_FX3\_BOOT\_UART\_NUM\_PARITY }

*List of parity settings supported by the UART interface.*

## Functions

- [CyFx3BootErrorCode\\_t CyFx3BootUartInit](#) (void)  
*Starts the UART hardware block on the device.*
- void [CyFx3BootUartDeInit](#) (void)  
*Stops the UART hardware block.*
- [CyFx3BootErrorCode\\_t CyFx3BootUartSetConfig](#) ([CyFx3BootUartConfig\\_t](#) \*config)  
*Sets the UART interface parameters.*
- void [CyFx3BootUartTxSetBlockXfer](#) (uint32\_t txSize)  
*Sets the UART transmit data count in the case of DMA mode operation.*
- void [CyFx3BootUartRxSetBlockXfer](#) (uint32\_t rxSize)  
*Sets the number of bytes to be received by the UART in DMA mode.*
- uint32\_t [CyFx3BootUartTransmitBytes](#) (uint8\_t \*data\_p, uint32\_t count)  
*Transmits data through the UART interface on a byte by byte basis.*
- uint32\_t [CyFx3BootUartReceiveBytes](#) (uint8\_t \*data\_p, uint32\_t count)  
*Receives data from the UART interface on a byte by byte basis.*
- [CyFx3BootErrorCode\\_t CyFx3BootUartDmaXferData](#) ([CyBool\\_t](#) isRead, uint32\_t address, uint32\_t length, uint32\_t timeout)  
*This function is used to setup a DMA from CPU to UART or vice versa.*
- [CyFx3BootErrorCode\\_t CyFx3BootUartPrintMessage](#) (uint8\_t \*outBuf\_p, uint16\_t outBufLen, char \*fmt,...)  
*Print a formatted string out through the UART.*
- void [CyFx3BootUartSetTimeout](#) (uint32\_t timeout)  
*Set the timeout duration for UART byte mode data transfers.*

### 5.8.1 Detailed Description

The UART interface manager module is responsible for handling the transfer of data through the UART interface on the device. This file defines the data structures and APIs for UART interface management.

### 5.8.2 Typedef Documentation

#### 5.8.2.1 [typedef enum CyFx3BootUartBaudrate\\_t CyFx3BootUartBaudrate\\_t](#)

List of baud rates supported by the UART.

#### Description

This enumeration lists the various baud rate settings that are supported by the UART interface and driver implementation. The specific baud rates achieved will be close approximations of these standard values based on the clock frequencies that can be obtained on the FX3 hardware.

#### See also

[CyFx3BootUartConfig\\_t](#)

### 5.8.2.2 `typedef struct CyFx3BootUartConfig_t CyFx3BootUartConfig_t`

Configuration parameters for the UART interface.

#### Description

This structure defines all of the configurable parameters for the UART interface such as baud rate, stop and parity bits etc. A pointer to this structure is passed in to the CyFx3BootUartSetConfig function to configure the UART interface.

#### See also

[CyFx3BootUartBaudrate\\_t](#)  
[CyFx3BootUartStopBit\\_t](#)  
[CyFx3BootUartParity\\_t](#)  
[CyFx3BootUartSetConfig](#)

### 5.8.2.3 `typedef enum CyFx3BootUartParity_t CyFx3BootUartParity_t`

List of parity settings supported by the UART interface.

#### Description

This enumeration lists the various parity settings that the UART interface can be configured to support.

#### See also

[CyFx3BootUartConfig\\_t](#)

### 5.8.2.4 `typedef enum CyFx3BootUartStopBit_t CyFx3BootUartStopBit_t`

List of number of stop bits to be used in UART communication.

#### Description

This enumeration lists the various number of stop bit settings that the UART interface can be configured to have. Only 1 and 2 are supported on the FX3 device.

#### See also

[CyFx3BootUartConfig\\_t](#)

## 5.8.3 Enumeration Type Documentation

### 5.8.3.1 `enum CyFx3BootUartBaudrate_t`

List of baud rates supported by the UART.

#### Description

This enumeration lists the various baud rate settings that are supported by the UART interface and driver implementation. The specific baud rates achieved will be close approximations of these standard values based on the clock frequencies that can be obtained on the FX3 hardware.

#### See also

[CyFx3BootUartConfig\\_t](#)

#### Enumerator

**CY\_FX3\_BOOT\_UART\_BAUDRATE\_4800** 4800 baud.

**CY\_FX3\_BOOT\_UART\_BAUDRATE\_9600** 9600 baud.

**CY\_FX3\_BOOT\_UART\_BAUDRATE\_19200** 19200 baud.  
**CY\_FX3\_BOOT\_UART\_BAUDRATE\_38400** 38400 baud.  
**CY\_FX3\_BOOT\_UART\_BAUDRATE\_57600** 57600 baud.  
**CY\_FX3\_BOOT\_UART\_BAUDRATE\_115200** 115200 baud.

#### 5.8.3.2 enum CyFx3BootUartParity\_t

List of parity settings supported by the UART interface.

##### Description

This enumeration lists the various parity settings that the UART interface can be configured to support.

See also

[CyFx3BootUartConfig\\_t](#)

Enumerator

**CY\_FX3\_BOOT\_UART\_NO\_PARITY** No parity bits.  
**CY\_FX3\_BOOT\_UART\_EVEN\_PARITY** Even parity.  
**CY\_FX3\_BOOT\_UART\_ODD\_PARITY** Odd parity.  
**CY\_FX3\_BOOT\_UART\_NUM\_PARITY** Number of parity enumerations.

#### 5.8.3.3 enum CyFx3BootUartStopBit\_t

List of number of stop bits to be used in UART communication.

##### Description

This enumeration lists the various number of stop bit settings that the UART interface can be configured to support.

**5.8.4.2 CyFx3BootErrorCode\_t CyFx3BootUartDmaXferData ( CyBool\_t *isRead*, uint32\_t *address*, uint32\_t *length*, uint32\_t *timeout* )**

This function is used to setup a DMA from CPU to UART or vice versa.

#### Description

This function is used to read/write a block of data through the UART interface in DMA mode. This is a blocking call which returns only when the transfer is complete, or the specified timeout duration has elapsed.

This API can only be called if the UART is setup for DMA mode of operation.

#### Return value

CY\_FX3\_BOOT\_SUCCESS - if data transfer is successful.  
 CY\_FX3\_BOOT\_ERROR\_INVALID\_DMA\_ADDR - if the address specified is not in the SYSMEM area.  
 CY\_FX3\_BOOT\_ERROR\_XFER\_FAILURE - if the data transfer encountered any error.  
 CY\_FX3\_BOOT\_ERROR\_TIMEOUT - if the data transfer times out.

#### See also

[CyFx3BootUartTxSetBlockXfer](#)  
[CyFx3BootUartRxSetBlockXfer](#)

#### Parameters

<i>isRead</i>	Transfer direction - isRead=CyTrue for read operations; and isRead=CyFalse for write operations.
<i>address</i>	Address of the buffer from/to which data is to be transferred.
<i>length</i>	Length of the data to be transferred.
<i>timeout</i>	Timeout duration in multiples of 10 us. Can be set to CY_FX3_BOOT_WAIT_FOREVER to compulsorily wait for end of transfer.

**5.8.4.3 CyFx3BootErrorCode\_t CyFx3BootUartInit ( void )**

Starts the UART hardware block on the device.

#### Description

This function powers up the UART hardware block on the device and should be the first UART related function called by the application.

#### Return value

CY\_FX3\_BOOT\_SUCCESS - when the UART block is successfully initialized.

#### See also

[CyFx3BootUartDeInit](#)  
[CyFx3BootUartSetConfig](#)

**5.8.4.4 CyFx3BootErrorCode\_t CyFx3BootUartPrintMessage ( uint8\_t \* *outBuf\_p*, uint16\_t *outBufLen*, char \* *fmt*, ... )**

Print a formatted string out through the UART.

#### Description

This function supports logging messages through the UART port of the FX3 device. A printf-style variable length argument list is supported with several restrictions. The function only supports a subset of the output conversion specifications supported by the snprintf function. The 'c', 'd', 'u', 'x' and 's' conversion specifications are supported. There is no support for float or double formats, or for flags, precision or type modifiers.

This function will use either the register or DMA mode of the UART to transmit the log data. The buffer to be used for storing the formatted string has to be passed in as an argument, along with maximum length of the string.

This is a blocking call and will only return after all of the data has been sent out from the FX3 device.

#### Return Value

CY\_FX3\_BOOT\_SUCCESS if the message has been sent out through UART. CY\_FX3\_BOOT\_ERROR\_NOT\_STARTED if the UART is not started. CY\_FX3\_BOOT\_ERROR\_NOT\_CONFIGURED if the UART is not enabled for transmission. CY\_FX3\_BOOT\_ERROR\_INVALID\_DMA\_ADDR if the buffer provided cannot be used for DMA transfer. CY\_FX3\_BOOT\_ERROR\_FAILURE if the message transmission fails on the UART side.

#### Parameters

<i>outBuf_p</i>	Pointer to buffer to be used for the formatted string.
<i>outBufLen</i>	Maximum length of the temporary buffer for formatted string.
<i>fmt</i>	Format string.

### 5.8.4.5 uint32\_t CyFx3BootUartReceiveBytes ( *uint8\_t \* data\_p, uint32\_t count* )

Receives data from the UART interface on a byte by byte basis.

#### Description

This function is used to read "count" number of bytes from the UART register interface. This function can only be used if the UART has been configured for register (non-DMA) transfer mode.

#### Return value

Amount of data actually transferred. A return of zero may indicate a transfer error.

#### See also

[CyFx3BootUartTransmitBytes](#)

#### Parameters

<i>data_p</i>	Pointer to location where the data read is to be placed.
<i>count</i>	Number of bytes to be received.

### 5.8.4.6 void CyFx3BootUartRxSetBlockXfer ( *uint32\_t rxSize* )

Sets the number of bytes to be received by the UART in DMA mode.

#### Description

This function sets the size of the desired data reception through the UART. This function is to be used when the UART is configured for DMA mode of transfer.

#### Return value

None

#### See also

[CyFx3BootUartTxSetBlockXfer](#)

#### Parameters

<i>rxSize</i>	Desired transfer size.
---------------	------------------------

### 5.8.4.7 CyFx3BootErrorCode\_t CyFx3BootUartSetConfig ( *CyFx3BootUartConfig\_t \* config* )

Sets the UART interface parameters.

#### Description

This function configures the UART block with the desired user parameters such as transfer mode, baud rate etc. This function should be called repeatedly to make any change to the set of configuration parameters. This can be called on the fly repetitively without calling CyFx3BootUartInit. But this will reset the FIFO and hence the data in pipe will be lost.

#### Return value

CY\_FX3\_BOOT\_SUCCESS - if the configuration was set successfully.

CY\_FX3\_BOOT\_ERROR\_NOT\_STARTED - if the UART block has not been initialized.

CY\_FX3\_BOOT\_ERROR\_NULL\_POINTER - if a NULL pointer is passed.

#### See also

[CyFx3BootUartConfig\\_t](#)

#### Parameters

<i>config</i>	Pointer to structure containing config information.
---------------	---

### 5.8.4.8 void CyFx3BootUartSetTimeout ( *uint32\_t timeout* )

Set the timeout duration for UART byte mode data transfers.

#### Description

The UART byte mode transfer functions (CyFx3BootUartTransmitBytes and CyFx3BootUartReceiveBytes) are implemented using a poll loop which continually checks for the completion of the transfer. The delay for each of these loops will be approximately 2 us, and the default value of the maximum loop count is 1048575. This API can be used to change the timeout loop count to a different value.

#### Return value

None

### 5.8.4.9 uint32\_t CyFx3BootUartTransmitBytes ( *uint8\_t \* data\_p, uint32\_t count* )

Transmits data through the UART interface on a byte by byte basis.

#### Description

This function is used to transfer "count" number of bytes out through the UART register interface. This function can only be used if the UART has been configured for register (non-DMA) transfer mode.

#### Return value

Amount of data actually transferred. A return of zero may indicate a transfer error.

#### See also

[CyFx3BootUartReceiveBytes](#)

#### Parameters

<i>data_p</i>	Pointer to the data to be transferred.
<i>count</i>	Number of bytes to be transferred.

#### 5.8.4.10 void CyFx3BootUartTxSetBlockXfer ( uint32\_t txSize )

Sets the UART transmit data count in the case of DMA mode operation.

##### Description

This function sets the size of the desired data transmission through the UART. This function is to be used when the UART is configured for DMA mode of transfer.

##### Return value

None

##### See also

[CyFx3BootUartRxSetBlockXfer](#)

##### Parameters

<code>txSize</code>	Desired transfer size.
---------------------	------------------------

## 5.9 firmware/boot\_fw/include/cyfx3usb.h File Reference

The FX3 boot library implements a USB driver that supports the device mode of USB operation (no host or OTG support). This file defines the data structures and APIs provided by the USB driver to control the USB operation.

```
#include <cyu3types.h>
#include <cyfx3error.h>
#include <cyu3externcstart.h>
#include <cyu3externcend.h>
```

## Data Structures

- struct [CyU3PUsbDescrPtrs](#)  
*Pointer to the various descriptors.*
- struct [CyFx3BootUsbEp0Pkt\\_t](#)  
*USB Setup Packet data structure.*
- struct [CyFx3BootUsbEpConfig\\_t](#)  
*Endpoint configuration information.*

## Macros

- #define [CY\\_FX3\\_USB\\_MAX\\_STRING\\_DESC\\_INDEX](#) (16)  
*Number of string descriptors that are supported by the FX3 booter.*

## Typedefs

- typedef enum [CyFx3BootUsbSpeed\\_t](#) [CyFx3BootUsbSpeed\\_t](#)  
*List of supported USB connection speeds.*
- typedef enum [CyFx3BootUsbEventType\\_t](#) [CyFx3BootUsbEventType\\_t](#)  
*Enumeration of USB event types.*
- typedef enum [CyFx3BootUsbEpType\\_t](#) [CyFx3BootUsbEpType\\_t](#)  
*Enumeration of the endpoint types.*

- **typedef enum CyU3PUSBSetDescType\_t CyU3PUSBSetDescType\_t**  
*Virtual descriptor types to be used to set descriptors.*
  - **typedef enum CyU3PUsbDescType CyU3PUsbDescType**  
*Enumeration of USB descriptor types.*
  - **typedef enum CyU3PUsbLinkPowerMode CyU3PUsbLinkPowerMode**  
*List of USB 3.0 Link Power States.*
  - **typedef struct CyU3PUsbDescrPtrs CyU3PUsbDescrPtrs**  
*Pointer to the various descriptors.*
  - **typedef struct CyFx3BootUsbEp0Pkt\_t CyFx3BootUsbEp0Pkt\_t**  
*USB Setup Packet data structure.*
  - **typedef struct CyFx3BootUsbEpConfig\_t CyFx3BootUsbEpConfig\_t**  
*Endpoint configuration information.*
  - **typedef void(\* CyFx3BootUSBEEventCb\_t) (CyFx3BootUsbEventType\_t event)**  
*USB Events notification callback.*
  - **typedef void(\* CyFx3BootUSBSetupCb\_t) (uint32\_t setupDat0, uint32\_t setupDat1)**  
*USB setup request handler type.*

## Enumerations

- enum CyFx3BootUsbSpeed\_t { CY\_FX3\_BOOT\_NOT\_CONNECTED = 0x00, CY\_FX3\_BOOT\_FULL\_SPEED, CY\_FX3\_BOOT\_HIGH\_SPEED, CY\_FX3\_BOOT\_SUPER\_SPEED }

*List of supported USB connection speeds.*

  - enum CyFx3BootUsbEventType\_t {
 CY\_FX3\_BOOT\_USB\_CONNECT = 0x00, CY\_FX3\_BOOT\_USB\_DISCONNECT, CY\_FX3\_BOOT\_USB\_RESET, CY\_FX3\_BOOT\_USB\_SUSPEND,
 CY\_FX3\_BOOT\_USB\_RESUME, CY\_FX3\_BOOT\_USB\_IN\_SS\_DISCONNECT, CY\_FX3\_BOOT\_USB\_COMPLIANCE }

*Enumeration of USB event types.*

  - enum CyFx3BootUsbEpType\_t { CY\_FX3\_BOOT\_USB\_EP\_CONTROL = 0, CY\_FX3\_BOOT\_USB\_EP\_ISO, CY\_FX3\_BOOT\_USB\_EP\_BULK, CY\_FX3\_BOOT\_USB\_EP\_INTR }

*Enumeration of the endpoint types.*

  - enum CyU3PUSBSetDescType\_t {
 CY\_U3P\_USB\_SET\_SS\_DEVICE\_DESCR, CY\_U3P\_USB\_SET\_HS\_DEVICE\_DESCR, CY\_U3P\_USB\_SET\_DEVQUAL\_DESCR, CY\_U3P\_USB\_SET\_FS\_CONFIG\_DESCR,
 CY\_U3P\_USB\_SET\_HS\_CONFIG\_DESCR, CY\_U3P\_USB\_SET\_STRING\_DESCR, CY\_U3P\_USB\_SET\_SS\_CONFIG\_DESCR, CY\_U3P\_USB\_SET\_SS\_BOS\_DESCR,
 CY\_U3P\_USB\_SET\_SS\_DEVICE\_DESCR, CY\_U3P\_USB\_SET\_HS\_DEVICE\_DESCR, CY\_U3P\_USB\_SET\_DEVQUAL\_DESCR, CY\_U3P\_USB\_SET\_FS\_CONFIG\_DESCR,
 CY\_U3P\_USB\_SET\_HS\_CONFIG\_DESCR, CY\_U3P\_USB\_SET\_STRING\_DESCR, CY\_U3P\_USB\_SET\_SS\_CONFIG\_DESCR, CY\_U3P\_USB\_SET\_SS\_BOS\_DESCR,
 CY\_U3P\_USB\_SET\_OTG\_DESCR }

*Virtual descriptor types to be used to set descriptors.*

  - enum CyU3PUusbDescType {
 CY\_U3P\_USB\_DEVICE\_DESCR = 0x01, CY\_U3P\_USB\_CONFIG\_DESCR, CY\_U3P\_USB\_STRING\_DESCRIPTOR, CY\_U3P\_USB\_INTRFC\_DESCRIPTOR,
 CY\_U3P\_USB\_ENDPNT\_DESCRIPTOR, CY\_U3P\_USB\_DEVQUAL\_DESCRIPTOR, CY\_U3P\_USB\_OTHERSPEED\_DESCRIPTOR, CY\_U3P\_USB\_INTRFC\_POWER\_DESCRIPTOR,
 CY\_U3P\_BOS\_DESCRIPTOR = 0x0F, CY\_U3P\_DEVICE\_CAPB\_DESCRIPTOR, CY\_U3P\_USB\_HID\_DESCRIPTOR = 0x21, CY\_U3P\_USB\_REPORT\_DESCRIPTOR,
 CY\_U3P\_SS\_EP\_COMPN\_DESCRIPTOR = 0x30, CY\_U3P\_USB\_DEVICE\_DESCRIPTOR = 0x01, CY\_U3P\_USB\_CONFIG\_DESCRIPTOR, CY\_U3P\_USB\_STRING\_DESCRIPTOR,
 CY\_U3P\_USB\_INTRFC\_DESCRIPTOR, CY\_U3P\_USB\_ENDPNT\_DESCRIPTOR, CY\_U3P\_USB\_DEVQUAL\_DESCRIPTOR, CY\_U3P\_USB\_OTHERSPEED\_DESCRIPTOR,
 CY\_U3P\_USB\_INTRFC\_POWER\_DESCRIPTOR, CY\_U3P\_USB\_OTG\_DESCRIPTOR, CY\_U3P\_BOS\_DESCRIPTOR = 0x0F, CY\_U3P\_BOS\_CONFIG\_DESCRIPTOR }

```
CY_U3P_DEVICE_CAPB_DESCR,
CY_U3P_USB_HID_DESCR = 0x21, CY_U3P_USB_REPORT_DESCR, CY_U3P_SS_EP_COMPN_DE-
SCR = 0x30 }
```

*Enumeration of USB descriptor types.*

- enum CyU3PUsbLinkPowerMode {
 CyU3PUsbLPM\_U0 = 0, CyU3PUsbLPM\_U1, CyU3PUsbLPM\_U2, CyU3PUsbLPM\_U3,
 CyU3PUsbLPM\_COMP, CyU3PUsbLPM\_Unknown, CyU3PUsbLPM\_U0 = 0, CyU3PUsbLPM\_U1,
 CyU3PUsbLPM\_U2, CyU3PUsbLPM\_U3, CyU3PUsbLPM\_COMP, CyU3PUsbLPM\_Unknown }

*List of USB 3.0 Link Power States.*

## Functions

- CyFx3BootErrorCode\_t CyFx3BootUsbStart (CyBool\_t noReEnum, CyFx3BootUSBEVENTCb\_t cb)
 

*Start the USB driver.*
- CyFx3BootErrorCode\_t CyFx3BootUsbSetDesc (CyU3PUSBSetDescType\_t descType, uint8\_t desc\_index, uint8\_t \*desc)
 

*Register a USB descriptor with the booter.*
- void CyFx3BootUsbConnect (CyBool\_t connect, CyBool\_t ssEnable)
 

*Enable/Disable USB connection.*
- void CyFx3BootRegisterSetupCallback (CyFx3BootUSBSetupCb\_t callback)
 

*Function to register a USB setup request handler.*
- CyFx3BootErrorCode\_t CyFx3BootUsbGetEpCfg (uint8\_t ep, CyBool\_t \*isNak, CyBool\_t \*isStall)
 

*Retrieve the current state of the specified endpoint.*
- CyFx3BootErrorCode\_t CyFx3BootUsbSetClrFeature (uint32\_t sc, CyBool\_t isConfigured, CyFx3BootUsbEp0Pkt\_t \*pEp0)
 

*Set/Clear Feature USB Request handler.*
- CyFx3BootErrorCode\_t CyFx3BootUsbDmaXferData (uint8\_t epNum, uint32\_t address, uint32\_t length, uint32\_t timeout)
 

*Function to transfer USB endpoint data using DMA.*
- CyFx3BootUsbSpeed\_t CyFx3BootUsbGetSpeed (void)
 

*Get the connection speed at which USB is operating.*
- void CyFx3BootUsbStall (uint8\_t ep, CyBool\_t stall, CyBool\_t toggle)
 

*Set or clear the stall status of an endpoint.*
- void CyFx3BootUsbAckSetup (void)
 

*Complete the status handshake of a USB control request.*
- CyU3PUsbDescrPtrs \* CyFx3BootUsbGetDesc (void)
 

*Retrieves the pointer to the USB descriptors.*
- CyFx3BootErrorCode\_t CyFx3BootUsbSetEpConfig (uint8\_t ep, CyFx3BootUsbEpConfig\_t \*epinfo)
 

*Configure a USB endpoint's properties.*
- void CyFx3BootUsbVBattEnable (CyBool\_t enable)
 

*Configure USB block on FX3 to work off Vbatt power instead of Vbus.*
- void CyFx3BootUsbEp0StatusCheck (void)
 

*Function to check if the status stage of a EP0 transfer is pending.*
- void CyFx3BootUsbCheckUsb3Disconnect (void)
 

*This function checks if the device state is in USB 3.0 disconnect state.*
- void CyFx3BootUsbSendCompliancePatterns (void)
 

*This function is used to send the compliance patterns.*
- void CyFx3BootUsbLPMDisable (void)
 

*Disable acceptance of U1/U2 entry requests from USB 3.0 host.*
- void CyFx3BootUsbLPMEnable (void)
 

*Re-enable acceptance of U1/U2 entry requests from USB 3.0 host.*

- [CyU3PUsbLinkPowerMode](#) [CyFx3BootUsbGetLinkPowerState](#) (void)  
*Get the current USB 3.0 link power state.*
- void [CyFx3BootUsbSetLinkPowerState](#) ([CyU3PUsbLinkPowerMode](#) mode)  
*Trigger a change to the desired USB 3.0 link power state.*
- void [CyFx3BootUsbSigResume](#) (void)  
*Trigger a USB 2.0 resume signal.*
- void [CyFx3BootUsbHandleEvents](#) (void)  
*Handler that manages USB link state changes.*

### 5.9.1 Detailed Description

The FX3 boot library implements a USB driver that supports the device mode of USB operation (no host or OTG support). This file defines the data structures and APIs provided by the USB driver to control the USB operation.

### 5.9.2 Macro Definition Documentation

#### 5.9.2.1 #define CY\_FX3\_USB\_MAX\_STRING\_DESC\_INDEX (16)

Number of string descriptors that are supported by the FX3 booter.

##### Description

The FX3 booter is capable of storing pointers to and handling a number of string descriptors. This constant represents the number of strings that the booter is capable of handling and is currently fixed to 16.

### 5.9.3 Typedef Documentation

#### 5.9.3.1 typedef struct CyFx3BootUsbEp0Pkt\_t CyFx3BootUsbEp0Pkt\_t

USB Setup Packet data structure.

##### Description

Control Endpoint setup packet data structure.

#### 5.9.3.2 typedef struct CyFx3BootUsbEpConfig\_t CyFx3BootUsbEpConfig\_t

Endpoint configuration information.

##### Description

This structure holds all the properties of a USB endpoint. This structure is used to provide information about the desired configuration of various endpoints in the system.

#### 5.9.3.3 typedef void(\* CyFx3BootUSBEVENTCb\_t) (CyFx3BootUsbEventType\_t event )

USB Events notification callback.

##### Description

Type of callback function that is invoked to notify the USB events. The FX3 booter does the necessary handling of the USB events and the application should NOT block on this function.

#### 5.9.3.4 typedef void(\* CyFx3BootUSBSetupCb\_t) (uint32\_t setupDat0, uint32\_t setupDat1 )

USB setup request handler type.

**Description**

Type of callback function that is invoked to handle USB setup requests. The booter doesn't handle any of the standard/vendor requests and this handler is expected to handle all such requests.

**5.9.3.5 `typedef struct CyU3PUsbDescrPtrs CyU3PUsbDescrPtrs`**

Pointer to the various descriptors.

**Description**

This data structure stores pointers to the various USB descriptors. These pointers are set as part of the [CyFx3BootUsbSetDesc\(\)](#) function.

**5.9.3.6 `typedef enum CyU3PUsbDescType CyU3PUsbDescType`**

Enumeration of USB descriptor types.

**Description**

Descriptor types as defined in the USB Specification.

**5.9.3.7 `typedef enum CyU3PUsbLinkPowerMode CyU3PUsbLinkPowerMode`**

List of USB 3.0 Link Power States.

**Description**

This is a partial list of the possible USB 3.0 Link Power states. This structure only comprehends the steady states like U0, U1, U2; and does not include transient states which may be hit while moving from one steady state to another (like polling or recovery).

**5.9.3.8 `typedef enum CyU3PUSBSetDescType_t CyU3PUSBSetDescType_t`**

Virtual descriptor types to be used to set descriptors.

**Description**

The user application can register different device and configuration descriptors with the USB driver, to be used at various USB connection speeds. To support this, the `CyU3PUsbSetDesc` call accepts a descriptor type parameter that is different from the USB standard descriptor type value that is embedded in the descriptor itself. This enumerated type is to be used by the application to register various descriptors with the USB driver.

## 5.9.4 Enumeration Type Documentation

**5.9.4.1 `enum CyFx3BootUsbEpType_t`**

Enumeration of the endpoint types.

Enumerator

**`CY_FX3_BOOT_USB_EP_CONTROL`** Control Endpoint Type  
**`CY_FX3_BOOT_USB_EP_ISO`** Isochronous Endpoint Type  
**`CY_FX3_BOOT_USB_EP_BULK`** Bulk Endpoint Type  
**`CY_FX3_BOOT_USB_EP_INTR`** Interrupt Endpoint Type

**5.9.4.2 `enum CyFx3BootUsbEventType_t`**

Enumeration of USB event types.

Enumerator

**CY\_FX3\_BOOT\_USB\_CONNECT** USB Connect Event  
**CY\_FX3\_BOOT\_USB\_DISCONNECT** USB DisConnect Event  
**CY\_FX3\_BOOT\_USB\_RESET** USB Reset Event  
**CY\_FX3\_BOOT\_USB\_SUSPEND** USB Suspend Event  
**CY\_FX3\_BOOT\_USB\_RESUME** USB Resume Event  
**CY\_FX3\_BOOT\_USB\_IN\_SS\_DISCONNECT** Event to check if the device is in SS Disconnect State  
**CY\_FX3\_BOOT\_USB\_COMPLIANCE** USB Compliance Event

#### 5.9.4.3 enum CyFx3BootUsbSpeed\_t

List of supported USB connection speeds.

Enumerator

**CY\_FX3\_BOOT\_NOT\_CONNECTED** USB device not connected.  
**CY\_FX3\_BOOT\_FULL\_SPEED** USB full speed.  
**CY\_FX3\_BOOT\_HIGH\_SPEED** High speed.  
**CY\_FX3\_BOOT\_SUPER\_SPEED** Super speed.

#### 5.9.4.4 enum CyU3PUsbDescType

Enumeration of USB descriptor types.

##### Description

Descriptor types as defined in the USB Specification.

Enumerator

**CY\_U3P\_USB\_DEVICE\_DESCR** Super Speed Device descr  
**CY\_U3P\_USB\_CONFIG\_DESCR** Configuration  
**CY\_U3P\_USB\_STRING\_DESCR** String  
**CY\_U3P\_USB\_INTRFC\_DESCR** Interface  
**CY\_U3P\_USB\_ENDPNT\_DESCR** End Point  
**CY\_U3P\_USB\_DEVQUAL\_DESCR** Device Qualifier  
**CY\_U3P\_USB\_OTHERSPEED\_DESCR** Other Speed Configuration  
**CY\_U3P\_USB\_INTRFC\_POWER\_DESCR** Interface power descriptor  
**CY\_U3P\_BOS\_DESCR** BOS descriptor  
**CY\_U3P\_DEVICE\_CAPB\_DESCR** Device Capability descriptor  
**CY\_U3P\_USB\_HID\_DESCR** HID descriptor  
**CY\_U3P\_USB\_REPORT\_DESCR** Report descriptor  
**CY\_U3P\_SS\_EP\_COMPN\_DESCR** End Point companion descriptor  
**CY\_U3P\_USB\_DEVICE\_DESCR** Device descriptor  
**CY\_U3P\_USB\_CONFIG\_DESCR** Configuration descriptor  
**CY\_U3P\_USB\_STRING\_DESCR** String descriptor  
**CY\_U3P\_USB\_INTRFC\_DESCR** Interface descriptor  
**CY\_U3P\_USB\_ENDPNT\_DESCR** Endpoint descriptor  
**CY\_U3P\_USB\_DEVQUAL\_DESCR** Device Qualifier descriptor

**CY\_U3P\_USB\_OTHERSPEED\_DESCR** Other Speed Configuration descriptor  
**CY\_U3P\_USB\_INTRFC\_POWER\_DESCR** Interface power descriptor descriptor  
**CY\_U3P\_USB\_OTG\_DESCR** OTG descriptor  
**CY\_U3P\_BOS\_DESCR** BOS descriptor  
**CY\_U3P\_DEVICE\_CAPB\_DESCR** Device Capability descriptor  
**CY\_U3P\_USB\_HID\_DESCR** HID descriptor  
**CY\_U3P\_USB\_REPORT\_DESCR** Report descriptor  
**CY\_U3P\_SS\_EP\_COMPN\_DESCR** Endpoint companion descriptor

#### 5.9.4.5 enum CyU3PUsbLinkPowerMode

List of USB 3.0 Link Power States.

##### Description

This is a partial list of the possible USB 3.0 Link Power states. This structure only comprehends the steady states like U0, U1, U2; and does not include transient states which may be hit while moving from one steady state to another (like polling or recovery).

##### Enumerator

**CyU3PUsbLPM\_U0** U0 (active) power state.  
**CyU3PUsbLPM\_U1** U1 power state.  
**CyU3PUsbLPM\_U2** U2 power state.  
**CyU3PUsbLPM\_U3** U3 (suspend) power state.  
**CyU3PUsbLPM\_COMP** Compliance state.  
**CyU3PUsbLPM\_Unknown** The link is not in any of the Ux states. This normally happens while the link training is ongoing.  
**CyU3PUsbLPM\_U0** U0 (active) power state.  
**CyU3PUsbLPM\_U1** U1 power state.  
**CyU3PUsbLPM\_U2** U2 power state.  
**CyU3PUsbLPM\_U3** U3 (suspend) power state.  
**CyU3PUsbLPM\_COMP** Compliance state.  
**CyU3PUsbLPM\_Unknown** The link is not in any of the Ux states. This normally happens while the link training is ongoing.

#### 5.9.4.6 enum CyU3PUSBSetDescType\_t

Virtual descriptor types to be used to set descriptors.

##### Description

The user application can register different device and configuration descriptors with the USB driver, to be used at various USB connection speeds. To support this, the CyU3PUsbSetDesc call accepts a descriptor type parameter that is different from the USB standard descriptor type value that is embedded in the descriptor itself. This enumerated type is to be used by the application to register various descriptors with the USB driver.

##### Enumerator

**CY\_U3P\_USB\_SET\_SS\_DEVICE\_DESCR** SuperSpeed (USB 3.0) device descriptor.  
**CY\_U3P\_USB\_SET\_HS\_DEVICE\_DESCR** USB 2.0 device descriptor.  
**CY\_U3P\_USB\_SET\_DEVQUAL\_DESCR** Device qualifier descriptor  
**CY\_U3P\_USB\_SET\_FS\_CONFIG\_DESCR** Full Speed configuration descriptor.

**CY\_U3P\_USB\_SET\_HS\_CONFIG\_DESCR** High Speed configuration descriptor.  
**CY\_U3P\_USB\_SET\_STRING\_DESCR** String descriptor.  
**CY\_U3P\_USB\_SET\_SS\_CONFIG\_DESCR** SuperSpeed configuration descriptor.  
**CY\_U3P\_USB\_SET\_SS\_BOS\_DESCR** BOS descriptor.  
**CY\_U3P\_USB\_SET\_SS\_DEVICE\_DESCR** Device descriptor for SuperSpeed.  
**CY\_U3P\_USB\_SET\_HS\_DEVICE\_DESCR** Device descriptor for Hi-Speed and Full Speed.  
**CY\_U3P\_USB\_SET\_DEVQUAL\_DESCR** Device Qualifier descriptor.  
**CY\_U3P\_USB\_SET\_FS\_CONFIG\_DESCR** Configuration descriptor for Full Speed.  
**CY\_U3P\_USB\_SET\_HS\_CONFIG\_DESCR** Configuration descriptor for Hi-Speed.  
**CY\_U3P\_USB\_SET\_STRING\_DESCR** String Descriptor  
**CY\_U3P\_USB\_SET\_SS\_CONFIG\_DESCR** Configuration descriptor for SuperSpeed.  
**CY\_U3P\_USB\_SET\_SS\_BOS\_DESCR** BOS descriptor.  
**CY\_U3P\_USB\_SET\_OTG\_DESCR** OTG descriptor.

## 5.9.5 Function Documentation

### 5.9.5.1 void CyFx3BootRegisterSetupCallback ( *CyFx3BootUSBSetupCb\_t callback* )

Function to register a USB setup request handler.

#### Description

This function is used to register a USB setup request handler with the booter. This setup request handler is expected to handle all the USB standard/vendor requests.

#### Return value

None

#### See also

[CyFx3BootUSBSetupCb\\_t](#)

#### Parameters

<i>callback</i>	Setup request handler
-----------------	-----------------------

### 5.9.5.2 void CyFx3BootUsbAckSetup ( *void* )

Complete the status handshake of a USB control request.

#### Description

This function is used to complete the status handshake of a USB control request that does not involve any data transfer.

### 5.9.5.3 void CyFx3BootUsbCheckUsb3Disconnect ( *void* )

This function checks if the device state is in USB 3.0 disconnect state.

#### Description

This function checks if the device is in USB 3.0 disconnect state. If this is the case then a fallback to USB 2.0 is attempted. The function returns immediately if the device is not in the disconnect state.

As this check cannot be done in the ISR context this has been deferred to the application context.

**Return value**

None

**5.9.5.4 void CyFx3BootUsbConnect ( CyBool\_t connect, CyBool\_t ssEnable )**

Enable/Disable USB connection.

**Description**

This function is used to enable the USB PHYs and to control the connection to the USB host in that manner.

The connect parameter enables/disables the USB connection.

The ssEnable parameter controls the SS or HS/FS enumeration. If SS enumeration is tried and fails, the device automatically falls back to HS/FS mode.

**Return value**

None

**Parameters**

<i>connect</i>	CyFalse - Disable USB Connection; CyTrue - Enable USB Connection
<i>ssEnable</i>	CyFalse - HS/FS Enumeration; CyTrue - SS Enumeration

**5.9.5.5 CyFx3BootErrorCode\_t CyFx3BootUsbDmaXferData ( uint8\_t epNum, uint32\_t address, uint32\_t length, uint32\_t timeout )**

Function to transfer USB endpoint data using DMA.

**Description**

This function can be used to transfer data on a USB endpoint. This works for the control endpoint as well as for other endpoints. The function does not validate the parameters, and can cause a fatal lock-up if called with the wrong parameters.

**Return value**

CY\_FX3\_BOOT\_SUCCESS - in case of successful DMA transfer.

CY\_FX3\_BOOT\_ERROR\_INVALID\_DMA\_ADDR - if the address specified is not in the SYSMEM area.

CY\_FX3\_BOOT\_ERROR\_XFER\_FAILURE - in case of DMA transfer failure.

CY\_FX3\_BOOT\_ERROR\_TIMEOUT - in case of DMA transfer times out.

**Parameters**

<i>epNum</i>	Endpoint number to/from which to transfer data. The MSB is used to identify whether this is a read or a write transfer.
<i>address</i>	SYSMEM address from/to which data is to be transferred.
<i>length</i>	Length of the transfer in bytes.
<i>timeout</i>	Timeout in multiples of 100 us. Also refer the macros CY_FX3_BOOT_NO_WAIT and CY_FX3_BOOT_WAIT_FOREVER

**5.9.5.6 void CyFx3BootUsbEp0StatusCheck ( void )**

Function to check if the status stage of a EP0 transfer is pending.

**Description**

This function is used to check if the status stage of a EP0 transfer is pending. In order for the device to handle the power management appropriately the device doesn't go into low power modes while a EP0 request is pending.

As this check cannot be done in the ISR context this has been deferred to the application context. This is applicable

when the device is functioning in SuperSpeed only.

#### Return value

None

### 5.9.5.7 CyU3PUsbDescrPtrs\* CyFx3BootUsbGetDesc( void )

Retrieves the pointer to the USB descriptors.

#### Description

The booter stores the descriptor pointers that are passed in as part of the [CyFx3BootUsbSetDesc\(\)](#) function. This function is used to retrieve pointer of type [CyU3PUsbDescrPtrs](#). This pointer can be used to retrieve the relevant data while handling the standard USB requests.

#### Return value

Pointer of type [CyU3PUsbDescrPtrs](#) on Success.  
NULL on Failure

#### See also

[CyFx3BootUsbSetDesc](#)

### 5.9.5.8 CyFx3BootErrorCode\_t CyFx3BootUsbGetEpCfg( uint8\_t ep, CyBool\_t \* isNak, CyBool\_t \* isStall )

Retrieve the current state of the specified endpoint.

#### Description

This function retrieves the current NAK and STALL status of the specified endpoint. The isNak return value will be CyTrue if the endpoint is forced to NAK all requests. The isStall return value will be CyTrue if the endpoint is currently stalled.

#### Return value

CY\_FX3\_BOOT\_SUCCESS - On Success.  
CY\_FX3\_BOOT\_ERROR\_NULL\_POINTER - If both isNak and isStall parameters are NULL.

#### Parameters

<i>ep</i>	Endpoint number to query.
<i>isNak</i>	Return parameter which will be filled with the NAK status.
<i>isStall</i>	Return parameter which will be filled with the STALL status.

### 5.9.5.9 CyU3PUsbLinkPowerMode CyFx3BootUsbGetLinkPowerState( void )

Get the current USB 3.0 link power state.

#### Description

This function retrieves the current USB 3.0 link power state, so that the application can decide whether to trigger a state change.

#### Return value

Current USB 3.0 link power state. Will return Unknown if the link is in polling or recovery.

#### See also

[CyU3PUsbLinkPowerMode](#)

**5.9.5.10 CyFx3BootUsbSpeed\_t CyFx3BootUsbGetSpeed( void )**

Get the connection speed at which USB is operating.

**Description**

This function is used to get the operating speed of the active USB connection.

**Return value**

Current USB connection speed.

**5.9.5.11 void CyFx3BootUsbHandleEvents( void )**

Handler that manages USB link state changes.

**Description**

This is an utility function that can be called periodically to have all USB 3.0 and 2.0 link states to be managed by the driver. Calling this function is equivalent to calling the CyFx3BootUsbCheckUsb3Disconnect, CyFx3BootUsbEp0StatusCheck and CyFx3BootUsbSendCompliancePatterns functions.

This function also ensures that the USB link does not get stuck in a low power mode for a long time.

**Return value**

None

**5.9.5.12 void CyFx3BootUsbLPMDisable( void )**

Disable acceptance of U1/U2 entry requests from USB 3.0 host.

**Description**

In some cases, accepting U1/U2 entry requests from the USB 3.0 host continuously can limit the performance that can be achieved through the FX3. This API can be used to temporarily disable the acceptance of U1/U2 requests by the FX3 device.

**Return value**

None

**See also**

[CyFx3BootUsbLPMEnable](#)

**5.9.5.13 void CyFx3BootUsbLPMEnable( void )**

Re-enable acceptance of U1/U2 entry requests from USB 3.0 host.

**Description**

This API is used to re-enable acceptance of U1/U2 entry requests by the FX3 device. It is required that LPM acceptance be kept enabled whenever a new USB connection is started up. This is required for passing USB compliance tests.

**Return value**

None

**See also**

[CyFx3BootUsbLPMDisable](#)

**5.9.5.14 void CyFx3BootUsbSendCompliancePatterns( void )**

This function is used to send the compliance patterns.

**Description**

This function is used to send the USB 3.0 compliance patterns. If the device's LTSSM state is currently in the compliance state, then compliance patterns are to be sent. This function returns when the LTSSM state is no longer in the compliance state.

As this task cannot be done in the ISR context this has been deferred to the application context.

**Return value**

None

**5.9.5.15 CyFx3BootErrorCode\_t CyFx3BootUsbSetClrFeature ( uint32\_t sc, CyBool\_t isConfigured, CyFx3BootUsbEp0Pkt\_t \* pEp0 )**

Set/Clear Feature USB Request handler.

**Description**

This function handles standard SET\_FEATURE and CLEAR\_FEATURE commands from the USB host. The requests handled include EP\_HALT, FUNCTION\_SUSPEND, Ux\_ENABLE, LTM\_ENABLE and TEST\_MODE features.

**Return value**

CY\_FX3\_BOOT\_SUCCESS - on success.  
CY\_FX3\_BOOT\_ERROR\_FAILURE - on failure.

**Parameters**

<i>sc</i>	1 - Set feature, 0 - Clear feature.
<i>isConfigured</i>	Whether the device is in configured (SET_CONFIG) state.
<i>pEp0</i>	Pointer to the control request structure.

**5.9.5.16 CyFx3BootErrorCode\_t CyFx3BootUsbSetDesc ( CyU3PUSBSetDescType\_t descType, uint8\_t desc\_index, uint8\_t \* desc )**

Register a USB descriptor with the booter.

**Description**

This function is used to register a USB descriptor with the booter. The booter is capable of remembering one descriptor each of the various supported types as well as upto 16 different string descriptors.

The booter only stores the descriptor pointers that are passed in to this function, and does not make copies of the descriptors. The caller therefore should not free up these descriptor buffers while the USB booter is active.

**Note\*\***

If the noReEnum option to the CyFx3BootUsbStart API is used, this API makes a copy of the descriptor to pass to the full firmware application. Since a finite memory space (total space of 3 KB) is available for these copied descriptors; this API will return an error when all of the available memory has been used up.

**Return value**

CY\_FX3\_BOOT\_SUCCESS - On Success.  
CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT - Bad descriptor index.  
CY\_FX3\_BOOT\_ERROR\_BAD\_DESCRIPTOR\_TYPE - Bad descriptor type.  
CY\_FX3\_BOOT\_ERROR\_MEMORY\_ERROR - out of memory when copying descriptors.

**Parameters**

<i>descType</i>	Type of the descriptor
<i>desc_index</i>	Descriptor index for string descriptors only.
<i>desc</i>	Pointer to the descriptor.

### 5.9.5.17 CyFx3BootErrorCode\_t CyFx3BootUsbSetEpConfig ( uint8\_t ep, CyFx3BootUsbEpConfig\_t \* epinfo )

Configure a USB endpoint's properties.

#### Description

The FX3 device has 30 user configurable endpoints (1-OUT to 15-OUT and 1-IN to 15-IN) which can be separately selected and configured with desired properties such as endpoint type and the maximum packet size.

All of these endpoints are kept disabled by default. This function is used to enable and set the properties for a specified endpoint. Separate calls need to be made to enable and configure each endpoint that needs to be used.

Note\*\*

Only BULK endpoints are supported by the boot firmware as of now.

#### Return value

CY\_FX3\_BOOT\_SUCCESS - when the call is successful.

CY\_FX3\_BOOT\_ERROR\_NULL\_POINTER - when the epinfo pointer is NULL.

CY\_FX3\_BOOT\_ERROR\_NOT\_SUPPORTED - if eptype is non-bulk endpoint.

#### Parameters

<i>ep</i>	Endpoint number to configured.
<i>epinfo</i>	EP configuration information

### 5.9.5.18 void CyFx3BootUsbSetLinkPowerState ( CyU3PUsbLinkPowerMode mode )

Trigger a change to the desired USB 3.0 link power state.

#### Description

This function triggers a USB 3.0 link power state change from U1/U2/U3 to U0; or from U0 to U1/U2. This function will do nothing if the USB 3.0 link is not active; or if it is in a polling or recovery state.

#### Return value

None

#### See also

[CyU3PUsbLinkPowerMode](#)

[CyFx3BootUsbGetLinkPowerState](#)

#### Parameters

<i>mode</i>	USB 3.0 link power mode to switch to.
-------------	---------------------------------------

### 5.9.5.19 void CyFx3BootUsbSigResume ( void )

Trigger a USB 2.0 resume signal.

#### Description

This function triggers a USB 2.0 resume signal to initiate a remote wake. The resume signal will be driven for about 20 ms if the link is suspended, and for about 50 us if the link is in the LPM-L1 state. The API does not check whether the link is actually suspended. It is the caller's responsibility to call it at the appropriate time.

This function should not be called directly from an ISR or callback function.

#### Return value

None

### 5.9.5.20 void CyFx3BootUsbStall ( uint8\_t ep, CyBool\_t stall, CyBool\_t toggle )

Set or clear the stall status of an endpoint.

#### Description

This function is to set or clear the stall status of a given endpoint. This function is to be used in response to SET\_FEATURE and CLEAR\_FEATURE requests from the host as well as for interface specific error handling. When the stall condition is being cleared, the data toggles for the endpoint can also be cleared. While an option is provided to leave the data toggles unmodified, this should only be used under specific conditions as recommended by Cypress.

#### Return value

None

#### Parameters

<i>ep</i>	Endpoint number to be modified.
<i>stall</i>	CyTrue: Set the stall condition, CyFalse: Clear the stall
<i>toggle</i>	CyTrue: Clear the data toggles in a Clear Stall call

### 5.9.5.21 CyFx3BootErrorCode\_t CyFx3BootUsbStart ( CyBool\_t noReEnum, CyFx3BootUSBEVENTCb\_t cb )

Start the USB driver.

#### Description

This function is used to start the USB driver of the booter. This function powers on the USB block on the FX3 device, if it is not already on. The application can register for the USB events of type CyFx3BootUSBEVENTCb\_t.

#### Return value

CY\_FX3\_BOOT\_ERROR\_NO\_REENUM\_REQUIRED - indicates that the USB block has been left on when transferring control from the main FX3 application to the boot firmware. CY\_FX3\_BOOT\_SUCCESS - USB block and driver have been started up.

#### Parameters

<i>noReEnum</i>	CyTrue: Do not re-enumerate if the USB connection is active; CyFalse: Force re-enumeration by resetting the USB block.
<i>cb</i>	USB Event handler.

### 5.9.5.22 void CyFx3BootUsbVBattEnable ( CyBool\_t enable )

Configure USB block on FX3 to work off Vbatt power instead of Vbus.

#### Description

The USB block on the FX3 device can be configured to work off Vbus power or Vbatt power, with the Vbus power being the default setting. This function is used to enable/disable the Vbatt power input to the USB block.

Note\*\*

This call is expected to be done prior to the CyFx3BootUsbConnect.

#### Return value

None

#### Parameters

<i>enable</i>	CyTrue: Work off VBatt, CyFalse: Do not work off VBatt
---------------	--

---

**Parameters**

<i>src</i>	Pointer to source buffer.
<i>count</i>	Size of data to be copied, in bytes.

**5.10.2.3 void CyFx3BootMemSet ( *uint8\_t \* buf, uint8\_t value, uint32\_t count*  )**

Initialize a memory block with fixed data.

**Description**

This function is a memset equivalent that initializes all bytes in a memory block to the same value. The memset is done by updating each byte one at a time.

**Return Value**

None

**Parameters**

<i>buf</i>	Pointer to memory buffer.
<i>value</i>	Value to be set at each byte.
<i>count</i>	Size of the memory buffer, in bytes.

**5.10.2.4 CyFx3BootErrorCode\_t CyFx3BootSNPrintf ( *uint8\_t \* buffer, uint16\_t maxLength, char \* fmt, ...*  )**

Print a formatted string to a buffer.

**Description**

This function is a reduced version of the snprintf C library function, and prints a formatted string into a buffer based on the input arguments.

The function only supports a subset of the output conversion specifications supported by the snprintf function. The 'c', 'd', 'u', 'x' and 's' conversion specifications are supported. There is no support for float or double formats, or for flags, precision or type modifiers.

**Return Value**

CY\_FX3\_BOOT\_SUCCESS if the formatted output is stored correctly. CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT if arguments are NULL, or if the target buffer is too short.

**Parameters**

<i>buffer</i>	Buffer into which the formatted string is to be printed.
<i>maxLength</i>	Length of the output buffer.
<i>fmt</i>	Format string.

**5.11 firmware/u3p\_firmware/inc/cyfx3\_api.h File Reference**

Defines the APIs provided by the FX3 core library.

```
#include <cyu3types.h>
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

## Data Structures

- struct [CyU3PloMatrixConfig\\_t](#)  
*IO matrix configuration parameters.*

## Typedefs

- typedef enum [CyU3PPartNumber\\_t CyU3PPartNumber\\_t](#)  
*Enumeration of EZ-USB FX3 part numbers.*
- typedef enum [CyU3PloMatrixLppMode\\_t CyU3PloMatrixLppMode\\_t](#)  
*Defines the enumerations for LPP IO line configurations.*
- typedef enum [CyU3PSPortMode\\_t CyU3PSPortMode\\_t](#)  
*Define the various operating modes for the storage ports on the FX3S device.*
- typedef struct [CyU3PloMatrixConfig\\_t CyU3PloMatrixConfig\\_t](#)  
*IO matrix configuration parameters.*

## Enumerations

- enum [CyU3PPartNumber\\_t {  
 CYPART\\_USB3014 = 0, CYPART\\_USB3012, CYPART\\_USB3013, CYPART\\_USB3011,  
 CYPART\\_USB3035, CYPART\\_USB3034, CYPART\\_USB3033, CYPART\\_USB3032,  
 CYPART\\_USB3031, CYPART\\_WB0263, CYPART\\_WB0163, CYPART\\_USB2035,  
 CYPART\\_USB2034, CYPART\\_USB2033, CYPART\\_USB2032, CYPART\\_USB2031,  
 CYPART\\_USB3025, CYPART\\_USB3024, CYPART\\_USB3023, CYPART\\_USB3021,  
 CYPART\\_USB2025, CYPART\\_USB2024, CYPART\\_USB2023, CYPART\\_USB3061,  
 CYPART\\_USB3062, CYPART\\_USB3063, CYPART\\_USB3064, CYPART\\_USB3065,  
 CYPART\\_USB2064, CYPART\\_USB2014, CYPART\\_USB2013, CYPART\\_USB2011,  
 CYPART\\_USB3075, CYPART\\_LASTDEV }](#)  
*Enumeration of EZ-USB FX3 part numbers.*
- enum [CyU3PloMatrixLppMode\\_t {  
 CY\\_U3P\\_IO\\_MATRIX\\_LPP\\_DEFAULT = 0, CY\\_U3P\\_IO\\_MATRIX\\_LPP\\_UART\\_ONLY, CY\\_U3P\\_IO\\_MATRIX\\_LPP\\_SPI\\_ONLY, CY\\_U3P\\_IO\\_MATRIX\\_LPP\\_I2S\\_ONLY, CY\\_U3P\\_IO\\_MATRIX\\_LPP\\_NONE }](#)  
*Defines the enumerations for LPP IO line configurations.*
- enum [CyU3PSPortMode\\_t { CY\\_U3P\\_SPORT\\_INACTIVE = 0, CY\\_U3P\\_SPORT\\_4BIT, CY\\_U3P\\_SPORT\\_8BIT, CY\\_U3P\\_SPORT\\_1BIT }](#)  
*Define the various operating modes for the storage ports on the FX3S device.*

## Functions

- void [CyFx3SetCpuFreq \(uint32\\_t cpuFreq\)](#)  
*Store the CPU clock frequency for use in BusyWait delay calculation.*
- void [CyFx3BusyWait \(uint16\\_t usWait\)](#)  
*Delay function.*
- void [CyFx3DevInitPageTables \(void\)](#)  
*Setup the default page tables for the FX3 device.*
- void [CyFx3DevClearSwInterrupt \(void\)](#)  
*Clear a FX3 device specific software interrupt.*
- [CyU3PPartNumber\\_t CyFx3DevIdentifyPart \(void\)](#)  
*Get the current FX3/FX3S device part number.*
- [CyBool\\_t CyFx3DevsUsb3Supported \(void\)](#)  
*Check if the part being used supports the USB 3.0 peripheral functionality.*

- **CyBool\_t CyFx3DevIsOtgSupported (void)**  
*Check if the part being used supports the USB OTG and USB 2.0 host functionality.*
- **CyBool\_t CyFx3DevIsRam512Supported (void)**  
*Check if the part being used supports 512 KB of System RAM.*
- **CyBool\_t CyFx3DevIsGpifSupported (void)**  
*Check if the part supports the PIB/GPIF II interface.*
- **CyBool\_t CyFx3DevIsGpif32Supported (void)**  
*Check if the part supports a 32 bit wide GPIF II data bus.*
- **CyBool\_t CyFx3DevIsSib0Supported (void)**  
*Check if the part supports the S0 storage port.*
- **CyBool\_t CyFx3DevIsSib1Supported (void)**  
*Check if the part supports the S1 storage port.*
- **CyBool\_t CyFx3DevIsI2sSupported (void)**  
*Check if the part supports the I2S interface.*
- **CyBool\_t CyFx3DevIsMipicsiSupported (void)**  
*Check if the part supports the MIPI CSI interface.*
- **uint8\_t CyFx3DevGetMipiLaneCount (void)**  
*Get the number of CSI lanes supported by the CX3 part in use.*
- **CyBool\_t CyFx3DevIOIsI2cConfigured (void)**  
*Check if I2C is enabled in the current IO configuration.*
- **CyBool\_t CyFx3DevIOIsUartConfigured (void)**  
*Check if UART is enabled in the current IO configuration.*
- **CyBool\_t CyFx3DevIOIsI2sConfigured (void)**  
*Check if I2S is enabled in the current IO configuration.*
- **CyBool\_t CyFx3DevIOIsSpiConfigured (void)**  
*Check if SPI is enabled in the current IO configuration.*
- **CyBool\_t CyFx3DevIOIsSib0Configured (void)**  
*Check if the SIB0 port is configured.*
- **CyBool\_t CyFx3DevIOIsSib1Configured (void)**  
*Check if the SIB1 port is configured.*
- **void CyFx3DevIODisableSib0 (void)**  
*Disable SIB0 port.*
- **void CyFx3DevIODisableSib1 (void)**  
*Disable SIB1 port.*
- **CyBool\_t CyFx3DevIOIsSib8BitWide (uint8\_t portId)**  
*Check if the specified SIB port is configured for 8 bit operation.*
- **CyBool\_t CyFx3DevIOIsSib1BitWide (uint8\_t portId)**  
*Check if the specified SIB port is restricted to 1 bit operation.*
- **CyBool\_t CyFx3DevIOIsGpio (uint32\_t gpioId, CyBool\_t isSimple)**  
*Check if the specified GPIO is configured as a GPIO.*
- **void CyFx3DevIOSelectGpio (uint8\_t gpioId, CyBool\_t enable, CyBool\_t isSimple)**  
*Control the override of a IO pin as a simple or complex GPIO.*
- **CyBool\_t CyFx3DevIOConfigure (CyU3PloMatrixConfig\_t \*cfg\_p)**  
*Configure the IO matrix on the device.*
- **CyBool\_t CyFx3UsbIsOn (void)**  
*Check whether the USB block is powered ON.*
- **void CyFx3UsbPowerOn (void)**  
*Power on the USB block on the FX3/FX3S device.*
- **void CyFx3UsbWritePhyReg (uint16\_t phyAddr, uint16\_t phyVal)**  
*Write to a USB 3.0 PHY Control Register.*
- **void CyFx3Usb2PhySetup (void)**

- void [CyFx3Usb3LnkSetup](#) (void)
 

*Configure the USB 2.0 PHY on FX3.*
- void [CyFx3Usb3SendTP](#) (uint32\_t \*tpData)
 

*Configure the USB 3.0 LINK on FX3.*
- void [CyFx3UsbDmaPrefetchEnable](#) (CyBool\_t streamEnable)
 

*Enable data prefetch from the DMA sockets on the USB egress (IN) data path.*
- void [CyFx3Usb3LnkRelaxHpTimeout](#) (void)
 

*Relax the USB 3.0 HP ACK Timeout period.*
- CyBool\_t [CyFx3PibIsOn](#) (void)
 

*Check whether the PIB block is powered ON.*
- void [CyFx3PibPowerOn](#) (void)
 

*Power on the PIB and GPIF II blocks on the FX3/FX3S device.*
- void [CyFx3PibPowerOff](#) (void)
 

*Power off the PIB and GPIF II blocks on the FX3/FX3S device.*
- void [CyFx3PibDlIEnable](#) (void)
 

*Enable the DLL in the PIB block.*
- uint16\_t [CyFx3PibGetDlIStatus](#) (void)
 

*Get the current status of the PIB DLL.*
- void [CyFx3SibPowerOn](#) (void)
 

*Power the storage interface block on the FX3S/SD3 device on.*
- void [CyFx3SibPowerOff](#) (void)
 

*Power the storage interface block on the FX3S/SD3 device off.*
- CyBool\_t [CyFx3LppIsOn](#) (void)
 

*Check if the LPP block is powered ON.*

### 5.11.1 Detailed Description

Defines the APIs provided by the FX3 core library.

#### Description

The FX3 Core library provides a minimal set of API which operate at the device level, and are restricted for IP protection. This header file defines the set of APIs provided by the core library.

### 5.11.2 Typedef Documentation

#### 5.11.2.1 [typedef struct CyU3PloMatrixConfig\\_t CyU3PloMatrixConfig\\_t](#)

IO matrix configuration parameters.

#### Description

The EZ-USB FX3 and FX3S devices have a flexible IO architecture that allows each IO Pin to serve multiple functions. The desired IO configuration for all of the pins needs to be specified before any of the FX3 internal blocks such as USB, GPIF or UART are powered on.

This structure captures the desired IO configuration for the FX3/FX3S device as a whole.

#### Note

A common structure including the storage port configuration is used for both FX3 and FX3S devices, in order to maintain a common API interface. The s0Mode and s1Mode fields should be set to CY\_U3P\_SPORT\_INACTIVE when using the EZ-USB FX3 device.

#### See also

[CyU3PloMatrixLppMode\\_t](#)  
[CyU3PSPortMode\\_t](#)  
[CyU3PDeviceConfigureLOMatrix](#)

### 5.11.2.2 `typedef enum CyU3PloMatrixLppMode_t CyU3PloMatrixLppMode_t`

Defines the enumerations for LPP IO line configurations.

#### Description

Most of the IO pins on the FX3 device are multi-purpose with the specific configuration being selected at system start-up. This enumeration lists the various IO operating modes relating to Serial peripheral interfaces.

The I2C interface is always available. If the GPIF data bus is configured as 32-bit wide, only the UART and I2S interfaces are available (SPI is not available). In this case, the configuration chosen should be CY\_U3P\_IO\_MATRIX\_LPP\_DEFAULT.

If the GPIF data bus is 8, 16 or 24 bits wide; it is possible to use all of the SPI, UART and I2S interfaces. However, the peripheral pins can be relocated in this case. Refer to the FX3 datasheet for more details and choose the desired configuration accordingly.

In the case of the FX3S device, none of the UART, SPI and I2S interfaces are available if the second storage port (S1) is used in 8-bit mode. In this case, the configuration chosen should CY\_U3P\_IO\_MATRIX\_LPP\_NONE.

#### See also

[CyU3PDeviceConfigureIMatrix](#)  
[CyU3PSPortMode\\_t](#)

### 5.11.2.3 `typedef enum CyU3PPPartNumber_t CyU3PPPartNumber_t`

Enumeration of EZ-USB FX3 part numbers.

#### Description

There are multiple EZ-USB FX3 parts which support varying feature sets. Please refer to the device data sheets or the Cypress device catalogue for information on the features supported by each FX3 part.

This enumerated type lists the various valid part numbers in the EZ-USB FX3 family.

#### See also

[CyU3PDeviceGetPartNumber](#)

### 5.11.2.4 `typedef enum CyU3PSPortMode_t CyU3PSPortMode_t`

Define the various operating modes for the storage ports on the FX3S device.

#### Description

The FX3S device supports two storage ports which can be connected to SD/eMMC/SDIO peripherals. Each of these ports can be configured in a variety of modes. This enumeration lists the possible IO configurations for each of these storage ports.

The selected storage port IO configuration has some implications on the selection of other IO configurations like lppMode. Please see the FX3S device datasheet to identify the supported combinations.

#### See also

[CyU3PDeviceConfigureIMatrix](#)  
[CyU3PloMatrixLppMode\\_t](#)

## 5.11.3 Enumeration Type Documentation

### 5.11.3.1 `enum CyU3PloMatrixLppMode_t`

Defines the enumerations for LPP IO line configurations.

**Description**

Most of the IO pins on the FX3 device are multi-purpose with the specific configuration being selected at system start-up. This enumeration lists the various IO operating modes relating to Serial peripheral interfaces.

The I2C interface is always available. If the GPIF data bus is configured as 32-bit wide, only the UART and I2S interfaces are available (SPI is not available). In this case, the configuration chosen should be CY\_U3P\_IO\_MATRIX ↔ RIX\_LPP\_DEFAULT.

If the GPIF data bus is 8, 16 or 24 bits wide; it is possible to use all of the SPI, UART and I2S interfaces. However, the peripheral pins can be relocated in this case. Refer to the FX3 datasheet for more details and choose the desired configuration accordingly.

In the case of the FX3S device, none of the UART, SPI and I2S interfaces are available if the second storage port (S1) is used in 8-bit mode. In this case, the configuration chosen should CY\_U3P\_IO\_MATRIX\_LPP\_NONE.

**See also**

[CyU3PDeviceConfigureIOMatrix](#)  
[CyU3PSPortMode\\_t](#)

**Enumerator**

**CY\_U3P\_IO\_MATRIX\_LPP\_DEFAULT** Default LPP mode where all peripherals are enabled.  
**CY\_U3P\_IO\_MATRIX\_LPP\_UART\_ONLY** LPP layout with GPIF 16-bit and UART only.  
**CY\_U3P\_IO\_MATRIX\_LPP\_SPI\_ONLY** LPP layout with GPIF 16-bit and SPI only.  
**CY\_U3P\_IO\_MATRIX\_LPP\_I2S\_ONLY** LPP layout with GPIF 16-bit and I2S only.  
**CY\_U3P\_IO\_MATRIX\_LPP\_NONE** FX3S specific configuration where UART, SPI and I2S are disabled.

**5.11.3.2 enum CyU3PPartNumber\_t**

Enumeration of EZ-USB FX3 part numbers.

**Description**

There are multiple EZ-USB FX3 parts which support varying feature sets. Please refer to the device data sheets or the Cypress device catalogue for information on the features supported by each FX3 part.

This enumerated type lists the various valid part numbers in the EZ-USB FX3 family.

**See also**

[CyU3PDeviceGetPartNumber](#)

**Enumerator**

**CYPART\_USB3014** CYUSB3014: 512 KB RAM; GPIF can be 32 bit; OTG and USB host supported  
**CYPART\_USB3012** CYUSB3012: 256 KB RAM; GPIF can be 32 bit  
**CYPART\_USB3013** CYUSB3013: 512 KB RAM; GPIF - 16 bit bus only  
**CYPART\_USB3011** CYUSB3011: 256 KB RAM; GPIF - 16 bit bus only  
**CYPART\_USB3035** CYUSB3035: 512 KB RAM; GPIF - 16 bit bus; Supports two SD/eMMC/SDIO ports  
**CYPART\_USB3034** CYUSB3034: 512 KB RAM; GPIF - 16 bit bus; Supports two SD/eMMC/SDIO ports  
**CYPART\_USB3033** CYUSB3033: 512 KB RAM; GPIF - 16 bit bus; Supports single SD/eMMC/SDIO port  
**CYPART\_USB3032** CYUSB3032: 256 KB RAM; GPIF - 16 bit bus; No USB host/OTG; Two storage ports  
**CYPART\_USB3031** CYUSB3031: 256 KB RAM; GPIF - 16 bit bus; No USB host/OTG; Single storage port  
**CYPART\_WB0263** CYWB0263 (Benicia): 512 KB RAM; USB 3.0; GPIF - 16 bit; Supports two SD/eMMC/SDIO ports  
**CYPART\_WB0163** CYWB0163 (Bay): 512 KB RAM; USB 2.0 only; GPIF - 16 bit; Supports two SD/eMMC/SDIO ports

**CYPART\_USB2035** CYUSB2035: 512 KB RAM; USB 2.0 only; GPIF - 16 bit; Supports two SD/eMMC/SDIO ports

**CYPART\_USB2034** CYUSB2034: 512 KB RAM; USB 2.0 only; GPIF - 16 bit; Supports two SD/eMMC/SDIO ports

**CYPART\_USB2033** CYUSB2033: 512 KB RAM; USB 2.0 only; GPIF - 16 bit; Supports single SD/eMMC/SDIO port

**CYPART\_USB2032** CYUSB2032: 256 KB RAM; USB 2.0 only; GPIF - 16 bit; No USB host/OTG; Two storage ports

**CYPART\_USB2031** CYUSB2031: 256 KB RAM; USB 2.0 only; GPIF - 16 bit; No USB host/OTG; Single storage port

**CYPART\_USB3025** CYUSB3025: 512 KB RAM; No GPIF port; USB host/OTG; Two storage ports

**CYPART\_USB3024** CYUSB3024: 512 KB RAM; No GPIF port; USB host/OTG; Two storage ports

**CYPART\_USB3023** CYUSB3023: 512 KB RAM; No GPIF port; No USB host/OTG; Single storage port

**CYPART\_USB3021** CYUSB3021: 256 KB RAM; No GPIF port; No USB host/OTG; Single storage port

**CYPART\_USB2025** CYUSB2025: 512 KB RAM; USB 2.0 only; No GPIF port; USB host/OTG; Two storage ports

**CYPART\_USB2024** CYUSB2024: 512 KB RAM; USB 2.0 only; No GPIF port; USB host/OTG; Two storage ports

**CYPART\_USB2023** CYUSB2023: 512 KB RAM; USB 2.0 only; No GPIF port; No USB host/OTG; Single storage port

**CYPART\_USB3061** CYUSB3061: 256 KB RAM; Fixed Function GPIF; 1 MIPI-CSI lane

**CYPART\_USB3062** CYUSB3062: 512 KB RAM; Fixed Function GPIF; 1 MIPI-CSI lane

**CYPART\_USB3063** CYUSB3063: 256 KB RAM; Fixed Function GPIF; 2 MIPI-CSI lanes

**CYPART\_USB3064** CYUSB3064: 512 KB RAM; Fixed Function GPIF; 2 MIPI-CSI lanes

**CYPART\_USB3065** CYUSB3065: 512 KB RAM; Fixed Function GPIF; 4 MIPI-CSI lanes

**CYPART\_USB2064** CYUSB2064: 512 KB RAM; USB 2.0 only; Fixed Function GPIF; 2 MIPI-CSI lanes

**CYPART\_USB2014** CYUSB2014: 512 KB RAM; USB 2.0 only; GPIF can be 32 bit; OTG and USB host supported

**CYPART\_USB2013** CYUSB2013: 512 KB RAM; USB 2.0 only; GPIF - 16 bit; OTG and USB host supported

**CYPART\_USB2011** CYUSB2011: 256 KB RAM; USB 2.0 only; GPIF - 16 bit; No USB host/OTG

**CYPART\_USB3075** CYUSB3075: 512 KB RAM; 4 MIPI-DSI lanes.

**CYPART\_LASTDEV** Unknown device type

### 5.11.3.3 enum CyU3PSPortMode\_t

Define the various operating modes for the storage ports on the FX3S device.

#### Description

The FX3S device supports two storage ports which can be connected to SD/eMMC/SDIO peripherals. Each of these ports can be configured in a variety of modes. This enumeration lists the possible IO configurations for each of these storage ports.

The selected storage port IO configuration has some implications on the selection of other IO configurations like lppMode. Please see the FX3S device datasheet to identify the supported combinations.

See also

[CyU3PDeviceConfigureIOMatrix](#)  
[CyU3PIoMatrixLppMode\\_t](#)

Enumerator

**CY\_U3P\_SPORT\_INACTIVE** Storage port not in use. Pins are available for other interfaces.

**CY\_U3P\_SPORT\_4BIT** SD/MMC interface with four bit data bus. Upper half of data bus is available for other interfaces.

**CY\_U3P\_SPORT\_8BIT** SD/MMC interface with eight bit data bus. All pins (CLK, CMD, D7-D0) are part of storage interface.

**CY\_U3P\_SPORT\_1BIT** SD/MMC interface with a single bit data bus. D1-D3 pins can be used as GPIO.

#### 5.11.4 Function Documentation

##### 5.11.4.1 void CyFx3BusyWait ( uint16\_t usWait )

Delay function.

###### Description

This function provides delays in multiples of 1 us. The ARM CPU is kept busy by this function, and this should not be used in cases where large delays are required.

###### Return Value

None

See also

[CyFx3SetCpuFreq](#)

###### Parameters

<i>usWait</i>	Delay required in micro-seconds.
---------------	----------------------------------

##### 5.11.4.2 void CyFx3DevClearSwInterrupt ( void )

Clear a FX3 device specific software interrupt.

###### Description

This function clears a device specific software interrupt. The software interrupt mechanism is not used in the SDK.

###### Return Value

None

##### 5.11.4.3 uint8\_t CyFx3DevGetMipiLaneCount ( void )

Get the number of CSI lanes supported by the CX3 part in use.

###### Description

Different CX3 parts can support different number of CSI lanes through image sensors can be connected. This API returns the number of CSI lanes that are supported by the CX3 part in use..

###### Return Value

Number of CSI lanes supported.

#### 5.11.4.4 CyU3PPartNumber\_t CyFx3DevIdentifyPart ( void )

Get the current FX3/FX3S device part number.

##### Description

This function gets the current FX3/FX3S part number by querying the device identification registers. This function can be used by the code to avoid calling functions that would cause failures because the Silicon does not support them.

##### Return Value

Device part number

#### 5.11.4.5 void CyFx3DevInitPageTables ( void )

Setup the default page tables for the FX3 device.

##### Description

The MMU on the FX3/FX3S device needs to be turned on for the data cache to work. This function sets up a default set of page tables that map each valid physical address to the corresponding virtual address, so that the device can operate with the MMU turned on.

##### Return Value

None

#### 5.11.4.6 CyBool\_t CyFx3DevIOConfigure ( CyU3PloMatrixConfig\_t \* cfg\_p )

Configure the IO matrix on the device.

##### Description

The FX3 and other devices in the family have a configurable IO matrix which allows multiplexing of multiple functions onto a IO pin at different times. This function validates the IO configuration specified by the user, and applies it if valid.

##### Return Value

CyTrue if the IO configuration is valid for the device and has been applied. CyFalse if the IO configuration is illegal or inconsistent.

##### Parameters

<i>cfg_p</i>	IO configuration parameters.
--------------	------------------------------

#### 5.11.4.7 void CyFx3DevIODisableSib0 ( void )

Disable SIB0 port.

##### Description

Disable SIB0 port for the current IO configuration of the FX3S/SD3 device.

#### 5.11.4.8 void CyFx3DevIODisableSib1 ( void )

Disable SIB1 port.

##### Description

Disable SIB1 port for the current IO configuration of the FX3S/SD3 device.

#### 5.11.4.9 CyBool\_t CyFx3DevIOIsGpio ( uint32\_t gpioid, CyBool\_t isSimple )

Check if the specified GPIO is configured as a GPIO.

##### Description

This function checks whether the specified FX3/FX3S GPIO pin has been configured as a GPIO of the specified type (simple or complex).

##### Return Value

CyTrue if the pin is currently configured as a GPIO of the specified type. CyFalse if the pin is invalid or if it is not configured as a GPIO.

##### Parameters

<i>gpioid</i>	GPIO whose state is to be queried.
<i>isSimple</i>	Whether to check for a simple GPIO (CyTrue) or a complex GPIO (CyFalse).

#### 5.11.4.10 CyBool\_t CyFx3DevIOIsI2cConfigured ( void )

Check if I2C is enabled in the current IO configuration.

##### Description

This function checks whether the I2C interface is enabled in the current IO configuration of the FX3 device.

##### Return Value

CyTrue if I2C is enabled. CyFalse if I2C is disabled.

#### 5.11.4.11 CyBool\_t CyFx3DevIOIsI2sConfigured ( void )

Check if I2S is enabled in the current IO configuration.

##### Description

This function checks whether the I2S interface is enabled in the current IO configuration of the FX3 device.

##### Return Value

CyTrue if I2S is enabled. CyFalse if I2S is disabled.

#### 5.11.4.12 CyBool\_t CyFx3DevIOIsSib0Configured ( void )

Check if the SIB0 port is configured.

##### Description

Check whether the SIB0 port is enabled in the current IO configuration of the FX3S/SD3 device.

##### Return Value

CyTrue if the SIB0 port is enabled in the IO configuration. CyFalse if the SIB0 port is not enabled in the IO configuration.

#### 5.11.4.13 CyBool\_t CyFx3DevIOIsSib1BitWide ( uint8\_t portId )

Check if the specified SIB port is restricted to 1 bit operation.

##### Description

The SIB ports on the FX3S/SD3 device can be configured for 1 bit, 4 bit or 8 bit data operation. This function checks whether the current IO configuration of the specified storage port is restricted to 1 bit.

##### Return Value

CyTrue if only 1 bit operation is allowed. CyFalse if data bus can be 4 or 8 bits wide.

**Parameters**

<i>port← Id</i>	Port being queried (0 or 1).
---------------------	------------------------------

**5.11.4.14 CyBool\_t CyFx3DevIOIsSib1Configured ( void )**

Check if the SIB1 port is configured.

**Description**

Check whether the SIB1 port is enabled in the current IO configuration of the FX3S/SD3 device.

**Return Value**

CyTrue if the SIB1 port is enabled in the IO configuration. CyFalse if the SIB1 port is not enabled in the IO configuration.

**5.11.4.15 CyBool\_t CyFx3DevIOIsSib8BitWide ( uint8\_t portId )**

Check if the specified SIB port is configured for 8 bit operation.

**Description**

The SIB ports on the FX3S/SD3 device can be configured for 1 bit, 4 bit or 8 bit data operation. This function checks whether the current IO configuration of the specified storage port allows 8 bit operation.

**Return Value**

CyTrue if 8 bit operation is allowed. CyFalse if 8 bit operation is not allowed.

**Parameters**

<i>port← Id</i>	Port being queried (0 or 1).
---------------------	------------------------------

**5.11.4.16 CyBool\_t CyFx3DevIOIsSpiConfigured ( void )**

Check if SPI is enabled in the current IO configuration.

**Description**

This function checks whether the SPI interface is enabled in the current IO configuration of the FX3 device.

**Return Value**

CyTrue if SPI is enabled. CyFalse if SPI is disabled.

**5.11.4.17 CyBool\_t CyFx3DevIOIsUartConfigured ( void )**

Check if UART is enabled in the current IO configuration.

**Description**

This function checks whether the UART interface is enabled in the current IO configuration of the FX3 device.

**Return Value**

CyTrue if UART is enabled. CyFalse if UART is disabled.

**5.11.4.18 void CyFx3DevIOSelectGpio ( uint8\_t gpioId, CyBool\_t enable, CyBool\_t isSimple )**

Control the override of a IO pin as a simple or complex GPIO.

**Description**

IO pins on the FX3/FX3S interface can be temporarily over-ridden as simple or complex GPIOs during device operation. This API is used to place or remove these GPIO override modes.

**Return Value**

None

**Parameters**

<i>gpioid</i>	GPIO to be modified.
<i>enable</i>	Whether the override is to be enabled (CyTrue) or removed (CyFalse).
<i>isSimple</i>	Whether over-riding as a simple (CyTrue) or a complex (CyFalse) GPIO. Is don't care when enable is CyFalse.

**5.11.4.19 CyBool\_t CyFx3DevsGpif32Supported( void )**

Check if the part supports a 32 bit wide GPIF II data bus.

**Description**

Some FX3 parts as well as the FX3S devices can only make use of the GPIF II interface with a 16 bit or narrower data bus. This function checks whether the part in use can support a 24 bit or 32 bit wide GPIF II data bus.

**Return Value**

CyTrue if GPIF II data bus can be 32 bits wide. CyFalse if GPIF II data bus can only be 16 bits or smaller.

**5.11.4.20 CyBool\_t CyFx3DevsGpifSupported( void )**

Check if the part supports the PIB/GPIF II interface.

**Description**

The FX3 SDK supports parts such as the SD3 and SD2 devices which do not provide a GPIF II interface. This function is used to check whether the part in use can support GPIF II functionality.

**Return Value**

CyTrue if GPIF II (PIB) is supported. CyFalse if GPIF II is not supported.

**5.11.4.21 CyBool\_t CyFx3DevsI2sSupported( void )**

Check if the part supports the I2S interface.

**Description**

Some parts in the FX3 family do not support the I2S interface. This function checks whether I2S is supported by the part in use.

**Return Value**

CyTrue if I2S interface is supported. CyFalse if I2S interface is not supported.

**5.11.4.22 CyBool\_t CyFx3DevsMipicsiSupported( void )**

Check if the part supports the MIPI CSI interface.

**Description**

Some parts in the FX3 family (CX3) support a MIPI CSI interface through which an image sensor can be connected. This function checks whether the part in use supports the CSI interface.

**Return Value**

CyTrue if the CSI interface is supported. CyFalse if the CSI interface is not supported.

#### 5.11.4.23 CyBool\_t CyFx3DevsOtgSupported( void )

Check if the part being used supports the USB OTG and USB 2.0 host functionality.

##### Description

Some devices in the FX3/FX3S product family do not support the USB 2.0 host and OTG functionality. This function is used to check whether the part being used supports the host/OTG functionality.

##### Return Value

CyTrue if USB 2.0 host and OTG is supported. CyFalse if USB 2.0 host and OTG is not supported.

#### 5.11.4.24 CyBool\_t CyFx3DevsRam512Supported( void )

Check if the part being used supports 512 KB of System RAM.

##### Description

Some devices in the FX3/FX3S product family support only 256 KB of System RAM, and the firmware needs to be constrained to work with the available memory. This function is used to check whether the part being used supports 512 KB of System RAM.

##### Return Value

CyTrue if the part supports 512 KB of RAM. CyFalse if the part supports only 256 KB of RAM.

#### 5.11.4.25 CyBool\_t CyFx3DevsSib0Supported( void )

Check if the part supports the S0 storage port.

##### Description

Some parts in the FX3 family (FX3S, SD3 and SD2) support a storage port through which SD cards, eMMC devices or SDIO cards can be connected. This function checks whether the part in use supports the S0 storage port.

##### Return Value

CyTrue if the S0 storage port is supported. CyFalse if the S0 storage port is not supported.

#### 5.11.4.26 CyBool\_t CyFx3DevsSib1Supported( void )

Check if the part supports the S1 storage port.

##### Description

Some parts in the FX3 family (FX3S, SD3 and SD2) support two storage ports through which SD cards, eMMC devices or SDIO cards can be connected. This function checks whether the part in use supports the S1 (second) storage port.

##### Return Value

CyTrue if the S1 storage port is supported. CyFalse if the S1 storage port is not supported.

#### 5.11.4.27 CyBool\_t CyFx3DevsUsb3Supported( void )

Check if the part being used supports the USB 3.0 peripheral functionality.

##### Description

Some low cost devices in the FX3 family (such as SD2 or FX2G2) do not support the USB 3.0 peripheral functionality, while retaining all of the other capabilities such as the ARM9 core, the high-performance distributed DMA architecture, the GPIF II port and the peripheral support. This function checks whether the active part supports the USB 3.0 function.

##### Return Value

CyTrue if USB 3.0 is supported. CyFalse if USB 3.0 is not supported.

**5.11.4.28 CyBool\_t CyFx3LppIsOn ( void )**

Check if the LPP block is powered ON.

**Description**

This function checks whether the LPP block (common hardware for all serial peripherals) is turned ON.

**Return value**

CyTrue if the LPP block is turned ON.

CyFalse if the LPP block is turned OFF.

**5.11.4.29 void CyFx3PibDIIEnable ( void )**

Enable the DLL in the PIB block.

**Description**

The PIB block on FX3 has an embedded DLL that can be used when FX3 is acting as a slave device with an incoming clock signal. This function enables the DLL in the PIB block.

**Return Value**

None

**5.11.4.30 uint16\_t CyFx3PibGetDIIStatus ( void )**

Get the current status of the PIB DLL.

**Description**

Get the locked/unlocked status of the PIB DLL.

**Return Value**

None

**5.11.4.31 CyBool\_t CyFx3PibIsOn ( void )**

Check whether the PIB block is powered ON.

**Description**

This function checks whether the PIB block on FX3 has been turned ON.

**Return value**

CyTrue if the PIB block has been turned ON.

CyFalse if the PIB block has not been turned ON.

**5.11.4.32 void CyFx3PibPowerOff ( void )**

Power off the PIB and GPIF II blocks on the FX3/FX3S device.

**Description**

A device specific sequence needs to be followed when powering the PIB and GPIF II blocks on the FX3 device off. This function implements the sequence.

Note: The PIB clock should only be disabled after this function returns.

**Return value**

None

**5.11.4.33 void CyFx3PibPowerOn ( void )**

Power on the PIB and GPIF II blocks on the FX3/FX3S device.

**Description**

A device specific sequence needs to be followed when powering the PIB and GPIF II blocks on the FX3 device on. This function implements the sequence.

Note: The PIB clock should have been enabled before calling this function.

**Return value**

None

**5.11.4.34 void CyFx3SetCpuFreq( uint32\_t *cpuFreq* )**

Store the CPU clock frequency for use in BusyWait delay calculation.

**Description**

The CyFx3BusyWait delay function uses CPU instructions to obtain the delay required by the user. The actual delay obtained will depend on the CPU clock frequency. This function stores the CPU clock frequency, so that the BusyWait implementation can be calibrated.

**Return value**

None

**See also**

[CyFx3BusyWait](#)

**Parameters**

<i>cpuFreq</i>	CPU clock frequency in Hertz.
----------------	-------------------------------

**5.11.4.35 void CyFx3SibPowerOff( void )**

Power the storage interface block on the FX3S/SD3 device off.

**Description**

This function powers the storage interface block (SIB) on the FX3S/SD3 device off. The SIB clock(s) can be turned off after calling this function.

**Return Value**

None

**5.11.4.36 void CyFx3SibPowerOn( void )**

Power the storage interface block on the FX3S/SD3 device on.

**Description**

This function powers the storage interface block (SIB) on the FX3S/SD3 device on. A common power-up sequence is used for all available storage ports. Clocks for the available storage ports need to be enabled before calling this function.

**Return Value**

None

**5.11.4.37 void CyFx3Usb2PhySetup( void )**

Configure the USB 2.0 PHY on FX3.

**Description**

This function sets up USB 2.0 PHY on the FX3 for operation. The settings made by this API are required for passing USB 2.0 electrical tests.

**Return Value**

None

**5.11.4.38 void CyFx3Usb3LnkRelaxHpTimeout( void )**

Relax the USB 3.0 HP ACK Timeout period.

**Description**

In some cases, FX3 does not receive the ACK for a USB 3.0 link header in time, and gets into the SS.Inactive state. This API relaxes the timeout for the ACK so that this kind of error is prevented.

**Return Value**

None

**5.11.4.39 void CyFx3Usb3LnkSetup( void )**

Configure the USB 3.0 LINK on FX3.

**Description**

This function sets up the USB 3.0 LINK on the FX3 for operation. The settings made by this API are required for passing USB 3.0 link layer tests.

**Return Value**

None

**5.11.4.40 void CyFx3Usb3SendTP( uint32\_t \* tpData )**

Send a USB 3.0 Transaction Packet.

**Description**

This function sends a USB 3.0 Transaction Packet (TP) with user specified data to the USB host.

**Return Value**

None

**Parameters**

<i>tpData</i>	Pointer to the TP data (3 dwords) to be sent.
---------------	---

**5.11.4.41 void CyFx3UsbDmaPrefetchEnable( CyBool\_t streamEnable )**

Enable data prefetch from the DMA sockets on the USB egress (IN) data path.

**Description**

This function enables prefetching of data from the DMA sockets on the USB egress (or IN endpoint) data path, so as to obtain better transfer performance. This setting is always enabled during USB 2.0 operation, but should be used with caution on a USB 3.0 connection.

**Return Value**

None

**Parameters**

<i>streamEnable</i>	Enable stream mode of data prefetch. Can cause endpoint freeze if used with endpoints that send a lot of short packets.
---------------------	---

#### 5.11.4.42 CyBool\_t CyFx3UsbIsOn ( void )

Check whether the USB block is powered ON.

##### Description

Function to check whether the USB block on FX3 has been turned on.

##### Return value

CyTrue if the USB block is powered ON.

CyFalse if the USB block is not powered ON.

#### 5.11.4.43 void CyFx3UsbPowerOn ( void )

Power on the USB block on the FX3/FX3S device.

##### Description

A device specific sequence needs to be followed when powering the USB block on the FX3 device on. This function implements the sequence.

##### Return value

None

#### 5.11.4.44 void CyFx3UsbWritePhyReg ( uint16\_t phyAddr, uint16\_t phyVal )

Write to a USB 3.0 PHY Control Register.

##### Description

The USB 3.0 PHY on the FX3 device has a few control registers that may need to be updated at various stages of device operation. This function writes a user specified value into one of the PHY control registers.

##### Return Value

None

##### Parameters

<i>phyAddr</i>	Address of the PHY register.
<i>phyVal</i>	Value to be written.

## 5.12 firmware/u3p\_firmware/inc/cyfxapidesc.h File Reference

Provides a brief description of the FX3 APIs.

### 5.12.1 Detailed Description

Provides a brief description of the FX3 APIs.

## 5.13 firmware/u3p\_firmware/inc/cyfxversion.h File Reference

Version information for the FX3 API library.

##### Macros

- #define CYFX\_VERSION\_MAJOR (1)

- `#define CYFX_VERSION_MINOR (3)`  
*Minor number of the release version.*
- `#define CYFX_VERSION_PATCH (4)`  
*Patch number for this release.*
- `#define CYFX_VERSION_BUILD (31)`  
*Build number.*

### 5.13.1 Detailed Description

Version information for the FX3 API library.

#### Description

The version information is composed of four values.

#### Note

The version information is compiled into the library and is provided here for reference. These values should not be changed and can only be used to conditionally enable code.

## 5.14 firmware/u3p\_firmware/inc/cyu3cardmgr.h File Reference

This file defines the data types and APIs provided by the low level SD/MMC/SDIO driver for FX3S.

```
#include <cyu3os.h>
#include <cyu3types.h>
#include <cyu3sib.h>
#include <cyu3cardmgr_fx3s.h>
#include <cyu3externcstart.h>
#include <cyu3externcend.h>
```

### Data Structures

- struct `CyU3PCardCtx`  
*Structure that holds properties of an SD/MMC peripheral.*

### Macros

- `#define CY_U3P_BOOT_PART_SIGN (0x42575943)`  
*Signature that identifies an FX3S Virtual Boot Partition.*
- `#define CY_U3P_MMC_ECS�_REVISION_LOC (192)`  
*Location of MMC revision field in MMC Extended CSD register.*
- `#define CY_U3P_MMC_ECS�_PARTCFG_LOC (179)`  
*Location of Partition Config field in MMC Extended CSD register.*
- `#define CY_U3P_MMC_ECS�_BOOTSIZE_LOC (226)`  
*Location of Boot Partition Size field in MMC Extended CSD register.*
- `#define CY_U3P_CLOCK_DIVIDER_400 (0x3FF)`  
*Clock divider value to get 400 KHz interface clock frequency.*
- `#define CY_U3P_CLOCK_DIVIDER_20M (0x14)`  
*Clock divider value to get 20 MHz interface clock frequency.*
- `#define CY_U3P_CLOCK_DIVIDER_26M (0x0F)`  
*Clock divider value to get 26 MHz interface clock frequency.*

- `#define CY_U3P_CLOCK_DIVIDER_46M (0x08)`  
*Clock divider value to get 46 MHz interface clock frequency.*
- `#define CY_U3P_CLOCK_DIVIDER_52M (0x07)`  
*Clock divider value to get 52 MHz interface clock frequency.*
- `#define CY_U3P_CLOCK_DIVIDER_84M (0x04)`  
*Clock divider value to get 84 MHz interface clock frequency.*
- `#define CY_U3P_CLOCK_DIVIDER_104M (0x03)`  
*Clock divider value to get 104 MHz interface clock frequency.*
- `#define CY_U3P_SIB_DEVICE_POSTRESET_DELAY (5)`  
*Post reset (CMD0) delay allowed for storage devices.*
- `#define CY_U3P_SIB_DEVICE_READY_TIMEOUT (0x3FF)`  
*Timeout (in ms) for the wait for device readiness.*
- `#define CY_U3P_SIB_POLLING_DELAY (5)`  
*Storage driver polling delay in us.*
- `#define CY_U3P_SIB_SENDCMD_DELAY (10)`  
*Storage driver command send delay.*
- `#define CY_U3P_SIB_HOTPLUG_DELAY (100)`  
*Delay required from hotplug event to handler start (in ms).*
- `#define CY_U3P_SIB_BLOCK_SIZE (0x0200)`  
*Default data block size is 512 bytes.*
- `#define CY_U3P_SIB_SD_CMD11_CLKSTOP_DURATION (10)`  
*Clock stop duration while switching operating voltage.*
- `#define CY_U3P_SIB_SD_CMD11_CLKACT_DURATION (2)`  
*Delay from SD clock start that is required for SD card to drive the IOs high.*
- `#define CY_U3P_SIB_WRITE_DATA_TIMEOUT (5000)`  
*Write data timeout value in clock ticks.*
- `#define CY_U3P_SIB_IDLE_DATA_TIMEOUT (2000)`  
*Idle Time out value.*
- `#define CY_U3P_SIB_SECTORCNT_TO_BYTERCNT(n) ((n) << 9)`  
*Macro to convert a sector address/count value to a byte address/count.*
- `#define CY_U3P_SIB_BYTERCNT_TO_SECTORCNT(n) ((n) >> 9)`  
*Macro to convert a byte address/count value to a sector address/count.*
- `#define CY_U3P_SD_SW_QUERY_FUNCTIONS (0x00FFFFFF1)`
- `#define CY_U3P_SD_SW_HIGHSP_PARAM (0x80FFFFFF1)`
- `#define CY_U3P_SD_SW_UHS1_PARAM (0x80FFFFFF2)`
- `#define CY_U3P_MMC_SW_PARTCFG_USER_PARAM (0x03B30000)`
- `#define CY_U3P_MMC_SW_PARTCFG_BOOT1_PARAM (0x03B30100)`
- `#define CY_U3P_MMC_SW_PARTCFG_BOOT2_PARAM (0x03B30200)`
- `#define CY_U3P_SIB_VBP_PARTSIZE_LOCATION (40)`  
*Offset of partition size byte in the Virtual Partition Header.*
- `#define CY_U3P_SIB_VBP_PARTTYPE_LOCATION (47)`  
*Offset of partition type byte in the Virtual Partition Header.*
- `#define CY_U3P_SIB_VBP_CHECKSUM_LOCATION (68)`  
*Offset of checksum byte in the Virtual Partition Header.*
- `#define CyU3PSibSetNumBlocks(portId, numBlks) (SIB->sdmmc[(portId)].block_count = (numBlks))`  
*Macro to set the number of data blocks for transfer on one SD port.*
- `#define CyU3PSibSetActiveSocket(portId, sockNum)`  
*Macro that sets the active socket for one SD port.*
- `#define CyU3PSibResetSibCtrlr(portId)`  
*Macro to reset the SIB Controller and then restore the operating mode.*
- `#define CyU3PSibClearIntr(portId)`

*Macro to clear all the interrupts on the given port.*

- #define [CyU3PSibConvertAddr](#)(portId, blkAddr)

*Macro to convert the block address to byte address if the target device is byte addressed.*

- #define [CyU3PSdioNumberOfFunction](#)(portId) (SIB->sdmmc[(portId)].resp\_reg0>>28)&0x7

*Get the number of functions supported by an SDIO device.*

- #define [CyU3PSdioMemoryPresent](#)(portId) (SIB->sdmmc[(portId)].resp\_reg0>>27)&0x1

*Macro to check whether an SDIO device define [Cy9g 0 0 f . 48 0 Td 93 -12.969 Td \(\) TJcro8\(b9OCRRG gis1.002ne\) TJ0 0 1 rg 0 0](#)*

- De-initialize a storage peripheral connected to FX3S.*
- void [CyU3PCardMgrSetClockFreq](#) (uint8\_t portId, uint16\_t clkDivider, [CyBool\\_t](#) halfDiv)  
*Select the interface clock frequency for a storage port.*
  - [CyU3PReturnStatus\\_t CyU3PCardMgrSetupWrite](#) (uint8\_t portId, uint8\_t unit, uint8\_t socket, uint16\_t numWriteBlks, uint32\_t blkAddr)  
*Initiate a data write request to an SD card or eMMC storage device.*
  - [CyU3PReturnStatus\\_t CyU3PCardMgrSetupRead](#) (uint8\_t portId, uint8\_t unit, uint8\_t socket, uint16\_t numReadBlks, uint32\_t blkAddr)  
*Initiate a data read request to an SD card or eMMC storage device.*
  - [CyU3PReturnStatus\\_t CyU3PCardMgrContinueReadWrite](#) (uint8\_t isRead, uint8\_t portId, uint8\_t socket, uint16\_t numBlks)  
*Continue reading/writing data from the address where the previous operation ended.*
  - [CyU3PReturnStatus\\_t CyU3PCardMgrStopTransfer](#) (uint8\_t portId)  
*Stop an ongoing SD/MMC data transfer.*
  - void [CyU3PCardMgrGetCSD](#) (uint8\_t portId, uint8\_t \*csd\_buffer)  
*Get the CSD register data for an SD/MMC peripheral.*
  - [CyU3PReturnStatus\\_t CyU3PCardMgrReadExtCsd](#) (uint8\_t portId, uint8\_t \*buffer\_p)  
*Get the Extended CSD register content for an SD/MMC peripheral.*
  - void [CyU3PCardMgrSendCmd](#) (uint8\_t portId, uint8\_t cmd, uint8\_t respLen, uint32\_t cmdArg, uint32\_t flags)  
*Send a command to an SD/MMC/SDIO peripheral.*
  - [CyU3PReturnStatus\\_t CyU3PCardMgrWaitForInterrupt](#) (uint8\_t portId, uint32\_t intr, uint32\_t timeout)  
*Function to wait for a SIB related interrupt.*
  - [CyU3PReturnStatus\\_t CyU3PCardMgrCheckStatus](#) (uint8\_t portId)  
*Check the status of SD/MMC peripheral and wait until it is in the transfer state.*
  - [CyU3PReturnStatus\\_t CyU3PCardMgrCompleteSDInit](#) (uint8\_t portId)  
*Complete the initialization of an SD card after it has been unlocked.*

## Variables

- [CyU3PCardCtx\\_t glCardCtx \[CY\\_U3P\\_SIB\\_NUM\\_PORTS\]](#)  
*Global variable that holds peripheral properties.*

### 5.14.1 Detailed Description

This file defines the data types and APIs provided by the low level SD/MMC/SDIO driver for FX3S.

#### Description

The EZ-USB FX3S device is an extension to the FX3 device, which has the capability to talk to SD/MMC/SDIO peripherals.

The SD/MMC/SDIO low level driver provides functions to send commands to these peripherals and perform data transfers. These functions are intended to be used internally by the FX3S storage API layer, and not expected to be called directly from user applications.

### 5.14.2 Macro Definition Documentation

#### 5.14.2.1 #define CY\_U3P\_MMU\_SW\_PARTCFG\_BOOT1\_PARAM (0x03B30100)

Switch parameter to select Boot1 partition.

#### 5.14.2.2 #define CY\_U3P\_MMU\_SW\_PARTCFG\_BOOT2\_PARAM (0x03B30200)

Switch parameter to select Boot2 partition.

5.14.2.3 #define CY\_U3P\_MMC\_SW\_PARTCFG\_USER\_PARAM (0x03B30000)

Switch parameter to select User Data area.

5.14.2.4 #define CY\_U3P\_SD\_SW\_HIGHSP\_PARAM (0x80FFFFF1)

Switch parameter to enable High Speed mode.

5.14.2.5 #define CY\_U3P\_SD\_SW\_QUERY\_FUNCTIONS (0x00FFFFFF1)

Switch parameter to query device functions.

5.14.2.6 #define CY\_U3P\_SD\_SW\_UHS1\_PARAM (0x80FFFFFF2)

Switch parameter to select UHS-1 mode.

5.14.2.7 #define CyU3PSibClearIntr( *portId* )

**Value:**

```
{ \
    SIB->sdmmc[(portId)].intr = ~(CY_U3P_SIB_SDMMC_INTR_DEFAULT); \
}
```

Macro to clear all the interrupts on the given port.

5.14.2.8 #define CyU3PSibConvertAddr( *portId*, *blkAddr* )

**Value:**

```
{ \
    if (glCardCtxt[(portId)].highCapacity == CyFalse) \
    { \
        (blkAddr) <= 9; \
    } \
}
```

Macro to convert the block address to byte address if the target device is byte addressed.

5.14.2.9 #define CyU3PSibResetSibCtrlr( *portId* )

**Value:**

```
{\
    uint32_t tmp321 = SIB->sdmmc[(portId)].mode_cfg; \
    SIB->sdmmc[(portId)].cs |= CY_U3P_SIB_RSTCONT; \
    while (SIB->sdmmc[(portId)].cs & CY_U3P_SIB_RSTCONT); \
    SIB->sdmmc[(portId)].mode_cfg = tmp321; \
}
```

Macro to reset the SIB Controller and then restore the operating mode.

5.14.2.10 #define CyU3PSibSetActiveSocket( *portId*, *sockNum* )

**Value:**

```
{ \
    SIB->sdmmc[ (portId) ].cs = ((SIB->sdmmc[ (portId) ].cs & ~CY_U3P_SIB_SOCKET_MASK) | \
                                    (CyU3PDmaGetSckNum( (sockNum) ) << CY_U3P_SIB_SOCKET_POS)); \
}
```

Macro that sets the active socket for one SD port.

### 5.14.3 Typedef Documentation

5.14.3.1 **typedef struct CyU3PCardCtxt CyU3PCardCtxt\_t**

Structure that holds properties of an SD/MMC peripheral.

**Description**

This structure holds status and state information about an SD/MMC peripheral connected to the FX3S device.

### 5.14.4 Enumeration Type Documentation

5.14.4.1 **enum CyU3PCardBusWidth\_t**

Enumeration of storage peripheral bus widths.

**Enumerator**

**CY\_U3P\_CARD\_BUS\_WIDTH\_1\_BIT** Single bit mode.

**CY\_U3P\_CARD\_BUS\_WIDTH\_4\_BIT** 4 bit wide mode.

**CY\_U3P\_CARD\_BUS\_WIDTH\_8\_BIT** 8 bit wide mode (eMMC only).

5.14.4.2 **enum CyU3PCardOpMode\_t**

Enumeration of card operating modes.

**Enumerator**

**CY\_U3P\_SIB\_SD\_CARD\_ID\_400** SD 400 KHz

**CY\_U3P\_SIB\_SDSC\_25** SDSC

**CY\_U3P\_SIB\_SDHC\_DS\_25** SDHC Default Speed Mode 0 - 25MHz

**CY\_U3P\_SIB\_SDHC\_HS\_50** SDHC 25MB/sec 0 - 50MHz

**CY\_U3P\_SIB\_UHS\_I\_DS** Default Speed up to 25MHz 3.3V signaling

**CY\_U3P\_SIB\_UHS\_I\_HS** High Speed up to 50MHz 3.3V signaling

**CY\_U3P\_SIB\_UHS\_I\_SDR12** SDR up to 25MHz 1.8V signaling

**CY\_U3P\_SIB\_UHS\_I\_SDR25** SDR up to 50MHz 1.8V signaling

**CY\_U3P\_SIB\_UHS\_I\_SDR50** SDR up to 100MHz 1.8V signaling

**CY\_U3P\_SIB\_UHS\_I\_SDR104** SDR up to 208MHz 1.8V signaling

**CY\_U3P\_SIB\_UHS\_I\_DDR50** DDR up to 50MHz 1.8V signaling

#### 5.14.4.3 enum CyU3PSDCardVer\_t

Enumeration of SD Card Versions.

Enumerator

- CY\_U3P\_SD\_CARD\_VER\_1\_X** SD 1.x cards: Byte addressed.
- CY\_U3P\_SD\_CARD\_VER\_2\_0** SD 2.0 cards: Sector addressed, high capacity.
- CY\_U3P\_SD\_CARD\_VER\_3\_0** SD 3.0 cards: Extended capacity, UHS-1 cards.

#### 5.14.4.4 enum CyU3PSdMmcStates\_t

SD/MMC peripheral states.

##### Description

List of SD/MMC peripheral states, as defined by the specifications.

Enumerator

- CY\_U3P\_SD\_MMCI\_DLE** Idle state.
- CY\_U3P\_SD\_MMCI\_READY** Ready state.
- CY\_U3P\_SD\_MMCI\_IDENTIFICATION** Identification state.
- CY\_U3P\_SD\_MMCI\_STANDBY** Standby state.
- CY\_U3P\_SD\_MMCI\_TRANSFER** Transfer state.
- CY\_U3P\_SD\_MMCI\_DATA** Data state.
- CY\_U3P\_SD\_MMCI\_RECEIVE** Receive state.
- CY\_U3P\_SD\_MMCI\_PROGRAMMING** Programming state.
- CY\_U3P\_SD\_MMCI\_DISCONNECT** Disconnect state.
- CY\_U3P\_MMCI\_BUS\_TEST** Bus test state.

#### 5.14.4.5 enum CyU3PSibSDRegs\_t

Enumeration of SD Card Registers.

Enumerator

- CY\_U3P\_SIB\_SD\_REG\_OCR** OCR Register
- CY\_U3P\_SIB\_SD\_REG\_CID** CID Register
- CY\_U3P\_SIB\_SD\_REG\_CSD** CSD Register

### 5.14.5 Function Documentation

#### 5.14.5.1 CyU3PReturnStatus\_t CyU3PCardMgrCheckStatus ( uint8\_t portId )

Check the status of SD/MMC peripheral and wait until it is in the transfer state.

##### Description

This function is used to read the status (CMD13) of the SD/MMC peripheral and wait until it is in the transfer state. This is used to ensure that the peripheral is ready to receive the next data transfer command.

##### Return value

- CY\_U3P\_SUCCESS if the card moved to the transfer state.
- CY\_U3P\_ERROR\_TIMEOUT if the card did not move to the transfer state.

**Parameters**

<i>port</i> ↵ <i>Id</i>	Port on which the peripheral state is to be checked.
----------------------------	--

**5.14.5.2 CyU3PReturnStatus\_t CyU3PCardMgrCompleteSDInit ( uint8\_t portId )**

Complete the initialization of an SD card after it has been unlocked.

**Description**

If an SD card is password locked, it cannot be fully initialized during the CyU3PCardMgrInit call. This function is used to complete the Initialization of the SD card after it has been unlocked.

**Return value**

CY\_U3P\_SUCCESS if the initialization is successful.

CY\_U3P\_ERROR\_TIMEOUT if the initialization times out.

**Parameters**

<i>port</i> ↵ <i>Id</i>	Port on which the card is connected.
----------------------------	--------------------------------------

**5.14.5.3 CyU3PReturnStatus\_t CyU3PCardMgrContinueReadWrite ( uint8\_t isRead, uint8\_t portId, uint8\_t socket, uint16\_t numBlks )**

Continue reading/writing data from the address where the previous operation ended.

**Description**

Data transfer performance from/to SD/MMC devices can be improved by combining sequential operations into a single read/write command. This function sets up the FX3S storage port to continue a previously paused storage read/write operation. This is generally used only for write transfers.

**Return value**

CY\_U3P\_SUCCESS if the transfer setup is successful.

CY\_U3P\_ERROR\_TIMEOUT if the transfer could not be resumed.

**Parameters**

<i>isRead</i>	CyTrue = Read operation, CyFalse = Write operation.
<i>portId</i>	Port to do the operation on.
<i>socket</i>	Socket to use for data transfer.
<i>numBlks</i>	Number of additional blocks (sectors) to transfer.

**5.14.5.4 void CyU3PCardMgrDelInit ( uint8\_t portId )**

De-initialize a storage peripheral connected to FX3S.

**Description**

De-initialize a storage peripheral connected to the specified storage port.

**Return value**

None

**Parameters**

<i>portId</i>	Port on which the peripheral to be de-initialized is connected.
---------------	---

**5.14.5.5 void CyU3PCardMgrGetCSD ( uint8\_t portId, uint8\_t \* csd\_buffer )**

Get the CSD register data for an SD/MMC peripheral.

**Description**

This function returns the contents of the CSD register of the SD/MMC peripheral connected on the specified port. As the CSD register cannot be read at all times, this returns values that are buffered during the peripheral initialization.

**Return value**

None

**Parameters**

<i>portId</i>	Port number.
<i>csd_buffer</i>	Return buffer to be filled with CSD data.

**5.14.5.6 CyU3PReturnStatus\_t CyU3PCardMgrInit ( uint8\_t portId )**

Initialize a SD/MMC/SDIO storage device attached to FX3S.

**Description**

Initialize a SD/MMC/SDIO storage device attached to the specified FX3S storage port. This will return immediately if the peripheral connected to the port has already been initialized.

**Return value**

CY\_U3P\_SUCCESS if the initialization is successful.

CY\_U3P\_ERROR\_TIMEOUT if the peripheral fails to respond.

**Parameters**

<i>portId</i>	Port on which the peripheral is connected.
---------------	--

**5.14.5.7 CyU3PReturnStatus\_t CyU3PCardMgrReadExtCsd ( uint8\_t portId, uint8\_t \* buffer\_p )**

Get the Extended CSD register content for an SD/MMC peripheral.

**Description**

This function reads and returns the content of the Extended CSD register on the SD card or MMC device connected to the specified storage device.

**Return value**

CY\_U3P\_SUCCESS if the read is successful.

CY\_U3P\_ERROR\_TIMEOUT if the read operation times out.

**Parameters**

<i>portId</i>	Port number.
<i>buffer_p</i>	Buffer to read the register data into. Should be able to hold 512 bytes.

---

5.14.5.8 void CyU3PCardMgrSendCmd ( uint8\_t *portId*, uint8\_t *cmd*, uint8\_t *respLen*, uint32\_t *cmdArg*, uint32\_t *flags* )

Send a command to an SD/MMC/SDIO peripheral.

**Description**

This function sends the specified command to the specified SD/MMC/SDIO peripheral and initiates the device to receive the response (if any).

**Return value**

CY\_U3P\_SUCCESS if the read is successful.

CY\_U3P\_ERROR\_TIMEOUT if the read operation times out.

**Parameters**

<i>portId</i>	Port number.
<i>cmd</i>	Command opcode.
<i>respLen</i>	Expected response length in bits.
<i>cmdArg</i>	Command argument value.
<i>flags</i>	Flags that indicate whether any data transfer is expected.

5.14.5.9 void CyU3PCardMgrSetClockFreq ( uint8\_t *portId*, uint16\_t *clkDivider*, CyBool\_t *halfDiv* )

Select the interface clock frequency for a storage port.

**Description**

This function updates the interface clock frequency for the specified FX3S storage port. The clock is stopped while the frequency is changed and then re-enabled.

**Return value**

None

**Parameters**

<i>portId</i>	Storage port to be updated.
<i>clkDivider</i>	Clock divider to be used. The base clock frequency is always 384 MHz.
<i>halfDiv</i>	Whether a 0.5 factor has to be added to the selected divider.

5.14.5.10 CyU3PReturnStatus\_t CyU3PCardMgrSetupRead ( uint8\_t *portId*, uint8\_t *unit*, uint8\_t *socket*, uint16\_t *numReadBlks*, uint32\_t *blkAddr* )

Initiate a data read request to an SD card or eMMC storage device.

**Description**

This function sends the read command (CMD17 or CMD18) to the SD/eMMC storage peripheral and prepares the FX3S block for the corresponding data transfer. The transfer completion will be notified through interrupt when done.

**Return value**

CY\_U3P\_SUCCESS if the read setup is successful.

CY\_U3P\_ERROR\_INVALID\_ADDR if the read parameter is invalid.

**Parameters**

<i>portId</i>	Port on which read operation is to be performed.
<i>unit</i>	Partition to read data from.
<i>socket</i>	DMA socket to be used for read.
<i>numReadBlks</i>	Number of blocks of data to read.

**Parameters**

<i>blkAddr</i>	Sector address to read data from.
----------------	-----------------------------------

5.14.5.11 **CyU3PReturnStatus\_t CyU3PCardMgrSetupWrite ( uint8\_t *portId*, uint8\_t *unit*, uint8\_t *socket*, uint16\_t *numWriteBlks*, uint32\_t *blkAddr* )**

Initiate a data write request to an SD card or eMMC storage device.

**Description**

This function sends the write command (CMD24 or CMD25) to the SD/eMMC storage peripheral and prepares the FX3S block for the corresponding data transfer. The transfer completion will be notified through interrupt when done.

**Return value**

CY\_U3P\_SUCCESS if the write setup is successful.

CY\_U3P\_ERROR\_INVALID\_ADDR if the write parameter is invalid.

**Parameters**

<i>portId</i>	Port on which to setup the write operation.
<i>unit</i>	Storage partition to write to. Used for address calculation.
<i>socket</i>	DMA socket to be used for the write operation.
<i>numWriteBlks</i>	Number of block of data to be written.
<i>blkAddr</i>	Sector address to write to (within the selected partition).

5.14.5.12 **CyU3PReturnStatus\_t CyU3PCardMgrStopTransfer ( uint8\_t *portId* )**

Stop an ongoing SD/MMC data transfer.

**Description**

This function is used to send a stop transmission command to the card to stop an ongoing data transfer.

**Return value**

CY\_U3P\_SUCCESS if the stop command is successful.

CY\_U3P\_ERROR\_TIMEOUT if the card failed to respond.

**Parameters**

<i>portId</i>	Port on which transfer is to be stopped.
---------------	--

5.14.5.13 **CyU3PReturnStatus\_t CyU3PCardMgrWaitForInterrupt ( uint8\_t *portId*, uint32\_t *intr*, uint32\_t *timeout* )**

Function to wait for a SIB related interrupt.

**Description**

This function is used to wait for a SIB related completion interrupt, or until an error or timeout is detected.

**Return value**

CY\_U3P\_SUCCESS if the specified interrupt is received.

CY\_U3P\_ERROR\_CRC if a CRC error is detected.

CY\_U3P\_ERROR\_TIMEOUT if the interrupt was not received.

### Parameters

<i>portId</i>	Port number
<i>intr</i>	Type of interrupt to wait for.
<i>timeout</i>	Timeout to be applied in 5 us units.

## 5.15 firmware/u3p\_firmware/inc/cyu3cardmgr\_fx3s.h File Reference

This file contains the provides the low level SD/MMC/SDIO driver related macros and definitions.

```
#include <cyu3os.h>
#include <cyu3types.h>
#include <cyu3sib.h>
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

### Macros

- #define CY\_U3P\_SD\_REG\_OCR\_LEN (4)  
*Length of the OCR register in bytes.*
- #define CY\_U3P\_SD\_REG\_CID\_LEN (16)  
*Length of the CID register in bytes.*
- #define CY\_U3P\_SD\_REG\_CSD\_LEN (16)  
*Length of the CSD register in bytes.*
- #define CY\_U3P\_SDIO\_SUSPEND (0x0)  
*SDIO operation code for function suspend.*
- #define CY\_U3P\_SDIO\_RESUME (0x1)  
*SDIO operation code for function resume.*
- #define CY\_U3P\_SDIO\_READ (0x0)  
*Operation code for read from SDIO card.*
- #define CY\_U3P\_SDIO\_WRITE (0x1)  
*Operation code for write to SDIO card.*
- #define CY\_U3P\_SDIO\_RW\_BYTE\_MODE (0x0)  
*Flag that indicates that the SDIO transfer requested is a byte-mode request.*
- #define CY\_U3P\_SDIO\_RW\_BLOCK\_MODE (0x1)  
*Flag that indicates that the SDIO transfer request is a block-mode request.*
- #define CY\_U3P\_SDIO\_RW\_ADDR\_FIXED (0x0)  
*Flag that indicates that the SDIO transfer should be performed with a fixed address.*
- #define CY\_U3P\_SDIO\_RW\_ADDR\_AUTO\_INCREMENT (0x1)  
*Flag that indicates that the address for the SDIO transfer should be auto incremented.*
- #define CY\_U3P\_SDIO\_REG\_CCCR\_REVISION (0x00)
- #define CY\_U3P\_SDIO\_REG\_SD\_SPEC\_REVISION (0x01)
- #define CY\_U3P\_SDIO\_REG\_IO\_ENABLE (0x02)
- #define CY\_U3P\_SDIO\_REG\_IO\_READY (0x03)
- #define CY\_U3P\_SDIO\_REG\_IO\_INTR\_ENABLE (0x04)
- #define CY\_U3P\_SDIO\_REG\_IO\_INTR\_PENDING (0x05)
- #define CY\_U3P\_SDIO\_REG\_IO\_ABORT (0x06)
- #define CY\_U3P\_SDIO\_REG\_BUS\_INTERFACE\_CONTROL (0x07)
- #define CY\_U3P\_SDIO\_REG\_CARD\_CAPABILITY (0x08)
- #define CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D0 (0x09)

- #define CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D1 (0x0A)
- #define CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D2 (0x0B)
- #define CY\_U3P\_SDIO\_REG\_BUS\_SUSPEND (0x0C)
- #define CY\_U3P\_SDIO\_REG\_FUNCTION\_SELECT (0x0D)
- #define CY\_U3P\_SDIO\_REG\_EXEC\_FLAGS (0x0E)
- #define CY\_U3P\_SDIO\_REG\_READY\_FLAGS (0x0F)
- #define CY\_U3P\_SDIO\_REG\_FUNCTION\_BLOCKSIZE\_D0 (0x10)
- #define CY\_U3P\_SDIO\_REG\_FUNCTION\_BLOCKSIZE\_D1 (0x11)
- #define CY\_U3P\_SDIO\_REG\_POWER\_CONTROL (0x12)
- #define CY\_U3P\_SDIO\_REG\_CCCR\_HIGH\_SPEED (0x13)
- #define CY\_U3P\_SDIO\_REG\_UHS\_I\_SUPPORT (0x14)
- #define CY\_U3P\_SDIO\_REG\_DRIVER\_STRENGTH (0x15)
- #define CY\_U3P\_SDIO\_REG\_INTERRUPT\_EXTENSION (0x16)
- #define CY\_U3P\_SDIO\_REG\_FBR\_INTERFACE\_CODE (0x00)
- #define CY\_U3P\_SDIO\_REG\_FBR\_EXT\_INTERFACE\_CODE (0x01)
- #define CY\_U3P\_SDIO\_REG\_FBR\_POWER\_SELECT (0x02)
- #define CY\_U3P\_SDIO\_REG\_FBR\_CIS\_PTR\_DO CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D0
- #define CY\_U3P\_SDIO\_REG\_FBR\_CIS\_PTR\_D1 CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D1
- #define CY\_U3P\_SDIO\_REG\_FBR\_CIS\_PTR\_D2 CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D2
- #define CY\_U3P\_SDIO\_REG\_FBR\_CSA\_PTR\_DO (0x0C)
- #define CY\_U3P\_SDIO\_REG\_FBR\_CSA\_PTR\_D1 (0x0D)
- #define CY\_U3P\_SDIO\_REG\_FBR\_CSA\_PTR\_D2 (0x0E)
- #define CY\_U3P\_SDIO\_REG\_FBR\_DATA\_ACCESS\_WINDOW (0x0F)
- #define CY\_U3P\_SDIO\_REG\_FBR\_IO\_BLOCKSIZE\_D0 CY\_U3P\_SDIO\_REG\_FUNCTION\_BLOCKSIZE\_D0
- #define CY\_U3P\_SDIO\_REG\_FBR\_IO\_BLOCKSIZE\_D1 CY\_U3P\_SDIO\_REG\_FUNCTION\_BLOCKSIZE\_D1
- #define CY\_U3P\_SDIO\_LOW\_SPEED (0x00)
- #define CY\_U3P\_SDIO\_FULL\_SPEED (0x01)
- #define CY\_U3P\_SDIO\_HIGH\_SPEED (0x02)
- #define CY\_U3P\_SDIO\_SSDR12\_SPEED (0x04)
- #define CY\_U3P\_SDIO\_SSDR25\_SPEED (0x10)
- #define CY\_U3P\_SDIO\_SSDR50\_SPEED (0x20)
- #define CY\_U3P\_SDIO\_SSDR104\_SPEED (0x40)
- #define CY\_U3P\_SDIO\_SDDR50\_SPEED (0x80)
- #define CY\_U3P\_SDIO\_CARD\_CAPABILITY\_SDC (0x01)
- #define CY\_U3P\_SDIO\_CARD\_CAPABILITY\_SMB (0x02)
- #define CY\_U3P\_SDIO\_CARD\_CAPABILITY\_SRW (0x04)
- #define CY\_U3P\_SDIO\_CARD\_CAPABILITY\_SBS (0x08)
- #define CY\_U3P\_SDIO\_CARD\_CAPABILITY\_S4MI (0x10)
- #define CY\_U3P\_SDIO\_CARD\_CAPABILITY\_E4MI (0x20)
- #define CY\_U3P\_SDIO\_CARD\_CAPABILITY\_LSC (0x40)
- #define CY\_U3P\_SDIO\_CARD\_CAPABILITY\_4BLS (0x80)
- #define CY\_U3P\_SDIO\_READ\_AFTER\_WRITE (0x01)
- #define CY\_U3P\_SDIO\_SUPPORT\_HIGH\_SPEED (0x01)
- #define CY\_U3P\_SDIO\_ENABLE\_HIGH\_SPEED (0x02)
- #define CY\_U3P\_SDIO\_RESET (0x08)
- #define CY\_U3P\_SDIO\_CIA\_FUNCTION (0x00)
- #define CY\_U3P\_SDIO\_CISTPL\_NULL (0x00)
- #define CY\_U3P\_SDIO\_CISTPL\_END (0xFF)
- #define CY\_U3P\_SDIO\_CISTPL\_MANFID (0x20)
- #define CY\_U3P\_SDIO\_CISTPL\_FUNCE (0x22)
- #define CY\_U3P\_SDIO\_CCCR\_Version\_1\_00 (0x00)
- #define CY\_U3P\_SDIO\_CCCR\_Version\_1\_10 (0x01)
- #define CY\_U3P\_SDIO\_CCCR\_Version\_2\_00 (0x02)

- #define CY\_U3P\_SDIO\_CCCR\_Version\_3\_00 (0x03)
  - #define CY\_U3P\_SDIO\_Version\_1\_00 (0x00)
  - #define CY\_U3P\_SDIO\_Version\_1\_10 (0x01)
  - #define CY\_U3P\_SDIO\_Version\_1\_20 (0x02)
  - #define CY\_U3P\_SDIO\_Version\_2\_00 (0x03)
  - #define CY\_U3P\_SDIO\_Version\_3\_00 (0x04)
  - #define CY\_U3P\_SDIO\_SD\_Version\_1\_00 (0x00)
  - #define CY\_U3P\_SDIO\_SD\_Version\_1\_10 (0x10)
  - #define CY\_U3P\_SDIO\_SD\_Version\_2\_00 (0x20)
  - #define CY\_U3P\_SDIO\_SD\_Version\_3\_00 (0x30)
  - #define CY\_U3P\_SDIO\_UHS\_SDDR50 (0x01)
  - #define CY\_U3P\_SDIO\_UHS\_SDDR104 (0x02)
  - #define CY\_U3P\_SDIO\_UHS\_SDDR50 (0x04)
  - #define CY\_U3P\_SDIO\_SAI (0x01)
  - #define CY\_U3P\_SDIO\_EAI (0x02)
  - #define CY\_U3P\_SDIO\_INTFC\_NONE (0x00)
  - #define CY\_U3P\_SDIO\_INTFC\_UART (0x01)
  - #define CY\_U3P\_SDIO\_INTFC\_BT\_A (0x02)
  - #define CY\_U3P\_SDIO\_INTFC\_BT\_B (0x03)
  - #define CY\_U3P\_SDIO\_INTFC\_GPS (0x04)
  - #define CY\_U3P\_SDIO\_INTFC\_CAM (0x05)
  - #define CY\_U3P\_SDIO\_INTFC\_PHS (0x06)
  - #define CY\_U3P\_SDIO\_INTFC\_WLAN (0x07)
  - #define CY\_U3P\_SDIO\_INTFC\_EMBD\_ATA (0x08)
  - #define CY\_U3P\_SDIO\_INTFC\_BT\_A\_AMP (0x09)
  - #define CY\_U3P\_SDIO\_INT\_MASTER (0x00)
  - #define CY\_U3P\_SDIO\_DISABLE\_INT (0x00)
  - #define CY\_U3P\_SDIO\_ENABLE\_INT (0x01)
  - #define CY\_U3P\_SDIO\_CHECK\_INT\_ENABLE\_REG (0x02)
  - #define CY\_U3P\_SDIO\_ENABLE\_ASYNC\_INT (0x03)
  - #define CyU3PSdioInterruptBit(funcNo) ((0x01) << (funcNo))
- Macro to generate interrupt enable bit mask for a SDIO function.*

- #define CyU3PSdioCheckInterruptEnableStatus(funcNo, ienReg) ((CyU3PSdioInterruptBit(funcNo) & (ienReg)) ? (1) : (0))

*Macro to check if interrupts are enabled or disabled in the returned data of the CyU3PSdioInterruptControl call.*

### 5.15.1 Detailed Description

This file contains the provides the low level SD/MMC/SDIO driver related macros and definitions.

### 5.15.2 Macro Definition Documentation

#### 5.15.2.1 #define CY\_U3P\_SDIO\_CARD\_CAPABILITY\_4BLS (0x80)

Low speed card with 4-bit support.

#### 5.15.2.2 #define CY\_U3P\_SDIO\_CARD\_CAPABILITY\_E4MI (0x20)

Enable Block Gap Interrupt.

#### 5.15.2.3 #define CY\_U3P\_SDIO\_CARD\_CAPABILITY\_LSC (0x40)

Low Speed Card.

5.15.2.4 #define CY\_U3P\_SDIO\_CARD\_CAPABILITY\_S4MI (0x10)

Support Block Gap Interrupt.

5.15.2.5 #define CY\_U3P\_SDIO\_CARD\_CAPABILITY\_SBS (0x08)

Support Bus Control.

5.15.2.6 #define CY\_U3P\_SDIO\_CARD\_CAPABILITY\_SDC (0x01)

Support direct command.

5.15.2.7 #define CY\_U3P\_SDIO\_CARD\_CAPABILITY\_SMB (0x02)

Support Multi-Block transfer.

5.15.2.8 #define CY\_U3P\_SDIO\_CARD\_CAPABILITY\_SRW (0x04)

Support Read Wait.

5.15.2.9 #define CY\_U3P\_SDIO\_CCCR\_Version\_1\_00 (0x00)

CCCR version 1.00

5.15.2.10 #define CY\_U3P\_SDIO\_CCCR\_Version\_1\_10 (0x01)

CCCR version 1.10

5.15.2.11 #define CY\_U3P\_SDIO\_CCCR\_Version\_2\_00 (0x02)

CCCR version 2.00

5.15.2.12 #define CY\_U3P\_SDIO\_CCCR\_Version\_3\_00 (0x03)

CCCR version 3.00

5.15.2.13 #define CY\_U3P\_SDIO\_CHECK\_INT\_ENABLE\_REG (0x02)

Mask to apply on interrupt enable register.

5.15.2.14 #define CY\_U3P\_SDIO\_CIA\_FUNCTION (0x00)

Function number for SDIO CIA.

5.15.2.15 #define CY\_U3P\_SDIO\_CISTPL\_END (0xFF)

Tuple Chain End

5.15.2.16 #define CY\_U3P\_SDIO\_CISTPL\_FUNCE (0x22)

Function Extensions.

5.15.2.17 #define CY\_U3P\_SDIO\_CISTPL\_MANFID (0x20)

Manufacturer ID

5.15.2.18 #define CY\_U3P\_SDIO\_CISTPL\_NULL (0x00)

Starting Tuple Code

5.15.2.19 #define CY\_U3P\_SDIO\_DISABLE\_INT (0x00)

Master interrupt disabled.

5.15.2.20 #define CY\_U3P\_SDIO\_EAI (0x02)

Enable Asynchronous Interrupt.

5.15.2.21 #define CY\_U3P\_SDIO\_ENABLE\_ASYNC\_INT (0x03)

SDIO asynchronous interrupt enable.

5.15.2.22 #define CY\_U3P\_SDIO\_ENABLE\_HIGH\_SPEED (0x02)

Enable high speed flag.

5.15.2.23 #define CY\_U3P\_SDIO\_ENABLE\_INT (0x01)

Master interrupt enabled.

5.15.2.24 #define CY\_U3P\_SDIO\_FULL\_SPEED (0x01)

Full speed.

5.15.2.25 #define CY\_U3P\_SDIO\_HIGH\_SPEED (0x02)

High speed.

5.15.2.26 #define CY\_U3P\_SDIO\_INT\_MASTER (0x00)

Master interrupt flag.

5.15.2.27 #define CY\_U3P\_SDIO\_INTFC\_BT\_A (0x02)

Function supports SDIO Bluetooth Type-A standard interface.

5.15.2.28 #define CY\_U3P\_SDIO\_INTFC\_BT\_A\_AMP (0x09)

Function supports SDIO Bluetooth Type-A AMP standard interface.

5.15.2.29 #define CY\_U3P\_SDIO\_INTFC\_BT\_B (0x03)

Function supports SDIO Bluetooth Type-B standard interface.

5.15.2.30 #define CY\_U3P\_SDIO\_INTFC\_CAM (0x05)

Function supports SDIO Camera standard interface.

5.15.2.31 #define CY\_U3P\_SDIO\_INTFC\_EMBD\_ATA (0x08)

Function supports Embedded SDIO-ATA standard UART.

5.15.2.32 #define CY\_U3P\_SDIO\_INTFC\_GPS (0x04)

Function supports SDIO GPS standard interface.

5.15.2.33 #define CY\_U3P\_SDIO\_INTFC\_NONE (0x00)

No standard Interface supported by the function

5.15.2.34 #define CY\_U3P\_SDIO\_INTFC\_PHS (0x06)

Function supports SDIO PHS standard interface.

5.15.2.35 #define CY\_U3P\_SDIO\_INTFC\_UART (0x01)

Function supports SDIO standard UART.

5.15.2.36 #define CY\_U3P\_SDIO\_INTFC\_WLAN (0x07)

Function supports SDIO WLAN standard interface.

5.15.2.37 #define CY\_U3P\_SDIO\_LOW\_SPEED (0x00)

Low speed.

5.15.2.38 #define CY\_U3P\_SDIO\_READ\_AFTER\_WRITE (0x01)

Read After Write flag for CMD52

5.15.2.39 #define CY\_U3P\_SDIO\_REG\_BUS\_INTERFACE\_CONTROL (0x07)

SDIO bus interface control register address.

5.15.2.40 #define CY\_U3P\_SDIO\_REG\_BUS\_SUSPEND (0x0C)

SDIO bus suspend register address.

5.15.2.41 #define CY\_U3P\_SDIO\_REG\_CARD\_CAPABILITY (0x08)

SDIO card capability register address.

5.15.2.42 #define CY\_U3P\_SDIO\_REG\_CCCR\_HIGH\_SPEED (0x13)

Bus speed select register address.

5.15.2.43 #define CY\_U3P\_SDIO\_REG\_CCCR\_REVISION (0x00)

CCCR revision register address.

5.15.2.44 #define CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D0 (0x09)

CIS pointer: First byte.

5.15.2.45 #define CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D1 (0x0A)

CIS pointer: Second byte.

5.15.2.46 #define CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D2 (0x0B)

CIS pointer: Third byte.

5.15.2.47 #define CY\_U3P\_SDIO\_REG\_DRIVER\_STRENGTH (0x15)

Driver strength register address.

5.15.2.48 #define CY\_U3P\_SDIO\_REG\_EXEC\_FLAGS (0x0E)

Exec flags register address.

5.15.2.49 #define CY\_U3P\_SDIO\_REG\_FBR\_CIS\_PTR\_D1 CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D1

CIS Pointer.

5.15.2.50 #define CY\_U3P\_SDIO\_REG\_FBR\_CIS\_PTR\_D2 CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D2

CIS Pointer.

5.15.2.51 #define CY\_U3P\_SDIO\_REG\_FBR\_CIS\_PTR\_D0 CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D0

CIS Pointer.

5.15.2.52 #define CY\_U3P\_SDIO\_REG\_FBR\_CSA\_PTR\_D1 (0x0D)

CSA pointer.

5.15.2.53 #define CY\_U3P\_SDIO\_REG\_FBR\_CSA\_PTR\_D2 (0x0E)

CSA pointer.

5.15.2.54 #define CY\_U3P\_SDIO\_REG\_FBR\_CSA\_PTR\_D0 (0x0C)

CSA pointer.

5.15.2.55 #define CY\_U3P\_SDIO\_REG\_FBR\_DATA\_ACCESS\_WINDOW (0x0F)

Data access window to CSA.

5.15.2.56 #define CY\_U3P\_SDIO\_REG\_FBR\_EXT\_INTERFACE\_CODE (0x01)

Extended SDIO function interface code.

5.15.2.57 #define CY\_U3P\_SDIO\_REG\_FBR\_INTERFACE\_CODE (0x00)

Standard function interface code.

5.15.2.58 #define CY\_U3P\_SDIO\_REG\_FBR\_IO\_BLOCKSIZE\_D0 CY\_U3P\_SDIO\_REG\_FUNCTION\_BLOCKSIZE\_D0

Block size.

5.15.2.59 #define CY\_U3P\_SDIO\_REG\_FBR\_IO\_BLOCKSIZE\_D1 CY\_U3P\_SDIO\_REG\_FUNCTION\_BLOCKSIZE\_D1

Block size.

5.15.2.60 #define CY\_U3P\_SDIO\_REG\_FBR\_POWER\_SELECT (0x02)

Power selection.

5.15.2.61 #define CY\_U3P\_SDIO\_REG\_FUNCTION\_BLOCKSIZE\_D0 (0x10)

Function 0 block size: First byte.

5.15.2.62 #define CY\_U3P\_SDIO\_REG\_FUNCTION\_BLOCKSIZE\_D1 (0x11)

Function 0 block size: Second byte.

5.15.2.63 #define CY\_U3P\_SDIO\_REG\_FUNCTION\_SELECT (0x0D)

Function select register address.

5.15.2.64 #define CY\_U3P\_SDIO\_REG\_INTERRUPT\_EXTENSION (0x16)

Interrupt extension register address.

5.15.2.65 #define CY\_U3P\_SDIO\_REG\_IO\_ABORT (0x06)

IO abort register address.

5.15.2.66 #define CY\_U3P\_SDIO\_REG\_IO\_ENABLE (0x02)

IO Enable register address.

5.15.2.67 #define CY\_U3P\_SDIO\_REG\_IO\_INTR\_ENABLE (0x04)

IO interrupt enable register address.

5.15.2.68 #define CY\_U3P\_SDIO\_REG\_IO\_INTR\_PENDING (0x05)

IO interrupt pending register address.

5.15.2.69 #define CY\_U3P\_SDIO\_REG\_IO\_READY (0x03)

IO ready register address.

5.15.2.70 #define CY\_U3P\_SDIO\_REG\_POWER\_CONTROL (0x12)

Power control register address.

5.15.2.71 #define CY\_U3P\_SDIO\_REG\_READY\_FLAGS (0x0F)

Ready flags register address.

5.15.2.72 #define CY\_U3P\_SDIO\_REG\_SD\_SPEC\_REVISION (0x01)

SD Spec revision register address.

5.15.2.73 #define CY\_U3P\_SDIO\_REG\_UHS\_I\_SUPPORT (0x14)

UHS-I support register address.

5.15.2.74 #define CY\_U3P\_SDIO\_RESET (0x08)

SDIO reset flag.

5.15.2.75 #define CY\_U3P\_SDIO\_SAI (0x01)

Support Asynchronous Interrupt.

5.15.2.76 #define CY\_U3P\_SDIO\_SD\_Version\_1\_00 (0x00)

SD physical layer version 1.00

5.15.2.77 #define CY\_U3P\_SDIO\_SD\_Version\_1\_10 (0x10)

SD physical layer version 1.10

5.15.2.78 #define CY\_U3P\_SDIO\_SD\_Version\_2\_00 (0x20)

SD physical layer version 2.00

5.15.2.79 #define CY\_U3P\_SDIO\_SD\_Version\_3\_00 (0x30)

SD physical layer version 3.00

5.15.2.80 #define CY\_U3P\_SDIO\_SDDR50\_SPEED (0x80)

DDR50.

5.15.2.81 #define CY\_U3P\_SDIO\_SDDR104\_SPEED (0x40)

SDR104.

5.15.2.82 #define CY\_U3P\_SDIO\_SDDR12\_SPEED (0x04)

SDR12.

5.15.2.83 #define CY\_U3P\_SDIO\_SDDR25\_SPEED (0x10)

SDR25.

5.15.2.84 #define CY\_U3P\_SDIO\_SDDR50\_SPEED (0x20)

SDR50.

5.15.2.85 #define CY\_U3P\_SDIO\_SUPPORT\_HIGH\_SPEED (0x01)

High Speed support flag.

5.15.2.86 #define CY\_U3P\_SDIO\_UHS\_SDDR50 (0x04)

DDR50 support.

5.15.2.87 #define CY\_U3P\_SDIO\_UHS\_SDDR104 (0x02)

SDR104 support.

5.15.2.88 #define CY\_U3P\_SDIO\_UHS\_SSDR50 (0x01)

SDR50 support.

5.15.2.89 #define CY\_U3P\_SDIO\_Version\_1\_00 (0x00)

SDIO version 1.00

5.15.2.90 #define CY\_U3P\_SDIO\_Version\_1\_10 (0x01)

SDIO version 1.10

5.15.2.91 #define CY\_U3P\_SDIO\_Version\_1\_20 (0x02)

SDIO version 1.20

5.15.2.92 #define CY\_U3P\_SDIO\_Version\_2\_00 (0x03)

SDIO version 2.00

5.15.2.93 #define CY\_U3P\_SDIO\_Version\_3\_00 (0x04)

SDIO version 3.00

## 5.16 firmware/u3p\_firmware/inc/cyu3descriptor.h File Reference

DMA descriptor management.

```
#include "cyu3types.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

### Data Structures

- struct [CyU3PDmaDescriptor\\_t](#)

*Descriptor data structure.*

### Macros

- #define [CY\\_U3P\\_DMA\\_DSCR0\\_LOCATION](#) (0x40000000)

*This is the location of the descriptor pool. As the DMA descriptors are fixed in the FX3 memory map by the device architecture, this definition should not be changed.*

- #define [CY\\_U3P\\_DMA\\_DSCR\\_COUNT](#) (512)

*This is the number of descriptors in the pool.*

- #define [CY\\_U3P\\_DMA\\_DSCR\\_SIZE](#) (16)

*This is the size of each descriptor in bytes. This value is defined by the FX3 device architecture.*

## Typedefs

- `typedef struct CyU3PDmaDescriptor_t CyU3PDmaDescriptor_t`  
*Descriptor data structure.*

## Functions

- `void CyU3PDmaDscrListCreate (void)`  
*Create and initialize the free descriptor list.*
- `void CyU3PDmaDscrListDestroy (void)`  
*Destroy the free descriptor list.*
- `CyU3PReturnStatus_t CyU3PDmaDscrGet (uint16_t *index_p)`  
*Get a descriptor number from the free list.*
- `CyU3PReturnStatus_t CyU3PDmaDscrPut (uint16_t index)`  
*Add a descriptor number to the free list.*
- `uint16_t CyU3PDmaDscrGetFreeCount (void)`  
*Get the number of free descriptors available.*
- `CyU3PReturnStatus_t CyU3PDmaDscrSetConfig (uint16_t index, CyU3PDmaDescriptor_t *dscr_p)`  
*Update the DMA descriptor configuration.*
- `CyU3PReturnStatus_t CyU3PDmaDscrGetConfig (uint16_t index, CyU3PDmaDescriptor_t *dscr_p)`  
*Get the descriptor configuration.*
- `CyU3PReturnStatus_t CyU3PDmaDscrChainCreate (uint16_t *dscreIndex_p, uint16_t count, uint16_t bufferSize, uint32_t dscrSync)`  
*Create a circular chain of descriptors.*
- `void CyU3PDmaDscrChainDestroy (uint16_t dscreIndex, uint16_t count, CyBool_t isProdChain, CyBool_t freeBuffer)`  
*Frees the previously created chain of descriptors.*

## Variables

- `CyU3PDmaDescriptor_t * gIDmaDescriptor`

### 5.16.1 Detailed Description

DMA descriptor management.

#### Summary

DMA descriptors are data structures that keep track of the source/destinations and the memory buffers for a data transfer through the FX3 device. The firmware maintains a queue of DMA descriptors that are allocated as required and used by the corresponding firmware modules.

This file defines the data structures and the interfaces for descriptor management.

### 5.16.2 Descriptor Functions

This section documents the utility functions that work with the DMA descriptors. These functions are supposed to be internal functions for the FX3 API library's use and are not expected to be called directly by the user application.

If an application chooses to call these functions, extreme care must be taken to validate the parameters being passed; as these functions do not perform any error checks. Passing in incorrect/invalid parameters can result in unpredictable behavior. In particular, the buffer address in the DMA descriptor needs to be in system memory area. Any other buffer address can cause the entire DMA engine to freeze, requiring a full device reset.

### 5.16.3 Typedef Documentation

#### 5.16.3.1 `typedef struct CyU3PDmaDescriptor_t CyU3PDmaDescriptor_t`

Descriptor data structure.

##### Description

This data structure contains the fields that make up a DMA descriptor on the FX3 device.

Each structure member is composed of multiple fields as shown below. Refer to the `sock_regs.h` header file for the definitions used.

buffer: (`CY_U3P_BUFFER_ADDR_MASK`)

sync: (`CY_U3P_EN_PROD_INT | CY_U3P_EN_PROD_EVENT | CY_U3P_PROD_IP_MASK | CY_U3P_PROD_SCK_MASK | CY_U3P_EN_CONS_INT | CY_U3P_EN_CONS_EVENT | CY_U3P_CONS_IP_MASK | CY_U3P_CONS_SCK_MASK`)

chain: (`CY_U3P_WR_NEXT_DSCR_MASK | CY_U3P_RD_NEXT_DSCR_MASK`)

size: (`CY_U3P_BYTE_COUNT_MASK | CY_U3P_BUFFER_SIZE_MASK | CY_U3P_BUFFER_OCCUPIED | CY_U3P_BUFFER_ERROR | CY_U3P_EOP | CY_U3P_MARKER`)

##### See also

[CyU3PDmaDscrGetConfig](#)

[CyU3PDmaDscrSetConfig](#)

### 5.16.4 Function Documentation

#### 5.16.4.1 `CyU3PReturnStatus_t CyU3PDmaDscrChainCreate( uint16_t * dscrIndex_p, uint16_t count, uint16_t bufferSize, uint32_t dscrSync )`

Create a circular chain of descriptors.

##### Description

The function creates a chain of descriptors for DMA operations. Both the producer and consumer chains are created with the same values. The descriptor sync parameter is updated as provided. A DMA buffer is allocated if the bufferSize variable is non zero. This function is mainly used by the DMA channel APIs, and can also be used to customize descriptors and do advanced DMA operations.

##### Return value

`CY_U3P_SUCCESS` - if the function call is successful.

`CY_U3P_ERROR_MEMORY_ERROR` - if descriptor or DMA buffer allocation failed.

`CY_U3P_ERROR_BAD_ARGUMENT` - if number of descriptors required is zero OR the index is NULL.

##### See also

[CyU3PDmaDscrChainDestroy](#)

##### Parameters

<code>dscrIndex_p</code>	Output parameter that specifies the index of the head descriptor in the chain.
<code>count</code>	Number of descriptors required in the chain.
<code>bufferSize</code>	Size of the buffers to be associated with each descriptor. If set to zero, no buffers will be allocated.
<code>dscrSync</code>	The sync field to be set in all descriptors. Specifies the producer and consumer socket information.

5.16.4.2 void CyU3PDmaDscrChainDestroy ( uint16\_t *dscrIndex*, uint16\_t *count*, CyBool\_t *isProdChain*, CyBool\_t *freeBuffer* )

Frees the previously created chain of descriptors.

#### Description

The function frees the chain of descriptors. This function must be invoked only after suspending or disabling the corresponding DMA sockets. The buffers pointed to by the descriptors can be optionally freed. This function is mainly used by the DMA channel APIs, and can also be used to do advanced DMA programming.

#### Return value

None

#### See also

[CyU3PDmaDscrChainDestroy](#)

#### Parameters

<i>dscrIndex</i>	Index of the head descriptor in the chain.
<i>count</i>	Number of descriptors in the chain.
<i>isProdChain</i>	Specifies whether to traverse the producer chain or the consumer chain to get the next descriptor.
<i>freeBuffer</i>	Whether the DMA buffers associated with the descriptors should be freed.

5.16.4.3 CyU3PReturnStatus\_t CyU3PDmaDscrGet ( uint16\_t \* *index\_p* )

Get a descriptor number from the free list.

#### Description

This function searches the free list for the first available descriptor, and returns the index. This function is used by the DMA channel APIs, and can be used to do advanced DMA programming based on direct socket access.

#### Return value

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - if a NULL pointer is passed.  
 CY\_U3P\_ERROR\_FAILURE - if no free descriptor is found.

#### See also

[CyU3PDmaDscrPut](#)  
[CyU3PDmaDscrGetFreeCount](#)

#### Parameters

<i>index_p</i>	Output parameter which is filled with the free descriptor index.
----------------	--

5.16.4.4 CyU3PReturnStatus\_t CyU3PDmaDscrGetConfig ( uint16\_t *index*, CyU3PDmaDescriptor\_t \* *dscr\_p* )

Get the descriptor configuration.

**Description**

Read the current contents of the specified DMA descriptor into the output descriptor data structure.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if an invalid index or a NULL pointer is passed.

**See also**

[CyU3PDmaDscrGetConfig](#)

**Parameters**

<i>index</i>	Index of the descriptor to read.
<i>dscr</i> — <i>p</i>	Output parameter that will be filled with the descriptor values.

**5.16.4.5 uint16\_t CyU3PDmaDscrGetFreeCount ( void )**

Get the number of free descriptors available.

**Description**

This function returns the number of free descriptors available for use.

**Return value**

The number of free descriptors available.

**See also**

[CyU3PDmaDscrGet](#)  
[CyU3PDmaDscrPut](#)

**5.16.4.6 void CyU3PDmaDscrListCreate ( void )**

Create and initialize the free descriptor list.

**Description**

This function initializes the free descriptor list, and marks all of the descriptors as available. This function is invoked internal to the library and is not expected to be called explicitly.

**Return value**

None

**See also**

[CyU3PDmaDscrListDestroy](#)

**5.16.4.7 void CyU3PDmaDscrListDestroy ( void )**

Destroy the free descriptor list.

**Description**

This function de-initializes the free descriptor list, and marks all of the descriptors as non-available. This function is invoked internal to the library and should not be called explicitly.

**Return value**

None

**See also**

[CyU3PDmaDscrListCreate](#)

**5.16.4.8 CyU3PReturnStatus\_t CyU3PDmaDscrPut ( uint16\_t *index* )**

Add a descriptor number to the free list.

**Description**

This function marks a descriptor as available in the free list. This function is used internally by the DMA channel APIs, and can be used to do advanced DMA programming based on direct socket access.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if an invalid index is passed.

**See also**

[CyU3PDmaDscrGet](#)

[CyU3PDmaDscrGetFreeCount](#)

**Parameters**

<i>index</i>	Descriptor index to be marked free.
--------------	-------------------------------------

**5.16.4.9 CyU3PReturnStatus\_t CyU3PDmaDscrSetConfig ( uint16\_t *index*, CyU3PDmaDescriptor\_t \* *dscr\_p* )**

Update the DMA descriptor configuration.

**Description**

Update the specified DMA descriptor in FX3 memory with data from the descriptor structure passed in as parameter.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if an invalid index or a NULL pointer is passed.

**See also**

[CyU3PDmaDscrGetConfig](#)

**Parameters**

<i>index</i>	Index of the descriptor to be updated.
<i>dscr_p</i>	Pointer to descriptor structure containing the desired configuration.

## 5.16.5 Variable Documentation

### 5.16.5.1 CyU3PDmaDescriptor\_t\* gIDmaDescriptor

Pointer to list of DMA descriptors

## 5.17 firmware/u3p\_firmware/inc/cyu3dma.h File Reference

DMA driver and API definitions for EZ-USB FX3.

```
#include "cyu3os.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

## Data Structures

- struct [CyU3PDmaBuffer\\_t](#)  
*DMA buffer data structure.*
- union [CyU3PDmaCBInput\\_t](#)  
*DMA channel callback input.*
- struct [CyU3PDmaChannelConfig\\_t](#)  
*DMA channel parameters.*
- struct [CyU3PDmaMultiChannelConfig\\_t](#)  
*DMA multi-channel parameter structure.*
- struct [CyU3PDmaChannel](#)  
*DMA Channel structure.*
- struct [CyU3PDmaMultiChannel](#)  
*DMA multi-channel structure.*

## Macros

- #define [CY\\_U3P\\_DMA\\_MIN\\_MULTI\\_SCK\\_COUNT](#) (2)  
*Macro defining the minimum required sockets for multi socket DMA channels. Since normal DMA channels can be used for single socket, the minimum number supported is 2.*
- #define [CY\\_U3P\\_DMA\\_MAX\\_MULTI\\_SCK\\_COUNT](#) (4)  
*Macro defining the maximum allowed socket for multi socket DMA channels.*
- #define [CY\\_U3P\\_DMA\\_CHANNEL\\_START\\_SIG](#) (0x43484E4C)  
*Signature used to identify start of a valid DMA channel structure.*
- #define [CY\\_U3P\\_DMA\\_MULTICHN\\_START\\_SIG](#) (0x4D4C4348)  
*Signature used to identify start of a valid DMA multi-channel structure.*
- #define [CY\\_U3P\\_DMA\\_CHANNEL\\_END\\_SIG](#) (0x454E4443)  
*Signature used to identify the end of a valid DMA channel or multi-channel structure.*
- #define [CY\\_U3P\\_DMA\\_BUFFER\\_MARKER](#) (1u << 0)  
*Software based marker that can be applied on a DMA buffer. This bit is part of the four bit status flag that is associated with each DMA buffer descriptor. The DMA manager makes use of this bit to indicate that a specific buffer is expected to be discarded.*
- #define [CY\\_U3P\\_DMA\\_BUFFER\\_EOP](#) (1u << 1)  
*This bit indicates this descriptor refers to the last buffer of a packet or transfer. Packets/transfers may span more than one buffer. The producing IP provides this marker by providing the EOP signal to its DMA adapter.*
- #define [CY\\_U3P\\_DMA\\_BUFFER\\_ERROR](#) (1u << 2)  
*This bit is part of the 4 bit DMA buffer status field and indicates that the buffer transfer has hit an error.*

- `#define CY_U3P_DMA_BUFFER_OCCUPIED (1u << 3)`  
*This status bit indicates the buffer has valid data. This bit can be set even if the buffer count is zero, because the device needs to handle Zero length packets as well.*
- `#define CY_U3P_DMA_BUFFER_STATUS_MASK (0x000F)`  
*Mask to retrieve all DMA buffer status bits.*
- `#define CY_U3P_DMA_BUFFER_STATUS_WRITE_MASK (0x000E)`  
*Mask to identify status bits that can be modified by the user application. Bit 0 (MARKER) is reserved for internal use by the DMA manager.*
- `#define CY_U3P_DMA_BUFFER_AREA_BASE (uint8_t *)(0x40000000)`  
*Start address of the allowed buffer memory area for DMA operations. Corresponds to the beginning of the FX3 System RAM.*
- `#define CY_U3P_DMA_BUFFER_AREA_LIMIT (uint8_t *)(0x40080000)`  
*End address of the allowed buffer memory area for DMA operations. Corresponds to the end of the System RAM for the CYUSB3014 device.*
- `#define CY_U3P_DMA_MAX_BUFFER_SIZE (0xFFFF0)`  
*Maximum allowed size for a single DMA buffer. This is defined by the FX3 device architecture.*
- `#define CY_U3P_DMA_MAX_AVAIL_COUNT (31)`  
*The maximum number of available buffers before a DMA channel can receive data from the producer. This is the maximum value that the prodAvailCount can be set to.*

## Typedefs

- `typedef enum CyU3PDmaSocketId_t CyU3PDmaSocketId_t`  
*DMA socket IDs for all sockets in the device.*
- `typedef enum CyU3PDmaType_t CyU3PDmaType_t`  
*List of DMA single channel types.*
- `typedef enum CyU3PDmaMultiType_t CyU3PDmaMultiType_t`  
*List of DMA multi-channel types.*
- `typedef enum CyU3PDmaState_t CyU3PDmaState_t`  
*List of different DMA channel states.*
- `typedef enum CyU3PDmaMode_t CyU3PDmaMode_t`  
*List of different DMA transfer modes.*
- `typedef enum CyU3PDmaCbType_t CyU3PDmaCbType_t`  
*List of DMA channel callback types.*
- `typedef enum CyU3PDmaSckSuspType_t CyU3PDmaSckSuspType_t`  
*List of DMA socket suspend options.*
- `typedef struct CyU3PDmaBuffer_t CyU3PDmaBuffer_t`  
*DMA buffer data structure.*
- `typedef union CyU3PDmaCBInput_t CyU3PDmaCBInput_t`  
*DMA channel callback input.*
- `typedef void(* CyU3PDmaCallback_t) (CyU3PDmaChannel *handle, CyU3PDmaCbType_t type, CyU3P<->DmaCBInput_t *input)`  
*DMA channel callback function type.*
- `typedef void(* CyU3PDmaMultiCallback_t) (CyU3PDmaMultiChannel *handle, CyU3PDmaCbType_t type, CyU3PDmaCBInput_t *input)`  
*Multi socket DMA channel callback function type.*
- `typedef struct CyU3PDmaChannelConfig_t CyU3PDmaChannelConfig_t`  
*DMA channel parameters.*
- `typedef struct CyU3PDmaMultiChannelConfig_t CyU3PDmaMultiChannelConfig_t`  
*DMA multi-channel parameter structure.*

## Enumerations

- enum CyU3PDmaSocketId\_t {  
CY\_U3P\_LPP\_SOCKET\_I2S\_LEFT = 0x0000, CY\_U3P\_LPP\_SOCKET\_I2S\_RIGHT, CY\_U3P\_LPP\_SOCKET\_I2C\_CONS, CY\_U3P\_LPP\_SOCKET\_UART\_CONS,  
CY\_U3P\_LPP\_SOCKET\_SPI\_CONS, CY\_U3P\_LPP\_SOCKET\_I2C\_PROD, CY\_U3P\_LPP\_SOCKET\_UART\_PROD, CY\_U3P\_LPP\_SOCKET\_SPI\_PROD,  
CY\_U3P\_PIB\_SOCKET\_0 = 0x0100, CY\_U3P\_PIB\_SOCKET\_1, CY\_U3P\_PIB\_SOCKET\_2, CY\_U3P\_PIB\_SOCKET\_3,  
CY\_U3P\_PIB\_SOCKET\_4, CY\_U3P\_PIB\_SOCKET\_5, CY\_U3P\_PIB\_SOCKET\_6, CY\_U3P\_PIB\_SOCKET\_7,  
CY\_U3P\_PIB\_SOCKET\_8, CY\_U3P\_PIB\_SOCKET\_9, CY\_U3P\_PIB\_SOCKET\_10, CY\_U3P\_PIB\_SOCKET\_11,  
CY\_U3P\_PIB\_SOCKET\_12, CY\_U3P\_PIB\_SOCKET\_13, CY\_U3P\_PIB\_SOCKET\_14, CY\_U3P\_PIB\_SOCKET\_15,  
CY\_U3P\_PIB\_SOCKET\_16, CY\_U3P\_PIB\_SOCKET\_17, CY\_U3P\_PIB\_SOCKET\_18, CY\_U3P\_PIB\_SOCKET\_19,  
CY\_U3P\_PIB\_SOCKET\_20, CY\_U3P\_PIB\_SOCKET\_21, CY\_U3P\_PIB\_SOCKET\_22, CY\_U3P\_PIB\_SOCKET\_23,  
CY\_U3P\_PIB\_SOCKET\_24, CY\_U3P\_PIB\_SOCKET\_25, CY\_U3P\_PIB\_SOCKET\_26, CY\_U3P\_PIB\_SOCKET\_27,  
CY\_U3P\_PIB\_SOCKET\_28, CY\_U3P\_PIB\_SOCKET\_29, CY\_U3P\_PIB\_SOCKET\_30, CY\_U3P\_PIB\_SOCKET\_31,  
CY\_U3P\_SIB\_SOCKET\_0 = 0x0200, CY\_U3P\_SIB\_SOCKET\_1, CY\_U3P\_SIB\_SOCKET\_2, CY\_U3P\_SIB\_SOCKET\_3,  
CY\_U3P\_SIB\_SOCKET\_4, IB\_SOCKET\_3,

```

U3P_DMA_PROD_OVERRIDE,
CY_U3P_DMA_CONS_OVERRIDE, CY_U3P_DMA_ERROR, CY_U3P_DMA_IN_COMPLETION, CY_U3P_DMA_ABORTED,
CY_U3P_DMA_NUM_STATES, CY_U3P_DMA_XFER_COMPLETED, CY_U3P_DMA_SEND_COMPLETED, CY_U3P_DMA_RECV_COMPLETED }

```

*List of different DMA channel states.*

- enum CyU3PDmaMode\_t { CY\_U3P\_DMA\_MODE\_BYTE = 0, CY\_U3P\_DMA\_MODE\_BUFFER, CY\_U3P\_DMA\_NUM\_MODES }

*List of different DMA transfer modes.*

- enum CyU3PDmaCbType\_t {
 CY\_U3P\_DMA\_CB\_XFER\_CPLT = (1 << 0), CY\_U3P\_DMA\_CB\_SEND\_CPLT = (1 << 1), CY\_U3P\_DMA\_CB\_RECV\_CPLT = (1 << 2), CY\_U3P\_DMA\_CB\_PROD\_EVENT = (1 << 3),
 CY\_U3P\_DMA\_CB\_CONS\_EVENT = (1 << 4), CY\_U3P\_DMA\_CB\_ABORTED = (1 << 5), CY\_U3P\_DMA\_CB\_ERROR = (1 << 6), CY\_U3P\_DMA\_CB\_PROD\_SUSP = (1 << 7),
 CY\_U3P\_DMA\_CB\_CONS\_SUSP = (1 << 8) }

*List of DMA channel callback types.*

- enum CyU3PDmaSckSuspType\_t { CY\_U3P\_DMA\_SCK\_SUSP\_NONE = 0, CY\_U3P\_DMA\_SCK\_SUSP\_EOP, CY\_U3P\_DMA\_SCK\_SUSP\_CUR\_BUF, CY\_U3P\_DMA\_SCK\_SUSP\_CONS\_PARTIAL\_BUF }

*List of DMA socket suspend options.*

## Functions

- CyU3PDmaChannel \* CyU3PDmaChannelGetHandle (CyU3PDmaSocketId\_t sckId)
 

*Fetch a handle to the DMA channel corresponding to the specified socket.*
- CyU3PReturnStatus\_t CyU3PDmaChannelCreate (CyU3PDmaChannel \*handle, CyU3PDmaType\_t type, CyU3PDmaChannelConfig\_t \*config)
 

*Create a one-to-one socket DMA channel.*
- CyU3PReturnStatus\_t CyU3PDmaChannelDestroy (CyU3PDmaChannel \*handle)
 

*Destroy a one-to-one socket DMA channel.*
- CyU3PReturnStatus\_t CyU3PDmaChannelUpdateMode (CyU3PDmaChannel \*handle, CyU3PDmaMode\_t dmaMode)
 

*Update the DMA mode for a one-to-one DMA channel.*
- CyU3PReturnStatus\_t CyU3PDmaChannelSetXfer (CyU3PDmaChannel \*handle, uint32\_t count)
 

*Setup a one-to-one DMA channel for data transfer.*
- CyU3PReturnStatus\_t CyU3PDmaChannelSetWrapUp (CyU3PDmaChannel \*handle)
 

*Wraps up the current active buffer for the channel from the producer side.*
- CyU3PReturnStatus\_t CyU3PDmaChannelSetSuspend (CyU3PDmaChannel \*handle, CyU3PDmaSckSuspType\_t prodSusp, CyU3PDmaSckSuspType\_t consSusp)
 

*Set the suspend options for the sockets associated with a DMA channel.*
- CyU3PReturnStatus\_t CyU3PDmaChannelResume (CyU3PDmaChannel \*handle, CyBool\_t isProdResume, CyBool\_t isConsResume)
 

*Resume a suspended DMA channel.*
- CyU3PReturnStatus\_t CyU3PDmaChannelAbort (CyU3PDmaChannel \*handle)
 

*Aborts a DMA channel.*
- CyU3PReturnStatus\_t CyU3PDmaChannelReset (CyU3PDmaChannel \*handle)
 

*Aborts and resets a DMA channel.*
- CyU3PReturnStatus\_t CyU3PDmaChannelSetupSendBuffer (CyU3PDmaChannel \*handle, CyU3PDmaBuffer\_t \*buffer\_p)
 

*Send the contents of a user provided buffer to the consumer.*
- CyU3PReturnStatus\_t CyU3PDmaChannelSetupRecvBuffer (CyU3PDmaChannel \*handle, CyU3PDmaBuffer\_t \*buffer\_p)
 

*Receive data into a user provided DMA buffer.*

- `CyU3PReturnStatus_t CyU3PDmaChannelWaitForRecvBuffer (CyU3PDmaChannel *handle, CyU3PDmaBuffer_t *buffer_p, uint32_t waitOption)`

*Wait until an override read operation is completed.*
- `CyU3PReturnStatus_t CyU3PDmaChannelGetBuffer (CyU3PDmaChannel *handle, CyU3PDmaBuffer_t *buffer_p, uint32_t waitOption)`

*Get the current buffer pointer.*
- `CyU3PReturnStatus_t CyU3PDmaChannelCommitBuffer (CyU3PDmaChannel *handle, uint16_t count, uint16_t bufStatus)`

*Commit a data buffer to be sent to the consumer.*
- `CyU3PReturnStatus_t CyU3PDmaChannelDiscardBuffer (CyU3PDmaChannel *handle)`

*Drop a data buffer without sending out to the consumer.*
- `CyU3PReturnStatus_t CyU3PDmaChannelWaitForCompletion (CyU3PDmaChannel *handle, uint32_t waitOption)`

*Wait for the current DMA transaction to complete.*
- `CyU3PReturnStatus_t CyU3PDmaChannelGetStatus (CyU3PDmaChannel *handle, CyU3PDmaState_t *state, uint32_t *prodXferCount, uint32_t *consXferCount)`

*This function returns the current channel status.*
- `CyU3PReturnStatus_t CyU3PDmaChannelCacheControl (CyU3PDmaChannel *handle, CyBool_t isDmaHandleDCache)`

*This function is used to enable/disable the Data cache coherency handling on a per DMA channel basis.*
- `CyU3PDmaMultiChannel * CyU3PDmaMultiChannelGetHandle (CyU3PDmaSocketId_t sckId)`

*Identifies the multi-channel that is associated with the specified socket.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelCreate (CyU3PDmaMultiChannel *handle, CyU3PDmaMultiType_t type, CyU3PDmaMultiChannelConfig_t *config)`

*Create a multi-socket DMA channel with the specified parameters.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelDestroy (CyU3PDmaMultiChannel *handle)`

*Destroy a multi-socket DMA channel.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelUpdateMode (CyU3PDmaMultiChannel *handle, CyU3PDmaMode_t dmaMode)`

*Update the DMA mode for a multi-channel.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelSetXfer (CyU3PDmaMultiChannel *handle, uint32_t count, uint16_t multiSckOffset)`

*Prepare a DMA multi-channel for data transfer.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelSetWrapUp (CyU3PDmaMultiChannel *handle, uint16_t multiSckOffset)`

*Wraps up the current active buffer on the producer socket.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelSetSuspend (CyU3PDmaMultiChannel *handle, CyU3PDmaSckSuspType_t prodSusp, CyU3PDmaSckSuspType_t consSusp)`

*Update the suspend options for the sockets associated with a DMA multi-channel.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelResume (CyU3PDmaMultiChannel *handle, CyBool_t isProdResume, CyBool_t isConsResume)`

*Resume a suspended DMA multi-channel.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelAbort (CyU3PDmaMultiChannel *handle)`

*Abort the ongoing transfer on a DMA multi-channel.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelReset (CyU3PDmaMultiChannel *handle)`

*Abort ongoing transfers on and resets a DMA multi-channel.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelSetupSendBuffer (CyU3PDmaMultiChannel *handle, CyU3PDmaBuffer_t *buffer_p, uint16_t multiSckOffset)`

*Send the contents of a user provided buffer to the consumer.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelSetupRecvBuffer (CyU3PDmaMultiChannel *handle, CyU3PDmaBuffer_t *buffer_p, uint16_t multiSckOffset)`

*Receive data into a user provided DMA buffer.*

- `CyU3PReturnStatus_t CyU3PDmaMultiChannelWaitForRecvBuffer (CyU3PDmaMultiChannel *handle, CyU3PDmaBuffer_t *buffer_p, uint32_t waitOption)`

*Wait until an override read operation is completed.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelGetBuffer (CyU3PDmaMultiChannel *handle, CyU3PDmaBuffer_t *buffer_p, uint32_t waitOption)`

*Get the current buffer pointer.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelCommitBuffer (CyU3PDmaMultiChannel *handle, uint16_t count, uint16_t bufStatus)`

*Commit the buffer to be sent to the consumer.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelDiscardBuffer (CyU3PDmaMultiChannel *handle)`

*Drop a data buffer without sending out to the consumer.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelWaitForCompletion (CyU3PDmaMultiChannel *handle, uint32_t waitOption)`

*Wait for the current DMA transaction to complete on a DMA multi-channel.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelGetStatus (CyU3PDmaMultiChannel *handle, CyU3PDmaState_t *state, uint32_t *prodXferCount, uint32_t *consXferCount, uint8_t scklIndex)`

*This function returns the current multi-channel status.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelCacheControl (CyU3PDmaMultiChannel *handle, CyBool_t isDmaHandleDCache)`

*This function is used to enable/disable the Data cache coherency handling on a per DMA channel basis.*
- `void CyU3PDmaEnableMulticast (void)`

*Enable creation and management of DMA multicast channels.*
- `CyU3PReturnStatus_t CyU3PDmaMulticastSocketSelect (CyU3PDmaMultiChannel *chHandle, uint32_t consMask)`

*Select the active consumers on a multicast DMA channel.*
- `CyU3PReturnStatus_t CyU3PDmaMulticastDisableConsumer (CyU3PDmaMultiChannel *chHandle, uint16_t consIndex)`

*Disable a consumer on a multi-cast socket while transfer is in flight.*
- `CyBool_t CyU3PDmaChannelsValid (CyU3PDmaChannel *handle)`

*Check whether a given DMA channel handle is valid.*
- `CyBool_t CyU3PDmaMultiChannelsValid (CyU3PDmaMultiChannel *handle)`

*Check whether a given DMA multi-channel handle is valid.*
- `CyU3PDmaChannel * CyU3PDmaUsbInEpGetChannel (uint8_t ep)`

*Get the DMA channel handle corresponding to a USB IN endpoint.*
- `CyU3PDmaMultiChannel * CyU3PDmaUsbInEpGetMultiChannel (uint8_t ep)`

*Get the DMA multi-channel handle corresponding to a USB IN endpoint.*
- `CyU3PReturnStatus_t CyU3PDmaChannelSuspendUsbConsumer (CyU3PDmaChannel *handle, uint32_t waitOption)`

*Suspend the USB IN (egress) socket corresponding to the DMA channel.*
- `CyU3PReturnStatus_t CyU3PDmaChannelResumeUsbConsumer (CyU3PDmaChannel *handle)`

*Resume the USB IN (egress) socket corresponding to the DMA channel.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelSuspendUsbConsumer (CyU3PDmaMultiChannel *handle, uint32_t waitOption)`

*Suspend the USB IN (egress) socket corresponding to the DMA multi-channel.*
- `CyU3PReturnStatus_t CyU3PDmaMultiChannelResumeUsbConsumer (CyU3PDmaMultiChannel *handle)`

*Resume the USB IN (egress) socket corresponding to the DMA multi-channel.*

### 5.17.1 Detailed Description

DMA driver and API definitions for EZ-USB FX3.

#### Description

The FX3 device architecture includes a distributed DMA fabric that allows high performance data transfer between any of the external interfaces of the device. This DMA engine is a custom implementation that makes use of multiple registers and data structures for the data path management.

The DMA driver in the FX3 firmware takes care of configuring the DMA data paths as desired and performs all runtime control operations such as interrupt handling. The FX3 library also provides a set of APIs through which the user application can set up and control DMA data paths.

### 5.17.2 Typedef Documentation

#### 5.17.2.1 `typedef struct CyU3PDmaBuffer_t CyU3PDmaBuffer_t`

DMA buffer data structure.

#### Description

The data structure is used to describe the status of a DMA buffer. It holds the address, size, valid data count, and the status information. This type is used in DMA callbacks and APIs to identify the properties of the data buffer to be transferred.

#### See also

[CyU3PDmaCBInput\\_t](#)  
[CyU3PDmaChannelGetBuffer](#)  
[CyU3PDmaChannelSetupSendBuffer](#)  
[CyU3PDmaChannelSetupRecvBuffer](#)  
[CyU3PDmaChannelWaitForRecvBuffer](#)  
[CyU3PDmaMultiChannelGetBuffer](#)  
[CyU3PDmaMultiChannelSetupSendBuffer](#)  
[CyU3PDmaMultiChannelSetupRecvBuffer](#)  
[CyU3PDmaMultiChannelWaitForRecvBuffer](#)

#### 5.17.2.2 `typedef void(* CyU3PDmaCallback_t) (CyU3PDmaChannel *handle, CyU3PDmaCbType_t type, CyU3PDmaCBInput_t *input )`

DMA channel callback function type.

#### Description

The FX3 DMA manager supports a number of event notifications that can be sent to the user application during system operation. These event notifications are provided per DMA channel, and the types of events required can be selected at the time of channel creation.

This type defines the prototype for the callback function which will be invoked to provide these event notifications.

No blocking calls should be made from these callback functions. If data processing is required, it should be done outside of the callback function.

If produce events are enabled, the application is expected to be fast enough to handle the incoming data rate. The input parameter can be stale if the handling of the buffer is not done in a timely fashion.

In the case of produce events on AUTO\_SIGNAL channels, the input parameter points to the latest produced buffer. If the callback handling is delayed, the producer socket may overwrite the buffer before the event is handled.

In case of MANUAL or MANUAL\_IN channels, the input parameter points to the first buffer left to be committed to the consumer socket. If the buffer is not committed before the next callback, then the input parameter shall be stale data.

See also

[CyU3PDmaCBInput\\_t](#)  
[CyU3PDmaChannelConfig\\_t](#)  
[CyU3PDmaChannelCreate](#)

#### 5.17.2.3 `typedef union CyU3PDmaCBInput_t CyU3PDmaCBInput_t`

DMA channel callback input.

##### Description

This data structure is used to provide event specific information when a DMA callback is called. This structure is defined as a union to facilitate future updates to the DMA manager.

See also

[CyU3PDmaBuffer\\_t](#)  
[CyU3PDmaCallback\\_t](#)

#### 5.17.2.4 `typedef enum CyU3PDmaCbType_t CyU3PDmaCbType_t`

List of DMA channel callback types.

##### Description

This type lists the various callback types that are supported by the DMA manager for various DMA channels. The user can register the combination of these events for which event notifications are required at channel creation time.

See also

[CyU3PDmaCallback\\_t](#)  
[CyU3PDmaChannelConfig\\_t](#)  
[CyU3PDmaMultiChannelConfig\\_t](#)  
[CyU3PDmaChannelCreate](#)  
[CyU3PDmaMultiChannelCreate](#)

#### 5.17.2.5 `typedef struct CyU3PDmaChannelConfig_t CyU3PDmaChannelConfig_t`

DMA channel parameters.

##### Description

This structure encapsulates the parameters that are provided at the time of DMA channel creation, and specifies the resources and events that are to be associated with the channel.

The size field specifies the size in bytes of each DMA buffer to be allocated for this DMA channel.

The offsets prodHeader, prodFooter and consHeader are used to do header addition and removal. These are valid only for manual channels and should be zero for auto channels.

The buffer address seen by the producer = (buffer + prodHeader).

The buffer size seen by the producer = (size - prodHeader - prodFooter).

The buffer address seen by the consumer = (buffer + consHeader).

The buffer size seen by the consumer = (buffer - consHeader).

For header addition to the buffer generated by the producer, the prodHeader should be the length of the header to be added and the other offsets should be zero. Once the buffer is generated, the header can be modified manually by the CPU and committed using the CyU3PDmaChannelCommitBuffer call.

For footer addition to the buffer generated by the producer, the prodFooter should be the length of the footer to be added and the other offsets should be zero. Once the buffer is generated, the footer can be added and committed using the CyU3PDmaChannelCommitBuffer call.

For header deletion from the buffer generated by the producer, the consHeader should be the length of the header to be removed and the other offsets should be zero. Once the buffer is generated, the buffer can be committed to the consumer socket with only change to the size of the data to be transmitted using the CommitBuffer call.

The size of the buffer as seen by the producer socket should always be a multiple of 16 bytes; ie, (size - prodHeader - prodFooter) must be a multiple of 16 bytes.

The prodAvailCount count should always be zero. This is used only for very specific use case where there should always be free buffers. Since there is no current use case for such a channel, this field should always be zero.

The count field specifies the number of buffers that should be allocated for this DMA channel. It is possible to obtain a large buffering depth by specifying a large count value, subject to availability of DMA buffer space and free descriptors.

#### See also

[CyU3PDmaChannelCreate](#)

### 5.17.2.6 `typedef enum CyU3PDmaMode_t CyU3PDmaMode_t`

List of different DMA transfer modes.

#### Description

The following are the different types of DMA transfer modes. The default mode of operation is byte mode. The buffer mode is useful only when there are variable data sized transfers and when firmware handling is required after a finite number of buffers.

#### See also

[CyU3PDmaChannelConfig\\_t](#)  
[CyU3PDmaMultiChannelConfig\\_t](#)  
[CyU3PDmaChannelCreate](#)  
[CyU3PDmaChannelUpdateMode](#)  
[CyU3PDmaMultiChannelUpdateMode](#)

### 5.17.2.7 `typedef void(* CyU3PDmaMultiCallback_t)(CyU3PDmaMultiChannel *handle, CyU3PDmaCbType_t type, CyU3PDmaCBInput_t *input)`

Multi socket DMA channel callback function type.

#### Description

Callback function type that is associated with DMA multi-channels. All of the usage restrictions and guidelines that apply to normal DMA channel callbacks (CyU3PDmaCallback\_t) also apply to these callbacks. The only difference is in the type of the first parameter passed to the callback function.

#### See also

[CyU3PDmaCallback\\_t](#)  
[CyU3PDmaCBInput\\_t](#)  
[CyU3PDmaMultiChannelConfig\\_t](#)  
[CyU3PDmaMultiChannelCreate](#)

### 5.17.2.8 `typedef struct CyU3PDmaMultiChannelConfig_t CyU3PDmaMultiChannelConfig_t`

DMA multi-channel parameter structure.

#### Description

This structure encapsulates all the parameters required to create a DMA multi-channel.

In the case of many to one channels, there shall be 'validSckCount' number of producer sockets and only one consumer socket. The producer sockets needs to be updated in the required order of operation. The first buffer shall be taken from the prodSckId[0], second from prodSckId[1] and so on. If only two producer sockets are used, then only prodSckId[0], prodSckId[1] and consSckId[0] shall be considered.

In the case of one to many operations, there shall be only one producer socket and 'validSckCount' number of consumer sockets.

The size field is the total buffer that needs to be allocated for DMA operations. This field has restrictions for DMA operations.

The size, count, prodHeader, prodFooter, consHeader and prodAvailCount fields are used in the same way as in the [CyU3PDmaChannelConfig\\_t](#) structure.

#### See also

[CyU3PDmaChannelConfig\\_t](#)  
[CyU3PDmaMultiChannelCreate](#)

### 5.17.2.9 `typedef enum CyU3PDmaMultiType_t CyU3PDmaMultiType_t`

List of DMA multi-channel types.

#### Description

In some cases, a simple one-to-one DMA channel is insufficient to handle the data flowing through the FX3 device. For example, a user may need to use multiple PIB (GPIF) sockets to collect the data coming from an image sensor without any data loss. Special handling is required to combine the data from these sockets into a single USB endpoint for streaming.

Such special DMA handling is achieved through the use of multi-channels. Different types of multi-channels are supported by the DMA manager, and this enumerated type lists them. Special APIs with a [CyU3PDmaMultiChannel](#) prefix need to be used for creating and managing DMA multi-channels.

#### See also

[CyU3PDmaType\\_t](#)  
[CyU3PDmaMultiChannelCreate](#)  
[CyU3PDmaEnableMulticast](#)

### 5.17.2.10 `typedef enum CyU3PDmaSckSuspType_t CyU3PDmaSckSuspType_t`

List of DMA socket suspend options.

#### Description

The producer and consume sockets associated with a DMA channel can be suspended based on various triggers. Each of these suspend triggers/options behaves differently.

#### Note

In case of multi channels, only the single socket side can be suspended. For a many to one channels, the producer sockets cannot be suspended; and for one to many channels, the consumer sockets cannot be suspended.

#### See also

[CyU3PDmaChannelCreate](#)  
[CyU3PDmaChannelSetSuspend](#)  
[CyU3PDmaChannelResume](#)  
[CyU3PDmaMultiChannelCreate](#)

### 5.17.2.11 `typedef enum CyU3PDmaSocketId_t CyU3PDmaSocketId_t`

DMA socket IDs for all sockets in the device.

#### Description

This is a software representation of all sockets on the device. The socket ID has two parts: IP number and socket number. Each peripheral (IP) has a fixed ID. LPP is 0, PIB is 1, Storage port is 2, USB egress is 3 and USB ingress is 4.

Each peripheral has a number of sockets. The LPP sockets are fixed and have to be used as defined. The P $\leftarrow$ IB sockets 0-15 can be used as both producer and consumer, but the PIB sockets 16-31 are strictly producer sockets. The UIB sockets are defined as 0-15 producer and 0-15 consumer sockets. The CPU sockets are virtual representations and have no backing hardware block.

#### See also

[CyU3PDmaChannelConfig\\_t](#)  
[CyU3PDmaMultiChannelConfig\\_t](#)  
[CyU3PDmaChannelCreate](#)  
[CyU3PDmaMultiChannelCreate](#)

### 5.17.2.12 `typedef enum CyU3PDmaState_t CyU3PDmaState_t`

List of different DMA channel states.

#### Description

The following are the different states that a DMA channel can be in. All states until CY\_U3P\_DMA\_NUM\_STATES are channel states, where-as those after it are virtual state values returned on the GetStatus calls only.

#### See also

[CyU3PDmaChannelGetStatus](#)

### 5.17.2.13 `typedef enum CyU3PDmaType_t CyU3PDmaType_t`

List of DMA single channel types.

#### Description

A DMA channel aggregates all of the resources required to manage a unique data flow through the FX3 device. A DMA channel needs to be created before any DMA operation can be performed.

In the normal case, each DMA channel has one producer socket through which data is received on the FX3 device; and one consumer socket through which data is sent out of the FX3 device. The DMA channels can be classified into multiple types based on the amount of firmware intervention and control that is required in the data flow.

This enumeration lists the different types of DMA single (one to one) channels. All APIs that operate on these channels have a [CyU3PDmaChannel](#) prefix.

#### See also

[CyU3PDmaChannelCreate](#)

## 5.17.3 Enumeration Type Documentation

### 5.17.3.1 `enum CyU3PDmaCbType_t`

List of DMA channel callback types.

#### Description

This type lists the various callback types that are supported by the DMA manager for various DMA channels. The user can register the combination of these events for which event notifications are required at channel creation time.

See also

[CyU3PDmaCallback\\_t](#)  
[CyU3PDmaChannelConfig\\_t](#)  
[CyU3PDmaMultiChannelConfig\\_t](#)  
[CyU3PDmaChannelCreate](#)  
[CyU3PDmaMultiChannelCreate](#)

Enumerator

**CY\_U3P\_DMA\_CB\_XFER\_CPLT** Transfer has been completed. This event is generated when a finite transfer queued with SetXfer is completed.

**CY\_U3P\_DMA\_CB\_SEND\_CPLT** SendBuffer call has been completed. This event is generated when the data queued with SendBuffer has been successfully sent.

**CY\_U3P\_DMA\_CB\_RECV\_CPLT** ReceiverBuffer call has been completed. This event is generated when data is received successfully on the producer socket.

**CY\_U3P\_DMA\_CB\_PROD\_EVENT** Buffer received from producer. This event is generated when a buffer is generated by the producer socket when a transfer is queued with SetXfer.

**CY\_U3P\_DMA\_CB\_CONS\_EVENT** Buffer consumed by the consumer. This event is generated when a buffer is sent out by the consumer socket when a transfer is queued with SetXfer.

**CY\_U3P\_DMA\_CB\_ABORTED** This event is generated when the Abort API is invoked.

**CY\_U3P\_DMA\_CB\_ERROR** This event is generated when the hardware detects an error.

**CY\_U3P\_DMA\_CB\_PROD\_SUSP** This event is generated when the producer socket is suspended.

**CY\_U3P\_DMA\_CB\_CONS\_SUSP** This event is generated when the consumer socket is suspended.

### 5.17.3.2 enum CyU3PDmaMode\_t

List of different DMA transfer modes.

Description

The following are the different types of DMA transfer modes. The default mode of operation is byte mode. The buffer mode is useful only when there are variable data sized transfers and when firmware handling is required after a finite number of buffers.

See also

[CyU3PDmaChannelConfig\\_t](#)  
[CyU3PDmaMultiChannelConfig\\_t](#)  
[CyU3PDmaChannelCreate](#)  
[CyU3PDmaChannelUpdateMode](#)  
[CyU3PDmaMultiChannelUpdateMode](#)

Enumerator

**CY\_U3P\_DMA\_MODE\_BYTE** Transfer is based on byte count. This is the default mode of operation. The transfer count is done based on number of bytes received / sent.

**CY\_U3P\_DMA\_MODE\_BUFFER** Transfer is based on buffer count. The transfer count is based on the number of buffers generated or consumed. This is useful only when the data size is variable but there should be finite handling after N buffers.

**CY\_U3P\_DMA\_NUM\_MODES** Count of DMA modes. This is just a place holder and not a valid mode.

### 5.17.3.3 enum CyU3PDmaMultiType\_t

List of DMA multi-channel types.

**Description**

In some cases, a simple one-to-one DMA channel is insufficient to handle the data flowing through the FX3 device. For example, a user may need to use multiple PIB (GPIF) sockets to collect the data coming from an image sensor without any data loss. Special handling is required to combine the data from these sockets into a single USB endpoint for streaming.

Such special DMA handling is achieved through the use of multi-channels. Different types of multi-channels are supported by the DMA manager, and this enumerated type lists them. Special APIs with a [CyU3PDmaMultiChannel](#) prefix need to be used for creating and managing DMA multi-channels.

## See also

[CyU3PDmaType\\_t](#)  
[CyU3PDmaMultiChannelCreate](#)  
[CyU3PDmaEnableMulticast](#)

## Enumerator

**CY\_U3P\_DMA\_TYPE\_AUTO\_MANY\_TO\_ONE** Auto mode many to one interleaved DMA channel.  
**CY\_U3P\_DMA\_TYPE\_AUTO\_ONE\_TO\_MANY** Auto mode one to many interleaved DMA channel.  
**CY\_U3P\_DMA\_TYPE\_MANUAL\_MANY\_TO\_ONE** Manual mode many to one interleaved DMA channel.  
**CY\_U3P\_DMA\_TYPE\_MANUAL\_ONE\_TO\_MANY** Manual mode one to many interleaved DMA channel.  
**CY\_U3P\_DMA\_TYPE\_MULTICAST** Multicast mode with one producer and multiple consumers for the same buffer. This is a manual channel. Please note that the [CyU3PDmaEnableMulticast](#) API needs to be called before any multicast DMA channels are created.  
**CY\_U3P\_DMA\_NUM\_TYPES** Number of DMA channel types.

## 5.17.3.4 enum CyU3PDmaSckSuspType\_t

List of DMA socket suspend options.

**Description**

The producer and consume sockets associated with a DMA channel can be suspended based on various triggers. Each of these suspend triggers/options behaves differently.

**Note**

In case of multi channels, only the single socket side can be suspended. For a many to one channels, the producer sockets cannot be suspended; and for one to many channels, the consumer sockets cannot be suspended.

## See also

[CyU3PDmaChannelCreate](#)  
[CyU3PDmaChannelSetSuspend](#)  
[CyU3PDmaChannelResume](#)  
[CyU3PDmaMultiChannelCreate](#)

## Enumerator

**CY\_U3P\_DMA\_SCK\_SUSP\_NONE** Socket will not be suspended. This option is used to remove any existing suspend triggers on the selected socket.  
**CY\_U3P\_DMA\_SCK\_SUSP\_EOP** Socket will be suspended after handling any buffer with the EOP bit set. Typically the EOP bit will be set on any data buffers that are wrapped up on the PIB (GPIF) side through a COMMIT action. The DMA buffer with the EOP bit set would have been handled before the socket gets suspended, meaning that it is not possible to make changes to the contents of that data packet. This option can be applied to both producer and consumer sockets, and is sticky.  
**CY\_U3P\_DMA\_SCK\_SUSP\_CUR\_BUF** Socket will be suspended after the current buffer is completed. This can be used to suspend the socket at a defined point to be resumed later. This option is valid for only the current buffer, and the socket option will change back to CY\_U3P\_DMA\_SCK\_SUSP\_NONE on resumption.

**CY\_U3P\_DMA\_SCK\_SUSP\_CONS\_PARTIAL\_BUF** This option is valid only for consumer sockets, and allows the socket to be suspended before handling any partially filled (count < size) DMA buffer. This mode al3Td [T3pend(us-280l3Td o80l3Tdma376(so80l3Tdc0(oges80l3Td o80l3Td [T3pen0(data3pend(ended)-375(ben0

**CY\_U3P\_PIB\_SOCKET\_15** P-port socket number 15.  
**CY\_U3P\_PIB\_SOCKET\_16** P-port socket number 16.  
**CY\_U3P\_PIB\_SOCKET\_17** P-port socket number 17.  
**CY\_U3P\_PIB\_SOCKET\_18** P-port socket number 18.  
**CY\_U3P\_PIB\_SOCKET\_19** P-port socket number 19.  
**CY\_U3P\_PIB\_SOCKET\_20** P-port socket number 20.  
**CY\_U3P\_PIB\_SOCKET\_21** P-port socket number 21.  
**CY\_U3P\_PIB\_SOCKET\_22** P-port socket number 22.  
**CY\_U3P\_PIB\_SOCKET\_23** P-port socket number 23.  
**CY\_U3P\_PIB\_SOCKET\_24** P-port socket number 24.  
**CY\_U3P\_PIB\_SOCKET\_25** P-port socket number 25.  
**CY\_U3P\_PIB\_SOCKET\_26** P-port socket number 26.  
**CY\_U3P\_PIB\_SOCKET\_27** P-port socket number 27.  
**CY\_U3P\_PIB\_SOCKET\_28** P-port socket number 28.  
**CY\_U3P\_PIB\_SOCKET\_29** P-port socket number 29.  
**CY\_U3P\_PIB\_SOCKET\_30** P-port socket number 30.  
**CY\_U3P\_PIB\_SOCKET\_31** P-port socket number 31.  
**CY\_U3P\_SIB\_SOCKET\_0** S-port socket number 0.  
**CY\_U3P\_SIB\_SOCKET\_1** S-port socket number 1.  
**CY\_U3P\_SIB\_SOCKET\_2** S-port socket number 2.  
**CY\_U3P\_SIB\_SOCKET\_3** S-port socket number 3.  
**CY\_U3P\_SIB\_SOCKET\_4** S-port socket number 4.  
**CY\_U3P\_SIB\_SOCKET\_5** S-port socket number 5.  
**CY\_U3P\_UIB\_SOCKET\_CONS\_0** U-port output socket number 0.  
**CY\_U3P\_UIB\_SOCKET\_CONS\_1** U-port output socket number 1.  
**CY\_U3P\_UIB\_SOCKET\_CONS\_2** U-port output socket number 2.  
**CY\_U3P\_UIB\_SOCKET\_CONS\_3** U-port output socket number 3.  
**CY\_U3P\_UIB\_SOCKET\_CONS\_4** U-port output socket number 4.  
**CY\_U3P\_UIB\_SOCKET\_CONS\_5** U-port output socket number 5.  
**CY\_U3P\_UIB\_SOCKET\_CONS\_6** U-port output socket number 6.  
**CY\_U3P\_UIB\_SOCKET\_CONS\_7** U-port output socket number 7.  
**CY\_U3P\_UIB\_SOCKET\_CONS\_8** U-port output socket number 8.  
**CY\_U3P\_UIB\_SOCKET\_CONS\_9** U-port output socket number 9.  
**CY\_U3P\_UIB\_SOCKET\_CONS\_10** U-port output socket number 10.  
**CY\_U3P\_UIB\_SOCKET\_CONS\_11** U-port output socket number 11.  
**CY\_U3P\_UIB\_SOCKET\_CONS\_12** U-port output socket number 12.  
**CY\_U3P\_UIB\_SOCKET\_CONS\_13** U-port output socket number 13.  
**CY\_U3P\_UIB\_SOCKET\_CONS\_14** U-port output socket number 14.  
**CY\_U3P\_UIB\_SOCKET\_CONS\_15** U-port output socket number 15.  
**CY\_U3P\_UIB\_SOCKET\_PROD\_0** U-port input socket number 0.  
**CY\_U3P\_UIB\_SOCKET\_PROD\_1** U-port input socket number 1.  
**CY\_U3P\_UIB\_SOCKET\_PROD\_2** U-port input socket number 2.  
**CY\_U3P\_UIB\_SOCKET\_PROD\_3** U-port input socket number 3.  
**CY\_U3P\_UIB\_SOCKET\_PROD\_4** U-port input socket number 4.  
**CY\_U3P\_UIB\_SOCKET\_PROD\_5** U-port input socket number 5.

**CY\_U3P\_UIB\_SOCKET\_PROD\_6** U-port input socket number 6.  
**CY\_U3P\_UIB\_SOCKET\_PROD\_7** U-port input socket number 7.  
**CY\_U3P\_UIB\_SOCKET\_PROD\_8** U-port input socket number 8.  
**CY\_U3P\_UIB\_SOCKET\_PROD\_9** U-port input socket number 9.  
**CY\_U3P\_UIB\_SOCKET\_PROD\_10** U-port input socket number 10.  
**CY\_U3P\_UIB\_SOCKET\_PROD\_11** U-port input socket number 11.  
**CY\_U3P\_UIB\_SOCKET\_PROD\_12** U-port input socket number 12.  
**CY\_U3P\_UIB\_SOCKET\_PROD\_13** U-port input socket number 13.  
**CY\_U3P\_UIB\_SOCKET\_PROD\_14** U-port input socket number 14.  
**CY\_U3P\_UIB\_SOCKET\_PROD\_15** U-port input socket number 15.  
**CY\_U3P\_CPU\_SOCKET\_CONS** Socket through which the FX3 CPU receives data.  
**CY\_U3P\_CPU\_SOCKET\_PROD** Socket through which the FX3 CPU produces data.

#### 5.17.3.6 enum CyU3PDmaState\_t

List of different DMA channel states.

##### Description

The following are the different states that a DMA channel can be in. All states until CY\_U3P\_DMA\_NUM\_STATES are channel states, where-as those after it are virtual state values returned on the GetStatus calls only.

##### See also

[CyU3PDmaChannelGetStatus](#)

##### Enumerator

**CY\_U3P\_DMA\_NOT\_CONFIGURED** DMA channel is unconfigured. This state occurs only when using a stale/uninitialized channel structure. This channel state is set to this when a channel is destroyed.

**CY\_U3P\_DMA\_CONFIGURED** DMA channel has been configured successfully. The channel reaches this state through the following conditions:

1. Channel is successfully created.
2. Channel is reset.
3. A finite transfer has been successfully completed. A GetStatus call in this case will return a virtual CY\_U3P\_DMA\_XFER\_COMPLETED state.
4. One of the override modes have completed successfully. A GetStatus call in this case will return a virtual CY\_U3P\_DMA\_SEND\_COMPLETED or CY\_U3P\_DMA\_RECV\_COMPLETED state.

**CY\_U3P\_DMA\_ACTIVE** Channel has active transaction going on. This state is reached when a SetXfer call is invoked and the transfer is on-going.

**CY\_U3P\_DMA\_PROD\_OVERRIDE** The channel is working in producer socket override mode. This state is reached when a SetupSend call is invoked and the transfer is on-going.

**CY\_U3P\_DMA\_CONS\_OVERRIDE** Channel is working in consumer socket override mode. This state is reached when a SetupRecv call is invoked and the transfer is on-going.

**CY\_U3P\_DMA\_ERROR** Channel has encountered an error. This state is reached when a DMA hardware error is detected.

**CY\_U3P\_DMA\_IN\_COMPLETION** Waiting for all data to drain out. This state is reached when transfer has completed from the producer side, but waiting for the consumer to drain all data.

**CY\_U3P\_DMA\_ABORTED** The channel is in aborted state. This state is reached when a Abort call is made.

**CY\_U3P\_DMA\_NUM\_STATES** Number of states. This is not a valid state and is just a place holder.

**CY\_U3P\_DMA\_XFER\_COMPLETED** This is a virtual state returned by GetStatus function. The actual state is CY\_U3P\_DMA\_CONFIGURED. This is just the value returned on GetStatus call after completion of finite transfer.

**CY\_U3P\_DMA\_SEND\_COMPLETED** This is a virtual state returned by GetStatus function. The actual state is CY\_U3P\_DMA\_CONFIGURED. This is just the value returned on GetStatus call after completion of producer override mode.

**CY\_U3P\_DMA\_RECV\_COMPLETED** This is a virtual state returned by GetStatus function. The actual state is CY\_U3P\_DMA\_CONFIGURED. This is just the value returned on GetStatus call after completion of consumer override mode.

#### 5.17.3.7 enum CyU3PDmaType\_t

List of DMA single channel types.

##### Description

A DMA channel aggregates all of the resources required to manage a unique data flow through the FX3 device. A DMA channel needs to be created before any DMA operation can be performed.

In the normal case, each DMA channel has one producer socket through which data is received on the FX3 device; and one consumer socket through which data is sent out of the FX3 device. The DMA channels can be classified into multiple types based on the amount of firmware intervention and control that is required in the data flow.

This enumeration lists the different types of DMA single (one to one) channels. All APIs that operate on these channels have a [CyU3PDmaChannel](#) prefix.

##### See also

[CyU3PDmaChannelCreate](#)

##### Enumerator

**CY\_U3P\_DMA\_TYPE\_AUTO** Auto mode DMA channel.

**CY\_U3P\_DMA\_TYPE\_AUTO\_SIGNAL** Auto mode with event signalling.

**CY\_U3P\_DMA\_TYPE\_MANUAL** Manual mode DMA channel.

**CY\_U3P\_DMA\_TYPE\_MANUAL\_IN** Manual mode producer socket to CPU.

**CY\_U3P\_DMA\_TYPE\_MANUAL\_OUT** Manual mode CPU to consumer socket.

**CY\_U3P\_DMA\_NUM\_SINGLE\_TYPES** Number of single DMA channel types.

#### 5.17.4 Function Documentation

##### 5.17.4.1 CyU3PReturnStatus\_t CyU3PDmaChannelAbort ( CyU3PDmaChannel \* handle )

Aborts a DMA channel.

##### Description

The function shall abort both the producer and consumer sockets associated with the channel. The data in transition is lost and any active transaction cannot be resumed. This function leaves the channel in an aborted state and requires a reset before the channel can be used again.

##### Return value

CY\_U3P\_SUCCESS - if the function call is successful.

CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.

CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.

CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

**See also**

[CyU3PDmaChannel](#)  
[CyU3PDmaChannelSetXfer](#)  
[CyU3PDmaChannelReset](#)

**Parameters**

<i>handle</i>	Handle to the channel to be aborted.
---------------	--------------------------------------

#### 5.17.4.2 CyU3PReturnStatus\_t CyU3PDmaChannelCacheControl ( CyU3PDmaChannel \* *handle*, CyBool\_t *isDmaHandleDCache* )

This function is used to enable/disable the Data cache coherency handling on a per DMA channel basis.

**Description**

The DMA driver in the FX3 library assumes that any manual data transfer on a DMA channel can be affected by the Data cache. Therefore, it ensures that the data buffer region is evicted from the cache before reading any data from a newly produced buffer. It also ensures that the data buffer region is written back to memory before committing the buffer to the consumer.

These cache operations can limit the transfer performance obtained on the DMA channel in some cases. If the caller can ensure that the contents of the data buffer are accessed by the firmware only in very rare cases, it is possible to get better performance by removing these cache operations.

This API is used to selectively turn off the data cache handling on a per DMA channel basis. This should only be used with caution, and the caller should ensure that the cache APIs are used directly where required.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.  
 CY\_U3P\_ERROR\_INVALID\_SEQUENCE - if the DMA channel is not idle (configured state).

**See also**

[CyU3PDmaChannel](#)  
[CyU3PDeviceCacheControl](#)  
[CyU3PDmaChannelCreate](#)

**Parameters**

<i>handle</i>	Handle to the DMA channel.
<i>isDmaHandleDCache</i>	Whether to enable handling or not.

#### 5.17.4.3 CyU3PReturnStatus\_t CyU3PDmaChannelCommitBuffer ( CyU3PDmaChannel \* *handle*, uint16\_t *count*, uint16\_t *bufStatus* )

Commit a data buffer to be sent to the consumer.

**Description**

This function is generally used with manual DMA channels, and allows the user to commit a data buffer which is to be sent out to the consumer. This operation is not valid for channels of type MANUAL\_IN.

The count provided is the exact size of the data that is to be sent out of the consumer socket.

This API can be used with AUTO DMA channels in the special case where the consumer socket is suspended. This allows the user to modify the data content of a partial data buffer, and then commit it for sending out of the device.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.  
CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the count is invalid.  
CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
CY\_U3P\_ERROR\_NOT\_SUPPORTED - if this operation is not supported for the DMA channel type.  
CY\_U3P\_ERROR\_NOT\_STARTED - if the DMA channel is not started.  
CY\_U3P\_ERROR\_INVALID\_SEQUENCE - if this sequence is not permitted.  
CY\_U3P\_ERROR\_DMA\_FAILURE - if the DMA transfer failed.  
CY\_U3P\_ERROR\_ABORTED - if the DMA transfer was aborted.  
CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

**See also**

[CyU3PDmaChannelGetBuffer](#)  
[CyU3PDmaChannelDiscardBuffer](#)

**Parameters**

<i>handle</i>	Handle to the DMA channel to be modified.
<i>count</i>	Size of data in the buffer being committed. The buffer address is implicit and is fetched from the active descriptor for the channel.
<i>bufStatus</i>	Current status (end of transfer bit) of the buffer being committed. The occupied bit will automatically be set by the API.

[CyU3PDmaChannelSetupRecvBuffer](#)  
[CyU3PDmaChannelWaitForCompletion](#)  
[CyU3PDmaChannelWaitForRecvBuffer](#)  
[CyU3PDmaChannelSetWrapUp](#)  
[CyU3PDmaChannelSetSuspend](#)  
[CyU3PDmaChannelResume](#)  
[CyU3PDmaChannelAbort](#)  
[CyU3PDmaChannelReset](#)  
[CyU3PDmaChannelCacheControl](#)

**Parameters**

<i>handle</i>	Pointer to channel structure that should be initialized.
<i>type</i>	Type of DMA channel desired.
<i>config</i>	Channel configuration parameters.

**5.17.4.5 CyU3PReturnStatus\_t CyU3PDmaChannelDestroy ( CyU3PDmaChannel \* *handle* )**

Destroy a one-to-one socket DMA channel.

**Description**

This function frees up a one-to-one socket DMA channel once it is no longer required. This should not be called from a DMA callback.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel structure is not valid.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

**See also**

[CyU3PDmaChannel](#)  
[CyU3PDmaChannelCreate](#)

**Parameters**

<i>handle</i>	Pointer to DMA channel structure to be de-initialized.
---------------	--

**5.17.4.6 CyU3PReturnStatus\_t CyU3PDmaChannelDiscardBuffer ( CyU3PDmaChannel \* *handle* )**

Drop a data buffer without sending out to the consumer.

**Description**

This function drops the content of the active buffer by moving the consumer socket ahead to the next buffer. This function is generally used with DMA channels of type MANUAL and MANUAL\_IN.

In the case of AUTO DMA channels, this API is used in the special case where the consumer socket is suspended using the CY\_U3P\_DMA\_SCK\_SUSP\_CONS\_PARTIAL\_BUF option. This allows the user to drop the content of a partially filled buffer without sending it out to the consumer.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.

CY\_U3P\_ERROR\_NOT\_SUPPORTED - if this operation is not supported for the DMA channel type.  
 CY\_U3P\_ERROR\_NOT\_STARTED - if the DMA channel is not started.  
 CY\_U3P\_ERROR\_INVALID\_SEQUENCE - if this sequence is not permitted.  
 CY\_U3P\_ERROR\_DMA\_FAILURE - if the DMA transfer failed.  
 CY\_U3P\_ERROR\_ABORTED - if the DMA transfer was aborted.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

#### See also

[CyU3PDmaChannel](#)  
[CyU3PDmaChannelGetBuffer](#)  
[CyU3PDmaChannelCommitBuffer](#)

#### Parameters

<i>handle</i>	Handle to the DMA channel to be modified.
---------------	---

#### 5.17.4.7 CyU3PReturnStatus\_t CyU3PDmaChannelGetBuffer ( CyU3PDmaChannel \* *handle*, CyU3PDmaBuffer\_t \* *buffer\_p*, uint32\_t *waitOption* )

Get the current buffer pointer.

#### Description

This function waits until a buffer is ready on the channel, and then returns a pointer to the buffer status.

The ready condition is defined differently for different DMA channel types:

1. For MANUAL and MANUAL\_IN DMA channels, a ready buffer is one which has been filled with data by the producer.
2. For MANUAL\_OUT channels, a ready buffer is an empty buffer that can be filled by the firmware.
3. For AUTO channels, this API can only be called when the consumer socket is suspended. This mechanism is supported to facilitate reading the buffer status when the channel has been suspended using the CY\_U3P\_DMA\_SCK\_SUSP\_CONS\_PARTIAL\_BUF option.

If this function is called from a DMA callback, the wait option should be set to CYU3P\_NO\_WAIT.

#### Return value

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the buffer parameters are invalid.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - if this operation is not supported for the DMA channel type.  
 CY\_U3P\_ERROR\_NOT\_STARTED - if the DMA channel is not started.  
 CY\_U3P\_ERROR\_INVALID\_SEQUENCE - if this sequence is not permitted.  
 CY\_U3P\_ERROR\_TIMEOUT - if the DMA transfer timed out.  
 CY\_U3P\_ERROR\_DMA\_FAILURE - if the DMA transfer failed.  
 CY\_U3P\_ERROR\_ABORTED - if the DMA transfer was aborted.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

See also

[CyU3PDmaBuffer\\_t](#)  
[CyU3PDmaChannel](#)  
[CyU3PDmaChannelSetXfer](#)  
[CyU3PDmaChannelCommitBuffer](#)  
[CyU3PDmaChannelDiscardBuffer](#)

Parameters

<i>handle</i>	Handle to the DMA channel on which to wait.
<i>buffer_p</i>	Output parameter that will be filled with data about the buffer that was obtained.
<i>waitOption</i>	Duration to wait before returning a timeout status.

#### 5.17.4.8 CyU3PDmaChannel\* CyU3PDmaChannelGetHandle ( CyU3PDmaSocketId\_t sckId )

Fetch a handle to the DMA channel corresponding to the specified socket.

**Description**

This function is used to identify if there is any DMA channel associated with a socket.

**Return value**

Handle of the channel associated with the specified socket. A NULL return indicates that the socket has not been associated with any DMA channel.

See also

[CyU3PDmaChannel](#)  
[CyU3PDmaSocketId\\_t](#)  
[CyU3PDmaChannelCreate](#)  
[CyU3PDmaChannelDestroy](#)

Parameters

<i>sck<sub>←</sub> Id</i>	ID of the socket whose channel handle is required.
-------------------------------	--

#### 5.17.4.9 CyU3PReturnStatus\_t CyU3PDmaChannelGetStatus ( CyU3PDmaChannel \* handle, CyU3PDmaState\_t \* state, uint32\_t \* prodXferCount, uint32\_t \* consXferCount )

This function returns the current channel status.

**Description**

This function returns the current state of the DMA channel as well as the current transfer counts on both producer and consumer sockets.

**Note**

The FX3 device only updates the transfer count values at buffer boundaries, and it is not possible to identify the transfer count at a partial buffer level.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.  
CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

## See also

[CyU3PDmaState\\_t](#)  
[CyU3PDmaChannelSetXfer](#)

## Parameters

<i>handle</i>	Handle to the DMA channel to query.
<i>state</i>	Output parameter that will be filled with state of the channel.
<i>prodXferCount</i>	Output parameter that will be filled with transfer count on the producer socket in DMA mode units. Will return zero in the case of MANUAL_OUT channels.
<i>consXferCount</i>	Output parameter that will be filled with transfer count on the consumer socket in DMA mode units. Will return zero in the case of MANUAL_IN channels.

5.17.4.10 CyBool\_t CyU3PDmaChannelsValid ( CyU3PDmaChannel \* *handle* )

Check whether a given DMA channel handle is valid.

**Description**

This function checks whether a DMA channel handle points to a valid DMA channel.

**Return Value**

CyTrue if the channel handle is valid.

CyFalse if the channel handle is not valid.

5.17.4.11 CyU3PReturnStatus\_t CyU3PDmaChannelReset ( CyU3PDmaChannel \* *handle* )

Aborts and resets a DMA channel.

**Description**

The function aborts the ongoing transfer on a DMA channel, and restores all sockets and descriptors to their initial state. At the end of the Reset call, all resources associated with the channel are back in the state that they are in immediately after channel creation.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.

CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.

CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.

CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

## See also

[CyU3PDmaChannel](#)  
[CyU3PDmaChannelCreate](#)  
[CyU3PDmaChannelAbort](#)

## Parameters

<i>handle</i>	Handle to the channel to be reset.
---------------	------------------------------------

5.17.4.12 CyU3PReturnStatus\_t CyU3PDmaChannelResume ( CyU3PDmaChannel \* *handle*, CyBool\_t *isProdResume*, CyBool\_t *isConsResume* )

Resume a suspended DMA channel.

**Description**

The function can be called to resume a suspended DMA channel. It can only be called when the channel was suspended from an active state. The producer and consumer suspend conditions can be individually cleared.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the channel type is invalid.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_NOT\_STARTED - if the DMA channel was not started.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

**See also**

[CyU3PDmaChannelSetSuspend](#)

**Parameters**

<i>handle</i>	Handle to the channel to be resumed.
<i>isProdResume</i>	Whether to resume the producer socket.
<i>isConsResume</i>	Whether to resume the consumer socket.

**5.17.4.13 CyU3PReturnStatus\_t CyU3PDmaChannelResumeUsbConsumer ( CyU3PDmaChannel \* handle )**

Resume the USB IN (egress) socket corresponding to the DMA channel.

**Description**

This function resumes the USB IN socket corresponding to the DMA channel. This will resume the socket only if it was suspended through the CyU3PDmaChannelSuspendUsbConsumer function.

**Return Value**

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the channel handle is not valid.  
 CY\_U3P\_SUCCESS if the socket was resumed as requested.

**See also**

[CyU3PDmaChannelSuspendUsbConsumer](#)

**Parameters**

<i>handle</i>	Handle to channel to be resumed.
---------------	----------------------------------

**5.17.4.14 CyU3PReturnStatus\_t CyU3PDmaChannelSetSuspend ( CyU3PDmaChannel \* handle, CyU3PDmaSckSuspType\_t prodSusp, CyU3PDmaSckSuspType\_t consSusp )**

Set the suspend options for the sockets associated with a DMA channel.

**Description**

The function sets the suspend options for the sockets. The sockets are by default set to SUSP\_NONE option. The API can be called only when the channel is in configured state or in active state. The suspend options are applied only in the active state (SetXfer mode) and is a don't care in the override mode of operation.

For manual channels, suspending the channel is largely not required as each buffer needs to be manually committed. However, these options are still available for manual DMA channels.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the channel type/suspend options are invalid.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_INVALID\_SEQUENCE - if the DMA channel is not in the required state.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

#### See also

[CyU3PDmaChannel](#)  
[CyU3PDmaChannelResume](#)  
[CyU3PDmaChannelReset](#)

#### Parameters

<i>handle</i>	Handle to the channel to be modified.
<i>prodSusp</i>	Suspend option for the producer socket.
<i>consSusp</i>	Suspend option for the consumer socket.

5.17.4.15 CyU3PReturnStatus\_t CyU3PDmaChannelSetupRecvBuffer ( CyU3PDmaChannel \* *handle*,  
 CyU3PDmaBuffer\_t \* *buffer\_p* )

Receive data into a user provided DMA buffer.

#### Description

This function initiates a read of incoming data from a DMA producer into a user provided buffer. This operation is performed in override mode of the DMA channel, and can only be initiated when the channel is in configured state.

The buffer that is passed as parameter for receiving the data has the following restrictions:

1. The buffer size should be a multiple of 16 bytes.
2. If the data cache is enabled then, the buffer should be 32 byte aligned and a multiple of 32 bytes. This is to match the 32 byte cache line. 32 byte check is not enforced by the API as the buffer can be over-allocated.

#### Return value

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the buffer parameters are invalid.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - if this operation is not supported for the DMA channel type.  
 CY\_U3P\_ERROR\_ALREADY\_STARTED - if the DMA channel is already started.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

#### See also

[CyU3PDmaBuffer\\_t](#)  
[CyU3PDmaChannel](#)  
[CyU3PDmaChannelSetupSendBuffer](#)  
[CyU3PDmaChannelWaitForRecvBuffer](#)

#### Parameters

<i>handle</i>	Handle to the DMA channel to be modified.
---------------	---

**Parameters**

<i>buffer</i> ← — <i>p</i>	Pointer to structure containing the address and size of the buffer to be filled up with received data.
-------------------------------	--

5.17.4.16 `CyU3PReturnStatus_t CyU3PDmaChannelSetupSendBuffer ( CyU3PDmaChannel * handle,  
CyU3PDmaBuffer_t * buffer_p )`

Send the contents of a user provided buffer to the consumer.

**Description**

This function initiates the sending of the content of a user provided buffer to the consumer of a DMA channel. This function is an override on the normal behavior of the DMA channel and can only be called when the channel is in the configured state.

The buffers used for DMA operations are expected to be allocated using the `CyU3PDmaBufferAlloc` call. If this is not the case, then the buffer has to be over allocated in such a way that the full buffer should be 32 byte aligned and should be a multiple of 32 bytes in size.

**Return value**

`CY_U3P_SUCCESS` - if the function call is successful.  
`CY_U3P_ERROR_NULL_POINTER` - if any pointer passed as parameter is NULL.  
`CY_U3P_ERROR_BAD_ARGUMENT` - if the buffer parameters are invalid.  
`CY_U3P_ERROR_NOT_CONFIGURED` - if the DMA channel was not configured.  
`CY_U3P_ERROR_NOT_SUPPORTED` - if this operation is not supported for the DMA channel type.  
`CY_U3P_ERROR_ALREADY_STARTED` - if the DMA channel is already started.  
`CY_U3P_ERROR_MUTEX_FAILURE` - if the DMA channel mutex could not be acquired.

**See also**

[CyU3PDmaChannel](#)  
[CyU3PDmaBuffer\\_t](#)  
[CyU3PDmaChannelCreate](#)  
[CyU3PDmaChannelReset](#)  
[CyU3PDmaChannelSetupRecvBuffer](#)

**Parameters**

<i>handle</i>	Handle to the DMA channel to be modified.
<i>buffer</i> ← — <i>p</i>	Pointer to structure containing address, size and status of the DMA buffer to be sent out.

5.17.4.17 `CyU3PReturnStatus_t CyU3PDmaChannelSetWrapUp ( CyU3PDmaChannel * handle )`

Wraps up the current active buffer for the channel from the producer side.

**Description**

Data received by a DMA producer socket is only committed (made available to the consumer) when the buffer is completely filled up, or when the producer signals and End Of Packet (EOP) condition. There may be cases where these conditions are not met, and a partially filled buffer is blocked waiting for more data.

This function can be used to forcibly commit the contents of the active DMA buffer on the producer side of the channel. A DMA produce event will be generated as a result of this API call, and the producer socket will move on to the next buffer in the chain without getting suspended.

The function cannot be used with `MANUAL_OUT` channels, and can only be called when the channel is the active or consumer override state.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - if this operation is not supported for the DMA channel type.  
 CY\_U3P\_ERROR\_INVALID\_SEQUENCE - if the DMA channel is not in the required state.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

**See also**

[CyU3PDmaState\\_t](#)  
[CyU3PDmaChannelGetStatus](#)  
[CyU3PDmaChannelSetXfer](#)  
[CyU3PDmaChannelGetBuffer](#)  
[CyU3PDmaChannelCommitBuffer](#)

**Parameters**

<i>handle</i>	Handle to the channel to be modified.
---------------	---------------------------------------

**5.17.4.18 CyU3PReturnStatus\_t CyU3PDmaChannelSetXfer ( CyU3PDmaChannel \* *handle*, uint32\_t *count* )**

Setup a one-to-one DMA channel for data transfer.

**Description**

The sockets corresponding to a DMA channel are left disabled when the channel is created, so that transfers are started only when desired by the user. This function enables a DMA channel to transfer a specified amount of data before suspending again. An infinite transfer can be started by specifying a transfer size of 0.

This function should be called only when the channel is in the CY\_U3P\_DMA\_CONFIGURED state.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - if no buffers were allocated for this DMA channel.  
 CY\_U3P\_ERROR\_ALREADY\_STARTED - if the channel is already active.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

**See also**

[CyU3PDmaChannel](#)  
[CyU3PDmaChannelCreate](#)  
[CyU3PDmaChannelGetStatus](#)  
[CyU3PDmaChannelSetupSendBuffer](#)  
[CyU3PDmaChannelSetupRecvBuffer](#)  
[CyU3PDmaChannelWaitForCompletion](#)  
[CyU3PDmaChannelReset](#)

**Parameters**

<i>handle</i>	Handle to the channel to be modified.
<i>count</i>	The desired transaction size in units corresponding to the selected DMA mode. Channel will revert to idle state when the specified amount of data has been transferred. Can be set to zero to request an infinite data transfer.

5.17.4.19 `CyU3PReturnStatus_t CyU3PDmaChannelSuspendUsbConsumer ( CyU3PDmaChannel * handle, uint32_t waitOption )`

Suspend the USB IN (egress) socket corresponding to the DMA channel.

#### Description

This function forcibly suspends the USB IN (egress) socket corresponding to the DMA channel. This function is intended for API internal usage (implementation if specific USB defect work-arounds) and is not expected to be called by user application code.

#### Return Value

`CY_U3P_ERROR_BAD_ARGUMENT` if the channel handle is not valid.

`CY_U3P_SUCCESS` if the socket was suspended as requested.

`CY_U3P_ERROR_TIMEOUT` if the socket suspend operation timed out.

#### See also

[CyU3PDmaChannelResumeUsbConsumer](#)

#### Parameters

<code>handle</code>	Handle to channel to be suspended.
<code>waitOption</code>	Amount of time to wait for channel suspension.

5.17.4.20 `CyU3PReturnStatus_t CyU3PDmaChannelUpdateMode ( CyU3PDmaChannel * handle, CyU3PDmaMode_t dmaMode )`

Update the DMA mode for a one-to-one DMA channel.

#### Description

The DMA mode of a channel decides whether transfers on the channel are tracked in terms of bytes or buffers. The mode is normally specified at the time of channel creation and left unmodified there-after. This API can be used if the mode for an existing channel needs to be changed. This can only be called when the channel is in the configured (just created or reset) state.

#### Return value

`CY_U3P_SUCCESS` - if the function call is successful.

`CY_U3P_ERROR_NULL_POINTER` - if any pointer passed as parameter is NULL.

`CY_U3P_ERROR_BAD_ARGUMENT` - if DMA mode is invalid.

`CY_U3P_ERROR_INVALID_SEQUENCE` - if the DMA channel is not in the Configured state.

`CY_U3P_ERROR_MUTEX_FAILURE` - if the DMA channel mutex could not be acquired.

#### See also

[CyU3PDmaMode\\_t](#)  
[CyU3PDmaChannel](#)  
[CyU3PDmaChannelCreate](#)  
[CyU3PDmaChannelGetStatus](#)

#### Parameters

<code>handle</code>	Handle to DMA channel to be modified.
<code>dmaMode</code>	Desired DMA operating mode. Can be byte mode or buffer mode.

5.17.4.21 `CyU3PReturnStatus_t CyU3PDmaChannelWaitForCompletion ( CyU3PDmaChannel * handle, uint32_t waitOption )`

Wait for the current DMA transaction to complete.

#### Description

This function waits until the current transfer on the DMA channel is completed, or until the specified timeout period has elapsed. This function is not supported if the DMA channel has been configured for infinite transfers.

This function should not be called from a DMA callback function. If the function returns CY\_U3P\_ERROR\_TIME\_OUT, the wait API call can be repeated without affecting the data transfer.

#### Return value

`CY_U3P_SUCCESS` - if the function call is successful.  
`CY_U3P_ERROR_NULL_POINTER` - if any pointer passed as parameter is NULL.  
`CY_U3P_ERROR_NOT_CONFIGURED` - if the DMA channel was not configured.  
`CY_U3P_ERROR_NOT_SUPPORTED` - if this operation is not supported for the DMA channel type.  
`CY_U3P_ERROR_NOT_STARTED` - if the DMA channel is not started.  
`CY_U3P_ERROR_DMA_FAILURE` - if the DMA transfer failed.  
`CY_U3P_ERROR_ABORTED` - if the DMA transfer was aborted.  
`CY_U3P_ERROR_TIMEOUT` - if the DMA transfer timed out.  
`CY_U3P_ERROR_MUTEX_FAILURE` - if the DMA channel mutex could not be acquired.

#### See also

[CyU3PDmaChannelSetXfer](#)  
[CyU3PDmaChannelSetupSendBuffer](#)  
[CyU3PDmaChannelSetupRecvBuffer](#)  
[CyU3PDmaChannelWaitForRecvBuffer](#)

#### Parameters

<code>handle</code>	Handle to the DMA channel to wait on.
<code>waitOption</code>	Duration for which to wait.

5.17.4.22 `CyU3PReturnStatus_t CyU3PDmaChannelWaitForRecvBuffer ( CyU3PDmaChannel * handle, CyU3PDmaBuffer_t * buffer_p, uint32_t waitOption )`

Wait until an override read operation is completed.

#### Description

This function waits until the read operation initiated through the `CyU3PDmaChannelSetupRecvBuffer` API is completed. Once the transfer is completed, it returns the status of the buffer that was filled with data. If not, a timeout error is returned. It is possible to retry this wait API without resetting or aborting the channel.

This function must not be called from a DMA callback function.

#### Return value

`CY_U3P_SUCCESS` - if the function call is successful.  
`CY_U3P_ERROR_NULL_POINTER` - if any pointer passed as parameter is NULL.  
`CY_U3P_ERROR_BAD_ARGUMENT` - if the buffer parameters are invalid.  
`CY_U3P_ERROR_NOT_CONFIGURED` - if the DMA channel was not configured.  
`CY_U3P_ERROR_NOT_SUPPORTED` - if this operation is not supported for the DMA channel type.  
`CY_U3P_ERROR_INVALID_SEQUENCE` - if this sequence is not permitted.  
`CY_U3P_ERROR_TIMEOUT` - if the DMA transfer timed out.  
`CY_U3P_ERROR_DMA_FAILURE` - if the DMA transfer failed.  
`CY_U3P_ERROR_ABORTED` - if the DMA transfer was aborted.  
`CY_U3P_ERROR_MUTEX_FAILURE` - if the DMA channel mutex could not be acquired.  
`CY_U3P_ERROR_NOT_STARTED` - if the DMA channel is not started.

See also

[CyU3PDmaChannel](#)  
[CyU3PDmaChannelSetupRecvBuffer](#)  
[CyU3PDmaChannelWaitForCompletion](#)

#### Parameters

<i>handle</i>	Handle to the DMA channel on which to wait.
<i>buffer_p</i>	Output parameter which will be filled up with the address, count and status values of the DMA buffer into which data was received.
<i>waitOption</i>	Duration to wait for the receive completion.

#### 5.17.4.23 void CyU3PDmaEnableMulticast( void )

Enable creation and management of DMA multicast channels.

#### Description

It is expected that DMA multicast channels will only rarely be used in FX3 applications. Since the multichannel creation code takes in the channel type as a parameter and then calls the appropriate handler functions, the code to setup and work with multicast channels gets linked into any FX3 application that uses multichannels, leading to un-necessary loss of code space. This function is provided to prevent this memory loss.

The multicast channel code will only be linked into the FX3 application if this function has been called. Therefore, this function needs to be called by the application before any multicast channels are created.

#### Return value

None

See also

[CyU3PDmaMultiChannelCreate](#)

#### 5.17.4.24 CyU3PReturnStatus\_t CyU3PDmaMulticastDisableConsumer( CyU3PDmaMultiChannel \* chHandle, uint16\_t consIndex )

Disable a consumer on a multi-cast socket while transfer is in flight.

#### Description

The CyU3PDmaMulticastSocketSelect API allows user to the select the consumer sockets that should be enabled for a data transfer. However, this API can only be used while the channel is idle, and cannot be used to modify a transfer that is in flight.

This API allows the user to disable one of the consumer sockets for a multicast transfer while the transfer is in flight (CyU3PDmaMultiChannelSetXfer has been called). Please note that the changes applied by this API are only valid for the ongoing transfer, and you will need to do a CyU3PDmaMultiChannelReset before starting the next transfer using the CyU3PDmaMultiChannelSetXfer API.

#### Return Value

CY\_U3P\_SUCCESS if the socket disable is successful. CY\_U3P\_ERROR\_NOT\_SUPPORTED if the channel specified is not a multicast channel. CY\_U3P\_ERROR\_INVALID\_SEQUENCE if the channel is not in active transfer mode. CY\_U3P\_ERROR\_BAD\_ARGUMENT if socket specified is out of range.

#### 5.17.4.25 CyU3PReturnStatus\_t CyU3PDmaMulticastSocketSelect( CyU3PDmaMultiChannel \* chHandle, uint32\_t consMask )

Select the active consumers on a multicast DMA channel.

**Description**

A multicast DMA channel has two or more consumer sockets, all of which receive data sent obtained through the producer socket. There may be cases where the firmware application needs to dynamically change the list of active consumers (that will receive the data). This API is used to select the active consumers on a multicast DMA channel.

The consMask parameter is a bitmask which represents the active consumers. Bit n of this bitmask needs to be set if consumer n is to be activated. By default, all consumers on the channel are left active. This API can only be used when the DMA channel is the idle (configured) state.

**Return value**

CY\_U3P\_SUCCESS if the socket selection is successful. CY\_U3P\_ERROR\_NOT\_SUPPORTED if the channel specified is not a multicast channel. CY\_U3P\_ERROR\_ALREADY\_STARTED if the channel is already active (SetXfer called).

#### 5.17.4.26 CyU3PReturnStatus\_t CyU3PDmaMultiChannelAbort ( CyU3PDmaMultiChannel \* handle )

Abort the ongoing transfer on a DMA multi-channel.

**Description**

The function shall abort all the producer and consumer sockets associated with the channel. The data in transition is lost and any active transaction cannot be resumed. This function leaves the channel in an aborted state and requires a reset before the channel can be used again.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

**See also**

[CyU3PDmaMultiChannel](#)  
[CyU3PDmaMultiChannelSetXfer](#)  
[CyU3PDmaMultiChannelReset](#)  
[CyU3PDmaMultiChannelGetStatus](#)

**Parameters**

<i>handle</i>	Handle to the multi-channel to be aborted.
---------------	--

#### 5.17.4.27 CyU3PReturnStatus\_t CyU3PDmaMultiChannelCacheControl ( CyU3PDmaMultiChannel \* handle, CyBool\_t isDmaHandleDCache )

This function is used to enable/disable the Data cache coherency handling on a per DMA channel basis.

**Description**

The DMA driver in the FX3 library assumes that any manual data transfer on a DMA channel can be affected by the Data cache. Therefore, it ensures that the data buffer region is evicted from the cache before reading any data from a newly produced buffer. It also ensures that the data buffer region is written back to memory before committing the buffer to the consumer.

These cache operations can limit the transfer performance obtained on the DMA channel in some cases. If the caller can ensure that the contents of the data buffer are accessed by the firmware only in very rare cases, it is possible to get better performance by removing these cache operations.

This API is used to selectively turn off the data cache handling on a per DMA channel basis. This should only be used with caution, and the caller should ensure that the cache APIs are used directly where required.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.

CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.  
 CY\_U3P\_ERROR\_INVALID\_SEQUENCE - if the DMA channel is not idle (configured state).

#### See also

[CyU3PDmaMultiChannel](#) [CyU3PDeviceCacheControl](#)

#### Parameters

<i>handle</i>	Handle to the DMA channel.
<i>isDmaHandleDCache</i>	Whether to enable handling or not.

#### 5.17.4.28 CyU3PReturnStatus\_t CyU3PDmaMultiChannelCommitBuffer ( CyU3PDmaMultiChannel \* *handle*, uint16\_t *count*, uint16\_t *bufStatus* )

Commit the buffer to be sent to the consumer.

#### Description

This function is generally used with manual DMA channels, and allows the user to commit a data buffer which is to be sent out to the consumer. In the case of one-to-many channels, the buffer will be sent out through the consumer channels on a round-robin basis.

The count provided is the exact size of the data that is to be sent out of the consumer socket.

This API can be used with many-to-one AUTO DMA channels in the special case where the consumer socket is suspended. This allows the user to modify the data content of a partial data buffer, and then commit it for sending out of the device.

#### Return value

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - if this operation is not supported for the DMA channel type.  
 CY\_U3P\_ERROR\_NOT\_STARTED - if the DMA channel is not started.  
 CY\_U3P\_ERROR\_INVALID\_SEQUENCE - if the DMA channel is not in the required state.  
 CY\_U3P\_ERROR\_DMA\_FAILURE - if the DMA transfer failed.  
 CY\_U3P\_ERROR\_ABORTED - if the DMA transfer was aborted.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

#### See also

[CyU3PDmaMultiChannel](#)  
[CyU3PDmaMultiChannelSetXfer](#)  
[CyU3PDmaMultiChannelGetBuffer](#)  
[CyU3PDmaMultiChannelDiscardBuffer](#)

#### Parameters

<i>handle</i>	Handle to the multi-channel to be modified.
<i>count</i>	Size of the memory buffer being committed. The address of the buffer is implicit and is taken from the active descriptor.
<i>bufStatus</i>	Status of the buffer being committed.

5.17.4.29 `CyU3PReturnStatus_t CyU3PDmaMultiChannelCreate ( CyU3PDmaMultiChannel * handle,  
CyU3PDmaMultiType_t type, CyU3PDmaMultiChannelConfig_t * config )`

Create a multi-socket DMA channel with the specified parameters.

#### Description

This function is used to create a multi-socket DMA channel based on the specified parameters. The multi-channel structure is to be allocated by the caller, and a pointer to this is to be passed as parameter to this function.

This function must not be called from a DMA callback function.

#### Return value

`CY_U3P_SUCCESS` - if the function call is successful.

`CY_U3P_ERROR_NULL_POINTER` - if any pointer passed as parameter is NULL.

`CY_U3P_ERROR_BAD_ARGUMENT` - if any of the configuration parameters are invalid.

`CY_U3P_ERROR_MEMORY_ERROR` - if the memory required for the channel could not be allocated.

`CY_U3P_ERROR_INVALID_SEQUENCE` - if a multicast channel is being created without calling `CyU3PDmaEnableMulticast`.

#### See also

[CyU3PDmaMultiType\\_t](#)  
[CyU3PDmaMultiChannel](#)  
[CyU3PDmaMultiChannelConfig\\_t](#)  
[CyU3PDmaMultiChannelDestroy](#)  
[CyU3PDmaChannelCreate](#)

#### Parameters

<code>handle</code>	Pointer to multi-channel structure that is to be initialized.
<code>type</code>	Type of DMA channel to be created.
<code>config</code>	Configuration information about the channel to be created.

5.17.4.30 `CyU3PReturnStatus_t CyU3PDmaMultiChannelDestroy ( CyU3PDmaMultiChannel * handle )`

Destroy a multi-socket DMA channel.

#### Description

This function is used to free up the resources used by a DMA multi-channel when it is no longer required. This function should not be called from a DMA callback function.

#### Return value

`CY_U3P_SUCCESS` - if the function call is successful.

`CY_U3P_ERROR_NULL_POINTER` - if any pointer passed as parameter is NULL.

`CY_U3P_ERROR_NOT_CONFIGURED` - if the DMA channel was not configured.

`CY_U3P_ERROR_MUTEX_FAILURE` - if the DMA channel mutex could not be acquired.

#### See also

[CyU3PDmaMultiChannel](#)  
[CyU3PDmaMultiChannelCreate](#)  
[CyU3PDmaChannelDestroy](#)

5.17.4.31 `CyU3PReturnStatus_t CyU3PDmaMultiChannelDiscardBuffer ( CyU3PDmaMultiChannel * handle )`

Drop a data buffer without sending out to the consumer.

**Description**

This function drops the content of the active buffer by moving the consumer socket ahead to the next buffer. This function is generally used with DMA channels of type MANUAL and MANUAL\_IN.

In the case of many-to-one AUTO DMA channels, this API is used in the special case where the consumer socket is suspended using the CY\_U3P\_DMA\_SCK\_SUSP\_CONS\_PARTIAL\_BUF option. This allows the user to drop the content of a partially filled buffer without sending it out to the consumer.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.  
CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
CY\_U3P\_ERROR\_NOT\_SUPPORTED - if this operation is not supported for the DMA channel type.  
CY\_U3P\_ERROR\_NOT\_STARTED - if the DMA channel is not started.  
CY\_U3P\_ERROR\_INVALID\_SEQUENCE - if the DMA channel is not in the required state.  
CY\_U3P\_ERROR\_DMA\_FAILURE - if the DMA transfer failed.  
CY\_U3P\_ERROR\_ABORTED - if the DMA transfer was aborted.  
CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

**See also**

[CyU3PDmaMultiChannel](#)  
[CyU3PDmaMultiChannelSetXfer](#)  
[CyU3PDmaMultiChannelGetBuffer](#)  
[CyU3PDmaMultiChannelCommitBuffer](#)

**Parameters**

<i>handle</i>	Handle to the multi-channel to be modified.
---------------	---

### 5.17.4.32 CyU3PReturnStatus\_t CyU3PDmaMultiChannelGetBuffer ( CyU3PDmaMultiChannel \* handle, CyU3PDmaBuffer\_t \* buffer\_p, uint32\_t waitOption )

Get the current buffer pointer.

**Description**

This function waits until a buffer is ready on the channel, and then returns a pointer to the buffer status.

In the case of many-to-one DMA channels, the buffers are received from the producer sockets in a round-robin fashion. The first producer is selected through the multiSckOffset parameter passed to the CyU3PDmaMultiChannelSetXfer API.

The ready condition is defined differently for different DMA channel types:

1. For MANUAL and MANUAL\_IN DMA channels, a ready buffer is one which has been filled with data by the producer.
2. For MANUAL\_OUT channels, a ready buffer is an empty buffer that can be filled by the firmware.
3. For many-to-one AUTO channels, this API can only be called when the consumer socket is suspended. This mechanism is supported to facilitate reading the buffer status when the channel has been suspended using the CY\_U3P\_DMA\_SCK\_SUSP\_CONS\_PARTIAL\_BUF option.

If this function is called from a DMA callback, the wait option should be set to CYU3P\_NO\_WAIT.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.

CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - if this operation is not supported for the DMA channel type.  
 CY\_U3P\_ERROR\_NOT\_STARTED - if the DMA channel is not started.  
 CY\_U3P\_ERROR\_INVALID\_SEQUENCE - if the DMA channel is not in the required state.  
 CY\_U3P\_ERROR\_DMA\_FAILURE - if the DMA transfer failed.  
 CY\_U3P\_ERROR\_ABORTED - if the DMA transfer was aborted.  
 CY\_U3P\_ERROR\_TIMEOUT - if the DMA transfer timed out.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

#### See also

[CyU3PDmaBuffer\\_t](#)  
[CyU3PDmaMultiChannel](#)  
[CyU3PDmaMultiChannelDestroy](#)  
[CyU3PDmaMultiChannelSetXfer](#)  
[CyU3PDmaMultiChannelCommitBuffer](#)  
[CyU3PDmaMultiChannelDiscardBuffer](#)

#### Parameters

<i>handle</i>	Handle to the multi-channel to be modified.
<i>buffer_p</i>	Output parameter that will be filled with address, size and status of the buffer with received data.
<i>waitOption</i>	Duration for which to wait for data.

#### 5.17.4.33 CyU3PDmaMultiChannel\* CyU3PDmaMultiChannelGetHandle ( CyU3PDmaSocketId\_t sckId )

Identifies the multi-channel that is associated with the specified socket.

#### Description

This function returns a pointer to the multi-channel that is associated with the specified DMA socket. This function will return NULL if there are no channels associated with the specified socket.

#### Return value

Handle to the Multi-channel structure corresponding to the socket.

#### See also

[CyU3PDmaSocketId\\_t](#)  
[CyU3PDmaMultiChannel](#)

#### Parameters

<i>sck</i> ↪ <i>Id</i>	ID of the socket whose channel handle is required.
---------------------------	--

#### 5.17.4.34 CyU3PReturnStatus\_t CyU3PDmaMultiChannelGetStatus ( CyU3PDmaMultiChannel \* handle, CyU3PDmaState\_t \* state, uint32\_t \* prodXferCount, uint32\_t \* consXferCount, uint8\_t sckIndex )

This functions returns the current multi-channel status.

#### Description

This function returns the current state of the DMA channel as well as the current transfer counts on both producer and consumer sockets.

**Note**

The FX3 device only updates the transfer count values at buffer boundaries, and it is not possible to identify the transfer count at a partial buffer level.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the sckIndex is invalid.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

**See also**

[CyU3PDmaState\\_t](#)  
[CyU3PDmaMultiChannelSetXfer](#)

**Parameters**

<i>handle</i>	Handle to the DMA channel to query.
<i>state</i>	Output parameter that will be filled with state of the channel.
<i>prodXferCount</i>	Output parameter that will be filled with transfer count on the producer socket in DMA mode units.
<i>consXferCount</i>	Output parameter that will be filled with transfer count on the consumer socket in DMA mode units.
<i>sckIndex</i>	The socket index for retrieving information transfer counts.

**5.17.4.35 CyBool\_t CyU3PDmaMultiChannelsValid ( CyU3PDmaMultiChannel \* handle )**

Check whether a given DMA multi-channel handle is valid.

**Description**

This function checks whether a DMA multi-channel handle points to a valid DMA multi-channel.

**Return Value**

CyTrue if the channel handle is valid.  
 CyFalse if the channel handle is not valid.

**5.17.4.36 CyU3PReturnStatus\_t CyU3PDmaMultiChannelReset ( CyU3PDmaMultiChannel \* handle )**

Abort ongoing transfers on and resets a DMA multi-channel.

**Description**

The function aborts the ongoing transfer on a DMA channel, and restores all sockets and descriptors to their initial state. At the end of the Reset call, all resources associated with the channel are back in the state that they are in immediately after channel creation.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

**See also**

[CyU3PDmaMultiChannel](#)  
[CyU3PDmaMultiChannelSetXfer](#)  
[CyU3PDmaMultiChannelSetupSendBuffer](#)  
[CyU3PDmaMultiChannelSetupRecvBuffer](#)

[CyU3PDmaMultiChannelAbort](#)  
[CyU3PDmaMultiChannelGetStatus](#)

#### Parameters

<i>handle</i>	Handle to the multi-channel to be reset.
---------------	--

### 5.17.4.37 CyU3PReturnStatus\_t CyU3PDmaMultiChannelResume ( CyU3PDmaMultiChannel \* *handle*, CyBool\_t *isProdResume*, CyBool\_t *isConsResume* )

Resume a suspended DMA multi-channel.

#### Description

The function can be called to resume a suspended DMA channel. It can only be called when the channel was suspended from an active state.

For many-to-one channels, only the consumer side can be resumed; and for one-to-many channels, only the producer side can be resumed.

#### Return value

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the resume options are invalid.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

#### See also

[CyU3PDmaMultiChannel](#) [CyU3PDmaChannelResume](#) [CyU3PDmaMultiChannelSetSuspend](#)

#### Parameters

<i>handle</i>	Handle to the multi-channel to be resumed.
<i>isProdResume</i>	Whether to resume the producer socket.
<i>isConsResume</i>	Whether to resume the consumer socket.

### 5.17.4.38 CyU3PReturnStatus\_t CyU3PDmaMultiChannelResumeUsbConsumer ( CyU3PDmaMultiChannel \* *handle* )

Resume the USB IN (egress) socket corresponding to the DMA multi-channel.

#### Description

This function resumes the USB IN socket corresponding to the DMA multi-channel. This will resume the socket only if it was suspended through the CyU3PDmaMultiChannelSuspendUsbConsumer function.

#### Return Value

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the channel handle is not valid.  
 CY\_U3P\_SUCCESS if the socket was resumed as requested.

#### See also

[CyU3PDmaMultiChannelSuspendUsbConsumer](#)

#### Parameters

<i>handle</i>	Handle to channel to be resumed.
---------------	----------------------------------

5.17.4.39 `CyU3PReturnStatus_t CyU3PDmaMultiChannelSetSuspend ( CyU3PDmaMultiChannel * handle,  
CyU3PDmaSckSuspType_t prodSusp, CyU3PDmaSckSuspType_t consSusp )`

Update the suspend options for the sockets associated with a DMA multi-channel.

#### Description

The function sets the suspend options for the sockets. The sockets are by default set to SUSP\_NONE option. The API can be called only when the channel is in configured state or in active state. The suspend options are applied only in the active state (SetXfer mode) and is a don't care in the override mode of operation.

For manual channels, suspending the channel is largely not required as each buffer needs to be manually committed. However, these options are still available for manual DMA channels.

Suspend options can only be set for the end where there is a single socket. i.e., Suspend options can only be set for the producer side on one-to-many and multicast channels; and only for the consumer side on many-to-one channels.

#### Return value

`CY_U3P_SUCCESS` - if the function call is successful.  
`CY_U3P_ERROR_NULL_POINTER` - if any pointer passed as parameter is NULL.  
`CY_U3P_ERROR_BAD_ARGUMENT` - if the suspend options are invalid.  
`CY_U3P_ERROR_NOT_CONFIGURED` - if the DMA channel was not configured.  
`CY_U3P_ERROR_INVALID_SEQUENCE` - if the DMA channel is not in the required state.  
`CY_U3P_ERROR_MUTEX_FAILURE` - if the DMA channel mutex could not be acquired.

#### See also

[CyU3PDmaMultiChannel](#)  
[CyU3PDmaSckSuspType\\_t](#)  
[CyU3PDmaChannelSetSuspend](#)  
[CyU3PDmaMultiChannelResume](#)

#### Parameters

<code>handle</code>	Handle to the multi-channel to be suspended.
<code>prodSusp</code>	Suspend option for the producer socket.
<code>consSusp</code>	Suspend option for the consumer socket.

5.17.4.40 `CyU3PReturnStatus_t CyU3PDmaMultiChannelSetupRecvBuffer ( CyU3PDmaMultiChannel * handle,  
CyU3PDmaBuffer_t * buffer_p, uint16_t multiSckOffset )`

Receive data into a user provided DMA buffer.

#### Description

This function initiates a read of incoming data from a DMA producer into a user provided buffer. This operation is performed in override mode of the DMA channel, and can only be initiated when the channel is in configured state.

The buffer that is passed as parameter for receiving the data has the following restrictions:

1. The buffer size should be a multiple of 16 bytes.
2. If the data cache is enabled then, the buffer should be 32 byte aligned and a multiple of 32 bytes. This is to match the 32 byte cache line. 32 byte check is not enforced by the API as the buffer can be over-allocated.

#### Return value

`CY_U3P_SUCCESS` - if the function call is successful.

CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - if an invalid parameters are passed.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_ALREADY\_STARTED - if the DMA channel was already started.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

#### See also

[CyU3PDmaBuffer\\_t](#)  
[CyU3PDmaMultiChannel](#)  
[CyU3PDmaMultiChannelSetupSendBuffer](#)  
[CyU3PDmaMultiChannelWaitForRecvBuffer](#)

#### Parameters

<i>handle</i>	Handle to the DMA multi channel to be modified.
<i>buffer_p</i>	Pointer to structure containing the address and size of the buffer to be filled up with received data.
<i>multiSckOffset</i>	Producer Socket id to receive the data. For a one to many channel, this should be zero; and for a many to one channel, this would be a producer socket offset.

5.17.4.41 CyU3PReturnStatus\_t CyU3PDmaMultiChannelSetupSendBuffer ( *CyU3PDmaMultiChannel \* handle*,  
*CyU3PDmaBuffer\_t \* buffer\_p*, *uint16\_t multiSckOffset* )

Send the contents of a user provided buffer to the consumer.

#### Description

This function initiates the sending of the content of a user provided buffer to the consumer of a DMA channel. This function is an override on the normal behavior of the DMA channel and can only be called when the channel is in the configured state.

The buffers used for DMA operations are expected to be allocated using the CyU3PDmaBufferAlloc call. If this is not the case, then the buffer has to be over allocated in such a way that the full buffer should be 32 byte aligned and should be a multiple of 32 bytes in size.

#### Return value

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - if an invalid parameters are passed.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_ALREADY\_STARTED - if the DMA channel was already started.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

#### See also

[CyU3PDmaBuffer\\_t](#)  
[CyU3PDmaMultiChannel](#)  
[CyU3PDmaMultiChannelSetupRecvBuffer](#)

#### Parameters

<i>handle</i>	Handle to the DMA multi channel to be modified.
<i>buffer_p</i>	Pointer to structure containing address, size and status of the DMA buffer to be sent out.
<i>multiSckOffset</i>	Consumer Socket id to send the data. For a many to one channel, this should be zero; and for a one to many channel, this would be a consumer socket offset.

5.17.4.42 `CyU3PReturnStatus_t CyU3PDmaMultiChannelSetWrapUp ( CyU3PDmaMultiChannel * handle, uint16_t multiSckOffset )`

Wraps up the current active buffer on the producer socket.

#### Description

This API is used to forcibly commit a DMA buffer to the consumer, and is useful in the case where data transfer has abruptly stopped without the producer being able to commit the data buffer.

The function can be called only when the channel is in active mode or in consumer override mode.

#### Return value

`CY_U3P_SUCCESS` - if the function call is successful.  
`CY_U3P_ERROR_NULL_POINTER` - if any pointer passed as parameter is NULL.  
`CY_U3P_ERROR_BAD_ARGUMENT` - if the multiSckOffset is invalid.  
`CY_U3P_ERROR_NOT_CONFIGURED` - if the DMA channel was not configured.  
`CY_U3P_ERROR_INVALID_SEQUENCE` - if the DMA channel is not in the required state.  
`CY_U3P_ERROR_MUTEX_FAILURE` - if the DMA channel mutex could not be acquired.

#### See also

[CyU3PDmaChannelSetWrapUp](#)  
[CyU3PDmaMultiChannel](#)

#### Parameters

<code>handle</code>	Handle to the channel to be modified.
<code>multiSckOffset</code>	Socket id to wrapup. For a many to one channel, this would be a producer socket offset; and for a one to many channel, this would always be zero.

5.17.4.43 `CyU3PReturnStatus_t CyU3PDmaMultiChannelSetXfer ( CyU3PDmaMultiChannel * handle, uint32_t count, uint16_t multiSckOffset )`

Prepare a DMA multi-channel for data transfer.

#### Description

This function prepares the sockets involved in a DMA multi-channel for data transfer. This function can only be called when the channel is in the `CY_U3P_DMA_CONFIGURED` state. The amount of data to be transferred before the channel gets suspended is specified as a parameter, and can be set to 0 to start infinite transfers.

The `multiSckOffset` parameter specifies the index of the producer (in the case of many-to-one channels) or the consumer (one-to-many channels) that will transfer the first buffer. The rest of the sockets will be used in the sequence specified at channel creation.

#### Return value

`CY_U3P_SUCCESS` - if the function call is successful.  
`CY_U3P_ERROR_NULL_POINTER` - if any pointer passed as parameter is NULL.  
`CY_U3P_ERROR_BAD_ARGUMENT` - if the multiSckOffset is invalid.  
`CY_U3P_ERROR_NOT_CONFIGURED` - if the DMA channel was not configured.  
`CY_U3P_ERROR_ALREADY_STARTED` - if the DMA channel was already started.  
`CY_U3P_ERROR_MUTEX_FAILURE` - if the DMA channel mutex could not be acquired.

#### See also

[CyU3PDmaMultiChannel](#)  
[CyU3PDmaMultiChannelCreate](#)  
[CyU3PDmaMultiChannelSetupSendBuffer](#)  
[CyU3PDmaMultiChannelSetupRecvBuffer](#)  
[CyU3PDmaMultiChannelWaitForCompletion](#)

[CyU3PDmaMultiChannelReset](#)  
[CyU3PDmaMultiChannelGetStatus](#)

#### Parameters

<i>handle</i>	Handle to the multi-channel to be modified.
<i>count</i>	Size of the transfer to be started. Can be zero to denote infinite data transfer.
<i>multiSckOffset</i>	Socket id to start the operation from. For a many to one channel, this would be a producer socket offset; and for a one to many channel, this would be a consumer socket offset.

5.17.4.44 `CyU3PReturnStatus_t CyU3PDmaMultiChannelSuspendUsbConsumer ( CyU3PDmaMultiChannel * handle,  
 uint32_t waitOption )`

Suspend the USB IN (egress) socket corresponding to the DMA multi-channel.

#### Description

This function forcibly suspends the USB IN (egress) socket corresponding to the DMA multi-channel. This function is intended for API internal usage (implementation if specific USB defect work-arounds) and is not expected to be called by user application code.

Note: This function only works for MANY-TO-ONE DMA channels, and not for channels have multiple consumers.

#### Return Value

`CY_U3P_ERROR_BAD_ARGUMENT` if the channel handle is not valid.

`CY_U3P_SUCCESS` if the socket was suspended as requested.

`CY_U3P_ERROR_TIMEOUT` if the socket suspend operation timed out.

#### See also

[CyU3PDmaMultiChannelResumeUsbConsumer](#)

#### Parameters

<i>handle</i>	Handle to channel to be suspended.
<i>waitOption</i>	Amount of time to wait for channel suspension.

5.17.4.45 `CyU3PReturnStatus_t CyU3PDmaMultiChannelUpdateMode ( CyU3PDmaMultiChannel * handle,  
 CyU3PDmaMode_t dmaMode )`

Update the DMA mode for a multi-channel.

#### Description

The DMA mode for a channel specifies the units in which data transfers on the channel are setup and measured. This is generally set during channel creation and retained unchanged there-after. This function can be used to dynamically change the DMA mode for a multi-channel.

The function can only be called when the DMA channel is in the configured state.

#### Return value

`CY_U3P_SUCCESS` - if the function call is successful.

`CY_U3P_ERROR_NULL_POINTER` - if any pointer passed as parameter is NULL.

`CY_U3P_ERROR_BAD_ARGUMENT` - if the DMA mode is invalid.

`CY_U3P_ERROR_NOT_CONFIGURED` - if the DMA channel was not configured.

`CY_U3P_ERROR_INVALID_SEQUENCE` - if the DMA channel is not in the Configured state.

`CY_U3P_ERROR_MUTEX_FAILURE` - if the DMA channel mutex could not be acquired.

## See also

[CyU3PDmaMode\\_t](#)  
[CyU3PDmaMultiChannel](#)  
[CyU3PDmaMultiChannelCreate](#)  
[CyU3PDmaMultiChannelSetXfer](#)  
[CyU3PDmaMultiChannelGetStatus](#)

## Parameters

<i>handle</i>	Handle to the multi-channel to be modified.
<i>dmaMode</i>	Desired DMA mode.

#### 5.17.4.46 CyU3PReturnStatus\_t CyU3PDmaMultiChannelWaitForCompletion ( *CyU3PDmaMultiChannel \* handle*, *uint32\_t waitOption* )

Wait for the current DMA transaction to complete on a DMA multi-channel.

**Description**

This function waits until the current transfer on the DMA channel is completed, or until the specified timeout period has elapsed. This function is not supported if the DMA channel has been configured for infinite transfers.

This function should not be called from a DMA callback function. If the function returns CY\_U3P\_ERROR\_TIME\_OUT, the wait API call can be repeated without affecting the data transfer.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - if this operation is not supported for the DMA channel type.  
 CY\_U3P\_ERROR\_NOT\_STARTED - if the DMA channel is not started.  
 CY\_U3P\_ERROR\_DMA\_FAILURE - if the DMA transfer failed.  
 CY\_U3P\_ERROR\_ABORTED - if the DMA transfer was aborted.  
 CY\_U3P\_ERROR\_TIMEOUT - if the DMA transfer timed out.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.

## See also

[CyU3PDmaMultiChannel](#)  
[CyU3PDmaMultiChannelSetXfer](#)  
[CyU3PDmaMultiChannelSetupSendBuffer](#)  
[CyU3PDmaMultiChannelSetupRecvBuffer](#)  
[CyU3PDmaMultiChannelWaitForRecvBuffer](#)  
[CyU3PDmaMultiChannelReset](#)

## Parameters

<i>handle</i>	Handle to the multi-channel to wait on.
<i>waitOption</i>	Duration for which to wait.

#### 5.17.4.47 CyU3PReturnStatus\_t CyU3PDmaMultiChannelWaitForRecvBuffer ( *CyU3PDmaMultiChannel \* handle*, *CyU3PDmaBuffer\_t \* buffer\_p*, *uint32\_t waitOption* )

Wait until an override read operation is completed.

**Description**

This function waits until the read operation initiated through the CyU3PDmaMultiChannelSetupRecvBuffer API is completed. Once the transfer is completed, it returns the status of the buffer that was filled with data. If not, a timeout error is returned. It is possible to retry this wait API without resetting or aborting the channel.

This function must not be called from a DMA callback function.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if any pointer passed as parameter is NULL.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the DMA channel was not configured.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - if this operation is not supported for the DMA channel type.  
 CY\_U3P\_ERROR\_INVALID\_SEQUENCE - if the DMA channel is not in the required state.  
 CY\_U3P\_ERROR\_DMA\_FAILURE - if the DMA transfer failed.  
 CY\_U3P\_ERROR\_ABORTED - if the DMA transfer was aborted.  
 CY\_U3P\_ERROR\_TIMEOUT - if the DMA transfer timed out.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - if the DMA channel mutex could not be acquired.  
 CY\_U3P\_ERROR\_NOT\_STARTED - if the DMA channel is not started.

**See also**

[CyU3PDmaBuffer\\_t](#)  
[CyU3PDmaMultiChannel](#)  
[CyU3PDmaMultiChannelSetupRecvBuffer](#)  
[CyU3PDmaMultiChannelWaitForCompletion](#)

**Parameters**

<i>handle</i>	Handle to the DMA channel on which to wait.
<i>buffer_p</i>	Output parameter which will be filled up with the address, count and status values of the DMA buffer into which data was received.
<i>waitOption</i>	Duration to wait for the receive completion.

**5.17.4.48 CyU3PDmaChannel\* CyU3PDmaUsbInEpGetChannel ( uint8\_t ep )**

Get the DMA channel handle corresponding to a USB IN endpoint.

**Description**

This function retrieves the DMA channel handle corresponding to a USB IN endpoint. This function is intended for API internal usage (implementation of specific USB defect work-arounds) and is not expected to be called by user application code.

**Return Value**

Pointer to DMA channel if it exists.

**5.17.4.49 CyU3PDmaMultiChannel\* CyU3PDmaUsbInEpGetMultiChannel ( uint8\_t ep )**

Get the DMA multi-channel handle corresponding to a USB IN endpoint.

**Description**

This function retrieves the DMA multi-channel handle corresponding to a USB IN endpoint. This function is intended for API internal usage (implementation of specific USB defect work-arounds) and is not expected to be called by user application code.

**Return Value**

Pointer to DMA channel if it exists.

## 5.18 firmware/u3p\_firmware/inc/cyu3error.h File Reference

This file lists the error codes that are returned by the firmware library functions.

```
#include "cyu3types.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

### TypeDefs

- `typedef enum CyU3PErrorCode_t CyU3PErrorCode_t`

*List of error codes defined and returned by the FX3 firmware library.*

### Enumerations

- `enum CyU3PErrorCode_t {`
- `CY_U3P_SUCCESS = 0, CY_U3P_ERROR_DELETED, CY_U3P_ERROR_BAD_POOL, CY_U3P_ERR←OR_BAD_POINTER,`
- `CY_U3P_ERROR_INVALID_WAIT, CY_U3P_ERROR_BAD_SIZE, CY_U3P_ERROR_BAD_EVENT_GRP,`
- `CY_U3P_ERROR_NO_EVENTS,`
- `CY_U3P_ERROR_BAD_OPTION, CY_U3P_ERROR_BAD_QUEUE, CY_U3P_ERROR_QUEUE_EMPTY,`
- `CY_U3P_ERROR_QUEUE_FULL,`
- `CY_U3P_ERROR_BAD_SEMAPHORE, CY_U3P_ERROR_SEMGET_FAILED, CY_U3P_ERROR_BAD←THREAD, CY_U3P_ERROR_BAD_PRIORITY,`
- `CY_U3P_ERROR_MEMORY_ERROR, CY_U3P_ERROR_DELETE_FAILED, CY_U3P_ERROR_RESU←ME_FAILED, CY_U3P_ERROR_INVALID_CALLER,`
- `CY_U3P_ERROR_SUSPEND_FAILED, CY_U3P_ERROR_BAD_TIMER, CY_U3P_ERROR_BAD_TICK,`
- `CY_U3P_ERROR_ACTIVATE_FAILED,`
- `CY_U3P_ERROR_BAD_THRESHOLD, CY_U3P_ERROR_SUSPEND_LIFTED, CY_U3P_ERROR_WAI←T_ABORTED, CY_U3P_ERROR_WAIT_ABORT_FAILED,`
- `CY_U3P_ERROR_BAD_MUTEX, CY_U3P_ERROR_MUTEX_FAILURE, CY_U3P_ERROR_MUTEX_PU←T_FAILED, CY_U3P_ERROR_INHERIT_FAILED,`
- `CY_U3P_ERROR_NOT_IDLE, CY_U3P_ERROR_BAD_ARGUMENT = 0x40, CY_U3P_ERROR_NULL←POINTER, CY_U3P_ERROR_NOT_STARTED,`
- `CY_U3P_ERROR_ALREADY_STARTED, CY_U3P_ERROR_NOT_CONFIGURED, CY_U3P_ERROR_T←IMEOUT, CY_U3P_ERROR_NOT_SUPPORTED,`
- `CY_U3P_ERROR_INVALID_SEQUENCE, CY_U3P_ERROR_ABORTED, CY_U3P_ERROR_DMA_FAIL←URE, CY_U3P_ERROR_FAILURE,`
- `CY_U3P_ERROR_BAD_INDEX, CY_U3P_ERROR_BAD_ENUM_METHOD, CY_U3P_ERROR_INVALID←D_CONFIGURATION, CY_U3P_ERROR_CHANNEL_CREATE_FAILED,`
- `CY_U3P_ERROR_CHANNEL_DESTROY_FAILED, CY_U3P_ERROR_BAD_DESCRIPTOR_TYPE, CY_U3P_ERROR_XFER_CANCELLED, CY_U3P_ERROR_FEATURE_NOT_ENABLED,`
- `CY_U3P_ERROR_STALLED, CY_U3P_ERROR_BLOCK_FAILURE, CY_U3P_ERROR_LOST_ARBITR←ATION, CY_U3P_ERROR_STANDBY_FAILED,`
- `CY_U3P_ERROR_INVALID_VOLTAGE_RANGE = 0x60, CY_U3P_ERROR_CARD_WRONG_RESPON←SE, CY_U3P_ERROR_UNSUPPORTED_CARD, CY_U3P_ERROR_CARD_WRONG_STATE,`
- `CY_U3P_ERROR_CMD_NOT_SUPPORTED, CY_U3P_ERROR_CRC, CY_U3P_ERROR_INVALID_AD←DR, CY_U3P_ERROR_INVALID_UNIT,`
- `CY_U3P_ERROR_INVALID_DEV, CY_U3P_ERROR_ALREADY_PARTITIONED, CY_U3P_ERROR_N←OT_PARTITIONED, CY_U3P_ERROR_BAD_PARTITION,`
- `CY_U3P_ERROR_READ_WRITE_ABORTED, CY_U3P_ERROR_WRITE_PROTECTED, CY_U3P_ERR←OR_SIB_INIT, CY_U3P_ERROR_CARD_LOCKED,`
- `CY_U3P_ERROR_CARD_LOCK_FAILURE, CY_U3P_ERROR_CARD_FORCE_ERASE, CY_U3P_ERR←OR_INVALID_BLOCKSIZE, CY_U3P_ERROR_INVALID_FUNCTION,`
- `CY_U3P_ERROR_TUPLE_NOT_FOUND, CY_U3P_ERROR_IO_ABORTED, CY_U3P_ERROR_IO_SU←`

```

SPENDED, CY_U3P_ERROR_ILLEGAL_CMD,
CY_U3P_ERROR_SDIO_UNKNOWN, CY_U3P_ERROR_BAD_CMD_ARG, CY_U3P_ERROR_UNINITIALIZED_FUNCTION, CY_U3P_ERROR_CARD_NOT_ACTIVE,
CY_U3P_ERROR_DEVICE_BUSY, CY_U3P_ERROR_NO_METADATA, CY_U3P_ERROR_CARD_UNHEALTHY, CY_U3P_ERROR_MEDIA_FAILURE,
CY_U3P_ERROR_NO_REENUM_REQUIRED = 0xFE, CY_U3P_ERROR_OPERN_DISABLED = 0xFF }
```

*List of error codes defined and returned by the FX3 firmware library.*

### 5.18.1 Detailed Description

This file lists the error codes that are returned by the firmware library functions.

### 5.18.2 Error Codes

The following return codes are used by the FX3 APIs to indicate success/failure and the cause of error in case of failure.

### 5.18.3 Typedef Documentation

#### 5.18.3.1 `typedef enum CyU3PErrorCode_t CyU3PErrorCode_t`

List of error codes defined and returned by the FX3 firmware library.

##### Description

The error codes below help identify the cause of the failure. These errors could be returned by the base RTOS or by the FX3 API itself. The initial error codes are all defined and returned by the RTOS. The API specific error code start from CY\_U3P\_ERROR\_BAD\_ARGUMENT.

### 5.18.4 Enumeration Type Documentation

#### 5.18.4.1 `enum CyU3PErrorCode_t`

List of error codes defined and returned by the FX3 firmware library.

##### Description

The error codes below help identify the cause of the failure. These errors could be returned by the base RTOS or by the FX3 API itself. The initial error codes are all defined and returned by the RTOS. The API specific error code start from CY\_U3P\_ERROR\_BAD\_ARGUMENT.

##### Enumerator

**CY\_U3P\_SUCCESS** Success code

**CY\_U3P\_ERROR\_DELETED** The OS object being accessed has been deleted.

**CY\_U3P\_ERROR\_BAD\_POOL** Bad memory pool passed to a function.

**CY\_U3P\_ERROR\_BAD\_POINTER** Bad (NULL or unaligned) pointer passed to a function.

**CY\_U3P\_ERROR\_INVALID\_WAIT** Non-zero wait requested from interrupt context.

**CY\_U3P\_ERROR\_BAD\_SIZE** Invalid size value passed into a function.

**CY\_U3P\_ERROR\_BAD\_EVENT\_GRP** Invalid event group passed into a function.

**CY\_U3P\_ERROR\_NO\_EVENTS** Failed to set/get the event flags specified.

**CY\_U3P\_ERROR\_BAD\_OPTION** Invalid task option value specified for the function.

**CY\_U3P\_ERROR\_BAD\_QUEUE** Invalid message queue passed to a function.

**CY\_U3P\_ERROR\_QUEUE\_EMPTY** The message queue being read is empty.

**CY\_U3P\_ERROR\_QUEUE\_FULL** The message queue being written to is full.

**CY\_U3P\_ERROR\_BAD\_SEMAPHORE** Invalid semaphore pointer passed to a function.

**CY\_U3P\_ERROR\_SEMGET\_FAILED** A semaphore get operation failed.

**CY\_U3P\_ERROR\_BAD\_THREAD** Invalid thread pointer passed to a function.

**CY\_U3P\_ERROR\_BAD\_PRIORITY** Invalid thread priority value passed to a function.

**CY\_U3P\_ERROR\_MEMORY\_ERROR** Failed to allocate memory.

**CY\_U3P\_ERROR\_DELETE\_FAILED** Failed to delete an object because it is not idle.

**CY\_U3P\_ERROR\_RESUME\_FAILED** Failed to resume a thread.

**CY\_U3P\_ERROR\_INVALID\_CALLER** OS function failed because the current caller is not allowed.

**CY\_U3P\_ERROR\_SUSPEND\_FAILED** Failed to suspend a thread.

**CY\_U3P\_ERROR\_BAD\_TIMER** Invalid timer pointer passed to a function.

**CY\_U3P\_ERROR\_BAD\_TICK** Invalid (0) tick value passed to a timer function.

**CY\_U3P\_ERROR\_ACTIVATE\_FAILED** Failed to activate a timer.

**CY\_U3P\_ERROR\_BAD\_THRESHOLD** Invalid thread pre-emption threshold value specified.

**CY\_U3P\_ERROR\_SUSPEND\_LIFTED** Thread suspension was cancelled.

**CY\_U3P\_ERROR\_WAIT\_ABORTED** Wait operation was aborted.

**CY\_U3P\_ERROR\_WAIT\_ABORT\_FAILED** Failed to abort wait operation on a thread.

**CY\_U3P\_ERROR\_BAD\_MUTEX** Invalid Mutex pointer passed to a function.

**CY\_U3P\_ERROR\_MUTEX\_FAILURE** Failed to get a mutex.

**CY\_U3P\_ERROR\_MUTEX\_PUT\_FAILED** Failed to put a mutex because it is not currently owned.

**CY\_U3P\_ERROR\_INHERIT\_FAILED** Error in priority inheritance.

**CY\_U3P\_ERROR\_NOT\_IDLE** Operation failed because relevant object is not idle or done.

**CY\_U3P\_ERROR\_BAD\_ARGUMENT** One or more parameters to a function are invalid.

**CY\_U3P\_ERROR\_NULL\_POINTER** A null pointer has been passed in unexpectedly.

**CY\_U3P\_ERROR\_NOT\_STARTED** The object/module being referred to has not been started.

**CY\_U3P\_ERROR\_ALREADY\_STARTED** An object/module that is already active is being started.

**CY\_U3P\_ERROR\_NOT\_CONFIGURED** Object/module referred to has not been configured.

**CY\_U3P\_ERROR\_TIMEOUT** Timeout on relevant operation.

**CY\_U3P\_ERROR\_NOT\_SUPPORTED** Operation requested is not supported in current mode.

**CY\_U3P\_ERROR\_INVALID\_SEQUENCE** Invalid function call sequence.

**CY\_U3P\_ERROR\_ABORTED** Function call failed as it was aborted by another thread/isr.

**CY\_U3P\_ERROR\_DMA\_FAILURE** DMA engine failed to completed requested operation.

**CY\_U3P\_ERROR\_FAILURE** Failure due to a non-specific system error.

**CY\_U3P\_ERROR\_BAD\_INDEX** Bad index value was passed in as parameter. Ex: for string descriptor.

**CY\_U3P\_ERROR\_BAD\_ENUM\_METHOD** Bad enumeration method specified.

**CY\_U3P\_ERROR\_INVALID\_CONFIGURATION** Invalid configuration specified.

**CY\_U3P\_ERROR\_CHANNEL\_CREATE\_FAILED** Internal DMA channel creation failed.

**CY\_U3P\_ERROR\_CHANNEL\_DESTROY\_FAILED** Internal DMA channel destroy failed.

**CY\_U3P\_ERROR\_BAD\_DESCRIPTOR\_TYPE** Invalid descriptor type specified.

**CY\_U3P\_ERROR\_XFER\_CANCELLED** USB transfer was cancelled.

**CY\_U3P\_ERROR\_FEATURE\_NOT\_ENABLED** When a USB feature like remote wakeup is not enabled.

**CY\_U3P\_ERROR\_STALLED** When a USB request / data transfer is stalled.

**CY\_U3P\_ERROR\_BLOCK\_FAILURE** The block accessed has a fatal error and needs to be re-initialized.

**CY\_U3P\_ERROR\_LOST\_ARBITRATION** Loss of bus arbitration, invalid bus behaviour or bus busy.

**CY\_U3P\_ERROR\_STANDBY\_FAILED** Failed to enter standby mode because one or more wakeup events are active.

**CY\_U3P\_ERROR\_INVALID\_VOLTAGE\_RANGE** Storage device's voltage range does not meet FX3S requirements.

**CY\_U3P\_ERROR\_CARD\_WRONG\_RESPONSE** Incorrect response received from storage device.

**CY\_U3P\_ERROR\_UNSUPPORTED\_CARD** Storage device features are not supported by FX3S host.

**CY\_U3P\_ERROR\_CARD\_WRONG\_STATE** Storage device failed to move to expected state.

**CY\_U3P\_ERROR\_CMD\_NOT\_SUPPORTED** Storage device failed to support required commands.

**CY\_U3P\_ERROR\_CRC** Response CRC error detected.

**CY\_U3P\_ERROR\_INVALID\_ADDR** Out of range address provided for read/write/erase access.

**CY\_U3P\_ERROR\_INVALID\_UNIT** Non-existent storage partition selected for transfer.

**CY\_U3P\_ERROR\_INVALID\_DEV** Access to port with no connected storage device.

**CY\_U3P\_ERROR\_ALREADY\_PARTITIONED** Request to partition a device which is already partitioned.

**CY\_U3P\_ERROR\_NOT\_PARTITIONED** Request to remove partitions on an unpartitioned device.

**CY\_U3P\_ERROR\_BAD\_PARTITION** Incorrect partition selected.

**CY\_U3P\_ERROR\_READ\_WRITE\_ABORTED** Read/write transfer was aborted (user cancellation or time-out).

**CY\_U3P\_ERROR\_WRITE\_PROTECTED** Write request addressed to a write protected storage device.

**CY\_U3P\_ERROR\_SIB\_INIT** Storage driver initialization failed.

**CY\_U3P\_ERROR\_CARD\_LOCKED** Access to password locked SD card.

**CY\_U3P\_ERROR\_CARD\_LOCK\_FAILURE** Failure while locking/unlocking the SD card.

**CY\_U3P\_ERROR\_CARD\_FORCE\_ERASE** Failure during SD force erase operation.

**CY\_U3P\_ERROR\_INVALID\_BLOCKSIZE** Block size specified for SDIO device is not supported.

**CY\_U3P\_ERROR\_INVALID\_FUNCTION** Non-existent SDIO function being accessed.

**CY\_U3P\_ERROR\_TUPLE\_NOT\_FOUND** Non-existent tuple of SDIO card being accessed.

**CY\_U3P\_ERROR\_IO\_ABORTED** IO operation to SDIO card aborted.

**CY\_U3P\_ERROR\_IO\_SUSPENDED** IO operation to SDIO card suspended.

**CY\_U3P\_ERROR\_ILLEGAL\_CMD** Invalid command sent to the SDIO card.

**CY\_U3P\_ERROR\_SDIO\_UNKNOWN** Generic error reported by SDIO card.

**CY\_U3P\_ERROR\_BAD\_CMD\_ARG** SDIO command argument is out of range.

**CY\_U3P\_ERROR\_UNINITIALIZED\_FUNCTION** Access to uninitialized SDIO function.

**CY\_U3P\_ERROR\_CARD\_NOT\_ACTIVE** Access to SDIO card which is not active.

**CY\_U3P\_ERROR\_DEVICE\_BUSY** The storage device is busy handling another request.

**CY\_U3P\_ERROR\_NO\_METADATA** No metadata present on card

**CY\_U3P\_ERROR\_CARD\_UNHEALTHY** Card RD/WR Threshold error crossed

**CY\_U3P\_ERROR\_MEDIA\_FAILURE** Card not responding to read/write transactions

**CY\_U3P\_ERROR\_NO\_REENUM\_REQUIRED** FX3 booter supports the NoReEnumeration feature that enables to have a single USB enumeration across the FX3 booter and the final application. This error code is returned by CyU3PUsbStart () to indicate that the NoReEnumeration is successful and the sequence followed for the regular enumeration post CyU3PUsbStart () is to be skipped.

**CY\_U3P\_ERROR\_OPERN\_DISABLED** The requested feature/operation is not enabled in the current configuration.

## 5.19 firmware/u3p\_firmware/inc/cyu3gpiif.h File Reference

This file defines the GPIF related data structures and APIs in the FX3 SDK.

```
#include "cyu3types.h"
#include "cyu3dma.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

## Data Structures

- struct [CyU3PGpifWaveData](#)  
*Information on a single GPIF transition from one state to another.*
- struct [CyU3PGpifConfig\\_t](#)  
*Structure that holds all configuration inputs for the GPIF hardware.*

## Macros

- #define [CYU3P\\_GPIF\\_NUM\\_STATES](#) (256)  
*Number of states supported by the GPIF hardware.*
- #define [CYU3P\\_GPIF\\_NUM\\_TRANS\\_FNS](#) (32)  
*Number of distinct transfer functions supported by the GPIF hardware.*
- #define [CYU3P\\_GPIF\\_INVALID\\_STATE](#) (0xFFFF)  
*Invalid state index for use in state machine control functions.*
- #define [CYU3P\\_PIB\\_THREAD\\_COUNT](#) (4)  
*Number of DMA threads on the GPIF (P-Port)*
- #define [CYU3P\\_PIB\\_SOCKET\\_COUNT](#) (32)  
*Number of DMA sockets on the GPIF (P-port)*
- #define [CYU3P\\_PIB\\_MAX\\_BURST\\_SETTING](#) (14)  
*Maximum burst size allowed for P-port sockets. The constant corresponds to Log(2) of size, which means that the max. size is 16 KB.*

## Typedefs

- typedef struct [CyU3PGpifWaveData](#) [CyU3PGpifWaveData](#)  
*Information on a single GPIF transition from one state to another.*
- typedef enum [CyU3PGpifEventType](#) [CyU3PGpifEventType](#)  
*List of GPIF related events tracked by the driver.*
- typedef enum [CyU3PGpifComparatorType](#) [CyU3PGpifComparatorType](#)  
*List of supported comparators in the GPIF hardware.*
- typedef void(\* [CyU3PGpifEventCb\\_t](#)) ([CyU3PGpifEventType](#) event, uint8\_t currentState)  
*Callback type that is invoked to inform the application about GPIF events.*
- typedef void(\* [CyU3PGpifSMIntrCb\\_t](#)) (uint8\_t stateld)  
*Special callback type used for notification of GPIF state machine interrupts.*
- typedef enum [CyU3PGpifOutput\\_t](#) [CyU3PGpifOutput\\_t](#)  
*List of control outputs and flags from the GPIF hardware.*
- typedef struct [CyU3PGpifConfig\\_t](#) [CyU3PGpifConfig\\_t](#)  
*Structure that holds all configuration inputs for the GPIF hardware.*

## Enumerations

- enum [CyU3PGpifEventType](#) {  
CYU3P\_GPIF\_EVT\_END\_STATE = 0, CYU3P\_GPIF\_EVT\_SM\_INTERRUPT, CYU3P\_GPIF\_EVT\_SWITCH\_TIMEOUT, CYU3P\_GPIF\_EVT\_ADDR\_COUNTER,  
CYU3P\_GPIF\_EVT\_DATA\_COUNTER, CYU3P\_GPIF\_EVT\_CTRL\_COUNTER, CYU3P\_GPIF\_EVT\_ADDRESS\_COMP,  
CYU3P\_GPIF\_EVT\_CTRL\_COMP, CYU3P\_GPIF\_EVT\_CRC\_ERROR }  
*List of GPIF related events tracked by the driver.*
- enum [CyU3PGpifComparatorType](#) { CYU3P\_GPIF\_COMP\_CTRL = 0, CYU3P\_GPIF\_COMP\_ADDR, CYU3P\_GPIF\_COMP\_DATA }

*List of supported comparators in the GPIF hardware.*

- enum CyU3PGpifOutput\_t {
 CYU3P\_GPIF\_OP\_ALPHA0 = 0, CYU3P\_GPIF\_OP\_ALPHA1, CYU3P\_GPIF\_OP\_ALPHA2, CYU3P\_GPIF\_OP\_ALPHA3,
 CYU3P\_GPIF\_OP\_BETA0 = 8, CYU3P\_GPIF\_OP\_BETA1, CYU3P\_GPIF\_OP\_BETA2, CYU3P\_GPIF\_OP\_BETA3,
 CYU3P\_GPIF\_OP\_THR0\_READY = 16, CYU3P\_GPIF\_OP\_THR1\_READY, CYU3P\_GPIF\_OP\_THR2\_READY,
 CYU3P\_GPIF\_OP\_THR3\_READY,
 CYU3P\_GPIF\_OP\_THR0\_PART, CYU3P\_GPIF\_OP\_THR1\_PART, CYU3P\_GPIF\_OP\_THR2\_PART, CYU3P\_GPIF\_OP\_THR3\_PART,
 CYU3P\_GPIF\_OP\_DMA\_READY, CYU3P\_GPIF\_OP\_PARTIAL, CYU3P\_GPIF\_OP\_PPDRQ }

*List of control outputs and flags from the GPIF hardware.*

## Functions

- CyU3PReturnStatus\_t CyU3PGpifLoad (const CyU3PGpifConfig\_t \*conf)
 

*Function to program the user defined state machine into the GPIF registers.*
- void CyU3PGpifRegisterCallback (CyU3PGpifEventCb\_t cbFunc)
 

*This function registers an event callback for the GPIF driver.*
- void CyU3PGpifRegisterSMIntrCallback (CyU3PGpifSMIntrCb\_t cb)
 

*Register a callback function for fast notification of GPIF state machine interrupts.*
- CyU3PReturnStatus\_t CyU3PGpifWaveformLoad (uint8\_t firstState, uint16\_t stateCnt, uint8\_t \*stateDataMap, CyU3PGpifWaveData \*transitionData)
 

*Initialize the GPIF state machine table.*
- CyU3PReturnStatus\_t CyU3PGpifConfigure (uint8\_t numRegs, const uint32\_t \*regData)
 

*Initialize the GPIF configuration registers.*
- CyU3PReturnStatus\_t CyU3PGpifRegisterConfig (uint16\_t numRegs, uint32\_t regData[ ][2])
 

*Initialize the GPIF configuration registers.*
- CyU3PReturnStatus\_t CyU3PGpifInitTransFunctions (uint16\_t \*fnTable)
 

*Initialize the GPIF transition function registers.*
- void CyU3PGpifDisable (CyBool\_t forceReload)
 

*Disables the GPIF state machine and hardware.*
- CyU3PReturnStatus\_t CyU3PGpifSMStart (uint8\_t stateIndex, uint8\_t initialAlpha)
 

*Start the waveform state machine from the specified state.*
- CyU3PReturnStatus\_t CyU3PGpifSMSwitch (uint16\_t fromState, uint16\_t toState, uint16\_t endState, uint8\_t initialAlpha, uint32\_t switchTimeout)
 

*This function is used to start GPIF state machine execution from a desired state.*
- void CyU3PGpifInitCtrlCounter (uint16\_t initValue, uint16\_t limit, CyBool\_t reload, CyBool\_t upCount, uint8\_t outputBit)
 

*Function to configure the GPIF control counter.*
- void CyU3PGpifInitAddrCounter (uint32\_t initValue, uint32\_t limit, CyBool\_t reload, CyBool\_t upCount, uint8\_t increment)
 

*Function to configure the GPIF address counter.*
- void CyU3PGpifInitDataCounter (uint32\_t initValue, uint32\_t limit, CyBool\_t reload, CyBool\_t upCount, uint8\_t increment)
 

*Function to configure the GPIF data counter.*
- CyU3PReturnStatus\_t CyU3PGpifInitComparator (CyU3PGpifComparatorType type, uint32\_t value, uint32\_t mask)
 

*This function configures one of the comparators in the GPIF hardware.*
- CyU3PReturnStatus\_t CyU3PGpifSMControl (CyBool\_t pause)
 

*Function to pause/resume the GPIF state machine.*
- void CyU3PGpifControlSWInput (CyBool\_t set)

*Function to set/clear the software controlled state machine input.*

- [CyU3PReturnStatus\\_t CyU3PGpifSocketConfigure](#) (uint8\_t threadIndex, [CyU3PDmaSocketId\\_t](#) socketNum, uint16\_t watermark, [CyBool\\_t](#) flagOnData, uint8\_t burst)

*Function to select an active socket on the P-port and to configure it.*

- [CyU3PReturnStatus\\_t CyU3PGpifReadDataWords](#) (uint32\_t threadIndex, [CyBool\\_t](#) selectThread, uint32\_t numWords, uint32\_t \*buffer\_p, uint32\_t waitOption)

*Read a specified number of data words from the GPIF interface.*

- [CyU3PReturnStatus\\_t CyU3PGpifWriteDataWords](#) (uint32\_t threadIndex, [CyBool\\_t](#) selectThread, uint32\_t numWords, uint32\_t \*buffer\_p, uint32\_t waitOption)

*Write a specified number of data words to the GPIF interface.*

- [CyU3PReturnStatus\\_t CyU3PGpifOutputConfigure](#) (uint8\_t ctrlPin, [CyU3PGpifOutput\\_t](#) opFlag, [CyBool\\_t](#) isActiveLow)

*Configure the functionality of a GPIF output pin.*

- [CyU3PReturnStatus\\_t CyU3PGpifGetSMState](#) (uint8\_t \*curState\_p)

*Get the current state of the GPIF state machine.*

### 5.19.1 Detailed Description

This file defines the GPIF related data structures and APIs in the FX3 SDK.

#### Description

The FX3 device includes a General Programmable Interface which can be used to connect it to any other processor, ASIC or FPGA using most master or slave protocols. The GPIF-II block on the FX3 device takes care of implementing the desired protocols by using a programmable state machine.

The GPIF-II APIs provide functions that can be used to configure the GPIF state machine to implement any desired protocol. Functions are also provided to control the execution of the state machine and to manage the actual data transfer across the GPIF configured interface.

### 5.19.2 Typedef Documentation

#### 5.19.2.1 [typedef enum CyU3PGpifComparatorType CyU3PGpifComparatorType](#)

List of supported comparators in the GPIF hardware.

#### Description

This function lists the hardware comparators that are part of the GPIF block. Each of these comparators is configured by calling the [CyU3PGpifInitComparator](#) function with the corresponding type.

#### See also

[CyU3PGpifInitComparator](#)

#### 5.19.2.2 [typedef struct CyU3PGpifConfig\\_t CyU3PGpifConfig\\_t](#)

Structure that holds all configuration inputs for the GPIF hardware.

#### Description

The GPIF block on the FX3 device has a set of general configuration registers, transition function registers and state descriptors that need to be initialized to make the GPIF state machine functional. This structure encapsulates all the data that is required to program the GPIF block to load a user defined state machine.

#### See also

[CyU3PGpifLoad](#)

[CyU3PGpifWaveData](#)

**5.19.2.3 `typedef void(* CyU3PGpifEventCb_t) (CyU3PGpifEventType event, uint8_t currentState )`**

Callback type that is invoked to inform the application about GPIF events.

**Description**

This type defines the signature for the callback function that is invoked by the GPIF driver to inform the user application about GPIF related events. A callback function of this type should be registered with the GPIF driver.

**See also**

[CyU3PGpifRegisterCallback](#)

**5.19.2.4 `typedef enum CyU3PGpifEventType CyU3PGpifEventType`**

List of GPIF related events tracked by the driver.

**Description**

This type lists the various GPIF hardware and state machine related events that may be notified to the user application.

**5.19.2.5 `typedef enum CyU3PGpifOutput_t CyU3PGpifOutput_t`**

List of control outputs and flags from the GPIF hardware.

**Description**

The GPIF block in the FX3 device can generate a set of control outputs and DMA status flags that can be used by the external processor to control data transfers. This enumeration lists the various control outputs and flags that are supported by the GPIF.

The four Alpha outputs and Beta outputs listed below correspond to the output signals configured in the GPIF II Designer project. The Alpha outputs are the early outputs (low latency) configured in GPIF II Designer, and the Beta outputs are the delayed (higher latency) output signals.

The user defined output signals as well as the various DMA flags can be selected for connection to a given GPIF control signal.

**See also**

[CyU3PGpifOutputConfigure](#)

**5.19.2.6 `typedef void(* CyU3PGpifSMIntrCb_t) (uint8_t stateld )`**

Special callback type used for notification of GPIF state machine interrupts.

**Description**

The generic GPIF interrupt callback mechanism provided by [CyU3PGpifRegisterCallback](#) delivers the callback in thread context. While this allows the user to call potentially blocking API calls from the callback, the latency from interrupt to callback will not be predictable and can extend to about 100 us.

This callback type provides notifications of only GPIF state machine interrupts (raised using the INTR\_CPU action in the state machine) in interrupt context. This notification can be used to obtain a low latency notification of state machine interrupts. Please note that API calls that require a mutex get (such as DMA channel APIs) cannot be made directly from this callback.

**See also**

[CyU3PGpifRegisterSMIntrCallback](#)

### 5.19.2.7 `typedef struct CyU3PGpifWaveData CyU3PGpifWaveData`

Information on a single GPIF transition from one state to another.

#### Description

The GPIF state machine on the FX3 device is defined through a set of transition descriptors. These descriptors include fields for specifying the next state, the conditions for transition, and the output values.

This structure encapsulates all of the information that forms the left and right transition descriptors for a state.

#### See also

[CyU3PGpifWaveformLoad](#)

## 5.19.3 Enumeration Type Documentation

### 5.19.3.1 `enum CyU3PGpifComparatorType`

List of supported comparators in the GPIF hardware.

#### Description

This function lists the hardware comparators that are part of the GPIF block. Each of these comparators is configured by calling the `CyU3PGpifInitComparator` function with the corresponding type.

#### See also

[CyU3PGpifInitComparator](#)

#### Enumerator

**`CYU3P_GPIF_COMP_CTRL`** Control signals comparator.

**`CYU3P_GPIF_COMP_ADDR`** Address bus comparator.

**`CYU3P_GPIF_COMP_DATA`** Data bus comparator.

### 5.19.3.2 `enum CyU3PGpifEventType`

List of GPIF related events tracked by the driver.

#### Description

This type lists the various GPIF hardware and state machine related events that may be notified to the user application.

#### Enumerator

**`CYU3P_GPIF_EVT_END_STATE`** State machine has reached the designated end state.

**`CYU3P_GPIF_EVT_SM_INTERRUPT`** State machine has raised a software interrupt.

**`CYU3P_GPIF_EVT_SWITCH_TIMEOUT`** Desired state machine switch has timed out.

**`CYU3P_GPIF_EVT_ADDR_COUNTER`** Address counter has reached the limit.

**`CYU3P_GPIF_EVT_DATA_COUNTER`** Data counter has reached the limit.

**`CYU3P_GPIF_EVT_CTRL_COUNTER`** Control counter has reached the limit.

**`CYU3P_GPIF_EVT_ADDR_COMP`** Address comparator match has been obtained.

**`CYU3P_GPIF_EVT_DATA_COMP`** Data comparator match has been obtained.

**`CYU3P_GPIF_EVT_CTRL_COMP`** Control comparator match has been obtained.

**`CYU3P_GPIF_EVT_CRC_ERROR`** Incorrect CRC received on a read operation.

### 5.19.3.3 enum CyU3PGpifOutput\_t

List of control outputs and flags from the GPIF hardware.

#### Description

The GPIF block in the FX3 device can generate a set of control outputs and DMA status flags that can be used by the external processor to control data transfers. This enumeration lists the various control outputs and flags that are supported by the GPIF.

The four Alpha outputs and Beta outputs listed below correspond to the output signals configured in the GPIF II Designer project. The Alpha outputs are the early outputs (low latency) configured in GPIF II Designer, and the Beta outputs are the delayed (higher latency) output signals.

The user defined output signals as well as the various DMA flags can be selected for connection to a given GPIF control signal.

#### See also

[CyU3PGpifOutputConfigure](#)

#### Enumerator

**CYU3P\_GPIF\_OP\_ALPHA0** First early output configured in GPIF II Designer.

**CYU3P\_GPIF\_OP\_ALPHA1** Second early output configured in GPIF II Designer.

**CYU3P\_GPIF\_OP\_ALPHA2** Third early output configured in GPIF II Designer.

**CYU3P\_GPIF\_OP\_ALPHA3** Fourth early output configured in GPIF II Designer.

**CYU3P\_GPIF\_OP\_BETA0** First delayed output configured in GPIF II Designer.

**CYU3P\_GPIF\_OP\_BETA1** Second delayed output configured in GPIF II Designer.

**CYU3P\_GPIF\_OP\_BETA2** Third delayed output configured in GPIF II Designer.

**CYU3P\_GPIF\_OP\_BETA3** Fourth delayed output configured in GPIF II Designer.

**CYU3P\_GPIF\_OP\_THR0\_READY** DMA ready flag for Thread 0.

**CYU3P\_GPIF\_OP\_THR1\_READY** DMA ready flag for Thread 1.

**CYU3P\_GPIF\_OP\_THR2\_READY** DMA ready flag for Thread 2.

**CYU3P\_GPIF\_OP\_THR3\_READY** DMA ready flag for Thread 3.

**CYU3P\_GPIF\_OP\_THR0\_PART** DMA watermark flag for Thread 0.

**CYU3P\_GPIF\_OP\_THR1\_PART** DMA watermark flag for Thread 1.

**CYU3P\_GPIF\_OP\_THR2\_PART** DMA watermark flag for Thread 2.

**CYU3P\_GPIF\_OP\_THR3\_PART** DMA watermark flag for Thread 3.

**CYU3P\_GPIF\_OP\_DMA\_READY** DMA ready flag for the active DMA thread.

**CYU3P\_GPIF\_OP\_PARTIAL** DMA watermark flag for the active DMA thread.

**CYU3P\_GPIF\_OP\_PPDRQ** PPDRQ interrupt status derived from the PP\_DRQR5\_MASK register.

### 5.19.4 Function Documentation

#### 5.19.4.1 CyU3PReturnStatus\_t CyU3PGpifConfigure ( uint8\_t numRegs, const uint32\_t \* regData )

Initialize the GPIF configuration registers.

#### Description

The GPIF hardware has a number of configuration registers that control the functioning of the state machine. These registers need to be initialized with values provided by the GPIF designer tool.

This function takes in an array containing the register values, and programs all of the configuration registers with these values.

**Return value**

CY\_U3P\_SUCCESS if successful.

CY\_U3P\_ERROR\_NOT\_SUPPORTED - if a 32 bit GPIF configuration is being used on a part that does not support this.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if any of the parameters are invalid.

CY\_U3P\_ERROR\_ALREADY\_STARTED if the GPIF hardware is running.

**See also**

[CyU3PGpifWaveformLoad](#)  
[CyU3PGpifInitTransFunctions](#)  
[CyU3PGpifRegisterConfig](#)

**Parameters**

<i>numRegs</i>	Number of registers to configure (size of array).
<i>regData</i>	Pointer to uint32_t[] array, where element [i] contains the data to be loaded into GPIF config register "i".

**5.19.4.2 void CyU3PGpifControlSWInput( CyBool\_t set )**

Function to set/clear the software controlled state machine input.

**Description**

The GPIF hardware supports one software driven internal input signal that can be used to control/direct the state machine functionality. This function is used to set the state of this input signal to a desired value.

**Return value**

None

**Parameters**

<i>set</i>	Whether to set (assert) the software input signal.
------------	--

**5.19.4.3 void CyU3PGpifDisable( CyBool\_t forceReload )**

Disables the GPIF state machine and hardware.

**Description**

This function is used to disable the GPIF state machine and hardware. If the forceReload parameter is set to true, the current configuration information is lost and needs to be restored using the CyU3PGpifLoad or equivalent functions. If the forceReload parameter is set to false, the GPIF configuration is retained and it is possible to restart the it by calling the CyU3PGpifSMStart API.

**Return value**

None

**See also**

[CyU3PGpifConfigure](#)

**Parameters**

<i>forceReload</i>	Whether a GPIF re-configuration is to be forced.
--------------------	--

#### 5.19.4.4 CyU3PReturnStatus\_t CyU3PGpifGetSMState ( uint8\_t \* curState\_p )

Get the current state of the GPIF state machine.

##### Description

This function is used to fetch the current state of the GPIF state machine. This is primarily used to provide debug information.

##### Return value

CY\_U3P\_SUCCESS if the query is successful.

CY\_U3P\_ERROR\_NOT\_CONFIGURED if the state machine has not been configured as yet.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the return parameter has not been provided as part of the call.

##### See also

[CyU3PGpifSMStart](#)  
[CyU3PGpifSMSwitch](#)

##### Parameters

<i>curState_p</i>	Return parameter to be filled with current state information.
-------------------	---

#### 5.19.4.5 void CyU3PGpifInitAddrCounter ( uint32\_t initialValue, uint32\_t limit, CyBool\_t reload, CyBool\_t upCount, uint8\_t increment )

Function to configure the GPIF address counter.

##### Description

This function is used to configure the GPIF address counter with the desired initial value, limit and counting mode.

##### Return value

None

##### Parameters

<i>initialValue</i>	Initial value to start the counter from.
<i>limit</i>	Counter limit at which the counter stops.
<i>reload</i>	Whether to reload the counter when the limit is reached.
<i>upCount</i>	Whether to count upwards from the initial value.
<i>increment</i>	The value to be incremented/decremented from the counter at each step.

#### 5.19.4.6 CyU3PReturnStatus\_t CyU3PGpifInitComparator ( CyU3PGpifComparatorType type, uint32\_t value, uint32\_t mask )

This function configures one of the comparators in the GPIF hardware.

##### Description

The GPIF hardware includes comparators that can be used to check if the control, address or data values match a desired pattern. There is a separate comparator for each class of signals. This function is used to configure and initialize one of these comparators as required.

##### Return value

CY\_U3P\_SUCCESS if successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the type specified is invalid.

See also

[CyU3PGpifComparatorType](#)

Parameters

<i>type</i>	The type of comparator to configure.
<i>value</i>	The value to compare the signals against.
<i>mask</i>	Mask that specifies which bits are to be used in the comparison.

5.19.4.7 void CyU3PGpifInitCtrlCounter ( *uint16\_t initialValue*, *uint16\_t limit*, *CyBool\_t reload*, *CyBool\_t upCount*, *uint8\_t outputBit* )

Function to configure the GPIF control counter.

#### Description

The GPIF hardware includes a 16-bit control counter, one of whose bits can be connected to the CTRL[9] output from the device. This function is used to configure and initialize the control counter and to select the bit that should be connected to the CTRL[9] output. Please note that the specified bit location will be truncated to a value less than 16.

#### Return value

None

Parameters

<i>initialValue</i>	Initial (reset) value for the counter.
<i>limit</i>	Value at which to stop the counter and flag an event.
<i>reload</i>	Whether to reload the counter and continue after limit is hit.
<i>upCount</i>	Whether to count upwards from the initial value.
<i>outputBit</i>	Selects counter bit to be connected to CTRL[9] output.

5.19.4.8 void CyU3PGpifInitDataCounter ( *uint32\_t initialValue*, *uint32\_t limit*, *CyBool\_t reload*, *CyBool\_t upCount*, *uint8\_t increment* )

Function to configure the GPIF data counter.

#### Description

This function is used to configure the GPIF data counter with the desired initial value, limit and counting mode.

#### Return value

None

Parameters

<i>initialValue</i>	Initial value to start the counter from.
<i>limit</i>	Counter limit at which the counter stops.
<i>reload</i>	Whether to reload the counter when the limit is reached.
<i>upCount</i>	Whether to count upwards from the initial value.
<i>increment</i>	The value to be incremented/decremented from the counter at each step.

### 5.19.4.9 CyU3PReturnStatus\_t CyU3PGpifInitTransFunctions ( uint16\_t \* *fnTable* )

Initialize the GPIF transition function registers.

#### Description

This function initializes the GPIF transition function registers with data provided by the GPIF designer tool.

#### Return value

CY\_U3P\_SUCCESS if successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if any of the parameters are invalid.

CY\_U3P\_ERROR\_ALREADY\_STARTED if the GPIF hardware is running.

#### See also

[CyU3PGpifConfigure](#)

#### Parameters

<i>fnTable</i>	Pointer to the table (array) containing the values with which the registers should be initialized.
----------------	--

### 5.19.4.10 CyU3PReturnStatus\_t CyU3PGpifLoad ( const CyU3PGpifConfig\_t \* *conf* )

Function to program the user defined state machine into the GPIF registers.

#### Description

This function allows all of the configuration values corresponding to the user defined state machine to be loaded into the GPIF registers. The data to be passed as parameter to this function is received as output from the GPIF Designer tool.

#### Return value

CY\_U3P\_SUCCESS - if the configuration was successful.

CY\_U3P\_ERROR\_NOT\_SUPPORTED - if a 32 bit GPIF configuration is being used on a part that does not support this.

CY\_U3P\_ERROR\_ALREADY\_STARTED - if the GPIF hardware has already been programmed.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the input to the API is inconsistent.

#### See also

[CyU3PGpifConfig\\_t](#)

#### Parameters

<i>conf</i>	Pointer to GPIF configuration structure.
-------------	--

### 5.19.4.11 CyU3PReturnStatus\_t CyU3PGpifOutputConfigure ( uint8\_t *ctrlPin*, CyU3PGpifOutput\_t *opFlag*, CyBool\_t *isActiveLow* )

Configure the functionality of a GPIF output pin.

#### Description

This function is used to configure the functionality of a GPIF output pin. This is primarily used to map a selected flag to a particular control output pin. The function configures the selected control pin as output, updates the polarity and connects the selected flag/signal to the pin.

#### Note

This configuration is likely to be overridden by any GPIF configuration API calls such as CyU3PGpifLoad or CyU3PGpifConfigure.

U3PGpifRegisterConfig.

#### Return value

CY\_U3P\_SUCCESS if the configuration is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the control pin or output selected is invalid.

#### See also

[CyU3PGpifOutput\\_t](#)

#### Parameters

<i>ctrlPin</i>	Control pin number to be configured.
<i>opFlag</i>	Selected output flag.
<i>isActiveLow</i>	Polarity: 0=Active high, 1=Active low.

5.19.4.12 `CyU3PReturnStatus_t CyU3PGpifReadDataWords( uint32_t threadIndex, CyBool_t selectThread, uint32_t numWords, uint32_t * buffer_p, uint32_t waitOption )`

Read a specified number of data words from the GPIF interface.

#### Description

The GPIF hardware supports a mode where data can be read from the P-port in terms of words of the specified interface width. The data can be read from any one of the four threads of data transfer across the P-port.

This function is used to read a specified number of data words into a buffer, one word at a time. Please note that each data word will be placed in the buffer after padding to 32 bits. A timeout period can be specified, and if any of the data words is not available within the specified period from the previous one, the operation will return with a timeout error.

#### Return value

CY\_U3P\_SUCCESS if successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the thread number is invalid.

CY\_U3P\_ERROR\_FAILURE if the operation is not permitted by the GPIF configuration.

#### Parameters

<i>threadIndex</i>	DMA thread index from which to read data.
<i>selectThread</i>	Whether the target thread should be enabled explicitly.
<i>numWords</i>	Number of words of data to read.
<i>buffer_p</i>	Buffer pointer into which the data should be read.
<i>waitOption</i>	Timeout duration to wait for data.

5.19.4.13 `void CyU3PGpifRegisterCallback( CyU3PGpifEventCb_t cbFunc )`

This function registers an event callback for the GPIF driver.

#### Description

The GPIF driver keeps track of GPIF related events and raises notifications to the application logic when required. This function is used to register the callback function that will be invoked to notify the application about GPIF events.

#### Return value

None

## See also

[CyU3PGpifEventCb\\_t](#)  
[CyU3PGpifEventType](#)

## Parameters

<i>cbFunc</i>	Pointer to callback function.
---------------	-------------------------------

5.19.4.14 `CyU3PReturnStatus_t CyU3PGpifRegisterConfig ( uint16_t numRegs, uint32_t regData[ ][2] )`

Initialize the GPIF configuration registers.

**Description**

This is an alternate flavor of the GPIF register configuration function that can be used when only a small subset of the registers needs to be initialized. The input is accepted in the form of a two-columned matrix, column 0 representing the register address to be initialized and column 1 representing the value to be written into this register.

Please note that the registers will be initialized in the same order they are provided in the array, and that it is possible to write multiple times to the same register.

**Return value**

`CY_U3P_SUCCESS` if successful.  
`CY_U3P_ERROR_NOT_SUPPORTED` - if a 32 bit GPIF configuration is being used on a part that does not support this.  
`CY_U3P_ERROR_BAD_ARGUMENT` if any of the parameters are invalid.  
`CY_U3P_ERROR_ALREADY_STARTED` if the GPIF hardware is running.

## See also

[CyU3PGpifConfigure](#)

## Parameters

<i>numRegs</i>	Number of registers to configure (size of the matrix).
<i>regData</i>	Reference to array of {address, data} tuples corresponding to GPIF registers to initialize.

5.19.4.15 `void CyU3PGpifRegisterSMIntrCallback ( CyU3PGpifSMIntrCb_t cb )`

Register a callback function for fast notification of GPIF state machine interrupts.

**Description**

This function registers a callback function that will be invoked from interrupt context when a GPIF state machine interrupt is detected (caused by the INTR\_CPU action in the GPIF state machine).

This callback provides a fast notification of the interrupt and is not subject to thread switching delays. Please note that API calls that require a mutex get or equivalent cannot be directly called from this callback function.

**Return value**

None

## See also

[CyU3PGpifSMIntrCb\\_t](#)

**Parameters**

<i>cb</i>	Callback function pointer.
-----------	----------------------------

**5.19.4.16 CyU3PReturnStatus\_t CyU3PGpifSMControl ( CyBool\_t pause )**

Function to pause/resume the GPIF state machine.

**Description**

The GPIF state machine continuously functions on the basis of configuration values that are set using the CyU3PGpifWaveformLoad API. While the state machine normally functions without any software control, it is possible for the software to stop/start the functioning of the state machine at will. This function allows the application to either pause or resume the state machine.

**Return value**

CY\_U3P\_SUCCESS if successful.

CY\_U3P\_ERROR\_NOT\_CONFIGURED if the state machine has not been configured.

**Parameters**

<i>pause</i>	Whether to pause the state machine or resume it.
--------------	--

**5.19.4.17 CyU3PReturnStatus\_t CyU3PGpifSMStart ( uint8\_t stateIndex, uint8\_t initialAlpha )**

Start the waveform state machine from the specified state.

**Description**

This function starts off the GPIF state machine from the specified state index. Please note that this function should only be used to start the state machine from an idle state. The CyU3PGpifSMSwitch function should be used to switch from one state to another in mid-execution.

**Return value**

CY\_U3P\_SUCCESS if successful.

CY\_U3P\_ERROR\_NOT\_CONFIGURED if GPIF has not been configured as yet.

CY\_U3P\_ERROR\_ALREADY\_STARTED if the state machine is already active.

**See also**

[CyU3PGpifSMSwitch](#)

**Parameters**

<i>stateIndex</i>	State from which to start execution. This should be 0 in most cases.
<i>initialAlpha</i>	Initial alpha values to start GPIF state machine operation with.

**5.19.4.18 CyU3PReturnStatus\_t CyU3PGpifSMSwitch ( uint16\_t fromState, uint16\_t toState, uint16\_t endState, uint8\_t initialAlpha, uint32\_t switchTimeout )**

This function is used to start GPIF state machine execution from a desired state.

**Description**

This function allows the caller to switch to a desired state and continue GPIF state machine execution from there. The toState parameter specifies the state to initiate operation from.

The fromState parameter can be used to ensure that the transition to toState happens only when the state machine

is in a well defined idle state. If a valid state id (0 - 255) is passed as the fromState, the transition is only allowed from that state index. If not, the state machine is immediately switched to the toState.

The endState can be used to obtain a notification when the state machine execution has reached the designated end state. Again, this functionality is only valid if a valid endState value is passed in.

The switchTimeout specifies the amount of time to wait for the transition to the desired toState. This is only meaningful if a fromState is specified, and the timeout value is specified in terms of GPIF hardware clock cycles.

#### Return value

CY\_U3P\_SUCCESS if successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if any of the parameters are invalid.

#### See also

[CyU3PGpifSMStart](#)

#### Parameters

<i>fromState</i>	The state from which to do the switch to the desired state.
<i>toState</i>	The state to which to transition fromState.
<i>endState</i>	The end state for this execution path.
<i>initialAlpha</i>	Initial Alpha values to use when switching states.
<i>switchTimeout</i>	Timeout setting for the switch operation in GPIF clock cycles.

### 5.19.4.19 CyU3PReturnStatus\_t CyU3PGpifSocketConfigure ( uint8\_t *threadIndex*, CyU3PDmaSocketId\_t *socketNum*, uint16\_t *watermark*, CyBool\_t *flagOnData*, uint8\_t *burst* )

Function to select an active socket on the P-port and to configure it.

#### Description

The GPIF hardware allows 4 different sockets on the P-port to be accessed at a time by supporting 4 independent DMA threads. The active socket for each thread and its properties can be selected by the user at run-time. This should be done in software only in the case where it is not being done through the PP registers or the state machine itself.

This function allows the user to select and configure the active socket in the case where software is responsible for these actions. The API will respond with an error if the hardware is taking care of socket configurations.

#### Return value

CY\_U3P\_SUCCESS if successful.

CY\_U3P\_ERROR\_FAILURE if the socket selection and configuration is being done automatically.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if one or more of the parameters are out of range.

#### Parameters

<i>threadIndex</i>	Thread index whose active socket is to be configured.
<i>socketNum</i>	The socket to be associated with this thread.
<i>watermark</i>	Watermark level for this socket in number of 4-byte words.
<i>flagOnData</i>	Whether the partial flag should be set when the socket contains more data than the watermark. If false, the flag will be set when the socket contains less data than the watermark.
<i>burst</i>	Logarithm (to the base 2) of the burst size for this socket. The burst size is the minimum number of words of data that will be sourced/sinked across the GPIF interface without further updates of the GPIF DMA flags. The device connected to FX3 is expected to complete a burst that it has started regardless of any flag changes in between. Please note that this has to be set to a non-zero value (burst size is greater than one), when the GPIF is being configured with a 32-bit data bus and functioning at 100 MHz.

5.19.4.20 `CyU3PReturnStatus_t CyU3PGpifWaveformLoad ( uint8_t firstState, uint16_t stateCnt, uint8_t * stateDataMap, CyU3PGpifWaveData * transitionData )`

Initialize the GPIF state machine table.

#### Description

The GPIF state machine is programmed in the form of a set of waveform table entries. This function is used to take the output state machine data from the designer tool and to initialize the state machine based on these values.

Note that the state data is generated in the form of a transition data array and a state index mapping table by the CyU3PGpifWav designer tool. The transitionData parameter contains all unique combinations of transition data that are part of this state machine. The stateDataMap maps each state index to the transition data entry from the array.

It is possible to load t]TJhip8(t20(le)-3disjo[(u(le)-325(state)-358(machine)-310( [(guener)10nations)-35n57(to)-3distre/u(le)-310(par)

## 5.20 firmware/u3p\_firmware/inc/cyu3gpio.h File Reference

The GPIO manager module is responsible for handling general purpose IO pins on the FX3 device.

```
#include "cyu3types.h"
#include "cyu3system.h"
#include "cyu3lpp.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

### Data Structures

- struct [CyU3PGpioSimpleConfig\\_t](#)  
*Configuration information for simple GPIOs.*
- struct [CyU3PGpioComplexConfig\\_t](#)  
*Configuration information for complex GPIO pins.*

### Typedefs

- typedef enum [CyU3PGpioComplexMode\\_t](#) [CyU3PGpioComplexMode\\_t](#)  
*Enumerated list of all GPIO complex modes.*
- typedef enum [CyU3PGpioTimerMode\\_t](#) [CyU3PGpioTimerMode\\_t](#)  
*List of possible clock sources for GPIO timers.*
- typedef enum [CyU3PGpioIntrMode\\_t](#) [CyU3PGpioIntrMode\\_t](#)  
*List of interrupt modes supported by FX3 GPIOs.*
- typedef struct [CyU3PGpioSimpleConfig\\_t](#) [CyU3PGpioSimpleConfig\\_t](#)  
*Configuration information for simple GPIOs.*
- typedef struct [CyU3PGpioComplexConfig\\_t](#) [CyU3PGpioComplexConfig\\_t](#)  
*Configuration information for complex GPIO pins.*
- typedef void(\* [CyU3PGpioIntrCb\\_t](#)) (uint8\_t gpioid)  
*GPIO callback function type.*

### Enumerations

- enum [CyU3PGpioComplexMode\\_t](#) {  
CY\_U3P\_GPIO\_MODE\_STATIC = 0, CY\_U3P\_GPIO\_MODE\_TOGGLE, CY\_U3P\_GPIO\_MODE\_SAMPLE\_NOW, CY\_U3P\_GPIO\_MODE\_PULSE\_NOW,  
CY\_U3P\_GPIO\_MODE\_PULSE, CY\_U3P\_GPIO\_MODE\_PWM, CY\_U3P\_GPIO\_MODE\_MEASURE\_LOW, CY\_U3P\_GPIO\_MODE\_MEASURE\_HIGH,  
CY\_U3P\_GPIO\_MODE\_MEASURE\_LOW\_ONCE, CY\_U3P\_GPIO\_MODE\_MEASURE\_HIGH\_ONCE, CY\_U3P\_GPIO\_MODE\_MEASURE\_NEG,  
CY\_U3P\_GPIO\_MODE\_MEASURE\_POS, CY\_U3P\_GPIO\_MODE\_MEASURE\_ANY, CY\_U3P\_GPIO\_MODE\_MEASURE\_NEG\_ONCE, CY\_U3P\_GPIO\_MODE\_MEASURE\_POS\_ONCE, CY\_U3P\_GPIO\_MODE\_MEASURE\_ANY\_ONCE }  
*Enumerated list of all GPIO complex modes.*
- enum [CyU3PGpioTimerMode\\_t](#) {  
CY\_U3P\_GPIO\_TIMER\_SHUTDOWN = 0, CY\_U3P\_GPIO\_TIMER\_HIGH\_FREQ, CY\_U3P\_GPIO\_TIMER\_LOW\_FREQ, CY\_U3P\_GPIO\_TIMER\_STANDBY\_FREQ,  
CY\_U3P\_GPIO\_TIMER\_POS\_EDGE, CY\_U3P\_GPIO\_TIMER\_NEG\_EDGE, CY\_U3P\_GPIO\_TIMER\_AN\_EDGE, CY\_U3P\_GPIO\_TIMER\_RESERVED }  
*List of possible clock sources for GPIO timers.*

- enum `CyU3PGpioIntrMode_t` {
 CY\_U3P\_GPIO\_NO\_INTR = 0, CY\_U3P\_GPIO\_INTR\_POS\_EDGE, CY\_U3P\_GPIO\_INTR\_NEG\_EDGE,
 CY\_U3P\_GPIO\_INTR\_BOTH\_EDGE,
 CY\_U3P\_GPIO\_INTR\_LOW\_LEVEL, CY\_U3P\_GPIO\_INTR\_HIGH\_LEVEL, CY\_U3P\_GPIO\_INTR\_TIME←
 R\_THRES, CY\_U3P\_GPIO\_INTR\_TIMER\_ZERO }

*List of interrupt modes supported by FX3 GPIOs.*

## Functions

- `CyU3PReturnStatus_t CyU3PGpioInit (CyU3PGpioClock_t *clk_p, CyU3PGpioIntrCb_t irq)`  
*Initializes the GPIO Manager.*
- `CyU3PReturnStatus_t CyU3PGpioDeInit (void)`  
*De-initializes the GPIO Manager.*
- `CyU3PReturnStatus_t CyU3PGpioSetSimpleConfig (uint8_t gpioid, CyU3PGpioSimpleConfig_t *cfg_p)`  
*Configures a simple GPIO.*
- `CyU3PReturnStatus_t CyU3PGpioSetComplexConfig (uint8_t gpioid, CyU3PGpioComplexConfig_t *cfg_p)`  
*Configures a complex GPIO pin.*
- `CyU3PReturnStatus_t CyU3PGpioDisable (uint8_t gpioid)`  
*Disables a GPIO pin.*
- `CyU3PReturnStatus_t CyU3PGpioGetValue (uint8_t gpioid, CyBool_t *value_p)`  
*Query the state of GPIO input.*
- `CyU3PReturnStatus_t CyU3PGpioSimpleGetValue (uint8_t gpioid, CyBool_t *value_p)`  
*Query the state of simple GPIO input.*
- `CyU3PReturnStatus_t CyU3PGpioSetValue (uint8_t gpioid, CyBool_t value)`  
*Update the state of a GPIO output.*
- `CyU3PReturnStatus_t CyU3PGpioSimpleSetValue (uint8_t gpioid, CyBool_t value)`  
*Update the state of a simple GPIO output pin.*
- `CyU3PReturnStatus_t CyU3PGpioGetIOValues (uint32_t *gpioVal0_p, uint32_t *gpioVal1_p)`  
*Get the current state of all GPIOs.*
- `CyU3PReturnStatus_t CyU3PGpioComplexUpdate (uint8_t gpioid, uint32_t threshold, uint32_t period)`  
*Update the timer period and threshold associated with a complex GPIO.*
- `CyU3PReturnStatus_t CyU3PGpioComplexGetThreshold (uint8_t gpioid, uint32_t *threshold_p)`  
*Read the threshold value associated with a complex GPIO.*
- `CyU3PReturnStatus_t CyU3PGpioComplexSampleNow (uint8_t gpioid, uint32_t *value_p)`  
*Sample the GPIO timer value at an instant.*
- `CyU3PReturnStatus_t CyU3PGpioComplexPulseNow (uint8_t gpioid, uint32_t threshold)`  
*Drive a pulse on the specified complex GPIO output immediately.*
- `CyU3PReturnStatus_t CyU3PGpioComplexPulse (uint8_t gpioid, uint32_t threshold)`  
*Configures a complex GPIO to drive an output pulse.*
- `CyU3PReturnStatus_t CyU3PGpioComplexMeasureOnce (uint8_t gpioid, CyU3PGpioComplexMode_t pin←Mode)`  
*Initiate an input signal duration measurement.*
- `CyU3PReturnStatus_t CyU3PGpioComplexWaitForCompletion (uint8_t gpioid, uint32_t *threshold_p, CyBool_t isWait)`  
*Wait for the completion of MeasureOnce, Pulse and PulseNow operations.*
- void `CyU3PRegisterGpioCallBack (CyU3PGpioIntrCb_t gpioIntrCb)`  
*Register a callback for notification of GPIO events.*

### 5.20.1 Detailed Description

The GPIO manager module is responsible for handling general purpose IO pins on the FX3 device.

#### Description

Almost any pin of the FX3 device can be used as a General Purpose I/O (GPIO), subject to the condition that it is not being used for any other purpose. The FX3 device supports two classes of GPIOs - Simple and Complex.

Simple GPIOs provide software controlled and observable input and output capability only. The only additional functionality supported on these are interrupt detection on edge or signal level. Any unused pin of the FX3 can be mapped and used as a simple GPIO.

Complex GPIOs are value-added blocks that support additional functions like PWM output, pulsing, time measurements and timers. A maximum of 8 pins on the FX3 can be configured as complex GPIOs, and each of these pins should have a different remainder module 8.

As each GPIO is controlled through a separate set of registers on the FX3 device, they can only be configured individually and multiple GPIOs cannot be updated simultaneously.

### 5.20.2 Typedef Documentation

#### 5.20.2.1 `typedef struct CyU3PGpioComplexConfig_t CyU3PGpioComplexConfig_t`

Configuration information for complex GPIO pins.

#### Description

Complex GPIOs on FX3 can be configured to behave in different ways. All of the parameters that are used for configuring simple GPIOs are also applicable for complex GPIOs.

In addition to these parameters, the pinMode parameter is used to specify the specific complex GPIO functionality desired. The timerMode, timer, period and threshold parameters are used to configure the complex GPIO timer for this pin; in cases where the timer is required.

#### See also

[CyU3PGpioSetComplexConfig](#)  
[CyU3PGpioSimpleConfig\\_t](#)  
[CyU3PGpioComplexMode\\_t](#)  
[CyU3PGpioTimerMode\\_t](#)  
[CyU3PGpioIntrMode\\_t](#)

#### 5.20.2.2 `typedef enum CyU3PGpioComplexMode_t CyU3PGpioComplexMode_t`

Enumerated list of all GPIO complex modes.

#### Description

This enumeration lists the complex GPIO modes supported by the FX3 device.

Of these, the CY\_U3P\_GPIO\_MODE\_SAMPLE\_NOW, CY\_U3P\_GPIO\_MODE\_PULSE\_NOW, CY\_U3P\_GPIO\_MODE\_PULSE, CY\_U3P\_GPIO\_MODE\_MEASURE\_LOW\_ONCE, CY\_U3P\_GPIO\_MODE\_MEASURE\_HI\_ONCE, CY\_U3P\_GPIO\_MODE\_MEASURE\_NEG\_ONCE, CY\_U3P\_GPIO\_MODE\_MEASURE\_POS\_ONCE and CY\_U3P\_GPIO\_MODE\_MEASURE\_ANY\_ONCE modes cannot be directly selected. To use any of these modes, the complex GPIO should be configured with CY\_U3P\_GPIO\_MODE\_STATIC configuration; and then special GPIO APIs need to be used for completing the desired operations.

#### See also

[CyU3PGpioTimerMode\\_t](#)  
[CyU3PGpioIntrMode\\_t](#)

### 5.20.2.3 `typedef void(* CyU3PGpioIntrCb_t)(uint8_t gpioid )`

GPIO callback function type.

#### Description

The GPIO manager in FX3 firmware supports event notifications when a GPIO related interrupt is detected. This data type defines the signature of the callback function that can be registered to receive the event notifications.

#### See also

## See also

[CyU3PGpioClock\\_t](#)  
[CyU3PGpioComplexMode\\_t](#)  
[CyU3PGpioIntrMode\\_t](#)

### 5.20.3 Enumeration Type Documentation

#### 5.20.3.1 enum CyU3PGpioComplexMode\_t

Enumerated list of all GPIO complex modes.

##### Description

This enumeration lists the complex GPIO modes supported by the FX3 device.

Of these, the CY\_U3P\_GPIO\_MODE\_SAMPLE\_NOW, CY\_U3P\_GPIO\_MODE\_PULSE\_NOW, CY\_U3P\_GPIO\_MODE\_PULSE, CY\_U3P\_GPIO\_MODE\_MEASURE\_LOW\_ONCE, CY\_U3P\_GPIO\_MODE\_MEASURE\_HI\_ONCE, CY\_U3P\_GPIO\_MODE\_MEASURE\_NEG\_ONCE, CY\_U3P\_GPIO\_MODE\_MEASURE\_POS\_ONCE and CY\_U3P\_GPIO\_MODE\_MEASURE\_ANY\_ONCE modes cannot be directly selected. To use any of these modes, the complex GPIO should be configured with CY\_U3P\_GPIO\_MODE\_STATIC configuration; and then special GPIO APIs need to be used for completing the desired operations.

## See also

[CyU3PGpioTimerMode\\_t](#)  
[CyU3PGpioIntrMode\\_t](#)

## Enumerator

**CY\_U3P\_GPIO\_MODE\_STATIC** Drives simple static values on GPIO.

**CY\_U3P\_GPIO\_MODE\_TOGGLE** Toggles the output when timer = threshold.

**CY\_U3P\_GPIO\_MODE\_SAMPLE\_NOW** Read current timer value into threshold. This is a one time operation and resets mode to static. This mode should not be set by the configure function. This will be done by the CyU3PGpioComplexSampleNow API.

**CY\_U3P\_GPIO\_MODE\_PULSE\_NOW** Mode used for driving a pulse out on a specified GPIO pin at a specified time. This mode should not be set directly by the configure function. This will be done by the CyU3PGpioComplexPulseNow API.

**CY\_U3P\_GPIO\_MODE\_PULSE** This is similar to the PULSE\_NOW mode, and is used to drive a pulse out when the timer associated with the GPIO reaches 0. This mode should not be set by the configure function and will be done by the CyU3PGpioComplexPulse API.

**CY\_U3P\_GPIO\_MODE\_PWM** Drive a Pulse Width Modulated (PWM) waveform out on the specified GPIO. The ON and OFF times for the signal are defined by the timer period and the threshold value. This is a continuous operation.

**CY\_U3P\_GPIO\_MODE\_MEASURE\_LOW** This mode is used to measure the length of time for which a specified input pin is low. This is a continuous operation which provides the interval duration every time there is a positive edge on the pin.

**CY\_U3P\_GPIO\_MODE\_MEASURE\_HIGH** This mode is similar to the MEASURE\_LOW mode and is used to measure the duration for which an input pin is driver high.

**CY\_U3P\_GPIO\_MODE\_MEASURE\_LOW\_ONCE** This mode is similar to the MEASURE\_LOW mode, except that the pulse duration is measured only once. The mode of the pin will revert to STATIC at the end of the first low pulse. This mode cannot be directly selected and is used internally by the CyU3PGpioComplexMeasureOnce API.

**CY\_U3P\_GPIO\_MODE\_MEASURE\_HIGH\_ONCE** This mode is similar to the MEASURE\_LOW\_ONCE mode, and is used to do a single duration measurement for a high pulse. This mode is also used internally by the CyU3PGpioComplexMeasureOnce API.

**CY\_U3P\_GPIO\_MODE\_MEASURE\_NEG** This mode is used to measure the delay until the specified GPIO input goes low. This is a continuous operation, where the Threshold value is updated with the current timer value at the point where the signal goes low.

**CY\_U3P\_GPIO\_MODE\_MEASURE\_POS** This mode is similar to the MEASURE\_NEG mode, and is a continuous mode which can be used to measure the delay until the GPIO input is driven high.

**CY\_U3P\_GPIO\_MODE\_MEASURE\_ANY** This mode is similar to the MEASURE\_NEG and MEASURE↔\_POS modes; and is used to measure the delay until there is any change in the GPIO input signal.

**CY\_U3P\_GPIO\_MODE\_MEASURE\_NEG\_ONCE** This mode is similar to the MEASURE\_NEG mode, except that only a single delay measurement is performed. This mode cannot be directly selected, and is used internally by the CyU3PGpioComplexMeasureOnce API.

**CY\_U3P\_GPIO\_MODE\_MEASURE\_POS\_ONCE** This mode is similar to the MEASURE\_POS mode, except that only a single delay measurement is performed. This mode cannot be directly selected, and is used internally by the CyU3PGpioComplexMeasureOnce API.

**CY\_U3P\_GPIO\_MODE\_MEASURE\_ANY\_ONCE** This mode is similar to the MEASURE\_POS mode, except that only a single delay measurement is performed. This mode cannot be directly selected, and is used internally by the CyU3PGpioComplexMeasureOnce API.

### 5.20.3.2 enum CyU3PGpioIntrMode\_t

List of interrupt modes supported by FX3 GPIOs.

#### Description

GPIOs on the FX3 device can be configured to interrupt the firmware under various conditions. For simple GPIOs, the interrupt conditions are based on specific input levels or edge detection. Complex GPIOs support additional interrupts based on the timer period and the threshold value.

This enumerated type lists the various interrupt modes supported by FX3 GPIOs.

#### See also

[CyU3PGpioComplexMode\\_t](#)  
[CyU3PGpioTimerMode\\_t](#)

#### Enumerator

**CY\_U3P\_GPIO\_NO\_INTR** GPIO interrupt is unused.

**CY\_U3P\_GPIO\_INTR\_POS\_EDGE** Interrupt is triggered for positive edge of input.

**CY\_U3P\_GPIO\_INTR\_NEG\_EDGE** Interrupt is triggered for negative edge of input.

**CY\_U3P\_GPIO\_INTR\_BOTH\_EDGE** Interrupt is triggered on either edge of input.

**CY\_U3P\_GPIO\_INTR\_LOW\_LEVEL** Interrupt is triggered when the input value is low.

**CY\_U3P\_GPIO\_INTR\_HIGH\_LEVEL** Interrupt is triggered when the input value is high.

**CY\_U3P\_GPIO\_INTR\_TIMER\_THRES** Interrupt is triggered when the timer crosses the threshold. Valid only for complex GPIO pins.

**CY\_U3P\_GPIO\_INTR\_TIMER\_ZERO** Interrupt is triggered when the timer reaches zero. Valid only for complex GPIO pins.

### 5.20.3.3 enum CyU3PGpioTimerMode\_t

List of possible clock sources for GPIO timers.

#### Description

The timer blocks associated with Complex GPIOs can be clocked internally, or based on the input signal connected to the corresponding pin. This enumerated type lists the various clocking options for these GPIO timers.

See also

[CyU3PGpioClock\\_t](#)  
[CyU3PGpioComplexMode\\_t](#)  
[CyU3PGpioIntrMode\\_t](#)

Enumerator

**CY\_U3P\_GPIO\_TIMER\_SHUTDOWN** Timer not in use (no clock applied).  
**CY\_U3P\_GPIO\_TIMER\_HIGH\_FREQ** Use GPIO fast clock.  
**CY\_U3P\_GPIO\_TIMER\_LOW\_FREQ** Use GPIO slow clock.  
**CY\_U3P\_GPIO\_TIMER\_STANDBY\_FREQ** Use FX3 back-up clock (32 KHz).  
**CY\_U3P\_GPIO\_TIMER\_POS\_EDGE** Timer updates on positive edge of input.  
**CY\_U3P\_GPIO\_TIMER\_NEG\_EDGE** Timer updates on negative edge of input.  
**CY\_U3P\_GPIO\_TIMER\_ANY\_EDGE** Timer updates on either edge of input.  
**CY\_U3P\_GPIO\_TIMER\_RESERVED** Reserved for future use.

## 5.20.4 Function Documentation

### 5.20.4.1 CyU3PReturnStatus\_t CyU3PGpioComplexGetThreshold( uint8\_t gpioid, uint32\_t \* threshold\_p )

Read the threshold value associated with a complex GPIO.

#### Description

This function reads the current threshold value associated with a complex GPIO. This function is used in the various MEASURE modes of the GPIO to get the measured pulse/signal duration.

#### Return value

CY\_U3P\_SUCCESS - If the operation is successful.  
CY\_U3P\_ERROR\_NOT\_STARTED - If the GPIO block has not been initialized.  
CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the IO specified is not valid.  
CY\_U3P\_ERROR\_NULL\_POINTER - If the threshold\_p is NULL.  
CY\_U3P\_ERROR\_NOT\_CONFIGURED - If the GPIO is not enabled.

See also

[CyU3PGpioSetComplexConfig](#)  
[CyU3PGpioComplexUpdate](#)  
[CyU3PGpioComplexMeasureOnce](#)

#### Parameters

<i>gpioid</i>	The complex GPIO to sample.
<i>threshold_p</i>	Returns the current threshold value for the GPIO.

### 5.20.4.2 CyU3PReturnStatus\_t CyU3PGpioComplexMeasureOnce( uint8\_t gpioid, CyU3PGpioComplexMode\_t pinMode )

Initiate an input signal duration measurement.

#### Description

The complex GPIOs on FX3 are capable of measuring various duration parameters on the input signal.

CY\_U3P\_GPIO\_MODE\_MEASURE\_LOW\_ONCE : Low period for input.  
CY\_U3P\_GPIO\_MODE\_MEASURE\_HIGH\_ONCE : High period for input.

CY\_U3P\_GPIO\_MODE\_MEASURE\_NEG\_ONCE : Time to negative edge.  
 CY\_U3P\_GPIO\_MODE\_MEASURE\_POS\_ONCE : Time to positive edge.  
 CY\_U3P\_GPIO\_MODE\_MEASURE\_ANY\_ONCE : Time to the next edge (transition).

This API is used to initiate a single measurement of the desired timing parameter on a specified GPIO. This API can only be used when the GPIO is in the CY\_U3P\_GPIO\_MODE\_STATIC mode and the timer is operational.

The API does not wait for the transition that ends the measurement, but returns immediately. The CyU3PGpioComplexWaitForCompletion API can be used to wait until the condition that ends the measurement is reached.

#### Return value

CY\_U3P\_SUCCESS - If the operation is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - If the GPIO block has not been initialized.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If IO or the mode specified is invalid.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - If the IO is not enabled.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - If the GPIO is not configured correctly for this operation.

#### See also

[CyU3PGpioComplexMode\\_t](#)  
[CyU3PGpioSetComplexConfig](#)

#### Parameters

<i>gpioid</i>	The complex GPIO to measure.
<i>pinMode</i>	Type of measurement to be performed.

### 5.20.4.3 CyU3PReturnStatus\_t CyU3PGpioComplexPulse ( uint8\_t *gpioid*, uint32\_t *threshold* )

Configures a complex GPIO to drive an output pulse.

#### Description

This API is a delayed version of the CyU3PGpioComplexPulseNow where the start of the pulse output happens when the GPIO timer reaches 0.

The pulse duration is again determined by the threshold value, and this function returns as soon as it has configured the GPIO to drive the pulse out.

#### Return value

CY\_U3P\_SUCCESS - If the operation is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - If the GPIO block has not been initialized.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the IO or threshold is invalid.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - If the IO is not enabled.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - If the GPIO is not configured correctly for this operation.

#### See also

[CyU3PGpioSetComplexConfig](#)  
[CyU3PGpioComplexPulseNow](#)

#### Parameters

<i>gpioid</i>	The complex GPIO to pulse.
<i>threshold</i>	Updates the threshold value used for setting the pulse duration.

#### 5.20.4.4 CyU3PReturnStatus\_t CyU3PGpioComplexPulseNow ( *uint8\_t gpioid*, *uint32\_t threshold* )

Drive a pulse on the specified complex GPIO output immediately.

##### Description

This function is used to drive a pulse out on the specified complex GPIO, starting immediately. The timer value associated with the GPIO is set to 0 at the beginning of the pulse. The pulse duration is determined by the threshold value and the signal is automatically driven low once the threshold is reached.

This API does not wait until the end of the pulse, but returns as soon as the pulse has been started. The mode for the GPIO is reverted to STATIC as soon as the pulse is completed.

##### Return value

CY\_U3P\_SUCCESS - If the operation is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - If the GPIO block has not been initialized.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the IO or threshold is invalid.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - If the IO is not enabled.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - If the GPIO is not configured correctly for this operation.

##### See also

[CyU3PGpioSetComplexConfig](#)  
[CyU3PGpioComplexPulse](#)

##### Parameters

<i>gpioid</i>	The complex GPIO to pulse.
<i>threshold</i>	Updates the threshold value used for setting the pulse duration.

#### 5.20.4.5 CyU3PReturnStatus\_t CyU3PGpioComplexSampleNow ( *uint8\_t gpioid*, *uint32\_t \* value\_p* )

Sample the GPIO timer value at an instant.

##### Description

This function is used to sample the current value of the GPIO timer at an instant. This can be used to estimate the elapsed time between multiple events processed by the firmware.

To use this feature, the GPIO should be configured as CY\_U3P\_GPIO\_MODE\_STATIC and the timer should be operational.

##### Return value

CY\_U3P\_SUCCESS - If the operation is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - If the GPIO block has not been initialized.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the IO is invalid.  
 CY\_U3P\_ERROR\_NULL\_POINTER - If the *value\_p* is NULL.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - If the GPIO is not enabled.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - Invalid configuration.

##### See also

[CyU3PGpioSetComplexConfig](#)

##### Parameters

<i>gpioid</i>	The complex GPIO to sample.
<i>value_p</i>	Returns the current value of the timer.

#### 5.20.4.6 CyU3PReturnStatus\_t CyU3PGpioComplexUpdate ( *uint8\_t gpioid*, *uint32\_t threshold*, *uint32\_t period* )

Update the timer period and threshold associated with a complex GPIO.

##### Description

The function updates the timer period and threshold value associated with a complex GPIO. This operation is only permitted if the corresponding is configured in the PWM or STATIC modes.

##### Return value

CY\_U3P\_SUCCESS - If the operation is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - If the GPIO block has not been initialized.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the IO pin is invalid.

##### See also

[CyU3PGpioSetComplexConfig](#)  
[CyU3PGpioComplexGetThreshold](#)

##### Parameters

<i>gpioid</i>	The complex GPIO to sample.
<i>threshold</i>	New timer threshold value to be updated.
<i>period</i>	New timer period value to be updated.

#### 5.20.4.7 CyU3PReturnStatus\_t CyU3PGpioComplexWaitForCompletion ( *uint8\_t gpioid*, *uint32\_t \* threshold\_p*, *CyBool\_t isWait* )

Wait for the completion of MeasureOnce, Pulse and PulseNow operations.

##### Description

This function checks or wait for the completion of a previously initiated CyU3PGpioComplexPulse, CyU3PGpioComplexPulseNow or CyU3PGpioComplexMeasureOnce operation.

If the *isWait* option is set to false, the API only checks for completion and returns a TIMEOUT error if it is not complete. If the *isWait* option is set to true, the API enters a busy-wait loop that is only terminated when the GPIO operation is complete.

##### Note

A busy wait is used to wait for the completion of the GPIO operation. Please use this with care, as this can affect the operation of other blocks and drivers in the FX3 firmware.

##### Return value

CY\_U3P\_SUCCESS - If the operation is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - If the GPIO block has not been initialized.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the Drive strength requested is invalid.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - If the IO is not enabled.  
 CY\_U3P\_ERROR\_TIMEOUT - If the operation is not complete at the time of checking.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - If the GPIO is not configured correctly for this operation.

##### See also

[CyU3PGpioComplexPulse](#)  
[CyU3PGpioComplexPulseNow](#)  
[CyU3PGpioComplexMeasureOnce](#)

##### Parameters

<i>gpioid</i>	The complex GPIO to wait for completion.
---------------	--

**Parameters**

<i>threshold</i> <i>_p</i>	The current threshold value after completion.
<i>isWait</i>	Whether to wait or just check for completion.

**5.20.4.8 CyU3PReturnStatus\_t CyU3PGpioDelInit ( void )**

De-initializes the GPIO Manager.

**Description**

De-initialize the GPIO manager in FX3 firmware and power off the GPIO block on the FX3 device.

**Return value**

CY\_U3P\_SUCCESS - If the GPIO de-init is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - If the GPIO block was not previously initialized.

**See also**

[CyU3PGpioInit](#)

**5.20.4.9 CyU3PReturnStatus\_t CyU3PGpioDisable ( uint8\_t gpioId )**

Disables a GPIO pin.

**Description**

This function is used to disable a GPIO pin which was previously configured as a simple or a complex GPIO.

**Return value**

CY\_U3P\_SUCCESS - If the operation is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - If the GPIO block has not been initialized.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the GPIO id is invalid.

**See also**

[CyU3PGpioSetSimpleConfig](#)

[CyU3PGpioSetComplexConfig](#)

**Parameters**

<i>gpio</i> <i>Id</i>	GPIO ID to be disabled
--------------------------	------------------------

**5.20.4.10 CyU3PReturnStatus\_t CyU3PGpioGetIOValues ( uint32\_t \* gpioVal0\_p, uint32\_t \* gpioVal1\_p )**

Get the current state of all GPIOs.

**Description**

This API returns the state of all FX3 IO pins. The state information is returned in the form of a bit vector, where each bit represents the state of the corresponding GPIO pin. The retrieved information is valid only for pins configured as GPIOs.

**Return value**

CY\_U3P\_SUCCESS - If the operation is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - If the GPIO block has not been initialized.

## See also

[CyU3PGpioGetValue](#)  
[CyU3PGpioSimpleGetValue](#)

## Parameters

<i>gpioVal0</i> ↵ — <i>p</i>	Bit vector that represents the states of GPIO 31 : 00.
<i>gpioVal1</i> ↵ — <i>p</i>	Bit vector that represents the states of GPIO 60 : 32. The upper three bits are unused and reserved.

5.20.4.11 CyU3PReturnStatus\_t CyU3PGpioGetValue ( uint8\_t *gpioid*, CyBool\_t \* *value\_p* )

Query the state of GPIO input.

## Description

Get the current state of an input pin configured as a simple or a complex GPIO. This function can also retrieve the last updated outValue for an output pin.

This API can be used to get the current state of an input pin, even if it is configured as an interrupt source.

## Return value

CY\_U3P\_SUCCESS - If the operation is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - If the GPIO block has not been initialized.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the GPIO id is invalid.  
 CY\_U3P\_ERROR\_NULL\_POINTER - If *value\_p* is NULL.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - If the pin is not selected as a GPIO in the IOMATRIX.

## See also

[CyU3PGpioSetSimpleConfig](#)  
[CyU3PGpioSetComplexConfig](#)  
[CyU3PGpioSimpleGetValue](#)  
[CyU3PGpioSetValue](#)  
[CyU3PGpioGetIOValues](#)

## Parameters

<i>gpioid</i>	GPIO id to be queried.
<i>value</i> ↵ — <i>p</i>	Output parameter that will be filled with the GPIO value.

5.20.4.12 CyU3PReturnStatus\_t CyU3PGpioInit ( CyU3PGpioClock\_t \* *clk\_p*, CyU3PGpioIntrCb\_t *irq* )

Initializes the GPIO Manager.

## Description

This function initializes the GPIO manager in the FX3 firmware and powers up the GPIO block on the FX3 device. All other GPIO functions in the SDK can be used only after this function is called.

The clock parameters for the GPIO block are defined through this function. Three different clocks are associated with the FX3 GPIO clock.

The FAST clock is derived from the FX3 system clock, and its frequency is determined by the clkSrc and fastClkDiv parameters. The SLOW clock is derived from the FAST clock and its frequency is determined by the slowClkDiv

value. The operating clock for various complex GPIO timers can be selected from among the FAST clock, the SLOW clock and a fixed 32 KHz frequency.

All simple GPIOs function (are updated or sampled) based on a separate clock which is also derived from the FAST clock. The frequency of this clock is determined by the simpleDiv value.

This function also allows for a GPIO interrupt callback to be registered. This function will be called when any GPIO related interrupt is raised by the device, and will receive the GPIO ID as a parameter. Please note that this interrupt call is made in interrupt context, which means that any blocking API calls cannot be made from this callback.

#### Return value

CY\_U3P\_SUCCESS - If the GPIO initialization is successful.  
 CY\_U3P\_ERROR\_ALREADY\_STARTED - If the GPIO has already been initialized.  
 CY\_U3P\_ERROR\_NULL\_POINTER - If clk\_p is NULL.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If an incorrect / invalid clock value is passed.

#### See also

[CyU3PGpioClock\\_t](#)  
[CyU3PGpioIntrCb\\_t](#)  
[CyU3PGpioDeInit](#)

#### Parameters

<i>clk_p</i>	Clock settings for the GPIO block.
<i>irq</i>	GPIO interrupt callback function.

### 5.20.4.13 CyU3PReturnStatus\_t CyU3PGpioSetComplexConfig ( uint8\_t gpioid, CyU3PGpioComplexConfig\_t \* cfg\_p )

Configures a complex GPIO pin.

#### Description

This function is used to configure and enable a complex GPIO pin. This function needs to be called before using the complex GPIO pin.

A maximum of 8 complex GPIOs can be used on FX3, and the assignment of pin to complex GPIO is done on a MODULO 8 basis. i.e., only one of the pins 0, 8, 16, .. 56 can be used as complex GPIO and so on.

It is possible to use the timer associated with a complex GPIO without actually configuring or using the corresponding pin as a GPIO. In this case, all of the inputEn, driveLowEn and driveHighEn values can be set to CyFalse.

#### Return value

CY\_U3P\_SUCCESS - If the operation is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - If the GPIO block has not been initialized.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If any of the parameters are incorrect/invalid.  
 CY\_U3P\_ERROR\_NULL\_POINTER - If cfg\_p is NULL.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - If the pin has not been selected as a complex GPIO in the IO matrix.

#### See also

[CyU3PGpioComplexMode\\_t](#)  
[CyU3PGpioComplexConfig\\_t](#)  
[CyU3PGpioSetSimpleConfig](#)  
[CyU3PGpioDisable](#)  
[CyU3PGpioComplexUpdate](#)  
[CyU3PGpioComplexGetThreshold](#)  
[CyU3PGpioComplexWaitForCompletion](#)  
[CyU3PGpioComplexSampleNow](#)

[CyU3PGpioComplexPulseNow](#)  
[CyU3PGpioComplexPulse](#)  
[CyU3PGpioComplexMeasureOnce](#)

#### Parameters

<i>gpio← Id</i>	GPIO id to be modified
<i>cfg← _p</i>	Desired GPIO configuration.

#### 5.20.4.14 CyU3PReturnStatus\_t CyU3PGpioSetSimpleConfig ( uint8\_t *gpioid*, CyU3PGpioSimpleConfig\_t \* *cfg\_p* )

Configures a simple GPIO.

#### Description

This function configures and enables a simple GPIO pin. The pin being configured needs to have been selected as a simple GPIO through the CyU3PDeviceConfigureIOMatrix or CyU3PDeviceGpioOverride API calls.

The operating parameters for the GPIO pin as passed in as parameters.

#### Return value

CY\_U3P\_SUCCESS - If the operation is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - If the GPIO block has not been initialized.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If any of the parameters are incorrect/invalid.  
 CY\_U3P\_ERROR\_NULL\_POINTER - If *cfg\_p* is NULL.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - If the pin has not been selected as a GPIO in the IO matrix configuration.

#### See also

[CyU3PGpioSimpleConfig\\_t](#)  
[CyU3PDeviceConfigureIOMatrix](#)  
[CyU3PDeviceGpioOverride](#)  
[CyU3PGpioDisable](#)  
[CyU3PGpioSimpleSetValue](#)  
[CyU3PGpioSimpleGetValue](#)  
[CyU3PGpioSetValue](#)  
[CyU3PGpioGetValue](#)

#### Parameters

<i>gpio← Id</i>	Id of the pin being selected as a GPIO.
<i>cfg← _p</i>	Desired GPIO configuration.

#### 5.20.4.15 CyU3PReturnStatus\_t CyU3PGpioSetValue ( uint8\_t *gpioid*, CyBool\_t *value* )

Update the state of a GPIO output.

#### Description

This function updates the state of a GPIO output pin. The driveLowEn and driveHighEn values set at the time of GPIO configuration will determine whether the pin is actually driven with the desired value or tri-stated.

**Return value**

CY\_U3P\_SUCCESS - If the operation is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - If the GPIO block has not been initialized.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the GPIO id is invalid.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - If the pin is not selected as a GPIO in the IOMATRIX.

**See also**

[CyU3PGpioSetSimpleConfig](#)  
[CyU3PGpioSetComplexConfig](#)  
[CyU3PGpioSimpleSetValue](#)  
[CyU3PGpioGetValue](#)

**Parameters**

<i>gpio</i> ← <i>Id</i>	GPIO id to be modified.
<i>value</i>	Value to set on the GPIO pin.

**5.20.4.16 CyU3PReturnStatus\_t CyU3PGpioSimpleGetValue ( uint8\_t *gpioid*, CyBool\_t \* *value\_p* )**

Query the state of simple GPIO input.

**Description**

Get the current signal state of a simple GPIO configured as an input signal. The CyU3PGpioGetValue API supports all kinds of GPIOs and has a fair amount of conditions checks that slow the operation down. This dedicated API works only on simple GPIOs and will give better GPIO access performance.

This API can be used to get the current state of an input pin, even if it is configured as an interrupt source.

**Return value**

CY\_U3P\_SUCCESS - If the operation is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - If the GPIO block has not been initialized.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the GPIO id is invalid.  
 CY\_U3P\_ERROR\_NULL\_POINTER - If *value\_p* is NULL.

**See also**

[CyU3PGpioSetSimpleConfig](#)  
[CyU3PGpioSimpleSetValue](#)  
[CyU3PGpioGetValue](#)  
[CyU3PGpioGetIOValues](#)

**Parameters**

<i>gpioid</i>	GPIO id to be queried.
<i>value</i> ← <i>_p</i>	Output parameter that will be filled with the GPIO value.

**5.20.4.17 CyU3PReturnStatus\_t CyU3PGpioSimpleSetValue ( uint8\_t *gpioid*, CyBool\_t *value* )**

Update the state of a simple GPIO output pin.

**Description**

This function updates the state of a simple GPIO output pin. This API works only for simple GPIOs and will be faster

than the more generic CyU3PGpioSetValue API call.

#### Return value

CY\_U3P\_SUCCESS - If the operation is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - If the GPIO block has not been initialized.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the GPIO id is invalid.

#### See also

[CyU3PGpioSetSimpleConfig](#)  
[CyU3PGpioSetValue](#)  
[CyU3PGpioSimpleGetValue](#)

#### Parameters

<i>gpioId</i>	GPIO id to be modified.
<i>value</i>	Value to set on the GPIO pin.

### 5.20.4.18 void CyU3PRegisterGpioCallBack ( CyU3PGpioIntrCb\_t gpiointrCb )

Register a callback for notification of GPIO events.

#### Description

This function registers a callback function for notification of GPIO related interrupts. The callback can also be registered through the CyU3PGpioInit API call itself.

#### Return value

None

#### See also

[CyU3PGpioIntrCb\\_t](#)  
[CyU3PGpioInit](#)

#### Parameters

<i>gpiointrCb</i>	Callback function pointer.
-------------------	----------------------------

## 5.21 firmware/u3p\_firmware/inc/cyu3i2c.h File Reference

The I2C driver in the FX3 firmware provides a set of APIs to configure the I2C interface properties and to perform I2C data transfers from/to one or more slave devices.

```
#include <cyu3types.h>
#include <cyu3system.h>
#include <cyu3lpp.h>
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

## Data Structures

- struct [CyU3PI2cConfig\\_t](#)

*Structure defining the I2C interface configuration.*

- struct [CyU3P\\_I2cPreamble\\_t](#)

*Structure defining the preamble to be sent on the I2C interface.*

## Macros

- #define [CY\\_U3P\\_I2C\\_DEFAULT\\_LOCK\\_TIMEOUT](#) (CYU3P\_WAIT\_FOREVER)

*Delay duration to wait to get a lock on the I2C block.*

## Typedefs

- typedef enum [CyU3P\\_I2cEvt\\_t](#) CyU3P\_I2cEvt\_t

*List of I2C related event types.*

- typedef enum [CyU3P\\_I2cError\\_t](#) CyU3P\_I2cError\_t

*List of I2C specific error/status codes.*

- typedef struct [CyU3P\\_I2cConfig\\_t](#) CyU3P\_I2cConfig\_t

*Structure defining the I2C interface configuration.*

- typedef struct [CyU3P\\_I2cPreamble\\_t](#) CyU3P\_I2cPreamble\_t

*Structure defining the preamble to be sent on the I2C interface.*

- typedef void(\* [CyU3P\\_I2cIntrCb\\_t](#)) ([CyU3P\\_I2cEvt\\_t](#) evt, [CyU3P\\_I2cError\\_t](#) error)

*I2C event callback function type.*

## Enumerations

- enum [CyU3P\\_I2cEvt\\_t](#) {

```
CY_U3P_I2C_EVENT_RX_DONE = 0, CY_U3P_I2C_EVENT_TX_DONE, CY_U3P_I2C_EVENT_TIMEOUT,
CY_U3P_I2C_EVENT_LOST_ARBITRATION,
CY_U3P_I2C_EVENT_ERROR }
```

*List of I2C related event types.*

- enum [CyU3P\\_I2cError\\_t](#) {

```
CY_U3P_I2C_ERROR_NAK_BYTE_0 = 0, CY_U3P_I2C_ERROR_NAK_BYTE_1, CY_U3P_I2C_ERROR_NAK_BYTE_2,
CY_U3P_I2C_ERROR_NAK_BYTE_3, CY_U3P_I2C_ERROR_NAK_BYTE_4, CY_U3P_I2C_ERROR_NAK_BYTE_5, CY_U3P_I2C_ERROR_NAK_BYTE_6,
CY_U3P_I2C_ERROR_NAK_BYTE_7, CY_U3P_I2C_ERROR_NAK_DATA, CY_U3P_I2C_ERROR_PREAMBLE_EXIT_NACK_ACK, CY_U3P_I2C_ERROR_PREAMBLE_EXIT,
CY_U3P_I2C_ERROR_NAK_TX_UNDERFLOW, CY_U3P_I2C_ERROR_NAK_TX_OVERFLOW, CY_U3P_I2C_ERROR_NAK_RX_UNDERFLOW, CY_U3P_I2C_ERROR_NAK_RX_OVERFLOW }
```

*List of I2C specific error/status codes.*

## Functions

- [CyU3PReturnStatus\\_t CyU3P\\_I2cInit](#) (void)

*Starts the I2C interface block on the FX3.*

- [CyU3PReturnStatus\\_t CyU3P\\_I2cDeInit](#) (void)

*Stops the I2C module.*

- [CyU3PReturnStatus\\_t CyU3P\\_I2cSetConfig](#) ([CyU3P\\_I2cConfig\\_t](#) \*config, [CyU3P\\_I2cIntrCb\\_t](#) cb)

*Sets the I2C interface parameters.*

- [CyU3PReturnStatus\\_t CyU3P\\_I2cSendCommand](#) ([CyU3P\\_I2cPreamble\\_t](#) \*preamble, [uint32\\_t](#) byteCount, [CyBool\\_t](#) isRead)

*Initiate a read or write operation to the I2C slave.*

- [CyU3PReturnStatus\\_t CyU3PI2cTransmitBytes](#) ([CyU3PI2cPreamble\\_t](#) \*preamble, [uint8\\_t](#) \*data, [uint32\\_t](#) byteCount, [uint32\\_t](#) retryCount)  
*Writes data to an I2C Slave in register mode.*
- [CyU3PReturnStatus\\_t CyU3PI2cReceiveBytes](#) ([CyU3PI2cPreamble\\_t](#) \*preamble, [uint8\\_t](#) \*data, [uint32\\_t](#) byteCount, [uint32\\_t](#) retryCount)  
*Read data from the I2C slave in register mode.*
- [CyU3PReturnStatus\\_t CyU3PI2cWaitForAck](#) ([CyU3PI2cPreamble\\_t](#) \*preamble, [uint32\\_t](#) retryCount)  
*Wait until the I2C slave ACKs the preamble.*
- [CyU3PReturnStatus\\_t CyU3PI2cWaitForBlockXfer](#) ([CyBool\\_t](#) isRead)  
*Wait until the ongoing I2C data transfer is finished.*
- [CyU3PReturnStatus\\_t CyU3PI2cGetErrorCode](#) ([CyU3PI2cError\\_t](#) \*error\_p)  
*Retrieves the error code as defined by [CyU3PI2cError\\_t](#).*
- [void CyU3PRegisterI2cCallBack](#) ([CyU3PI2cIntrCb\\_t](#) i2cIntrCb)  
*Register a callback function for I2C event notifications.*
- [CyU3PReturnStatus\\_t CyU3PI2cSetTimeout](#) ([uint32\\_t](#) readLoopCnt, [uint32\\_t](#) writeLoopCnt, [uint32\\_t](#) flushLoopCnt)  
*Set the timeout duration for I2C read/write transfer APIs.*
- [CyU3PReturnStatus\\_t CyU3PI2cSendStopCondition](#) (void)  
*Send a stop condition on the I2C bus.*

## 5.21.1 Detailed Description

The I2C driver in the FX3 firmware provides a set of APIs to configure the I2C interface properties and to perform I2C data transfers from/to one or more slave devices.

## 5.21.2 Typedef Documentation

### 5.21.2.1 [typedef struct CyU3PI2cConfig\\_t CyU3PI2cConfig\\_t](#)

Structure defining the I2C interface configuration.

#### Description

This structure encapsulates all of the configurable parameters that can be selected for the I2C interface. The [CyU3PI2cSetConfig\(\)](#) function accepts a pointer to this structure, and updates all of the interface parameters.

The I2C block can function in the bit rate range of 100 KHz to 1MHz. In the default mode of operation, the timeouts need to be kept disabled.

In the register mode of operation (isDma is false), the data transfer APIs are blocking and return only after the requested amount of data has been read or written. In such a case, the I2C specific callbacks are meaningless; and the [CyU3PI2cSetConfig](#) API expects that no callback is specified when the register mode is selected.

#### See also

[CyU3PI2cSetConfig](#)

### 5.21.2.2 [typedef enum CyU3PI2cError\\_t CyU3PI2cError\\_t](#)

List of I2C specific error/status codes.

#### Description

This type lists the various I2C specific error/status codes that are sent to the event callback as event data, when the event type is CY\_U3P\_I2C\_ERROR\_EVT.

**See also**

[CyU3PI2cEvt\\_t](#)  
[CyU3PI2cIntrCb\\_t](#)

**5.21.2.3 `typedef enum CyU3PI2cEvt_t CyU3PI2cEvt_t`**

List of I2C related event types.

**Description**

This enumerated type lists the various I2C related event codes that are sent to the user application through the I2C event callback.

**Note**

In the case of a DMA read of data that does not fill the DMA buffer(s) associated with the read DMA channel, the DMA transfer remains pending after the CY\_U3P\_I2C\_EVENT\_RX\_DONE event is delivered. The data can only be retrieved from the DMA buffer after the DMA transfer is terminated through the CyU3PDmaChannelSetWrapUp API.

**See also**

[CyU3PI2cError\\_t](#)  
[CyU3PI2cIntrCb\\_t](#)

**5.21.2.4 `typedef void(* CyU3PI2cIntrCb_t)(CyU3PI2cEvt_t evt, CyU3PI2cError_t error )`**

I2C event callback function type.

**Description**

This type defines the signature of the callback functions to be registered to receive I2C related events. I2C event callbacks can only be used in the DMA mode of data transfer.

**See also**

[CyU3PI2cEvt\\_t](#)  
[CyU3PI2cError\\_t](#)  
[CyU3PRegisterI2cCallBack](#)

**5.21.2.5 `typedef struct CyU3PI2cPreamble_t CyU3PI2cPreamble_t`**

Structure defining the preamble to be sent on the I2C interface.

**Description**

All I2C data transfers start with a preamble where the first byte contains the slave address and the direction of transfer. The preamble can optionally contain other bytes where device specific address values or other commands are sent to the slave device.

The FX3 device supports associating a preamble with a maximum length of 8 bytes to any I2C data transfer. This allows the user to specify a multi-byte preamble which covers the slave address, device specific address fields and then initiate the data transfer.

The ctrlMask indicate the start / stop bit conditions after each byte of the preamble.

For example, an I2C EEPROM read requires the byte address for the read to be written first. These two I2C operations can be combined into one I2C API call using the parameters of the structure.

Typical I2C EEPROM page Read operation:

Byte 0:

Bit 7 - 1: Slave address.

Bit 0 : 0 - Indicating this is a write from master.

Byte 1, 2: Address to which the data has to be written.

Byte 3:

Bit 7 - 1: Slave address.

Bit 0 : 1 - Indicating this is a read operation.

The buffer field shall hold the above four bytes, the length field shall be four; and ctrlMask field will be 0x0004 as a start bit is required after the third byte (third bit is set).

Typical I2C EEPROM page Write operation:

Byte 0:

Bit 7 - 1: Slave address.

Bit 0 : 0 - Indicating this is a write from master.

Byte 1, 2: Address to which the data has to be written.

The buffer field shall hold the above three bytes, the length field shall be three, and the ctrlMask field is zero as no additional start/stop conditions are needed.

Please note that the user is expected to set direction bit in the preamble bytes properly based on the type of transfer to be performed. The API does not check whether the preamble direction matches the type of transfer API being called.

See also

[CyU3P\\_I2cTransmitBytes](#)

[CyU3P\\_I2cReceiveBytes](#)

### 5.21.3 Enumeration Type Documentation

#### 5.21.3.1 enum CyU3P\_I2cError\_t

List of I2C specific error/status codes.

##### Description

This type lists the various I2C specific error/status codes that are sent to the event callback as event data, when the event type is CY\_U3P\_I2C\_ERROR\_EVT.

See also

[CyU3P\\_I2cEvt\\_t](#)

[CyU3P\\_I2cIntrCb\\_t](#)

##### Enumerator

**CY\_U3P\_I2C\_ERROR\_NAK\_BYTE\_0** Slave NACK-ed the zeroth byte of the preamble.

**CY\_U3P\_I2C\_ERROR\_NAK\_BYTE\_1** Slave NACK-ed the first byte of the preamble.

**CY\_U3P\_I2C\_ERROR\_NAK\_BYTE\_2** Slave NACK-ed the second byte of the preamble.

**CY\_U3P\_I2C\_ERROR\_NAK\_BYTE\_3** Slave NACK-ed the third byte of the preamble.

**CY\_U3P\_I2C\_ERROR\_NAK\_BYTE\_4** Slave NACK-ed the fourth byte of the preamble.

**CY\_U3P\_I2C\_ERROR\_NAK\_BYTE\_5** Slave NACK-ed the fifth byte of the preamble.

**CY\_U3P\_I2C\_ERROR\_NAK\_BYTE\_6** Slave NACK-ed the sixth byte of the preamble.

**CY\_U3P\_I2C\_ERROR\_NAK\_BYTE\_7** Slave NACK-ed the seventh byte of the preamble.

**CY\_U3P\_I2C\_ERROR\_NAK\_DATA** Slave sent a NACK during the data phase of a transfer.

**CY\_U3P\_I2C\_ERROR\_PREAMBLE\_EXIT\_NACK\_ACK** Poll operation has exited due to the slave returning an ACK or a NACK handshake.

**CY\_U3P\_I2C\_ERROR\_PREAMBLE\_EXIT** Poll operation with address repetition timed out.

**CY\_U3P\_I2C\_ERROR\_NAK\_TX\_UNDERFLOW** Underflow in buffer during transmit/write operation.

**CY\_U3P\_I2C\_ERROR\_NAK\_TX\_OVERFLOW** Overflow of buffer during transmit operation.

**CY\_U3P\_I2C\_ERROR\_NAK\_RX\_UNDERFLOW** Underflow in buffer during receive/read operation.

**CY\_U3P\_I2C\_ERROR\_NAK\_RX\_OVERFLOW** Overflow of buffer during receive operation.

### 5.21.3.2 enum CyU3PI2cEvt\_t

List of I2C related event types.

#### Description

This enumerated type lists the various I2C related event codes that are sent to the user application through the I2C event callback.

#### Note

In the case of a DMA read of data that does not fill the DMA buffer(s) associated with the read DMA channel, the DMA transfer remains pending after the CY\_U3P\_I2C\_EVENT\_RX\_DONE event is delivered. The data can only be retrieved from the DMA buffer after the DMA transfer is terminated through the CyU3PDmaChannelSetWrapUp API.

See also

[CyU3PI2cError\\_t](#)  
[CyU3PI2cIntrCb\\_t](#)

Enumerator

**CY\_U3P\_I2C\_EVENT\_RX\_DONE** Reception is completed  
**CY\_U3P\_I2C\_EVENT\_TX\_DONE** Transmission is done  
**CY\_U3P\_I2C\_EVENT\_TIMEOUT** Bus timeout has happened  
**CY\_U3P\_I2C\_EVENT\_LOST\_ARBITRATION** Lost I2C bus arbitration, probably due to another I2C bus master.  
**CY\_U3P\_I2C\_EVENT\_ERROR** I2C transfer has resulted in an error.

## 5.21.4 Function Documentation

### 5.21.4.1 CyU3PReturnStatus\_t CyU3PI2cDelInit( void )

Stops the I2C module.

#### Description

This function disables and powers off the I2C interface. This function can be used to shut off the interface, to save power when it is not in use.

#### Return value

CY\_U3P\_SUCCESS - If the DelInit is successful.  
CY\_U3P\_ERROR\_NOT\_STARTED - If the I2C module has not been previously initialized.

See also

[CyU3PI2cInit](#)

### 5.21.4.2 CyU3PReturnStatus\_t CyU3PI2cGetErrorCode( CyU3PI2cError\_t \* error\_p )

Retrieves the error code as defined by CyU3PI2cError\_t.

#### Description

This function can be used to retrieve the actual I2C error code once the register mode I2C transfer APIs have returned the CY\_U3P\_ERROR\_FAILURE error code.

#### Return value

CY\_U3P\_SUCCESS - If the error code is fetched successfully.  
CY\_U3P\_ERROR\_NULL\_POINTER - When the error\_p pointer passed is NULL.  
CY\_U3P\_ERROR\_NOT\_STARTED - When there is no error flagged.  
CY\_U3P\_ERROR\_MUTEX\_FAILURE - Failure to obtain a lock on the I2C block.

## See also

[CyU3PI2cTransmitBytes](#)  
[CyU3PI2cReceiveBytes](#)  
[CyU3PI2cWaitForAck](#)

## Parameters

<code>error ← _p</code>	Return pointer to be filled with the error code.
-----------------------------	--

5.21.4.3 `CyU3PReturnStatus_t CyU3PI2cInit( void )`

Starts the I2C interface block on the FX3.

**Description**

This function powers up the I2C interface block on the FX3 device and is expected to be the first I2C API function that is called by the application. This function also sets up the I2C interface at a default rate of 100KHz.

**Return value**

`CY_U3P_SUCCESS` - When the Init is successful.  
`CY_U3P_ERROR_NOT_CONFIGURED` - When I2C has not been enabled in IO configuration.  
`CY_U3P_ERROR_ALREADY_STARTED` - When the I2C has been already initialized.

## See also

[CyU3PI2cDelInit](#)  
[CyU3PI2cSetConfig](#)

5.21.4.4 `CyU3PReturnStatus_t CyU3PI2cReceiveBytes( CyU3PI2cPreamble_t * preamble, uint8_t * data, uint32_t byteCount, uint32_t retryCount )`

Read data from the I2C slave in register mode.

**Description**

This function is used to read data one byte at a time from an I2C slave. This function requires that the I2C interface be configured in register (non-DMA) mode. The function call can be repeated when `CY_U3P_ERROR_TIMEOUT` is returned without any error recovery. The retry is done continuously without any delay. If any delay is required, then it should be added by the caller.

The API will return when FX3 has received the data. If the slave device requires additional time before responding to other commands, then either sufficient delay must be provided; or the `CyU3PI2cWaitForAck` API can be used. Refer to the I2C slave datasheet for more details on how to identify when the slave is ready.

This function internally uses the `CyU3PI2cSendCommand` function.

**Return value**

`CY_U3P_SUCCESS` - When the transfer is completed successfully.  
`CY_U3P_ERROR_NULL_POINTER` - If the preamble or data parameter is NULL.  
`CY_U3P_ERROR_FAILURE` - When a transfer fails with an error defined in `CyU3PI2cError_t`.  
`CY_U3P_ERROR_BLOCK_FAILURE` - When the I2C block encounters a fatal error and requires to be re-initialized.  
`CY_U3P_ERROR_NOT_CONFIGURED` - If the I2C block is not configured for register mode transfers.  
`CY_U3P_ERROR_TIMEOUT` - I2C bus timeout occurred.  
`CY_U3P_ERROR_LOST_ARBITRATION` - Failure due to I2C arbitration error or invalid bus activity.  
`CY_U3P_ERROR_MUTEX_FAILURE` - Failure to obtain a lock on the I2C block.

## See also

[CyU3PI2cPreamble\\_t](#)  
[CyU3PI2cSetConfig](#)  
[CyU3PI2cSendCommand](#)  
[CyU3PI2cTransmitBytes](#)  
[CyU3PI2cWaitForAck](#)

## Parameters

<i>preamble</i>	Preamble information to be sent out before the data transfer.
<i>data</i>	Pointer to buffer where the data is to be placed.
<i>byteCount</i>	Size of the transfer in bytes.
<i>retryCount</i>	Number of times to retry request in case of a NAK response or error.

#### 5.21.4.5 CyU3PReturnStatus\_t CyU3PI2cSendCommand ( CyU3PI2cPreamble\_t \* *preamble*, uint32\_t *byteCount*, CyBool\_t *isRead* )

Initiate a read or write operation to the I2C slave.

**Description**

This function is used to send the extended preamble over the I2C bus. This is used in conjunction with data transfer phase in DMA mode. The function is also called from the API library for register mode operation.

The byteCount represents the amount of data to be read or written in the data phase and the isRead parameter specifies the direction of transfer. The transfer will happen through the I2C Consumer / Producer DMA channel if the I2C interface is configured in DMA mode, or through the I2C Ingress/Egress registers if the interface is configured in register mode.

The CyU3PI2cWaitForBlockXfer API or the CY\_U3P\_I2C\_EVENT\_RX\_DONE or CY\_U3P\_I2C\_EVENT\_TX\_DONE event callbacks can be used to detect the end of a DMA data transfer that is requested through this function.

**Return value**

CY\_U3P\_SUCCESS - When the SendCommand is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - When the preamble is NULL.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the preamble or byteCount arguments are invalid.  
 CY\_U3P\_ERROR\_TIMEOUT - When the I2C bus is busy.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - When there is a failure in acquiring a mutex lock.

## See also

[CyU3PI2cPreamble\\_t](#)  
[CyU3PI2cSetConfig](#)  
[CyU3PI2cTransmitBytes](#)  
[CyU3PI2cReceiveBytes](#)  
[CyU3PI2cWaitForAck](#)  
[CyU3PI2cWaitForBlockXfer](#)

## Parameters

<i>preamble</i>	Preamble information to be sent out before the data transfer.
<i>byteCount</i>	Size of the transfer in bytes.
<i>isRead</i>	Direction of transfer: CyTrue: Read, CyFalse: Write.

#### 5.21.4.6 CyU3PReturnStatus\_t CyU3PI2cSendStopCondition ( void )

Send a stop condition on the I2C bus.

##### Description

This function overrides the I2C pins of the FX3 as GPIOs and drives a stop condition on the bus. This function requires the GPIO module to be initialized, and will return an error otherwise.

##### Return value

CY\_U3P\_SUCCESS if the stop sending is successful. CY\_U3P\_ERROR\_NOT\_STARTED if the GPIO module has not been started.

#### 5.21.4.7 CyU3PReturnStatus\_t CyU3PI2cSetConfig ( CyU3PI2cConfig\_t \* config, CyU3PI2cIntrCb\_t cb )

Sets the I2C interface parameters.

##### Description

This function is used to configure the I2C master interface based on the desired baud rate and address length settings to talk to the desired slave.

This function should be called repeatedly to change the settings if different settings are to be used to communicate with different slave devices. This can be called on the fly repetitively without calling CyU3PI2cInit.

In DMA mode, the callback parameter "cb" passed to this function is used to notify the user of data transfer completion or error conditions. In register mode, all APIs are blocking in nature. So in these cases, the callback argument is not relevant and the user must pass NULL as cb when using the register mode.

Bitrate calculation: The maximum bitrate supported is 1 MHz and minimum bitrate is 100 KHz. It should be noted that even though the dividers and the API allows frequencies above and below the rated range, the device behaviour is not guaranteed.

The I2C block on the FX3 needs to be clocked at 10X the desired bit rate. As the I2C block clock is derived from the FX3 SYSTEM clock, the actual bit rate will not be the exact programmed value. As the hardware only supports clock division by integer and half dividers, the firmware uses the closest approximation possible.

##### Return value

CY\_U3P\_SUCCESS - When the SetConfig is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - When the I2C has not been initialized.

CY\_U3P\_ERROR\_NULL\_POINTER - When the config parameter is NULL.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - When the arguments are incorrect.

CY\_U3P\_ERROR\_TIMEOUT - When there is timeout happening during configuration.

CY\_U3P\_ERROR\_MUTEX\_FAILURE - When there is a failure in acquiring a mutex lock.

##### See also

[CyU3PI2cIntrCb\\_t](#) [CyU3PI2cConfig\\_t](#) [CyU3PI2cInit](#) [CyU3PI2cSendCommand](#) [CyU3PI2cTransmitBytes](#) [CyU3PI2cReceiveBytes](#)

##### Parameters

<i>config</i>	I2C configuration settings
<i>cb</i>	Callback for getting the events

#### 5.21.4.8 CyU3PReturnStatus\_t CyU3PI2cSetTimeout ( uint32\_t readLoopCnt, uint32\_t writeLoopCnt, uint32\_t flushLoopCnt )

Set the timeout duration for I2C read/write transfer APIs.

##### Description

The CyU3PI2cTransmitBytes, CyU3PI2cReceiveBytes, CyU3PI2cWaitForAck and CyU3PI2cWaitForBlockXfer APIs.

Pls wait for the completion of the requested data transfer. The default timeout is 1 million loops with a delay of about 1 us.

The default timeout for read, write and error recovery operations can be changed using this API. The loop count to be used during read and write operations can be set to different values.

#### Return value

CY\_U3P\_SUCCESS if the timeout duration is updated. CY\_U3P\_ERROR\_NOT\_STARTED if the I2C block has not been initialized.

#### See also

[CyU3PI2cTransmitBytes](#)  
[CyU3PI2cReceiveBytes](#)  
[CyU3PI2cWaitForAck](#)  
[CyU3PI2cWaitForBlockXfer](#)

#### Parameters

<i>readLoopCnt</i>	Wait loop count used during read operations.
<i>writeLoopCnt</i>	Wait loop count used during write operations.
<i>flushLoopCnt</i>	Wait loop count used during flush (error recovery) operations.

### 5.21.4.9 CyU3PReturnStatus\_t CyU3PI2cTransmitBytes ( CyU3PI2cPreamble\_t \* preamble, uint8\_t \* data, uint32\_t byteCount, uint32\_t retryCount )

Writes data to an I2C Slave in register mode.

#### Description

This function is used to write data one byte at a time to an I2C slave. This function requires that the I2C interface be configured in register (non-DMA) mode. The function call can be repeated when CY\_U3P\_ERROR\_TIMEOUT is returned without any error recovery. The retry is done continuously without any delay. If any delay is required, then it should be added by the caller.

The API will return when FX3 has transmitted the data. If the slave device requires additional time for completing the write operation, then either sufficient delay must be provided; or the CyU3PI2cWaitForAck API can be used. Refer to the I2C slave datasheet for more details on how to identify when the write is complete.

This function internally uses the CyU3PI2cSendCommand function.

#### Return value

CY\_U3P\_SUCCESS - When the TransmitBytes is successful.  
 CY\_U3P\_ERROR\_NULL\_POINTER - If the preamble or data parameter is NULL.  
 CY\_U3P\_ERROR\_FAILURE - When a transfer fails with an error defined in CyU3PI2cError\_t.  
 CY\_U3P\_ERROR\_BLOCK\_FAILURE - When the I2C block encounters a fatal error and requires to be re-initialized.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - If the I2C block is not configured for register mode transfers.  
 CY\_U3P\_ERROR\_TIMEOUT - I2C bus timeout occurred.  
 CY\_U3P\_ERROR\_LOST\_ARBITRATION - Failure due to I2C arbitration error or invalid bus activity.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - Failure to obtain a lock on the I2C block.

#### See also

[CyU3PI2cPreamble\\_t](#)  
[CyU3PI2cSetConfig](#)  
[CyU3PI2cSendCommand](#)  
[CyU3PI2cReceiveBytes](#)  
[CyU3PI2cWaitForAck](#)

**Parameters**

<i>preamble</i>	Preamble information to be sent out before the data transfer.
<i>data</i>	Pointer to buffer containing data to be written.
<i>byteCount</i>	Size of the transfer in bytes.
<i>retryCount</i>	Number of times to retry request in case of a slave NAK response.

**5.21.4.10 CyU3PReturnStatus\_t CyU3PI2cWaitForAck ( CyU3PI2cPreamble\_t \* *preamble*, uint32\_t *retryCount* )**

Wait until the I2C slave ACKs the preamble.

**Description**

This function waits for a ACK handshake from the slave, and can be used to ensure that the slave device has reached a desired state before issuing the next transaction or shutting the interface down.

The API repeats the provided preamble bytes continuously until all bytes of the preamble are ACKed, or the specified *retryCount* has been reached.

**Return value**

CY\_U3P\_SUCCESS - When the device returns the desired handshake.  
 CY\_U3P\_ERROR\_NULL\_POINTER - If the preamble is NULL.  
 CY\_U3P\_ERROR\_FAILURE - When a preamble transfer fails with error defined by CyU3PI2cError\_t.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - When the I2C is not initialized or not configured.  
 CY\_U3P\_ERROR\_BLOCK\_FAILURE - When the I2C block encounters a fatal error and requires to be re-initialized.  
 CY\_U3P\_ERROR\_TIMEOUT - I2C bus timeout occurred.  
 CY\_U3P\_ERROR\_LOST\_ARBITRATION - Failure due to I2C arbitration error or invalid bus activity.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - Failure to obtain a lock on the I2C block.

**See also**

[CyU3PI2cPreamble\\_t](#)  
[CyU3PI2cSendCommand](#)  
[CyU3PI2cTransmitBytes](#)  
[CyU3PI2cReceiveBytes](#)

**Parameters**

<i>preamble</i>	Preamble information to be sent out before the data transfer.
<i>retryCount</i>	Number of times to retry request if the slave continues to NAK.

**5.21.4.11 CyU3PReturnStatus\_t CyU3PI2cWaitForBlockXfer ( CyBool\_t *isRead* )**

Wait until the ongoing I2C data transfer is finished.

**Description**

This function waits until the ongoing DMA based I2C transaction is complete. The function returns when FX3 has finished transferring the requested amount of data.

**Note**

In the case of a DMA read of data that does not fill the DMA buffer(s) associated with the read DMA channel, the DMA transfer remains pending after this API call returns. The data can only be retrieved from the DMA buffer after the DMA transfer is terminated through the CyU3PDmaChannelSetWrapUp API.

**Return value**

CY\_U3P\_SUCCESS - When the data transfer has been completed successfully.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - When the I2C is not initialized or not configured.

CY\_U3P\_ERROR\_NOT\_SUPPORTED - If a I2C callback function has been registered.  
 CY\_U3P\_ERROR\_FAILURE - When there is a failure defined by CyU3PI2cError\_t.  
 CY\_U3P\_ERROR\_BLOCK\_FAILURE - When the I2C block encounters a fatal error and requires to be re-initialized.  
 CY\_U3P\_ERROR\_TIMEOUT - I2C bus timeout occurred.  
 CY\_U3P\_ERROR\_LOST\_ARBITRATION - Failure due to I2C arbitration error or invalid bus activity.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - Failure to obtain a lock on the I2C block.

#### See also

[CyU3PI2cSendCommand](#)

#### Parameters

<i>isRead</i>	Type of operation to wait on: CyTrue: Read, CyFalse: Write
---------------	--

### 5.21.4.12 void CyU3PRegisterI2cCallBack ( CyU3PI2cIntrCb\_t i2cIntrCb )

Register a callback function for I2C event notifications.

#### Description

This function registers a callback function that will be called for notification of I2C interrupts.

#### Return value

None

#### See also

[CyU3PI2cIntrCb\\_t](#)

#### Parameters

<i>i2cIntrCb</i>	Callback function pointer.
------------------	----------------------------

## 5.22 firmware/u3p\_firmware/inc/cyu3i2s.h File Reference

The I2S (Inter-IC Sound) interface is a serial interface defined for communication of stereophonic audio data between devices. The FX3 device includes a I2S master interface which can be connected to an I2S peripheral. The I2S driver module provides functions to configure the I2S interface and to send mono or stereo audio output on the I2S link.

```
#include "cyu3types.h"
#include "cyu3lpp.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

### Data Structures

- struct [CyU3PI2sConfig\\_t](#)

*I2S interface configuration parameters.*

## Macros

- `#define CY_U3P_I2S_DEFAULT_LOCK_TIMEOUT (CYU3P_WAIT_FOREVER)`  
*Delay duration to wait to get a lock on the I2S block.*

## Typedefs

- `typedef enum CyU3PI2sEvt_t CyU3PI2sEvt_t`  
*List of I2S related event types.*
- `typedef enum CyU3PI2sError_t CyU3PI2sError_t`  
*List of I2S specific error/status codes.*
- `typedef enum CyU3PI2sSampleWidth_t CyU3PI2sSampleWidth_t`  
*List of supported bit widths for the I2S interface.*
- `typedef enum CyU3PI2sSampleRate_t CyU3PI2sSampleRate_t`  
*List of supported sample rates.*
- `typedef enum CyU3PI2sPadMode_t CyU3PI2sPadMode_t`  
*List of the supported padding modes.*
- `typedef struct CyU3PI2sConfig_t CyU3PI2sConfig_t`  
*I2S interface configuration parameters.*
- `typedef void(* CyU3PI2sIntrCb_t) (CyU3PI2sEvt_t evt, CyU3PI2sError_t error)`  
*Prototype of I2S event callback function.*

## Enumerations

- `enum CyU3PI2sEvt_t { CY_U3P_I2S_EVENT_TXL_DONE = 0, CY_U3P_I2S_EVENT_TXR_DONE, CY_U3P_I2S_EVENT_PAUSED, CY_U3P_I2S_EVENT_ERROR }`  
*List of I2S related event types.*
- `enum CyU3PI2sError_t { CY_U3P_I2S_ERROR_LTX_UNDERFLOW = 11, CY_U3P_I2S_ERROR_RTX_UNDERFLOW, CY_U3P_I2S_ERROR_LTX_OVERFLOW, CY_U3P_I2S_ERROR_RTX_OVERFLOW }`  
*List of I2S specific error/status codes.*
- `enum CyU3PI2sSampleWidth_t { CY_U3P_I2S_WIDTH_8_BIT = 0, CY_U3P_I2S_WIDTH_16_BIT, CY_U3P_I2S_WIDTH_18_BIT, CY_U3P_I2S_WIDTH_24_BIT, CY_U3P_I2S_WIDTH_32_BIT, CY_U3P_I2S_NUM_BIT_WIDTH }`  
*List of supported bit widths for the I2S interface.*
- `enum CyU3PI2sSampleRate_t { CY_U3P_I2S_SAMPLE_RATE_8KHz = 8000, CY_U3P_I2S_SAMPLE_RATE_16KHz = 16000, CY_U3P_I2S_SAMPLE_RATE_32KHz = 32000, CY_U3P_I2S_SAMPLE_RATE_44_1KHz = 44100, CY_U3P_I2S_SAMPLE_RATE_48KHz = 48000, CY_U3P_I2S_SAMPLE_RATE_96KHz = 96000, CY_U3P_I2S_SAMPLE_RATE_192KHz = 192000 }`  
*List of supported sample rates.*
- `enum CyU3PI2sPadMode_t { CY_U3P_I2S_PAD_MODE_NORMAL = 0, CY_U3P_I2S_PAD_MODE_LEFT_JUSTIFIED, CY_U3P_I2S_PAD_MODE_RIGHT_JUSTIFIED, CY_U3P_I2S_PAD_MODE_RESERVED, CY_U3P_I2S_PAD_MODE_CONTINUOUS, CY_U2P_I2S_NUM_PAD_MODES }`  
*List of the supported padding modes.*

## Functions

- [CyU3PReturnStatus\\_t CyU3PI2sEnableExternalMclk \(void\)](#)  
*Select an external input as the MCLK for the I2S interface.*
- [CyU3PReturnStatus\\_t CyU3PI2sInit \(void\)](#)  
*Starts the I2S interface block on the FX3.*
- [CyU3PReturnStatus\\_t CyU3PI2sDelInit \(void\)](#)  
*Stops the I2S interface block on the FX3.*
- [CyU3PReturnStatus\\_t CyU3PI2sSetConfig \(CyU3PI2sConfig\\_t \\*config, CyU3PI2sIntrCb\\_t cb\)](#)  
*Sets the I2S interface parameters.*
- [CyU3PReturnStatus\\_t CyU3PI2sTransmitBytes \(uint8\\_t \\*lData, uint8\\_t \\*rData, uint8\\_t lByteCount, uint8\\_t rByteCount\)](#)  
*Transmits data byte by byte over the I2S interface.*
- [CyU3PReturnStatus\\_t CyU3PI2sSetMute \(CyBool\\_t isMute\)](#)  
*Mute or unmute the I2S output stream.*
- [CyU3PReturnStatus\\_t CyU3PI2sSetPause \(CyBool\\_t isPause\)](#)  
*Pause or resume I2S data transfers.*
- [void CyU3PRegisterI2sCallBack \(CyU3PI2sIntrCb\\_t i2sIntrCb\)](#)  
*This function registers a callback function for notification of I2S interrupts.*

### 5.22.1 Detailed Description

The I2S (Inter-IC Sound) interface is a serial interface defined for communication of stereophonic audio data between devices. The FX3 device includes a I2S master interface which can be connected to an I2S peripheral. The I2S driver module provides functions to configure the I2S interface and to send mono or stereo audio output on the I2S link.

### 5.22.2 Typedef Documentation

#### 5.22.2.1 [typedef struct CyU3PI2sConfig\\_t CyU3PI2sConfig\\_t](#)

I2S interface configuration parameters.

##### Description

This structure encapsulates all of the configurable parameters that can be selected for the I2S interface. The [CyU3PI2sSetConfig\(\)](#) function accepts a pointer to this structure, and updates all of the interface parameters.

##### See also

[CyU3PI2sSetConfig](#)

#### 5.22.2.2 [typedef enum CyU3PI2sError\\_t CyU3PI2sError\\_t](#)

List of I2S specific error/status codes.

##### Description

This type lists the various I2S specific error/status codes that are sent to the event callback as event data, when the event type is CY\_U3P\_I2S\_ERROR\_EVT.

##### See also

[CyU3PI2sEvt\\_t](#)  
[CyU3PI2sIntrCb\\_t](#)

### 5.22.2.3 `typedef enum CyU3PI2sEvt_t CyU3PI2sEvt_t`

List of I2S related event types.

#### Description

This enumeration lists the various I2S related event codes that are notified to the user application through an event callback.

#### See also

[CyU3PI2sIntrCb\\_t](#)  
[CyU3PI2sError\\_t](#)

### 5.22.2.4 `typedef void(* CyU3PI2sIntrCb_t)(CyU3PI2sEvt_t evt, CyU3PI2sError_t error )`

Prototype of I2S event callback function.

#### Description

This function type defines a callback to be called when an I2S interrupt has been received. A function of this type can be registered with the I2S driver as a callback function and will be called whenever an event of interest occurs.

#### See also

[CyU3PI2sEvt\\_t](#)  
[CyU3PI2sError\\_t](#)  
[CyU3PRegisterI2sCallBack](#)

### 5.22.2.5 `typedef enum CyU3PI2sPadMode_t CyU3PI2sPadMode_t`

List of the supported padding modes.

#### Description

This type lists the padding modes supported on the I2S interface.

#### See also

[CyU3PI2sConfig\\_t](#)

### 5.22.2.6 `typedef enum CyU3PI2sSampleRate_t CyU3PI2sSampleRate_t`

List of supported sample rates.

#### Description

This type lists the supported sample rates for audio playback through the I2S interface.

#### See also

[CyU3PI2sConfig\\_t](#)

### 5.22.2.7 `typedef enum CyU3PI2sSampleWidth_t CyU3PI2sSampleWidth_t`

List of supported bit widths for the I2S interface.

#### Description

This type lists the supported bit-widths on the I2S interface.

#### See also

[CyU3PI2sConfig\\_t](#)

### 5.22.3 Enumeration Type Documentation

#### 5.22.3.1 enum CyU3P12sError\_t

List of I2S specific error/status codes.

##### Description

This type lists the various I2S specific error/status codes that are sent to the event callback as event data, when the event type is CY\_U3P\_I2S\_ERROR\_EVT.

##### See also

[CyU3P12sEvt\\_t](#)  
[CyU3P12sIntrCb\\_t](#)

##### Enumerator

**CY\_U3P\_I2S\_ERROR\_LTX\_UNDERFLOW** A left channel underflow occurred.  
**CY\_U3P\_I2S\_ERROR\_RTX\_UNDERFLOW** A right channel underflow occurred.  
**CY\_U3P\_I2S\_ERROR\_LTX\_OVERFLOW** A left channel overflow occurred.  
**CY\_U3P\_I2S\_ERROR\_RTX\_OVERFLOW** A right channel overflow occurred.

#### 5.22.3.2 enum CyU3P12sEvt\_t

List of I2S related event types.

##### Description

This enumeration lists the various I2S related event codes that are notified to the user application through an event callback.

##### See also

[CyU3P12sIntrCb\\_t](#)  
[CyU3P12sError\\_t](#)

##### Enumerator

**CY\_U3P\_I2S\_EVENT\_TXL\_DONE** Transmission of left channel data is complete.  
**CY\_U3P\_I2S\_EVENT\_TXR\_DONE** Transmission of right channel data is complete.  
**CY\_U3P\_I2S\_EVENT\_PAUSED** Pause has taken effect.  
**CY\_U3P\_I2S\_EVENT\_ERROR** An I2S error has been detected.

#### 5.22.3.3 enum CyU3P12sPadMode\_t

List of the supported padding modes.

##### Description

This types lists the padding modes supported on the I2S interface.

##### See also

[CyU3P12sConfig\\_t](#)

##### Enumerator

**CY\_U3P\_I2S\_PAD\_MODE\_NORMAL** I2S normal operation.  
**CY\_U3P\_I2S\_PAD\_MODE\_LEFT\_JUSTIFIED** Left justified.

**CY\_U3P\_I2S\_PAD\_MODE\_RIGHT\_JUSTIFIED** Right justified.

**CY\_U3P\_I2S\_PAD\_MODE\_RESERVED** Reserved mode.

**CY\_U3P\_I2S\_PAD\_MODE\_CONTINUOUS** Without padding.

**CY\_U2P\_I2S\_NUM\_PAD\_MODES** Number of pad modes.

#### 5.22.3.4 enum CyU3PI2sSampleRate\_t

List of supported sample rates.

##### Description

This type lists the supported sample rates for audio playback through the I2S interface.

See also

[CyU3PI2sConfig\\_t](#)

Enumerator

**CY\_U3P\_I2S\_SAMPLE\_RATE\_8KHz** 8 KHz

**CY\_U3P\_I2S\_SAMPLE\_RATE\_16KHz** 16 KHz

**CY\_U3P\_I2S\_SAMPLE\_RATE\_32KHz** 32 KHz

**CY\_U3P\_I2S\_SAMPLE\_RATE\_44\_1KHz** 44.1 KHz

**CY\_U3P\_I2S\_SAMPLE\_RATE\_48KHz** 48 KHz

**CY\_U3P\_I2S\_SAMPLE\_RATE\_96KHz** 96 KHz

**CY\_U3P\_I2S\_SAMPLE\_RATE\_192KHz** 192 KHz

#### 5.22.3.5 enum CyU3PI2sSampleWidth\_t

List of supported bit widths for the I2S interface.

##### Description

This type lists the supported bit-widths on the I2S interface.

See also

[CyU3PI2sConfig\\_t](#)

Enumerator

**CY\_U3P\_I2S\_WIDTH\_8\_BIT** 8 bit

**CY\_U3P\_I2S\_WIDTH\_16\_BIT** 16 bit

**CY\_U3P\_I2S\_WIDTH\_18\_BIT** 18 bit

**CY\_U3P\_I2S\_WIDTH\_24\_BIT** 24 bit

**CY\_U3P\_I2S\_WIDTH\_32\_BIT** 32 bit

**CY\_U3P\_I2S\_NUM\_BIT\_WIDTH** Number of options.

#### 5.22.4 Function Documentation

##### 5.22.4.1 CyU3PReturnStatus\_t CyU3PI2sDelInit( void )

Stops the I2S interface block on the FX3.

**Description**

This function disables and powers off the I2S interface. This function can be used to shut off the interface to save power when it is not in use.

**Return value**

CY\_U3P\_SUCCESS - When the de-init is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - When the module has not been previously initialized.

**See also**

[CyU3PI2sInit](#)

#### 5.22.4.2 CyU3PReturnStatus\_t CyU3PI2sEnableExternalMclk ( void )

Select an external input as the MCLK for the I2S interface.

**Description**

During normal operation, the I2S block on the FX3 internally generates the I2S bit clock (SCK) required using its master SYSCLK and a configurable frequency divider block. However, some applications may require the I2S interface on FX3 to work on the basis of an externally provided MCLK input.

This API is used to enable the I2S block to use an external MCLK input instead of generating the clock internally. It is not allowed to change the direction of the clock while the I2S block on FX3 is running. Therefore, this API has to be called before the [CyU3PI2sInit\(\)](#) function is called.

If the external clock input is enabled, a clock that runs 4 times as fast as the bit clock required should be connected to the I2S\_SCK pin of the FX3 device.

**Return value**

CY\_U3P\_SUCCESS if this API is called prior to calling [CyU3PI2sInit\(\)](#).

CY\_U3P\_ERROR\_ALREADY\_STARTED if the API is called after [CyU3PI2sInit\(\)](#).

**See also**

[CyU3PI2sInit](#)

#### 5.22.4.3 CyU3PReturnStatus\_t CyU3PI2sInit ( void )

Starts the I2S interface block on the FX3.

**Description**

This function powers up the I2S interface block on the FX3 device, and is expected to be the first I2S API function that is called by the application.

This function sets up the clock to a default value of CY\_U3P\_I2S\_SAMPLE\_RATE\_8KHz.

**Return value**

CY\_U3P\_SUCCESS - When the init is successful.

CY\_U3P\_ERROR\_ALREADY\_STARTED - When the I2S block has already been initialized.

CY\_U3P\_ERROR\_NOT\_CONFIGURED - When the I2S block has not been enabled in the IOMatrix.

**See also**

[CyU3PI2sDeinit](#)

[CyU3PI2sSetConfig](#)

[CyU3PI2sTransmitBytes](#)

[CyU3PI2sSetMute](#)

[CyU3PI2sSetPause](#)

#### 5.22.4.4 CyU3PReturnStatus\_t CyU3PI2sSetConfig ( *CyU3PI2sConfig\_t \* config*, *CyU3PI2sIntrCb\_t cb* )

Sets the I2S interface parameters.

##### Description

This function is used to configure the I2S master interface based on the desired settings. This function should be called repeatedly every time the output stream parameters change.

The callback parameter is used to specify an event callback function that will be called by the driver when an I2S interrupt occurs.

##### Return value

*CY\_U3P\_SUCCESS* - When the SetConfig is successful.

*CY\_U3P\_ERROR\_NOT\_STARTED* - When the I2S has not been initialized.

*CY\_U3P\_ERROR\_NULL\_POINTER* - When the config pointer is NULL.

*CY\_U3P\_ERROR\_BAD\_ARGUMENT* - When the configuration parameters are incorrect.

*CY\_U3P\_ERROR\_TIMEOUT* - If there is a timeout while trying to clear the left and right channel pipes.

*CY\_U3P\_ERROR\_MUTEX\_FAILURE* - Failed to get a mutex lock on the I2S block.

##### See also

[CyU3PI2sConfig\\_t](#)  
[CyU3PI2sIntrCb\\_t](#)  
[CyU3PI2sInit](#)  
[CyU3PI2sTransmitBytes](#)  
[CyU3PI2sSetMute](#)  
[CyU3PI2sSetPause](#)

##### Parameters

<i>config</i>	I2S configuration settings.
<i>cb</i>	Callback for getting the events.

#### 5.22.4.5 CyU3PReturnStatus\_t CyU3PI2sSetMute ( *CyBool\_t isMute* )

Mute or unmute the I2S output stream.

##### Description

This function sets the I2C master to mute or unmute the output stream. When configured for mute, the I2S master drives zero samples instead of actual data output.

##### Return value

*CY\_U3P\_SUCCESS* - When the mute control is successful.

*CY\_U3P\_ERROR\_NOT\_CONFIGURED* - When the I2S interface has not been configured.

*CY\_U3P\_ERROR\_MUTEX\_FAILURE* - Failed to get a mutex lock on the I2S block.

##### See also

[CyU3PI2sInit](#)  
[CyU3PI2sSetConfig](#)

##### Parameters

<i>isMute</i>	CyTrue: Mute the I2S, CyFalse: Un-mute the I2S.
---------------	---

#### 5.22.4.6 CyU3PReturnStatus\_t CyU3PI2sSetPause ( CyBool\_t *isPause* )

Pause or resume I2S data transfers.

##### Description

This function is used to pause or resume the data transfer on the I2S interface.

##### Return value

CY\_U3P\_SUCCESS - When the pause control is successful.

CY\_U3P\_ERROR\_NOT\_CONFIGURED - When the I2S interface has not been configured.

CY\_U3P\_ERROR\_MUTEX\_FAILURE - Failed to get a mutex lock on the I2S block.

##### See also

[CyU3PI2sInit](#)

[CyU3PI2sSetConfig](#)

##### Parameters

<i>isPause</i>	CyTrue: pause the I2S, CyFalse: resume the I2S.
----------------	---

#### 5.22.4.7 CyU3PReturnStatus\_t CyU3PI2sTransmitBytes ( uint8\_t \* *lData*, uint8\_t \* *rData*, uint8\_t *lByteCount*, uint8\_t *rByteCount* )

Transmits data byte by byte over the I2S interface.

##### Description

This function sends the data over the I2S interface in register mode. This is allowed only when the I2S interface block is configured for register mode transfer. The data involved in this transfer is always left justified.

##### Return value

CY\_U3P\_SUCCESS - When the data transfer is successful.

CY\_U3P\_ERROR\_NULL\_POINTER - When the data pointers are NULL.

CY\_U3P\_ERROR\_TIMEOUT - When a timeout occurs.

CY\_U3P\_ERROR\_NOT\_CONFIGURED - If the I2S has not been configured for register mode transfers.

CY\_U3P\_ERROR\_MUTEX\_FAILURE - Failed to get a mutex lock on the I2S block.

##### See also

[CyU3PI2sSetConfig](#)

[CyU3PI2sSetMute](#)

[CyU3PI2sSetPause](#)

##### Parameters

<i>lData</i>	Buffer containing data to be sent on the left channel.
<i>rData</i>	Buffer containing data to be sent on the right channel.
<i>lByteCount</i>	Number of bytes to be transferred on the left channel.
<i>rByteCount</i>	Number of bytes to be transferred on the right channel.

#### 5.22.4.8 void CyU3PRegisterI2sCallBack ( CyU3PI2sIntrCb\_t *i2sIntrCb* )

This function registers a callback function for notification of I2S interrupts.

**Description**

This function registers a callback function that will be called for notification of I2S interrupts. This can also be done through the CyU3PI2sInit API call.

**Return value**

None

**See also**

[CyU3PI2sIntrCb\\_t](#)  
[CyU3PI2sInit](#)

**Parameters**

<i>i2sIntrCb</i>	Callback function pointer.
------------------	----------------------------

**5.23 firmware/u3p\_firmware/inc/cyu3lpp.h File Reference**

All of the serial peripherals and GPIOs on the FX3 device share a set of common hardware resources that need to be initialized before any of these interfaces can be used. Further, the drivers for all of the serial peripherals share a single RTOS thread; because the CPU load due to each of these is expected to be minimal. This file defines the data types and APIs that are used for the central management of all these serial peripheral blocks.

```
#include <cyu3os.h>
#include <cyu3types.h>
#include <cyu3system.h>
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

**Data Structures**

- struct [CyU3PGpioClock\\_t](#)  
*Clock configuration information for the GPIO block.*

**TypeDefs**

- typedef enum [CyU3PGpioIoMode\\_t](#) [CyU3PGpioIoMode\\_t](#)  
*List of IO modes.*
- typedef enum [CyU3PGpioSimpleClkDiv\\_t](#) [CyU3PGpioSimpleClkDiv\\_t](#)  
*Clock divider values for sampling simple GPIOs.*
- typedef struct [CyU3PGpioClock\\_t](#) [CyU3PGpioClock\\_t](#)  
*Clock configuration information for the GPIO block.*
- typedef void(\* [CyU3PLppInterruptHandler](#)) (void)  
*Serial peripheral interrupt handler function type.*

**Enumerations**

- enum [CyU3PGpioIoMode\\_t](#){ CY\_U3P\_GPIO\_IO\_MODE\_NONE = 0, CY\_U3P\_GPIO\_IO\_MODE\_WPU, CY\_U3P\_GPIO\_IO\_MODE\_WPD }
- List of IO modes.*

- enum `CyU3PGpioSimpleClkDiv_t` {
 `CY_U3P_GPIO_SIMPLE_DIV_BY_2` = 0, `CY_U3P_GPIO_SIMPLE_DIV_BY_4`, `CY_U3P_GPIO_SIMPLE_DIV_BY_16`, `CY_U3P_GPIO_SIMPLE_DIV_BY_64`,  
`CY_U3P_GPIO_SIMPLE_NUM_DIV` }

*Clock divider values for sampling simple GPIOs.*

## Functions

- `CyBool_t CyU3PLppGpioBlockIsOn (void)`  
*Check if the boot firmware has left the GPIO block powered ON.*
- `CyU3PReturnStatus_t CyU3PLppInit (CyU3PLppModule_t lppModule, CyU3PLppInterruptHandler intrHandler)`  
*Register the specified peripheral block as active.*
- `CyU3PReturnStatus_t CyU3PLppDeInit (CyU3PLppModule_t lppModule)`  
*Register that a specified peripheral block has been made inactive.*
- `CyU3PReturnStatus_t CyU3PGpioSetClock (CyU3PGpioClock_t *clk_p)`  
*Enable the GPIO block clocks on the FX3 device.*
- `CyU3PReturnStatus_t CyU3PI2sSetClock (uint32_t clkRate)`  
*Set the frequency for and enable the I2S clock.*
- `CyU3PReturnStatus_t CyU3PI2cSetClock (uint32_t bitRate)`  
*Set the frequency for and enable the I2C clock.*
- `CyU3PReturnStatus_t CyU3PUartSetClock (uint32_t baudRate)`  
*Set the frequency for and enable the UART block.*
- `CyU3PReturnStatus_t CyU3PSpiSetClock (uint32_t clock)`  
*Set the frequency for and enable the SPI block.*
- `CyU3PReturnStatus_t CyU3PSpiStopClock (void)`  
*Disable the SPI block clock.*
- `CyU3PReturnStatus_t CyU3PI2cStopClock (void)`  
*Disable the I2C block clock.*
- `CyU3PReturnStatus_t CyU3PGpioStopClock (void)`  
*Disable the GPIO block clocks.*
- `CyU3PReturnStatus_t CyU3PI2sStopClock (void)`  
*Disable the I2S block clock.*
- `CyU3PReturnStatus_t CyU3PUartStopClock (void)`  
*Disable the UART block clock.*
- `CyU3PReturnStatus_t CyU3PSetI2cDriveStrength (CyU3PDriveStrengthState_t i2cDriveStrength)`  
*Set the IO drive strength for the I2C interface.*
- `CyU3PReturnStatus_t CyU3PGpioSetIoMode (uint8_t gpioId, CyU3PGpioIoMode_t ioMode)`  
*Set IO mode for the selected GPIO.*
- `CyU3PReturnStatus_t CyU3PSetGpioDriveStrength (CyU3PDriveStrengthState_t gpioDriveStrength)`  
*Set the IO drive strength for all FX3 GPIOs.*

### 5.23.1 Detailed Description

All of the serial peripherals and GPIOs on the FX3 device share a set of common hardware resources that need to be initialized before any of these interfaces can be used. Further, the drivers for all of the serial peripherals share a single RTOS thread; because the CPU load due to each of these is expected to be minimal. This file defines the data types and APIs that are used for the central management of all these serial peripheral blocks.

## 5.23.2 Typedef Documentation

### 5.23.2.1 `typedef struct CyU3PGpioClock_t CyU3PGpioClock_t`

Clock configuration information for the GPIO block.

#### Description

The GPIO block on the FX3 makes use of three different clocks. The master (fast) clock for this block is directly divided down from the SYS\_CLK. The block also uses a slow clock which can be derived from this fast clock. The complex GPIOs on FX3 can be clocked on either the fast or the slow clock.

The simple GPIOs are clocked by another clock which is separately derived from the fast clock.

This structure encapsulates all of the clock parameters for the GPIO clock.

#### See also

[CyU3PGpioSetClock](#)  
[CyU3PSysClockSrc\\_t](#)  
[CyU3PGpioSimpleClkDiv\\_t](#)

### 5.23.2.2 `typedef enum CyU3PGpioloMode_t CyU3PGpioloMode_t`

List of IO modes.

#### Description

The FX3 device supports internal weak pull-ups or pull-downs on its GPIOs. These terminations can be used even on signals that are not used as GPIOs.

This type lists the possible internal IO modes that can be selected for a FX3 GPIO.

#### See also

[CyU3PGpioSetIoMode](#)

### 5.23.2.3 `typedef enum CyU3PGpioSimpleClkDiv_t CyU3PGpioSimpleClkDiv_t`

Clock divider values for sampling simple GPIOs.

#### Description

Sampling and updates of simple GPIOs on the FX3 device are performed based on a clock that is derived from the GPIO fast clock. This type lists the possible divisor values that can be used to derive this clock from the fast clock.

#### See also

[CyU3PGpioClock\\_t](#)  
[CyU3PGpioSetClock](#)

### 5.23.2.4 `typedef void(* CyU3PLpplInterruptHandler) (void)`

Serial peripheral interrupt handler function type.

#### Description

Each serial peripheral (I2C, I2S, SPI, UART and GPIO) on the FX3 device can have interrupts associated with it. The drivers for these blocks can register an interrupt handler function that the FX3 firmware framework will call when an interrupt is received. This block specific interrupt handler is registered through the [CyU3PLpplInit](#) function.

#### See also

[CyU3PLpplInit](#)

### 5.23.3 Enumeration Type Documentation

#### 5.23.3.1 enum CyU3PGpioIoMode\_t

List of IO modes.

##### Description

The FX3 device supports internal weak pull-ups or pull-downs on its GPIOs. These terminations can be used even on signals that are not used as GPIOs.

This type lists the possible internal IO modes that can be selected for a FX3 GPIO.

##### See also

[CyU3PGpioSetIoMode](#)

##### Enumerator

**CY\_U3P\_GPIO\_IO\_MODE\_NONE** No internal pull-up or pull-down. Default condition.

**CY\_U3P\_GPIO\_IO\_MODE\_WPU** A weak pull-up is provided on the IO.

**CY\_U3P\_GPIO\_IO\_MODE\_WPD** A weak pull-down is provided on the IO.

#### 5.23.3.2 enum CyU3PGpioSimpleClkDiv\_t

Clock divider values for sampling simple GPIOs.

##### Description

Sampling and updates of simple GPIOs on the FX3 device are performed based on a clock that is derived from the GPIO fast clock. This type lists the possible divisor values that can be used to derive this clock from the fast clock.

##### See also

[CyU3PGpioClock\\_t](#)

[CyU3PGpioSetClock](#)

##### Enumerator

**CY\_U3P\_GPIO\_SIMPLE\_DIV\_BY\_2** Simple GPIO clock frequency is fast clock by 2.

**CY\_U3P\_GPIO\_SIMPLE\_DIV\_BY\_4** Simple GPIO clock frequency is fast clock by 4.

**CY\_U3P\_GPIO\_SIMPLE\_DIV\_BY\_16** Simple GPIO clock frequency is fast clock by 16.

**CY\_U3P\_GPIO\_SIMPLE\_DIV\_BY\_64** Simple GPIO clock frequency is fast clock by 64.

**CY\_U3P\_GPIO\_SIMPLE\_NUM\_DIV** Number of divider enumerations.

### 5.23.4 Function Documentation

#### 5.23.4.1 CyU3PReturnStatus\_t CyU3PGpioSetClock ( CyU3PGpioClock\_t \* clk\_p )

Enable the GPIO block clocks on the FX3 device.

##### Description

The GPIO block on the FX3 device makes use of multiple clocks. This function is used to select the frequency for these clocks and to turn them on.

##### Return value

CY\_U3P\_SUCCESS - If the clocks were successfully configured and enabled.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the clock configuration specified is invalid.

See also

[CyU3PGpioClock\\_t](#)  
[CyU3PGpioStopClock](#)

**Parameters**

<i>clk_p</i>	The desired clock parameters.
--------------	-------------------------------

**5.23.4.2 CyU3PReturnStatus\_t CyU3PGpioSetIoMode ( uint8\_t gpioid, CyU3PGpioloMode\_t ioMode )**

Set IO mode for the selected GPIO.

**Description**

This function enables or disables internal pull-ups/pull-downs on the selected FX3 IO pin. This IO mode change will take about 5 us to become active.

**Return value**

CY\_U3P\_SUCCESS - If the operation is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the gpioid or ioMode requested is invalid.

**See also**

[CyU3PGpioloMode\\_t](#)

**Parameters**

<i>gpioid</i>	GPIO Pin to be updated.
<i>ioMode</i>	Desired pull-up/pull-down mode.

**5.23.4.3 CyU3PReturnStatus\_t CyU3PGpioStopClock ( void )**

Disable the GPIO block clocks.

**Description**

This function disables all of the clocks associated with the GPIO block. This should only be called after the GPIO block has been de-initialized.

**Return value**

CY\_U3P\_SUCCESS - If the GPIO clocks were successfully turned off.

**See also**

[CyU3PGpioSetClock](#)

**5.23.4.4 CyU3PReturnStatus\_t CyU3PI2cSetClock ( uint32\_t bitRate )**

Set the frequency for and enable the I2C clock.

**Description**

The clock frequency for the I2C block depends on the desired output bit rate. This function configures this clock frequency as required and then enables the clock.

The internal clock for the I2C block needs to run at 10X the desired output clock frequency. Further, the FX3 device only supports integer and half divisors for deriving the internal clock from the SYS\_CLK.

This function sets the internal clock for the I2C block to the value that provides the closest approximation of the desired output bit rate.

**Return value**

CY\_U3P\_SUCCESS - If the I2C interface clock was setup as required.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - When invalid argument is passed to the function.

See also

[CyU3PI2cStopClock](#)

Parameters

<i>bitRate</i>	Desired interface clock frequency.
----------------	------------------------------------

#### 5.23.4.5 CyU3PReturnStatus\_t CyU3PI2cStopClock ( void )

Disable the I2C block clock.

**Description**

This function disables the I2C block clock. This should only be called after the I2C block has been de-initialized.

**Return value**

CY\_U3P\_SUCCESS - If the I2C clock was successfully turned off.

See also

[CyU3PI2cSetClock](#)

#### 5.23.4.6 CyU3PReturnStatus\_t CyU3PI2sSetClock ( uint32\_t *clkRate* )

Set the frequency for and enable the I2S clock.

**Description**

The I2S block on the FX3 device needs to be clocked at different rates based on the output audio sample rate. This function sets the I2C block clock frequency and enables the clock.

The internal clock for the I2S block needs to run at 16X the desired output clock frequency. Further, the FX3 device only supports integer and half divisors for deriving the internal clock from the SYS\_CLK.

This function sets the internal clock for the I2S block to the value that provides the closest approximation of the desired output clock frequency.

**Return value**

CY\_U3P\_SUCCESS - If the I2S interface clock was setup as required.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - When invalid argument is passed to the function.

See also

[CyU3PI2sStopClock](#)

Parameters

<i>clkRate</i>	Desired interface clock frequency.
----------------	------------------------------------

#### 5.23.4.7 CyU3PReturnStatus\_t CyU3PI2sStopClock ( void )

Disable the I2S block clock.

**Description**

This function disables the I2S block clock. This should only be called after the I2S block has been de-initialized.

**Return value**

CY\_U3P\_SUCCESS - If the I2S clock is successfully turned off.

**See also**

[CyU3PI2sSetClock](#)

**5.23.4.8 CyU3PReturnStatus\_t CyU3PLppDelInit( CyU3PLppModule\_t lppModule )**

Register that a specified peripheral block has been made inactive.

**Description**

This function registers that a specified peripheral block is no longer in use. The function along with [CyU3PLppInit\(\)](#) keeps track of the peripheral blocks that are ON; and manages the power on/off of the shared resources for these blocks.

The clock for the block being disabled should be turned off only after this function is called.

**Return value**

CY\_U3P\_SUCCESS - If the call is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - If the serial peripheral being stopped has not been started.

**See also**

[CyU3PLppInit](#)

**Parameters**

<i>lppModule</i>	The peripheral block being de-initialized.
------------------	--

**5.23.4.9 CyBool\_t CyU3PLppGpioBlockIsOn( void )**

Check if the boot firmware has left the GPIO block powered ON.

**Description**

In systems which make use of the boot firmware, it is possible that some of the GPIOs have been left configured by the boot firmware. In such a case, the GPIO block on the FX3 device should not be reset during firmware initialization. This function is used to check whether the boot firmware has requested the GPIO block to be left powered ON across firmware initialization.

**Note**

Please note that all the pins that have been left configured by the boot firmware need to be selected as simple GPIOs during IO Matrix configuration. Otherwise, these pins will be tri-state because the pin functionality is overridden.

**Return value**

CyTrue if the boot firmware has left GPIO on.

CyFalse if the GPIO block is turned off.

**See also**

[CyU3PDeviceConfigureIOMatrix](#)

**5.23.4.10 CyU3PReturnStatus\_t CyU3PLppInit( CyU3PLppModule\_t lppModule, CyU3PLppInterruptHandler intrHandler )**

Register the specified peripheral block as active.

**Description**

The serial peripheral blocks on the FX3 device share some resources. While the individual peripheral blocks (GPIO, I2C, UART, SPI and I2S) can be turned on/off at runtime; the shared resources need to be kept initialized while any of these blocks are on. This function is used to manage the shared peripheral resources and also to keep track of which peripheral blocks are on.

This function need to be called after initializing the clock for the corresponding peripheral interface.

**Return value**

CY\_U3P\_SUCCESS - If the call is successful.

CY\_U3P\_ERROR\_ALREADY\_STARTED - The block is already initialized.

CY\_U3P\_ERROR\_INVALID\_SEQUENCE - If the block init is being called without turning on the corresponding clock.

**See also**

[CyU3PLppDelInit](#)

[CyU3PUartSetClock](#)

[CyU3PI2cSetClock](#)

[CyU3PI2sSetClock](#)

[CyU3PSpiSetClock](#)

**Parameters**

<i>lppModule</i>	The peripheral block being initialized.
<i>intrHandler</i>	Interrupt handler function for the peripheral block.

**5.23.4.11 CyU3PReturnStatus\_t CyU3PSetGpioDriveStrength ( CyU3PDriveStrengthState\_t gpioDriveStrength )**

Set the IO drive strength for all FX3 GPIOs.

**Description**

This function updates the drive strength of all FX3 GPIOs. This is an alternative for setting the drive strength for each port separately.

**Return value**

CY\_U3P\_SUCCESS - If the operation is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - If the GPIO block has not been initialized.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the drive strength requested is invalid.

**See also**

[CyU3PDriveStrengthState\\_t](#)

**Parameters**

<i>gpioDriveStrength</i>	Desired GPIO Drive strength
--------------------------	-----------------------------

**5.23.4.12 CyU3PReturnStatus\_t CyU3PSetI2cDriveStrength ( CyU3PDriveStrengthState\_t i2cDriveStrength )**

Set the IO drive strength for the I2C interface.

**Description**

The function sets the IO Drive strength for the I2C interface signals. The default IO drive strength for I2C is set to CY\_U3P\_DS\_THREE\_QUARTER\_STRENGTH.

**Return value**

CY\_U3P\_SUCCESS - If the operation is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - If the I2C block has not been initialized.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the drive strength requested is invalid.

**See also**

[CyU3PDriveStrengthState\\_t](#)

**Parameters**

<i>i2cDriveStrength</i>	Drive strength desired for I2C signals.
-------------------------	---

**5.23.4.13 CyU3PReturnStatus\_t CyU3PSpiSetClock ( uint32\_t *clock* )**

Set the frequency for and enable the SPI block.

**Description**

The clock frequency for the SPI block depends on the desired output bit rate. This function configures this clock frequency as required and then enables the clock.

The internal clock for the SPI block needs to run at 2X the desired output clock frequency. Further, the FX3 device only supports integer and half divisors for deriving the internal clock from the SYS\_CLK.

This function sets the internal clock for the SPI block to the value that provides the closest approximation of the desired output bit rate.

**Return value**

CY\_U3P\_SUCCESS - If the SPI clock has been successfully turned on.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - When invalid argument is passed to the function.

**See also**

[CyU3PSpiStopClock](#)

**Parameters**

<i>clock</i>	Desired output bit rate.
--------------	--------------------------

**5.23.4.14 CyU3PReturnStatus\_t CyU3PSpiStopClock ( void )**

Disable the SPI block clock.

**Description**

This function disables the SPI block clock. This should only be called after the SPI block has been de-initialized.

**Return value**

CY\_U3P\_SUCCESS - if the SPI clock is turned off successfully.

**See also**

[CyU3PSpiSetClock](#)

**5.23.4.15 CyU3PReturnStatus\_t CyU3PUartSetClock ( uint32\_t *baudRate* )**

Set the frequency for and enable the UART block.

**Description**

The clock frequency for the UART block depends on the desired output baud rate. This function configures this clock frequency as required and then enables the clock.

The internal clock for the UART block needs to run at 16X the desired output clock frequency. Further, the FX3 device only supports integer and half divisors for deriving the internal clock from the SYS\_CLK.

This function sets the internal clock for the UART block to the value that provides the closest approximation of the desired output baud rate.

**Return value**

CY\_U3P\_SUCCESS - If the UART clock and baud rate was setup as required.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - When invalid argument is passed to the function.

**See also**

[CyU3PUartStopClock](#)

**Parameters**

<i>baudRate</i>	Desired baud rate for the UART interface.
-----------------	---

**5.23.4.16 CyU3PReturnStatus\_t CyU3PUartStopClock( void )**

Disable the UART block clock.

**Description**

This function disables the UART block clock. This should only be called after the UART block has been de-initialized.

**Return value**

CY\_U3P\_SUCCESS - if the UART clock was successfully turned off.

**See also**

[CyU3PUartSetClock](#)

## 5.24 firmware/u3p\_firmware/inc/cyu3mbox.h File Reference

The mailbox handler is responsible for sending/receiving short messages from an external processor through the mailbox registers.

```
#include "cyu3types.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

### Data Structures

- struct [CyU3PMbox](#)

*Structure that holds a packet of mailbox data.*

### TypeDefs

- typedef struct [CyU3PMbox CyU3PMbox](#)  
*Structure that holds a packet of mailbox data.*
- typedef void(\* [CyU3PMboxCb\\_t](#)) ([CyBool\\_t](#) mboxEvt)

*Type of function to be called to notify about a mailbox related interrupt.*

## Functions

- void [CyU3PMboxInit](#) ([CyU3PMboxCb\\_t](#) callback)  
*Initialize the mailbox handler.*
- void [CyU3PMboxDelInit](#) (void)  
*De-initialize the mailbox handler.*
- void [CyU3PMboxReset](#) (void)  
*Reset the mailbox handler.*
- [CyU3PReturnStatus\\_t CyU3PMboxWrite](#) ([CyU3PMbox](#) \*mbox)  
*This function sends a mailbox message to the external processor.*
- [CyU3PReturnStatus\\_t CyU3PMboxRead](#) ([CyU3PMbox](#) \*mbox)  
*This function reads an incoming mailbox message.*
- [CyU3PReturnStatus\\_t CyU3PMboxWait](#) (void)  
*Wait until the Mailbox register to send messages to P-port is free.*

## Variables

- [CyU3PMboxCb\\_t gIMboxCb](#)

### 5.24.1 Detailed Description

The mailbox handler is responsible for sending/receiving short messages from an external processor through the mailbox registers.

#### Description

The FX3 device implements a set of mailbox registers that can be used to exchange short general purpose messages between FX3 and the external device connected on the P-port. Messages of upto 8 bytes can be sent in each direction at a time. The mailbox handler is responsible for handling mailbox data in both directions.

### 5.24.2 Typedef Documentation

#### 5.24.2.1 [typedef struct CyU3PMbox CyU3PMbox](#)

Structure that holds a packet of mailbox data.

#### Description

The FX3 device has 8 byte mailbox registers that can be used when the P-port mode is enabled. This structure represents the eight bytes to be written to or read from the corresponding mailbox registers.

#### 5.24.2.2 [typedef void\(\\* CyU3PMboxCb\\_t\)\(CyBool\\_t mboxEvt \)](#)

Type of function to be called to notify about a mailbox related interrupt.

#### Description

This type is the prototype for a callback function that will be called to notify the application about a mailbox interrupt. The mboxEvt parameter will identify the type of interrupt.

If a read interrupt is received, the mailbox registers have to be read; and if a write interrupt is received, any pending data can be written to the registers.

### 5.24.3 Function Documentation

#### 5.24.3.1 void CyU3PMboxDelInit ( void )

De-initialize the mailbox handler.

##### Description

Destroys the mailbox related structures.

##### See also

[CyU3PMboxInit](#)

#### 5.24.3.2 void CyU3PMboxInit ( CyU3PMboxCb\_t callback )

Initialize the mailbox handler.

##### Description

Initiate the mailbox related structures and register a callback function that will be called on every mailbox related interrupt.

##### See also

[CyU3PMboxDelInit](#)

##### Parameters

<i>callback</i>	Callback function to be called on interrupt.
-----------------	--

#### 5.24.3.3 CyU3PReturnStatus\_t CyU3PMboxRead ( CyU3PMbox \* mbox )

This function reads an incoming mailbox message.

##### Description

This function is used to read the contents of an incoming mailbox message. This needs to be called in response to a read event callback.

##### Return value

CY\_U3P\_SUCCESS - If the operation is successful.

CY\_U3P\_ERROR\_NULL\_POINTER - If the mbox pointer provided is NULL.

##### See also

[CyU3PMboxWrite](#)

##### Parameters

<i>mbox</i>	Pointer to buffer to hold the incoming message data.
-------------	--

#### 5.24.3.4 void CyU3PMboxReset ( void )

Reset the mailbox handler.

##### Description

Clears the data structures and state related to mailbox handler. Can be used for error recovery.

See also

[CyU3PMboxInit](#)

#### 5.24.3.5 CyU3PReturnStatus\_t CyU3PMboxWait( void )

Wait until the Mailbox register to send messages to P-port is free.

##### Description

This function waits until the mailbox register used to send messages to the processor/device connected on FX3's P-port is free. This is expected to be used in cases where the application needs to ensure that the last message that was sent out has been read by the external processor or device.

##### Return value

CY\_U3P\_SUCCESS - If the operation is successful.

CY\_U3P\_ERROR\_FAILURE - If there is a failure getting a mutex lock on the mailboxes.

See also

[CyU3PMboxWrite](#)

#### 5.24.3.6 CyU3PReturnStatus\_t CyU3PMboxWrite( CyU3PMbox \*mbox )

This function sends a mailbox message to the external processor.

##### Description

This function writes 8 bytes of data to the outgoing mailbox registers after ensuring that the external processor has read out the previous message. If the previous message has not been read out, this function will block until the registers become free.

##### Return value

CY\_U3P\_SUCCESS - If the operation is successful.

CY\_U3P\_ERROR\_NULL\_POINTER - If the mbox pointer provided is NULL.

CY\_U3P\_ERROR\_FAILURE - If there is an error getting a mutex lock on the mailboxes.

See also

[CyU3PMboxRead](#)

[CyU3PMboxWait](#)

##### Parameters

<i>mbox</i>	Pointer to mailbox message data.
-------------	----------------------------------

#### 5.24.4 Variable Documentation

##### 5.24.4.1 CyU3PMboxCb\_t gIMboxCb

Callback to be called after mailbox intr has happened

### 5.25 firmware/u3p\_firmware/inc/cyu3mipicsi.h File Reference

This file contains the MIPI-CSI interface management data structures, functions and macros for CX3 devices.

```
#include <cyu3os.h>
#include <cyu3types.h>
#include <cyu3sib.h>
#include <cyu3gpif.h>
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

## Data Structures

- struct [CyU3PMipicsiErrorCounts\\_t](#)  
*Structure defining MIPI-CSI block Phy errors.*
- struct [CyU3PMipicsiCfg\\_t](#)  
*Structure defining the MIPI-CSI block interface configuration.*

## Macros

- #define CX3\_START\_SCK0 0  
*The following macros define the various states supported by the fixed function GPIF on the CX3 parts.*
- #define CX3\_IDLE\_SCK0 1
- #define CX3\_WAIT\_FOR\_FRAME\_START\_SCK0 2
- #define CX3\_PUSH\_DATA\_SCK0 3
- #define CX3\_PUSH\_DATA\_SCK1 4
- #define CX3\_WAIT\_TO\_FILL\_SCK0 5
- #define CX3\_WAIT\_TO\_FILL\_SCK1 7
- #define CX3\_WAIT\_FULL\_SCK0\_NEXT\_SCK1 6
- #define CX3\_WAIT\_FULL\_SCK1\_NEXT\_SCK0 8
- #define CX3\_PARTIAL\_BUFFER\_IN\_SCK0 9
- #define CX3\_PARTIAL\_BUFFER\_IN\_SCK1 10
- #define CX3\_FULL\_BUFFER\_IN\_SCK0 11
- #define CX3\_FULL\_BUFFER\_IN\_SCK1 12
- #define CX3\_START\_SCK1 15
- #define CX3\_IDLE\_SCK1 16
- #define CX3\_WAIT\_FOR\_FRAME\_START\_SCK1 17
- #define FW\_WAIT\_SCK0 13
- #define FW\_WAIT\_SCK1 14
- #define ALPHA\_CX3\_START\_SCK0 0x0
- #define ALPHA\_CX3\_START\_SCK1 0x0
- #define CX3\_NUMBER\_OF\_STATES 18
- #define CX3\_START CX3\_START\_SCK0
- #define CX3\_IDLE CX3\_IDLE\_SCK0
- #define CX3\_WAIT\_FOR\_FRAME\_START CX3\_WAIT\_FOR\_FRAME\_START\_SCK0
- #define CX3\_PUSH\_DATA\_TO\_SCK0 CX3\_PUSH\_DATA\_SCK0
- #define CX3\_PUSH\_DATA\_TO\_SCK1 CX3\_PUSH\_DATA\_SCK1
- #define CX3\_ALPHA\_START ALPHA\_CX3\_START\_SCK0
- #define CY\_U3P\_CSI\_DF\_UVC\_YUV2 (CY\_U3P\_CSI\_DF\_YUV422\_8\_1)

*Data Format used for the UVC YUV422 (UVC YUV2) format.*

## Typedefs

- **typedef enum CyU3PMipicsil2cFreq\_t CyU3PMipicsil2cFreq\_t**  
*I2C frequency for MIPI-CSI interface communication.*
- **typedef enum CyU3PMipicsiSensorlo\_t CyU3PMipicsiSensorlo\_t**  
*MIPI Control lines for Image Sensor.*
- **typedef enum CyU3PMipicsiReset\_t CyU3PMipicsiReset\_t**  
*MIPI-CSI interface Reset Type.*
- **typedef struct CyU3PMipicsiErrorCounts\_t CyU3PMipicsiErrorCounts\_t**  
*Structure defining MIPI-CSI block Phy errors.*
- **typedef enum CyU3PMipicsiDataFormat\_t CyU3PMipicsiDataFormat\_t**  
*MIPI-CSI Data Formats.*
- **typedef enum CyU3PMipicsiPllClkDiv\_t CyU3PMipicsiPllClkDiv\_t**  
*Clock Divider Values.*
- **typedef enum CyU3PMipicsiPllClkFrs\_t CyU3PMipicsiPllClkFrs\_t**  
*Frequency range selection for the MIPI-CSI PLL Clock.*
- **typedef enum CyU3PMipicsiBusWidth\_t CyU3PMipicsiBusWidth\_t**  
*Bus Widths supported by the fixed function GPIF interface.*
- **typedef struct CyU3PMipicsiCfg\_t CyU3PMipicsiCfg\_t**  
*Structure defining the MIPI-CSI block interface configuration.*

## Enumerations

- **enum CyU3PMipicsil2cFreq\_t { CY\_U3P\_MIPICSI\_I2C\_100KHZ, CY\_U3P\_MIPICSI\_I2C\_400KHZ }**  
*I2C frequency for MIPI-CSI interface communication.*
- **enum CyU3PMipicsiSensorlo\_t { CY\_U3P\_CSI\_IO\_XRES = 2, CY\_U3P\_CSI\_IO\_XSHUTDOWN = 4 }**  
*MIPI Control lines for Image Sensor.*
- **enum CyU3PMipicsiReset\_t { CY\_U3P\_CSI\_SOFT\_RST = 0, CY\_U3P\_CSI\_HARD\_RST }**  
*MIPI-CSI interface Reset Type.*
- **enum CyU3PMipicsiDataFormat\_t { CY\_U3P\_CSI\_DF\_RAW8 = 0x00, CY\_U3P\_CSI\_DF\_RAW10 = 0x01, CY\_U3P\_CSI\_DF\_RAW12 = 0x02, CY\_U3P\_CSI\_DF\_RAW14 = 0x08, CY\_U3P\_CSI\_DF\_RGB888 = 0x03, CY\_U3P\_CSI\_DF\_RGB666\_0 = 0x04, CY\_U3P\_CSI\_DF\_RGB666\_1 = 0x14, CY\_U3P\_CSI\_DF\_RGB565\_0 = 0x05, CY\_U3P\_CSI\_DF\_RGB565\_1 = 0x15, CY\_U3P\_CSI\_DF\_RGB565\_2 = 0x25, CY\_U3P\_CSI\_DF\_YUV422\_8\_0 = 0x06, CY\_U3P\_CSI\_DF\_YUV422\_8\_1 = 0x16, CY\_U3P\_CSI\_DF\_YUV422\_8\_2 = 0x26, CY\_U3P\_CSI\_DF\_YUV422\_10 = 0x09 }**  
*MIPI-CSI Data Formats.*
- **enum CyU3PMipicsiPllClkDiv\_t { CY\_U3P\_CSI\_PLL\_CLK\_DIV\_8 = 0, CY\_U3P\_CSI\_PLL\_CLK\_DIV\_4, CY\_U3P\_CSI\_PLL\_CLK\_DIV\_2, CY\_U3P\_CSI\_PLL\_CLK\_DIV\_INVALID }**  
*Clock Divider Values.*
- **enum CyU3PMipicsiPllClkFrs\_t { CY\_U3P\_CSI\_PLL\_FRS\_500\_1000M = 0, CY\_U3P\_CSI\_PLL\_FRS\_250\_500M, CY\_U3P\_CSI\_PLL\_FRS\_125\_250M, CY\_U3P\_CSI\_PLL\_FRS\_63\_125M }**  
*Frequency range selection for the MIPI-CSI PLL Clock.*
- **enum CyU3PMipicsiBusWidth\_t { CY\_U3P\_MIPICSI\_BUS\_8 = 0, CY\_U3P\_MIPICSI\_BUS\_16, CY\_U3P\_MIPICSI\_BUS\_24 }**  
*Bus Widths supported by the fixed function GPIF interface.*

## Functions

- `CyU3PReturnStatus_t CyU3PMipicsiInit (void)`  
*Initialize MIPI-CSI block.*
- `CyU3PReturnStatus_t CyU3PMipicsiDeInit (void)`  
*De-Initialize MIPI-CSI block.*
- `CyU3PReturnStatus_t CyU3PMipicsiSetIntfParams (CyU3PMipicsiCfg_t *csiCfg, CyBool_t wakeOnConfigure)`  
*Configure MIPI-CSI interface.*
- `CyU3PReturnStatus_t CyU3PMipicsiQueryIntfParams (CyU3PMipicsiCfg_t *csiCfg)`  
*Read MIPI-CSI interface configuration from the block.*
- `CyU3PReturnStatus_t CyU3PMipicsiSetPhyTimeDelay (uint8_t tdTerm, uint8_t thsSettleDelay)`  
*Setup MIPI CSI-2 Receiver PHY Time Delay register.*
- `CyU3PReturnStatus_t CyU3PMipicsiReset (CyU3PMipicsiReset_t resetType)`  
*Reset the MIPI-CSI interface.*
- `void CyU3PCx3DeviceReset (CyBool_t isWarmReset, CyBool_t sensorResetHigh)`  
*Reset the CX3 Device.*
- `CyU3PReturnStatus_t CyU3PMipicsiGetErrors (CyBool_t clrErrCnts, CyU3PMipicsiErrorCounts_t *errorCounts)`  
*Get MIPI-CSI block Phy error counts.*
- `CyU3PReturnStatus_t CyU3PMipicsiWakeup (void)`  
*MIPI-CSI block Wakeup.*
- `CyU3PReturnStatus_t CyU3PMipicsiSleep (void)`  
*Mipi-CSI block Sleep.*
- `CyU3PReturnStatus_t CyU3PMipicsiSetSensorControl (CyU3PMipicsiSensorIo_t io, CyBool_t value)`  
*Sensor XRES and XSHUTDOWN Signals.*
- `CyBool_t CyU3PMipicsiCheckBlockActive (void)`  
*Check if MIPI-CSI interface is Active or in Low power sleep.*
- `CyU3PReturnStatus_t CyU3PMipicsiInitializeGPIO (void)`  
*Initialize the GPIO block on the CX3.*
- `CyU3PReturnStatus_t CyU3PMipicsiInitializeI2c (CyU3PMipicsiI2cFreq_t freq)`  
*Configure and Initialize the I2C Block on the CX3.*
- `CyU3PReturnStatus_t CyU3PMipicsiInitializePIB (void)`  
*Initialize and configure the PIB block on the CX3.*
- `CyU3PReturnStatus_t CyU3PMipicsiGpiLoad (CyU3PMipicsiBusWidth_t busWidth, uint32_t bufferSize)`  
*Fixed Function GPIF Waveform Load.*

### 5.25.1 Detailed Description

This file contains the MIPI-CSI interface management data structures, functions and macros for CX3 devices.

### 5.25.2 Macro Definition Documentation

#### 5.25.2.1 #define ALPHA\_CX3\_START\_SCK0 0x0

Initial value of early outputs from the CX3 GPIF state machine.

#### 5.25.2.2 #define ALPHA\_CX3\_START\_SCK1 0x0

Initial value of early outputs from the CX3 GPIF state machine.

5.25.2.3 `#define CX3_ALPHA_START ALPHA_CX3_START_SCK0`

Mapping for legacy value from SDK 1.3

5.25.2.4 `#define CX3_FULL_BUFFER_IN_SCK0 11`

Frame end with full buffer in socket 0.

5.25.2.5 `#define CX3_FULL_BUFFER_IN_SCK1 12`

Frame end with full buffer in socket 1.

5.25.2.6 `#define CX3_IDLE CX3_IDLE_SCK0`

Mapping for legacy value from SDK 1.3

5.25.2.7 `#define CX3_IDLE_SCK0 1`

Delay state #1.

5.25.2.8 `#define CX3_IDLE_SCK1 16`

Delay state #2.

5.25.2.9 `#define CX3_NUMBER_OF_STATES 18`

Number of states in the CX3 GPIO state machine

5.25.2.10 `#define CX3_PARTIAL_BUFFER_IN_SCK0 9`

Frame end with partial buffer in socket 0.

5.25.2.11 `#define CX3_PARTIAL_BUFFER_IN_SCK1 10`

Frame end with partial buffer in socket 1.

5.25.2.12 `#define CX3_PUSH_DATA_SCK0 3`

Push data into socket 0.

5.25.2.13 `#define CX3_PUSH_DATA_SCK1 4`

Push data into socket 1.

5.25.2.14 `#define CX3_PUSH_DATA_TO_SCK0 CX3_PUSH_DATA_SCK0`

Mapping for legacy value from SDK 1.3

5.25.2.15 #define CX3\_PUSH\_DATA\_TO\_SCK1 CX3\_PUSH\_DATA\_SCK1

Mapping for legacy value from SDK 1.3

5.25.2.16 #define CX3\_START CX3\_START\_SCK0

Mapping for legacy value from SDK 1.3

5.25.2.17 #define CX3\_START\_SCK0 0

The following macros define the various states supported by the fixed function GPIF on the CX3 parts.

### 5.25.3 Fixed Function GPIF States

5.25.3.1 #define CX3\_START\_SCK1 15

Start of data capture into GPIF socket 1.

5.25.3.2 #define CX3\_WAIT\_FOR\_FRAME\_START CX3\_WAIT\_FOR\_FRAME\_START\_SCK0

Mapping for legacy value from SDK 1.3

5.25.3.3 #define CX3\_WAIT\_FOR\_FRAME\_START\_SCK0 2

Wait for frame start.

5.25.3.4 #define CX3\_WAIT\_FOR\_FRAME\_START\_SCK1 17

Wait for frame start.

5.25.3.5 #define CX3\_WAIT\_FULL\_SCK0\_NEXT\_SCK1 6

Line blanking and socket 0 full.

5.25.3.6 #define CX3\_WAIT\_FULL\_SCK1\_NEXT\_SCK0 8

Line blanking and socket 1 full.

5.25.3.7 #define CX3\_WAIT\_TO\_FILL\_SCK0 5

Line blanking while filling socket 0.

5.25.3.8 #define CX3\_WAIT\_TO\_FILL\_SCK1 7

Line blanking while filling socket 1.

5.25.3.9 #define FW\_WAIT\_SCK0 13

Wait for firmware trigger to move to socket 1.

### 5.25.3.10 #define FW\_WAIT\_SCK1 14

Wait for firmware trigger to move to socket 0.

## 5.25.4 Typedef Documentation

### 5.25.4.1 typedef enum CyU3PMipicsiBusWidth\_t CyU3PMipicsiBusWidth\_t

Bus Widths supported by the fixed function GPIF interface.

#### Description

This enumeration defines the bus widths supported by the fixed function GPIF interface on the CX3 parts. CX3 supports bus widths of 8-Bits, 16-Bits and 24-Bits.

#### Note

The DMA buffer size being used to transfer data from the GPIF interface must be aligned to the data width of the interface used. The data buffer size in bytes must be a multiple of 16 for 16-Bit interfaces and a multiple of 24 for 24-Bit buffers.

#### See also

[CyU3PMipicsiGpifLoad](#)

### 5.25.4.2 typedef struct CyU3PMipicsiCfg\_t CyU3PMipicsiCfg\_t

Structure defining the MIPI-CSI block interface configuration.

#### Description

This structure encapsulates all the configurable parameters that can be selected for the MIPI-CSI interface. The [CyU3PMipicsiSetIntfParams\(\)](#) function accepts a pointer to this structure and updates the interface parameters.

#### Note

This structure has changed from the 1.3 SDK release (CX3 BETA release). If you are using CX3 code from the 1.3 SDK release, please update your code to use the current version of this structure. The changes between the 1.3 release and 1.3.1 release are as follows: 1) The order of structure members has changed. 2) A new member fifoDelay has been added. 3) The names for some members has changed (ppiClkDiv is now csiRxClkDiv, and sClkDiv is now parClkDiv).

#### See also

[CyU3PMipicsiSetIntfParams](#)  
[CyU3PMipicsiQueryIntfParams](#)  
[CyU3PMipicsiPliClkFrs\\_t](#)  
[CyU3PMipicsiPliClkDiv\\_t](#)

### 5.25.4.3 typedef enum CyU3PMipicsiDataFormat\_t CyU3PMipicsiDataFormat\_t

MIPI-CSI Data Formats.

#### Description

This enumerated type lists the MIPI CSI data formats supported by the MIPI-CSI block on the CX3. The MIPI-CSI block on the CX3 supports the listed data formats in 8, 16 and 24 bit modes. Some data formats (RAW8, RG↔B888) support only a fixed data width, while some formats support more than one data widths with different padding mechanisms and byte orders on the received data.

#### Note

The user needs to set the Fixed Function GPIF interface on the CX3 to an appropriate width based on the Data Format being selected. Selection of an incorrect GPIF bus width can lead to loss of data or introduction of garbage data into the data stream.

See also

[CyU3PMipicsiCfg\\_t](#)  
[CyU3PMipicsiSetIntfParams](#)  
[CyU3PMipicsiQueryIntfParams](#)

#### 5.25.4.4 `typedef struct CyU3PMipicsiErrorCounts_t CyU3PMipicsiErrorCounts_t`

Structure defining MIPI-CSI block Phy errors.

**Description**

The following structure is used to fetch count of MIPI-CSI Phy level errors from the MIPI-CSI block on the CX3.

See also

[CyU3PMipicsiGetErrors](#)

#### 5.25.4.5 `typedef enum CyU3PMipicsiI2cFreq_t CyU3PMipicsiI2cFreq_t`

I2C frequency for MIPI-CSI interface communication.

**Description**

This enumeration defines the I2C frequency for communication with the Image Sensor and the MIPI-CSI interface. The CX3 part supports communication at 100KHz and 400 KHz.

See also

[CyU3PMipicsiInitializeI2c](#)

#### 5.25.4.6 `typedef enum CyU3PMipicsiPIIClkDiv_t CyU3PMipicsiPIIClkDiv_t`

Clock Divider Values.

**Description**

This enumerated type lists Clock divider values permitted for the various clocks on the MIPI-CSI block of the CX3 device. The clocks on MIPI-CSI block are derived from a primary PLL clock which is generated using 19.2 MHz System Reference Clock.

See also

[CyU3PMipicsiCfg\\_t](#)  
[CyU3PMipicsiSetIntfParams](#)  
[CyU3PMipicsiQueryIntfParams](#)

#### 5.25.4.7 `typedef enum CyU3PMipicsiPIIClkFrs_t CyU3PMipicsiPIIClkFrs_t`

Frequency range selection for the MIPI-CSI PLL Clock.

**Description**

This enumeration is used to define the frequency rage in which the PLL clock on the MIPI-CSI block is operating.

See also

[CyU3PMipicsiCfg\\_t](#)  
[CyU3PMipicsiSetIntfParams](#)  
[CyU3PMipicsiQueryIntfParams](#)

#### 5.25.4.8 `typedef enum CyU3PMipicsiReset_t CyU3PMipicsiReset_t`

MIPI-CSI interface Reset Type.

##### Description

This enumerated type lists the reset types supported for the MIPI-CSI interface block on the CX3. The MIPI-CSI interface provides two reset modes - Hard reset, which power cycles the entire block, and Soft reset, which does not reset the I2C communication channel used by the block.

##### See also

[CyU3PMipicsiReset](#)

#### 5.25.4.9 `typedef enum CyU3PMipicsiSensorIo_t CyU3PMipicsiSensorIo_t`

MIPI Control lines for Image Sensor.

##### Description

This enumeration defines the IO lines for MIPI XReset and XShutdown signals. The user drives these lines high or low using the `CyU3PMipicsiSetSensorControl` function

##### See also

[CyU3PMipicsiSetSensorControl](#)

### 5.25.5 Enumeration Type Documentation

#### 5.25.5.1 `enum CyU3PMipicsiBusWidth_t`

Bus Widths supported by the fixed function GPIF interface.

##### Description

This enumeration defines the bus widths supported by the fixed function GPIF interface on the CX3 parts. CX3 supports bus widths of 8-Bits, 16-Bits and 24-Bits.

##### Note

The DMA buffer size being used to transfer data from the GPIF interface must be aligned to the data width of the interface used. The data buffer size in bytes must be a multiple of 16 for 16-Bit interfaces and a multiple of 24 for 24-Bit buffers.

##### See also

[CyU3PMipicsiGpifLoad](#)

##### Enumerator

- `CY_U3P_MIPICSI_BUS_8` Use an 8-Bit data bus
- `CY_U3P_MIPICSI_BUS_16` Use a 16-Bit data bus
- `CY_U3P_MIPICSI_BUS_24` Use a 24-Bit data bus

#### 5.25.5.2 `enum CyU3PMipicsiDataFormat_t`

MIPI-CSI Data Formats.

##### Description

This enumerated type lists the MIPI CSI data formats supported by the MIPI-CSI block on the CX3. The MIPI-CSI block on the CX3 supports the listed data formats in 8, 16 and 24 bit modes. Some data formats (RAW8, RG $\leftrightarrow$ B888) support only a fixed data width, while some formats support more than one data widths with different padding mechanisms and byte orders on the received data.

**Note**

The user needs to set the Fixed Function GPIF interface on the CX3 to an appropriate width based on the Data Format being selected. Selection of an incorrect GPIF bus width can lead to loss of data or introduction of garbage data into the data stream.

**See also**

[CyU3PMipicsiCfg\\_t](#)  
[CyU3PMipicsiSetIntfParams](#)  
[CyU3PMipicsiQueryIntfParams](#)

**Enumerator**

**CY\_U3P\_CSI\_DF\_RAW8** RAW8 Data Type. CSI-2 Data Type 0x2A  
 8 Bit Output = RAW[7:0]

**CY\_U3P\_CSI\_DF\_RAW10** RAW10 Data Type. CSI-2 Data Type 0x2B  
 16 Bit Output = 6'b0,RAW[9:0]

**CY\_U3P\_CSI\_DF\_RAW12** RAW12 Data Type. CSI-2 Data Type 0x2C  
 16 Bit Output = 4'b0,RAW[11:0]

**CY\_U3P\_CSI\_DF\_RAW14** RAW14 Data Type. CSI-2 Data Type 0x2D  
 16 Bit Output = 2'b0,RAW[13:0]

**CY\_U3P\_CSI\_DF\_RGB888** RGB888 Data type. CSI-2 Data Type 0x24.  
 24 Bit Output = R[7:0],G[7:0],B[7:0]

**CY\_U3P\_CSI\_DF\_RGB666\_0** RGB666 Data type. CSI-2 Data Type 0x23.  
 24 Bit Output = 2'b0,R[5:0],2'b0,G[5:0], 2'b0,B[5:0]

**CY\_U3P\_CSI\_DF\_RGB666\_1** RGB666 Data type. CSI-2 Data Type 0x23.  
 24 Bit Output = 6'b0,R[5:0],G[5:0], B[5:0]

**CY\_U3P\_CSI\_DF\_RGB565\_0** RGB565 Data type. CSI-2 Data Type 0x22.  
 24 Bit Output = 2'b0,R[4:0],3'b0,G[5:0], 2'b0,B[4:0],1'b0

**CY\_U3P\_CSI\_DF\_RGB565\_1** RGB565 Data type. CSI-2 Data Type 0x22.  
 24 Bit Output = 3'b0,R[4:0],2'b0,G[5:0], 3'b0,B[4:0]

**CY\_U3P\_CSI\_DF\_RGB565\_2** RGB565 Data type. CSI-2 Data Type 0x22.  
 24 Bit Output = 8'b0,R[4:0],G[5:0], B[4:0]  
 16 Bit Output = R[4:0],G[5:0], B[4:0]

**CY\_U3P\_CSI\_DF\_YUV422\_8\_0** YUV422 8-Bit Data type. CSI-2 Type 0x1E.  
 24 Bit Output = 16'b0,P[7:0]  
 16 Bit Output = 8'b0, P[7:0]  
 8 Bit Output = P[7:0]  
 Data Order: U1,Y1,V1,Y2,U3,Y3,V3,Y4....

**CY\_U3P\_CSI\_DF\_YUV422\_8\_1** YUV422 8-Bit Data type. CSI-2 Type 0x1E.  
 24 Bit Output = 8'b0,P[15:0]  
 16 Bit Output = P[15:0]  
 Data Order: {U1,Y1},{V1,Y2},{U3,Y3},{V3,Y4}....

**CY\_U3P\_CSI\_DF\_YUV422\_8\_2** YUV422 8-Bit Data type. CSI-2 Type 0x1E.  
 24 Bit Output = 8'b0,P[15:0]  
 16 Bit Output = P[15:0]  
 Data Order: {Y1,U1},{Y2,V1},{Y3,U3},{Y4,V3}....

**CY\_U3P\_CSI\_DF\_YUV422\_10** YUV422 10-Bit Data type. CSI-2 Type 0x1F.  
 24 Bit Output = 14'b0,P[9:0]  
 16 Bit Output = 6'b0,P[9:0]  
 Data Order: U1,Y1,V1,Y2,U3,Y3,V3,Y4....

### 5.25.5.3 enum CyU3PMipicsil2cFreq\_t

I2C frequency for MIPI-CSI interface communication.

#### Description

This enumeration defines the I2C frequency for communication with the Image Sensor and the MIPI-CSI interface. The CX3 part supports communication at 100KHz and 400 KHz.

#### See also

[CyU3PMipicsiInitializeI2c](#)

#### Enumerator

**CY\_U3P\_MIPICSI\_I2C\_100KHZ** 100 KHz operation

**CY\_U3P\_MIPICSI\_I2C\_400KHZ** 400 KHz operation

### 5.25.5.4 enum CyU3PMipicsiPiIClkDiv\_t

Clock Divider Values.

#### Description

This enumerated type lists Clock divider values permitted for the various clocks on the MIPI-CSI block of the CX3 device. The clocks on MIPI-CSI block are derived from a primary PLL clock which is generated using 19.2 MHz System Reference Clock.

#### See also

[CyU3PMipicsiCfg\\_t](#)

[CyU3PMipicsiSetIntfParams](#)

[CyU3PMipicsiQueryIntfParams](#)

#### Enumerator

**CY\_U3P\_CSI\_PLL\_CLK\_DIV\_8** Clk = PLL\_CI<sub>K</sub>/8

**CY\_U3P\_CSI\_PLL\_CLK\_DIV\_4** Clk = PLL\_CI<sub>K</sub>/8

**CY\_U3P\_CSI\_PLL\_CLK\_DIV\_2** Clk = PLL\_CI<sub>K</sub>/8

**CY\_U3P\_CSI\_PLL\_CLK\_DIV\_INVALID** Invalid Value

### 5.25.5.5 enum CyU3PMipicsiPiIClkFrs\_t

Frequency range selection for the MIPI-CSI PLL Clock.

#### Description

This enumeration is used to define the frequency rage in which the PLL clock on the MIPI-CSI block is operating.

#### See also

[CyU3PMipicsiCfg\\_t](#)

[CyU3PMipicsiSetIntfParams](#)

[CyU3PMipicsiQueryIntfParams](#)

#### Enumerator

**CY\_U3P\_CSI\_PLL\_FRS\_500\_1000M** PLL Clock output range 500MHz-1GHz

**CY\_U3P\_CSI\_PLL\_FRS\_250\_500M** PLL Clock output range 250-500MHz

**CY\_U3P\_CSI\_PLL\_FRS\_125\_250M** PLL Clock output range 125-250MHz

**CY\_U3P\_CSI\_PLL\_FRS\_63\_125M** PLL Clock output range 62.5-125MHz

### 5.25.5.6 enum CyU3PMipicsiReset\_t

MIPI-CSI interface Reset Type.

#### Description

This enumerated type lists the reset types supported for the MIPI-CSI interface block on the CX3. The MIPI-CSI interface provides two reset modes - Hard reset, which power cycles the entire block, and Soft reset, which does not reset the I2C communication channel used by the block.

#### See also

[CyU3PMipicsiReset](#)

#### Enumerator

**CY\_U3P\_CSI\_SOFT\_RST** Perform a soft reset on the MIPI-CSI interface block

**CY\_U3P\_CSI\_HARD\_RST** Perform a hard reset on the MIPI-CSI interface block

### 5.25.5.7 enum CyU3PMipicsiSensorIo\_t

MIPI Control lines for Image Sensor.

#### Description

This enumeration defines the IO lines for MIPI XReset and XShutdown signals. The user drives these lines high or low using the CyU3PMipicsiSetSensorControl function

#### See also

[CyU3PMipicsiSetSensorControl](#)

#### Enumerator

**CY\_U3P\_CSI\_IO\_XRES** Image Sensor Reset IO (XRES)

**CY\_U3P\_CSI\_IO\_XSHUTDOWN** Image Sensor Shutdown IO (XSHUTDOWN)

## 5.25.6 Function Documentation

### 5.25.6.1 void CyU3PCx3DeviceReset ( **CyBool\_t isWarmReset**, **CyBool\_t sensorResetHigh** )

Reset the CX3 Device.

#### Description

This function is similar to the CyU3PDeviceReset function used for FX3/FX3S devices. In addition to resetting the CX3 device, this function also drives the XRES line output to reset the Image Sensor before the CX3 reset.

#### Return value

None as this function does not return.

#### See also

[CyU3PDeviceReset](#)

#### Parameters

<b>isWarmReset</b>	Whether this should be a warm reset or a cold reset. In the case of a warm reset, the previously loaded firmware will start executing again. In the case of a cold reset, the firmware download to CX3 needs to be performed again.
<b>sensorResetHigh</b>	True if Sensor is to be reset by driving XRES High, False if Sensor reset needs XRES as low.

### 5.25.6.2 CyBool\_t CyU3PMipicsiCheckBlockActive ( void )

Check if MIPI-CSI interface is Active or in Low power sleep.

#### Description

This function is used to check if the MIPI-CSI interface is Active or in low power sleep.

#### Return value

CyTrue - If the interface block is Active.

CyFalse - If the interface block is in Low power sleep

### 5.25.6.3 CyU3PReturnStatus\_t CyU3PMipicsiDelInit ( void )

De-Initialize MIPI-CSI block.

#### Description

This function de-initializes MIPI-CSI interface block on the CX3 device and is expected to be called prior to calling CyU3PSysEnterStandbyMode. This function should be called before the I2C block or GPIO blocks are de-initialized. MIPI XReset and XShutdown signals shall not be driven by the CX3 after this call.

#### Return value

CY\_U3P\_SUCCESS - If the operation completes successfully.

CY\_U3P\_ERROR\_TIMEOUT - If an I2C read/write timeout occurs during the operation.

CY\_U3P\_ERROR\_NOT\_SUPPORTED - If the part on which this function is called is not a CX3 device.

CY\_U3P\_ERROR\_UNKNOWN - If a general or unknown error occurred during the operation.

#### See also

[CyU3PSysEnterStandbyMode](#)

### 5.25.6.4 CyU3PReturnStatus\_t CyU3PMipicsiGetErrors ( CyBool\_t clrErrCnts, CyU3PMipicsiErrorCounts\_t \* errorCounts )

Get MIPI-CSI block Phy error counts.

#### Description

This function is used to get a count of CSI protocol and physical layer errors from the MIPI-CSI block. The function takes a parameter which determines whether or not the error counts on the interface are cleared. The error counts for each error type is retrieved via an pointer of type [CyU3PMipicsiErrorCounts\\_t](#) passed to this function. The error count values for each type can reach a maximum count of 0xFF. The count values will continue to report the existing error value on each call unless the function explicitly clears the counts using clrErrCnts.

#### Return value

CY\_U3P\_SUCCESS - If the operation completes successfully.

CY\_U3P\_ERROR\_TIMEOUT - If an I2C read/write timeout occurs during the operation.

CY\_U3P\_ERROR\_NOT\_SUPPORTED - If the part on which this function is called is not a CX3 device.

CY\_U3P\_ERROR\_UNKNOWN - If a general or unknown error occurred during the operation.

CY\_U3P\_ERROR\_NOT\_STARTED - If the interface has not been initialized.

CY\_U3P\_ERROR\_NULL\_POINTER - If the clrErrCnts object is uninitialized.

#### See also

[CyU3PMipicsiErrorCounts\\_t](#)

#### Parameters

<i>clrErrCnts</i>	Set to CyTrue to clear the error counts
<i>errorCounts</i>	Error Counts

### 5.25.6.5 CyU3PReturnStatus\_t CyU3PMipicsiGpifLoad ( CyU3PMipicsiBusWidth\_t busWidth, uint32\_t bufferSize )

Fixed Function GPIF Waveform Load.

#### Description

The CX3 has a fixed function GPIF interface designed for Image sensor data acquisition from the MIPI-CSI interface. This function allows selection of the GPIF data bus-width and configuration of the size of the DMA buffer provided for GPIF transfers. The PIB block should have been initialized prior to calling this function.

#### Return value

CY\_U3P\_SUCCESS - If the operation is successful  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - If the part on which this function is called is not a CX3 device.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - If an invalid argument is passed to the function.

CY\_U3P\_ERROR\_UNKNOWN - If a general or unknown error occurred during the operation.

CY\_U3P\_ERROR\_ALREADY\_STARTED - If the GPIF hardware is running.

#### See also

[CyU3PMipicsiBusWidth\\_t](#)

[CyU3PMipicsiInitializePIB](#)

#### Parameters

<i>busWidth</i>	Load GPIF State machine and settings for selected busWidth
<i>bufferSize</i>	Set Data counter based on DMA buffer size being used. This value MUST match the size of the buffer actual data buffer being used i.e. (The size being passed to the CyU3PDmaChannelCreate API) - (The Size of Header and Footer) e.g. If dmaCfg is <a href="#">CyU3PDmaChannelConfig_t</a> object being passed to the ChannelCreate API the value of bufferSize should be (dmaCfg.size - (dmaCfg.prodHeader + dmaCfg.prodFooter))

### 5.25.6.6 CyU3PReturnStatus\_t CyU3PMipicsiInit ( void )

Initialize MIPI-CSI block.

#### Description

This function initializes MIPI-CSI interface block on the CX3 device and is expected to be called prior to any other calls to the MIPI-CSI block. The GPIO Block, the PIB block and the I2C blocks should have been initialized prior to calling this function. Helper functions [CyU3PMipicsiInitializeGPIO\(\)](#), [CyU3PMipicsiInitializePIB\(\)](#) and [CyU3PMipicsiInitializeI2c\(\)](#) have been provided to initialize these blocks to the default values to be used for the MIPI-CSI block. Please see documentation on each of those functions for more information on the specific settings.

The CX3 part uses the I2C block to communicate with both the MIPI-CSI block and any connected Image Sensor. The 7 bit I2C slave address 8b'0000111X (Read 0x0F, Write 0x0E) is used internally by the CX3 part to configure and control the MIPI-CSI interface block and should not be used by any external I2C slaves connected to the CX3 on the I2C bus.

#### Note

This function sets the default state of the MIPI XReset and XShutdown signals as Drive 0. If either of these signals, needs to be in Drive 1 state for sensor operation, it needs to be explicitly set to Drive 1 state using [CyU3PMipicsiSetSensorControl\(\)](#) after calling [CyU3PMipicsiInit\(\)](#). This call leaves the interface in a low-power mode and [CyU3PMipicsiWakeup\(\)](#) should be called to enable the clocks on the MIPI-CSI interface.

#### Return value

CY\_U3P\_SUCCESS - If the operation completes successfully.

CY\_U3P\_ERROR\_TIMEOUT - If an I2C read/write timeout occurs during the operation.

CY\_U3P\_ERROR\_NOT\_SUPPORTED - If the part on which this function is called is not a CX3 device.

CY\_U3P\_ERROR\_UNKNOWN - If a general or unknown error occurred during the operation.

**See also**

[CyU3PMipicsiInitializeI2c](#)  
[CyU3PMipicsiInitializeGPIO](#)  
[CyU3PMipicsiInitializePIB](#)  
[CyU3PCx3DeviceReset](#)  
[CyU3PMipicsiSetSensorControl](#)

**5.25.6.7 CyU3PReturnStatus\_t CyU3PMipicsiInitializeGPIO ( void )**

Initialize the GPIO block on the CX3.

**Description**

This function is a helper routine to initialize the GPIO block with default values for the CX3 device.

The function internally calls CyU3PGpioInit with the following parameters:

[CyU3PGpioClock\\_t](#) settings:

```
fastClkDiv = 2;
slowClkDiv = 0;
simpleDiv = CY_U3P_GPIO_SIMPLE_DIV_BY_2;
clkSrc = CY_U3P_SYS_CLK;
halfDiv = 0;
Callback set to NULL.
```

This function should be called before initializing the PIB block or calling [CyU3PMipicsiInit\(\)](#). In case different parameters are required, the application can use its own code to initialize the GPIO block prior to initializing the PIB block or calling [CyU3PMipicsiInit\(\)](#).

**Return value**

CY\_U3P\_SUCCESS - If the GPIO initialization is successful CY\_U3P\_ERROR\_ALREADY\_STARTED - If the GPIO block has already been initialized

**See also**

[CyU3PGpioInit](#)

**5.25.6.8 CyU3PReturnStatus\_t CyU3PMipicsiInitializeI2c ( CyU3PMipicsiI2cFreq\_t freq )**

Configure and Initialize the I2C Block on the CX3.

**Description**

This function is a helper routine to initialize the I2C block to either 100KHz or 400KHz for the CX3 device.

The function internally calls CyU3PI2cInit and then calls CyU3PI2cSetConfig to configure the I2C block with the following parameters:

[CyU3PI2cConfig\\_t](#) settings:

```
bitRate = 100000 or 400000 depending on freq
isDma = CyFalse;
busTimeout = 0xFFFFFFFF
dmaTimeout = 0xFFFF;
Callback is set to NULL.
```

The I2C block needs to be initialized prior to calling [CyU3PMipicsiInit\(\)](#). In case different parameters are required, the application can use its own code to initialize the I2C block prior to calling [CyU3PMipicsiInit\(\)](#) but must ensure that the block is initialized to either 100KHz or 400KHz.

**Note**

If the I2C block is already initialized before this function is called, it will override the current configuration of the I2C block with the configuration specified in the description of this function.

**Return value**

CY\_U3P\_SUCCESS - If the initialization is successful CY\_U3P\_ERROR\_NOT\_CONFIGURED - When I2C has

not been enabled in IO configuration. CY\_U3P\_ERROR\_TIMEOUT - When there is timeout happening during configuration CY\_U3P\_ERROR\_MUTEX\_FAILURE- When there is a failure in acquiring a mutex lock

#### See also

[CyU3PI2cInit](#)  
[CyU3PI2cSetConfig](#)

#### Parameters

<i>freq</i>	Frequency 100KHz or 400KHz
-------------	----------------------------

#### 5.25.6.9 CyU3PReturnStatus\_t CyU3PMipicsiInitializePIB ( void )

Initialize and configure the PIB block on the CX3.

#### Description

This function is a helper routine to initialize the PIB block on the CX3 part used by the fixed function GPIF .

The function internally calls [CyU3PPibInit\(\)](#) with the following parameters:

[CyU3PPibClock\\_t](#) settings:  
clkDiv = 2;  
clkSrc = CY\_U3P\_SYS\_CLK  
isHalfDiv = CyFalse;  
isDIIEnable = CyFalse;  
dolnIt is set to CyTrue.

The PIB block needs to be initialized, prior to calling [CyU3PMipicsiGpifLoad\(\)](#), to configure the fixed function GPIF for the CX3 part.

#### Return value

CY\_U3P\_SUCCESS - If the operation is successful

#### See also

[CyU3PPibInit](#)

#### 5.25.6.10 CyU3PReturnStatus\_t CyU3PMipicsiQueryIntfParams ( CyU3PMipicsiCfg\_t \* csiCfg )

Read MIPI-CSI interface configuration from the block.

#### Description

This function is used to read back the MIPI-CSI interface parameters from the block. The parameters read back are provided to the calling function via the pointer of type [CyU3PMipicsiCfg\\_t](#) passed in from the calling function.

#### Note

Memory for the pointer csiCfg must be allocated in the calling function.

#### Return value

CY\_U3P\_SUCCESS if the operation completes successfully.  
CY\_U3P\_ERROR\_TIMEOUT - If an I2C read/write timeout occurs during the operation.  
CY\_U3P\_ERROR\_NOT\_SUPPORTED - If the part on which this function is called is not a CX3 device.  
CY\_U3P\_ERROR\_BAD\_ARGUMENT - If an invalid argument is read from the interface.  
CY\_U3P\_ERROR\_UNKNOWN - If a general or unknown error occurred during the operation.  
CY\_U3P\_ERROR\_NOT\_STARTED - If the interface has not been initialized.  
CY\_U3P\_ERROR\_NULL\_POINTER - If the csiCfg object is uninitialized.

**See also**

[CyU3PMipicsiCfg\\_t](#)  
[CyU3PMipicsiQueryIntfParams](#)

**Parameters**

<i>csiCfg</i>	Configuration Data read back from the MIPI-CSI interface
---------------	--

**5.25.6.11 CyU3PReturnStatus\_t CyU3PMipicsiReset ( CyU3PMipicsiReset\_t resetType )**

Reset the MIPI-CSI interface.

**Description**

This function is used to reset the MIPI-CSI block on the CX3. The MIPI-CSI block provides two reset modes - Hard reset, which power-cycles the entire block, and Soft reset, which does not reset the I2C communication channel used by the block. The reset mode is selected via the parameter of type [CyU3PMipicsiReset\\_t](#) passed to this function.

**Note**

The Soft Reset cannot be called on the interface until the block has been initialized. A Hard reset however, can be called on the interface at any point in time. The [CyU3PMipicsiInit\(\)](#) call internally performs a Hard reset on the interface before initializing it. A Hard reset sets the default state of the MIPI XReset and XShutdown signals as Drive 0. If either of these signals, needs to be in Drive 1 state for sensor operation, it needs to be explicitly set to Drive 1 state using [CyU3PMipicsiSetSensorControl\(\)](#) after calling [CyU3PMipicsiReset\(CY\\_U3P\\_CSI\\_HARD\\_RST\)](#). A Soft reset does not change the state of the XReset and XShutdown signals.

**Return value**

[CY\\_U3P\\_SUCCESS](#) if the operation completes successfully.  
[CY\\_U3P\\_ERROR\\_TIMEOUT](#) - If an I2C read/write timeout occurs during the operation.  
[CY\\_U3P\\_ERROR\\_NOT\\_SUPPORTED](#) - If the part on which this function is called is not a CX3 device.  
[CY\\_U3P\\_ERROR\\_BAD\\_ARGUMENT](#) - If an invalid argument is passed to the function.  
[CY\\_U3P\\_ERROR\\_UNKNOWN](#) - If a general or unknown error occurred during the operation.  
[CY\\_U3P\\_ERROR\\_NOT\\_STARTED](#) - If the interface has not been initialized (Soft Reset Only).

**See also**

[CyU3PMipicsiInit](#)  
[CyU3PMipicsiReset\\_t](#)

**Parameters**

<i>resetType</i>	Reset Type
------------------	------------

**5.25.6.12 CyU3PReturnStatus\_t CyU3PMipicsiSetIntfParams ( CyU3PMipicsiCfg\_t \* csiCfg, CyBool\_t wakeOnConfigure )**

Configure MIPI-CSI interface.

**Description**

This function is used to configure the MIPI-CSI interface parameters over I2C. The function takes in an object of the type [CyU3PMipicsiCfg\\_t](#) and configures the MIPI-CSI interface. The function powers off the interface clocks before changing/setting the configuration. Parameter [wakeOnConfigure](#) is used to turn the clocks ON immediately after the configuration has been completed or to leave the clocks powered down. If the clocks are left powered down, [CyU3PMipicsiWakeup\(\)](#) should be called to start the clocks. The state of the interface (Active or Powered down) can be determined using the [CyU3PMipicsiCheckBlockActive\(\)](#) call.

**Note**

This function can be called when the interface has been put into low power mode using the [CyU3PMipicsiSleep\(\)](#) function.

**Return value**

CY\_U3P\_SUCCESS - If the operation completes successfully.  
 CY\_U3P\_ERROR\_TIMEOUT - If an I2C read/write timeout occurs during the operation.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - If the part on which this function is called is not a CX3 device.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If an invalid argument is passed to this function.  
 CY\_U3P\_ERROR\_UNKNOWN - If a general or unknown error occurred during the operation.  
 CY\_U3P\_ERROR\_NOT\_STARTED - If the interface has not been initialized.  
 CY\_U3P\_ERROR\_NULL\_POINTER - If the csiCfg object is uninitialized.

**See also**

[CyU3PMipicsiCfg\\_t](#)  
[CyU3PMipicsiQueryIntfParams](#)  
[CyU3PMipicsiCheckBlockActive](#)  
[CyU3PMipicsiWakeup](#)

**Parameters**

<i>csiCfg</i>	Configuration Data to be set to the MIPI-CSI interface
<i>wakeOnConfigure</i>	Start the MIPI-CSI interface clocks immediately after configuring If wakeOnConfigure is CyFalse, <a href="#">CyU3PMipicsiWakeup()</a> must be called to start the clocks when ready for transfers.

**5.25.6.13 CyU3PReturnStatus\_t CyU3PMipicsiSetPhyTimeDelay ( uint8\_t *tdTerm*, uint8\_t *thsSettleDelay* )**

Setup MIPI CSI-2 Reciever PHY Time Delay register.

**Description**

This function is used to set the CX3\_PHY\_TIME\_DELAY register values Please refer to CX3 TRM for details on this register.

**Note**

This function should be not be called while the MIPI-CSI PLL clocks are active. Either call after calling [CyU3PMipicsiSetIntfParams\(\)](#) with wakeOnConfigure set to False (before calling [CyU3PMipicsiWakeup\(\)](#) ), or call [CyU3PMipicsiSleep\(\)](#) before calling this API.

**Return value**

CY\_U3P\_SUCCESS if the operation completes successfully.  
 CY\_U3P\_ERROR\_TIMEOUT - If an I2C read/write timeout occurs during the operation.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If an invalid argument is read from the interface.  
 CY\_U3P\_ERROR\_UNKNOWN - If a general or unknown error occurred during the operation.  
 CY\_U3P\_ERROR\_NOT\_STARTED - If the interface has not been initialized.

**See also**

[CyU3PMipicsiQueryIntfParams](#)

**Parameters**

<i>tdTerm</i>	TD TERM Selection for MIPI CSI-2 reciever PHY. Valid values 0 and 1.
<i>thsSettleDelay</i>	THS Settle timer delay value. Valid range 0x00-0x7F.

### 5.25.6.14 CyU3PReturnStatus\_t CyU3PMipicsiSetSensorControl ( CyU3PMipicsiSensorIo\_t io, CyBool\_t value )

Sensor XRES and XSHUTDOWN Signals.

#### Description

This function is used to drive the XRES and XSHUTDOWN signals from the CX3 to Image sensor. The function allows for the signals to be driven high or low. The function can drive either one of the two signals or both signals (both driven high or both driven low) simultaneously. To set both signals to the same value use the mask (CY\_U3P\_CSI\_IO\_XRES | CY\_U3P\_CSI\_IO\_XSHUTDOWN) as value for io. To set the two signals to separate values multiple calls to this function are required (once for each signal).

#### Return value

CY\_U3P\_SUCCESS - If the operation completes successfully.  
 CY\_U3P\_ERROR\_TIMEOUT - If an I2C read/write timeout occurs during the operation.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - If the part on which this function is called is not a CX3 device.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - If an invalid argument is passed to the function.  
 CY\_U3P\_ERROR\_UNKNOWN - If a general or unknown error occurred during the operation.  
 CY\_U3P\_ERROR\_NOT\_STARTED - If the interface has not been initialized (Soft Reset Only).

#### See also

[CyU3PMipicsiSensorIo\\_t](#)

#### Parameters

<i>io</i>	IO to be driven.
<i>value</i>	CyTrue to drive Signal High, CyFalse for Low.

### 5.25.6.15 CyU3PReturnStatus\_t CyU3PMipicsiSleep ( void )

Mipi-CSI block Sleep.

#### Description

This function is used to disable the PLL clocks on the MIPI-CSI interface block and place it in low-power sleep. No data transfers from the Image Sensor to the CX3 will occur while the block is in low power sleep mode.

#### Return value

CY\_U3P\_SUCCESS - If the operation completes successfully.  
 CY\_U3P\_ERROR\_TIMEOUT - If an I2C read/write timeout occurs during the operation.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - If the part on which this function is called is not a CX3 device.  
 CY\_U3P\_ERROR\_UNKNOWN - If a general or unknown error occurred during the operation.  
 CY\_U3P\_ERROR\_NOT\_STARTED - If the interface has not been initialized.

#### See also

[CyU3PMipicsiWakeup](#)

### 5.25.6.16 CyU3PReturnStatus\_t CyU3PMipicsiWakeup ( void )

MIPI-CSI block Wakeup.

#### Description

This function is used enable the clocks on the MIPI-CSI interface block to take it from Low power sleep to Active.

#### Return value

CY\_U3P\_SUCCESS - If the operation completes successfully.

CY\_U3P\_ERROR\_TIMEOUT - If an I2C read/write timeout occurs during the operation.  
CY\_U3P\_ERROR\_NOT\_SUPPORTED - If the part on which this function is called is not a CX3 device.  
CY\_U3P\_ERROR\_UNKNOWN - If a general or unknown error occurred during the operation.  
CY\_U3P\_ERROR\_NOT\_STARTED - If the interface has not been initialized.

#### See also

[CyU3PMipicsiSleep](#)

## 5.26 firmware/u3p\_firmware/inc/cyu3mmu.h File Reference

Cache and Memory Management for the FX3 firmware.

```
#include "cyu3types.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

#### Macros

- #define CYU3P\_ITCM\_BASE\_ADDR (0x00000000)
- #define CYU3P\_ITCM\_SIZE (0x00004000)
- #define CYU3P\_ITCM\_SZ\_EN (0x00000015)
- #define CYU3P\_DTCM\_BASE\_ADDR (0x10000000)
- #define CYU3P\_DTCM\_SIZE (0x00002000)
- #define CYU3P\_DTCM\_SZ\_EN (0x00000011)
- #define CYU3P\_SYSMEM\_BASE\_ADDR (0x40000000)
- #define CYU3P\_SYSMEM\_SIZE (0x00080000)
- #define CYU3P\_MMIO\_BASE\_ADDR (0xE0000000)
- #define CYU3P\_MMIO\_SIZE (0x10000000)
- #define CYU3P\_ROM\_BASE\_ADDR (0xF0000000)
- #define CYU3P\_ROM\_SIZE (0x00008000)
- #define CYU3P\_VIC\_BASE\_ADDR (0xFFFFF000)
- #define CYU3P\_VIC\_SIZE (0x00001000)
- #define CYU3P\_GCTL\_PAGE\_TABLE\_ADDR (0xE0058000)
- #define CYU3P\_CACHE\_LINE\_SZ (5)
- #define CYU3P\_CACHE\_SIZE (13)
- #define CYU3P\_CACHE\_NWAYS (2)
- #define CYU3P\_CACHE\_WAY\_SZ (CYU3P\_CACHE\_SIZE - CYU3P\_CACHE\_NWAYS - CYU3P\_CACHE\_LINE\_SZ)  
*Log(2) of the size of a cache way in bytes.*

- #define CYU3P\_ICACHE\_EN\_MASK (0x1000)
- #define CYU3P\_DCACHE\_EN\_MASK (0x04)
- #define CYU3P\_MMU\_EN\_MASK (0x01)
- #define CYU3P\_CACHE\_MMU\_EN\_MASK (0x1005)
- #define CYU3P\_CACHE\_REPLACEMENT\_MASK (0x4000)
- #define CYU3P\_TCMREG\_ADDRESS\_MASK (0xFFFFF000)

## Functions

- void [CyU3PSysDisableICache](#) (void)  
*Disable the instruction cache.*
- void [CyU3PSysDisableDCache](#) (void)  
*Disable the data cache.*
- void [CyU3PSysDisableMMU](#) (void)  
*Disable the MMU.*
- void [CyU3PSysDisableCacheMMU](#) (void)  
*Disable the caches and MMU.*
- void [CyU3PSysEnableICache](#) (void)  
*Enable the instruction cache.*
- void [CyU3PSysEnableDCache](#) (void)  
*Enable the Data Cache.*
- void [CyU3PSysEnableMMU](#) (void)  
*Enable the MMU.*
- void [CyU3PSysEnableCacheMMU](#) (void)  
*Enable the Caches and the MMU.*
- void [CyU3PSysFlushCaches](#) (void)  
*Flush both I and D Caches.*
- void [CyU3PSysFlushICache](#) (void)  
*Flush the I-Cache.*
- void [CyU3PSysFlushDCache](#) (void)  
*Flush the D-Cache.*
- void [CyU3PSysCleanDCache](#) (void)  
*Clean the entire D-Cache.*
- void [CyU3PSysClearDCache](#) (void)  
*Clean and Flush the entire D-Cache.*
- void [CyU3PSysFlushIRegion](#) (uint32\_t \*addr, uint32\_t len)  
*Flush a code region from the I-Cache.*
- void [CyU3PSysFlushDRegion](#) (uint32\_t \*addr, uint32\_t len)  
*Flush a data region from the D-Cache.*
- void [CyU3PSysCleanDRegion](#) (uint32\_t \*addr, uint32\_t len)  
*Clean a data region from the D-Cache.*
- void [CyU3PSysClearDRegion](#) (uint32\_t \*addr, uint32\_t len)  
*Clean and flush a data region from the D-Cache.*
- void [CyU3PSysBarrierSync](#) (void)  
*Memory barrier synchronization function.*
- uint32\_t [CyU3PSysCacheIRegion](#) (uint32\_t \*addr, uint32\_t len)  
*Function to lock a code region into the I-Cache.*
- uint32\_t [CyU3PSysCacheDRegion](#) (uint32\_t \*addr, uint32\_t len)  
*Function to lock a data region into the D-Cache.*
- void [CyU3PSysInitTCMs](#) (void)  
*Function to initialize the TCM regions.*
- void [CyU3PInitPageTable](#) (void)  
*Function to create the page tables for the device memory map.*
- void [CyU3PSysLockTLBEntry](#) (uint32\_t \*addr)  
*Function to lock the TLB entry for a page walk.*
- void [CyU3PSysFlushTLBEntry](#) (uint32\_t \*addr)  
*Invalidate the TLB entry corresponding to a specified MVA.*
- void [CyU3PSysLoadTLB](#) (void)  
*Function to lock all valid page walks into TLB.*

## 5.26.1 Detailed Description

Cache and Memory Management for the FX3 firmware.

### Description

The FX3 device has 8KB of I-cache and 8KB of D-cache. The default cache handling is done internal to the library. The cache and MMU are initialized. The cache control should be done using CyU3PDeviceCacheControl API. This section explains the extended cache functions that can be used for customized / advanced cache handling. Most of these functions are standard implementations for the ARM926EJ-S core.

If the data cache is enabled, then all buffers used for DMA operation has to be 32 byte aligned and 32 byte multiple. This is because the size of the cache line in the core is 32 bytes. The DMA buffers include all buffers used with DMA APIs as well as used with library APIs like CyU3PUsbSetDesc, CyU3PUsbSendEP0Data, CyU3PUsbGetEP0Data, CyU3PUsbHostSendSetupRqt etc.

## 5.26.2 Macro Definition Documentation

### 5.26.2.1 #define CYU3P\_CACHE\_LINE\_SZ (5)

Log(2) of cache line size in bytes.

### 5.26.2.2 #define CYU3P\_CACHE\_MMU\_EN\_MASK (0x1005)

Mask for Cache and MMU enable bits.

### 5.26.2.3 #define CYU3P\_CACHE\_NWAYS (2)

Log(2) of number of cache ways.

### 5.26.2.4 #define CYU3P\_CACHE\_REPLACEMENT\_MASK (0x4000)

Mask for Cache replacement policy bit in cp15:c1

### 5.26.2.5 #define CYU3P\_CACHE\_SIZE (13)

Log(2) of cache size in bytes.

### 5.26.2.6 #define CYU3P\_DCACHE\_EN\_MASK (0x04)

Mask for D-Cache enable bit in cp15:c1

### 5.26.2.7 #define CYU3P\_DTCM\_BASE\_ADDR (0x10000000)

D-TCM base address.

### 5.26.2.8 #define CYU3P\_DTCM\_SIZE (0x00002000)

D-TCM size in bytes.

### 5.26.2.9 #define CYU3P\_DTCM\_SZ\_EN (0x00000011)

D-TCM size in the format for cp15:c9

5.26.2.10 #define CYU3P\_GCTL\_PAGE\_TABLE\_ADDR (0xE0058000)

Address of MMIO mapped page table.

5.26.2.11 #define CYU3P\_ICACHE\_EN\_MASK (0x1000)

Mask for I-Cache enable bit in cp15:c1

5.26.2.12 #define CYU3P\_ITCM\_BASE\_ADDR (0x00000000)

I-TCM base address.

5.26.2.13 #define CYU3P\_ITCM\_SIZE (0x00004000)

I-TCM size in bytes.

5.26.2.22 #define CYU3P\_TCMREG\_ADDRESS\_MASK (0xFFFFF000)

Mask for TCM base address in cp15:c9

5.26.2.23 #define CYU3P\_VIC\_BASE\_ADDR (0xFFFFF000)

VIC base address.

5.26.2.24 #define CYU3P\_VIC\_SIZE (0x00001000)

VIC size in bytes.

### 5.26.3 Function Documentation

5.26.3.1 void CyU3PInitPageTable ( void )

Function to create the page tables for the device memory map.

#### Description

This function uses the GCTL registers to create a one-level (section entries only) page table for the FX3 device memory map. The function is not expected to be explicitly invoked and is called by the library during initialization. It is provided for debug purposes.

#### Return value

None

5.26.3.2 void CyU3PSysBarrierSync ( void )

Memory barrier synchronization function.

#### Description

Function that ensures that all write buffers to memory have been drained. This call can be used as a synchronization barrier.

#### Return value

None

5.26.3.3 uint32\_t CyU3PSysCacheDRegion ( uint32\_t \* addr, uint32\_t len )

Function to lock a data region into the D-Cache.

#### Description

Function to load a specified data region into the D-cache and lock it. A maximum of 3 lock operations can be called, with the size of the data region limited to under 2 KB in each case. The caller should ensure that the D-Cache has been flushed, to avoid the possibility of the content already being in the cache.

#### Return value

The way index into which the region was locked.

#### Parameters

<i>addr</i>	Start address of the region to cache.
<i>len</i>	Length of the region in bytes.

#### 5.26.3.4 uint32\_t CyU3PSysCacheRegion ( uint32\_t \*addr, uint32\_t len )

Function to lock a code region into the I-Cache.

##### Description

Function to load a specified code region into the I-cache and lock it. A maximum of 3 lock operations can be called, with the size of the code region limited to under 2 KB in each case. The caller should ensure that the I-Cache has been flushed, to avoid the possibility of the content already being in the cache.

##### Return value

The way index into which the region was locked.

##### Parameters

<i>addr</i>	Start address of the region to cache.
<i>len</i>	Length of the region in bytes.

#### 5.26.3.5 void CyU3PSysCleanDCache ( void )

Clean the entire D-Cache.

##### Description

Function to clean the entire D-Cache. The Test/Clean functionality of the Arm 926 core is used here. The Cache is not flushed by this function.

##### Return value

None

#### 5.26.3.6 void CyU3PSysCleanDRegion ( uint32\_t \*addr, uint32\_t len )

Clean a data region from the D-Cache.

##### Description

Function to clean a specified data region from the Data Cache. If the D-cache is enabled and the cache is handled by the application, then this API should be invoked before sending any data using DMA.

##### Return value

None

##### Parameters

<i>addr</i>	Start address of the region to clean.
<i>len</i>	Length of the region in bytes.

#### 5.26.3.7 void CyU3PSysClearDCache ( void )

Clean and Flush the entire D-Cache.

##### Description

Function to clean and flush the entire Data cache using the test/clean command.

##### Return value

None

#### 5.26.3.8 void CyU3PSysClearDRegion ( uint32\_t \*addr, uint32\_t len )

Clean and flush a data region from the D-Cache.

**Description**

Function to clean and flush a specified data region from the Data Cache.

**Return value**

None

**Parameters**

<i>addr</i>	Start address of the region to flush.
<i>len</i>	Length of the region in bytes.

**5.26.3.9 void CyU3PSysDisableCacheMMU ( void )**

Disable the caches and MMU.

**Description**

Function to disable the Instruction and Data caches as well as the MMU. This function should not be explicitly called and is invoked from the library during initialization. It is provided for debug purposes only.

**Return value**

None

**See also**

[CyU3PSysDisableCacheMMU](#)  
[CyU3PSysDisableDCache](#)  
[CyU3PSysDisableICache](#)

**5.26.3.10 void CyU3PSysDisableDCache ( void )**

Disable the data cache.

**Description**

Function to disable the Data Cache. Should be called from a privileged mode, after ensuring that the D-Cache has been cleaned and flushed. This function should not be explicitly called. The CyU3PDeviceCacheControl API should be called instead. The D-cache is kept disabled by default at firmware start-up.

**Return value**

None

**See also**

[CyU3PSysDisableCacheMMU](#)  
[CyU3PSysEnableDCache](#)

**5.26.3.11 void CyU3PSysDisableICache ( void )**

Disable the instruction cache.

**Description**

Function to disable the Instruction Cache. Should be called from a privileged mode, after ensuring that the I-Cache has been flushed. This function should not be explicitly called. The CyU3PDeviceCacheControl API should be called instead. The I-cache is kept disabled by default at firmware start-up.

**Return value**

None

See also

[CyU3PSysDisableCacheMMU](#)  
[CyU3PSysEnableICache](#)

#### 5.26.3.12 void CyU3PSysDisableMMU ( void )

Disable the MMU.

**Description**

Function to disable the Memory Management Unit. All memory will be accessed through physical addresses once MMU has been disabled. Caller should ensure that caches have been flushed and disabled before disabling the MMU. This function should not be explicitly called and is invoked from the library. This function is provided for debug purposes only.

**Return value**

None

See also

[CyU3PSysDisableCacheMMU](#)  
[CyU3PSysEnableMMU](#)

#### 5.26.3.13 void CyU3PSysEnableCacheMMU ( void )

Enable the Caches and the MMU.

**Description**

Function to enable the Instruction and Data caches as well as the MMU. Sets up both caches to use random replacement strategy. This function should not be explicitly called and is invoked from the library during initialization. It is provided for debug purposes.

**Return value**

None

See also

[CyU3PSysDisableCacheMMU](#)  
[CyU3PSysEnableICache](#)  
[CyU3PSysEnableDCache](#)  
[CyU3PSysEnableMMU](#)

#### 5.26.3.14 void CyU3PSysEnableDCache ( void )

Enable the Data Cache.

**Description**

Function to enable the Data Cache. The MMU must also be enabled for the D-Cache to function properly. This function should not be explicitly called. The CyU3PDeviceCacheControl API should be called instead.

**Return value**

None

See also

[CyU3PSysDisableDCache](#)  
[CyU3PSysEnableCacheMMU](#)

**5.26.3.15 void CyU3PSysEnableCache( void )**

Enable the instruction cache.

**Description**

Function to enable the instruction cache. The function also sets up both caches to use random replacement strategy. This function should not be explicitly called. The CyU3PDeviceCacheControl API should be called instead. The function is available for debug purposes.

**Return value**

None

**See also**

[CyU3PSysDisableCache](#)  
[CyU3PSysEnableCacheMMU](#)

**5.26.3.16 void CyU3PSysEnableMMU( void )**

Enable the MMU.

**Description**

Function to enable the MMU. The page tables should be setup before calling this function. This function should not be explicitly called and is invoked from the library during initialization. It is provided for debug purposes.

**Return value**

None

**See also**

[CyU3PSysDisableMMU](#)  
[CyU3PSysEnableCacheMMU](#)

**5.26.3.17 void CyU3PSysFlushCaches( void )**

Flush both I and D Caches.

**Description**

Function to flush both the Instruction and Data Caches. The caller is responsible for ensuring that the Data cache is clean.

**Return value**

None

**5.26.3.18 void CyU3PSysFlushDCache( void )**

Flush the D-Cache.

**Description**

Function to flush the Data Cache. The caller should ensure that the cache has been cleaned before calling this function.

**Return value**

None

**5.26.3.19 void CyU3PSysFlushDRegion( uint32\_t \*addr, uint32\_t len )**

Flush a data region from the D-Cache.

**Description**

Function to flush a specified data region from the Data Cache. If the D-cache is enabled and the cache is handled by the application, then this API should be invoked before receiving any data using DMA.

**Return value**

None

**Parameters**

<i>addr</i>	Start address of the region to clean.
<i>len</i>	Length of the region in bytes.

5.26.3.20 void CyU3PSysFlushICache ( void )

Flush the I-Cache.

**Description**

Function to flush the Instruction Cache.

**Return value**

None

5.26.3.21 void CyU3PSysFlushIRegion ( uint32\_t \* *addr*, uint32\_t *len* )

Flush a code region from the I-Cache.

**Description**

Function to remove a specified code region from the Instruction Cache.

**Return value**

None

**Parameters**

<i>addr</i>	Start address of the region to flush.
<i>len</i>	Length of the region in bytes.

5.26.3.22 void CyU3PSysFlushTLBEntry ( uint32\_t \* *addr* )

Invalidate the TLB entry corresponding to a specified MVA.

**Description**

This function invalidates one of the TLB entries corresponding to the specified Modified Virtual Address. Even locked down entries will be invalidated. Multiple calls of this function may be needed if multi-level page tables are in use. This function is should not be called explicitly and is provided for debug purposes.

**Return value**

None

**Parameters**

<i>addr</i>	Address whose TLB entry is to be invalidated.
-------------	---

**5.26.3.23 void CyU3PSysInitTCMs ( void )**

Function to initialize the TCM regions.

**Description**

This function enables the TCMs by writing to the TCM region registers. This function should not be explicitly invoked. The library calls this function during initialization. It is provided for debug purposes.

**Return value**

None

**5.26.3.24 void CyU3PSysLoadTLB ( void )**

Function to lock all valid page walks into TLB.

**Description**

This function locks the page table walk information for all valid addresses on the FX3 device into the TLB. This function should be called after CyU3PInitPageTable has been called. This function is should not be called explicitly and is provided for debug purposes.

**Return value**

None

**5.26.3.25 void CyU3PSysLockTLBEntry ( uint32\_t \* addr )**

Function to lock the TLB entry for a page walk.

**Description**

This function loads the page table walk information corresponding to a specified address into the TLB and locks it. This function is should not be called explicitly and is provided for debug purposes.

**Return value**

None

**Parameters**

<code>addr</code>	Address range whose TLB entry is to be locked in place.
-------------------	---

## 5.27 firmware/u3p\_firmware/inc/cyu3os.h File Reference

This file defines the RTOS services and wrappers that are provided as part of the FX3 firmware solution. Most of these services are used by the drivers in the FX3 library and need to be provided when porting to a new OS environment.

```
#include "cyu3types.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

### Data Structures

- struct [MemBlockInfo](#)

*Structure representing a block of memory that has been dynamically allocated and is in use.*

## Typedefs

- **typedef struct CyU3PBytePool CyU3PBytePool**  
*Byte pool structure.*
- **typedef struct CyU3PBlockPool CyU3PBlockPool**  
*Block pool structure.*
- **typedef struct CyU3PThread CyU3PThread**  
*Thread structure.*
- **typedef void(\* CyU3PThreadEntry\_t) (uint32\_t threadArg)**  
*Thread entry function type.*
- **typedef struct CyU3PQueue CyU3PQueue**  
*Message queue structure.*
- **typedef struct CyU3PMutex CyU3PMutex**  
*Mutex structure.*
- **typedef struct CyU3PSemaphore CyU3PSemaphore**  
*Semaphore structure.*
- **typedef struct CyU3PEvent CyU3PEvent**  
*Event group structure.*
- **typedef struct CyU3PTimer CyU3PTimer**  
*Timer structure.*
- **typedef void(\* CyU3PTimerCb\_t) (uint32\_t timerArg)**  
*Type of callback function called on timer expiration.*
- **typedef struct MemBlockInfo MemBlockInfo**  
*Structure representing a block of memory that has been dynamically allocated and is in use.*
- **typedef void(\* CyU3PMemCorruptCallback) (void \*mem\_p)**  
*Type of callback function used for notifying user about memory corruption.*

## Functions

- **void CyU3PUndefinedHandler (void)**  
*The undefined instruction exception handler.*
- **void CyU3PPrefetchHandler (void)**  
*The pre-fetch error exception handler.*
- **void CyU3PAbortHandler (void)**  
*The abort error exception handler.*
- **void CyU3PKernelEntry (void)**  
*RTOS kernel entry function.*
- **void CyU3PApplicationDefine (void)**  
*The driver specific OS primitives and threads shall be created in this function.*
- **void CyU3POsTimerInit (uint16\_t intervalMs)**  
*Initialize the OS scheduler timer.*
- **void CyU3POsTimerHandler (void)**  
*The OS scheduler.*
- **void CyU3PIrqContextSave (void)**  
*This function saves the active thread context when entering the IRQ context.*
- **void CyU3PIrqVectoredContextSave (void)**  
*This function saves the active thread context when entering the IRQ context.*
- **void CyU3PIrqContextRestore (void)**  
*This function restores the thread context after handling an interrupt.*
- **void CyU3PFIqContextSave (void)**  
*This function saves the active thread context when entering the FIQ context.*

- void **CyU3PFIqContextRestore** (void)  
*This function restores the thread context after handling an FIQ interrupt.*
- void **CyU3PIrqNestingStart** (void)  
*This function switches the ARM mode from IRQ to SYS to allow nesting of interrupts.*
- void **CyU3PIrqNestingStop** (void)  
*This function switches the ARM mode from SYS to IRQ at the end of a interrupt handler.*
- void **CyU3PMemInit** (void)  
*Initialize the custom heap manager for OS specific allocation calls.*
- void \* **CyU3PMemAlloc** (uint32\_t size)  
*Allocate memory from the dynamic memory pool.*
- void **CyU3PMemFree** (void \*mem\_p)  
*Free memory allocated from the dynamic memory pool.*
- void **CyU3PFreeHeaps** (void)  
*Free up the custom heap manager and the buffer allocator.*
- void **CyU3PDmaBufferInit** (void)  
*Initialize the DMA buffer allocator.*
- void **CyU3PDmaBufferDelinit** (void)  
*De-initialize the buffer allocator.*
- void \* **CyU3PDmaBufferAlloc** (uint16\_t size)  
*Allocate the required amount of buffer memory.*
- int **CyU3PDmaBufferFree** (void \*buffer)  
*Free the previously allocated buffer area.*
- CyU3PReturnStatus\_t **CyU3PMemEnableChecks** (CyBool\_t enable, CyU3PMemCorruptCallback cb)  
*Control memory leak and corruption checks in the CyU3PMemAlloc memory manager.*
- CyU3PReturnStatus\_t **CyU3PBufEnableChecks** (CyBool\_t enable, CyU3PMemCorruptCallback cb)  
*Control memory leak and corruption checks in the CyU3PDmaBufferAlloc memory manager.*
- void **CyU3PMemGetCounts** (uint32\_t \*allocCnt\_p, uint32\_t \*freeCnt\_p)  
*Get memory allocation statistics.*
- MemBlockInfo \* **CyU3PMemGetActiveList** (void)  
*Get list of all unfreed allocations using CyU3PMemAlloc.*
- void **CyU3PBufGetCounts** (uint32\_t \*allocCnt\_p, uint32\_t \*freeCnt\_p)  
*Get memory allocation statistics.*
- MemBlockInfo \* **CyU3PBufGetActiveList** (void)  
*Get list of all unfreed allocations using CyU3PDmaBufferAlloc.*
- CyU3PReturnStatus\_t **CyU3PMemCorruptionCheck** (void)  
*Check for memory corruption around all buffers allocated using CyU3PMemAlloc.*
- CyU3PReturnStatus\_t **CyU3PBufCorruptionCheck** (void)  
*Check for memory corruption around all buffers allocated using CyU3PDmaBufferAlloc.*
- void **CyU3PMemSet** (uint8\_t \*ptr, uint8\_t data, uint32\_t count)  
*Fill a region of memory with a specified value.*
- void **CyU3PMemCopy** (uint8\_t \*dest, uint8\_t \*src, uint32\_t count)  
*Copy data from one memory location to another.*
- int32\_t **CyU3PMemCmp** (const void \*s1, const void \*s2, uint32\_t n)  
*Compare the contents of two memory buffers.*
- uint32\_t **CyU3PBytePoolCreate** (CyU3PBytePool \*pool\_p, void \*poolStart, uint32\_t poolSize)  
*This function creates and initializes a memory byte pool.*
- uint32\_t **CyU3PBytePoolDestroy** (CyU3PBytePool \*pool\_p)  
*De-initialize and free up a memory byte pool.*
- uint32\_t **CyU3PByteAlloc** (CyU3PBytePool \*pool\_p, void \*\*mem\_p, uint32\_t memSize, uint32\_t waitOption)  
*Allocate memory from a byte pool.*
- uint32\_t **CyU3PByteFree** (void \*mem\_p)

- `uint32_t CyU3PBlockPoolCreate (CyU3PBlockPool *pool_p, uint32_t blockSize, void *poolStart, uint32_t poolSize)`

*Frees memory allocated by the CyU3PByteAlloc function.*
- `uint32_t CyU3PBlockPoolDestroy (CyU3PBlockPool *pool_p)`

*Creates and initializes a memory block pool.*
- `uint32_t CyU3PBlockAlloc (CyU3PBlockPool *pool_p, void **block_p, uint32_t waitOption)`

*De-initialize a block memory pool.*
- `uint32_t CyU3PBlockFree (void *block_p)`

*Allocate a memory buffer from a block pool.*
- `uint32_t CyU3PThreadCreate (CyU3PThread *thread_p, char *threadName, CyU3PThreadEntry_t entryFn, uint32_t entryInput, void *stackStart, uint32_t stackSize, uint32_t priority, uint32_t preemptThreshold, uint32_t timeSlice, uint32_t autoStart)`

*Frees a memory buffer allocated from a block pool.*
- `uint32_t CyU3PThreadDestroy (CyU3PThread *thread_ptr)`

*This function creates a new thread.*
- `uint32_t CyU3PThreadDestroy (CyU3PThread *thread_ptr)`

*Free up and remove a thread from the RTOS scheduler.*
- `CyU3PThread * CyU3PThreadIdentify (void)`

*Get the thread structure corresponding to the current thread.*
- `uint32_t CyU3PThreadCreateInfoGet (CyU3PThread *thread_p, uint8_t **name_p, uint32_t *priority, uint32_t *preemptionThreshold, uint32_t *timeSlice)`

*Retrieve information regarding a specified thread.*
- `uint32_t CyU3PThreadPriorityChange (CyU3PThread *thread_p, uint32_t newPriority, uint32_t *oldPriority)`

*Change the priority of a thread.*
- `void CyU3PThreadRelinquish (void)`

*Relinquish control to the OS scheduler.*
- `uint32_t CyU3PThreadSleep (uint32_t timerTicks)`

*Puts a thread to sleep for the specified timer ticks.*
- `uint32_t CyU3PThreadSuspend (CyU3PThread *thread_p)`

*Suspends the specified thread.*
- `uint32_t CyU3PThreadResume (CyU3PThread *thread_p)`

*Resume operation of a thread that was previously suspended.*
- `uint32_t CyU3PThreadWaitAbort (CyU3PThread *thread_p)`

*Returns a thread to ready state by aborting all waits on the thread.*
- `uint32_t CyU3PQueueCreate (CyU3PQueue *queue_p, uint32_t messageSize, void *queueStart, uint32_t queueSize)`

*Create a message queue.*
- `uint32_t CyU3PQueueDestroy (CyU3PQueue *queue_p)`

*Free up a previously created message queue.*
- `uint32_t CyU3PQueueSend (CyU3PQueue *queue_p, void *src_p, uint32_t waitOption)`

*Queue a new message on the specified message queue.*
- `uint32_t CyU3PQueuePrioritySend (CyU3PQueue *queue_p, void *src_p, uint32_t waitOption)`

*Add a new message at the head of a message queue.*
- `uint32_t CyU3PQueueReceive (CyU3PQueue *queue_p, void *dest_p, uint32_t waitOption)`

*Receive a message from a message queue.*
- `uint32_t CyU3PQueueFlush (CyU3PQueue *queue_p)`

*Flushes all messages on a queue.*
- `uint32_t CyU3PMutexCreate (CyU3PMutex *mutex_p, uint32_t priorityInherit)`

*Create a mutex variable.*
- `uint32_t CyU3PMutexDestroy (CyU3PMutex *mutex_p)`

*Destroy a mutex variable.*
- `uint32_t CyU3PMutexGet (CyU3PMutex *mutex_p, uint32_t waitOption)`

- `uint32_t CyU3PMutexPut (CyU3PMutex *mutex_p)`

*Get the lock on a mutex variable.*
- `uint32_t CyU3PMutexCreate (CyU3PMutex *mutex_p, uint32_t initialCount)`

*Create a mutex object.*
- `uint32_t CyU3PSemaphoreDestroy (CyU3PSemaphore *semaphore_p)`

*Destroy a semaphore object.*
- `uint32_t CyU3PSemaphoreGet (CyU3PSemaphore *semaphore_p, uint32_t waitOption)`

*Get an instance from the specified counting semaphore.*
- `uint32_t CyU3PSemaphorePut (CyU3PSemaphore *semaphore_p)`

*Release an instance of the specified counting semaphore.*
- `uint32_t CyU3PEventCreate (CyU3PEvent *event_p)`

*Create an event flag group.*
- `uint32_t CyU3PEventDestroy (CyU3PEvent *event_p)`

*Destroy an event flag group.*
- `uint32_t CyU3PEventSet (CyU3PEvent *event_p, uint32_t rqtFlag, uint32_t setOption)`

*Update one or more flags in an event group.*
- `uint32_t CyU3PEventGet (CyU3PEvent *event_p, uint32_t rqtFlag, uint32_t getOption, uint32_t *flag_p, uint32_t waitOption)`

*Wait for or get the status of an event flag group.*
- `uint32_t CyU3PTimerCreate (CyU3PTimer *timer_p, CyU3PTimerCb_t expirationFunction, uint32_t expirationInput, uint32_t initialTicks, uint32_t rescheduleTicks, uint32_t timerOption)`

*Create an application timer.*
- `uint32_t CyU3PTimerDestroy (CyU3PTimer *timer_p)`

*Destroy an application timer object.*
- `uint32_t CyU3PTimerStart (CyU3PTimer *timer_p)`

*Start an application timer.*
- `uint32_t CyU3PTimerStop (CyU3PTimer *timer_p)`

*Stop operation of an application timer.*
- `uint32_t CyU3PTimerModify (CyU3PTimer *timer_p, uint32_t initialTicks, uint32_t rescheduleTicks)`

*Modify parameters of an application timer.*
- `uint32_t CyU3PGetTime (void)`

*Get the number of elapsed OS timer ticks since FX3 system start-up.*
- `void CyU3PSetTime (uint32_t newTime)`

*Update the timer tick count.*
- `uint32_t CyU3PThreadSystemPerfGet (uint32_t *resumeCnt_p, uint32_t *suspendCnd_p, uint32_t *reqPreemptionCnt_p, uint32_t *intrPreemptionCnt_p, uint32_t *prioInvertCnt_p, uint32_t *relinquishCnt_p, uint32_t *timeoutCnt_p, uint32_t *busyReturnCnt_p, uint32_t *idleReturnCnt_p)`

*Get OS performance profile information.*
- `uint32_t CyU3PThreadPerfGet (CyU3PThread *thread_p, uint32_t *resumeCnt_p, uint32_t *suspendCnd_p, uint32_t *reqPreemptionCnt_p, uint32_t *intrPreemptionCnt_p, uint32_t *prioInvertCnt_p, uint32_t *relinquishCnt_p, uint32_t *timeoutCnt_p)`

*Get thread specific performance profile information.*
- `uint32_t CyU3PTimerSystemPerfGet (uint32_t *activateCnt_p, uint32_t *reactivateCnt_p, uint32_t *deactivateCnt_p, uint32_t *expiryCnt_p)`

*Get global OS timer performance information.*
- `uint32_t CyU3PTimerPerfGet (CyU3PTimer *timer_p, uint32_t *activateCnt_p, uint32_t *reactivateCnt_p, uint32_t *deactivateCnt_p, uint32_t *expiryCnt_p)`

*Get timer specific performance profile information.*
- `uint32_t CyU3PMutexSystemPerfGet (uint32_t *putCnt_p, uint32_t *getCnt_p, uint32_t *suspCnt_p, uint32_t *timeoutCnt_p, uint32_t *invertCnt_p, uint32_t *inheritCnt_p)`

*Get global mutex performance profile information.*

- `uint32_t CyU3PMutexPerfGet (CyU3PMutex *mutex_p, uint32_t *putCnt_p, uint32_t *getCnt_p, uint32_t *suspCnt_p, uint32_t *timeoutCnt_p, uint32_t *invertCnt_p, uint32_t *inheritCnt_p)`  
*Get mutex specific performance profile information.*
- `uint32_t CyU3PEventSystemPerfGet (uint32_t *setCnt_p, uint32_t *getCnt_p, uint32_t *suspCnt_p, uint32_t *timeoutCnt_p)`  
*Get global event flag performance profile information.*
- `uint32_t CyU3PEventPerfGet (CyU3PEvent *event_p, uint32_t *setCnt_p, uint32_t *getCnt_p, uint32_t *suspCnt_p, uint32_t *timeoutCnt_p)`  
*Get event flag specific performance profile information.*
- `uint32_t CyU3PQueueSystemPerfGet (uint32_t *sentCnt_p, uint32_t *recvCnt_p, uint32_t *emptyCnt_p, uint32_t *fullCnt_p, uint32_t *fullErrCnt_p, uint32_t *timeoutCnt_p)`  
*Get global message queue performance profile information.*
- `uint32_t CyU3PQueuePerfGet (CyU3PQueue *queue_p, uint32_t *sentCnt_p, uint32_t *recvCnt_p, uint32_t *emptyCnt_p, uint32_t *fullCnt_p, uint32_t *fullErrCnt_p, uint32_t *timeoutCnt_p)`  
*Get message queue specific performance profile information.*
- `uint32_t CyU3PThreadSetActivityGpio (CyU3PThread *thread_p, uint32_t gpio_id)`  
*Register a GPIO to be updated when a thread is made active or inactive.*
- `uint32_t CyU3PMutexSetActivityGpio (CyU3PMutex *mutex_p, uint32_t gpio_id)`  
*Register a GPIO to be updated based on the status of a Mutex variable.*
- `uint32_t CyU3PSemaphoreSetActivityGpio (CyU3PSemaphore *semaphore_p, uint32_t gpio_id)`  
*Register a GPIO to be updated based on the status of a Semaphore variable.*
- `uint32_t CyU3PEventSetActivityGpio (CyU3PEvent *event_p, uint32_t gpio_id)`  
*Register a GPIO to be updated based on the status of a event flags group.*

### 5.27.1 Detailed Description

This file defines the RTOS services and wrappers that are provided as part of the FX3 firmware solution. Most of these services are used by the drivers in the FX3 library and need to be provided when porting to a new OS environment.

### 5.27.2 RTOS Constants

The RTOS wrappers used by the FX3 firmware library define and make use of the following constants.

#### Description

The following constants are special parameter values passed to some OS functions to specify the desired behaviour.

**CYU3P\_NO\_WAIT** : This is a special value for waitOption which indicates that the function should not wait / block the thread and should immediately return. This is used for OS functions like CyU3PMutexGet and also for various firmware API functions which internally wait for mutexes or events.

**CYU3P\_WAIT\_FOREVER** : This is a special value for waitOption which indicates that the function should wait until the particular mutex or event has been flagged. This is used for OS functions like CyU3PMutexGet and also for various firmware API functions which internally wait for mutexes and events.

**CYU3P\_EVENT\_AND** : This is a special value for getopt / setopt in the CyU3PEventGet and CyU3PEventSet functions which specifies the bit-wise operation to be performed on the event flag. This variable is used when the mask and the flag have to be ANDed without modification to the actual flags in the case of CyU3PEventGet.

**CYU3P\_EVENT\_AND\_CLEAR** : This is a special value for getopt in the CyU3PEventGet function which specifies the bit operation to be performed on the event flag. This option is used when the mask and the flag have to be ANDed and the flag cleared.

**CYU3P\_EVENT\_OR** : This is a special value for the getopt / setopt in CyU3PEventGet and CyU3PEventSet functions which specifies the bit operation to be performed on the event flag. This option is used when the mask and the flag have to be ORed without modification to the flags.

**CYU3P\_EVENT\_OR\_CLEAR** : This is a special value for the getOption in CyU3PEventGet function which specifies the bit operation to be performed on the event flag. This option is used when the mask and the flag have to be ORed and the flag cleared.

**CYU3P\_NO\_TIME\_SLICE** : This is a special value for the timeSlice option in CyU3PThreadCreate function. The value specifies that the thread shall not be pre-empted based on the timeSlice value provided.

**CYU3P\_AUTO\_START** : This is a special value for the autoStart option in CyU3PThreadCreate function. The value specifies that the thread should be started immediately without waiting for a CyU3PThreadResume call.

**CYU3P\_DONT\_START** : This is a special value for the autoStart option in CyU3PThreadCreate function. The value specifies that the thread should be suspended on create and shall be started only after a subsequent CyU3PThreadResume call.

**CYU3P\_AUTO\_ACTIVATE** : This is a special value for the timerOption parameter in CyU3PTimerCreate function. This value specifies that the timer shall be automatically started after create.

**CYU3P\_NO\_ACTIVATE** : This is a special value for the timerOption parameter in CyU3PTimerCreate function. This value specifies that the timer shall not be activated on create and needs to be explicitly activated.

**CYU3P\_INHERIT** : This is a special value for the priorityInherit parameter of the CyU3PMutexCreate function. This value specifies that the mutex supports priority inheritance.

**CYU3P\_NO\_INHERIT** : This is a special value for the priorityInherit parameter of the CyU3PMutexCreate function. This value specifies that the mutex does not support priority inheritance.

### 5.27.3 Exceptions and Interrupts

This section describes the operating modes, exceptions and interrupts in the FX3 firmware.

#### Description

The FX3 firmware generally executes in the Supervisor (SVC) mode of ARM processors, which is a privileged mode. The User and FIQ modes are currently unused by the FX3 firmware. The IRQ mode is used for the initial part of interrupt handler execution and the System mode is used for handling the second halves of long interrupts in a nested manner.

The Abort and Undefined modes are only executed when the ARM CPU encounters an execution error such as an undefined instruction, or a instruction/data access abort.

The FX3 device has an internal PL192 Vectored Interrupt Controller which is used for managing all interrupts raised by the device. The drivers for various hardware blocks in the FX3 firmware library register interrupt handlers for all interrupt sources. Event callbacks are raised to the user firmware for all relevant interrupt conditions.

Each interrupt source has a 4 bit priority value associated with it. These priority settings are unused as of now; and interrupt priority is enforced through nesting and pre-emption.

The RTOS used by the FX3 firmware allows interrupts to be nested, and this mechanism is used to allow higher priority interrupts to pre-empt lower priority ones. Thus the interrupts are classified into two groups: high priority ones that cannot be pre-empted and low priority ones that can be pre-empted.

The high priority (non pre-emptable interrupts) are:

USB interrupt

DMA interrupt for USB sockets

DMA interrupt for GPIF sockets

DMA interrupt for Serial peripheral sockets

The low priority (pre-emptable interrupts) are: System control interrupt (used for suspend/wakeup)

OS scheduler interrupt (timer based)

GPIF interrupt

SPI interrupt

I2C interrupt

I2S interrupt

UART interrupt

GPIO interrupt

The respective interrupt handlers in the drivers are responsible for enabling/disabling these interrupt sources at the

appropriate time. Since disabling one or more of these interrupts at arbitrary times can cause system errors and crashes; user accessible functions to control these interrupts individually are not provided.

The FX3 SDK only provides APIs that can disable and re-enable all interrupt sources so as to ensure atomicity of critical code sections.

#### 5.27.3.1 Exception Handler Functions

This section provides information on ARM Exception handlers in the FX3 firmware.

##### Description

Exceptions such as data or instruction abort, and undefined instruction may happen if the firmware image is corrupted during loading or at runtime. Since these are unexpected conditions, the FX3 firmware library does not provide any specific code to handle them. Default handlers for these conditions are provided in the cyfxtx.c file, and these can be modified by the users to match their requirements (for example, reset the FX3 device and restart operation).

#### 5.27.4 RTOS Memory Functions

This section documents the functions that the FX3 firmware provides for dynamic memory management. These are implemented as wrappers on top of the corresponding RTOS services.

##### Description

The FX3 firmware makes use of a set of memory management services that are provided by the RTOS used. The firmware library also provides a set of high level dynamic memory management functions that are wrappers on top of the corresponding RTOS services.

Two flavors of memory allocation services are provided:

Byte Pool: This is a generic memory pool similar to a fixed sized heap. Memory buffers of any size can be allocated from a byte pool.

Block pool: This is a memory pool that is optimized for handling buffers of a fixed size only. The block pool has lesser memory overhead per buffer allocated and can be used in cases where the application requires a large number of buffers of a specific size.

The firmware library or application is not expected to use the compiler provided heap for memory allocation to avoid portability issues across different compilers and tool chains. The library provides general purpose memory allocation functions that can be used to replace the standard C library memory allocation calls.

These functions can be implemented using the above mentioned byte pool and block pool functions. Implementations of these functions in source form are provided for reference, and can be modified as required by the application.

#### 5.27.5 Thread Functions

This section documents the thread functions that are provided as part of the FX3 firmware.

##### Description

The FX3 firmware provides a set of thread functions that can be used by the application to create and manage threads. These functions are wrappers around the corresponding RTOS services.

The firmware framework itself consists of a number of threads that run the drivers for various peripheral blocks on the device. It is expected that these driver threads will have higher priorities than any threads created by the firmware application.

Co-operative scheduling is typically used in the firmware, and time slices are not used. This means that thread switches will only happen when the active thread is blocked.

#### 5.27.6 Message Queue Functions

This section documents the message queue functions that are provided as part of the FX3 firmware.

**Description**

The FX3 firmware makes use of message queues for inter-thread data communication. A set of wrapper functions on top of the corresponding OS services are provided to allow the use of message queues from application threads.

### 5.27.7 Mutex functions

This section documents the mutex functions that are provided as part of the FX3 firmware.

**Description**

The FX3 firmware provides a set of mutex functions that can be used for protection of global data structures in a multi-thread environment. These functions are all wrappers around the corresponding OS services.

### 5.27.8 Semaphore Functions

This section documents the semaphore functions supported by the FX3 firmware.

**Description**

In addition to mutexes used for ownership control of shared data and mutual exclusion, the FX3 firmware also provides counting semaphores that can be used for synchronization and signaling. Each semaphore is created with an initial count, and the count is decremented on each successful get operation. The get operation is failed when the count reaches zero. Each put operation on the semaphore increments the associated count.

### 5.27.9 Event Flag Functions

This section documents the event flag functions provided by the FX3 firmware.

**Description**

Event flags are an advanced means for inter-thread synchronization that is provided as part of the FX3 firmware. These functions are wrappers around the corresponding functionality provided by the RTOS.

Event flag groups are groups of single bit flags or signals that can be used for inter-thread communication. Each flag in a event group can be individually signaled or waited upon by any given thread. It is possible for multiple threads to wait on different flags that are implemented by a single event group. This facility makes event flags a memory efficient means for inter-thread synchronization.

Event flags are frequently used for synchronization between various driver threads in the FX3 device.

### 5.27.10 Timer Functions

This section documents the application timer functions that are provided by the FX3 firmware.

**Description**

Application timers are OS services that support the implementation of periodic tasks in the firmware application. Any number of application timers (subject to memory constraints) can be created by the application and the time intervals specified should be multiples of the OS timer ticks.

The OS keeps track of the registered application timers and calls the application provided callback functions every time the corresponding interval has elapsed.

### 5.27.11 Typedef Documentation

#### 5.27.11.1 `typedef struct CyU3PBlockPool CyU3PBlockPool`

Block pool structure.

**Description**

A block pool is a form of heap (memory allocator) which allocated memory blocks of a fixed size selected during pool creation. This structure is more memory efficient than the more generic byte pool.

Though the block pool definition and services are part of the API library, the FX3 driver code does not make internal references to these services.

See also

[CyU3PBlockPoolCreate](#)  
[CyU3PBlockPoolDestroy](#)  
[CyU3PBlockAlloc](#)  
[CyU3PBlockFree](#)

#### 5.27.11.2 `typedef struct CyU3PBytePool CyU3PBytePool`

Byte pool structure.

**Description**

A byte pool is a form of heap that supports allocation of arbitrarily sized memory blocks from a memory region specified at creation time. The total memory used by the byte pool is fixed at creation time, and allocation will fail once all of the available memory is used up.

See also

[CyU3PBytePoolCreate](#)  
[CyU3PBytePoolDestroy](#)  
[CyU3PByteAlloc](#)  
[CyU3PByteFree](#)

#### 5.27.11.3 `typedef struct CyU3PEvent CyU3PEvent`

Event group structure.

**Description**

An event group is a set of event flags that an OS thread can wait for, and other threads or interrupts can notify. Each event group provides a maximum of 32 separate event flags that can be used independently. This structure is associated with each event group used in the FX3 firmware application.

See also

[CyU3PEventCreate](#)  
[CyU3PEventDestroy](#)  
[CyU3PEventSet](#)  
[CyU3PEventGet](#)

#### 5.27.11.4 `typedef void(* CyU3PMemCorruptCallback) (void *mem_p )`

Type of callback function used for notifying user about memory corruption.

**Description**

This type defines a callback function that will be used to notify the user when the allocator detects memory corruption around the boundaries of a memory block allocated through the `CyU3PMemAlloc` or `CyU3PDmaBufferAlloc` calls.

See also

[CyU3PMemEnableChecks](#)  
[CyU3PBufEnableChecks](#)  
[CyU3PMemCorruptionCheck](#)  
[CyU3PBufCorruptionCheck](#)

### 5.27.11.5 `typedef struct CyU3PMutex CyU3PMutex`

Mutex structure.

#### Description

This structure is associated with each Mutex object that is used in the FX3 firmware application.

#### See also

[CyU3PMutexCreate](#)  
[CyU3PMutexDestroy](#)  
[CyU3PMutexPut](#)  
[CyU3PMutexGet](#)

### 5.27.11.6 `typedef struct CyU3PQueue CyU3PQueue`

Message queue structure.

#### Description

Message queues are used for inter-thread communication in FX3 firmware. They are also used by the interrupt handlers for various FX3 blocks to schedule work for the driver threads to handle. This structure is associated with each message queue that is used in the firmware application.

#### See also

[CyU3PQueueCreate](#)  
[CyU3PQueueDestroy](#)  
[CyU3PQueueSend](#)  
[CyU3PQueuePrioritySend](#)  
[CyU3PQueueReceive](#)  
[CyU3PQueueFlush](#)

### 5.27.11.7 `typedef struct CyU3PSemaphore CyU3PSemaphore`

Semaphore structure.

#### Description

The semaphore service in the FX3 OS Library is a typical counting semaphore implementation. This structure is associated with each semaphore object that is used in the firmware application.

#### See also

[CyU3PSemaphoreCreate](#)  
[CyU3PSemaphoreDestroy](#)  
[CyU3PSemaphoreGet](#)  
[CyU3PSemaphorePut](#)

### 5.27.11.8 `typedef struct CyU3PThread CyU3PThread`

Thread structure.

#### Description

This data structure is associated with each RTOS thread that is created in the FX3 firmware. The driver modules for various FX3 blocks make use of some threads internally as well.

**See also**

[CyU3PThreadCreate](#)  
[CyU3PThreadDestroy](#)  
[CyU3PThreadIdentify](#)  
[CyU3PThreadInfoGet](#)  
[CyU3PThreadSleep](#)  
[CyU3PThreadSuspend](#)  
[CyU3PThreadResume](#)  
[CyU3PThreadWaitAbort](#)  
[CyU3PThreadRelinquish](#)  
[CyU3PThreadPriorityChange](#)  
[CyU3PThreadStackErrorNotify](#)

**5.27.11.9 `typedef void(* CyU3PThreadEntry_t)(uint32_t threadArg )`**

Thread entry function type.

**Description**

This type represents the entry function for an RTOS thread created by a FX3 firmware application. The `threadArg` argument is a context input that is specified by the user when creating the thread.

**See also**

[CyU3PThread](#)  
[CyU3PThreadCreate](#)

**5.27.11.10 `typedef struct CyU3PTimer CyU3PTimer`**

Timer structure.

**Description**

OS timers provide a mechanism for setting up periodic tasks which will be triggered once every period number of timer ticks. This structure is associated with each OS timer object used in the FX3 firmware application.

**See also**

[CyU3PTimerCreate](#)  
[CyU3PTimerDestroy](#)  
[CyU3PTimerStart](#)  
[CyU3PTimerStop](#)  
[CyU3PTimerModify](#)

**5.27.11.11 `typedef void(* CyU3PTimerCb_t)(uint32_t timerArg )`**

Type of callback function called on timer expiration.

**Description**

This type defines the signature of the handler callback for an OS timer object. A function of this type is associated with each timer at the time of creation and is called by the OS scheduler on a periodic basis. The `timerArg` parameter passed to this function is also specified at the time of creating the timer object.

**See also**

[CyU3PTimer](#)  
[CyU3PTimerCreate](#)

### 5.27.11.12 `typedef struct MemBlockInfo MemBlockInfo`

Structure representing a block of memory that has been dynamically allocated and is in use.

#### Description

This structure represents the header associated with a memory block allocated through the CyU3PMemAlloc and CyU3PDmaBufferAlloc functions. This header structure is used to check for memory leak and corruption occurrences.

## 5.27.12 Function Documentation

### 5.27.12.1 `void CyU3PAbortHandler ( void )`

The abort error exception handler.

#### Description

This function gets invoked when the ARM CPU encounters an data pre-fetch abort error. As virtual memory is not used in the system, this can only happen if data is being read from a non-existent memory location.

#### Return value

None

#### See also

[CyU3PUndefinedHandler](#)  
[CyU3PPrefetchHandler](#)

### 5.27.12.2 `void CyU3PApplicationDefine ( void )`

The driver specific OS primitives and threads shall be created in this function.

#### Description

This function shall be implemented by the firmware library and needs to be invoked from the kernel during initialization. This is where all the threads and OS primitives for all module drivers shall be created. Though some OS calls can be safely made at this point; scheduling, wait options and sleep functions are not expected to be functional at this point.

#### Return value

None

#### See also

[CyU3PKernelEntry](#)

### 5.27.12.3 `uint32_t CyU3PBlockAlloc ( CyU3PBlockPool * pool_p, void ** block_p, uint32_t waitOption )`

Allocate a memory buffer from a block pool.

#### Description

This function allocates a memory buffer from a block pool. The size of the buffer is fixed during the pool creation. The waitOption parameter specifies the timeout duration before the alloc call is failed.

#### Return value

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

#### See also

[CyU3PBlockPoolCreate](#)  
[CyU3PBlockFree](#)

**Parameters**

<i>pool_p</i>	Handle to the memory block pool.
<i>block_p</i>	Output variable that will be filled with a pointer to the allocated buffer.
<i>waitOption</i>	Timeout duration in timer ticks.

5.27.12.4 `uint32_t CyU3PBlockFree ( void * block_p )`

Frees a memory buffer allocated from a block pool.

**Description**

This function frees a memory buffer allocated through the CyU3PBlockAlloc call.

**Return value**

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

**See also**

[CyU3PBlockPoolCreate](#)  
[CyU3PBlockAlloc](#)

**Parameters**

<i>block_p</i>	Pointer to buffer to be freed.
----------------	--------------------------------

5.27.12.5 `uint32_t CyU3PBlockPoolCreate ( CyU3PBlockPool * pool_p, uint32_t blockSize, void * poolStart, uint32_t poolSize )`

Creates and initializes a memory block pool.

**Description**

This function initializes a memory block pool from which buffers of a fixed size can be dynamically allocated. The size of the buffer is specified as a parameter to this pool create function.

**Return value**

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

**See also**

[CyU3PBlockPoolDestroy](#)  
[CyU3PBlockAlloc](#)  
[CyU3PBlockFree](#)

**Parameters**

<i>pool_p</i>	Handle to block pool structure. The caller needs to allocate the structure, and this function only initializes the fields of the structure.
<i>blockSize</i>	Size of memory blocks that this pool will manage. The block size needs to be a multiple of 16 bytes.
<i>poolStart</i>	Pointer to memory region provided for the block pool.
<i>poolSize</i>	Size of memory region provided for the block pool.

### 5.27.12.6 uint32\_t CyU3PBlockPoolDestroy ( CyU3PBlockPool \* *pool\_p* )

De-initialize a block memory pool.

#### Description

This function de-initializes a memory block pool. The memory region used by the block pool can be re-used after this function returns.

#### Return value

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

#### See also

[CyU3PBlockPoolCreate](#)

#### Parameters

<i>pool_p</i>	Handle to the block pool to be destroyed.
---------------	---

### 5.27.12.7 CyU3PReturnStatus\_t CyU3PBufCorruptionCheck ( void )

Check for memory corruption around all buffers allocated using CyU3PDmaBufferAlloc.

#### Description

Check for memory corruption around all buffers allocated using CyU3PDmaBufferAlloc. The callback registered using the [CyU3PBufEnableChecks\(\)](#) API will be called for each instance of memory corruption detected during the checks.

#### Return value

CY\_U3P\_ERROR\_FAILURE if any memory corruption is found. CY\_U3P\_SUCCESS if no memory corruption is found.

### 5.27.12.8 CyU3PReturnStatus\_t CyU3PBufEnableChecks ( CyBool\_t *enable*, CyU3PMemCorruptCallback *cb* )

Control memory leak and corruption checks in the CyU3PDmaBufferAlloc memory manager.

#### Description

This function enables/disables memory leak and corruption checks in the CyU3PDmaBufferAlloc memory manager. This has to be done before the [CyU3PDmaBufferInit\(\)](#) function is called (in the main function before [CyU3PKernelEntry\(\)](#) is called).

#### Return value

CY\_U3P\_SUCCESS if the checks were correctly enabled/disabled. CY\_U3P\_ERROR\_ALREADY\_STARTED if the CyU3PDmaBufferInit function has already been called.

#### See also

[CyU3PMemCorruptCallback](#)  
[CyU3PBufGetActiveList](#)  
[CyU3PBufCorruptionCheck](#)

#### Parameters

<i>enable</i>	Enable leak and corruption checks.
<i>cb</i>	Callback to be used for memory corruption notification.

### 5.27.12.9 MemBlockInfo\* CyU3PBufGetActiveList( void )

Get list of all unfreed allocations using CyU3PDmaBufferAlloc.

#### Description

Get list of all unfreed allocations that have been made using CyU3PDmaBufferAlloc.

#### Return value

Start of the allocated memory list.

### 5.27.12.10 void CyU3PBufGetCounts( uint32\_t \* allocCnt\_p, uint32\_t \* freeCnt\_p )

Get memory allocation statistics.

#### Description

This function returns the total memory allocation and free counts using the CyU3PDmaBufferAlloc and CyU3P $\leftarrow$ DmaBufferFree calls.

#### Return value

None

#### Parameters

<i>allocCnt<math>\leftarrow</math>_p</i>	Return pointer to be filled with allocation count.
<i>freeCnt<math>\leftarrow</math>_p</i>	Return pointer to be filled with free count.

### 5.27.12.11 uint32\_t CyU3PByteAlloc( CyU3PBytePool \* pool\_p, void \*\* mem\_p, uint32\_t memSize, uint32\_t waitOption )

Allocate memory from a byte pool.

#### Description

This function is used to allocate memory buffers from a previously created memory byte pool. The waitOption specifies the timeout duration after which the memory allocation should be failed. It is permitted to use this function from an interrupt context as long as the waitOption is set to zero or CYU3P\_NO\_WAIT.

#### Return value

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

#### See also

[CyU3PBytePoolCreate](#)  
[CyU3PByteFree](#)

#### Parameters

<i>pool_p</i>	Handle to the byte pool to allocate memory from.
<i>mem_p</i>	Output variable that points to the allocated memory buffer.
<i>memSize</i>	Size of memory buffer required in bytes.
<i>waitOption</i>	Timeout for memory allocation operation in terms of OS timer ticks. Can be set to CYU3P_NO_WAIT or CYU3P_WAIT_FOREVER.

### 5.27.12.12 uint32\_t CyU3PByteFree ( void \* *mem\_p* )

Frees memory allocated by the CyU3PByteAlloc function.

#### Description

This function frees memory allocated from a byte pool using the CyU3PByteAlloc function. This function can also be invoked from an interrupt context.

#### Return value

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

#### See also

[CyU3PBytePoolCreate](#)  
[CyU3PByteAlloc](#)

#### Parameters

<i>mem_p</i>	Pointer to memory buffer to be freed
--------------	--------------------------------------

### 5.27.12.13 uint32\_t CyU3PBytePoolCreate ( CyU3PBytePool \* *pool\_p*, void \* *poolStart*, uint32\_t *poolSize* )

This function creates and initializes a memory byte pool.

#### Description

This function is a wrapper around the byte pool service provided by the RTOS. It creates a fixed size general purpose heap structure that can be used for dynamic memory allocation.

#### Return value

CY\_U3P\_SUCCESS if pool was successfully initialized.

Other RTOS specific error codes in case of failure.

#### See also

[CyU3PBytePoolDestroy](#)  
[CyU3PByteAlloc](#)  
[CyU3PBlockFree](#)

#### Parameters

<i>pool_p</i>	Handle to the byte pool structure. The memory for the structure is to be allocated by the caller. This function only initializes the structure with the pool information.
<i>poolStart</i>	Pointer to memory region provided for the byte pool. This address needs to be 16 byte aligned.
<i>poolSize</i>	Size of the memory region provided for the byte pool. This size needs to be a multiple of 16 bytes.

### 5.27.12.14 uint32\_t CyU3PBytePoolDestroy ( CyU3PBytePool \* *pool\_p* )

De-initialize and free up a memory byte pool.

#### Description

This function cleans up the previously created byte pool pointed to by the *pool\_p* structure. Memory allocated for the pool can be re-used after this call returns.

**Return value**

CY\_U3P\_SUCCESS on success.  
Other RTOS error codes on failure.

**See also**

[CyU3PBytePoolCreate](#)

**Parameters**

<i>pool</i> <i>p</i>	Handle to the byte pool to be freed up
-------------------------	--

5.27.12.15 void\* CyU3PDmaBufferAlloc ( uint16\_t size )

Allocate the required amount of buffer memory.

**Description**

This function allocates buffers of the required size from the memory region reserved through the CyU3PDmaBufferInit function. It is called by the DMA APIs when creating DMA channels, and can also be used directly by the user application code.

This implementation is expected to ensure that the buffers allocated are aligned to data cache lines.

**Return value**

Pointer to the allocated buffer, or NULL in case of error.

**See also**

[CyU3PDmaBufferInit](#)  
[CyU3PDmaBufferDelInit](#)  
[CyU3PDmaBufferFree](#)

**Parameters**

<i>size</i>	The size in bytes of the buffer required.
-------------	---

5.27.12.16 void CyU3PDmaBufferDelInit ( void )

De-initialize the buffer allocator.

**Description**

This function de-initializes the DMA buffer manager. The function shall be invoked on DMA module de-init. This is provided in source format so that it can be modified as per user requirement. This function must not be called directly by the user application.

**Return value**

None

**See also**

[CyU3PDmaBufferInit](#)

5.27.12.17 int CyU3PDmaBufferFree ( void \* *buffer* )

Free the previously allocated buffer area.

**Description**

Frees up a buffer previously allocated through the CyU3PDmaBufferAlloc function. This function is used internally by the DMA API when freeing up DMA channels, and can also be used directly by the user application code.

**Return value**

0 if the buffer is freed successfully.  
Non-zero value otherwise.

**See also**

[CyU3PDmaBufferInit](#)  
[CyU3PDmaBufferDelInit](#)  
[CyU3PDmaBufferAlloc](#)

**Parameters**

<i>buffer</i>	Address of buffer to be freed.
---------------	--------------------------------

**5.27.12.18 void CyU3PDmaBufferInit( void )**

Initialize the DMA buffer allocator.

**Description**

This function creates an allocator that provides the DMA buffers required for data transfers through the FX3 device. The DMA buffers are also allocated from a pre-assigned memory region within the FX3 RAM.

This allocator needs to ensure that all buffers managed by it are aligned to data cache lines. i.e., That all buffers start at addresses that are multiples of 32 bytes and span a multiple of 32 bytes.

This function is called internally by the FX3 firmware library and should not be called directly by the user application.

**Return value**

None

**See also**

[CyU3PDmaBufferDelInit](#)  
[CyU3PDmaBufferAlloc](#)  
[CyU3PDmaBufferFree](#)

**5.27.12.19 uint32\_t CyU3PEventCreate( CyU3PEvent \* event\_p )**

Create an event flag group.

**Description**

This function creates an event flag group consisting of 32 independent event flags. The application is free to use as many flags as required from this group. If more than 32 flags are required, multiple event flag groups have to be created.

As with other OS service create functions, the caller is expected to allocate memory for the Event data structure.

**Return value**

CY\_U3P\_SUCCESS (0) on success.  
Other error codes on failure.

**See also**

[CyU3PEventDestroy](#)  
[CyU3PEventSet](#)  
[CyU3PEventGet](#)

**Parameters**

<i>event_p</i>	Pointer to event structure to be initialized.
----------------	---

5.27.12.20 uint32\_t CyU3PEventDestroy ( CyU3PEvent \* *event\_p* )

Destroy an event flag group.

**Description**

This function frees up an event flag group. Any threads waiting on one or more flags in the group will be re-activated and will receive an error code from the event wait function.

**Return value**

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

**See also**

[CyU3PEventCreate](#)

**Parameters**

<i>event_p</i>	Pointer to event group to be destroyed.
----------------	---

5.27.12.21 uint32\_t CyU3PEventGet ( CyU3PEvent \* *event\_p*, uint32\_t *rqtFlag*, uint32\_t *getOption*, uint32\_t \* *flag\_p*, uint32\_t *waitOption* )

Wait for or get the status of an event flag group.

**Description**

This function is used to the get the current status of the flags in a event group. This can also be used to wait until one or more flags of interest have been signaled. The maximum amount of time to wait is specified through the waitOption parameter.

**Return value**

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

**See also**

[CyU3PEventSet](#)

**Parameters**

<i>event_p</i>	Pointer to event group to be queried.
<i>rqtFlag</i>	Bit-mask that selects event flags of interest.
<i>getOption</i>	Type of operation to perform: CYU3P_EVENT_AND: Use to wait until all selected flags are signaled. CYU3P_EVENT_AND_CLEAR: Same as above, and clear the flags on read. CYU3P_EVENT_OR: Wait until any of selected flags are signaled. CYU3P_EVENT_OR_CLEAR: Same as above, and clear the flags on read.
<i>flag_p</i>	Output parameter filled in with the state of the flags.
<i>waitOption</i>	Timeout duration in timer ticks.

---

```
5.27.12.22 uint32_t CyU3PEventPerfGet ( CyU3PEvent * event_p, uint32_t * setCnt_p, uint32_t * getCnt_p, uint32_t * suspCnt_p, uint32_t * timeoutCnt_p )
```

Get event flag specific performance profile information.

#### Description

The ThreadX operating system supports collection of system performance data on all OS objects during firmware execution. This function retrieves the performance statistics associated with a specific event flag group object.

#### Return Value

CY\_U3P\_SUCCESS if the data has been retrieved.

CY\_U3P\_ERROR\_BAD\_POINTER if the event flag group pointer is not valid.

CY\_U3P\_ERROR\_OPERN\_DISABLED if the performance collection is not enabled in the library.

#### Parameters

<i>event_p</i>	Pointer to the event flag group to be queried.
<i>setCnt_p</i>	Destination pointer to be filled with number of event set operations.
<i>getCnt_p</i>	Destination pointer to be filled with number of event get operations.
<i>suspCnt_p</i>	Destination pointer to be filled with number of thread suspensions due to event get operations.
<i>timeoutCnt_p</i>	Destination pointer to be filled with number of event get timeouts.

---

```
5.27.12.23 uint32_t CyU3PEventSet ( CyU3PEvent * event_p, uint32_t rqtFlag, uint32_t setOption )
```

Update one or more flags in an event group.

#### Description

This function is used to set or clear one or more flags that are part of a specified event group. The parameters can be selected so as to set a number of specified flags, or to clear of number of specified flags.

#### Return value

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

#### See also

[CyU3PEventGet](#)

#### Parameters

<i>event_p</i>	Pointer to event group to update.
<i>rqtFlag</i>	Bit-mask that selects event flags of interest.
<i>setOption</i>	Type of set operation to perform: CYU3P_EVENT_AND: The rqtFlag will be ANDed with the current flags. CYU3P_EVENT_OR: The rqtFlag will be ORed with the current flags.

---

```
5.27.12.24 uint32_t CyU3PEventSetActivityGpio ( CyU3PEvent * event_p, uint32_t gpio_id )
```

Register a GPIO to be updated based on the status of a event flags group.

#### Description

This function registers an FX3 GPIO (simple GPIO) that will be updated by the RTOS services when making changes to the state of an event flags group. The GPIO will be driven high when at least one of the events in the group is active, and will be driven low when all of the events are inactive.

This function is only available in a profiling enabled build of the FX3 libraries. The caller is responsible for ensuring

that the GPIO block is active and that the corresponding GPIO is enabled.

#### Return Value

CY\_U3P\_SUCCESS if the data has been retrieved.  
 CY\_U3P\_ERROR\_INVALID\_CALLER if the GPIO block has not been enabled.  
 CY\_U3P\_ERROR\_BAD\_OPTION if the GPIO specified is invalid.  
 CY\_U3P\_ERROR\_OPERN\_DISABLED if this function is not supported (regular FX3 build).

#### Parameters

<i>event_p</i>	Pointer to the semaphore object to be monitored.
<i>gpio_id</i>	GPIO to be used for this semaphore.

5.27.12.25 uint32\_t CyU3PEventSystemPerfGet ( uint32\_t \* *setCnt\_p*, uint32\_t \* *getCnt\_p*, uint32\_t \* *suspCnt\_p*, uint32\_t \* *timeoutCnt\_p* )

Get global event flag performance profile information.

#### Description

The ThreadX operating system supports collection of system performance data on all OS objects during firmware execution. This function retrieves the performance statistics associated with all event flag group objects in the system.

#### Return Value

CY\_U3P\_SUCCESS if the data has been retrieved.  
 CY\_U3P\_ERROR\_OPERN\_DISABLED if the performance collection is not enabled in the library.

#### Parameters

<i>setCnt_p</i>	Destination pointer to be filled with number of event set operations.
<i>getCnt_p</i>	Destination pointer to be filled with number of event get operations.
<i>suspCnt_p</i>	Destination pointer to be filled with number of thread suspensions due to event get operations.
<i>timeoutCnt_p</i>	Destination pointer to be filled with number of event get timeouts.

5.27.12.26 void CyU3PFiqContextRestore ( void )

This function restores the thread context after handling an FIQ interrupt.

#### Description

This function is provided as part of the OS and restores thread state saved at the beginning of the FIQ handler using the CyU3PFiqContextSave API.

#### Return value

None

#### See also

[CyU3PFiqContextSave](#)

5.27.12.27 void CyU3PFiqContextSave ( void )

This function saves the active thread context when entering the FIQ context.

**Description**

This function is provided as part of the OS and saves thread state when entering a Fast Interrupt (FIQ) handler routine. As the FX3 firmware library currently does not use a FIQ, this function is not used.

**Return value**

None

**See also**

[CyU3PFIqContextRestore](#)

**5.27.12.28 void CyU3PFreeHeaps ( void )**

Free up the custom heap manager and the buffer allocator.

**Description**

This function is called in preparation to a reset of the FX3 device and is intended to allow the user to free up the custom heap manager and the buffer allocator that are created in the CyU3PMemInit and the CyU3PDmaBufferInit functions respectively.

**Return value**

None

**See also**

[CyU3PMemInit](#)

[CyU3PDmaBufferInit](#)

**5.27.12.29 uint32\_t CyU3PGetTime ( void )**

Get the number of elapsed OS timer ticks since FX3 system start-up.

**Description**

The function returns the number of elapsed OS timer ticks since the FX3 OS was started up.

**Return value**

Current OS timer tick count.

**See also**

[CyU3PSetTime](#)

**5.27.12.30 void CyU3PIrqContextRestore ( void )**

This function restores the thread context after handling an interrupt.

**Description**

This function is provided as part of the OS and allows the previously backed up thread state to be restored at the end of a IRQ handler call. The same function is used to restore thread state saved by the CyU3PIrqContextSave and CyU3PIrqVectoredContextSave functions.

**Return value**

None

**See also**

[CyU3PIrqContextSave](#)

[CyU3PIrqVectoredContextSave](#)

**5.27.12.31 void CyU3PIrqContextSave ( void )**

This function saves the active thread context when entering the IRQ context.

**Description**

This function is provided as part of the OS to allow any thread state to be backed up on entry to a IRQ handler routine. This function is only applicable when the VIC is not being used, and is generally not used in FX3 firmware.

See the CyU3PIrqVectoredContextSave function for the equivalent function used in FX3 firmware.

**Return value**

None

**See also**

[CyU3PIrqContextRestore](#)  
[CyU3PIrqVectoredContextSave](#)  
[CyU3PFIqContextSave](#)

**5.27.12.32 void CyU3PIrqNestingStart ( void )**

This function switches the ARM mode from IRQ to SYS to allow nesting of interrupts.

**Description**

Nesting of interrupts on an ARMv5 controller requires that the current interrupt handler switch the ARM execution mode to SYS mode. This function is used to do this switching. In case interrupt nesting is not required for this platform, this function can be a No-Operation.

**Return value**

None

**See also**

[CyU3PIrqNestingStop](#)

**5.27.12.33 void CyU3PIrqNestingStop ( void )**

This function switches the ARM mode from SYS to IRQ at the end of a interrupt handler.

**Description**

Nesting of interrupts on an ARMv5 controller requires that the ARM execution mode be switched to SYS mode at the beginning of the handler and back to IRQ mode at the end. This function is used to do the switch the mode back to IRQ at the end of an interrupt handler. In case interrupt nesting is not required for this platform, this function can be a No-operation.

**Return value**

None

**See also**

[CyU3PIrqNestingStart](#)

**5.27.12.34 void CyU3PIrqVectoredContextSave ( void )**

This function saves the active thread context when entering the IRQ context.

**Description**

This function is provided as part of the OS to allow any thread state to be backed up on entry to a IRQ handler routine. This function is applicable when the device in use has a Vectored Interrupt Controller (VIC) and is used by all interrupt handlers in the FX3 firmware.

**Return value**

None

**See also**

[CyU3PIrqContextRestore](#)  
[CyU3PIrqContextSave](#)  
[CyU3PFIqContextSave](#)

**5.27.12.35 void CyU3PKernelEntry ( void )**

RTOS kernel entry function.

**Description**

The function call is expected to initialize the RTOS. This function needs to be invoked as the last function from the main () function. It is expected that the firmware shall always function in the SVC mode.

**Return value**

None - The function does not return at all.

**See also**

[CyU3PAplicationDefine](#)

**5.27.12.36 void\* CyU3PMemAlloc ( uint32\_t size )**

Allocate memory from the dynamic memory pool.

**Description**

This function is the malloc equivalent for allocating memory from the pool created by the CyU3PMemInit function. This function needs to be implemented as part of the RTOS porting to the FX3 device. This function needs to be capable of allocating memory even if called from an interrupt handler.

**Return value**

Pointer to the allocated buffer, or NULL in case of failure.

**See also**

[CyU3PMemInit](#)  
[CyU3PMemFree](#)

**Parameters**

<i>size</i>	The size in bytes of the buffer to be allocated.
-------------	--

**5.27.12.37 int32\_t CyU3PMemCmp ( const void \* s1, const void \* s2, uint32\_t n )**

Compare the contents of two memory buffers.

**Description**

This is a memcmp equivalent function that does a byte by byte comparison of two memory buffers.

**Return value**

0 if the two buffers hold the same data.

Negative if the data in s1 has a lower ASCII value than that in s2.

Positive if the data in s1 has a higher ASCII value than that in s2.

**Parameters**

<i>s1</i>	Pointer to first memory buffer.
<i>s2</i>	Pointer to second memory buffer.
<i>n</i>	Size in bytes of the buffers to compare.

5.27.12.38 void CyU3PMemCopy ( uint8\_t \* *dest*, uint8\_t \* *src*, uint32\_t *count* )

Copy data from one memory location to another.

**Description**

This is a memcpy equivalent function that is provided and used by the firmware library. The implementation does not handle the case of overlapping buffers.

**Return value**

None

**Parameters**

<i>dest</i>	Pointer to destination buffer.
<i>src</i>	Pointer to source buffer.
<i>count</i>	Size of the buffer to be copied.

5.27.12.39 CyU3PReturnStatus\_t CyU3PMemCorruptionCheck ( void )

Check for memory corruption around all buffers allocated using CyU3PMemAlloc.

**Description**

Check for memory corruption around all buffers allocated using CyU3PMemAlloc. The callback registered using the [CyU3PMemEnableChecks\(\)](#) API will be called for each instance of memory corruption detected during the checks.

**Return value**

CY\_U3P\_ERROR\_FAILURE if any memory corruption is found. CY\_U3P\_SUCCESS if no memory corruption is found.

5.27.12.40 CyU3PReturnStatus\_t CyU3PMemEnableChecks ( CyBool\_t *enable*, CyU3PMemCorruptCallback *cb* )

Control memory leak and corruption checks in the CyU3PMemAlloc memory manager.

**Description**

This function enables/disables memory leak and corruption checks in the CyU3PMemAlloc memory manager. This has to be done before the [CyU3PMemInit\(\)](#) function is called (in the main function before [CyU3PKernelEntry\(\)](#) is called).

**Return value**

CY\_U3P\_SUCCESS if the checks were correctly enabled/disabled. CY\_U3P\_ERROR\_ALREADY\_STARTED if the CyU3PMemInit function has already been called.

**See also**

[CyU3PMemCorruptCallback](#)  
[CyU3PMemGetActiveList](#)  
[CyU3PMemCorruptionCheck](#)

**Parameters**

<i>enable</i>	Enable leak and corruption checks.
<i>cb</i>	Callback to be used for memory corruption notification.

5.27.12.41 void CyU3PMemFree ( void \* *mem\_p* )

Free memory allocated from the dynamic memory pool.

**Description**

This function is the free equivalent that frees memory allocated through the CyU3PMemAlloc function. This function is also expected to be able to free memory in an interrupt context.

**Return value**

None

**See also**

[CyU3PMemInit](#)  
[CyU3PMemAlloc](#)

**Parameters**

<i>mem_p</i>	Pointer to memory buffer to be freed.
--------------	---------------------------------------

5.27.12.42 MemBlockInfo\* CyU3PMemGetActiveList ( void )

Get list of all unfreed allocations using CyU3PMemAlloc.

**Description**

Get list of all unfreed allocations that have been made using CyU3PMemAlloc.

**Return value**

Start of the allocated memory list.

5.27.12.43 void CyU3PMemGetCounts ( uint32\_t \* *allocCnt\_p*, uint32\_t \* *freeCnt\_p* )

Get memory allocation statistics.

**Description**

This function returns the total memory allocation and free counts using the CyU3PMemAlloc and CyU3PMemFree calls.

**Return value**

None

**Parameters**

<i>allocCnt_p</i>	Return pointer to be filled with allocation count.
<i>freeCnt_p</i>	Return pointer to be filled with free count.

### 5.27.12.44 void CyU3PMemInit( void )

Initialize the custom heap manager for OS specific allocation calls.

#### Description

This function creates a memory pool that can be used in place of the system heap. All dynamic memory allocation for OS data structures, thread stacks, and firmware data buffers should be allocated out of this memory pool. The size and location of this memory pool needs to be adjusted as per the user requirements by modifying this function. This function is called internally by the FX3 library, and should not be called directly by the user code.

#### Return value

None

#### See also

[CyU3PMemAlloc](#)  
[CyU3PMemFree](#)  
[CyU3PFreeHeaps](#)

### 5.27.12.45 void CyU3PMemSet( uint8\_t \* ptr, uint8\_t data, uint32\_t count )

Fill a region of memory with a specified value.

#### Description

This function is a memset equivalent and is used by the firmware library code. It can also be used by the application firmware.

#### Return value

None

#### Parameters

<i>ptr</i>	Pointer to the buffer to be filled.
<i>data</i>	Value to fill the buffer with.
<i>count</i>	Size of the buffer in bytes.

### 5.27.12.46 uint32\_t CyU3PMutexCreate( CyU3PMutex \* mutex\_p, uint32\_t priorityInherit )

Create a mutex variable.

#### Description

This function creates a mutex variable. The mutex data structure has to be allocated by the caller, and will be initialized by the function.

#### Return value

CY\_U3P\_SUCCESS (0) on success.  
 Other error codes on failure.

#### See also

[CyU3PMutexDestroy](#)  
[CyU3PMutexGet](#)  
[CyU3PMutexPut](#)

#### Parameters

<i>mutex_p</i>	Pointer to Mutex data structure to be initialized.
<i>priorityInherit</i>	Whether priority inheritance should be allowed for this mutex. Allowed values for this parameter are CYU3P_NO_INHERIT and CYU3P_INHERIT.

### 5.27.12.47 uint32\_t CyU3PMutexDestroy ( CyU3PMutex \* mutex\_p )

Destroy a mutex variable.

#### Description

This function destroys a mutex that was created using the CyU3PMutexCreate API.

#### Return value

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

#### See also

[CyU3PMutexCreate](#)

#### Parameters

<i>mutex_p</i>	Pointer to mutex to be destroyed.
----------------	-----------------------------------

### 5.27.12.48 uint32\_t CyU3PMutexGet ( CyU3PMutex \* mutex\_p, uint32\_t waitOption )

Get the lock on a mutex variable.

#### Description

This function is used to wait on a mutex variable and to get a lock on it. The maximum amount of time to wait is specified through the waitOption parameter.

#### Return value

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

#### See also

[CyU3PMutexCreate](#)  
[CyU3PMutexDestroy](#)

#### Parameters

<i>mutex_p</i>	Pointer to mutex to be acquired.
<i>waitOption</i>	Timeout duration in timer ticks.

### 5.27.12.49 uint32\_t CyU3PMutexPerfGet ( CyU3PMutex \* mutex\_p, uint32\_t \* putCnt\_p, uint32\_t \* getCnt\_p, uint32\_t \* suspCnt\_p, uint32\_t \* timeoutCnt\_p, uint32\_t \* invertCnt\_p, uint32\_t \* inheritCnt\_p )

Get mutex specific performance profile information.

#### Description

The ThreadX operating system supports collection of system performance data on all OS objects during firmware execution. This function retrieves the performance statistics associated with a specific mutex object.

#### Return Value

CY\_U3P\_SUCCESS if the data has been retrieved.

CY\_U3P\_ERROR\_BAD\_POINTER if the mutex pointer is not valid.

CY\_U3P\_ERROR\_OPERN\_DISABLED if the performance collection is not enabled in the library.

**Parameters**

<i>mutex_p</i>	Pointer to mutex to be queried.
<i>putCnt_p</i>	Destination pointer to be filled with number of Mutex put operations.
<i>getCnt_p</i>	Destination pointer to be filled with number of Mutex get operations.
<i>suspCnt_p</i>	Destination pointer to be filled with number of mutex get suspensions.
<i>timeoutCnt_p</i>	Destination pointer to be filled with number of Mutex get timeouts.
<i>invertCnt_p</i>	Destination pointer to be filled with number of thread priority inversions on mutexes.
<i>inheritCnt_p</i>	Destination pointer to be filled with number of thread priority inheritances on mutexes.

**5.27.12.50 uint32\_t CyU3PMutexPut ( CyU3PMutex \* *mutex\_p* )**

Release the lock on a mutex variable.

**Description**

This function is used to release the lock on a mutex variable. The mutex can then be acquired by the highest priority thread from among those waiting for it.

**Return value**

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

**See also**

[CyU3PMutexGet](#)

**Parameters**

<i>mutex_p</i>	Pointer to mutex to be released.
----------------	----------------------------------

**5.27.12.51 uint32\_t CyU3PMutexSetActivityGpio ( CyU3PMutex \* *mutex\_p*, uint32\_t *gpio\_id* )**

Register a GPIO to be updated based on the status of a Mutex variable.

**Description**

This function registers an FX3 GPIO (simple GPIO) that will be updated by the RTOS services when making changes to the state of a mutex variable. The GPIO will be driven high when the mutex is free (available) and driven low when the mutex is not available.

This function is only available in a profiling enabled build of the FX3 libraries. The caller is responsible for ensuring that the GPIO block is active and that the corresponding GPIO is enabled.

**Return Value**

CY\_U3P\_SUCCESS if the data has been retrieved.

CY\_U3P\_ERROR\_INVALID\_CALLER if the GPIO block has not been enabled.

CY\_U3P\_ERROR\_BAD\_OPTION if the GPIO specified is invalid.

CY\_U3P\_ERROR\_OPERN\_DISABLED if this function is not supported (regular FX3 build).

**Parameters**

<i>mutex_p</i>	Pointer to the mutex object to be monitored.
<i>gpio_id</i>	GPIO to be used for this mutex.

---

5.27.12.52 uint32\_t CyU3PMutexSystemPerfGet ( uint32\_t \* *putCnt\_p*, uint32\_t \* *getCnt\_p*, uint32\_t \* *suspCnt\_p*, uint32\_t \* *timeoutCnt\_p*, uint32\_t \* *invertCnt\_p*, uint32\_t \* *inheritCnt\_p* )

Get global mutex performance profile information.

#### Description

The ThreadX operating system supports collection of system performance data on all OS objects during firmware execution. This function retrieves the performance statistics associated with all mutex objects in the system.

#### Return Value

CY\_U3P\_SUCCESS if the data has been retrieved.

CY\_U3P\_ERROR\_OPERN\_DISABLED if the performance collection is not enabled in the library.

#### Parameters

<i>putCnt_p</i>	Destination pointer to be filled with number of Mutex put operations.
<i>getCnt_p</i>	Destination pointer to be filled with number of Mutex get operations.
<i>suspCnt_p</i>	Destination pointer to be filled with number of mutex get suspensions.
<i>timeoutCnt_p</i>	Destination pointer to be filled with number of Mutex get timeouts.
<i>invertCnt_p</i>	Destination pointer to be filled with number of thread priority inversions on mutexes.
<i>inheritCnt_p</i>	Destination pointer to be filled with number of thread priority inheritances on mutexes.

---

5.27.12.53 void CyU3POsTimerHandler ( void )

The OS scheduler.

#### Description

This function is implemented by the RTOS kernel and is called from the OS timer interrupt. This function call notifies the kernel scheduler that another time tick has elapsed.

#### Return value

None

---

5.27.12.54 void CyU3POsTimerInit ( uint16\_t *intervalMs* )

Initialize the OS scheduler timer.

#### Description

This function is implemented by the firmware library and is invoked from the RTOS kernel during initialization.

The OS\_TIMER\_TICK shall be defined by this function and all wait durations are calculated based on this. By default, the OS timer tick is configured to be 1ms. It is recommended that this default tick duration be retained to guarantee fast and accurate system response.

If the timer tick needs to be modified, this API can be invoked only after the RTOS has been initialized. This function can support a timer tick range from 1ms to 1000ms. Any other value of the timer tick will be discarded and the timer tick will be set to 1ms.

The clock for this timer is derived from the 32KHz standby clock. The actual tick interval can have an error of upto +/- 4%. The time interval will be accurate only as long as the interrupts are running.

#### Return value

None

#### Parameters

<i>intervalMs</i>	OS Timer tick interval in millisecond.
-------------------	--

### 5.27.12.55 void CyU3PPrefetchHandler( void )

The pre-fetch error exception handler.

#### Description

This function gets invoked when the ARM CPU encounters an instruction pre-fetch error. Since virtual memory is not used in the system, this can only happen if the device is trying to fetch instructions from non-existent memory.

#### Return value

None

#### See also

[CyU3PUndefinedHandler](#)  
[CyU3PAbortHandler](#)

### 5.27.12.56 uint32\_t CyU3PQueueCreate ( CyU3PQueue \* queue\_p, uint32\_t messageSize, void \* queueStart, uint32\_t queueSize )

Create a message queue.

#### Description

Create a message queue that can hold a specified number of messages of a specified size. The memory for the queue should be allocated and passed in by the caller.

#### Return value

CY\_U3P\_SUCCESS (0) on success.  
 Other error codes on failure.

#### See also

[CyU3PQueueDestroy](#)  
[CyU3PQueueSend](#)  
[CyU3PQueuePrioritySend](#)  
[CyU3PQueueReceive](#)  
[CyU3PQueueFlush](#)

#### Parameters

<i>queue_p</i>	Pointer to the queue structure. This should be allocated by the caller and will be initialized by the queue create function.
<i>messageSize</i>	Size of each message in 4 byte words. Allowed values are from 1 to 16 (4 bytes to 64 bytes).
<i>queueStart</i>	Pointer to memory buffer to be used for message storage. Should be aligned to 4 bytes.
<i>queueSize</i>	Total size of the queue in bytes.

### 5.27.12.57 uint32\_t CyU3PQueueDestroy ( CyU3PQueue \* queue\_p )

Free up a previously created message queue.

#### Description

This function frees up a previously created message queue. Any function call that is waiting to send or receive messages on this queue will return with an error.

#### Return value

CY\_U3P\_SUCCESS (0) on success.  
 Other error codes on failure.

See also

[CyU3PQueueCreate](#)

#### Parameters

<i>queue_p</i>	Pointer to the queue to be destroyed.
----------------	---------------------------------------

5.27.12.58 uint32\_t CyU3PQueueFlush ( **CyU3PQueue \* queue\_p** )

Flushes all messages on a queue.

#### Description

The function removes all pending messages on the specified queue.

#### Return value

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

See also

[CyU3PQueueSend](#)  
[CyU3PQueuePrioritySend](#)  
[CyU3PQueueReceive](#)

#### Parameters

<i>queue_p</i>	Pointer to the queue to be flushed.
----------------	-------------------------------------

5.27.12.59 uint32\_t CyU3PQueuePerfGet ( **CyU3PQueue \* queue\_p, uint32\_t \* sentCnt\_p, uint32\_t \* recvCnt\_p, uint32\_t \* emptyCnt\_p, uint32\_t \* fullCnt\_p, uint32\_t \* fullErrCnt\_p, uint32\_t \* timeoutCnt\_p** )

Get message queue specific performance profile information.

#### Description

The ThreadX operating system supports collection of system performance data on all OS objects during firmware execution. This function retrieves the performance statistics associated with a specific message queue object.

#### Return Value

CY\_U3P\_SUCCESS if the data has been retrieved.

CY\_U3P\_ERROR\_BAD\_POINTER if the message queue pointer is not valid.

CY\_U3P\_ERROR\_OPERN\_DISABLED if the performance collection is not enabled in the library.

#### Parameters

<i>queue_p</i>	Pointer to the queue to be queried.
<i>sentCnt_p</i>	Destination pointer to be filled with number of messages sent.
<i>recvCnt_p</i>	Destination pointer to be filled with number of messages received.
<i>emptyCnt_p</i>	Destination pointer to be filled with number of queue receive suspensions due to queue emptiness.
<i>fullCnt_p</i>	Destination pointer to be filled with number of queue send suspensions due to queue fullness.
<i>fullErrCnt_p</i>	Destination pointer to be filled with number of queue send failures due to queue fullness.

**Parameters**

<i>timeoutCnt← _p</i>	Destination pointer to be filled with number of queue timeouts.
---------------------------	---

5.27.12.60 `uint32_t CyU3PQueuePrioritySend ( CyU3PQueue *queue_p, void *src_p, uint32_t waitOption )`

Add a new message at the head of a message queue.

**Description**

This function is used to send a high priority message to a message queue. This message will be placed at the head of the queue instead of at the tail. As in the case of the CyU3PQueueSend API, this waits for the specified duration or until a message slot is free in the queue.

**Return value**

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

**See also**

[CyU3PQueueSend](#)  
[CyU3PQueueReceive](#)  
[CyU3PQueueFlush](#)

**Parameters**

<i>queue_p</i>	Pointer to the message queue structure.
<i>src_p</i>	Pointer to message buffer. Should be aligned to 4 bytes.
<i>waitOption</i>	Timeout duration in timer ticks.

5.27.12.61 `uint32_t CyU3PQueueReceive ( CyU3PQueue *queue_p, void *dest_p, uint32_t waitOption )`

Receive a message from a message queue.

**Description**

This function receives the message at the head of the specified queue. If the queue is empty, it waits until the specified timeout period has elapsed, or until the queue is non-empty. If calling from an interrupt handler, the waitOption parameter needs to be set to CYU3P\_NO\_WAIT.

**Return value**

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

**See also**

[CyU3PQueueSend](#)  
[CyU3PQueuePrioritySend](#)  
[CyU3PQueueFlush](#)

**Parameters**

<i>queue_p</i>	Pointer to the message queue.
<i>dest_p</i>	Pointer to buffer where the message should be copied. Should be aligned to 4 bytes and large enough to hold the full message.
<i>waitOption</i>	Timeout duration in timer ticks.

5.27.12.62 `uint32_t CyU3PQueueSend ( CyU3PQueue * queue_p, void * src_p, uint32_t waitOption )`

Queue a new message on the specified message queue.

#### Description

This function adds a new message on to the specified message queue. If the queue is full, it waits for a message slot to be freed. The amount of time to wait is specified as a parameter. In case this function is called from an interrupt handler, the time-out should be specified as CYU3P\_NO\_WAIT.

#### Return value

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

#### See also

[CyU3PQueuePrioritySend](#)  
[CyU3PQueueReceive](#)  
[CyU3PQueueFlush](#)

#### Parameters

<code>queue_p</code>	Queue to add a new message to.
<code>src_p</code>	Pointer to buffer containing message. Should be aligned to 4 bytes.
<code>waitOption</code>	Timeout value to wait on the queue.

5.27.12.63 `uint32_t CyU3PQueueSystemPerfGet ( uint32_t * sentCnt_p, uint32_t * recvCnt_p, uint32_t * emptyCnt_p, uint32_t * fullCnt_p, uint32_t * fullErrCnt_p, uint32_t * timeoutCnt_p )`

Get global message queue performance profile information.

#### Description

The ThreadX operating system supports collection of system performance data on all OS objects during firmware execution. This function retrieves the performance statistics associated with all message queue objects in the system.

#### Return Value

CY\_U3P\_SUCCESS if the data has been retrieved.

CY\_U3P\_ERROR\_OPERN\_DISABLED if the performance collection is not enabled in the library.

#### Parameters

<code>sentCnt_p</code>	Destination pointer to be filled with number of messages sent.
<code>recvCnt_p</code>	Destination pointer to be filled with number of messages received.
<code>emptyCnt_p</code>	Destination pointer to be filled with number of queue receive suspensions due to queue emptiness.
<code>fullCnt_p</code>	Destination pointer to be filled with number of queue send suspensions due to queue fullness.
<code>fullErrCnt_p</code>	Destination pointer to be filled with number of queue send failures due to queue fullness.
<code>timeoutCnt_p</code>	Destination pointer to be filled with number of queue timeouts.

5.27.12.64 `uint32_t CyU3PSemaphoreCreate ( CyU3PSemaphore * semaphore_p, uint32_t initialCount )`

Create a semaphore object.

**Description**

This function creates a semaphore object with the specified initial count. The semaphore data structure has to be allocated by the caller and will be initialized by this function call.

**Return value**

CY\_U3P\_SUCCESS (0) on success.  
Other error codes on failure.

## See also

[CyU3PSemaphoreDestroy](#)  
[CyU3PSemaphoreGet](#)  
[CyU3PSemaphorePut](#)

## Parameters

<i>semaphore</i> <i>_p</i>	Pointer to semaphore to be initialized.
<i>initialCount</i>	Initial count to associate with semaphore.

5.27.12.65 uint32\_t CyU3PSemaphoreDestroy ( CyU3PSemaphore \* *semaphore\_p* )

Destroy a semaphore object.

**Description**

This function destroys a semaphore object. All threads waiting to get the semaphore will receive an error code identifying that the object has been removed.

**Return value**

CY\_U3P\_SUCCESS (0) on success.  
Other error codes on failure.

## See also

[CyU3PSemaphoreCreate](#)

## Parameters

<i>semaphore</i> <i>_p</i>	Pointer to semaphore to be destroyed.
-------------------------------	---------------------------------------

5.27.12.66 uint32\_t CyU3PSemaphoreGet ( CyU3PSemaphore \* *semaphore\_p*, uint32\_t *waitOption* )

Get an instance from the specified counting semaphore.

**Description**

This function is used to get an instance (i.e., decrement the count by one) from the specified counting semaphore. If the count is already zero, the function will wait until the count becomes non-zero. The maximum interval to wait for is specified using the *waitOption* parameter.

**Return value**

CY\_U3P\_SUCCESS (0) on success.  
Other error codes on failure.

See also

[CyU3PSemaphorePut](#)

Parameters

<i>semaphore_p</i>	Pointer to semaphore to get.
<i>waitOption</i>	Timeout duration in timer ticks.

### 5.27.12.67 uint32\_t CyU3PSemaphorePut ( CyU3PSemaphore \* *semaphore\_p* )

Release an instance of the specified counting semaphore.

**Description**

This function releases an instance (increments the count by one) of the specified counting semaphore. The semaphore put can be done from any thread and does not need to be done by the same thread as called the get function.

**Return value**

CY\_U3P\_SUCCESS (0) on success.  
Other error codes on failure.

See also

[CyU3PSemaphoreCreate](#)  
[CyU3PSemaphoreDestroy](#)  
[CyU3PSemaphoreGet](#)

Parameters

<i>semaphore_p</i>	Pointer to semaphore to put.
--------------------	------------------------------

### 5.27.12.68 uint32\_t CyU3PSemaphoreSetActivityGpio ( CyU3PSemaphore \* *semaphore\_p*, uint32\_t *gpio\_id* )

Register a GPIO to be updated based on the status of a Semaphore variable.

**Description**

This function registers an FX3 GPIO (simple GPIO) that will be updated by the RTOS services when making changes to the state of a semaphore variable. The GPIO will be driven high when the semaphore count is non-zero, and driven low when the semaphore count is zero.

This function is only available in a profiling enabled build of the FX3 libraries. The caller is responsible for ensuring that the GPIO block is active and that the corresponding GPIO is enabled.

**Return Value**

CY\_U3P\_SUCCESS if the data has been retrieved.  
CY\_U3P\_ERROR\_INVALID\_CALLER if the GPIO block has not been enabled.  
CY\_U3P\_ERROR\_BAD\_OPTION if the GPIO specified is invalid.  
CY\_U3P\_ERROR\_OPERN\_DISABLED if this function is not supported (regular FX3 build).

Parameters

<i>semaphore_p</i>	Pointer to the semaphore object to be monitored.
<i>gpio_id</i>	GPIO to be used for this semaphore.

### 5.27.12.69 void CyU3PSetTime ( uint32\_t newTime )

Update the timer tick count.

#### Description

This function modifies the timer tick count that is started at system reset. This function should ideally not be called after the application is started up, as it can affect the operation of timers and threads.

#### Return value

None

#### See also

[CyU3PGetTime](#)

#### Parameters

<i>newTime</i>	New timer tick value to set.
----------------	------------------------------

### 5.27.12.70 uint32\_t CyU3PThreadCreate ( CyU3PThread \* thread\_p, char \* threadName, CyU3PThreadEntry\_t entryFn, uint32\_t entryInput, void \* stackStart, uint32\_t stackSize, uint32\_t priority, uint32\_t preemptThreshold, uint32\_t timeSlice, uint32\_t autoStart )

This function creates a new thread.

#### Description

This function creates a new application thread with the specified parameters. This function call must be made only after the RTOS kernel has been started.

The application threads should take only priority values from 7 to 15. The higher priorities (0 - 6) are reserved for the driver threads used by the library.

#### Return value

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

#### See also

[CyU3PThreadDestroy](#)  
[CyU3PThreadIdentify](#)  
[CyU3PThreadInfoGet](#)  
[CyU3PThreadPriorityChange](#)  
[CyU3PThreadRelinquish](#)  
[CyU3PThreadSleep](#)  
[CyU3PThreadSuspend](#)  
[CyU3PThreadResume](#)  
[CyU3PThreadWaitAbort](#)  
[CyU3PThreadStackErrorNotify](#)

#### Parameters

<i>thread_p</i>	Pointer to the thread structure. Memory for this structure has to be allocated by the caller.
<i>threadName</i>	Name string to associate with the thread. All threads should be named with the "Thread_Number:Description" convention.
<i>entryFn</i>	Thread entry function.
<i>entryInput</i>	Parameter to be passed into the thread entry function.
<i>stackStart</i>	Start address of the thread stack.
<i>stackSize</i>	Size of the thread stack in bytes.

**Parameters**

<i>priority</i>	Priority to be assigned to this thread.
<i>preemptThreshold</i>	Threshold value for thread pre-emption. Only threads with higher priorities than this value will be allowed to pre-empt this thread.
<i>timeSlice</i>	Maximum execution time allowed for this thread in timer ticks. It is recommended that time slices be disabled by specifying CYU3P_NO_TIME_SLICE as the value.
<i>autoStart</i>	Whether this thread should be suspended or started immediately. Can be set to CYU3P_AUTO_START or CYU3P_DONT_START.

5.27.12.71 `uint32_t CyU3PThreadDestroy ( CyU3PThread * thread_ptr )`

Free up and remove a thread from the RTOS scheduler.

**Description**

This function removes a previously created thread from the scheduler, and frees up the resources associated with it.

**Return value**

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

**See also**

[CyU3PThreadCreate](#)

**Parameters**

<i>thread_ptr</i>	Pointer to the thread structure.
-------------------	----------------------------------

5.27.12.72 `CyU3PThread* CyU3PThreadIdentify ( void )`

Get the thread structure corresponding to the current thread.

**Description**

This function returns a pointer to the thread structure corresponding to the active thread, or NULL if called from interrupt context.

**Return value**

Pointer to the thread structure of the currently running thread.

**See also**

[CyU3PThreadCreate](#)  
[CyU3PThreadInfoGet](#)

5.27.12.73 `uint32_t CyU3PThreadInfoGet ( CyU3PThread * thread_p, uint8_t ** name_p, uint32_t * priority, uint32_t * preemptionThreshold, uint32_t * timeSlice )`

Retrieve information regarding a specified thread.

**Description**

This function is used to retrieve information about a thread whose pointer is provided. This function is used by the debug mechanism in the firmware library to get information about the source thread which is queuing log messages.

Valid (non-NULL) pointers need to be provided only for the output fields that need to be retrieved.

**Return value**

CY\_U3P\_SUCCESS (0) on success.  
Other error codes on failure.

**See also**

[CyU3PThreadCreate](#)  
[CyU3PThreadIdentify](#)

**Parameters**

<i>thread_p</i>	Pointer to thread to be queried.
<i>name_p</i>	Output variable to be filled with the thread's name string.
<i>priority</i>	Output variable to be filled with the thread priority.
<i>preemptionThreshold</i>	Output variable to be filled with the pre-emption threshold.
<i>timeSlice</i>	Output variable to be filled with the time slice value.

5.27.12.74 uint32\_t CyU3PThreadPerfGet ( CyU3PThread \* *thread\_p*, uint32\_t \* *resumeCnt\_p*, uint32\_t \* *suspendCnd\_p*, uint32\_t \* *reqPreemptionCnt\_p*, uint32\_t \* *intrPreemptionCnt\_p*, uint32\_t \* *prioInvertCnt\_p*, uint32\_t \* *relinquishCnt\_p*, uint32\_t \* *timeoutCnt\_p* )

Get thread specific performance profile information.

**Description**

The ThreadX operating system supports collection of system performance data during firmware execution. This API returns profiling information specific to one thread object.

**Return Value**

CY\_U3P\_SUCCESS if the data has been retrieved.  
CY\_U3P\_ERROR\_BAD\_POINTER if the thread pointer is not valid.  
CY\_U3P\_ERROR\_OPERN\_DISABLED if the performance collection is not enabled in the library.

**Parameters**

<i>thread_p</i>	Pointer to thread to be queried.
<i>resumeCnt_p</i>	Destination pointer to be filled with thread resume count.
<i>suspendCnd_p</i>	Destination pointer to be filled with thread suspend count.
<i>reqPreemptionCnt_p</i>	Destination pointer to be filled with count of thread pre-emptions due to an RTOS API call.
<i>intrPreemptionCnt_p</i>	Destination pointer to be filled with count of thread pre-emptions due to interrupts.
<i>prioInvertCnt_p</i>	Destination pointer to be filled with count of thread priority inversions.
<i>relinquishCnt_p</i>	Destination pointer to be filled with count of thread relinquish calls.
<i>timeoutCnt_p</i>	Destination pointer to be filled with count of thread suspension timeouts.

5.27.12.75 uint32\_t CyU3PThreadPriorityChange ( CyU3PThread \* *thread\_p*, uint32\_t *newPriority*, uint32\_t \* *oldPriority* )

Change the priority of a thread.

**Description**

This function can be used to change the priority of a specified thread. Though this is not expected to be used commonly, it can be used for temporary priority changes to prevent deadlocks.

**Return value**

CY\_U3P\_SUCCESS (0) on success.  
 Other error codes on failure.

**See also**

[CyU3PThreadCreate](#)  
[CyU3PThreadRelinquish](#)  
[CyU3PThreadSleep](#)

**Parameters**

<i>thread_p</i>	Pointer to thread to be modified.
<i>newPriority</i>	Priority value to assign to the thread.
<i>oldPriority</i>	Output variable that will hold the original priority.

**5.27.12.76 void CyU3PThreadRelinquish ( void )**

Relinquish control to the OS scheduler.

**Description**

This is a RTOS call for fair scheduling which relinquishes control to other ready threads that are at the same priority level. The thread that relinquishes control remains in ready state and can regain control if there are no other ready threads with the same priority level.

**Return value**

None

**See also**

[CyU3PThreadCreate](#)  
[CyU3PThreadSleep](#)  
[CyU3PThreadSuspend](#)

**5.27.12.77 uint32\_t CyU3PThreadResume ( CyU3PThread \* *thread\_p* )**

Resume operation of a thread that was previously suspended.

**Description**

This function is used to resume operation of a thread that was suspended using the CyU3PThreadSuspend call. Threads that are suspended because they are blocked on Mutexes or Events cannot be resumed using this call.

**Return value**

CY\_U3P\_SUCCESS (0) on success.  
 Other error codes on failure.

**See also**

[CyU3PThreadSuspend](#)

**Parameters**

<i>thread_p</i>	Pointer to thread to be resumed.
-----------------	----------------------------------

### 5.27.12.78 uint32\_t CyU3PThreadSetActivityGpio ( *CyU3PThread \* thread\_p, uint32\_t gpio\_id* )

Register a GPIO to be updated when a thread is made active or inactive.

#### Description

This function registers an FX3 GPIO (simple GPIO) that will be updated by the RTOS scheduler when it is making the specified thread active or inactive. The GPIO number cannot be 0, and has to be a valid GPIO number on the FX3 device. The GPIO will be driven high when the thread is made active and driven low when it is made inactive.

This function is only available in a profiling enabled build of the FX3 libraries. The caller is responsible for ensuring that the GPIO block is active and that the corresponding GPIO is enabled.

#### Return Value

`CY_U3P_SUCCESS` if the data has been retrieved.

`CY_U3P_ERROR_INVALID_CALLER` if the GPIO block has not been enabled.

`CY_U3P_ERROR_BAD_OPTION` if the GPIO specified is invalid.

`CY_U3P_ERROR_OPERN_DISABLED` if this function is not supported (regular FX3 build).

#### Parameters

<i>thread_p</i>	Pointer to the thread object to be monitored.
<i>gpio_id</i>	GPIO to be used for this thread.

### 5.27.12.79 uint32\_t CyU3PThreadSleep ( *uint32\_t timerTicks* )

Puts a thread to sleep for the specified timer ticks.

#### Description

This function puts the current thread to sleep for the specified number of timer ticks.

#### Return value

`CY_U3P_SUCCESS` (0) on success.

Other error codes on failure.

#### See also

[CyU3PThreadCreate](#)  
[CyU3PThreadRelinquish](#)  
[CyU3PThreadSuspend](#)

#### Parameters

<i>timerTicks</i>	Number of timer ticks to sleep.
-------------------	---------------------------------

### 5.27.12.80 uint32\_t CyU3PThreadSuspend ( *CyU3PThread \* thread\_p* )

Suspends the specified thread.

#### Description

This function is used to suspend a thread that is in the ready state, and can be called from any thread.

#### Return value

`CY_U3P_SUCCESS` (0) on success.

Other error codes on failure.

**See also**

[CyU3PThreadCreate](#)  
[CyU3PThreadRelinquish](#)  
[CyU3PThreadSleep](#)  
[CyU3PThreadResume](#)

**Parameters**

<i>thread</i> <sub>↔</sub> _p	Pointer to the thread to be suspended.
----------------------------------	--

5.27.12.81 uint32\_t CyU3PThreadSystemPerfGet ( uint32\_t \* *resumeCnt\_p*, uint32\_t \* *suspendCnd\_p*, uint32\_t \* *reqPreemptionCnt\_p*, uint32\_t \* *intrPreemptionCnt\_p*, uint32\_t \* *prioInvertCnt\_p*, uint32\_t \*

## See also

[CyU3PThreadCreate](#)  
[CyU3PThreadSleep](#)  
[CyU3PThreadSuspend](#)  
[CyU3PThreadResume](#)

## Parameters

<i>thread</i> — <i>p</i>	Pointer to the thread to restore.
--------------------------------	-----------------------------------

5.27.12.83 uint32\_t CyU3PTimerCreate ( *CyU3PTimer \* timer\_p*, *CyU3PTimerCb\_t expirationFunction*, *uint32\_t expirationInput*, *uint32\_t initialTicks*, *uint32\_t rescheduleTicks*, *uint32\_t timerOption* )

Create an application timer.

**Description**

This function creates an application timer than can be configured as a one-shot timer or as a auto-reload timer.

**Return value**

CY\_U3P\_SUCCESS (0) on success.  
 Other error codes on failure.

## See also

[CyU3PTimerCb\\_t](#)  
[CyU3PTimerDestroy](#)  
[CyU3PTimerStart](#)  
[CyU3PTimerStop](#)  
[CyU3PTimerModify](#)  
[CyU3PGetTime](#)  
[CyU3PSetTime](#)

## Parameters

<i>timer_p</i>	Pointer to the timer structure to be initialized.
<i>expirationFunction</i>	Pointer to callback function called on timer expiration.
<i>expirationInput</i>	Parameter to be passed to the callback function.
<i>initialTicks</i>	Initial value for the timer. This timer count will be decremented once per timer tick and the callback will be invoked once the count reaches zero.
<i>rescheduleTicks</i>	The reload value for the timer. If set to zero, the timer will be a one-shot timer.
<i>timerOption</i>	Timer start control: CYU3P_AUTO_ACTIVATE: Start the timer immediately. CYU3P_NO_ACTIVATE: Timer needs to be started explicitly.

5.27.12.84 uint32\_t CyU3PTimerDestroy ( *CyU3PTimer \* timer\_p* )

Destroy an application timer object.

**Description**

This function destroys and application timer object.

**Return value**

CY\_U3P\_SUCCESS (0) on success.  
 Other error codes on failure.

See also

[CyU3PTimerCreate](#)

#### Parameters

<i>timer_p</i>	Pointer to the timer to be destroyed.
----------------	---------------------------------------

5.27.12.85 uint32\_t CyU3PTimerModify ( CyU3PTimer \* *timer\_p*, uint32\_t *initialTicks*, uint32\_t *rescheduleTicks* )

Modify parameters of an application timer.

#### Description

This function is used to modify the periodicity of an application timer and can be called only when the timer is stopped.

#### Return value

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

See also

[CyU3PTimerCreate](#)

[CyU3PTimerStart](#)

[CyU3PTimerStop](#)

#### Parameters

<i>timer_p</i>	Pointer to the timer.
<i>initialTicks</i>	Initial count to be set for the timer.
<i>rescheduleTicks</i>	Reload count for the timer.

5.27.12.86 uint32\_t CyU3PTimerPerfGet ( CyU3PTimer \* *timer\_p*, uint32\_t \* *activateCnt\_p*, uint32\_t \* *reactivateCnt\_p*, uint32\_t \* *deactivateCnt\_p*, uint32\_t \* *expiryCnt\_p* )

Get timer specific performance profile information.

#### Description

The ThreadX operating system supports collection of system performance data on all OS objects during firmware execution. This function retrieves the performance statistics associated with a specific OS timer object.

#### Return Value

CY\_U3P\_SUCCESS if the data has been retrieved.

CY\_U3P\_ERROR\_BAD\_POINTER if the timer pointer is not valid.

CY\_U3P\_ERROR\_OPERN\_DISABLED if the performance collection is not enabled in the library.

#### Parameters

<i>timer_p</i>	Pointer to timer to be queried.
<i>activateCnt_p</i>	Destination pointer to be filled with count of explicit timer activations.
<i>reactivateCnt_p</i>	Destination pointer to be filled with count of automatic timer re-activations.
<i>deactivateCnt_p</i>	Destination pointer to be filled with count of timer de-activations.

**Parameters**

<i>expiryCnt_p</i>	Destination pointer to be filled with count of timer expirations.
--------------------	---

5.27.12.87 uint32\_t CyU3PTimerStart ( CyU3PTimer \* *timer\_p* )

Start an application timer.

**Description**

This function activates a previously stopped timer. This operation can be used to start a timer that was created with the CYU3P\_NO\_ACTIVATE option, or to re-start a one-shot or continuous timer that was stopped previously.

**Return value**

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

**See also**

[CyU3PTimerCreate](#)  
[CyU3PTimerStop](#)  
[CyU3PTimerModify](#)

**Parameters**

<i>timer_p</i>	Timer to be started.
----------------	----------------------

5.27.12.88 uint32\_t CyU3PTimerStop ( CyU3PTimer \* *timer\_p* )

Stop operation of an application timer.

**Description**

This function can be used to stop operation of an application timer. The parameters associated with the timer can then be modified using the CyU3PTimerModify call.

**Return value**

CY\_U3P\_SUCCESS (0) on success.

Other error codes on failure.

**See also**

[CyU3PTimerStart](#)  
[CyU3PTimerModify](#)

**Parameters**

<i>timer_p</i>	Pointer to timer to be stopped.
----------------	---------------------------------

5.27.12.89 uint32\_t CyU3PTimerSystemPerfGet ( uint32\_t \* *activateCnt\_p*, uint32\_t \* *reactivateCnt\_p*, uint32\_t \* *deactivateCnt\_p*, uint32\_t \* *expiryCnt\_p* )

Get global OS timer performance information.

**Description**

The ThreadX operating system supports collection of system performance data on all OS objects during firmware execution. This function retrieves the performance statistics associated with all timer objects in the system.

**Return Value**

CY\_U3P\_SUCCESS if the data has been retrieved.

CY\_U3P\_ERROR\_BAD\_POINTER if the timer pointer is not valid.

CY\_U3P\_ERROR\_OPERN\_DISABLED if the performance collection is not enabled in the library.

**Parameters**

<i>activateCnt_p</i>	Destination pointer to be filled with count of explicit timer activations.
<i>reactivateCnt_p</i>	Destination pointer to be filled with count of automatic timer re-activations.
<i>deactivateCnt_p</i>	Destination pointer to be filled with count of timer de-activations.
<i>expiryCnt_p</i>	Destination pointer to be filled with count of timer expirations.

**5.27.12.90 void CyU3PUndefinedHandler( void )**

The undefined instruction exception handler.

**Description**

This function gets invoked when the ARM CPU encounters an undefined instruction. This happens when the firmware loses control and jumps to unknown locations. This should not occur as a part of normal operation sequence, and may be seen in case of memory corruption.

**Return value**

None

**See also**

[CyU3PPrefetchHandler](#)  
[CyU3PAbortHandler](#)

**5.28 firmware/u3p\_firmware/inc/cyu3pib.h File Reference**

The Processor Interface Block (PIB) on the FX3 device contains the GPIF-II controller and the associated DMA sockets and configuration registers. The PIB driver in FX3 firmware is responsible for managing PIB bound data transfers and interrupts. This file contains the data types and API definitions for the general P-port functionality that is independent of the electrical protocol implemented by GPIF-II.

```
#include "cyu3os.h"
#include "cyu3types.h"
#include "cyu3system.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

**Data Structures**

- struct [CyU3PPibClock\\_t](#)

*Clock configuration information for the PIB block.*

## Macros

- #define `CYU3P_GET_PIB_ERROR_TYPE`(param) ((`CyU3PPibErrorType`)((param) & 0x3F))  
*Get the PIB error code from the CYU3P\_PIB\_INTR\_ERROR callback argument.*
- #define `CYU3P_GET_GPIF_ERROR_TYPE`(param) ((`CyU3PGpifErrorType`)((param) & 0x7C00))  
*Get the GPIF error code from the CYU3P\_PIB\_INTR\_ERROR callback argument.*

## Typedefs

- typedef struct `CyU3PPibClock_t` `CyU3PPibClock_t`  
*Clock configuration information for the PIB block.*
- typedef enum `CyU3PPibDlMode_t` `CyU3PPibDlMode_t`  
*PIB DLL operating mode.*
- typedef enum `CyU3PPibClockPhase_t` `CyU3PPibClockPhase_t`  
*List of PIB clock phases generated by the DLL.*
- typedef enum `CyU3PPibErrorType` `CyU3PPibErrorType`  
*Enumeration of P-port error types.*
- typedef enum `CyU3PGpifErrorType` `CyU3PGpifErrorType`  
*Enumeration of GPIF error types.*
- typedef enum `CyU3PPibIntrType` `CyU3PPibIntrType`  
*Enumeration of P-port interrupt event types.*
- typedef void(\* `CyU3PPibIntrCb_t`) (`CyU3PPibIntrType` cbType, `uint16_t` cbArg)  
*Type of callback function that will be called on receiving a generic P-port interrupt.*
- typedef enum `CyU3PPmmcState` `CyU3PPmmcState`  
*List of MMC-Slave states.*
- typedef enum `CyU3PPmmcEventType` `CyU3PPmmcEventType`  
*List of MMC-Slave Event notifications.*
- typedef void(\* `CyU3PPmmcIntrCb_t`) (`CyU3PPmmcEventType` cbType, `uint32_t` cbArg)  
*Callback function type used for MMC-Slave event notifications.*

## Enumerations

- enum `CyU3PPibDlMode_t` { `CYU3P_PIB_DLL_MASTER` = 0, `CYU3P_PIB_DLL_MASTER_SLAVE`, `CYU3P_PIB_DLL_SLAVE` }  
*PIB DLL operating mode.*
- enum `CyU3PPibClockPhase_t`{  
`CYU3P_PIB_CLK_PHASE_00` = 0, `CYU3P_PIB_CLK_PHASE_01`, `CYU3P_PIB_CLK_PHASE_02`, `CYU3P_PIB_CLK_PHASE_03`,  
`CYU3P_PIB_CLK_PHASE_04`, `CYU3P_PIB_CLK_PHASE_05`, `CYU3P_PIB_CLK_PHASE_06`, `CYU3P_PIB_CLK_PHASE_07`,  
`CYU3P_PIB_CLK_PHASE_08`, `CYU3P_PIB_CLK_PHASE_09`, `CYU3P_PIB_CLK_PHASE_10`, `CYU3P_PIB_CLK_PHASE_11`,  
`CYU3P_PIB_CLK_PHASE_12`, `CYU3P_PIB_CLK_PHASE_13`, `CYU3P_PIB_CLK_PHASE_14`, `CYU3P_PIB_CLK_PHASE_15` }  
*List of PIB clock phases generated by the DLL.*
- enum `CyU3PPibErrorType`{  
`CYU3P_PIB_ERR_NONE` = 0, `CYU3P_PIB_ERR_THR0_DIRECTION`, `CYU3P_PIB_ERR_THR1_DIRECTION`,  
`CYU3P_PIB_ERR_THR2_DIRECTION`,  
`CYU3P_PIB_ERR_THR3_DIRECTION`, `CYU3P_PIB_ERR_THR0_WR_OVERRUN`, `CYU3P_PIB_ERR_THR1_WR_OVERRUN`,  
`CYU3P_PIB_ERR_THR2_WR_OVERRUN`,  
`CYU3P_PIB_ERR_THR3_WR_OVERRUN`, `CYU3P_PIB_ERR_THR0_RD_UNDERRUN`, `CYU3P_PIB_ERR_THR1_RD_UNDERRUN`,  
`CYU3P_PIB_ERR_THR2_RD_UNDERRUN`,  
`CYU3P_PIB_ERR_THR3_RD_UNDERRUN`, `CYU3P_PIB_ERR_THR0_SCK_INACTIVE` = 0x12, `CYU3P_PIB_ERR_SCK_INACTIVE` = 0x13 }  
*PIB error types.*

```
_PIB_ERR_THR0_ADAP_OVERRUN, CYU3P_PIB_ERR_THR0_ADAP_UNDERRUN,
CYU3P_PIB_ERR THR0_RD_FORCE_END, CYU3P_PIB_ERR THR0_RD_BURST, CYU3P_PIB_ERR
THR1_SCK_INACTIVE = 0x1A, CYU3P_PIB_ERR THR1_ADAP_OVERRUN,
CYU3P_PIB_ERR THR1_ADAP_UNDERRUN, CYU3P_PIB_ERR THR1_RD_FORCE_END, CYU3P_P
IB_ERR THR1_RD_BURST, CYU3P_PIB_ERR THR2_SCK_INACTIVE = 0x22,
CYU3P_PIB_ERR THR2_ADAP_OVERRUN, CYU3P_PIB_ERR THR2_ADAP_UNDERRUN, CYU3P_P
IB_ERR THR2_RD_FORCE_END, CYU3P_PIB_ERR THR2_RD_BURST,
CYU3P_PIB_ERR THR3_SCK_INACTIVE = 0x2A, CYU3P_PIB_ERR THR3_ADAP_OVERRUN, CYU3
P_PIB_ERR THR3_ADAP_UNDERRUN, CYU3P_PIB_ERR THR3_RD_FORCE_END,
CYU3P_PIB_ERR THR3_RD_BURST }
```

*Enumeration of P-port error types.*

- enum CyU3PGpifErrorType {
 CYU3P\_GPIF\_ERR\_NONE = 0, CYU3P\_GPIF\_ERR\_INADDR\_OVERWRITE = 0x0400, CYU3P\_GPIF\_E
 RR\_EGADDR\_INVALID = 0x0800, CYU3P\_GPIF\_ERR\_DATA\_READ\_ERR = 0x0C00,
 CYU3P\_GPIF\_ERR\_DATA\_WRITE\_ERR = 0x1000, CYU3P\_GPIF\_ERR\_ADDR\_READ\_ERR = 0x1400,
 CYU3P\_GPIF\_ERR\_ADDR\_WRITE\_ERR = 0x1800, CYU3P\_GPIF\_ERR\_INVALID\_STATE = 0x2000 }

*Enumeration of GPIF error types.*

- enum CyU3PPbibIntrType { CYU3P\_PIB\_INTR\_DLL\_UPDATE = 1, CYU3P\_PIB\_INTR\_PPCONFIG = 2, C
 YU3P\_PIB\_INTR\_ERROR = 4 }

*Enumeration of P-port interrupt event types.*

- enum CyU3PPmmcState {
 CY\_U3P\_PMMC\_IDLE = 0, CY\_U3P\_PMMC\_READY, CY\_U3P\_PMMC\_IDENTIFICATION, CY\_U3P\_PMM
 MC\_STANDBY,
 CY\_U3P\_PMMC\_TRANS, CY\_U3P\_PMMC\_SENDDATA, CY\_U3P\_PMMC\_RECVDATA, CY\_U3P\_PMM
 C\_PGM,
 CY\_U3P\_PMMC\_DISCONNECT, CY\_U3P\_PMMC\_BUSTEST, CY\_U3P\_PMMC\_SLEEP, CY\_U3P\_PMM
 C\_WAITIRQ,
 CY\_U3P\_PMMC\_INACTIVE, CY\_U3P\_PMMC\_BOOT, CY\_U3P\_PMMC\_PREIDLE, CY\_U3P\_PMMC\_PR
 EBOOT }

*List of MMC-Slave states.*

- enum CyU3PPmmcEventType {
 CYU3P\_PMMC\_GOIDLE\_CMD = 0, CYU3P\_PMMC\_CMD5\_SLEEP, CYU3P\_PMMC\_CMD5\_AWAKE, C
 YU3P\_PMMC\_CMD6\_SWITCH,
 CYU3P\_PMMC\_CMD12\_STOP, CYU3P\_PMMC\_CMD15\_INACTIVE, CYU3P\_PMMC\_DIRECT\_WRITE,
 CYU3P\_PMMC\_DIRECT\_READ,
 CYU3P\_PMMC\_SOCKET\_NOT\_READY, CYU3P\_PMMC\_CMD7\_SELECT }

*List of MMC-Slave Event notifications.*

## Functions

- CyU3PReturnStatus\_t CyU3PPibSelectMmcSlaveMode (void)
 

*Select the MMC Slave mode of operation for the P-port interface block.*
- CyU3PReturnStatus\_t CyU3PPibInit (CyBool\_t doInit, CyU3PPibClock\_t \*pibClock)
 

*Initialize the P-port interface block.*
- CyU3PReturnStatus\_t CyU3PPibDelInit (void)
 

*De-initialize the P-port interface block.*
- CyU3PReturnStatus\_t CyU3PPibDllConfigure (CyU3PPibDllMode\_t dll\_mode, uint16\_t slave\_delay, Cy
 U3PPibClockPhase\_t core\_phase, CyU3PPibClockPhase\_t sync\_phase, CyU3PPibClockPhase\_t output
 \_phase, CyBool\_t should\_lock)
 

*Function to configure the PIB DLL.*
- CyU3PReturnStatus\_t CyU3PSetPportDriveStrength (CyU3PDriveStrengthState\_t pportDriveStrength)
 

*Set the IO drive strength for the Pport.*
- void CyU3PPibRegisterCallback (CyU3PPibIntrCb\_t cbFunc, uint32\_t intMask)
 

*Register a callback function for notification of PIB interrupts.*

- [CyU3PReturnStatus\\_t CyU3PPibSetInterruptPriority \(CyBool\\_t isHigh\)](#)  
*Set the priority level for PIB/GPIF interrupts in the FX3 system.*
- [void CyU3PPibSelectIntSources \(CyBool\\_t pibSockEn, CyBool\\_t gpifIntEn, CyBool\\_t pibErrEn, CyBool\\_t mboxIntEn, CyBool\\_t wakeupEn\)](#)  
*Select the sources that trigger an FX3 interrupt to external processor.*
- [void CyU3PPmmcRegisterCallback \(CyU3PPmmcIntrCb\\_t cbFunc\)](#)  
*Register a callback for notification of MMC-Slave (PMMC) events.*
- [CyU3PReturnStatus\\_t CyU3PPmmcEnableDirectAccess \(CyBool\\_t enable, uint8\\_t wrSock, uint8\\_t rdSock\)](#)  
*Enable MMC direct read/write functionality.*
- [void CyU3PPibSetPmmcBusyMode \(uint32\\_t waitBuffer, uint32\\_t waitCount\)](#)  
*Control the busy signalling on the PMMC interface.*
- [void CyU3PPibSetSocketEop \(uint32\\_t eopMask\)](#)  
*Select the sockets on which all PMMC input buffers will be treated as End-Of-Transfer.*

### 5.28.1 Detailed Description

The Processor Interface Block (PIB) on the FX3 device contains the GPIF-II controller and the associated DMA sockets and configuration registers. The PIB driver in FX3 firmware is responsible for managing PIB bound data transfers and interrupts. This file contains the data types and API definitions for the general P-port functionality that is independent of the electrical protocol implemented by GPIF-II.

### 5.28.2 Macro Definition Documentation

#### 5.28.2.1 #define CYU3P\_GET\_GPIF\_ERROR\_TYPE( param ) ((CyU3PGpifErrorType)((param) & 0x7C00))

Get the GPIF error code from the CYU3P\_PIB\_INTR\_ERROR callback argument.

##### Description

This macro is used to get the GPIF error code part from the cbArg parameter passed to the PIB callback as part of a CYU3P\_PIB\_INTR\_ERROR event.

##### See also

[CyU3PGpifErrorType](#)

#### 5.28.2.2 #define CYU3P\_GET\_PIB\_ERROR\_TYPE( param ) ((CyU3PPibErrorType)((param) & 0x3F))

Get the PIB error code from the CYU3P\_PIB\_INTR\_ERROR callback argument.

##### Description

This macro is used to get the PIB error code part from the cbArg parameter passed to the PIB callback as part of a CYU3P\_PIB\_INTR\_ERROR event.

##### See also

[CyU3PPibErrorType](#)

### 5.28.3 Typedef Documentation

#### 5.28.3.1 typedef enum CyU3PGpifErrorType CyU3PGpifErrorType

Enumeration of GPIF error types.

**Description**

In addition to the PIB level errors, there can be cases where GPIF state machine specific errors occur during G $\leftarrow$ PIF operation. The cbArg parameter passed to the callback in the case of a CYU3P\_PIB\_INTR\_ERROR event is composed of a PIB error code as well as a GPIF error code. This enumerated type lists the various GPIF specific error types that are defined for the FX3 device.

The CYU3P\_GET\_GPIF\_ERROR\_TYPE macro can be used to get the GPIF error code from the cbArg parameter.

**See also**

[CyU3PPibErrorType](#)  
[CYU3P\\_GET\\_GPIF\\_ERROR\\_TYPE](#)

**5.28.3.2 typedef struct CyU3PPibClock\_t CyU3PPibClock\_t**

Clock configuration information for the PIB block.

**Description**

The clock for the PIB block is derived from the SYS\_CLK. The base clock and frequency divider values are selected through this structure.

The default values used by the driver are:

clkDiv = 2, isHalfDiv = CyFalse, isDIIEnable = CyFalse, and clkSrc = CY\_U3P\_SYS\_CLK.

**See also**

[CyU3PSysClockSrc\\_t](#)  
[CyU3PPibInit](#)

**5.28.3.3 typedef enum CyU3PPibClockPhase\_t CyU3PPibClockPhase\_t**

List of PIB clock phases generated by the DLL.

**Description**

The DLL block in the PIB takes in the master clock (external or internal clock divided from the system clock) and generates multiple delayed versions of it. The delays applied are in increments of 360 / 16 (22.5 degrees). This type lists the various clock phases that the DLL can generate.

**See also**

[CyU3PPibDIIConfigure](#)

**5.28.3.4 typedef enum CyU3PPibDIIMode\_t CyU3PPibDIIMode\_t**

PIB DLL operating mode.

**Description**

The DLL block is used to provide multiple delayed clock signals that are required to meet the PIB (GPIF II) Interface timing across all device operating corners for various interface types. The DLL needs to be configured differently when the GPIF is being used in different modes (synchronous/asynchronous, input/output clock etc.). This type lists the possible PIB DLL operating modes.

When operating in the normal (master) mode, the DLL block has limitations with respect to the minimum interface clock frequency that can be supported.

When the clock is running at slower rates, the DLL needs to be operated in a master-slave mode. Here, two DLLs are used in sequence; with the first (master) DLL generating the delay word that controls the operation of the second (slave) DLL.

It is also possible to have the slave delay use a user programmed fixed delay value instead of using the delay value generated by the master DLL. This mode is equivalent to having a single slave mode DLL.

See also

[CyU3PPibDllConfigure](#)

#### 5.28.3.5 `typedef enum CyU3PPibErrorType CyU3PPibErrorType`

Enumeration of P-port error types.

##### Description

The P-port interface block (PIB) of the FX3 device can encounter various errors during data transfers performed across the GPIF interface. This enumerated type lists the PIB level error conditions that are notified to the user application through the CYU3P\_PIB\_INTR\_ERROR interrupt callback. The cbArg parameter passed to the callback will indicate the PIB error code as well as the GPIF error code.

The CYU3P\_GET\_PIB\_ERROR\_TYPE macro can be used to decode the PIB error type from the callback argument.

See also

[CyU3PPibIntrType](#)  
[CyU3PPibRegisterCallback](#)  
[CyU3PGpifErrorType](#)  
[CYU3P\\_GET\\_PIB\\_ERROR\\_TYPE](#)

#### 5.28.3.6 `typedef void(* CyU3PPibIntrCb_t)(CyU3PPibIntrType cbType, uint16_t cbArg )`

Type of callback function that will be called on receiving a generic P-port interrupt.

##### Description

The P-port interface block (PIB) of the FX3 device has some interrupt sources that are unconnected with the GPIF hardware or state machine. This function is used to register a callback function that will be invoked when one of these interrupts is triggered.

See also

[CyU3PPibRegisterCallback](#)

#### 5.28.3.7 `typedef enum CyU3PPibIntrType CyU3PPibIntrType`

Enumeration of P-port interrupt event types.

##### Description

The P-port interface block (PIB) of the FX3 device has some interrupt sources that are unconnected with the GPIF hardware or state machine. These interrupts indicate events such as error conditions, mailbox message reception and Interface Config register updates. This enumerated type lists the various interrupt events that are valid for the PIB interrupt.

See also

[CyU3PPibIntrCb\\_t](#)  
[CyU3PPibRegisterCallback](#)

#### 5.28.3.8 `typedef enum CyU3PPmmcEventType CyU3PPmmcEventType`

List of MMC-Slave Event notifications.

##### Description

This type lists the various event notifications that are provided by the FX3 device when configured in the MMC-Slave mode.

See also

[CyU3PPmmcIntrCb\\_t](#)

#### 5.28.3.9 `typedef void(* CyU3PPmmcIntrCb_t)(CyU3PPmmcEventType cbType, uint32_t cbArg )`

Callback function type used for MMC-Slave event notifications.

**Description**

This function pointer type defines the signature of the callback function that will be called to notify the user of MMC-Slave (PMMC) Mode events.

See also

[CyU3PPmmcEventType](#)  
[CyU3PPmmcRegisterCallback](#)

#### 5.28.3.10 `typedef enum CyU3PPmmcState CyU3PPmmcState`

List of MMC-Slave states.

**Description**

When the FX3 is configured as an MMC-Slave device, the MMC interface goes through a set of states (as defined in the MMC specification) in response to commands issued by the MMC host. This type lists the various MMC Slave states for the FX3 device.

### 5.28.4 Enumeration Type Documentation

#### 5.28.4.1 `enum CyU3PGpifErrorType`

Enumeration of GPIF error types.

**Description**

In addition to the PIB level errors, there can be cases where GPIF state machine specific errors occur during G $\leftrightarrow$ PIF operation. The cbArg parameter passed to the callback in the case of a CYU3P\_PIB\_INTR\_ERROR event is composed of a PIB error code as well as a GPIF error code. This enumerated type lists the various GPIF specific error types that are defined for the FX3 device.

The CYU3P\_GET\_GPIF\_ERROR\_TYPE macro can be used to get the GPIF error code from the cbArg parameter.

See also

[CyU3PPibErrorType](#)  
[CYU3P\\_GET\\_GPIF\\_ERROR\\_TYPE](#)

Enumerator

**CYU3P\_GPIF\_ERR\_NONE** No GPIF state machine errors.

**CYU3P\_GPIF\_ERR\_INADDR\_OVERWRITE** Content of INGRESS\_ADDR register is overwritten before read.

**CYU3P\_GPIF\_ERR\_EGADDR\_INVALID** Attempt to read EGRESS\_ADDR register before it is written to.

**CYU3P\_GPIF\_ERR\_DATA\_READ\_ERR** Read from DMA data thread which is not ready.

**CYU3P\_GPIF\_ERR\_DATA\_WRITE\_ERR** Write to DMA data thread which is not ready.

**CYU3P\_GPIF\_ERR\_ADDR\_READ\_ERR** Read from DMA address thread which is not ready.  
**CYU3P\_GPIF\_ERR\_ADDR\_WRITE\_ERR** Write to DMA address thread which is not ready.  
**CYU3P\_GPIF\_ERR\_INVALID\_STATE** GPIF state machine has reached an invalid state.

#### 5.28.4.2 enum CyU3PPibClockPhase\_t

List of PIB clock phases generated by the DLL.

##### Description

The DLL block in the PIB takes in the master clock (external or internal clock divided from the system clock) and generates multiple delayed versions of it. The delays applied are in increments of 360 / 16 (22.5 degrees). This type lists the various clock phases that the DLL can generate.

##### See also

[CyU3PPibDllConfigure](#)

##### Enumerator

**CYU3P\_PIB\_CLK\_PHASE\_00** Clock phase 0. No delay from master clock.  
**CYU3P\_PIB\_CLK\_PHASE\_01** Clock phase 1. Lags phase 0 by 22.5 degrees.  
**CYU3P\_PIB\_CLK\_PHASE\_02** Clock phase 2. Lags phase 0 by 45.0 degrees.  
**CYU3P\_PIB\_CLK\_PHASE\_03** Clock phase 3. Lags phase 0 by 67.5 degrees.  
**CYU3P\_PIB\_CLK\_PHASE\_04** Clock phase 4. Lags phase 0 by 90.0 degrees.  
**CYU3P\_PIB\_CLK\_PHASE\_05** Clock phase 5. Lags phase 0 by 112.5 degrees.  
**CYU3P\_PIB\_CLK\_PHASE\_06** Clock phase 6. Lags phase 0 by 135.0 degrees.  
**CYU3P\_PIB\_CLK\_PHASE\_07** Clock phase 7. Lags phase 0 by 157.5 degrees.  
**CYU3P\_PIB\_CLK\_PHASE\_08** Clock phase 8. Lags phase 0 by 180.0 degrees.  
**CYU3P\_PIB\_CLK\_PHASE\_09** Clock phase 9. Lags phase 0 by 202.5 degrees.  
**CYU3P\_PIB\_CLK\_PHASE\_10** Clock phase 10. Lags phase 0 by 225.0 degrees.  
**CYU3P\_PIB\_CLK\_PHASE\_11** Clock phase 11. Lags phase 0 by 247.5 degrees.  
**CYU3P\_PIB\_CLK\_PHASE\_12** Clock phase 12. Lags phase 0 by 270.0 degrees.  
**CYU3P\_PIB\_CLK\_PHASE\_13** Clock phase 13. Lags phase 0 by 292.5 degrees.  
**CYU3P\_PIB\_CLK\_PHASE\_14** Clock phase 14. Lags phase 0 by 315.0 degrees.  
**CYU3P\_PIB\_CLK\_PHASE\_15** Clock phase 15. Lags phase 0 by 337.5 degrees.

#### 5.28.4.3 enum CyU3PPibDllMode\_t

PIB DLL operating mode.

##### Description

The DLL block is used to provide multiple delayed clock signals that are required to meet the PIB (GPIF II) Interface timing across all device operating corners for various interface types. The DLL needs to be configured differently when the GPIF is being used in different modes (synchronous/asynchronous, input/output clock etc.). This type lists the possible PIB DLL operating modes.

When operating in the normal (master) mode, the DLL block has limitations with respect to the minimum interface clock frequency that can be supported.

When the clock is running at slower rates, the DLL needs to be operated in a master-slave mode. Here, two DLLs are used in sequence; with the first (master) DLL generating the delay word that controls the operation of the second (slave) DLL.

It is also possible to have the slave delay use a user programmed fixed delay value instead of using the delay value generated by the master DLL. This mode is equivalent to having a single slave mode DLL.

See also

[CyU3PPibDllConfigure](#)

Enumerator

**CYU3P\_PIB\_DLL\_MASTER** Single Master mode DLL.

**CYU3P\_PIB\_DLL\_MASTER\_SLAVE** Two DLLs operating in a master-slave configuration.

**CYU3P\_PIB\_DLL\_SLAVE** Single Slave mode DLL.

#### 5.28.4.4 enum CyU3PPibErrorType

Enumeration of P-port error types.

**Description**

The P-port interface block (PIB) of the FX3 device can encounter various errors during data transfers performed across the GPIF interface. This enumerated type lists the PIB level error conditions that are notified to the user application through the CYU3P\_PIB\_INTR\_ERROR interrupt callback. The cbArg parameter passed to the callback will indicate the PIB error code as well as the GPIF error code.

The CYU3P\_GET\_PIB\_ERROR\_TYPE macro can be used to decode the PIB error type from the callback argument.

See also

[CyU3PPibIntrType](#)

[CyU3PPibRegisterCallback](#)

[CyU3PGpifErrorType](#)

[CYU3P\\_GET\\_PIB\\_ERROR\\_TYPE](#)

Enumerator

**CYU3P\_PIB\_ERR\_NONE** No errors detected.

**CYU3P\_PIB\_ERR\_THR0\_DIRECTION** Bad transfer direction (read on a write socket or vice versa) error on one of the Thread 0 sockets.

**CYU3P\_PIB\_ERR\_THR1\_DIRECTION** Bas transfer direction (read on a write socket or vice versa) error on one of the Thread 1 sockets.

**CYU3P\_PIB\_ERR\_THR2\_DIRECTION** Bad transfer direction (read on a write socket or vice versa) error on one of the Thread 2 sockets.

**CYU3P\_PIB\_ERR\_THR3\_DIRECTION** Bas transfer direction (read on a write socket or vice versa) error on one of the Thread 3 sockets.

**CYU3P\_PIB\_ERR\_THR0\_WR\_OVERRUN** Write overrun (write beyond available buffer size) error on one of the Thread 0 sockets.

**CYU3P\_PIB\_ERR\_THR1\_WR\_OVERRUN** Write overrun (write beyond available buffer size) error on one of the Thread 1 sockets.

**CYU3P\_PIB\_ERR\_THR2\_WR\_OVERRUN** Write overrun (write beyond available buffer size) error on one of the Thread 2 sockets.

**CYU3P\_PIB\_ERR\_THR3\_WR\_OVERRUN** Write overrun (write beyond available buffer size) error on one of the Thread 3 sockets.

**CYU3P\_PIB\_ERR\_THR0\_RD\_UNDERRUN** Read underrun (read beyond available data size) error on one of the Thread 0 sockets.

**CYU3P\_PIB\_ERR\_THR1\_RD\_UNDERRUN** Read underrun (read beyond available data size) error on one of the Thread 1 sockets.

**CYU3P\_PIB\_ERR\_THR2\_RD\_UNDERRUN** Read underrun (read beyond available data size) error on one of the Thread 2 sockets.

**CYU3P\_PIB\_ERR\_THR3\_RD\_UNDERRUN** Read underrun (read beyond available data size) error on one of the Thread 3 sockets.

**CYU3P\_PIB\_ERR\_THR0\_SCK\_INACTIVE** One of the Thread 0 sockets became inactive during transfer.

**CYU3P\_PIB\_ERR\_THR0\_ADAP\_OVERRUN** DMA controller overrun on a write to one of the Thread 0 sockets. This typically happens if the DMA controller cannot keep up with the incoming data rate.

**CYU3P\_PIB\_ERR\_THR0\_ADAP\_UNDERRUN** DMA controller underrun on a read from one of the Thread 0 sockets. This is also a result of the DMA controller not being able to keep up with the desired interface data rate.

**CYU3P\_PIB\_ERR\_THR0\_RD\_FORCE\_END** A DMA read operation from Thread 0 socket was forcibly ended by wrapping up the socket.

**CYU3P\_PIB\_ERR\_THR0\_RD\_BURST** A socket switch was forced in the middle of a read burst from a Thread 0 socket.

**CYU3P\_PIB\_ERR\_THR1\_SCK\_INACTIVE** One of the Thread 1 sockets became inactive during transfer.

**CYU3P\_PIB\_ERR\_THR1\_ADAP\_OVERRUN** DMA controller overrun on a write to one of the Thread 1 sockets. This typically happens if the DMA controller cannot keep up with the incoming data rate.

**CYU3P\_PIB\_ERR\_THR1\_ADAP\_UNDERRUN** DMA controller underrun on a read from one of the Thread 1 sockets. This is also a result of the DMA controller not being able to keep up with the desired interface data rate.

**CYU3P\_PIB\_ERR\_THR1\_RD\_FORCE\_END** A DMA read operation from Thread 1 socket was forcibly ended by wrapping up the socket.

**CYU3P\_PIB\_ERR\_THR1\_RD\_BURST** A socket switch was forced in the middle of a read burst from a Thread 1 socket.

**CYU3P\_PIB\_ERR\_THR2\_SCK\_INACTIVE** One of the Thread 2 sockets became inactive during transfer.

**CYU3P\_PIB\_ERR\_THR2\_ADAP\_OVERRUN** DMA controller overrun on a write to one of the Thread 2 sockets. This typically happens if the DMA controller cannot keep up with the incoming data rate.

**CYU3P\_PIB\_ERR\_THR2\_ADAP\_UNDERRUN** DMA controller underrun on a read from one of the Thread 2 sockets. This is also a result of the DMA controller not being able to keep up with the desired interface data rate.

**CYU3P\_PIB\_ERR\_THR2\_RD\_FORCE\_END** A DMA read operation from Thread 2 socket was forcibly ended by wrapping up the socket.

**CYU3P\_PIB\_ERR\_THR2\_RD\_BURST** A socket switch was forced in the middle of a read burst from a Thread 2 socket.

**CYU3P\_PIB\_ERR\_THR3\_SCK\_INACTIVE** One of the Thread 3 sockets became inactive during transfer.

**CYU3P\_PIB\_ERR\_THR3\_ADAP\_OVERRUN** DMA controller overrun on a write to one of the Thread 3 sockets. This typically happens if the DMA controller cannot keep up with the incoming data rate.

**CYU3P\_PIB\_ERR\_THR3\_ADAP\_UNDERRUN** DMA controller underrun on a read from one of the Thread 3 sockets. This is also a result of the DMA controller not being able to keep up with the desired interface data rate.

**CYU3P\_PIB\_ERR\_THR3\_RD\_FORCE\_END** A DMA read operation from Thread 3 socket was forcibly ended by wrapping up the socket.

**CYU3P\_PIB\_ERR\_THR3\_RD\_BURST** A socket switch was forced in the middle of a read burst from a Thread 3 socket.

#### 5.28.4.5 enum CyU3PPibIntrType

Enumeration of P-port interrupt event types.

##### Description

The P-port interface block (PIB) of the FX3 device has some interrupt sources that are unconnected with the GPIF hardware or state machine. These interrupts indicate events such as error conditions, mailbox message reception and Interface Config register updates. This enumerated type lists the various interrupt events that are valid for the PIB interrupt.

See also

[CyU3PPibIntrCb\\_t](#)  
[CyU3PPibRegisterCallback](#)

Enumerator

**CYU3P\_PIB\_INTR\_DLL\_UPDATE** Indicates that a PIB DLL lock or unlock event has occurred. The cbArg parameter indicates whether the DLL is currently locked (non-zero) or unlocked (zero).

**CYU3P\_PIB\_INTR\_PPCONFIG** Indicates that the PIB config register has been written to through the GPIF interface. The value written into the PP\_CONFIG register is passed as the cbArg parameter. This value typically has the following format:

Bit 15 : User defined.  
 Bit 14 : DRQ polarity. 0 = active low, 1 = active high.  
 Bit 13 : User defined.  
 Bit 12 : DRQ override. 0 = Normal. 1 = Force DRQ on.  
 Bit 11 : INT polarity. 0 = active low. 1 = active high.  
 Bit 10 : INT value. Specifies INT state when override is on.  
 Bit 9 : INT override. 0 = Normal. 1 = Overridden.  
 Bits 8 - 7 : User defined.  
 Bit 7 : User defined.  
 Bit 6 : CFGMODE. Must be 1.  
 Bits 5 - 4 : User defined.  
 Bits 3 - 0 : User defined. Usually set to log2(burst size).

**CYU3P\_PIB\_INTR\_ERROR** Indicates that an error condition has been encountered by the PIB/GPIF block. The type of error encountered is passed through the cbArg parameter. See CyU3PPibErrorType for the possible error types.

#### 5.28.4.6 enum CyU3PPmmcEventType

List of MMC-Slave Event notifications.

##### Description

This type lists the various event notifications that are provided by the FX3 device when configured in the MMC-Slave mode.

See also

[CyU3PPmmcIntrCb\\_t](#)

Enumerator

**CYU3P\_PMMC\_GOIDLE\_CMD** GO\_IDLE\_STATE (CMD0) command has been received.

**CYU3P\_PMMC\_CMD5\_SLEEP** CMD5(SLEEP) command has been used to place FX3 into suspend mode.

**CYU3P\_PMMC\_CMD5\_AWAKE** CMD5(AWAKE) command has been used to wake FX3 from suspend mode.

**CYU3P\_PMMC\_CMD6\_SWITCH** SWITCH (CMD6) command has been received.

**CYU3P\_PMMC\_CMD12\_STOP** STOP\_TRANSMISSION (CMD12) command has been received.

**CYU3P\_PMMC\_CMD15\_INACTIVE** GO\_INACTIVE\_STATE (CMD15) command has been received.

**CYU3P\_PMMC\_DIRECT\_WRITE** Write command has been received on the designated Direct Write socket.

**CYU3P\_PMMC\_DIRECT\_READ** Read command has been received on the designated Direct Read socket.

**CYU3P\_PMMC\_SOCKET\_NOT\_READY** Socket being read/written by the host is not ready.

**CYU3P\_PMMC\_CMD7\_SELECT** SELECT\_CARD (CMD7) command has been received.

### 5.28.4.7 enum CyU3PPmmcState

List of MMC-Slave states.

#### Description

When the FX3 is configured as an MMC-Slave device, the MMC interface goes through a set of states (as defined in the MMC specification) in response to commands issued by the MMC host. This type lists the various MMC Slave states for the FX3 device.

Enumerator

- CY\_U3P\_PMMC\_IDLE** Idle (reset) state.
- CY\_U3P\_PMMC\_READY** Ready state.
- CY\_U3P\_PMMC\_IDENTIFICATION** Identification state.
- CY\_U3P\_PMMC\_STANDBY** Standby state.
- CY\_U3P\_PMMC\_TRANS** Transfer (tran) state.
- CY\_U3P\_PMMC\_SENDDATA** Send data (read) state.
- CY\_U3P\_PMMC\_RECVDATA** Receive data (write) state.
- CY\_U3P\_PMMC\_PGM** Program state.
- CY\_U3P\_PMMC\_DISCONNECT** Disconnect state.
- CY\_U3P\_PMMC\_BUSTEST** Bus-test state.
- CY\_U3P\_PMMC\_SLEEP** Sleep state.
- CY\_U3P\_PMMC\_WAITIRQ** Wait IRQ state.
- CY\_U3P\_PMMC\_INACTIVE** Inactive state.
- CY\_U3P\_PMMC\_BOOT** Boot state (MMC 4.3)
- CY\_U3P\_PMMC\_PREIDLE** Pre-Idle state (MMC 4.4)
- CY\_U3P\_PMMC\_PREBOOT** Pre-boot state (MMC 4.4)

## 5.28.5 Function Documentation

### 5.28.5.1 CyU3PReturnStatus\_t CyU3PPibDelInit( void )

De-initialize the P-port interface block.

#### Description

This function disables and powers off the P-port interface block.

#### Return value

CY\_U3P\_SUCCESS - If the operation is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - If the block has not been initialized.

#### See also

[CyU3PPibInit](#)

### 5.28.5.2 CyU3PReturnStatus\_t CyU3PPibDIIConfigure( CyU3PPibDIIMode\_t dll\_mode, uint16\_t slave\_delay, CyU3PPibClockPhase\_t core\_phase, CyU3PPibClockPhase\_t sync\_phase, CyU3PPibClockPhase\_t output\_phase, CyBool\_t should\_lock )

Function to configure the PIB DLL.

#### Description

The DLL block is used to provide multiple delayed clock signals that are required to meet the PIB (GPIF II) Interface timing across all device operating corners for various interface types. The DLL needs to be configured differently

when the GPIF is being used in different modes (synchronous/asynchronous, input/output clock etc.). This API should be called after [CyU3PPibInit\(\)](#) is called, and before the GPIF configuration is loaded. If the GPIF configuration uses an external clock (PCLK) input, it should be present at the time of calling this API. The `isDll←Enable` parameter to [CyU3PPibInit\(\)](#) should be set to CyFalse when this API is called. Please refer to the PIB DLL documentation (provided separately) to identify the parameters that should be passed to this API for various GPIF-II configurations.

#### Return value

`CY_U3P_SUCCESS` - If the operation is successful.

`CY_U3P_ERROR_NOT_STARTED` - If the block has not been initialized.

`CY_U3P_ERROR_BAD_ARGUMENT` - If the `dll_mode` specified is not valid.

#### See also

[CyU3PPibDllMode\\_t](#)  
[CyU3PPibClockPhase\\_t](#)

#### Parameters

<code>dll_mode</code>	DLL operating mode.
<code>slave_delay</code>	Fixed delay value to be used when DLL is in slave mode. This is a 10 bit field (maximum value is 1023).
<code>core_phase</code>	Phase used for clock used to latch input signals.
<code>sync_phase</code>	Phase used for output PCLK. Only relevant in ASYNC mode.
<code>output_phase</code>	Phase used for clock used to drive output signals.
<code>should_lock</code>	Whether the API should wait for the DLL to lock. There is no need for the DLL to lock when the operating frequency is low.

### 5.28.5.3 `CyU3PReturnStatus_t CyU3PPibInit ( CyBool_t dolInit, CyU3PPibClock_t * pibClock )`

Initialize the P-port interface block.

#### Description

This function powers up the P-port interface block. This needs to be the first P-port related function call and should be called before any GPIF related calls are made.

#### Return value

`CY_U3P_SUCCESS` - If the operation is successful.

`CY_U3P_ERROR_NOT_SUPPORTED` - If the FX3 part in use does not support the PIB port.

`CY_U3P_ERROR_BAD_ARGUMENT` - If an incorrect/invalid clock configuration is passed.

`CY_U3P_ERROR_NULL_POINTER` - If the argument `pibClock` is a NULL.

#### See also

[CyU3PPibSelectMmcSlaveMode](#)  
[CyU3PPibDelInit](#)  
[CyU3PPibClock\\_t](#)

#### Parameters

<code>dolInit</code>	Whether to initialize the PIB block. Should generally be set to CyTrue.
<code>pibClock</code>	PIB clock configuration.

#### 5.28.5.4 void CyU3PPibRegisterCallback ( *CyU3PPibIntrCb\_t cbFunc*, *uint32\_t intMask* )

Register a callback function for notification of PIB interrupts.

##### Description

This function registers a callback function that will be called for notification of PIB interrupts and also selects the PIB interrupt sources of interest.

##### Return value

None

##### See also

[CyU3PPibIntrCb\\_t](#)  
[CyU3PPibIntrType](#)

##### Parameters

<i>cbFunc</i>	Callback function pointer being registered.
<i>intMask</i>	Bitmask representing the various interrupts callbacks to be enabled. This value is derived by ORing the various callback types of interest from the <a href="#">CyU3PPibIntrType</a> enumeration.

#### 5.28.5.5 void CyU3PPibSelectIntSources ( *CyBool\_t pibSockEn*, *CyBool\_t gpifIntEn*, *CyBool\_t pibErrEn*, *CyBool\_t mboxIntEn*, *CyBool\_t wakeupEn* )

Select the sources that trigger an FX3 interrupt to external processor.

##### Description

The FX3 device is capable of generating interrupts to the external processor connected on the GPIF port for various reasons. This function is used to select the interrupt sources allowed to interrupt the external processor.

##### Return value

None

##### Parameters

<i>pibSockEn</i>	Whether PIB socket DMA ready status should trigger an interrupt.
<i>gpifIntEn</i>	Whether GPIF state machine can trigger an interrupt.
<i>pibErrEn</i>	Whether an error condition in the PIB/GPIF should trigger an interrupt. The type of error can be detected by reading the PP_ERROR register.
<i>mboxIntEn</i>	Whether a mailbox message ready for reading should trigger an interrupt. The PP_RD_MAILBOX registers should be read to fetch the message indicated by this interrupt.
<i>wakeupEn</i>	Whether a FX3 wakeup from sleep mode should trigger an interrupt. This interrupt source will not be triggered by the FX3 device using the current firmware library.

#### 5.28.5.6 *CyU3PReturnStatus\_t CyU3PPibSelectMmcSlaveMode ( void )*

Select the MMC Slave mode of operation for the P-port interface block.

##### Description

The P-port interface block (PIB) on the FX3/FX3S devices can be configured to function in the GPIF-II mode or in the MMC Slave mode. In the MMC slave mode, the device appears as a MMC 4.2 spec compliant slave device which accepts commands from a MMC host controller.

This function is used to switch the PIB block to the MMC Slave mode (from the default GPIF-II mode). As this is a static mode selection, this function is expected to be called before the PIB block is initialized.

**Return Value**

CY\_U3P\_SUCCESS if the switch request is registered. CY\_U3P\_ERROR\_ALREADY\_STARTED if the PIB block is already active.

**See also**

[CyU3PPibInit](#)  
[CyU3PPibDeInit](#)

**5.28.5.7 CyU3PReturnStatus\_t CyU3PPibSetInterruptPriority ( CyBool\_t *isHigh* )**

Set the priority level for PIB/GPIF interrupts in the FX3 system.

**Description**

By default, PIB and GPIF interrupts in the FX3 system have low priority and can be pre-empted by other interrupts such as the USB interrupt. This API is provided to switch to a higher priority non pre-emptable version of the interrupt handler; or back to the default setting.

**Note**

It is not advisable to set the PIB interrupt as high priority if the CYU3P\_PIB\_INTR\_ERROR callback type is enabled.

**Return value**

CY\_U3P\_SUCCESS if the interrupt priority was updated as specified.  
CY\_U3P\_ERROR\_INVALID\_CALLER if this function is called from interrupt context.

**Parameters**

<i>isHigh</i>	Whether the PIB (GPIF) interrupt is to be assigned high priority.
---------------	---

**5.28.5.8 void CyU3PPibSetPmmcBusyMode ( uint32\_t *waitBuffer*, uint32\_t *waitCount* )**

Control the busy signalling on the PMMC interface.

**Description**

The PMMC busy signalling from FX3 during a write operation can be controlled. Under normal conditions, the busy condition will depend on the state of the DMA buffer. However, this can be changed to reflect a ready status after a fixed number of clock cycles.

**Return Value**

None

**Parameters**

<i>waitBuffer</i>	Bitmask representing the sockets where ready depends on buffer status.
<i>waitCount</i>	Number of clock cycles for busy signaling where it is not based on buffer status.

**5.28.5.9 void CyU3PPibSetSocketEop ( uint32\_t *eopMask* )**

Select the sockets on which all PMMC input buffers will be treated as End-Of-Transfer.

**Description**

It is possible that the DMA buffer used on a PMMC socket is larger than the size of the incoming data transfer. In such cases, the hardware can forcibly set the End-Of-Transfer bit to allow the buffer to be finished and made available. This API selects the PMMC sockets on which the EOT bit is always set by hardware.

**Return Value**

None

**Parameters**

<i>eopMask</i>	Bitmask representing sockets on which the EOT should be set.
----------------	--

**5.28.5.10 CyU3PReturnStatus\_t CyU3PPmmcEnableDirectAccess ( CyBool\_t enable, uint8\_t wrSock, uint8\_t rdSock )**

Enable MMC direct read/write functionality.

**Description**

In the normal mode of operation, the MMC-Slave interface requires a high level protocol layer that maps specific PIB sockets to corresponding data paths. Pass-through data transfers from/to SD/MMC slave devices connected on the Storage port can be facilitated through a pair of designated direct read and write sockets.

If direct read/write is enabled, the firmware is notified whenever a read/write addressed to a specific address range is received. The firmware can use the address to initiate data transfers on the Storage port, and then complete the data transfer through the direct access sockets.

**Return Value**

CY\_U3P\_SUCCESS if the direct access enable/disable is successful. CY\_U3P\_ERROR\_NOT\_STARTED if the PIB block is not enabled. CY\_U3P\_ERROR\_INVALID\_SEQUENCE if the PIB is not configured in PMMC mode. CY\_U3P\_ERROR\_BAD\_ARGUMENT if the sockets specified are not valid.

**Parameters**

<i>enable</i>	Whether to enable or disable the direct data access.
<i>wrSock</i>	PIB socket to be used for direct write operations (should be in the range 16 - 31).
<i>rdSock</i>	PIB socket to be used for direct read operations (should be in the range 0 - 15).

**5.28.5.11 void CyU3PPmmcRegisterCallback ( CyU3PPmmcIntrCb\_t cbFunc )**

Register a callback for notification of MMC-Slave (PMMC) events.

**Description**

This API registers a callback function that will be used to provide notifications of MMC-Slave related events. Please note that this callback will be invoked from the interrupt context, and cannot be used to perform any blocking operations.

**Return Value**

None

**See also**

[CyU3PPmmcIntrCb\\_t](#)  
[CyU3PPmmcEventType](#)

**Parameters**

<i>cbFunc</i>	Callback function pointer.
---------------	----------------------------

### 5.28.5.12 CyU3PReturnStatus\_t CyU3PSetPportDriveStrength ( CyU3PDriveStrengthState\_t pportDriveStrength )

Set the IO drive strength for the Pport.

#### Description

The function sets the IO Drive strength for the P-port signals. The default IO drive strength is set to CY\_U3P\_DS\_THREE\_QUARTER\_STRENGTH.

#### Return value

CY\_U3P\_SUCCESS - If the operation is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the Drive strength requested is invalid.

#### See also

[CyU3PDriveStrengthState\\_t](#)

#### Parameters

<i>pportDriveStrength</i>	Desired drive strength
---------------------------	------------------------

## 5.29 firmware/u3p\_firmware/inc/cyu3sib.h File Reference

This file defines the data types and APIs that support SD/MMC/SDIO peripheral access on the EZ-USB FX3S device.

```
#include <cyu3os.h>
#include <cyu3dma.h>
#include <cyu3types.h>
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

## Data Structures

- struct [CyU3PSibIntfParams](#)  
*Storage interface control parameters.*
- struct [CyU3PSibLunInfo](#)  
*Logical unit properties.*
- struct [CyU3PSibDevInfo](#)  
*Storage (SD/MMC) device properties.*
- struct [CyU3PSdioCardRegs](#)  
*SDIO Card Generic Information and CCCR registers.*

## Macros

- #define [CY\\_U3P\\_SIB0\\_DETECT\\_GPIO](#) (44)  
*GPIO used for card detection on Storage port 0.*
- #define [CY\\_U3P\\_SIB0\\_WRPROT\\_GPIO](#) (43)  
*GPIO used for write protection detection on Storage port 0.*
- #define [CY\\_U3P\\_SIB1\\_WRPROT\\_GPIO](#) (52)  
*GPIO used for write protection detection on Storage port 1.*

## Typedefs

- **typedef enum CyU3PSibPortId CyU3PSibPortId**  
*List of storage ports.*
- **typedef enum CyU3PSibCardDetect CyU3PSibCardDetect**  
*List of card detection methods.*
- **typedef enum CyU3PSibDevType CyU3PSibDevType**  
*List the Storage Device types supported by FX3S.*
- **typedef enum CyU3PSibIntfVoltage CyU3PSibIntfVoltage**  
*List of SD/MMC operating voltage ranges.*
- **typedef enum CyU3PSibEventType CyU3PSibEventType**  
*List of storage related events.*
- **typedef enum CyU3PSibLunType CyU3PSibLunType**  
*List of logical unit (LUN) types.*
- **typedef enum CyU3PSibDevRegType CyU3PSibDevRegType**  
*SD register types.*
- **typedef enum CyU3PSibLunLocation CyU3PSibLunLocation**  
*List of logical unit locations.*
- **typedef enum CyU3PSibOpFreq CyU3PSibOpFreq**  
*List of storage operating frequencies.*
- **typedef struct CyU3PSibIntfParams CyU3PSibIntfParams\_t**  
*Storage interface control parameters.*
- **typedef struct CyU3PSibLunInfo CyU3PSibLunInfo\_t**  
*Logical unit properties.*
- **typedef struct CyU3PSibDeviceInfo CyU3PSibDeviceInfo\_t**  
*Storage (SD/MMC) device properties.*
- **typedef void(\* CyU3PSibEvtCbk\_t) (uint8\_t portId, CyU3PSibEventType evt, CyU3PReturnStatus\_t status)**  
*Storage event callback function type.*
- **typedef struct CyU3PSdioCardRegs CyU3PSdioCardRegs\_t**  
*SDIO Card Generic Information and CCCR registers.*
- **typedef enum CyU3PSibEraseMode CyU3PSibEraseMode**  
*Erase command mode.*

## Enumerations

- **enum CyU3PSibPortId { CY\_U3P\_SIB\_PORT\_0 = 0, CY\_U3P\_SIB\_PORT\_1, CY\_U3P\_SIB\_NUM\_PORTS }**  
*List of storage ports.*
- **enum CyU3PSibCardDetect { CY\_U3P\_SIB\_DETECT\_NONE = 0, CY\_U3P\_SIB\_DETECT\_GPIO, CY\_U3P\_SIB\_DETECT\_DAT\_3 }**  
*List of card detection methods.*
- **enum CyU3PSibDevType { CY\_U3P\_SIB\_DEV\_NONE = 0, CY\_U3P\_SIB\_DEV\_MMC, CY\_U3P\_SIB\_DEV\_SD, CY\_U3P\_SIB\_DEV\_SDIO, CY\_U3P\_SIB\_DEV\_SDIO\_COMBO }**  
*List the Storage Device types supported by FX3S.*
- **enum CyU3PSibIntfVoltage { CY\_U3P\_SIB\_VOLTAGE\_NORMAL = 0, CY\_U3P\_SIB\_VOLTAGE\_LOW }**  
*List of SD/MMC operating voltage ranges.*
- **enum CyU3PSibEventType { CY\_U3P\_SIB\_EVENT\_INSERT = 0, CY\_U3P\_SIB\_EVENT\_REMOVE, CY\_U3P\_SIB\_EVENT\_XFER\_CPLT, CY\_U3P\_SIB\_EVENT\_SDIO\_INTR, CY\_U3P\_SIB\_EVENT\_RELEASE, CY\_U3P\_SIB\_EVENT\_DATA\_ERROR, CY\_U3P\_SIB\_EVENT\_ABORT }**

- List of storage related events.*
- enum `CyU3PSibLunType` { `CY_U3P_SIB_LUN_RSVD` = 0xAB, `CY_U3P_SIB_LUN_BOOT` = 0xBB, `CY_U3P_SIB_LUN_DATA` = 0xDA }

*List of logical unit (LUN) types.*

  - enum `CyU3PSibDevRegType` { `CY_U3P_SD_REG_OCR` = 0, `CY_U3P_SD_REG_CID`, `CY_U3P_SD_REG_G_CSD` }

*SD register types.*

  - enum `CyU3PSibLunLocation` { `CY_U3P_SIB_LOCATION_USER` = 0, `CY_U3P_SIB_LOCATION_BOOT1`, `CY_U3P_SIB_LOCATION_BOOT2` }

*List of logical unit locations.*

  - enum `CyU3PSibOpFreq` { `CY_U3P_SIB_FREQ_400KHZ`, `CY_U3P_SIB_FREQ_20MHZ`, `CY_U3P_SIB_FREQ_26MHZ`, `CY_U3P_SIB_FREQ_52MHZ`, `CY_U3P_SIB_FREQ_104MHZ` }

*List of storage operating frequencies.*

  - enum `CyU3PSibEraseMode` { `CY_U3P_SIB_ERASE_STANDARD` = 0, `CY_U3P_SIB_ERASE_SECURE`, `CY_U3P_SIB_ERASE_TRIM_STEP1`, `CY_U3P_SIB_ERASE_TRIM_STEP2` }

*Erase command mode.*

## Functions

- `CyU3PReturnStatus_t CyU3PSibSetIntfParams (uint8_t portId, CyU3PSibIntfParams_t *intfParams)`

*Define the interface parameters for a storage port.*
- `CyU3PReturnStatus_t CyU3PSibStart (void)`

*Start the storage driver and try to initialize any storage devices connected.*
- void `CyU3PSibStop (void)`

*Stop the storage driver on the FX3S device.*
- `CyU3PReturnStatus_t CyU3PSibInit (uint8_t portId)`

*Initialize the SD/MMC/SDIO device on the specified port.*
- void `CyU3PSibDelInit (uint8_t portId)`

*De-Initialize the storage device on the specified port.*
- `CyU3PReturnStatus_t CyU3PSibRegisterCbk (CyU3PSibEvtCbk_t sibEvtCbk)`

*Register a function for notification of storage related events.*
- `CyU3PReturnStatus_t CyU3PSibQueryDevice (uint8_t portId, CyU3PSibDevInfo_t *devInfo_p)`

*Query storage device information.*
- `CyU3PReturnStatus_t CyU3PSibQueryUnit (uint8_t portId, uint8_t unitId, CyU3PSibLunInfo_t *unitInfo_p)`

*Query storage partition information.*
- `CyU3PReturnStatus_t CyU3PSibGetCSD (uint8_t portId, uint8_t *csd_buffer)`

*Get the CSD register content for an SD/MMC card connected.*
- `CyU3PReturnStatus_t CyU3PSibGetMMCExtCsd (uint8_t portId, uint8_t *buffer_p)`

*Read the Extended CSD register from an MMC card.*
- `CyU3PReturnStatus_t CyU3PSibSendSwitchCommand (uint8_t portId, uint32_t argument, uint32_t *resp_p)`

*Send a SWITCH command to update the Ext. CSD register on an MMC device.*
- void `CyU3PSibSetBlockLen (uint8_t portId, uint16_t blkLen)`

*Define the block length to be used for storage data transfers.*
- void `CyU3PSibWriteTimerModify (uint32_t newTimerVal)`

*Modify default storage write timer interval for data transfers.*
- `CyU3PReturnStatus_t CyU3PSibReadWriteRequest (CyBool_t isRead, uint8_t portId, uint8_t unitId, uint16_t numBlks, uint32_t blkAddr, uint8_t socket)`

*Initiates a read/write data request.*
- `CyU3PReturnStatus_t CyU3PSibCommitReadWrite (uint8_t portId)`

- **CyU3PReturnStatus\_t CyU3PSibSetWriteCommitSize** (uint8\_t portId, uint32\_t numSectors)
 

*Commits any pending read/write transaction.*
- **CyU3PReturnStatus\_t CyU3PSibAbortRequest** (uint8\_t portId)
 

*Set the sector granularity at which write operations are committed to the SD/MMC storage.*
- **CyU3PReturnStatus\_t CyU3PSibEraseBlocks** (uint8\_t portId, uint16\_t numEraseUnits, uint32\_t startAddr, CyU3PSibEraseMode mode)
 

*Erase blocks on the storage card.*
- **CyU3PReturnStatus\_t CyU3PSibGetCardStatus** (uint8\_t portId, uint32\_t \*status\_p)
 

*Function to get current status of the SD/MMC card.*
- **CyU3PReturnStatus\_t CyU3PSibPartitionStorage** (uint8\_t portId, uint8\_t partCount, uint32\_t \*partSizes\_p, uint8\_t \*partType\_p)
 

*Partition a storage device into multiple logical units.*
- **CyU3PReturnStatus\_t CyU3PSibReadRegister** (uint8\_t portId, CyU3PSibDevRegType regType, uint8\_t \*dataBuffer, uint8\_t \*dataLen)
 

*Read the contents of an internal register on the SD/eMMC device connected.*
- **CyU3PReturnStatus\_t CyU3PSibRemovePartitions** (uint8\_t portId)
 

*Remove all partitions and re-unify a storage device.*
- **CyU3PReturnStatus\_t CyU3PSibLockUnlockCard** (uint8\_t portId, CyBool\_t lockCard, CyBool\_t clrPasswd, uint8\_t \*passwd, uint8\_t passwdLen)
 

*Lock/Unlock an SD Card.*
- **CyU3PReturnStatus\_t CyU3PSibSetPasswd** (uint8\_t portId, CyBool\_t lockCard, uint8\_t \*passwd, uint8\_t passwdLen, uint8\_t \*newPasswd, uint8\_t newPasswdLen)
 

*Set/replace password on an SD Card.*
- **CyU3PReturnStatus\_t CyU3PSibRemovePasswd** (uint8\_t portId, uint8\_t \*passwd, uint8\_t passwdLen)
 

*Clear the password on an SD Card.*
- **CyU3PReturnStatus\_t CyU3PSibForceErase** (uint8\_t portId)
 

*Force erase the user content and the lock/unlock related information on an SD Card.*
- **CyU3PReturnStatus\_t CyU3PSibVendorAccess** (uint8\_t portId, uint8\_t cmd, uint32\_t cmdArg, uint8\_t respLen, uint8\_t \*respData, uint16\_t numBlks, CyBool\_t isRead, uint8\_t socket)
 

*Vendor SD/MMC command access.*
- **CyU3PReturnStatus\_t CyU3PSdioCardReset** (uint8\_t portId)
 

*Reset an SDIO card.*
- **CyU3PReturnStatus\_t CyU3PSdioQueryCard** (uint8\_t portId, CyU3PSdioCardRegs\_t \*data)
 

*Fetch SDIO card information and Card common register information.*
- **CyU3PReturnStatus\_t CyU3PSdioByteReadWrite** (uint8\_t portId, uint8\_t functionNumber, uint8\_t isWrite, uint8\_t readAfterWriteFlag, uint32\_t registerAddr, uint8\_t \*data)
 

*Perform a Direct I/O operation (Single Byte) to an SDIO card using CMD52.*
- **CyU3PReturnStatus\_t CyU3PSdioExtendedReadWrite** (uint8\_t portId, uint8\_t functionNumber, uint8\_t isWrite, uint8\_t blockMode, uint8\_t opCode, uint32\_t registerAddr, uint16\_t transferCount, uint16\_t sckId)
 

*Perform Extended I/O operation (Multi-Byte/Block) to/from SDIO card using CMD53.*
- **CyU3PReturnStatus\_t CyU3PSdioDirectReadWrite** (uint8\_t portId, uint8\_t functionNumber, uint8\_t isWrite, uint8\_t opCode, uint8\_t \*data\_p, uint32\_t registerAddr, uint16\_t transferCount)
 

*Perform a Direct I/O operation (Single Byte) to an SDIO card using CMD52.*
- **CyU3PReturnStatus\_t CyU3PSdioSuspendResumeFunction** (uint8\_t portId, uint8\_t functionNumber, uint8\_t operation, uint8\_t \*isDataAvailable)
 

*Suspend or Resume SDIO I/O Function.*
- **CyU3PReturnStatus\_t CyU3PSdioAbortFunctionIO** (uint8\_t portId, uint8\_t functionNumber)
 

*Abort an ongoing extended read or extended write operation.*
- **CyU3PReturnStatus\_t CyU3PSdioInterruptControl** (uint8\_t portId, uint8\_t functionNumber, uint8\_t operation, uint8\_t \*intEnReg)
 

*Enable or Disable Interrupts on the SDIO Card.*

- `CyU3PReturnStatus_t CyU3PSdioGetCISAddress (uint8_t portId, uint8_t functionNumber, uint32_t *addressCIS)`  
*Get Address for the CIS for the specified card function.*
- `CyU3PReturnStatus_t CyU3PSdioGetTuples (uint8_t portId, uint8_t funcNo, uint8_t tupleId, uint32_t addrCIS, uint8_t *buffer, uint8_t *tupleSize)`  
*Read data for a CIS Tuple into a buffer.*
- `CyU3PReturnStatus_t CyU3PSdioSetBlockSize (uint8_t portId, uint8_t funcNo, uint16_t blockSize)`  
*Set the block size for an IO function.*
- `CyU3PReturnStatus_t CyU3PSdioGetBlockSize (uint8_t portId, uint8_t funcNo, uint16_t *blockSize)`  
*Get block size which has been set for IO function to the device.*
- `CyU3PReturnStatus_t CyU3PSdioReadWaitEnable (uint8_t portId, uint8_t isRdWtEnable)`  
*Enable Read Wait on the SDIO BUS.*

## 5.29.1 Detailed Description

This file defines the data types and APIs that support SD/MMC/SDIO peripheral access on the EZ-USB FX3S device.

### Description

The EZ-USB FX3S device is an extension to the FX3 device, which has the capability to talk to SD/MMC/SDIO peripherals.

The storage module in FX3S firmware manages accesses to SD, MMC and SDIO devices. The storage module is composed of a storage driver and convenience API; and is compiled into a separate static library. The storage library (cyu3sport.a) only needs to be linked in by FX3S applications that make use of the SD/MMC/SDIO interfaces.

### Note

The storage driver internally makes use of GPIO functions. Therefore, applications linking with the storage library will also need to link with the serial peripheral library (cyu3lpp.a).

## 5.29.2 Typedef Documentation

### 5.29.2.1 `typedef struct CyU3PSdioCardRegs CyU3PSdioCardRegs_t`

SDIO Card Generic Information and CCCR registers.

### Description

The following structure stores information about the SDIO card attached to a storage port of the FX3S.

### 5.29.2.2 `typedef enum CyU3PSibCardDetect CyU3PSibCardDetect`

List of card detection methods.

### Description

The card insertion and removal detection can be based on either GPIO card detect or based on signal changes on the DAT3 line.

### 5.29.2.3 `typedef struct CyU3PSibDeviceInfo CyU3PSibDeviceInfo_t`

Storage (SD/MMC) device properties.

### Description

The following structure stores information about the storage device attached to the FX3S's storage ports. Please note that most of these fields are relevant only for SD and MMC devices. SDIO specific query APIs should be used to get the properties of an SDIO device.

See also

[CyU3PSibDevType](#)

#### 5.29.2.4 `typedef enum CyU3PSibDevRegType CyU3PSibDevRegType`

SD register types.

**Description**

List of SD registers that can be read.

#### 5.29.2.5 `typedef enum CyU3PSibDevType CyU3PSibDevType`

List the Storage Device types supported by FX3S.

**Description**

This enumeration lists the various storage device types supported by the storage module in FX3S firmware.

#### 5.29.2.6 `typedef enum CyU3PSibEraseMode CyU3PSibEraseMode`

Erase command mode.

**Description**

This enumeration lists the possible modes for the erase command. These modes are only applicable to eMMC media, and only the standard erase mode is supported for SD cards.

#### 5.29.2.7 `typedef enum CyU3PSibEventType CyU3PSibEventType`

List of storage related events.

**Description**

The storage driver propagates events of interest such as card insertion/removal and data transfer completion through a callback function.

#### 5.29.2.8 `typedef void(* CyU3PSibEvtCbk_t)(uint8_t portId, CyU3PSibEventType evt, CyU3PReturnStatus_t status )`

Storage event callback function type.

**Description**

This function type defines a callback for the storage events as well as for the status of the storage read/write operations.

See also

[CyU3PSibEventType](#)  
[CyU3PSibRegisterCbk](#)

#### 5.29.2.9 `typedef struct CyU3PSibIntfParams CyU3PSibIntfParams_t`

Storage interface control parameters.

**Description**

This structure encapsulates the desired operating conditions for the storage ports.

**5.29.2.10 `typedef enum CyU3PSibIntfVoltage CyU3PSibIntfVoltage`**

List of SD/MMC operating voltage ranges.

**Description**

The storage host driver supports storage functionality at both the normal voltage (3.3V) and low voltage (1.8V). Low voltage is necessary for the use of the SDR50 signaling rate on the SD 3.0 interface.

**5.29.2.11 `typedef struct CyU3PSibLunInfo CyU3PSibLunInfo_t`**

Logical unit properties.

**Description**

The following structure stores information about each logical unit (partition) on the storage devices connected on each storage port. Each boot partition is counted as a separate logical unit. The logical units can be virtual partitions managed by the firmware, in the case where the storage device used does not support native partitions. A maximum of four logical units can be supported on each storage device.

**See also**

[CyU3PSibLunType](#)  
[CyU3PSibLunLocation](#)

**5.29.2.12 `typedef enum CyU3PSibLunLocation CyU3PSibLunLocation`**

List of logical unit locations.

**Description**

MMC devices can support one or more boot partitions in addition to the user data area, and a given storage LUN may be located in the boot partition or in the user data area. This enumerated type lists the possible location of a logical unit.

**5.29.2.13 `typedef enum CyU3PSibLunType CyU3PSibLunType`**

List of logical unit (LUN) types.

**Description**

The storage devices can be partitioned to store boot images as well as for storing user content. The boot partitions are only accessible under boot or boot update modes.

**5.29.2.14 `typedef enum CyU3PSibOpFreq CyU3PSibOpFreq`**

List of storage operating frequencies.

**Description**

This enumerated type lists the various operating frequencies supported on the storage port by the FX3S device. This type is used to set a constraint on the maximum operating frequency for the storage devices.

**5.29.2.15 `typedef enum CyU3PSibPortId CyU3PSibPortId`**

List of storage ports.

**Description**

The FX3S device has two storage ports which support SD/MMC/SDIO cards to be attached. These storage ports can be accessed independently of each other.

### 5.29.3 Enumeration Type Documentation

#### 5.29.3.1 enum CyU3PSibCardDetect

List of card detection methods.

##### Description

The card insertion and removal detection can be based on either GPIO card detect or based on signal changes on the DAT3 line.

##### Enumerator

**CY\_U3P\_SIB\_DETECT\_NONE** Don't use any card detection mechanism

**CY\_U3P\_SIB\_DETECT\_GPIO** Use a GPIO connected to a micro-switch on the socket. This is only supported for the S0 storage port.

**CY\_U3P\_SIB\_DETECT\_DAT\_3** Use voltage changes on the DAT[3] pin for card detection.

#### 5.29.3.2 enum CyU3PSibDevRegType

SD register types.

##### Description

List of SD registers that can be read.

##### Enumerator

**CY\_U3P\_SD\_REG\_OCR** OCR register.

**CY\_U3P\_SD\_REG\_CID** CID register.

**CY\_U3P\_SD\_REG\_CSD** CSD register.

#### 5.29.3.3 enum CyU3PSibDevType

List the Storage Device types supported by FX3S.

##### Description

This enumeration lists the various storage device types supported by the storage module in FX3S firmware.

##### Enumerator

**CY\_U3P\_SIB\_DEV\_NONE** No device

**CY\_U3P\_SIB\_DEV\_MMCC** MMC/eMMC device

**CY\_U3P\_SIB\_DEV\_SD** SD Memory card

**CY\_U3P\_SIB\_DEV\_SDIO** SDIO IO only Device.

**CY\_U3P\_SIB\_DEV\_SDIO\_COMBO** SDIO Combo Device. Data transfers to the combo card are not supported.

#### 5.29.3.4 enum CyU3PSibEraseMode

Erase command mode.

##### Description

This enumeration lists the possible modes for the erase command. These modes are only applicable to eMMC media, and only the standard erase mode is supported for SD cards.

##### Enumerator

**CY\_U3P\_SIB\_ERASE\_STANDARD** Standard erase mode.

**CY\_U3P\_SIB\_ERASE\_SECURE** Secure erase mode. Only supported for eMMC.

**CY\_U3P\_SIB\_ERASE\_TRIM\_STEP1** Secure trim STEP1. Mark the blocks for secure erase.

**CY\_U3P\_SIB\_ERASE\_TRIM\_STEP2** Secure trim STEP2. Erase blocks marked in trim STEP1.

#### 5.29.3.5 enum CyU3PSibEventType

List of storage related events.

##### Description

The storage driver propagates events of interest such as card insertion/removal and data transfer completion through a callback function.

##### Enumerator

**CY\_U3P\_SIB\_EVENT\_INSERT** Card Insert Event

**CY\_U3P\_SIB\_EVENT\_REMOVE** Card Remove Event

**CY\_U3P\_SIB\_EVENT\_XFER\_CPLT** Data transfer completion. The status field will indicate success/failure.

**CY\_U3P\_SIB\_EVENT\_SDIO\_INTR** SDIO interrupt event.

**CY\_U3P\_SIB\_EVENT\_RELEASE** Notification that device is free for access: Not supported.

**CY\_U3P\_SIB\_EVENT\_DATA\_ERROR** Errors like CRC16, Data Timeout and Data Error handler event.

**CY\_U3P\_SIB\_EVENT\_ABORT** Transaction aborted event.

#### 5.29.3.6 enum CyU3PSibIntfVoltage

List of SD/MMC operating voltage ranges.

##### Description

The storage host driver supports storage functionality at both the normal voltage (3.3V) and low voltage (1.8V). Low voltage is necessary for the use of the SDR50 signaling rate on the SD 3.0 interface.

##### Enumerator

**CY\_U3P\_SIB\_VOLTAGE\_NORMAL** Normal voltage, 3.3 V.

**CY\_U3P\_SIB\_VOLTAGE\_LOW** Low voltage, 1.8 V.

#### 5.29.3.7 enum CyU3PSibLunLocation

List of logical unit locations.

##### Description

MMC devices can support one or more boot partitions in addition to the user data area, and a given storage LUN may be located in the boot partition or in the user data area. This enumerated type lists the possible location of a logical unit.

##### Enumerator

**CY\_U3P\_SIB\_LOCATION\_USER** The LUN is located in the user data area.

**CY\_U3P\_SIB\_LOCATION\_BOOT1** The LUN is located in the first boot partition.

**CY\_U3P\_SIB\_LOCATION\_BOOT2** The LUN is located in the second boot partition.

### 5.29.3.8 enum CyU3PSibLunType

List of logical unit (LUN) types.

#### Description

The storage devices can be partitioned to store boot images as well as for storing user content. The boot partitions are only accessible under boot or boot update modes.

#### Enumerator

**CY\_U3P\_SIB\_LUN\_RSVD** LUN type reserved for future use.

**CY\_U3P\_SIB\_LUN\_BOOT** LUN holding FX3S Boot code

**CY\_U3P\_SIB\_LUN\_DATA** LUN holding user data

### 5.29.3.9 enum CyU3PSibOpFreq

List of storage operating frequencies.

#### Description

This enumerated type lists the various operating frequencies supported on the storage port by the FX3S device. This type is used to set a constraint on the maximum operating frequency for the storage devices.

#### Enumerator

**CY\_U3P\_SIB\_FREQ\_400KHZ** 400 KHz. Used for device initialization only. Cannot be set as the maximum frequency.

**CY\_U3P\_SIB\_FREQ\_20MHZ** 20 MHz SDR.

**CY\_U3P\_SIB\_FREQ\_26MHZ** 26 MHz SDR.

**CY\_U3P\_SIB\_FREQ\_52MHZ** 52 MHz SDR.

**CY\_U3P\_SIB\_FREQ\_104MHZ** Max. frequency possible: 104 MHz SDR or 52 MHz DDR.

### 5.29.3.10 enum CyU3PSibPortId

List of storage ports.

#### Description

The FX3S device has two storage ports which support SD/MMC/SDIO cards to be attached. These storage ports can be accessed independently of each other.

#### Enumerator

**CY\_U3P\_SIB\_PORT\_0** Storage Port 0

**CY\_U3P\_SIB\_PORT\_1** Storage Port 1

**CY\_U3P\_SIB\_NUM\_PORTS** Maximum number of storage ports supported

## 5.29.4 Function Documentation

### 5.29.4.1 CyU3PReturnStatus\_t CyU3PSdioAbortFunctionIO ( uint8\_t portId, uint8\_t functionNumber )

Abort an ongoing extended read or extended write operation.

#### Description

This function is used abort an ongoing extended I/O operation.

#### Return value

CY\_U3P\_SUCCESS

CY\_U3P\_ERROR\_BAD\_ARGUMENT

### Parameters

<i>portId</i>	Port ID for card on which I/O abort is to be executed.
<i>functionNumber</i>	SDIO Card Function number on which I/O is to be aborted.

**5.29.4.2 CyU3PReturnStatus\_t CyU3PSdioByteReadWrite ( uint8\_t *portId*, uint8\_t *functionNumber*, uint8\_t *isWrite*, uint8\_t *readAfterWriteFlag*, uint32\_t *registerAddr*, uint8\_t \* *data* )**

Perform a Direct I/O operation (Single Byte) to an SDIO card using CMD52.

### Description

This function is used to read or write a single byte of data from/to a register on the SDIO card. When the read-after-write flag is set, the data from the register is read back after the write has been completed.

### Return value

CY\_U3P\_SUCCESS if the I/O operation completes successfully.  
 CY\_U3P\_ERROR\_TIMEOUT if a timeout occurs during the I/O operation.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED if the device does not support the operation.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT if function number or register address is out of range.  
 CY\_U3P\_ERROR\_CARD\_NOT\_ACTIVE if card has not been initialized or is in Standby or Inactive states.  
 CY\_U3P\_ERROR\_BAD\_CMD\_ARG if the command argument was out of the allowed range for the card.  
 CY\_U3P\_ERROR\_INVALID\_FUNCTION if an invalid function number was requested.  
 CY\_U3P\_ERROR\_UNKNOWN if a general or unknown error occurred during the operation.  
 CY\_U3P\_ERROR\_ILLEGAL\_CMD if the command is not legal for the card state.  
 CY\_U3P\_ERROR\_CRC if the CRC check of the previous command failed.

### Parameters

<i>portId</i>	Port ID for card on which CMD52 operation is to be executed.
<i>functionNumber</i>	SDIO Card Function number (0-7) for the Byte IO operation.
<i>isWrite</i>	CY_U3P_SDIO_WRITE indicates Write a operation. CY_U3P_SDIO_READ indicates a Read operation.
<i>readAfterWriteFlag</i>	CY_U3P_SDIO_READ_AFTER_WRITE indicates request for Read back of register after Write.
<i>registerAddr</i>	Register address to which IO operation is to be performed.
<i>data</i>	In case of Write operation, the byte of data to be written to the card. In case of Read (or Write with readAfterWriteFlag=1) the data read from the SDIO card will be returned through the same parameter.

**5.29.4.3 CyU3PReturnStatus\_t CyU3PSdioCardReset ( uint8\_t *portId* )**

Reset an SDIO card.

### Description

This function resets an I/O only SDIO card (or the I/O portion of a combo card) by writing 1 to the RES bit in the CCCR. This function does not reset the memory portion in case of a combo card.

### Return value

CY\_U3P\_SUCCESS if the operation completes successfully.  
 CY\_U3P\_ERROR\_TIMEOUT if a timeout occurs during the operation.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED if the device on the addressed port is not an SDIO device.  
 CY\_U3P\_ERROR\_CARD\_NOT\_ACTIVE if card has not been initialized or is in Standby or Inactive states.  
 CY\_U3P\_ERROR\_UNKNOWN if a general or unknown error occurred during the operation.  
 CY\_U3P\_ERROR\_ILLEGAL\_CMD if the command is not legal for the card state.  
 CY\_U3P\_ERROR\_CRC if the CRC check of the previous command failed.

**Parameters**

<i>portId</i>	Port ID for card to be Reset.
---------------	-------------------------------

**5.29.4.4 CyU3PReturnStatus\_t CyU3PSdioDirectReadWrite ( uint8\_t portId, uint8\_t functionNumber, uint8\_t isWrite, uint8\_t opCode, uint8\_t \* data\_p, uint32\_t registerAddr, uint16\_t transferCount )**

Perform a Direct I/O operation (Single Byte) to an SDIO card using CMD52.

**Description**

This function is used to read or write less than 16 bytes of data from/to a register on the SDIO card. This API should be called in place of CyU3PSdioExtendedReadWrite () API for data transfer length of less than 16 bytes.

**Return value**

CY\_U3P\_SUCCESS if the I/O operation completes successfully.  
 CY\_U3P\_ERROR\_TIMEOUT if a timeout occurs during the I/O operation.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED if the device does not support the operation.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT if function number or register address is out of range.  
 CY\_U3P\_ERROR\_CARD\_NOT\_ACTIVE if card has not been initialized or is in Standby or Inactive states.  
 CY\_U3P\_ERROR\_BAD\_CMD\_ARG if the command argument was out of the allowed range for the card.  
 CY\_U3P\_ERROR\_INVALID\_FUNCTION if an invalid function number was requested.  
 CY\_U3P\_ERROR\_UNKNOWN if a general or unknown error occurred during the operation.  
 CY\_U3P\_ERROR\_ILLEGAL\_CMD if the command is not legal for the card state.  
 CY\_U3P\_ERROR\_CRC if the CRC check of the previous command failed.

**Parameters**

<i>portId</i>	Port ID for card on which CMD52 operation is to be executed.
<i>functionNumber</i>	SDIO Card Function number (0-7) for the Multi Byte IO operation.
<i>isWrite</i>	CY_U3P_SDIO_WRITE indicates a Write operation. CY_U3P_SDIO_READ indicates a Read operation.
<i>opCode</i>	CY_U3P_SDIO_RW_ADDR_FIXED indicates all I/O to a single address (FIFO access); CY_U3P_SDIO_RW_ADDR_AUTO_INCREMENT indicates I/O to incrementing addresses (RAM like data buffer).
<i>data_p</i>	In case of Write operation, the byte of data to be written to the card. In case of Read (or Write with readAfterWriteFlag=1) the data read from the SDIO card will be returned through the same parameter.
<i>registerAddr</i>	Register address to which I/O operation is to be performed. This will be the starting address in case of auto-increment mode.
<i>transferCount</i>	Number of Bytes to be transferred.

**5.29.4.5 CyU3PReturnStatus\_t CyU3PSdioExtendedReadWrite ( uint8\_t portId, uint8\_t functionNumber, uint8\_t isWrite, uint8\_t blockMode, uint8\_t opCode, uint32\_t registerAddr, uint16\_t transferCount, uint16\_t sckId )**

Perform Extended I/O operation (Multi-Byte/Block) to/from SDIO card using CMD53.

**Description**

This API reads or writes multiple bytes of data from/to the address specified. The length of data to be written needs to be specified as the parameter. The transfer size can be specified in the form of a byte count or a block count. The API provides the option of auto-incrementing the address after writing or reading each data byte/block. The data is read/written to/from the dma channel associated with the socket ID passed in for the operation. This operation should use one of the SIB Sockets as producer (for read) or consumer (for write).

Transferring infinite blocks of data (by setting transferCount to 0 in block mode) is not supported on the FX3S

devices. If transferCount is set to 0, FX3S will transfer 512 bytes of data in Multi-Byte transfer mode, but will throw an error in Multi-Block mode.

The following table shows the number of bytes and blocks transferred based on transferCount values:

transferCount Value	0x000	0x001	0x002	...	0x1FF
Bytes Transferred	512	1	2		511
Blocks Transferred	illegal	1	2		511

This API will fail with error code INVALID\_FUNCTION if data transfer length is less than 16 bytes. CyU3PSdioDirectReadWrite API should be called in place of current API.

#### Return value

CY\_U3P\_SUCCESS if the IO operation completes successfully.  
 CY\_U3P\_ERROR\_TIMEOUT if a timeout occurs during the operation.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED if the device does not support CMD53 operations.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT if the function number or register address is out of range.  
 CY\_U3P\_ERROR\_CARD\_NOT\_ACTIVE if card has not been initialized or is in Standby or Inactive states.  
 CY\_U3P\_ERROR\_BAD\_CMD\_ARG if the command argument was out of the allowed range for the card.  
 CY\_U3P\_ERROR\_INVALID\_FUNCTION if an invalid function number or data transfer length less than 16 byte was requested.  
 CY\_U3P\_ERROR\_UNKNOWN if a general or unknown error occurred during the operation.  
 CY\_U3P\_ERROR\_ILLEGAL\_CMD if the command is not legal for the card state.  
 CY\_U3P\_ERROR\_CRC if the CRC check of the previous command failed.

#### Parameters

<i>portId</i>	Port ID for card on which CMD53 operation is to be executed.
<i>functionNumber</i>	SDIO Card Function number (0-7) for the Extended IO operation.
<i>isWrite</i>	CY_U3P_SDIO_WRITE indicates a Write operation. CY_U3P_SDIO_READ indicates a Read operation.
<i>blockMode</i>	CY_U3P_SDIO_RW_BYTE_MODE indicates multi-byte transfer. CY_U3P_SDIO_RW_BLOCK_MODE indicates multi-block (1 or more) transfer.
<i>opCode</i>	CY_U3P_SDIO_RW_ADDR_FIXED indicates all I/O to a single address (FIFO access); CY_U3P_SDIO_RW_ADDR_AUTO_INCREMENT indicates I/O to incrementing addresses (RAM like data buffer).
<i>registerAddr</i>	Register address to which I/O operation is to be performed. This will be the starting address in case of auto-increment mode.
<i>transferCount</i>	Number of Bytes or Blocks to be transferred.
<i>sckId</i>	DMA Socket to be used to produce/consume data for the I/O operation.

#### 5.29.4.6 CyU3PReturnStatus\_t CyU3PSdioGetBlockSize ( uint8\_t portId, uint8\_t funcNo, uint16\_t \* blockSize )

Get block size which has been set for IO function to the device.

#### Description

This function gets the block size for a function by reading the block size registers for the function. The API also updates the saved transfer block size in the driver which is used in case of extended transfers.

#### Return value

CY\_U3P\_SUCCESS if the IO operation completes successfully.  
 CY\_U3P\_ERROR\_TIMEOUT if a timeout occurs during the operation.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED if the device does not support the operation.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT if the function number is out of range.  
 CY\_U3P\_ERROR\_CARD\_NOT\_ACTIVE if card has not been initialized or is in Standby or Inactive states.

CY\_U3P\_ERROR\_BAD\_CMD\_ARG if the command's argument was out of the allowed range for the card.  
 CY\_U3P\_ERROR\_INVALID\_FUNCTION if an invalid function number was requested.  
 CY\_U3P\_ERROR\_UNKNOWN if a general or unknown error occurred during the operation.  
 CY\_U3P\_ERROR\_ILLEGAL\_CMD if the command is not legal for the card state.  
 CY\_U3P\_ERROR\_CRC if the CRC check of the previous command failed.

#### Parameters

<i>portId</i>	Port number for the card on which operation is to be executed.
<i>funcNo</i>	Function whose block size is to be set.
<i>blockSize</i>	Pointer for buffer through which the block size will be returned.

5.29.4.7 `CyU3PReturnStatus_t CyU3PSdioGetCISAddress ( uint8_t portId, uint8_t functionNumber, uint32_t * addressCIS )`

Get Address for the CIS for the specified card function.

#### Description

This function gets the Card Information Structure base address for the specified function on the SDIO card.

#### Return value

CY\_U3P\_SUCCESS if the operation completes successfully.  
 CY\_U3P\_ERROR\_TIMEOUT if a timeout occurs while reading the CIS address bytes.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED if the device does not support this operation.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT if the function number is out of range.  
 CY\_U3P\_ERROR\_CARD\_NOT\_ACTIVE if card has not been initialized or is in Standby or Inactive states.  
 CY\_U3P\_ERROR\_INVALID\_FUNCTION if an invalid function number was requested.  
 CY\_U3P\_ERROR\_UNKNOWN if a general or unknown error occurred during the operation.  
 CY\_U3P\_ERROR\_ILLEGAL\_CMD if the command is not legal for the card state.  
 CY\_U3P\_ERROR\_CRC if the CRC check of the previous command failed.

#### Parameters

<i>portId</i>	Port number for the card on which operation is to be executed.
<i>functionNumber</i>	Function number whose CIS address is to be fetched.
<i>addressCIS</i>	Value of the Function's 17-Bit CIS Address

5.29.4.8 `CyU3PReturnStatus_t CyU3PSdioGetTuples ( uint8_t portId, uint8_t funcNo, uint8_t tupleId, uint32_t addrCIS, uint8_t * buffer, uint8_t * tupleSize )`

Read data for a CIS Tuple into a buffer.

#### Description

This function reads out a CIS Tuple from the card's CIS area into a buffer. The size of the tuple is returned in the tupleSize parameter. The buffer memory should be initialized before being provided to this call. As per the SDIO specification, no SDIO tuple can have more than 255 bytes of data.

#### Return value

CY\_U3P\_SUCCESS if the operation completes successfully.  
 CY\_U3P\_ERROR\_TIMEOUT if a timeout occurs during the operation.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED if the device does not support this operation.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT if the function number is out of range.  
 CY\_U3P\_ERROR\_CARD\_NOT\_ACTIVE if card has not been initialized or is in Standby or Inactive states.  
 CY\_U3P\_ERROR\_INVALID\_FUNCTION if an invalid function number was requested.  
 CY\_U3P\_ERROR\_UNKNOWN if a general or unknown error occurred during the operation.

CY\_U3P\_ERROR\_ILLEGAL\_CMD if the command is not legal for the card state.  
CY\_U3P\_ERROR\_CRC if the CRC check of the previous command failed.

#### Parameters

<i>portId</i>	Port number for the card on which operation is to be executed.
<i>funcNo</i>	Function whose Tuple Data is to be fetched.
<i>tupleId</i>	TUPLE_CODE for which data is to be fetched.
<i>addrCIS</i>	Address to the Function's CIS.
<i>buffer</i>	Pointer for buffer into which data will be read into.
<i>tupleSize</i>	Size of tuple body as read from the TUPLE_LINK

#### 5.29.4.9 CyU3PReturnStatus\_t CyU3PSdioInterruptControl ( uint8\_t portId, uint8\_t functionNumber, uint8\_t operation, uint8\_t \* intEnReg )

Enable or Disable Interrupts on the SDIO Card.

#### Description

This function is used to enable or disable interrupts on the SDIO card. A request to enable interrupts for an SDIO Function (1-7) will automatically enable the card's Interrupt Master.

#### Return value

CY\_U3P\_SUCCESS if the register update is successful.  
CY\_U3P\_ERROR\_TIMEOUT if a timeout occurs during the register update.  
CY\_U3P\_ERROR\_NOT\_SUPPORTED if the device does not support this operation.  
CY\_U3P\_ERROR\_BAD\_ARGUMENT if the function number is out of range.  
CY\_U3P\_ERROR\_CARD\_NOT\_ACTIVE if card has not been initialized or is in Standby or Inactive states.  
CY\_U3P\_ERROR\_BAD\_CMD\_ARG if the command argument was out of the allowed range for the card.  
CY\_U3P\_ERROR\_INVALID\_FUNCTION if an invalid function number was requested.  
CY\_U3P\_ERROR\_UNKNOWN if a general or unknown error occurred during the operation.  
CY\_U3P\_ERROR\_ILLEGAL\_CMD if the command is not legal for the card state.  
CY\_U3P\_ERROR\_CRC if the CRC check of the previous command failed.

#### See also

[CyU3PSdioCheckInterruptEnableStatus](#)

#### Parameters

<i>portId</i>	Port number for card on which interrupt control operation is to be executed.
<i>functionNumber</i>	Function number (1-7) or CY_U3P_SDIO_INT_MASTER (for card interrupt master), for which interrupt control operation is requested
<i>operation</i>	CY_U3P_SDIO_DISABLE_INT indicates interrupt for function (or global) needs to be disabled. CY_U3P_SDIO_ENABLE_INT indicates interrupt for function (or global) needs to be enabled. CY_U3P_SDIO_CHECK_INT_ENABLE_REG indicates a request to read and return the interrupt enable register from the card's common registers.
<i>intEnReg</i>	Value of card's Interrupt Enable register post execution of enable / disable / check operation. Status for current function can be checked using <a href="#">CyU3PSdioCheckInterruptEnableStatus</a> .

#### 5.29.4.10 CyU3PReturnStatus\_t CyU3PSdioQueryCard ( uint8\_t portId, CyU3PSdioCardRegs\_t \* data )

Fetch SDIO card information and Card common register information.

**Description**

This function returns an CYU3PSdioCardRegs\_t object which contains card common register information from an initialized SDIO card on the port specified by portId.

**Return value**

CY\_U3P\_SUCCESS if the operation completes successfully.

CY\_U3P\_ERROR\_NOT\_SUPPORTED if the device on the addressed port is not an SDIO device.

**See also**

[CyU3PSdioCardRegs\\_t](#)

**Parameters**

<i>portId</i>	Port id on which SDIO card is connected.
<i>data</i>	Output parameter to be filled with SDIO register information.

**5.29.4.11 CyU3PReturnStatus\_t CyU3PSdioReadWaitEnable ( uint8\_t portId, uint8\_t isRdWtEnable )**

Enable Read Wait on the SDIO BUS.

**Description**

This function triggers Read-Wait on card which supports the feature using the DAT[2] of the SDIO Data bus.

**Return value**

CY\_U3P\_SUCCESS if the operation completes successfully.

CY\_U3P\_ERROR\_NOT\_SUPPORTED if the device does not support the operation.

**Parameters**

<i>portId</i>	Port number for the card on which operation is to be executed.
<i>isRdWtEnable</i>	1 to Trigger Read-Wait on the Data Bus, 0 to restart I/O on the Data Bus

**5.29.4.12 CyU3PReturnStatus\_t CyU3PSdioSetBlockSize ( uint8\_t portId, uint8\_t funcNo, uint16\_t blockSize )**

Set the block size for an IO function.

**Description**

This function sets the block size for a function. The block size set should be lower than the maximum block size value read from the CISTPL\_FUNCE CIS tuple for the function. The value set to the card using this API is saved in the driver and is used as the transfer block size in case of extended transfers.

**Return value**

CY\_U3P\_SUCCESS if the IO operation completes successfully.

CY\_U3P\_ERROR\_TIMEOUT if a timeout occurs during the operation.

CY\_U3P\_ERROR\_NOT\_SUPPORTED if the device does not support the operation.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the function number is out of range.

CY\_U3P\_ERROR\_CARD\_NOT\_ACTIVE if card has not been initialized or is in Standby or Inactive states.

CY\_U3P\_ERROR\_BAD\_CMD\_ARG if the command's argument was out of the allowed range for the card.

CY\_U3P\_ERROR\_INVALID\_FUNCTION if an invalid function number was requested.

CY\_U3P\_ERROR\_UNKNOWN if a general or unknown error occurred during the operation.

CY\_U3P\_ERROR\_ILLEGAL\_CMD if the command is not legal for the card state.

CY\_U3P\_ERROR\_CRC if the CRC check of the previous command failed.

**Parameters**

<i>portId</i>	Port number for the card on which operation is to be executed.
<i>funcNo</i>	Function whose block size is to be set.
<i>blockSize</i>	Block size to be set for the function.

5.29.4.13 **CyU3PReturnStatus\_t CyU3PSdioSuspendResumeFunction ( uint8\_t portId, uint8\_t functionNumber, uint8\_t operation, uint8\_t \* isDataAvailable )**

Suspend or Resume SDIO I/O Function.

**Description**

This function is used to suspend or resume the specified SDIO Function. Suspend and resume needs to be supported by the card to which these calls are being addressed. Support for suspend and resume can be ascertained by looking at the CY\_U3P\_SDIO\_CARD\_CAPABILITY\_SBS bit of the card's card-capability register. When the function is resumed, it also checks if any data is available.

**Note**

This function is not supported in the SDK 1.3 Beta release.

**Return value**

CY\_U3P\_SUCCESS

CY\_U3P\_ERROR\_BAD\_ARGUMENT

CY\_U3P\_ERROR\_NOT\_SUPPORTED

CY\_U3P\_ERROR\_RESUME\_FAILED

**Parameters**

<i>portId</i>	Port ID for card on which suspend/resume operation is to be executed.
<i>functionNumber</i>	SDIO Card Function number to be suspended.
<i>operation</i>	CY_U3P_SDIO_SUSPEND indicates a request to Suspend the function. CY_U3P_SDIO_RESUME indicates a request to Resume the function.
<i>isDataAvailable</i>	Non-zero if data is available after resuming the function

5.29.4.14 **CyU3PReturnStatus\_t CyU3PSibAbortRequest ( uint8\_t portId )**

Abort an ongoing transaction.

**Description**

This function is used to abort any ongoing transactions on the specified port. Prior to calling this function, the corresponding DMA channel should be reset.

**Return value**

CY\_U3P\_SUCCESS if the operation completed successfully.

**Parameters**

<i>portId</i>	Port on which the transaction is to be aborted.
---------------	---

5.29.4.15 **CyU3PReturnStatus\_t CyU3PSibCommitReadWrite ( uint8\_t portId )**

Commits any pending read/write transaction.

**Description**

Function to commit a pending read/write operation to the SD/MMC storage; by making use of the STOP\_TRANS $\leftarrow$ MISSION command.

**Return value**

CY\_U3P\_SUCCESS if the operation was committed successfully.  
 CY\_U3P\_ERROR\_NOT\_STARTED if the storage driver has not been started.  
 CY\_U3P\_ERROR\_INVALID\_DEV if there is no device connected to the port.

## See also

[CyU3PSibReadWriteRequest](#)  
[CyU3PSibSetWriteCommitSize](#)

## Parameters

<i>port<math>\leftarrow</math> Id</i>	Port on which the operation is to be committed.
---	---

## 5.29.4.16 void CyU3PSibDelInit( uint8\_t portId )

De-Initialize the storage device on the specified port.

**Description**

Request to selectively de-initialize the devices connected on one of the storage ports.

**Return value**

None

## See also

[CyU3PSibInit](#)  
[CyU3PSibStop](#)

## Parameters

<i>port<math>\leftarrow</math> Id</i>	The storage port which has to be uninitialized.
---	---

## 5.29.4.17 CyU3PReturnStatus\_t CyU3PSibEraseBlocks( uint8\_t portId, uint16\_t numEraseUnits, uint32\_t startAddr, CyU3PSibEraseMode mode )

Erase blocks on the storage card.

**Description**

This function is used to erase the specified number of erase units on an SD card or eMMC device. Please see the SD/eMMC specifications for the definition of an erase unit and its computation.

On an eMMC device, this API can be used to erase the contents of the boot partition(s) as well. Please use the [CyU3PSibSendSwitchCommand\(\)](#) API to select the partition to be erased, prior to calling this API to initiate the erase operation.

**Return value**

CY\_U3P\_SUCCESS - if the operation completed successfully.  
 CY\_U3P\_ERROR\_NOT\_STARTED - if the storage driver has not been started.  
 CY\_U3P\_ERROR\_INVALID\_DEV - if no storage device is found on the specified port.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - if this operation is requested on a device other than SD card.

CY\_U3P\_ERROR\_INVALID\_ADDR - if the address given is not valid.  
 CY\_U3P\_ERROR\_TIMEOUT - if the operation timed out while erasing the blocks.  
 CY\_U3P\_ERROR\_DEVICE\_BUSY - if the device is processing another request.

#### See also

[CyU3PSibEraseMode](#)  
[CyU3PSibSendSwitchCommand](#)

#### Parameters

<i>portId</i>	Port Id on which the erase operation is to be carried out.
<i>numEraseUnits</i>	Number of Erase Units to be erased.
<i>startAddr</i>	Starting address from which the blocks are to be erased.
<i>mode</i>	Type of erase operation to be performed.

#### 5.29.4.18 CyU3PReturnStatus\_t CyU3PSibForceErase ( uint8\_t portId )

Force erase the user content and the lock/unlock related information on an SD Card.

#### Description

This function is used force erase the user content on the SD Card and also clear the current password and unlock the card.

#### Return value

CY\_U3P\_SUCCESS if the operation completed successfully.  
 CY\_U3P\_ERROR\_CARD\_FORCE\_ERASE if the operation failed while force erasing.

#### Parameters

<i>portId</i>	Port on which the force erase is to be done.
---------------	--

#### 5.29.4.19 CyU3PReturnStatus\_t CyU3PSibGetCardStatus ( uint8\_t portId, uint32\_t \* status\_p )

Function to get current status of the SD/MMC card.

#### Description

This function sends the CMD13 (SEND\_STATUS) command to the SD/MMC device connected on the specified storage port and provides the card status response that is received. This API can be used to check whether the storage device is accessible and is in the TRAN state where it can receive new commands.

#### Return value

CY\_U3P\_SUCCESS - if a R1 response was received from the storage device.  
 CY\_U3P\_ERROR\_NOT\_STARTED - if the storage driver has not been started.  
 CY\_U3P\_ERROR\_INVALID\_DEV - if no storage device is found on the specified port.  
 CY\_U3P\_ERROR\_TIMEOUT - if there is a response timeout or the card is signalling busy condition.  
 CY\_U3P\_ERROR\_DEVICE\_BUSY - if the device is processing another request.

#### Parameters

<i>portId</i>	Port id on which the card is connected.
<i>status_p</i>	Return parameter through which the card status is retrieved.

#### 5.29.4.20 CyU3PReturnStatus\_t CyU3PSibGetCSD ( *uint8\_t portId, uint8\_t \* csd\_buffer* )

Get the CSD register content for an SD/MMC card connected.

##### Description

This function is used to access the CSD register data of a card on a particular port. This function only returns the CSD value that is read during device initialization.

##### Return value

CY\_U3P\_SUCCESS if the query function is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED if the storage driver has not been started.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT if the portId or csd\_buffer argument is invalid.  
 CY\_U3P\_ERROR\_INVALID\_DEV if the storage device does not exist.

##### Parameters

<i>portId</i>	Port on which to query the CSD.
<i>csd_buffer</i>	Buffer to be filled with CSD content.

#### 5.29.4.21 CyU3PReturnStatus\_t CyU3PSibGetMMCExtCsd ( *uint8\_t portId, uint8\_t \* buffer\_p* )

Read the Extended CSD register from an MMC card.

##### Description

This function reads the extended CSD register content from an MMC device connected to FX3S.

##### Return value

CY\_U3P\_SUCCESS if the read is successful.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT if the portId or buffer\_p argument is invalid.  
 CY\_U3P\_ERROR\_INVALID\_DEV if the device addressed is not an MMC card.  
 CY\_U3P\_ERROR\_DEVICE\_BUSY if the storage device is performing a data transfer.

##### See also

[CyU3PSibSendSwitchCommand](#)

##### Parameters

<i>portId</i>	Port on which to read the Ext. CSD register.
<i>buffer_p</i>	Buffer to read the register value into. Should be able to hold 512 bytes.

#### 5.29.4.22 CyU3PReturnStatus\_t CyU3PSibInit ( *uint8\_t portId* )

Initialize the SD/MMC/SDIO device on the specified port.

##### Description

This function initializes any storage device connected on the specified port. Also creates the DMA Channels and locks required for the driver to access the device safely.

##### Return value

CY\_U3P\_SUCCESS if the operation completed successfully.  
 CY\_U3P\_ERROR\_SIB\_INIT if the DMA channel creation associated with the storage port fails.

**See also**

[CyU3PSibDeInit](#)  
[CyU3PSibStart](#)

#### 5.29.4.23 CyU3PReturnStatus\_t CyU3PSibLockUnlockCard ( uint8\_t portId, CyBool\_t lockCard, CyBool\_t clrPasswd, uint8\_t \* passwd, uint8\_t passwdLen )

Lock/Unlock an SD Card.

**Description**

This function is used to lock or unlock an SD Card on the specified port.

**Return value**

CY\_U3P\_SUCCESS if the operation completed successfully.

CY\_U3P\_ERROR\_CARD\_LOCK\_UNLOCK if the operation to lock/unlock the card fails.

**Parameters**

<i>portId</i>	Port on which the lock/unlock operation is to be done.
<i>lockCard</i>	Specifies whether to lock (1) or unlock (0) the card.
<i>clrPasswd</i>	Used along with the Unlock card to clear the password.
<i>passwd</i>	The current password set in the card.
<i>passwdLen</i>	The current password length in bytes.

#### 5.29.4.24 CyU3PReturnStatus\_t CyU3PSibPartitionStorage ( uint8\_t portId, uint8\_t partCount, uint32\_t \* partSizes\_p, uint8\_t \* partType\_p )

Partition a storage device into multiple logical units.

**Description**

The available storage on a SD/MMC device connected to FX3S can be partitioned into multiple logical units which can be separately accessed. This function is used to setup the partitions as required. The number of partitions to be created, the sizes for all except the last partition; and the types associated with each partition are specified as parameters. The size of the last partition is inferred as the number of remaining blocks available on the storage device, and does not need to be specified.

As a custom partitioning scheme is used by the firmware, partitions created by FX3S cannot be identified by another SD/MMC host controller. Therefore, this feature should only be used in cases where there is no need for the storage device to be read through another controller.

**Note**

The FX3S boot loader will only be able to boot from a storage partition at the top of the available storage. Therefore, it is not useful to set any of the partitions other than 0, as CY\_U3P\_SIB\_LUN\_BOOT partitions.

**See also**

[CyU3PSibLunInfo\\_t](#)  
[CyU3PSibLunType](#)  
[CyU3PSibQueryUnit](#)  
[CyU3PSibRemovePartitions](#)

**Return value**

CY\_U3P\_SUCCESS if all partitions were created successfully.

CY\_U3P\_ERROR\_NOT\_STARTED if the storage driver has not been started.

CY\_U3P\_ERROR\_INVALID\_DEV if the storage device specified does not exist.

CY\_U3P\_ERROR\_ALREADY\_PARTITIONED if the storage device has already been partitioned.

CY\_U3P\_ERROR\_INVALID\_ADDR if the partition sizes specified cannot be setup.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the partition count or types specified are invalid.  
CY\_U3P\_ERROR\_DEVICE\_BUSY if the device is busy processing another request.

#### Parameters

<i>portId</i>	Storage port to do the partitioning on.
<i>partCount</i>	Total number of partitions required. Should be less than or equal to 4.
<i>partSizes</i> ← _p	Sizes for each of the first (N - 1) partitions. The remaining storage on the device will be assigned to the last partition.
<i>partType</i> ← _p	Type for each partition (boot or data partition).

### 5.29.4.25 CyU3PReturnStatus\_t CyU3PSibQueryDevice ( uint8\_t *portId*, CyU3PSibDeviceInfo\_t \* *devInfo\_p* )

Query storage device information.

#### Description

This function returns the storage device information for the specified port. If the operation completes successfully the structure will be populated with the storage device information.

#### Return value

CY\_U3P\_SUCCESS if the operation completes successfully.  
CY\_U3P\_ERROR\_BAD\_ARGUMENT if the portId or devInfo\_p arguments are bad.  
CY\_U3P\_ERROR\_INVALID\_DEV if the device is not valid.

#### See also

[CyU3PSibDeviceInfo\\_t](#)

#### Parameters

<i>portId</i>	The storage device to be queried.
<i>devInfo</i> ← _p	Pointer to the storage device info structure.

### 5.29.4.26 CyU3PReturnStatus\_t CyU3PSibQueryUnit ( uint8\_t *portId*, uint8\_t *unitId*, CyU3PSibLunInfo\_t \* *unitInfo\_p* )

Query storage partition information.

#### Description

This function returns the storage partition (LUN) information for the specified port and unit number. If the operation completes successfully, the structure will be populated with the storage unit information.

#### Return value

CY\_U3P\_SUCCESS if the operation completes successfully.  
CY\_U3P\_ERROR\_INVALID\_DEV if the device is not valid.  
CY\_U3P\_ERROR\_INVALID\_UNIT if the unit is not valid.

#### See also

[CyU3PSibLunInfo\\_t](#)

#### Parameters

<i>portId</i>	The storage device to be queried.
---------------	-----------------------------------

**Parameters**

<i>unitId</i>	The storage unit to be queried.
<i>unitInfo</i> ← — <i>p</i>	Pointer to the storage unit info structure.

5.29.4.27 **CyU3PReturnStatus\_t CyU3PSibReadRegister ( uint8\_t portId, CyU3PSibDevRegType regType, uint8\_t \* dataBuffer, uint8\_t \* dataLen )**

Read the contents of an internal register on the SD/eMMC device connected.

**Description**

This function returns the contents of an internal register on the SD/eMMC device connected to the FX3S or SD3 device. Since these internal registers can only be read at init time, the API returns the values that would have been read and cached during the init process.

**Return value**

CY\_U3P\_SUCCESS if the read is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED if the storage driver has not been started.  
 CY\_U3P\_ERROR\_INVALID\_DEV if the device specified does not exist.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT if the register specified is invalid.

**See also**

[CyU3PSibDevRegType](#)

5.29.4.28 **CyU3PReturnStatus\_t CyU3PSibReadWriteRequest ( CyBool\_t isRead, uint8\_t portId, uint8\_t unitId, uint16\_t numBlks, uint32\_t blkAddr, uint8\_t socket )**

Initiates a read/write data request.

**Description**

This function is used to initiate a read/write request to a storage device. The open-ended read/write commands (C←MD18 and CMD25) are used in all cases. The CY\_U3P\_SIB\_EVENT\_XFER\_CPLT event will be sent to the caller through the register storage callback function when the specified amount of data has been completely transferred.

This function internally initiates the DMA channel for transferring the specified amount of data. It is therefore expected that the DMA channel is left in the idle state when calling this API.

**Return value**

CY\_U3P\_SUCCESS if the operation completed successfully.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT if the port or unit number is out of range.  
 CY\_U3P\_ERROR\_INVALID\_DEV if the device specified is not present.  
 CY\_U3P\_ERROR\_CARD\_LOCKED if the storage device being accessed is password locked.  
 CY\_U3P\_ERROR\_WRITE\_PROTECTED if the storage device is write protected.  
 CY\_U3P\_ERROR\_INVALID\_UNIT if the LUN specified is not a valid LUN.  
 CY\_U3P\_ERROR\_INVALID\_ADDR if the address specified is not valid.  
 CY\_U3P\_ERROR\_DEVICE\_BUSY if the storage device is already busy processing another request.

**See also**

[CyU3PSibCommitReadWrite](#)  
[CyU3PSibSetWriteCommitSize](#)  
[CyU3PSibWriteTimerModify](#)

**Parameters**

<i>isRead</i>	Read or a write operation
---------------	---------------------------

**Parameters**

<i>portId</i>	Storage port number to access.
<i>unitId</i>	Storage partition (unit) number to access.
<i>numBlks</i>	Number of blocks to be read or written.
<i>blkAddr</i>	Start address of the operation
<i>socket</i>	The SIB socket on the which the operation is to happen.

**5.29.4.29 CyU3PReturnStatus\_t CyU3PSibRegisterCbk ( CyU3PSibEvtCbk\_t sibEvtCbk )**

Register a function for notification of storage related events.

**Description**

This function is used to register a callback function which will be called to provide notification of storage related events.

**Return value**

CY\_U3P\_SUCCESS if the operation completed successfully.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the parameter is not valid.

**See also**

[CyU3PSibEvtCbk\\_t](#)

**Parameters**

<i>sibEvtCbk</i>	The event callback function to be called.
------------------	---

**5.29.4.30 CyU3PReturnStatus\_t CyU3PSibRemovePartitions ( uint8\_t portId )**

Remove all partitions and re-unify a storage device.

**Description**

This function is used to remove all partitions created using the CyU3PSibPartitionStorage API, and re-unify the complete storage available on a device as a single data partition (volume).

**Return value**

CY\_U3P\_SUCCESS if the partitions were successfully removed.

CY\_U3P\_ERROR\_NOT\_STARTED if the storage driver has not been started.

CY\_U3P\_ERROR\_INVALID\_DEV if the device specified does not exist.

CY\_U3P\_ERROR\_NOT\_PARTITIONED if the device has not been partitioned.

CY\_U3P\_ERROR\_DEVICE\_BUSY if the device is busy processing another request.

**See also**

[CyU3PSibPartitionStorage](#)

**Parameters**

<i>portId</i>	Storage port to remove partitions on.
---------------	---------------------------------------

### 5.29.4.31 CyU3PReturnStatus\_t CyU3PSibRemovePasswd ( *uint8\_t portId*, *uint8\_t \* passwd*, *uint8\_t passwdLen* )

Clear the password on an SD Card.

#### Description

This function is used to clear the password on an SD Card on the specified port.

#### Return value

CY\_U3P\_SUCCESS if the operation completed successfully.

CY\_U3P\_ERROR\_CARD\_LOCK\_UNLOCK if the operation failed while removing password.

#### Parameters

<i>portId</i>	Port on which the clear password operation is to be done.
<i>passwd</i>	The current password set in the card.
<i>passwdLen</i>	The current password length in bytes.

### 5.29.4.32 CyU3PReturnStatus\_t CyU3PSibSendSwitchCommand ( *uint8\_t portId*, *uint32\_t argument*, *uint32\_t \* resp\_p* )

Send a SWITCH command to update the Ext. CSD register on an MMC device.

#### Description

This API can be used to send a SWITCH command to update a Ext. CSD register byte on an MMC device. While SD cards support the SWITCH command, their usage model is different; and this API does not support them.

#### Return value

CY\_U3P\_SUCCESS if the switch command was sent successfully.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if an invalid port ID is specified.

CY\_U3P\_ERROR\_INVALID\_DEV if the addressed device is not an MMC device.

CY\_U3P\_ERROR\_DEVICE\_BUSY if the MMC device is busy performing data transfer.

#### See also

[CyU3PSibGetMMCExtCsd](#)

#### Parameters

<i>portId</i>	Port on which to send the Switch command.
<i>argument</i>	Switch command argument. No validation is done on this parameter.
<i>resp_p</i>	Buffer to read the Switch command response into (optional, can be NULL).

### 5.29.4.33 void CyU3PSibSetBlockLen ( *uint8\_t portId*, *uint16\_t blkLen* )

Define the block length to be used for storage data transfers.

#### Description

This function is used to configure the block length to be used for storage data transfers through the FX3S device. Note that this API sets the block length only in the FX3S device registers. If a corresponding SD/MMC command is to be sent to the storage device, that needs to be done separately through the CyU3PSibVendorAccess API.

#### Return value

None

#### See also

[CyU3PSibVendorAccess](#)

**Parameters**

<i>portId</i>	Port id on which to change the block length.
<i>blkLen</i>	Desired block length in bytes.

**5.29.4.34 CyU3PReturnStatus\_t CyU3PSibSetIntfParams ( uint8\_t portId, CyU3PSibIntfParams\_t \* intfParams )**

Define the interface parameters for a storage port.

**Description**

This function is used to define the interface parameters such as low voltage operation, DDR clocking support, card detection and write protection support for one of the FX3S storage ports. This function can be called before the storage driver is started up, so that these parameters are effective when CyU3PSibStart is called.

**Return value**

CY\_U3P\_SUCCESS if the parameters were successfully updated.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the parameters passed are incorrect.

**See also**

[CyU3PSibStart](#)  
[CyU3PSibIntfParams\\_t](#)

**Parameters**

<i>portId</i>	Storage port to be updated.
<i>intfParams</i>	Parameters to be selected for the storage interface.

**5.29.4.35 CyU3PReturnStatus\_t CyU3PSibSetPasswd ( uint8\_t portId, CyBool\_t lockCard, uint8\_t \* passwd, uint8\_t passwdLen, uint8\_t \* newPasswd, uint8\_t newPasswdLen )**

Set/replace password on an SD Card.

**Description**

This function is used to set or replace the password on an SD Card on the specified port. The card can also be locked while setting the password by setting the lockCard variable.

**Return value**

CY\_U3P\_SUCCESS if the operation completed successfully.

CY\_U3P\_ERROR\_CARD\_LOCK\_UNLOCK if setting the password failed.

**Parameters**

<i>portId</i>	Port on which the set/replace password operation is to be done.
<i>lockCard</i>	Specifies whether to lock the card while setting the password.
<i>passwd</i>	The password to be set on the card. In case the password is being replaced this indicates the current password in use.
<i>passwdLen</i>	The length of the password in bytes. In case the password is being replaced this indicates the current password length in bytes.
<i>newPasswd</i>	The new password to be set on the card.
<i>newPasswdLen</i>	The new password length. This should be set to zero in case the password is not being replaced.

### 5.29.4.36 CyU3PReturnStatus\_t CyU3PSibSetWriteCommitSize ( uint8\_t portId, uint32\_t numSectors )

Set the sector granularity at which write operations are committed to the SD/MMC storage.

#### Description

Committing write data to SD/MMC storage in small chunks can lead to poor write transfer performance. The write performance to these devices can be improved by combining write operations to sequential addresses, and performing a single commit (STOP\_TRANSMISSION) operation.

This function is used to set the granularity at which write operations are committed to the storage device. By default, this value is set to 1, meaning that a write of one sector or more will be immediately committed to the storage device.

#### Return value

CY\_U3P\_SUCCESS if the write granularity is set properly.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the portId is invalid or the numSectors value is set to 0.

#### See also

[CyU3PSibReadWriteRequest](#)  
[CyU3PSibCommitReadWrite](#)

#### Parameters

<i>portId</i>	Storage port being configured.
<i>numSectors</i>	Number of sectors after which a write operation should be committed to the SD/MMC storage.

### 5.29.4.37 CyU3PReturnStatus\_t CyU3PSibStart ( void )

Start the storage driver and try to initialize any storage devices connected.

#### Description

This function starts the storage driver on the FX3S device, and initializes any storage devices that are connected on the two storage ports.

#### Return value

CY\_U3P\_SUCCESS if the operation completed successfully.

CY\_U3P\_ERROR\_NOT\_SUPPORTED if the part in use does not support the SD/MMC interface.

CY\_U3P\_ERROR\_NOT\_CONFIGURED if the IO Matrix configuration is not correct for the storage operation.

CY\_U3P\_ERROR\_SIB\_INIT if DMA channel creation associated with the storage port fails.

#### See also

[CyU3PSibStop](#)  
[CyU3PSibInit](#)

### 5.29.4.38 void CyU3PSibStop ( void )

Stop the storage driver on the FX3S device.

#### Description

De-initializes any connected storage devices and stops the storage driver.

#### Return value

None

#### See also

[CyU3PSibStart](#)  
[CyU3PSibDelInit](#)

5.29.4.39 `CyU3PReturnStatus_t CyU3PSibVendorAccess ( uint8_t portId, uint8_t cmd, uint32_t cmdArg, uint8_t respLen, uint8_t * respData, uint16_t numBlks, CyBool_t isRead, uint8_t socket )`

Vendor SD/MMC command access.

#### Description

This function is used to send general SD/MMC commands to storage device/cards attached to FX3S device and receive the corresponding response. Custom commands can be implemented using this function.

The response data will be placed in the output parameter respData along with the CRC7 and end bits. respLen parameter indicates the expected response length (in bits) not including the start, transmit, CRC7 and end bits.

If the command being executed has an associated data phase; the numBlks, socket and pChHandle parameters can be used to manage the data transfer.

#### Return value

`CY_U3P_SUCCESS` if the operation completed successfully.

`CY_U3P_ERROR_BAD_ARGUMENT` if the parameters passed to the function are not valid.

`CY_U3P_ERROR_TIMEOUT` timed out while waiting for the response from the card.

`CY_U3P_ERROR_DEVICE_BUSY` if the device is busy processing another request.

#### See also

[CyU3PSibSetBlockLen](#)

#### Parameters

<code>portId</code>	Storage port on which command is to be executed.
<code>cmd</code>	SD/MMC Commande to be sent
<code>cmdArg</code>	Command argument
<code>respLen</code>	Length of response in bits
<code>respData</code>	Pointer to response data
<code>numBlks</code>	Length of data to be transferred in blocks. Set to 0 if there is no data phase.
<code>isRead</code>	Direction of data transfer, in case there is a data phase.
<code>socket</code>	S-Port socket to be used.

5.29.4.40 `void CyU3PSibWriteTimerModify ( uint32_t newTimerVal )`

Modify default storage write timer interval for data transfers.

#### Description

This function is used to modify the default timer interval i.e. 5 seconds used for storage write data transfers through the FX3S device.

This function should be called every time SIB is initialized, if expecting storage write transactions ranging more than 5 seconds. Storage write timer resets to default interval every time SIB gets de-initialized.

#### Return value

None

#### See also

#### Parameters

<code>newTimerVal</code>	New Timer Interval to be used for current transaction.
--------------------------	--

## 5.30 firmware/u3p\_firmware/inc/cyu3sibpp.h File Reference

This is an FX3S internal header file that defines some global data structures that are used by the storage driver module.

```
#include <cyu3sib.h>
#include <cyu3externcstart.h>
#include <cyu3externcend.h>
```

### Data Structures

- struct [CyU3PSibGlobalData](#)  
*Data structure that holds the storage driver state.*
- struct [CyU3PSibCtxt](#)  
*Structure that holds transfer state corresponding to each storage port.*

### Macros

- #define [CY\\_U3P\\_SIB\\_STACK\\_SIZE](#) (0x800)
- #define [CY\\_U3P\\_SIB\\_THREAD\\_PRIORITY](#) (4)
- #define [CY\\_U3P\\_SIB\\_EVT\\_DRVENTRY](#) (1 << 4)
- #define [CY\\_U3P\\_SIB\\_EVT\\_PORT\\_POS](#) (5)
- #define [CY\\_U3P\\_SIB\\_EVT\\_PORT\\_0](#) (1 << 5)
- #define [CY\\_U3P\\_SIB\\_EVT\\_PORT\\_1](#) (1 << 6)
- #define [CY\\_U3P\\_SIB\\_EVT\\_ABORT](#) (1 << 7)
- #define [CY\\_U3P\\_SIB\\_TIMER0\\_EVT](#) (1 << 15)
- #define [CY\\_U3P\\_SIB\\_TIMER1\\_EVT](#) (1 << 16)
- #define [CY\\_U3P\\_PARTITION\\_TYPE\\_DATA\\_AREA](#) 0xDA
- #define [CY\\_U3P\\_PARTITION\\_TYPE\\_AP\\_BOOT](#) 0xAB
- #define [CY\\_U3P\\_PARTITION\\_TYPE\\_BENICIA\\_BOOT](#) 0xBB
- #define [CY\\_U3P\\_BOOT\\_PARTITION\\_NUM\\_BLKS](#) 0x08
- #define [CyU3PSibEnableCoreIntr](#)(portId)  
*Enable the core interrupt for the specified storage port.*
- #define [CyU3PSibDisableCoreIntr](#)(portId)  
*Disable the core interrupt for the specified storage port.*
- #define [CYU3P\\_SIB\\_INT\\_READ\\_SOCKET](#) ([CY\\_U3P\\_SIB\\_SOCKET\\_6](#))
- #define [CYU3P\\_SIB\\_INT\\_WRITE\\_SOCKET](#) ([CY\\_U3P\\_SIB\\_SOCKET\\_7](#))
- #define [CY\\_U3P\\_DMA\\_SIB\\_NUM\\_SCK](#) (8)  
*Number of storage sockets supported by FX3S.*

### Typedefs

- typedef struct [CyU3PSibGlobalData](#) [CyU3PSibGlobalData](#)  
*Data structure that holds the storage driver state.*
- typedef struct [CyU3PSibCtxt](#) [CyU3PSibCtxt\\_t](#)  
*Structure that holds transfer state corresponding to each storage port.*

## Enumerations

- enum `CyU3PSibDevPartition` {
 `CY_U3P_SIB_DEV_PARTITION_0` = 0, `CY_U3P_SIB_DEV_PARTITION_1`, `CY_U3P_SIB_DEV_PARTITION_2`, `CY_U3P_SIB_DEV_PARTITION_3`,  
`CY_U3P_SIB_NUM_PARTITIONS` }

*Enumeration identifying partitions on storage peripherals.*

- enum `CyU3PSibSocketId_t` {
 `CY_U3P_SIB_SOCKET0` = 0x0200, `CY_U3P_SIB_SOCKET1`, `CY_U3P_SIB_SOCKET2`, `CY_U3P_SIB_SOCKET3`,  
`CY_U3P_SIB_SOCKET4`, `CY_U3P_SIB_SOCKET5`, `CY_U3P_SIB_SOCKET6`, `CY_U3P_SIB_SOCKET7` }

*List of storage interface DMA sockets.*

## Functions

- void `CyU3PSibUpdateLunInfo` (`uint8_t portId`, `uint8_t partNum`, `uint8_t partType`, `uint8_t location`, `uint32_t startAddr`, `uint32_t partSize`)  
*Register information about a storage partition.*
- `CyU3PReturnStatus_t CyU3PSibInitCard` (`uint8_t portId`)  
*Initialize the storage device connected on the specified storage port.*

## Variables

- `CyU3PSibGlobalData glSibDevInfo`
- `CyU3PThread glSibThread`
- `CyU3PEvent glSibEvent`
- `CyU3PSibCtxt_t glSibCtxt [CY_U3P_SIB_NUM_PORTS]`
- `CyU3PSibIntfParams_t glSibIntfParams [CY_U3P_SIB_NUM_PORTS]`
- `CyU3PSibLunInfo_t glSibLunInfo [CY_U3P_SIB_NUM_PORTS][CY_U3P_SIB_NUM_PARTITIONS]`
- `CyU3PSibEvtCbk_t glSibEvtCbk`

### 5.30.1 Detailed Description

This is an FX3S internal header file that defines some global data structures that are used by the storage driver module.

### 5.30.2 Macro Definition Documentation

#### 5.30.2.1 `#define CY_U3P_BOOT_PARTITION_NUM_BLKS 0x08`

Number of blocks reserved for partition header.

#### 5.30.2.2 `#define CY_U3P_PARTITION_TYPE_AP_BOOT 0xAB`

Partition type definition for external processor boot.

#### 5.30.2.3 `#define CY_U3P_PARTITION_TYPE_BENICIA_BOOT 0xBB`

Partition type definition for FX3S boot.

5.30.2.4 `#define CY_U3P_PARTITION_TYPE_DATA_AREA 0xDA`

Partition type definition for user data area.

5.30.2.5 `#define CY_U3P_SIB_EVT_ABORT (1 << 7)`

Event notifying an abort request.

5.30.2.6 `#define CY_U3P_SIB_EVT_DRVENTRY (1 << 4)`

Event id used to signal thread to start executing.

5.30.2.7 `#define CY_U3P_SIB_EVT_PORT_0 (1 << 5)`

Event notifying an interrupt on port 0.

5.30.2.8 `#define CY_U3P_SIB_EVT_PORT_1 (1 << 6)`

Event notifying an interrupt on port 1.

5.30.2.9 `#define CY_U3P_SIB_EVT_PORT_POS (5)`

Position of port specified event bit.

5.30.2.10 `#define CY_U3P_SIB_STACK_SIZE (0x800)`

Size of stack used by driver thread: 2 KB.

5.30.2.11 `#define CY_U3P_SIB_THREAD_PRIORITY (4)`

Priority of driver thread.

5.30.2.12 `#define CY_U3P_SIB_TIMER0_EVT (1 << 15)`

Event notifying a write timeout on port 0.

5.30.2.13 `#define CY_U3P_SIB_TIMER1_EVT (1 << 16)`

Event notifying a write timeout on port 1.

5.30.2.14 `#define CYU3P_SIB_INT_READ_SOCKET (CY_U3P_SIB_SOCKET_6)`

Socket used by driver to read from devices.

5.30.2.15 `#define CYU3P_SIB_INT_WRITE_SOCKET (CY_U3P_SIB_SOCKET_7)`

Socket used by driver to write to devices.

5.30.2.16 #define CyU3PSibDisableCoreIntr( *portId* )

**Value:**

```
{\
    if ((portId) == CY_U3P_SIB_PORT_0) \
    { \
        CyU3PVicDisableInt (CY_U3P_VIC_SIB0_CORE_VECTOR); \
    } \
    else \
    { \
        CyU3PVicDisableInt (CY_U3P_VIC_SIB1_CORE_VECTOR); \
    } \
}
```

Disable the core interrupt for the specified storage port.

5.30.2.17 #define CyU3PSibEnableCoreIntr( *portId* )

**Value:**

```
{\
    if ((portId) == CY_U3P_SIB_PORT_0) \
    { \
        CyU3PVicEnableInt (CY_U3P_VIC_SIB0_CORE_VECTOR); \
    } \
    else \
    { \
        CyU3PVicEnableInt (CY_U3P_VIC_SIB1_CORE_VECTOR); \
    } \
}
```

Enable the core interrupt for the specified storage port.

### 5.30.3 Typedef Documentation

5.30.3.1 **typedef struct CyU3PSibGlobalData CyU3PSibGlobalData**

Data structure that holds the storage driver state.

#### Description

This type defines a global data structure that holds all state information corresponding to the FX3S storage driver.

### 5.30.4 Enumeration Type Documentation

5.30.4.1 **enum CyU3PSibDevPartition**

Enumeration identifying partitions on storage peripherals.

#### Enumerator

<b>CY_U3P_SIB_DEV_PARTITION_0</b>	Device partition 0
<b>CY_U3P_SIB_DEV_PARTITION_1</b>	Device partition 1
<b>CY_U3P_SIB_DEV_PARTITION_2</b>	Device partition 2
<b>CY_U3P_SIB_DEV_PARTITION_3</b>	Device partition 3
<b>CY_U3P_SIB_NUM_PARTITIONS</b>	Maximum partitions allowed

### 5.30.4.2 enum CyU3PSibSocketId\_t

List of storage interface DMA sockets.

Enumerator

- CY\_U3P\_SIB\_SOCKET0** SIB socket #0.
- CY\_U3P\_SIB\_SOCKET1** SIB socket #1.
- CY\_U3P\_SIB\_SOCKET2** SIB socket #2.
- CY\_U3P\_SIB\_SOCKET3** SIB socket #3.
- CY\_U3P\_SIB\_SOCKET4** SIB socket #4.
- CY\_U3P\_SIB\_SOCKET5** SIB socket #5.
- CY\_U3P\_SIB\_SOCKET\_6** S-port socket number 6: Reserved for driver usage.
- CY\_U3P\_SIB\_SOCKET\_7** S-port socket number 7: Reserved for driver usage.

## 5.30.5 Function Documentation

### 5.30.5.1 CyU3PReturnStatus\_t CyU3PSibInitCard ( uint8\_t portId )

Initialize the storage device connected on the specified storage port.

#### Description

Initialize the storage device connected on the specified storage port.

#### Parameters

<i>portId</i>	Port on which the device should be initialized.
---------------	---

### 5.30.5.2 void CyU3PSibUpdateLunInfo ( uint8\_t portId, uint8\_t partNum, uint8\_t partType, uint8\_t location, uint32\_t startAddr, uint32\_t partSize )

Register information about a storage partition.

#### Description

This function stores information about a storage partition in the driver data structures.

#### Parameters

<i>portId</i>	Port on which storage device is connected.
<i>partNum</i>	Partition number which is being updated.
<i>partType</i>	Type of storage partition.
<i>location</i>	Location of this storage partition (boot area or data area).
<i>startAddr</i>	Start address for this storage partition.
<i>partSize</i>	Size of the partition in sectors.

## 5.30.6 Variable Documentation

### 5.30.6.1 CyU3PSibCtxt\_t glSibCtxt[CY\_U3P\_SIB\_NUM\_PORTS]

Storage device/port information.

### 5.30.6.2 CyU3PSibGlobalData gISibDevInfo

Storage driver status.

### 5.30.6.3 CyU3PEvent gISibEvent

Event used to signal driver.

### 5.30.6.4 CyU3PSibEvtCbk\_t gISibEvtCbk

SIB Event Call back function

### 5.30.6.5 CyU3PSibIntfParams\_t gISibIntfParams[CY\_U3P\_SIB\_NUM\_PORTS]

Per-port interface parameters.

### 5.30.6.6 CyU3PSibLunInfo\_t gISibLunInfo[CY\_U3P\_SIB\_NUM\_PORTS][CY\_U3P\_SIB\_NUM\_PARTITIONS]

Partition info.

### 5.30.6.7 CyU3PThread gISibThread

Storage driver thread

## 5.31 firmware/u3p\_firmware/inc/cyu3socket.h File Reference

All block based data transfers IN or OUT of the FX3 device are performed through hardware blocks called sockets. An end-to-end data flow through the FX3 device (DMA channel) is created by tying two or more sockets together using a shared set of data buffers for intermediate storage. This file provides a set of low level APIs that operate on these sockets, and allow custom data pipes to be created.

```
#include "cyu3types.h"
#include "cyu3descriptor.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

## Data Structures

- struct [CyU3PDmaSocket\\_t](#)  
*DMA socket register structure.*
- struct [CyU3PDmaSocketConfig\\_t](#)  
*DMA socket configuration structure.*

## Macros

- #define [CyU3PDmaGetSckNum\(sckId\)](#) ((sckId) & CY\_U3P\_DMA\_SCK\_MASK)  
*Get the socket number field from the socket ID.*
- #define [CyU3PDmaGetIpNum\(sckId\)](#) (((sckId) >> CY\_U3P\_IP\_BLOCK\_POS) & CY\_U3P\_IP\_BLOCK\_MASK)

- `#define CyU3PDmaGetSckId(ipNum, sckNum)`  
*Get the socket ID from the socket number and the ip number.*

## Typedefs

- `typedef enum CyU3PBlockId_t CyU3PBlockId_t`  
*DMA IP block IDs.*
- `typedef struct CyU3PDmaSocket_t CyU3PDmaSocket_t`  
*DMA socket register structure.*
- `typedef struct CyU3PDmaSocketConfig_t CyU3PDmaSocketConfig_t`  
*DMA socket configuration structure.*
- `typedef void(* CyU3PDmaSocketCallback_t) (uint16_t sckId, uint32_t status)`  
*DMA socket interrupt handler callback function.*

## Enumerations

- `enum CyU3PBlockId_t { CY_U3P_LPP_IP_BLOCK_ID = 0, CY_U3P_PIB_IP_BLOCK_ID = 1, CY_U3P_SIB_IP_BLOCK_ID = 2, CY_U3P_UIB_IP_BLOCK_ID = 3, CY_U3P_UIBIN_IP_BLOCK_ID = 4, CY_U3P_NUM_IP_BLOCK_ID = 5, CY_U3P_CPU_IP_BLOCK_ID = 0x3F }`  
*DMA IP block IDs.*

## Functions

- `CyBool_t CyU3PDmaSocketIsValid (uint16_t sckId)`  
*Validates the socket ID.*
- `CyBool_t CyU3PDmaSocketIsValidProducer (uint16_t sckId)`  
*Check whether the specified socket can be used as a producer.*
- `CyBool_t CyU3PDmaSocketIsValidConsumer (uint16_t sckId)`  
*Check whether the specified socket can be used as a consumer.*
- `Cy3PReturnStatus_t CyU3PDmaSocketSetConfig (uint16_t sckId, CyU3PDmaSocketConfig_t *sck_p)`  
*Sets the socket configuration.*
- `Cy3PReturnStatus_t CyU3PDmaSocketClearInterrupts (uint16_t sckId, uint32_t intVal)`  
*Clears interrupt status on a socket.*
- `Cy3PReturnStatus_t CyU3PDmaSocketGetConfig (uint16_t sckId, CyU3PDmaSocketConfig_t *sck_p)`  
*Fetch the current socket configuration.*
- `void CyU3PDmaSocketSetWrapUp (uint16_t sckId)`  
*Request a socket to wrap up.*
- `void CyU3PDmaSocketDisable (uint16_t sckId)`  
*Disable the selected socket.*
- `void CyU3PDmaSocketEnable (uint16_t sckId)`  
*Enable the selected socket.*
- `void CyU3PDmaUpdateSocketSuspendOption (uint16_t sckId, uint16_t suspendOption)`  
*Update the options for the socket suspend.*
- `void CyU3PDmaUpdateSocketResume (uint16_t sckId, uint16_t suspendOption)`  
*Resume a socket from the suspended state.*
- `void CyU3PDmaSocketSendEvent (uint16_t sckId, uint16_t dsrIndex, CyBool_t isOccupied)`  
*Send an event to the socket.*
- `void CyU3PDmaSocketRegisterCallback (CyU3PDmaSocketCallback_t cb)`  
*Register a callback handler for custom socket interrupts.*

### 5.31.1 Detailed Description

All block based data transfers IN or OUT of the FX3 device are performed through hardware blocks called sockets. An end-to-end data flow through the FX3 device (DMA channel) is created by tying two or more sockets together using a shared set of data buffers for intermediate storage. This file provides a set of low level APIs that operate on these sockets, and allow custom data pipes to be created.

#### Note

These functions are primarily used by the FX3 library to implement the DMA channel APIs. It is recommended that the channel APIs be used instead of directly calling these functions.

### 5.31.2 Macro Definition Documentation

#### 5.31.2.1 `#define CyU3PDmaGetSckId( ipNum, sckNum )`

##### **Value:**

```
(( (sckNum) & CY_U3P_DMA_SCK_MASK) | \
(( (ipNum) & CY_U3P_IP_BLOCK_MASK) << CY_U3P_IP_BLOCK_POS))
```

Get the socket ID from the socket number and the ip number.

### 5.31.3 Typedef Documentation

#### 5.31.3.1 `typedef enum CyU3PBlockId_t CyU3PBlockId_t`

DMA IP block IDs.

##### **Description**

The distributed DMA fabric on the FX3 device identifies sockets based on a two part address. The first part identifies the IP block which implements the socket, and the second part identifies the specific socket within the IP block.

This type lists the various IP blocks on the FX3/FX3S devices that provide DMA sockets.

##### **See also**

[CyU3PDmaSocketId\\_t](#)

#### 5.31.3.2 `typedef struct CyU3PDmaSocket_t CyU3PDmaSocket_t`

DMA socket register structure.

##### **Description**

Each hardware block on the FX3 device implements a number of DMA sockets through which it handles data transfers with the external world. Each DMA socket serves as an endpoint for an independent data stream going through the hardware block.

Each socket has a set of registers associated with it, that reflect the configuration and status information for that socket. The CyU3PDmaSocket structure is a replica of the config/status registers for a socket and is designed to perform socket configuration and status checks directly from firmware.

See the `sock_regs.h` header file for the definitions of the fields that make up each of these registers.

##### **See also**

[CyU3PDmaSocketConfig\\_t](#)

5.31.3.3 `typedef void(* CyU3PDmaSocketCallback_t)(uint16_t sckId, uint32_t status )`

DMA socket interrupt handler callback function.

#### Description

When the DMA channel APIs are used for data transfer, all socket specific interrupts are handled by the DMA driver and reported to the user through the DMA channel callbacks.

If a particular socket is being controlled directly using the socket APIs in this file, the driver will be unable to determine how the socket interrupts should be handled. These interrupts are reported to the user application for handling through a callback function of this type.

5.31.3.4 `typedef struct CyU3PDmaSocketConfig_t CyU3PDmaSocketConfig_t`

DMA socket configuration structure.

#### Description

This structure holds all the configuration fields that can be directly updated on a DMA socket. Refer to the `sock_regs.h` file for the detailed break up into bit-fields for each of the structure members.

See also

[CyU3PDmaSocket\\_t](#)  
[CyU3PDmaSocketSetConfig](#)  
[CyU3PDmaSocketGetConfig](#)

## 5.31.4 Enumeration Type Documentation

5.31.4.1 `enum CyU3PBlockId_t`

DMA IP block IDs.

#### Description

The distributed DMA fabric on the FX3 device identifies sockets based on a two part address. The first part identifies the IP block which implements the socket, and the second part identifies the specific socket within the IP block.

This type lists the various IP blocks on the FX3/FX3S devices that provide DMA sockets.

See also

[CyU3PDmaSocketId\\_t](#)

Enumerator

**`CY_U3P_LPP_IP_BLOCK_ID`** LPP block contains UART, I2C, I2S and SPI.

**`CY_U3P_PIB_IP_BLOCK_ID`** P-port interface (GPIF) block.

**`CY_U3P_SIB_IP_BLOCK_ID`** Storage interface block (FX3S only).

**`CY_U3P_UIB_IP_BLOCK_ID`** USB interface block for egress sockets.

**`CY_U3P_UIBIN_IP_BLOCK_ID`** USB interface block for ingress sockets.

**`CY_U3P_NUM_IP_BLOCK_ID`** Sentinel value. Count of valid DMA IPs.

**`CY_U3P_CPU_IP_BLOCK_ID`** Special value used to denote CPU as DMA endpoint.

## 5.31.5 Function Documentation

5.31.5.1 `CyU3PReturnStatus_t CyU3PDmaSocketClearInterrupts( uint16_t sckId, uint32_t intVal )`

Clears interrupt status on a socket.

**Description**

This function clears the interrupt status on a socket without affecting any of the other status fields.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the socket specified is not valid.

**Parameters**

<i>sck</i> <i>Id</i>	Socket id whose interrupt status is to be updated.
<i>intVal</i>	Interrupt status to be cleared.

**5.31.5.2 void CyU3PDmaSocketDisable ( uint16\_t sckId )**

Disable the selected socket.

**Description**

This function disables the specified socket, and returns only after it has been disabled. If the socket is forcibly disabled during data transfer, there will be loss of data.

**Return value**

None

**See also**

[CyU3PDmaSocketEnable](#)

**Parameters**

<i>sck</i> <i>Id</i>	Id of socket to be disabled.
-------------------------	------------------------------

**5.31.5.3 void CyU3PDmaSocketEnable ( uint16\_t sckId )**

Enable the selected socket.

**Description**

This function enables the specified DMA socket. This should be called only after the configuring the socket with valid values using the CyU3PDmaSocketSetConfig API.

**Return value**

None

**See also**

[CyU3PDmaSocketDisable](#)  
[CyU3PDmaSocketSetConfig](#)

**Parameters**

<i>sck</i> <i>Id</i>	Id of socket to be enabled.
-------------------------	-----------------------------

### 5.31.5.4 CyU3PReturnStatus\_t CyU3PDmaSocketGetConfig ( uint16\_t sckId, CyU3PDmaSocketConfig\_t \* sck\_p )

Fetch the current socket configuration.

#### Description

Fetch the current configuration and status of a DMA socket.

#### Return value

CY\_U3P\_SUCCESS - if the function call is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the sck\_p pointer in NULL.

#### See also

[CyU3PDmaSocketSetConfig](#)

#### Parameters

<i>sckId</i>	Socket id whose configuration is to be retrieved.
<i>sck_p</i>	Output parameter to be filled with the socket configuration.

### 5.31.5.5 CyBool\_t CyU3PDmaSocketIsValid ( uint16\_t sckId )

Validates the socket ID.

#### Description

The function validates if the given socket id exists on the device. CPU sockets are virtual sockets and so the function considers them as invalid. The function considers a socket invalid if the IP block that implements it has not been started.

#### Return value

CyTrue if the socket is valid

CyFalse if the socket is invalid.

#### See also

[CyU3PDmaSocketIsValidProducer](#)  
[CyU3PDmaSocketIsValidConsumer](#)

#### Parameters

<i>sckId</i>	Socket id to be validated.
--------------	----------------------------

### 5.31.5.6 CyBool\_t CyU3PDmaSocketIsValidConsumer ( uint16\_t sckId )

Check whether the specified socket can be used as a consumer.

#### Description

Some of the DMA sockets on the FX3 device can only be used as producers or ingress sockets. This function checks whether the specified socket is valid and can be used as a consumer.

#### Return value

CyTrue if the socket is a valid consumer.

CyFalse if the socket is invalid or is ingress-only.

See also

[CyU3PDmaSocketIsValid](#)  
[CyU3PDmaSocketIsValidProducer](#)

Parameters

<i>sck</i> ↪ <i>Id</i>	Socket id to be validated.
---------------------------	----------------------------

### 5.31.5.7 CyBool\_t CyU3PDmaSocketIsValidProducer ( uint16\_t *sckId* )

Check whether the specified socket can be used as a producer.

**Description**

Some of the DMA sockets on the FX3 device can only be used as consumers or egress sockets. This function checks whether the specified socket is valid and can be used as a producer.

**Return value**

CyTrue if the socket is a valid producer.

CyFalse if the socket is invalid or if it is egress-only.

See also

[CyU3PDmaSocketIsValid](#)  
[CyU3PDmaSocketIsValidConsumer](#)

Parameters

<i>sck</i> ↪ <i>Id</i>	Socket id to be validated.
---------------------------	----------------------------

### 5.31.5.8 void CyU3PDmaSocketRegisterCallback ( CyU3PDmaSocketCallback\_t *cb* )

Register a callback handler for custom socket interrupts.

**Description**

This function registers a callback function that will be called to notify the user about socket interrupts. These are only used on sockets that are not associated with any DMA channels.

**Return value**

None

See also

[CyU3PDmaSocketCallback\\_t](#)

Parameters

<i>cb</i>	Callback function pointer. Can be zero to unregister previous callback.
-----------	---

### 5.31.5.9 void CyU3PDmaSocketSendEvent ( uint16\_t *sckId*, uint16\_t *dsrIndex*, CyBool\_t *isOccupied* )

Send an event to the socket.

**Description**

Send a produce / consume event to the selected socket. This API is used to synchronize sockets on manual DMA channels.

**Return value**

None

**See also**

[CyU3PDmaSocketResume](#)

Updates th36 8.9663 Tf -444.459 -24.6591whoseconual DMA

**Parameters**

<i>sckId</i>	Id of socket that should receive the event.
<i>dscriIndex</i>	The descriptor index to associate with the event.
<i>isOccupied</i>	Status of the buffer associated with the event.

**5.31.5.10 CyU3PReturnStatus\_t CyU3PDmaSocketSetConfig ( uint16\_t sckId, CyU3PDmaSocketConfig\_t \* sck\_p )**

Sets the socket configuration.

**Description**

Updates the configuration for a DMA socket based on the values in the structure provided.

**Return value**

CY\_U3P\_SUCCESS - if the function call is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the sck\_p pointer is NULL.

**See also**

[CyU3PDmaSocketConfig\\_t](#)

[CyU3PDmaSocketGetConfig](#)

**Parameters**

<i>sckId</i>	Socket id whose configuration is to be updated.
--------------	---

## See also

[CyU3PDmaSocketEnable](#)  
[CyU3PDmaSocketDisable](#)

## Parameters

<i>sckId</i>	Id of socket to be wrapped up.
<i>Id</i>	

5.31.5.12 void CyU3PDmaUpdateSocketResume ( *uint16\_t sckId, uint16\_t suspendOption* )

Resume a socket from the suspended state.

**Description**

This function clears all active suspend conditions on the selected socket and resumes its operation.

**Return value**

None

## See also

[CyU3PDmaUpdateSocketSuspendOption](#)

## Parameters

<i>sckId</i>	Id of socket to set the option.
<i>suspendOption</i>	Active suspend options. This should match the suspend options as set by the CyU3PDmaUpdateSocketSuspendOption API.

5.31.5.13 void CyU3PDmaUpdateSocketSuspendOption ( *uint16\_t sckId, uint16\_t suspendOption* )

Update the options for the socket suspend.

**Description**

This API updates the suspend options for the selected socket. This does not actually suspend the socket, and the socket suspension will only happen when the pre-conditions for the selected suspend option have been satisfied.

**Return value**

None

## See also

[CyU3PDmaUpdateSocketResume](#)

## Parameters

<i>sckId</i>	Id of socket to set the option.
<i>suspendOption</i>	Suspend option.

## 5.32 firmware/u3p\_firmware/inc/cyu3spi.h File Reference

SPI (Serial Peripheral Interface) is a serial interface defined for inter-device communication. The FX3 device includes a SPI master that can connect to a variety of SPI slave devices and function at various speeds and modes. The SPI driver module provides functions to configure the SPI interface parameters and to perform data read/writes from/to the slave devices.

```
#include "cyu3types.h"
#include "cyu3lpp.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

### Data Structures

- struct [CyU3PSpiConfig\\_t](#)

*Structure defining the configuration of SPI interface.*

### Macros

- #define [CY\\_U3P\\_SPI\\_DEFAULT\\_LOCK\\_TIMEOUT](#) (CYU3P\_WAIT\_FOREVER)

*Delay duration to wait to get a lock on the SPI block.*

### Typedefs

- typedef enum [CyU3PSpiEvt\\_t](#) CyU3PSpiEvt\_t

*List of SPI related event types.*

- typedef enum [CyU3PSpiError\\_t](#) CyU3PSpiError\_t

*List of SPI specific error / status codes.*

- typedef enum [CyU3PSpiSsnLagLead\\_t](#) CyU3PSpiSsnLagLead\_t

*Enumeration defining the lead and lag times of SSN with respect to SCK.*

- typedef enum [CyU3PSpiSsnCtrl\\_t](#) CyU3PSpiSsnCtrl\_t

*Enumeration defining the various ways in which the SSN for a SPI device can be controlled.*

- typedef struct [CyU3PSpiConfig\\_t](#) CyU3PSpiConfig\_t

*Structure defining the configuration of SPI interface.*

- typedef void(\*) [CyU3PSpiIntrCb\\_t](#)) ([CyU3PSpiEvt\\_t](#) evt, [CyU3PSpiError\\_t](#) error)

*Prototype of SPI event callback function.*

### Enumerations

- enum [CyU3PSpiEvt\\_t](#){ [CY\\_U3P\\_SPI\\_EVENT\\_RX\\_DONE](#) = 0, [CY\\_U3P\\_SPI\\_EVENT\\_TX\\_DONE](#), [CY\\_U3P\\_SPI\\_EVENT\\_ERROR](#) }

*List of SPI related event types.*

- enum [CyU3PSpiError\\_t](#){ [CY\\_U3P\\_SPI\\_ERROR\\_TX\\_OVERFLOW](#) = 12, [CY\\_U3P\\_SPI\\_ERROR\\_RX\\_UNDERFLOW](#) = 13, [CY\\_U3P\\_SPI\\_ERROR\\_NONE](#) = 15 }

*List of SPI specific error / status codes.*

- enum [CyU3PSpiSsnLagLead\\_t](#){

```
CY\_U3P\_SPI\_SSN\_LAG\_LEAD\_ZERO\_CLK = 0, CY\_U3P\_SPI\_SSN\_LAG\_LEAD\_HALF\_CLK, CY\_U3P\_SPI\_SSN\_LAG\_LEAD\_ONE\_CLK, CY\_U3P\_SPI\_SSN\_LAG\_LEAD\_ONE\_HALF\_CLK,  
CY\_U3P\_SPI\_NUM\_SSN\_LAG LEAD }
```

*Enumeration defining the lead and lag times of SSN with respect to SCK.*

- enum `CyU3PSpiSsnCtrl_t` {
 `CY_U3P_SPI_SSN_CTRL_FW` = 0, `CY_U3P_SPI_SSN_CTRL_HW_END_OF_XFER`, `CY_U3P_SPI_SS_N_CTRL_HW_EACH_WORD`, `CY_U3P_SPI_SSN_CTRL_HW_CPHA_BASED`,  
`CY_U3P_SPI_SSN_CTRL_NONE`, `CY_U3P_SPI_NUM_SSN_CTRL` }

*Enumeration defining the various ways in which the SSN for a SPI device can be controlled.*

## Functions

- `CyU3PReturnStatus_t CyU3PSpiInit (void)`  
*Starts the SPI interface block on the device.*
- `CyU3PReturnStatus_t CyU3PSpiDeInit (void)`  
*Stops the SPI module.*
- `CyU3PReturnStatus_t CyU3PSpiSetConfig (CyU3PSpiConfig_t *config, CyU3PSpiIntrCb_t cb)`  
*Configures the SPI interface on the FX3 device.*
- `CyU3PReturnStatus_t CyU3PSpiSetSsnLine (CyBool_t isHigh)`  
*Assert or de-assert the SSN Line.*
- `CyU3PReturnStatus_t CyU3PSpiTransmitWords (uint8_t *data, uint32_t byteCount)`  
*Transmits data word by word over the SPI interface.*
- `CyU3PReturnStatus_t CyU3PSpiReceiveWords (uint8_t *data, uint32_t byteCount)`  
*Receives data word by word over the SPI interface.*
- `CyU3PReturnStatus_t CyU3PSpiTransferWords (uint8_t *txBuf, uint32_t txByteCount, uint8_t *rxBuf, uint32_t rxByteCount)`  
*Perform byte-by-byte bi-directional transfers across the SPI interface.*
- `CyU3PReturnStatus_t CyU3PSpiSetBlockXfer (uint32_t txSize, uint32_t rxSize)`  
*Initiate SPI data transfers in DMA mode.*
- `CyU3PReturnStatus_t CyU3PSpiDisableBlockXfer (CyBool_t rxDisable, CyBool_t txDisable)`  
*This function disables the SPI DMA TX/RX functionality.*
- `CyU3PReturnStatus_t CyU3PSpiWaitForBlockXfer (CyBool_t isRead)`  
*Wait until the ongoing SPI DMA transfer is finished.*
- void `CyU3PRegisterSpiCallBack (CyU3PSpiIntrCb_t spilntrCb)`  
*This function registers the callback function for notification of SPI events.*
- `CyU3PReturnStatus_t CyU3PSpiSetTimeout (uint32_t readLoopCnt, uint32_t writeLoopCnt)`  
*Set the timeout duration for SPI read/write transfer APIs.*

### 5.32.1 Detailed Description

SPI (Serial Peripheral Interface) is a serial interface defined for inter-device communication. The FX3 device includes a SPI master that can connect to a variety of SPI slave devices and function at various speeds and modes. The SPI driver module provides functions to configure the SPI interface parameters and to perform data read/writes from/to the slave devices.

### 5.32.2 Typedef Documentation

#### 5.32.2.1 `typedef struct CyU3PSpiConfig_t CyU3PSpiConfig_t`

Structure defining the configuration of SPI interface.

##### Description

This structure encapsulates all of the configurable parameters that can be selected for the SPI interface. The `CyU3PSpiSetConfig()` function accepts a pointer to this structure, and updates all of the interface parameters.

See also

[CyU3PSpiSsnCtrl\\_t](#)  
[CyU3PSpiSclkParam\\_t](#)  
[CyU3PSpiSetConfig](#)

### 5.32.2.2 `typedef enum CyU3PSpiError_t CyU3PSpiError_t`

List of SPI specific error / status codes.

#### Description

This type lists the various SPI specific error / status codes that are sent to the event callback as event data, when the event type is CY\_U3P\_SPI\_EVENT\_ERROR.

See also

[CyU3PSpiEvt\\_t](#)

### 5.32.2.3 `typedef enum CyU3PSpiEvt_t CyU3PSpiEvt_t`

List of SPI related event types.

#### Description

This enumeration lists the various SPI related event codes that are notified to the user application through an event callback.

See also

[CyU3PSpiError\\_t](#)  
[CyU3PSpiIntrCb\\_t](#)

### 5.32.2.4 `typedef void(* CyU3PSpiIntrCb_t)(CyU3PSpiEvt_t evt, CyU3PSpiError_t error )`

Prototype of SPI event callback function.

#### Description

This function type defines a callback to be called when an SPI interrupt has been detected. A function of this type can be registered with the SPI driver as a callback function and will be called whenever an event of interest occurs.

See also

[CyU3PSpiEvt\\_t](#)  
[CyU3PSpiError\\_t](#)  
[CyU3PRegisterSpiCallBack](#)

### 5.32.2.5 `typedef enum CyU3PSpiSsnCtrl_t CyU3PSpiSsnCtrl_t`

Enumeration defining the various ways in which the SSN for a SPI device can be controlled.

#### Description

The SSN line can be controlled by the firmware or the FX3 hardware.

If the SSN is controlled by the firmware API, it will be asserted at the beginning of a transfer and is de-asserted at the end of the transfer. If it is controlled by hardware, the SSN assertion and de-assertion is done in sync with the SPI clock.

In addition to these modes, the SSN can be controlled by the firmware application through a GPIO. In this case, it is the responsibility of the application to drive the line appropriately.

The APIs allow only control of one SSN line. The SSN for any additional slaves must be controlled through GPIOs by the application.

See also

[CyU3PSpiConfig\\_t](#)  
[CyU3PSpiSetConfig](#)

### 5.32.2.6 `typedef enum CyU3PSpiSsnLagLead_t CyU3PSpiSsnLagLead_t`

Enumeration defining the lead and lag times of SSN with respect to SCK.

#### Description

The SSN needs to lead the SCK at the beginning of a transaction and SSN needs to lag the SCK at the end of a transfer. This enumeration gives customization allowed for this. This enumeration is required only for hardware controlled SSN.

See also

[CyU3PSpiConfig\\_t](#)  
[CyU3PSpiSetConfig](#)

### 5.32.3 Enumeration Type Documentation

#### 5.32.3.1 `enum CyU3PSpiError_t`

List of SPI specific error / status codes.

#### Description

This type lists the various SPI specific error / status codes that are sent to the event callback as event data, when the event type is CY\_U3P\_SPI\_EVENT\_ERROR.

See also

[CyU3PSpiEvt\\_t](#)

Enumerator

**CY\_U3P\_SPI\_ERROR\_TX\_OVERFLOW** Write to TX\_FIFO when FIFO is full.

**CY\_U3P\_SPI\_ERROR\_RX\_UNDERFLOW** Read from RX\_FIFO when FIFO is empty.

**CY\_U3P\_SPI\_ERROR\_NONE** No error has happened.

#### 5.32.3.2 `enum CyU3PSpiEvt_t`

List of SPI related event types.

#### Description

This enumeration lists the various SPI related event codes that are notified to the user application through an event callback.

See also

[CyU3PSpiError\\_t](#)  
[CyU3PSpiIntrCb\\_t](#)

Enumerator

**CY\_U3P\_SPI\_EVENT\_RX\_DONE** Reception is completed

**CY\_U3P\_SPI\_EVENT\_TX\_DONE** Transmission is done

**CY\_U3P\_SPI\_EVENT\_ERROR** Error has occurred

### 5.32.3.3 enum CyU3PSpiSsnCtrl\_t

Enumeration defining the various ways in which the SSN for a SPI device can be controlled.

#### Description

The SSN line can be controlled by the firmware or the FX3 hardware.

If the SSN is controlled by the firmware API, it will be asserted at the beginning of a transfer and is de-asserted at the end of the transfer. If it is controlled by hardware, the SSN assertion and de-assertion is done in sync with the SPI clock.

In addition to these modes, the SSN can be controlled by the firmware application through a GPIO. In this case, it is the responsibility of the application to drive the line appropriately.

The APIs allow only control of one SSN line. The SSN for any additional slaves must be controlled through GPIOs by the application.

#### See also

[CyU3PSpiConfig\\_t](#)  
[CyU3PSpiSetConfig](#)

#### Enumerator

**CY\_U3P\_SPI\_SSN\_CTRL\_FW** SSN is controlled by API and is not at clock boundaries.

**CY\_U3P\_SPI\_SSN\_CTRL\_HW\_END\_OF\_XFER** SSN is controlled by hardware and is done in sync with clock. The SSN is asserted at the beginning of a transfer, and de-asserted at the end of a transfer or when no data is available to transmit.

**CY\_U3P\_SPI\_SSN\_CTRL\_HW\_EACH\_WORD** SSN is controlled by the hardware and is done in sync with clock. The SSN is asserted at the beginning of transfer of every word, and de-asserted at the end of the transfer of that word.

**CY\_U3P\_SPI\_SSN\_CTRL\_HW\_CPHA\_BASED** If CPHA is 0, the SSN control is per word; and if CPHA is 1, then the SSN control is per transfer.

**CY\_U3P\_SPI\_SSN\_CTRL\_NONE** SSN control is done externally by the firmware application.

**CY\_U3P\_SPI\_NUM\_SSN\_CTRL** Number of enumerations.

### 5.32.3.4 enum CyU3PSpiSsnLagLead\_t

Enumeration defining the lead and lag times of SSN with respect to SCK.

#### Description

The SSN needs to lead the SCK at the beginning of a transaction and SSN needs to lag the SCK at the end of a transfer. This enumeration gives customization allowed for this. This enumeration is required only for hardware controlled SSN.

#### See also

[CyU3PSpiConfig\\_t](#)  
[CyU3PSpiSetConfig](#)

#### Enumerator

**CY\_U3P\_SPI\_SSN\_LAG\_LEAD\_ZERO\_CLK** SSN is in sync with SCK.

**CY\_U3P\_SPI\_SSN\_LAG\_LEAD\_HALF\_CLK** SSN leads / lags SCK by a half clock cycle.

**CY\_U3P\_SPI\_SSN\_LAG\_LEAD\_ONE\_CLK** SSN leads / lags SCK by one clock cycle.

**CY\_U3P\_SPI\_SSN\_LAG\_LEAD\_ONE\_HALF\_CLK** SSN leads / lags SCK by one and half clock cycles.

**CY\_U3P\_SPI\_NUM\_SSN\_LAG\_LEAD** Number of possible values.

## 5.32.4 Function Documentation

### 5.32.4.1 void CyU3PRegisterSpiCallBack ( *CyU3PSpiIntrCb\_t spilntrCb* )

This function registers the callback function for notification of SPI events.

#### Description

This function registers a callback function that will be called for notification of SPI interrupts. This can also be done through the CyU3PSpiInit API.

#### Return value

None

#### See also

[CyU3PSpiIntrCb\\_t](#)  
[CyU3PSpiInit](#)

#### Parameters

<i>spilntrCb</i>	Callback function pointer.
------------------	----------------------------

### 5.32.4.2 CyU3PReturnStatus\_t CyU3PSpiDeInit ( void )

Stops the SPI module.

#### Description

This function disables and powers off the SPI interface. This function can be used to shut off the interface to save power when it is not in use.

#### Return value

CY\_U3P\_SUCCESS - When the de-init is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - When the SPI block has not been initialized.

#### See also

[CyU3PSpiInit](#)

### 5.32.4.3 CyU3PReturnStatus\_t CyU3PSpiDisableBlockXfer ( *CyBool\_t rxDisable*, *CyBool\_t txDisable* )

This function disables the SPI DMA TX/RX functionality.

#### Description

This function disables DMA transfers through the SPI block in the TX and/or RX directions. It is possible to disable only the RX or the TX operation, and leave the transfer in the other direction running.

Both TX and RX transfers need to be disabled through this API before register mode transfers can be used.

#### Return value

CY\_U3P\_SUCCESS - When the DisableBlockXfer is successful.

CY\_U3P\_ERROR\_NOT\_CONFIGURED - When SPI has not been configured or if no DMA transfer has been configured.

CY\_U3P\_ERROR\_MUTEX\_FAILURE - Failed to get a mutex lock on the SPI block.

#### See also

[CyU3PSpiSetConfig](#)  
[CyU3PSpiSetBlockXfer](#)  
[CyU3PSpiWaitForBlockXfer](#)

**Parameters**

<i>rxDisable</i>	CyTrue: Disable TX block if active; CyFalse: Ignore TX block.
<i>txDisable</i>	CyTrue: Disable RX block if active; CyFalse: Ignore RX block.

**5.32.4.4 CyU3PReturnStatus\_t CyU3PSpiInit( void )**

Starts the SPI interface block on the device.

**Description**

This function powers up the SPI interface block on the device and is expected to be the first SPI API function that is called by the application.

**Return value**

CY\_U3P\_SUCCESS - When the init is successful.

CY\_U3P\_ERROR\_NOT\_CONFIGURED - When SPI has not been enabled in the IO configuration.

CY\_U3P\_ERROR\_ALREADY\_STARTED - When the SPI block has already been initialized.

**See also**

[CyU3PSpiDeInit](#)  
[CyU3PSpiSetConfig](#)  
[CyU3PSpiSetSsnLine](#)  
[CyU3PSpiTransmitWords](#)  
[CyU3PSpiReceiveWords](#)  
[CyU3PSpiTransferWords](#)  
[CyU3PSpiSetBlockXfer](#)  
[CyU3PSpiWaitForBlockXfer](#)  
[CyU3PSpiDisableBlockXfer](#)

**5.32.4.5 CyU3PReturnStatus\_t CyU3PSpiReceiveWords( uint8\_t \* data, uint32\_t byteCount )**

Receives data word by word over the SPI interface.

**Description**

Reads data from the SPI slave in register mode. This API should only be called when there is no active DMA transfer.

**Return value**

CY\_U3P\_SUCCESS - When the data transfer is successful.

CY\_U3P\_ERROR\_NOT\_CONFIGURED - When SPI block has not been configured or initialized.

CY\_U3P\_ERROR\_NULL\_POINTER - When the data pointer is NULL.

CY\_U3P\_ERROR\_ALREADY\_STARTED - When a DMA transfer is active.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - When a configured word length is invalid.

CY\_U3P\_ERROR\_TIMEOUT - If the data transfer times out.

CY\_U3P\_ERROR\_MUTEX\_FAILURE - Failed to get a mutex lock on the SPI block.

**See also**

[CyU3PSpiSetConfig](#)  
[CyU3PSpiSetSsnLine](#)  
[CyU3PSpiTransmitWords](#)  
[CyU3PSpiSetBlockXfer](#)

**Parameters**

<i>data</i>	Buffer to read the data into.
<i>byteCount</i>	Amount of data to be read. This should be an integral multiple of the SPI word length aligned to the next byte.
Generated for EZ-USB FX3 SDK by Doxygen	



CY\_U3P\_ERROR\_TIMEOUT - Attempt to clear the SPI block FIFOs timed out.

CY\_U3P\_ERROR\_MUTEX\_FAILURE - Failed to get a mutex lock on the SPI block.

#### See also

[CyU3PSpiConfig\\_t](#)  
[CyU3PSpiIntrCb\\_t](#)  
[CyU3PSpiInit](#)

#### Parameters

<i>config</i>	Pointer to the SPI config structure.
<i>cb</i>	Callback for receiving SPI events.

#### 5.32.4.8 CyU3PReturnStatus\_t CyU3PSpiSetSsnLine ( CyBool\_t *isHigh* )

Assert or de-assert the SSN Line.

#### Description

Asserts or de-asserts the SSN Line for the default slave device. This is possible only if the SSN line control is configured for FW controlled mode.

#### Return value

CY\_U3P\_SUCCESS - When the call is successful.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - When SPI has not been configured or if SSN is not firmware controlled.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - Failed to get a mutex lock on the SPI block.

#### See also

[CyU3PSpiInit](#)  
[CyU3PSpiSetConfig](#)  
[CyU3PSpiTransmitWords](#)  
[CyU3PSpiReceiveWords](#)  
[CyU3PSpiTransferWords](#)  
[CyU3PSpiSetBlockXfer](#)

#### Parameters

<i>isHigh</i>	CyFalse: Pull down the SSN line, CyTrue: Pull up the SSN line.
---------------	--

#### 5.32.4.9 CyU3PReturnStatus\_t CyU3PSpiSetTimeout ( uint32\_t *readLoopCnt*, uint32\_t *writeLoopCnt* )

Set the timeout duration for SPI read/write transfer APIs.

#### Description

The CyU3PSpiTransmitWords, CyU3PSpiReceiveWords and CyU3PSpiWaitForBlockXfer APIs wait for the completion of the requested data transfer. In the case of the byte mode transfer APIs, the default timeout is 1 million loops with a delay of about 1 us. In the case of the CyU3PSpiWaitForBlockXfer API, the default timeout is 1 million loops with a delay of 10 us.

The default timeout for read and write operations can be changed using this API. The loop count to be used during read and write operations can be set to different values.

#### Return value

CY\_U3P\_SUCCESS if the timeout duration is updated. CY\_U3P\_ERROR\_NOT\_STARTED if the SPI block has not been initialized.

## See also

[CyU3PSpiTransmitWords](#)  
[CyU3PSpiReceiveWords](#)  
[CyU3PSpiWaitForBlockXfer](#)

## Parameters

<i>readLoopCnt</i>	Wait loop count used during read operations.
<i>writeLoopCnt</i>	Wait loop count used during write operations.

### 5.32.4.10 CyU3PReturnStatus\_t CyU3PSpiTransferWords ( *uint8\_t \* txBuf, uint32\_t txByteCount, uint8\_t \* rxBuf, uint32\_t rxByteCount* )

Perform byte-by-byte bi-directional transfers across the SPI interface.

**Description**

This function is used to perform bi-directional SPI data transfers in register mode. The function can be called only if there is no active DMA transfer on the bus. If CyU3PSpiSetBlockXfer was called, then CyU3PSpiDisableBlockXfer needs to be called before this function.

**Return value**

CY\_U3P\_SUCCESS - When the data transfer is successful.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - When SPI block has not been configured or initialized.  
 CY\_U3P\_ERROR\_NULL\_POINTER - When the data pointer is NULL.  
 CY\_U3P\_ERROR\_ALREADY\_STARTED - When a DMA transfer is active.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - When a configured word length is invalid.  
 CY\_U3P\_ERROR\_TIMEOUT - If the data transfer times out.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - Failed to get a mutex lock on the SPI block.

## See also

[CyU3PSpiSetConfig](#)  
[CyU3PSpiSetSsnLine](#)  
[CyU3PSpiTransmitWords](#)  
[CyU3PSpiReceiveWords](#)  
[CyU3PSpiSetBlockXfer](#)

## Parameters

<i>txBuf</i>	Source buffer containing data to transmit. Can be NULL if txByteCount is 0.
<i>txByteCount</i>	Number of data bytes to transmit. Needs to be a multiple of the word length aligned to the next byte.
<i>rxBuf</i>	Destination buffer to receive data into. Can be NULL if rxByteCount is 0.
<i>rxByteCount</i>	Number of data bytes to receive. Needs to be a multiple of the word length aligned to the next byte.

### 5.32.4.11 CyU3PReturnStatus\_t CyU3PSpiTransmitWords ( *uint8\_t \* data, uint32\_t byteCount* )

Transmits data word by word over the SPI interface.

**Description**

This function is used to transmit data in register mode. The function can be called only if there is no active DMA transfer on the bus. If CyU3PSpiSetBlockXfer was called, then CyU3PSpiDisableBlockXfer needs to be called.

**Return value**

CY\_U3P\_SUCCESS - When the data transfer is successful.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - When SPI block has not been configured or initialized.  
 CY\_U3P\_ERROR\_NULL\_POINTER - When the data pointer is NULL.  
 CY\_U3P\_ERROR\_ALREADY\_STARTED - When a DMA transfer is active.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - When a configured word length is invalid.  
 CY\_U3P\_ERROR\_TIMEOUT - If the data transfer times out.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - Failed to get a mutex lock on the SPI block.

**See also**

[CyU3PSpiSetConfig](#)  
[CyU3PSpiSetSsnLine](#)  
[CyU3PSpiReceiveWords](#)  
[CyU3PSpiSetBlockXfer](#)

**Parameters**

<i>data</i>	Source data pointer. This needs to be padded to nearest byte if the word length is not byte aligned.
<i>byteCount</i>	This needs to be a multiple of the word length aligned to the next byte.

**5.32.4.12 CyU3PReturnStatus\_t CyU3PSpiWaitForBlockXfer ( CyBool\_t *isRead* )**

Wait until the ongoing SPI DMA transfer is finished.

**Description**

This function can be used to ensure that a previous SPI transaction has completed, in the case where the callback is not being used.

**Note**

This function will keep waiting for a transfer complete event until the pre-defined timeout period, if it is invoked before any SPI transfers are requested.

**Return value**

CY\_U3P\_SUCCESS - When the WaitForBlockXfer is successful.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - When SPI has not been configured or initialized.  
 CY\_U3P\_ERROR\_TIMEOUT - If the transfer is not completed within the timeout period.  
 CY\_U3P\_ERROR\_FAILURE - When the operation encounters an error.  
 CY\_U3P\_ERROR\_MUTEX\_FAILURE - Failed to get a mutex lock on the SPI block.

**See also**

[CyU3PSpiSetConfig](#)  
[CyU3PSpiSetBlockXfer](#)  
[CyU3PSpiDisableBlockXfer](#)

**Parameters**

<i>isRead</i>	CyFalse: Write operation, CyTrue: Read operation.
---------------	---

**5.33 firmware/u3p\_firmware/inc/cyu3system.h File Reference**

This file defines the device initialization and configuration functions associated with the FX3 SDK.

```
#include <cyu3types.h>
#include <cyfx3_api.h>
#include <cyu3dma.h>
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

## Data Structures

- struct [CyU3PSysClockConfig\\_t](#)  
*Clock configuration for FX3 CPU, DMA and register access.*
- struct [CyU3PDebugLog\\_t](#)  
*FX3 debug logger data type.*

## Macros

- #define [CY\\_U3P\\_SYS\\_PPORT\\_WAKEUP\\_SRC](#) (0x01)  
*Bit mask for selecting the GPIF (P-port) source to wake FX3 from suspend / standby.*
- #define [CY\\_U3P\\_SYS\\_UART\\_WAKEUP\\_SRC](#) (0x02)  
*Bit mask for selecting the UART CTS as a source to wake FX3 from suspend / standby.*
- #define [CY\\_U3P\\_SYS\\_USB\\_VBUS\\_WAKEUP\\_SRC](#) (0x04)  
*Bit mask for selecting the VBus signal as a source to wake FX3 from suspend / standby.*
- #define [CY\\_U3P\\_SYS\\_USB\\_BUS\\_ACTVTY\\_WAKEUP\\_SRC](#) (0x08)  
*Bit mask for selecting USB bus activity as a source to wake FX3 from suspend.*
- #define [CY\\_U3P\\_SYS\\_USB\\_OTGID\\_WAKEUP\\_SRC](#) (0x10)  
*Bit mask for selecting the USB OTG ID pin as a source to wake FX3 from suspend.*
- #define [CY\\_U3P\\_DEBUG\\_DMA\\_BUFFER\\_SIZE](#) (0x100)  
*Buffer size used on the DMA channel used to output debug log messages.*
- #define [CY\\_U3P\\_DEBUG\\_DMA\\_BUFFER\\_COUNT](#) (8)  
*Buffer count used on the DMA channel used to output debug log messages.*

## Typedefs

- typedef enum [CyU3PLppModule\\_t](#) [CyU3PLppModule\\_t](#)  
*List of Low Performance (Serial) Peripherals supported by the FX3 device.*
- typedef enum [CyU3PDriveStrengthState\\_t](#) [CyU3PDriveStrengthState\\_t](#)  
*Drive Strength configuration for various FX3 IOs.*
- typedef enum [CyU3PSysClockSrc\\_t](#) [CyU3PSysClockSrc\\_t](#)  
*Clock source for a peripheral block.*
- typedef struct [CyU3PSysClockConfig\\_t](#) [CyU3PSysClockConfig\\_t](#)  
*Clock configuration for FX3 CPU, DMA and register access.*
- typedef enum [CyU3PSysThreadId\\_t](#) [CyU3PSysThreadId\\_t](#)  
*FX3 API library thread information.*
- typedef struct [CyU3PDebugLog\\_t](#) [CyU3PDebugLog\\_t](#)  
*FX3 debug logger data type.*

## Enumerations

- enum `CyU3PLppModule_t` {
 `CY_U3P_LPP_I2C` = 0, `CY_U3P_LPP_I2S`, `CY_U3P_LPP_SPI`, `CY_U3P_LPP_UART`, `CY_U3P_LPP_GPIO` }
 

*List of Low Performance (Serial) Peripherals supported by the FX3 device.*
- enum `CyU3PDriveStrengthState_t` { `CY_U3P_DS_QUARTER_STRENGTH` = 0, `CY_U3P_DS_HALF_STRENGTH`, `CY_U3P_DS_THREE_QUARTER_STRENGTH`, `CY_U3P_DS_FULL_STRENGTH` }
 

*Drive Strength configuration for various FX3 IOs.*
- enum `CyU3PSysClockSrc_t` {
 `CY_U3P_SYS_CLK_BY_16` = 0, `CY_U3P_SYS_CLK_BY_4`, `CY_U3P_SYS_CLK_BY_2`, `CY_U3P_SYS_CLK`, `CY_U3P_NUM_CLK_SRC` }
 

*Clock source for a peripheral block.*
- enum `CyU3PSysThreadId_t` {
 `CY_U3P_THREAD_ID_INT` = 0, `CY_U3P_THREAD_ID_DMA`, `CY_U3P_THREAD_ID_SYSTEM`, `CY_U3P_THREAD_ID_PIB`, `CY_U3P_THREAD_ID_UIB`, `CY_U3P_THREAD_ID_LPP`, `CY_U3P_THREAD_ID_DEBUG` = 8, `CY_U3P_THREAD_ID_LIB_MAX` = 15 }
 

*FX3 API library thread information.*

## Functions

- void `CyU3PFirmwareEntry` (void)
 

*This is the entry routine for the firmware.*
- void `CyU3PToolChainInit` (void)
 

*This is the normal entry routine provided by the tool-chain.*
- `CyU3PReturnStatus_t CyU3PDeviceInit` (`CyU3PSysClockConfig_t` \*clkCfg)
 

*This function initializes the device.*
- `CyU3PReturnStatus_t CyU3PDeviceCacheControl` (`CyBool_t` isICacheEnable, `CyBool_t` isDCacheEnable, `CyBool_t` isDmaHandleDCache)
 

*This function initializes the caches on the ARM926 core.*
- void `CyFxApplicationDefine` (void)
 

*This is the FX3 application definition function.*
- `CyU3PReturnStatus_t CyU3PSysGetApiVersion` (`uint16_t` \*majorVersion, `uint16_t` \*minorVersion, `uint16_t` \*patchNumer, `uint16_t` \*buildNumer)
 

*This function returns the API library version.*
- `CyU3PReturnStatus_t CyU3PDeviceGetSysClkFreq` (`uint32_t` \*freq)
 

*This function returns the current SYS\_CLK frequency.*
- void `CyU3PDeviceReset` (`CyBool_t` isWarmReset)
 

*This function resets the FX3 device.*
- `CyU3PReturnStatus_t CyU3PSysCheckSuspendParams` (`uint16_t` wakeupFlags, `uint16_t` polarity)
 

*Verify FX3 device is ready to enter in low power suspend mode.*
- void `CyU3PSysSendEnterSuspendStatus` (void)
 

*Wait for P-Port to receive last mailbox response from FX3 device prior to entering in suspend mode.*
- `CyU3PReturnStatus_t CyU3PSysEnterSuspendMode` (`uint16_t` wakeupFlags, `uint16_t` polarity, `uint16_t` \*wakeupSource)
 

*Places the FX3 device in low power suspend mode.*
- `CyU3PReturnStatus_t CyU3PSysCheckStandbyParam` (`uint16_t` wakeupFlags, `uint16_t` polarity, `uint8_t` \*bkp\_buff\_p)
 

*Verify FX3 device is ready to enter in low power standby mode.*
- `CyU3PReturnStatus_t CyU3PSysEnterStandbyMode` (`uint16_t` wakeupFlags, `uint16_t` polarity, `uint8_t` \*bkp\_buff\_p)

- Places the FX3 device in low power standby mode.*
- void [CyU3PSysWatchDogConfigure](#) ([CyBool\\_t](#) enable, [uint32\\_t](#) period)
 

*Configure the watchdog reset control.*
  - void [CyU3PSysWatchDogClear](#) (void)
 

*Clear the watchdog timer to prevent a device reset.*
  - [CyU3PReturnStatus\\_t CyU3PDeviceConfigureIOMatrix](#) ([CyU3PloMatrixConfig\\_t](#) \*cfg\_p)
 

*Configures the IO matrix for the device.*
  - [CyU3PReturnStatus\\_t CyU3PDeviceGpioOverride](#) ([uint8\\_t](#) gpioId, [CyBool\\_t](#) isSimple)
 

*This function is used to override an IO as a GPIO.*
  - [CyU3PReturnStatus\\_t CyU3PDeviceGpioRestore](#) ([uint8\\_t](#) gpioId)
 

*This function is used to restore IO to normal mode.*
  - [CyU3PReturnStatus\\_t CyU3PSetSerialIODriveStrength](#) ([CyU3PDriveStrengthState\\_t](#) driveStrength)
 

*Set the IO drive strength for UART, SPI and I2S interfaces.*
  - [CyBool\\_t CyU3PIsGpioSimpleIOConfigured](#) ([uint32\\_t](#) gpioId)
 

*Check whether a specific pin has been selected as a simple GPIO.*
  - [CyBool\\_t CyU3PIsGpioComplexIOConfigured](#) ([uint32\\_t](#) gpioId)
 

*Check whether a specific pin has been selected as a complex GPIO.*
  - [CyBool\\_t CyU3PIsGpioValid](#) ([uint8\\_t](#) gpioId)
 

*Check whether a specified GPIO ID is valid on the current FX3 part.*
  - [CyBool\\_t CyU3PIsLppIOConfigured](#) ([CyU3PLppModule\\_t](#) lppModule)
 

*Check whether a specific serial peripheral has been enabled in the IO Matrix.*
  - [CyU3PPartNumber\\_t CyU3PDeviceGetPartNumber](#) (void)
 

*This function is used to get the part number of the FX3 device in use.*
  - [CyU3PReturnStatus\\_t CyU3PDebugInit](#) ([CyU3PDmaSocketId\\_t](#) destSckId, [uint8\\_t](#) traceLevel)
 

*Initialize the firmware logging functionality.*
  - [CyU3PReturnStatus\\_t CyU3PDebugSetTimeout](#) ([uint32\\_t](#) timeout)
 

*Set the timeout period for a debug log function.*
  - [CyU3PReturnStatus\\_t CyU3PDebugDelInit](#) (void)
 

*De-initialize the logging function.*
  - [CyU3PReturnStatus\\_t CyU3PDebugSysMemInit](#) ([uint8\\_t](#) \*buffer, [uint16\\_t](#) size, [CyBool\\_t](#) isWrapAround, [uint8\\_t](#) traceLevel)
 

*Initializes the SYS\_MEM based logger facility.*
  - [CyU3PReturnStatus\\_t CyU3PDebugSysMemDelInit](#) (void)
 

*Disables the SYS\_MEM logger.*
  - void [CyU3PDebugPreamble](#) ([CyBool\\_t](#) sendPreamble)
 

*Inhibits the preamble bytes preceding a verbose log message.*
  - [CyU3PReturnStatus\\_t CyU3PDebugPrint](#) ([uint8\\_t](#) priority, char \*message,...)
 

*Free form message logging function.*
  - [CyU3PReturnStatus\\_t CyU3PDebugStringPrint](#) ([uint8\\_t](#) \*buffer, [uint16\\_t](#) maxLength, char \*fmt,...)
 

*Miniatute version of snprintf routine.*
  - [CyU3PReturnStatus\\_t CyU3PDebugLog](#) ([uint8\\_t](#) priority, [uint16\\_t](#) message, [uint32\\_t](#) parameter)
 

*Log a codified message.*
  - [CyU3PReturnStatus\\_t CyU3PDebugLogFlush](#) (void)
 

*Flush any pending logs out of the internal buffers.*
  - [CyU3PReturnStatus\\_t CyU3PDebugLogClear](#) (void)
 

*Drop any pending logs in the internal buffers.*
  - void [CyU3PDebugSetTraceLevel](#) ([uint8\\_t](#) level)
 

*This function sets the priority threshold above which debug traces will be logged.*
  - void [CyU3PDebugEnable](#) ([uint16\\_t](#) threadMask)
 

*Enable log messages from specified threads.*
  - [uint16\\_t CyU3PDebugDisable](#) (void)

- Disable log messages from all library threads.*
- void [CyU3PSystemRegisterDriver](#) ([CyU3PThread](#) \*thread\_p)  
*Internal function used to register driver threads.*
  - uint32\_t [CyU3PDeviceGetCpuLoad](#) (void)  
*Get the average CPU load from start of firmware operation.*
  - uint32\_t [CyU3PDeviceGetThreadLoad](#) ([CyU3PThread](#) \*thread\_p)  
*Get the CPU loading due to a given thread.*
  - uint32\_t [CyU3PDeviceGetDriverLoad](#) (void)  
*Get the CPU loading due to SDK internal driver threads.*
  - void [CyU3PJumpToAddress](#) (uint32\_t address)  
*Jump to the specified instruction address.*

### 5.33.1 Detailed Description

This file defines the device initialization and configuration functions associated with the FX3 SDK.

### 5.33.2 Typedef Documentation

#### 5.33.2.1 [typedef struct CyU3PDebugLog\\_t CyU3PDebugLog\\_t](#)

FX3 debug logger data type.

##### Description

The structure defines the header data type sent out by the debug logger function. For CyU3PDebugPrint function, the preamble will be the same with msg = 0xFFFF and param specifying the size of the message in bytes.

##### See also

[CyU3PDebugLog](#)  
[CyU3PDebugPrint](#)

#### 5.33.2.2 [typedef enum CyU3PDriveStrengthState\\_t CyU3PDriveStrengthState\\_t](#)

Drive Strength configuration for various FX3 IOs.

##### Description

IOs on the FX3 device are grouped based on function into multiple interfaces (GPIF, I2C, I2S, SPI, UART). The IO drive strength for each of these groups can be separately selected from this list.

##### See also

[CyU3PSetPportDriveStrength](#)  
[CyU3PSetI2cDriveStrength](#)  
[CyU3PSetGpioDriveStrength](#)  
[CyU3PSetSerialIoDriveStrength](#)

#### 5.33.2.3 [typedef enum CyU3PLppModule\\_t CyU3PLppModule\\_t](#)

List of Low Performance (Serial) Peripherals supported by the FX3 device.

##### Description

This enumeration lists the various serial (low performance) peripherals supported by the FX3 device. This type is used to notify the LPP driver about the initialization of individual peripheral modules.

**See also**

[CyU3PIsLppIOConfigured](#)

#### 5.33.2.4 `typedef struct CyU3PSysClockConfig_t CyU3PSysClockConfig_t`

Clock configuration for FX3 CPU, DMA and register access.

**Description**

This structure holds information to set the clock divider for CPU, DMA and internal register access. The DMA and register (MMIO) clocks are derived from the CPU clock.

There is an additional condition: DMA clock = N \* MMIO clock, where N is a positive integer.

The useStandbyClk parameter specifies whether a 32KHz clock has been supplied on the CLKIN\_32 pin of the device. This clock is the standby clock for the device. If this pin is not connected, then the device internally generates a clock from the SYS\_CLK.

The setSysClk400 parameter specifies whether the FX3 device's master clock is to be set to a frequency greater than 400 MHz. By default, the FX3 master clock is set to 384 MHz when using a 19.2 MHz crystal or clock source. This frequency setting may lead to DMA overflow errors on the GPIF, if the GPIF is configured as 32-bit wide and is running at 100 MHz. Setting this parameter will switch the master clock frequency to 403.2 MHz during the CyU3PDeviceInit call.

This structure is passed as parameter to CyU3PDeviceInit call.

**See also**

[CyU3PSysClockSrc\\_t](#)  
[CyU3PDeviceInit](#)  
[CyU3PDeviceGetSysClkFreq](#)

#### 5.33.2.5 `typedef enum CyU3PSysClockSrc_t CyU3PSysClockSrc_t`

Clock source for a peripheral block.

**Description**

Each peripheral block on FX3 can take various clock values based on the system clock supplied. All clocks are derived from the SYS\_CLK\_PLL value which depends on the input clock or crystal connected to the device.

The FX3 device identifies the input clock frequency based on the FSLC pins, and these pins should be configured correctly for proper device operation.

The SYS\_CLK\_PLL value is 416 MHz if the input clock frequency is 26 MHz or 52 MHz; and it is 384 MHz if the input clock frequency is 19.2 MHz or 38.4 MHz. The SYS\_CLK\_PLL frequency can be changed to 403.2 MHz from 384 MHz by passing appropriate parameters to the CyU3PDeviceInit function.

**See also**

[CyU3PSysClockConfig\\_t](#)  
[CyU3PDeviceGetSysClkFreq](#)  
[CyU3PDeviceInit](#)

#### 5.33.2.6 `typedef enum CyU3PSysThreadId_t CyU3PSysThreadId_t`

FX3 API library thread information.

**Description**

This enumeration holds the thread IDs used by various FX3 API library threads. The thread ID is defined by the first two characters of the thread name provided during thread creation. For example, the DMA thread name is "01\_DMA\_THREAD".

Thread IDs 0-15 are reserved by the library. Thread IDs from 16 onwards can be used by FX3 application. All threads created in FX3 application are expected to have the first two characters as the thread ID. Please note that this is a convention used for readability of debug logs output through the CyU3PDebugLog function, and is not enforced otherwise.

#### See also

[CyU3PDebugEnable](#)  
[CyU3PDebugDisable](#)  
[CyU3PDebugLog](#)

### 5.33.3 Enumeration Type Documentation

#### 5.33.3.1 enum CyU3PDriveStrengthState\_t

Drive Strength configuration for various FX3 IOs.

##### Description

IOs on the FX3 device are grouped based on function into multiple interfaces (GPIF, I2C, I2S, SPI, UART). The IO drive strength for each of these groups can be separately selected from this list.

#### See also

[CyU3PSetPportDriveStrength](#)  
[CyU3PSetI2cDriveStrength](#)  
[CyU3PSetGpioDriveStrength](#)  
[CyU3PSetSerialIODriveStrength](#)

##### Enumerator

**CY\_U3P\_DS\_QUARTER\_STRENGTH** The drive strength is one-fourth the maximum

**CY\_U3P\_DS\_HALF\_STRENGTH** The drive strength is half the maximum

**CY\_U3P\_DS\_THREE\_QUARTER\_STRENGTH** The drive strength is three-fourth the maximum

**CY\_U3P\_DS\_FULL\_STRENGTH** The drive strength is the maximum

#### 5.33.3.2 enum CyU3PLppModule\_t

List of Low Performance (Serial) Peripherals supported by the FX3 device.

##### Description

This enumeration lists the various serial (low performance) peripherals supported by the FX3 device. This type is used to notify the LPP driver about the initialization of individual peripheral modules.

#### See also

[CyU3PIsLppIOConfigured](#)

##### Enumerator

**CY\_U3P\_LPP\_I2C** I2C Master Module.

**CY\_U3P\_LPP\_I2S** I2S Master Module.

**CY\_U3P\_LPP\_SPI** SPI Master Module.

**CY\_U3P\_LPP\_UART** UART Module.

**CY\_U3P\_LPP\_GPIO** GPIO Block.

### 5.33.3.3 enum CyU3PSysClockSrc\_t

Clock source for a peripheral block.

#### Description

Each peripheral block on FX3 can take various clock values based on the system clock supplied. All clocks are derived from the SYS\_CLK\_PLL value which depends on the input clock or crystal connected to the device.

The FX3 device identifies the input clock frequency based on the FSLC pins, and these pins should be configured correctly for proper device operation.

The SYS\_CLK\_PLL value is 416 MHz if the input clock frequency is 26 MHz or 52 MHz; and it is 384 MHz if the input clock frequency is 19.2 MHz or 38.4 MHz. The SYS\_CLK\_PLL frequency can be changed to 403.2 MHz from 384 MHz by passing appropriate parameters to the CyU3PDeviceInit function.

#### See also

[CyU3PSysClockConfig\\_t](#)  
[CyU3PDeviceGetSysClkFreq](#)  
[CyU3PDeviceInit](#)

#### Enumerator

**CY\_U3P\_SYS\_CLK\_BY\_16** SYS\_CLK divided by 16.  
**CY\_U3P\_SYS\_CLK\_BY\_4** SYS\_CLK divided by 4.  
**CY\_U3P\_SYS\_CLK\_BY\_2** SYS\_CLK divided by 2.  
**CY\_U3P\_SYS\_CLK** SYS\_CLK frequency.  
**CY\_U3P\_NUM\_CLK\_SRC** Number of clock source enumerations.

### 5.33.3.4 enum CyU3PSysThreadId\_t

FX3 API library thread information.

#### Description

This enumeration holds the thread IDs used by various FX3 API library threads. The thread ID is defined by the first two characters of the thread name provided during thread creation. For example, the DMA thread name is "01\_DMA\_THREAD".

Thread IDs 0-15 are reserved by the library. Thread IDs from 16 onwards can be used by FX3 application. All threads created in FX3 application are expected to have the first two characters as the thread ID. Please note that this is a convention used for readability of debug logs output through the CyU3PDebugLog function, and is not enforced otherwise.

#### See also

[CyU3PDebugEnable](#)  
[CyU3PDebugDisable](#)  
[CyU3PDebugLog](#)

#### Enumerator

**CY\_U3P\_THREAD\_ID\_INT** Interrupt context.  
**CY\_U3P\_THREAD\_ID\_DMA** Library DMA module thread.  
**CY\_U3P\_THREAD\_ID\_SYSTEM** Library system module thread.  
**CY\_U3P\_THREAD\_ID\_PIB** Library p-port module thread.  
**CY\_U3P\_THREAD\_ID\_UIB** Library USB module thread.  
**CY\_U3P\_THREAD\_ID\_LPP** Library low performance peripheral module thread.  
**CY\_U3P\_THREAD\_ID\_DEBUG** Library debug module thread.  
**CY\_U3P\_THREAD\_ID\_LIB\_MAX** Max library reserved thread ID.

## 5.33.4 Function Documentation

### 5.33.4.1 void CyFxApplicationDefine ( void )

This is the FX3 application definition function.

#### Description

This function is called by the FX3 RTOS once all of the OS and driver initialization is completed. The RTOS objects and threads required by the application can be created in this function.

This function needs to be defined by the FX3 application firmware. At-least one thread must be created for meaningful operation. This function should not be explicitly called.

#### Return value

None

See also

[CyU3PKernelEntry](#)

### 5.33.4.2 CyU3PReturnStatus\_t CyU3PDebugDelInit ( void )

De-initialize the logging function.

#### Description

This function de-initializes the firmware logging function and destroys the DMA channel created to output the logs.

#### Return value

CY\_U3P\_SUCCESS - When successful.

CY\_U3P\_ERROR\_NOT\_STARTED - When the logger is not started.

See also

[CyU3PDebugInit](#)

### 5.33.4.3 uint16\_t CyU3PDebugDisable ( void )

Disable log messages from all library threads.

#### Description

The API returns the previous threadMask set, so that it can be used for re-enabling the logs.

#### Return value

The previous threadMask value.

See also

[CyU3PDebugEnable](#)

### 5.33.4.4 void CyU3PDebugEnable ( uint16\_t *threadMask* )

Enable log messages from specified threads.

#### Description

Thread ID of 0 means interrupt context. Only threadIds 1 - 15, and interrupt context can be controlled by this API call. This call is intended for only controlling logs from library threads. By default, logs from all library threads are disabled.

#### Return value

None

See also

[CyU3PDebugDisable](#)

Parameters

<i>threadMask</i>	ThreadID mask whose logs are to be enabled. 1 means enabled.
-------------------	--

#### 5.33.4.5 CyU3PReturnStatus\_t CyU3PDebugInit ( *CyU3PDmaSocketId\_t destSckId*, *uint8\_t traceLevel* )

Initialize the firmware logging functionality.

##### Description

This function initializes the firmware logging functionality and creates a DMA channel to send the log traces out through the selected DMA socket.

##### Return value

CY\_U3P\_SUCCESS - when successfully initialized.

CY\_U3P\_ERROR\_ALREADY\_STARTED - when the logger is already initialized.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - when bad socket ID is passed in as parameter.

Other DMA specific error codes are also returned.

See also

[CyU3PDebugDeInit](#)

[CyU3PDebugSetTimeout](#)

Parameters

<i>destSckId</i>	Socket through which the logs are to be output.
<i>traceLevel</i>	Priority level beyond which logs should be output.

#### 5.33.4.6 CyU3PReturnStatus\_t CyU3PDebugLog ( *uint8\_t priority*, *uint16\_t message*, *uint32\_t parameter* )

Log a codified message.

##### Description

This function is used to output a codified log message which contains a two byte message ID and a four byte parameter. The message ID is expected to be in the range 0x0000 to 0xFFFF.

The log messages are written to a log buffer. If CyU3PDebugSysMemInit has been used, the logs can be cleared using the CyU3PDebugLogClear function.

If CyU3PDebugInit has been used, the log buffer will be written out to the debug socket when it is full. The CyU3PDebugLogFlush can be used to flush the contents of the log buffer to the debug socket. The CyU3PDebugLogClear can be used to drop the current contents of the log buffer.

##### Return value

CY\_U3P\_SUCCESS - if the Debug log is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - if the debug module has not been initialized.

CY\_U3P\_ERROR\_FAILURE - if the logging fails.

See also

[CyU3PDebugInit](#)

[CyU3PDebugSysMemInit](#)

[CyU3PDebugLogFlush](#)

[CyU3PDebugLogClear](#)**Parameters**

<i>priority</i>	Priority level for the message.
<i>message</i>	Message ID for the message.
<i>parameter</i>	Parameter associated with the message.

**5.33.4.7 CyU3PReturnStatus\_t CyU3PDebugLogClear( void )**

Drop any pending logs in the internal buffers.

**Description**

This function is used to ask the logger to drop any pending logs in its internal buffers.

**Return value**

CY\_U3P\_SUCCESS - if the Debug log flush is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - if the debug module has not been initialized.

CY\_U3P\_ERROR\_INVALID\_CALLER - if called from an interrupt.

**See also**[CyU3PDebugLog](#)[CyU3PDebugLogFlush](#)**5.33.4.8 CyU3PReturnStatus\_t CyU3PDebugLogFlush( void )**

Flush any pending logs out of the internal buffers.

**Description**

All log messages are collected into internal memory buffers on the FX3 device, and then sent out to the target when the bufferU3P\_ERROR\_NOT\_0 J 0.3985]0 d 0 p(36-(b)20()-278(the)-278(imrbos78(message)15(.RROR\_NOT\_0 J committ]TJ 0 -11

**Return value** CY\_U3P\_SUCCESS - if the Debug log flush is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - if the debug module has not been initialized.

CY\_U3P\_ERROR\_INVALID\_CALLER - if called from an interrupt. See also [CyU3PDebugLog](#) [CyU3PDebugLogClear](#)

## See also

[CyU3PDebugPrint](#)

## Parameters

<i>sendPreamble</i>	CyFalse: Do not send preamble; CyTrue: Send preamble.
---------------------	---

## 5.33.4.10 CyU3PReturnStatus\_t CyU3PDebugPrint( uint8\_t priority, char \* message, ... )

Free form message logging function.

**Description**

This function can be used by the application code to log free form message strings, and causes the output message to be written to the UART immediately.

The function supports a subset of the output conversion specifications supported by the printf function. The 'c', 'd', 'u', 'x' and 's' conversion specifications are supported. There is no support for float or double formats, or for flags, precision or type modifiers.

**Note**

If the full functionality supported by printf such as all conversion specifications, flags, precision, type modifiers etc. are required; please use the standard C library functions to print the formatted message into a string, and then use this function to send the string out through the UART port.

**Return value**

CY\_U3P\_SUCCESS - if the Debug print is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - if the debug module has not been initialized.  
 CY\_U3P\_ERROR\_INVALID\_CALLER - if called from an interrupt.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the total length of the formatted string exceeds the debug buffer size.

## See also

[CyU3PDebugInit](#)  
[CyU3PDebugPreamble](#)

## Parameters

<i>priority</i>	Priority level for this message.
<i>message</i>	Format specifier string.

## 5.33.4.11 CyU3PReturnStatus\_t CyU3PDebugSetTimeout( uint32\_t timeout )

Set the timeout period for a debug log function.

**Description**

When multiple log messages are output back-to-back through the CyU3PDebugPrint or CyU3PDebugLog functions, it is possible that the log function is blocked by non-availability of data buffers. This normally only gets cleared up when all previously logged messages have been sent out of the FX3 device (printed through UART or read out from other socket). The default behavior of the Debug API is to block forever until a buffer is available.

This function is used to set a limit on the amount of time the CyU3PDebugPrint, CyU3PDebugLog or CyU3PDebugFlush API will wait for the data to be drained. If a timeout occurs, all current debug messages in the pipe will be cleared; so that new messages can be output.

**Return value**

CY\_U3P\_SUCCESS - if parameters are set properly.

CY\_U3P\_ERROR\_NOT\_STARTED - if the debug module has not been initialized.

#### See also

[CyU3PDebugPrint](#)  
[CyU3PDebugLog](#)  
[CyU3PDebugFlush](#)

#### Parameters

<i>timeout</i>	Timeout duration. Can be CYU3P_WAIT_FOREVER or CYU3P_NO_WAIT as well.
----------------	---

### 5.33.4.12 void CyU3PDebugSetTraceLevel ( *uint8\_t level* )

This function sets the priority threshold above which debug traces will be logged.

#### Description

The logger can suppress log messages that have a priority level lower than a user specified threshold. This function is used to set the priority threshold below which logs will be suppressed.

#### Return value

None

#### See also

[CyU3PDebugPrint](#)  
[CyU3PDebugLog](#)

#### Parameters

<i>level</i>	Lowest debug trace priority level that is enabled.
--------------	--

### 5.33.4.13 CyU3PReturnStatus\_t CyU3PDebugStringPrint ( *uint8\_t \* buffer, uint16\_t maxLength, char \* fmt, ...* )

Miniature version of sprintf routine.

#### Description

This is a mini version of the sprintf routine, which prints formatted debug information into a user provided buffer. The constraints mentioned for the CyU3PDebugPrint function apply for this function as well.

This function only supports the 'c', 'd', 'u', 'x' and 's' conversion specifications. It also does not support precision or type modifiers.

#### Return value

CY\_U3P\_SUCCESS - if the print to buffer is successful. CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the total length of the formatted string exceeds maxLength.

#### See also

[CyU3PDebugPrint](#)

#### Parameters

<i>buffer</i>	Buffer into which the data is to be printed.
<i>maxLength</i>	Buffer size.
<i>fmt</i>	Format specifier string.

#### 5.33.4.14 CyU3PReturnStatus\_t CyU3PDebugSysMemDeInit ( void )

Disables the SYS\_MEM logger.

##### Description

The function de-initializes the SYS\_MEM logger. The normal logger can now be initialized using CyU3PDebugInit call.

##### Return value

CY\_U3P\_SUCCESS - when successfully initialized.

CY\_U3P\_ERROR\_NOT\_STARTED - when the logger is not yet started.

##### See also

[CyU3PDebugSysMemInit](#)

#### 5.33.4.15 CyU3PReturnStatus\_t CyU3PDebugSysMemInit ( uint8\_t \* buffer, uint16\_t size, CyBool\_t isWrapAround, uint8\_t traceLevel )

Initializes the SYS\_MEM based logger facility.

##### Description

This function cannot be used when the CyU3PDebugInit is invoked. The SYS\_MEM logging is faster as the writes are done directly to the system memory. But the limitation is that only CyU3PDebugLog function can be used. The log buffer can be cleared by calling CyU3PDebugLogClear.

##### Return value

CY\_U3P\_SUCCESS - when successfully initialized.

CY\_U3P\_ERROR\_ALREADY\_STARTED - The logger is already started.

CY\_U3P\_ERROR\_NULL\_POINTER - If NULL buffer pointer is passed as argument.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - If any of the arguments passed is invalid.

##### See also

[CyU3PDebugSysMemDeInit](#)  
[CyU3PDebugLog](#)  
[CyU3PDebugLogClear](#)

##### Parameters

<i>buffer</i>	The buffer memory to be used for writing the log.
<i>size</i>	The size of memory available for logging.
<i>isWrapAround</i>	CyTrue - wrap up and start logging from beginning; CyFalse - Stop logging on reaching the last memory address.
<i>traceLevel</i>	Priority level beyond which logs should be output.

#### 5.33.4.16 CyU3PReturnStatus\_t CyU3PDeviceCacheControl ( CyBool\_t isICacheEnable, CyBool\_t isDCacheEnable, CyBool\_t isDmaHandleDCache )

This function initializes the caches on the ARM926 core.

##### Description

The function is expected to be called immediately after the CyU3PDeviceInit API. The function controls the cache handling in the system. By default (if this API is not called), both Instruction and Data caches are disabled.

It should be noted that once D-cache is enabled, all buffers used for DMA in the system has to be cache line aligned. This means that all DMA buffers have to be 32 byte aligned and 32 byte multiples. This is ensured by the CyU3P↔

DmaBufferAlloc function. Any buffer allocated outside of this function has to follow the 32 byte alignment / multiple rule. This rule is also applicable for all DMA buffer pointers passed to library APIs including the USB descriptor buffers assigned using CyU3PUsbSetDesc, data pointers provided to CyU3PUsbSendEP0Data, CyU3PUsbGetEP0Data, CyU3PUsbHostSendSetupRqt etc.

The isDmaHandleDCache determines whether the DMA APIs perform cache cleans and cache flushes internally. If this is CyFalse, then all cache cleans and flushes for buffers used with DMA APIs have to be done explicitly by the user. If this is CyTrue, then the DMA APIs will clean the cache lines for buffers used to send out data from the device and will flush the cache lines for buffers used to receive data.

It is recommended that the isDmaHandleDCache parameter be set to CyTrue if the Data cache is being enabled.

#### Return value

CY\_U3P\_SUCCESS - If the call is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if isDmaHandleDCache is set to true without enabling the D-cache.

#### See also

[CyU3PDeviceInit](#)

#### Parameters

<i>isICacheEnable</i>	Whether to enable the I-cache.
<i>isDCacheEnable</i>	Whether to enable the D-cache.
<i>isDmaHandleDCache</i>	Whether the DMA APIs should internally take care of cache handling.

#### 5.33.4.17 CyU3PReturnStatus\_t CyU3PDeviceConfigureIOMatrix ( CyU3PloMatrixConfig\_t \* *cfg\_p* )

Configures the IO matrix for the device.

#### Description

The FX3/FX3S devices have upto 61 IO pins that can be configured to function in a variety of modes. The functional mode for each of these pins (or groups of pins) should be set based on the desired system level functionality.

The processor port (P-port) of the device can be configured as a GPIF interface. The mode selection for this interface should be based on the means of connecting the external device / processor. Some pins may not be used depending upon the configuration used. The free pins can be used as GPIOs.

If a 16 bit GPIF interface is used, then the DQ[15:0], CTL[4:0] and PMODE pins are reserved for their default usage. CTL[12:5] are not locked and can be configured as GPIO. If any of the CTL[12:5] lines are used for GPIF interface they should not be configured as GPIOs. Similarly if some lines of CTL[4:0] are not used for GPIF they can be overridden to be GPIOs using the CyU3PDeviceGpioOverride call.

If 32 bit GPIF interface is used, all of DQ[31:0] is reserved for the GPIF interface.

Some devices in the family, such as the CYUSB3011 and CYUSB3013 devices, as well as the FX3S device; support only a 16 bit wide GPIF interface. The isDQ32Bit field should be set to false when using these devices.

The FX3S device can support upto two storage ports that can be connected to SD/MMC/SDIO peripherals. The first storage port (S0) is always available and can be configured in a 4 bit SD mode or in a 8 bit MMC mode. The second storage port (S1) shares pins with the serial peripheral interfaces (UART, SPI and I2S).

The serial peripheral interfaces have various configurations that can be used depending on the type of interfaces used. The default mode of operation has all the low performance (serial) peripherals enabled. I2C lines are not multiplexed and are available in all configurations except when used as GPIOs. If a peripheral is marked as not used (CyFalse), then those IO lines can be configured as GPIOs.

IOs that are not configured for peripheral interfaces can be used as GPIOs. This selection has to be explicitly made. Otherwise these lines will be configured as per their default function.

The GPIOs available are further classified as simple (set / get a value or receive a GPIO interrupt); or as complex (PWM, timer, counter etc). The selection of this is made via four 32-bit bitmasks where each IO is represented with

(1 << IO number).

Please refer to the device data sheet for the complete list of supported modes on each of the device interfaces.

It is recommended that this function be called immediately after the CyU3PDeviceInit call from the main () function; and that the IO matrix not be dynamically changed. However, this API can be invoked when the peripherals affected are disabled.

#### Return value

CY\_U3P\_SUCCESS - when the IO configuration is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if some configuration value is invalid.

CY\_U3P\_ERROR\_NOT\_SUPPORTED - if the part in use does not support any of the selected features.

#### See also

[CyU3PIoMatrixConfig\\_t](#)  
[CyU3PDeviceGpioOverride](#)  
[CyU3PDeviceGpioRestore](#)

#### Parameters

<i>cfg</i>  <i>p</i>	Pointer to Configuration parameters.
---	--------------------------------------

### 5.33.4.18 uint32\_t CyU3PDeviceGetCpuLoad( void )

Get the average CPU load from start of firmware operation.

#### Description

This API returns the average CPU load from start of firmware operation in terms of percentage points. The load is calculated using profiling information provided by the ThreadX RTOS, and is available only in profiling enabled builds of the firmware.

#### Return value

CPU load in percentage, if profiling is enabled. 0 if profiling is disabled.

### 5.33.4.19 uint32\_t CyU3PDeviceGetDriverLoad( void )

Get the CPU loading due to SDK internal driver threads.

#### Description

This API returns the average CPU load due to all of the FX3 SDK internal driver threads. The load is calculated using profiling information provided by the ThreadX RTOS, and is available only in profiling enabled builds of the firmware.

#### Return value

Driver CPU load in percentage, if profiling is enabled. 0 if profiling is disabled.

### 5.33.4.20 CyU3PPartNumber\_t CyU3PDeviceGetPartNumber( void )

This function is used to get the part number of the FX3 device in use.

#### Description

The EZ-USB FX3 family has multiple parts which support various sets of features. This function can be used to query the part number of the current device so as to check whether specific functionality is supported or not.

#### Return value

Part number of the FX3 device in use.

See also

[CyU3PPartNumber\\_t](#)

#### 5.33.4.21 CyU3PReturnStatus\_t CyU3PDeviceGetSysClkFreq ( uint32\_t \* freq )

This function returns the current SYS\_CLK frequency.

##### Description

This function can be used to identify the current SYS\_CLK frequency on the FX3 device. This should only be called after the CyU3PDeviceInit API has been called.

##### Return value

CY\_U3P\_SUCCESS - If the call succeeds.

CY\_U3P\_ERROR\_NULL\_POINTER - if NULL pointer was passed as parameter.

CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the device is not initialized.

See also

[CyU3PSysClockConfig\\_t](#)

##### Parameters

<i>freq</i>	Pointer to return the current SYS_CLK frequency.
-------------	--

#### 5.33.4.22 uint32\_t CyU3PDeviceGetThreadLoad ( CyU3PThread \* thread\_p )

Get the CPU loading due to a given thread.

##### Description

This API returns the average CPU loading due to a given thread, from start of firmware operation. The load is calculated using profiling information provided by the ThreadX RTOS, and is available only in profiling enabled builds of the firmware.

##### Return value

Thread CPU load in percentage, if profiling is enabled. 0 if profiling is disabled.

#### 5.33.4.23 CyU3PReturnStatus\_t CyU3PDeviceGpioOverride ( uint8\_t gpioId, CyBool\_t isSimple )

This function is used to override an IO as a GPIO.

##### Description

This is an override mechanism and can be used to enable any IO line as simple / complex GPIO. This should be done with caution as a wrong setting can cause damage to hardware.

The API is expected to be invoked before the corresponding peripheral module is enabled. The specific GPIO configuration has to be still done after this call.

The CyU3PDeviceConfigureIOMatrix API checks for validity of the configuration, whereas this API does not. Use this API with great caution.

##### Return value

CY\_U3P\_SUCCESS if the operation is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the GPIO specified is invalid.

See also

[CyU3PDeviceConfigureIOMatrix](#)  
[CyU3PDeviceGpioRestore](#)

**Parameters**

<i>gpioId</i>	The IO to be converted to GPIO.
<i>isSimple</i>	CyTrue: simple GPIO, CyFalse: complex GPIO

**5.33.4.24 CyU3PReturnStatus\_t CyU3PDeviceGpioRestore ( uint8\_t gpioId )**

This function is used to restore IO to normal mode.

**Description**

IOs that are overridden to be GPIO can be restored to normal mode of operation by invoking this API.

**Return value**

CY\_U3P\_SUCCESS if the operation is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the GPIO specified is invalid.

**See also**

[CyU3PDeviceGpioOverride](#)  
[CyU3PDeviceConfigureIOMatrix](#)

**Parameters**

<i>gpioId</i>	The GPIO to be restored.
---------------	--------------------------

**5.33.4.25 CyU3PReturnStatus\_t CyU3PDeviceInit ( CyU3PSysClockConfig\_t \* clkCfg )**

This function initializes the device.

**Description**

This function initializes the FX3 device by setting up the clocks for the CPU, DMA and register accesses; and then initializing the Vectored Interrupt Controller (VIC). This function also restores the state of the GPIO block if it is called as part of the resumption from low power standby mode.

This function is expected to be the first one called from the main () function, and should be called only once.

**Return value**

CY\_U3P\_SUCCESS - If the call is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the parameters are invalid.

**See also**

[CyU3PSysClockConfig\\_t](#)

**Parameters**

<i>clkCfg</i>	The clock configuration for CPU, DMA and register (MMIO) access. The default configuration (CPU divider = 2, DMA divider = 2, MMIO divider = 2, setSysClk400 = CyFalse, useStandbyClk = CyTrue) is selected if the clkCfg parameter is set to NULL.
---------------	---

**5.33.4.26 void CyU3PDeviceReset ( CyBool\_t isWarmReset )**

This function resets the FX3 device.

**Description**

This function is used to reset the FX3 device as a whole. Only a whole system reset is supported, as a CPU only reset will leave hardware blocks within the device in an inconsistent state.

If the warm boot option is selected, the firmware in the FX3 RAM shall be maintained; otherwise it shall be discarded and freshly loaded. If the warm boot option is used, the firmware should be capable of initializing any global variables explicitly.

**Note**

It is recommended that all of the FX3 blocks are de-initialized before triggering a warm boot operation.

**Return value**

None as this function does not return.

**Parameters**

<i>isWarmReset</i>	Whether this should be a warm reset or a cold reset.
--------------------	--

**5.33.4.27 void CyU3PFirmwareEntry( void )**

This is the entry routine for the firmware.

**Description**

This function should be set as the entry routine for the firmware by choosing the appropriate linker settings. This function is defined by the FX3 API library and sets up the runtime stacks that are required for the proper C/C++ language code to execute.

Once all of the initial CPU/memory initialization is completed, this function will call the user define CyU3PToolChainInit function to return control to the user application.

**Return value**

None

**See also**

[CyU3PToolChainInit](#)

**5.33.4.28 CyBool\_t CyU3PIsGpioComplexIOConfigured( uint32\_t gpioid )**

Check whether a specific pin has been selected as a complex GPIO.

**Description**

This function checks whether a specific FX3 pin has been selected to function as a complex GPIO.

**Return value**

CyTrue if the pin is selected as a complex GPIO.

CyFalse is the pin is not selected as a complex GPIO.

**See also**

[CyU3PDeviceConfigureIOMatrix](#)

[CyU3PIsGpioSimpleIOConfigured](#)

**Parameters**

<i>gpioid</i>	Pin number to be queried.
<i>Id</i>	

### 5.33.4.29 CyBool\_t CyU3PIsGpioSimpleIOConfigured ( uint32\_t gpioid )

Check whether a specific pin has been selected as a simple GPIO.

#### Description

This function checks whether a specific FX3 pin has been selected to function as a simple GPIO.

#### Return value

CyTrue if the pin is selected as a simple GPIO.

CyFalse if the pin is not selected as a simple GPIO.

#### See also

[CyU3PDeviceConfigureIOMatrix](#)  
[CyU3PIsGpioComplexIOConfigured](#)

#### Parameters

<i>gpiod</i>	Pin number to be queried.
<i>Id</i>	

### 5.33.4.30 CyBool\_t CyU3PIsGpioValid ( uint8\_t gpioid )

Check whether a specified GPIO ID is valid on the current FX3 part.

#### Description

Different parts in the FX3 family support different numbers of GPIOs. This function is used to check whether a specific GPIO number is valid on the current part.

#### Return value

CyTrue if the GPIO number is valid and usable.

CyFalse if the GPIO number is invalid.

#### Parameters

<i>gpiod</i>	GPIO number to be checked for validity.
<i>Id</i>	

### 5.33.4.31 CyBool\_t CyU3PIsLppIOConfigured ( CyU3PLppModule\_t lppModule )

Check whether a specific serial peripheral has been enabled in the IO Matrix.

#### Description

This function checks whether a specific serial peripheral interface has been enabled in the IO matrix.

#### Return value

CyTrue if the specified LPP block is enabled.

CyFalse if the specified LPP block is not enabled.

#### See also

[CyU3PDeviceConfigureIOMatrix](#)  
[CyU3PLppModule\\_t](#)

#### Parameters

<i>lppModule</i>	Serial interface to be queried.
------------------	---------------------------------

### 5.33.4.32 void CyU3PJumpToAddress ( *uint32\_t address* )

Jump to the specified instruction address.

#### Description

This function transfers control to the specified instruction address, without updating the runtime stack or the link register. This is intended to be used as the transition to a different application firmware binary. It is expected that the user would have disabled all interrupts and shut down the device blocks prior to calling this function.

#### Parameters

<i>address</i>	Address to jump to.
----------------	---------------------

### 5.33.4.33 CyU3PReturnStatus\_t CyU3PSetSerialIODriveStrength ( *CyU3PDriveStrengthState\_t driveStrength* )

Set the IO drive strength for UART, SPI and I2S interfaces.

#### Description

This function is used to update the drive strength for the UART, SPI and I2S interface signals. This is set to CY\_U3P\_DS\_THREE\_QUARTER\_STRENGTH by default.

#### Return value

CY\_U3P\_SUCCESS - If the operation is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - If the Drive strength requested is invalid.

#### See also

[CyU3PDriveStrengthState\\_t](#)

[CyU3PSetI2cDriveStrength](#)

#### Parameters

<i>driveStrength</i>	Desired serial IO drive strength.
----------------------	-----------------------------------

### 5.33.4.34 CyU3PReturnStatus\_t CyU3PSysCheckStandbyParam ( *uint16\_t wakeupFlags*, *uint16\_t polarity*, *uint8\_t \* bkp\_buff\_p* )

Verify FX3 device is ready to enter in low power standby mode.

#### Description

The function will verify whether FX3 device is ready for putting into low power standby mode, by verifying whether wakeup sources configured are correct and all the peripheral blocks have been disabled.

This function can be called to check whether the device can be placed in the low power standby state. If it is not called by the user, this will be called internally within the CyU3PSysEnterSuspendMode API.

#### Return value

CY\_U3P\_SUCCESS - if the call was successful.

CY\_U3P\_ERROR\_INVALID\_CALLER - if called from interrupt context.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the wakeup sources specified are invalid.

CY\_U3P\_ERROR\_ABORTED - if one of the wakeup source is already triggering.

#### See also

[CyU3PSysEnterSuspendMode](#)

### 5.33.4.35 CyU3PReturnStatus\_t CyU3PSysCheckSuspendParams ( uint16\_t *wakeupFlags*, uint16\_t *polarity* )

Verify FX3 device is ready to enter in low power suspend mode.

#### Description

The function will verify whether FX3 device is ready for putting into low power suspend mode, as well as prepare device low power state entry plan.

This function needs to be called prior to calling CyU3PSysEnterSuspendMode, for making device enter into suspend mode.

#### Return value

CY\_U3P\_SUCCESS - if the call was successful.

CY\_U3P\_ERROR\_INVALID\_CALLER - if called from interrupt context.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the wakeup sources specified are invalid.

CY\_U3P\_ERROR\_ABORTED - if one of the wakeup source is already triggering.

#### See also

[CyU3PSysEnterSuspendMode](#)

### 5.33.4.36 CyU3PReturnStatus\_t CyU3PSysEnterStandbyMode ( uint16\_t *wakeupFlags*, uint16\_t *polarity*, uint8\_t \* *bkp\_buff\_p* )

Places the FX3 device in low power standby mode.

#### Description

This function places the FX3 device in low power standby mode where the power consumption on the device is lowest. As the power to all the device blocks is removed when in standby mode; this function can only be called after shutting down (de-initializing) all FX3 blocks such as USB, GPIF, UART, I2C etc. Only the GPIO block can be left on, and its state will be restored when the device wakes up.

On wakeup from standby, the device starts firmware execution from the original entry point. In this sense, the entry into and wakeup from standby mode is similar to a warm reset being applied to the FX3 device.

Only P-Port, UART CTS and VBus based wakeup sources are supported to trigger a wake up of the FX3 device from standby mode. While the FX3 System RAM retains its content, the TCM blocks lose power while the device is in standby. This API backs up the content of the I-TCM region into a user specified buffer location before entering standby mode. The I-TCM contents are then automatically restored during re-initialization of the FX3 device.

#### Return value

No return if the device is actually placed in standby mode.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the wakeup sources, polarity or buffer pointer provided is invalid.

CY\_U3P\_ERROR\_INVALID\_SEQUENCE if this API is called while any of the FX3 blocks is still active.

CY\_U3P\_ERROR\_STANDBY\_FAILED if the specified wakeup sources are already active.

#### See also

[CyU3PSysEnterSuspendMode](#)

#### Parameters

<i>wakeupFlags</i>	List of selected wakeup sources from standby.
<i>polarity</i>	Polarity of the wakeup source which causes the device to wakeup.
<i>bkp_buff_p</i>	Pointer to buffer where the I-TCM content can be backed up. Should be located in the SYSMEM and be able to hold 18 KB of data.

5.33.4.37 `CyU3PReturnStatus_t CyU3PSysEnterSuspendMode ( uint16_t wakeupFlags, uint16_t polarity, uint16_t * wakeupSource )`

Places the FX3 device in low power suspend mode.

#### Description

The function can be called only after initializing the device completely. The device will enter into the suspended mode until any of the wakeup sources are triggered. This function does not return until the device has already resumed normal operation.

The CPU stops running and the device enters a low power state. Any combination of CY\_U3P\_SYS\_PPOT\_WAKEUP\_SRC\_EN, CY\_U3P\_SYS\_USB\_WAKEUP\_SRC\_EN and CY\_U3P\_SYS\_UART\_WAKEUP\_SRC\_EN can be used as the wakeup trigger.

This function does not affect the state of the USB PHY on the FX3 device. If the USB PHY is to be powered off while in the low power mode, the caller should explicitly call the CyU3PConnectState API to do this.

#### Note

If the watchdog is enabled with an external 32 KHz clock input to the FX3 device, the watchdog timer will still be operational when the FX3 device is in the suspend mode with clocks disabled. In this case, the FX3 device will get reset if the device stays in suspend mode for longer than the reset period specified. If the 32 KHz clock input is not provided, or the useStandbyClk parameter to CyU3PDeviceInit is set to CyFalse, the watchdog timer will be off when the device is in suspend mode. In such cases, the device will not get reset while it is in the suspend mode.

#### Return value

`CY_U3P_SUCCESS` - if the call was successful.  
`CY_U3P_ERROR_INVALID_CALLER` - if called from interrupt context.  
`CY_U3P_ERROR_BAD_ARGUMENT` - if the wakeup sources specified are invalid.  
`CY_U3P_ERROR_ABORTED` - if one of the wakeup source is already triggering.

#### See also

[CyU3PSysEnterStandbyMode](#)  
[CyU3PDeviceInit](#)  
[CyU3PSysWatchDogConfigure](#)

#### Parameters

<code>wakeupFlags</code>	Bit mask representing the wakeup sources that are allowed to bring FX3 out of suspend mode.
<code>polarity</code>	Polarity for each of the Wakeup Sources. This field is valid only for the CY_U3P_SYS_UART_WAKEUP_SRC and CY_U3P_SYS_USB_VBUS_WAKEUP_SRC wakeup sources. 0 - Wakeup when the corresponding source goes low. 1 - Wakeup when the corresponding source goes high.
<code>wakeupSource</code>	Output parameter indicating the source responsible for waking the FX3 from the Suspend mode.

5.33.4.38 `CyU3PReturnStatus_t CyU3PSysGetApiVersion ( uint16_t * majorVersion, uint16_t * minorVersion, uint16_t * patchNumer, uint16_t * buildNumer )`

This function returns the API library version.

#### Description

This function is used to identify the version information for the FX3 API library. It is expected that the [cyfxversion.h](#) file corresponding to the actual library version will be used by the application.

Valid pointers need to be passed for each of the return parameters, only if the corresponding return value is required. Otherwise, NULL can be passed.

**Return value**

CY\_U3P\_SUCCESS - If the call is successful.

**Parameters**

<i>majorVersion</i>	Major version number for the release
<i>minorVersion</i>	Minor version number for the release
<i>patchNumer</i>	Patch version for the release
<i>buildNumer</i>	The build number for the release

**5.33.4.39 void CyU3PSysSendEnterSuspendStatus ( void )**

Wait for P-Port to receive last mailbox response from FX3 device prior to entering in suspend mode.

**Description**

This function will make FX3 device to wait for 100 mseconds, after sending mailbox response to P-Port indicating that system is entering into suspend mode.

This function needs to be called prior to calling CyU3PSysEnterSuspendMode, if any external processor is waiting for enter suspend mailbox response.

**See also**

[CyU3PSysEnterSuspendMode](#)

**5.33.4.40 void CyU3PSystemRegisterDriver ( CyU3PThread \* *thread\_p* )**

Internal function used to register driver threads.

**Description**

This API is used by SDK internal drivers to register all the driver threads. Please do not call this API from user code.

**5.33.4.41 void CyU3PSysWatchDogClear ( void )**

Clear the watchdog timer to prevent a device reset.

**Description**

This function is used to clear the watchdog timer so as to prevent it from resetting the FX3 device. This function should be called more frequently than the specified watchdog timer period to avoid unexpected resets.

**Return value**

None

**See also**

[CyU3PSysWatchDogConfigure](#)

**5.33.4.42 void CyU3PSysWatchDogConfigure ( CyBool\_t *enable*, uint32\_t *period* )**

Configure the watchdog reset control.

**Description**

The FX3 device implements a watchdog timer that can be used to reset the device when it is not responsive. This function is used to enable the watchdog feature and to set the period for this watchdog timer.

**Note**

If the watchdog is enabled with an external 32 KHz clock input to the FX3 device, the watchdog timer will still be operational when the FX3 device is in the suspend mode with clocks disabled. In this case, the FX3 device will get reset if the device stays in suspend mode for longer than the reset period specified. If the 32 KHz clock input is not provided, or the useStandbyClk parameter to CyU3PDeviceInit is set to CyFalse, the watchdog timer will be off when the device is in suspend mode. In such cases, the device will not get reset while it is in the suspend mode.

**Return value**

None

## See also

[CyU3PSysWatchDogClear](#)  
[CyU3PDeviceInit](#)  
[CyU3PSysEnterSuspendMode](#)

## Parameters

<i>enable</i>	Whether the watch dog should be enabled.
<i>period</i>	Period in milliseconds. Is valid only for enable calls.

## 5.33.4.43 void CyU3PToolChainInit( void )

This is the normal entry routine provided by the tool-chain.

**Description**

This routine maps to the start-up code created by the compiler tool-chain. The FX3 firmware will jump to this function after completing the initial CPU, memory and device initialization.

The FX3 SDK includes a version of this function for the GNU ARM tool-chain. This implementation only clears the BSS segment and then jumps to the main function.

If the user application requires a heap (uses malloc / new), it should be initialized in this function.

**Return value**

None

## See also

[CyU3PFirmwareEntry](#)

## 5.34 firmware/u3p\_firmware/inc/cyu3types.h File Reference

This file contains definitions for the basic data types used by the FX3 firmware library. If a compiler provided version is available, this can be used instead of the custom ones, by disabling the CYU3\_DEFINE\_BASIC\_TYPES definition.

```
#include <stdint.h>
```

## Macros

- #define [CyTrue](#) (1)
- #define [CyFalse](#) (0)

## Typedefs

- `typedef volatile unsigned char uvint8_t`
- `typedef volatile unsigned short uvint16_t`
- `typedef volatile unsigned long uvint32_t`
- `typedef int CyBool_t`
- `typedef unsigned int CyU3PReturnStatus_t`

*Return status from FX3 APIs.*

### 5.34.1 Detailed Description

This file contains definitions for the basic data types used by the FX3 firmware library. If a compiler provided version is available, this can be used instead of the custom ones, by disabling the CYU3\_DEFINE\_BASIC\_TYPES definition.

### 5.34.2 Macro Definition Documentation

#### 5.34.2.1 `#define CyFalse (0)`

False value.

#### 5.34.2.2 `#define CyTrue (1)`

Truth value.

### 5.34.3 Typedef Documentation

#### 5.34.3.1 `typedef int CyBool_t`

Boolean data type.

#### 5.34.3.2 `typedef unsigned int CyU3PReturnStatus_t`

Return status from FX3 APIs.

##### Description

This is a custom data type used for return status from all FX3 APIs. The specific error codes that can be returned by these APIs are listed in [cyu3error.h](#).

##### See also

[CyU3PErrorCode\\_t](#)

#### 5.34.3.3 `typedef volatile unsigned short uvint16_t`

Volatile 16-bit unsigned value. This is not used in the library.

#### 5.34.3.4 `typedef volatile unsigned long uvint32_t`

Volatile 32-bit unsigned value. Used to represent FX3 device registers.

### 5.34.3.5 `typedef volatile unsigned char uvint8_t`

Volatile 8-bit unsigned value. This is not used in the library.

## 5.35 firmware/u3p\_firmware/inc/cyu3uart.h File Reference

The UART driver in FX3 firmware is responsible for handling data transfers through the UART interface on the device. This file defines the data structures and APIs for UART interface management.

```
#include "cyu3types.h"
#include "cyu3lpp.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

### Data Structures

- struct `CyU3PUartConfig_t`  
*Configuration parameters for the UART interface.*

### Macros

- #define `CY_U3P_UART_DEFAULT_LOCK_TIMEOUT` (CYU3P\_WAIT\_FOREVER)  
*Delay duration to wait to get a lock on the UART block.*

### TypeDefs

- typedef enum `CyU3PUartEvt_t` `CyU3PUartEvt_t`  
*List of UART related event types.*
- typedef enum `CyU3PUartError_t` `CyU3PUartError_t`  
*List of UART specific error/status codes.*
- typedef enum `CyU3PUartBaudrate_t` `CyU3PUartBaudrate_t`  
*List of baud rates supported by the UART.*
- typedef enum `CyU3PUartStopBit_t` `CyU3PUartStopBit_t`  
*List of number of stop bits to be used in UART communication.*
- typedef enum `CyU3PUartParity_t` `CyU3PUartParity_t`  
*List of parity settings supported by the UART interface.*
- typedef struct `CyU3PUartConfig_t` `CyU3PUartConfig_t`  
*Configuration parameters for the UART interface.*
- typedef void(\*) `CyU3PUartIntrCb_t`) (`CyU3PUartEvt_t` evt, `CyU3PUartError_t` error)  
*Prototype of UART event callback function.*

### Enumerations

- enum `CyU3PUartEvt_t` { `CY_U3P_UART_EVENT_RX_DONE` = 0, `CY_U3P_UART_EVENT_TX_DONE`, `CY_U3P_UART_EVENT_ERROR`, `CY_U3P_UART_EVENT_RX_DATA` }  
*List of UART related event types.*
- enum `CyU3PUartError_t` {  
`CY_U3P_UART_ERROR_NAK_BYTE_0` = 0, `CY_U3P_UART_ERROR_RX_PARITY_ERROR` = 1, `CY_U3P_UART_ERROR_TX_OVERFLOW` = 12, `CY_U3P_UART_ERROR_RX_UNDERFLOW` = 13, `CY_U3P_UART_ERROR_RX_OVERFLOW` = 14 }

*List of UART specific error/status codes.*

- enum `CyU3PUartBaudrate_t` {
   
CY\_U3P\_UART\_BAUDRATE\_100 = 100, CY\_U3P\_UART\_BAUDRATE\_300 = 300, CY\_U3P\_UART\_BAUDRATE\_600 = 600, CY\_U3P\_UART\_BAUDRATE\_1200 = 1200,
   
CY\_U3P\_UART\_BAUDRATE\_2400 = 2400, CY\_U3P\_UART\_BAUDRATE\_4800 = 4800, CY\_U3P\_UART\_BAUDRATE\_9600 = 9600, CY\_U3P\_UART\_BAUDRATE\_10000 = 10000,
   
CY\_U3P\_UART\_BAUDRATE\_14400 = 14400, CY\_U3P\_UART\_BAUDRATE\_19200 = 19200, CY\_U3P\_UART\_BAUDRATE\_38400 = 38400, CY\_U3P\_UART\_BAUDRATE\_50000 = 50000,
   
CY\_U3P\_UART\_BAUDRATE\_57600 = 57600, CY\_U3P\_UART\_BAUDRATE\_75000 = 75000, CY\_U3P\_UART\_BAUDRATE\_100000 = 100000, CY\_U3P\_UART\_BAUDRATE\_115200 = 115200,
   
CY\_U3P\_UART\_BAUDRATE\_153600 = 153600, CY\_U3P\_UART\_BAUDRATE\_200000 = 200000, CY\_U3P\_UART\_BAUDRATE\_225000 = 225000, CY\_U3P\_UART\_BAUDRATE\_230400 = 230400,
   
CY\_U3P\_UART\_BAUDRATE\_300000 = 300000, CY\_U3P\_UART\_BAUDRATE\_400000 = 400000, CY\_U3P\_UART\_BAUDRATE\_460800 = 460800, CY\_U3P\_UART\_BAUDRATE\_500000 = 500000,
   
CY\_U3P\_UART\_BAUDRATE\_750000 = 750000, CY\_U3P\_UART\_BAUDRATE\_921600 = 921600, CY\_U3P\_UART\_BAUDRATE\_1M = 1000000, CY\_U3P\_UART\_BAUDRATE\_2M = 2000000,
   
CY\_U3P\_UART\_BAUDRATE\_3M = 3000000, CY\_U3P\_UART\_BAUDRATE\_4M = 4000000, CY\_U3P\_UART\_BAUDRATE\_4M608K = 4608000
 }

*List of baud rates supported by the UART.*

- enum `CyU3PUartStopBit_t` { CY\_U3P\_UART\_ONE\_STOP\_BIT = 1, CY\_U3P\_UART\_TWO\_STOP\_BIT = 2 }

*List of number of stop bits to be used in UART communication.*

- enum `CyU3PUartParity_t` { CY\_U3P\_UART\_NO\_PARITY = 0, CY\_U3P\_UART\_EVEN\_PARITY, CY\_U3P\_UART\_ODD\_PARITY, CY\_U3P\_UART\_NUM\_PARITY }

*List of parity settings supported by the UART interface.*

## Functions

- `CyU3PReturnStatus_t CyU3PUartInit (void)`
  - Starts the UART block on the FX3 device.*
- `CyU3PReturnStatus_t CyU3PUartDeInit (void)`
  - Stops the UART hardware block.*
- `CyU3PReturnStatus_t CyU3PUartSetConfig (CyU3PUartConfig_t *config, CyU3PUartIntrCb_t cb)`
  - Sets the UART interface parameters.*
- `CyU3PReturnStatus_t CyU3PUartTxSetBlockXfer (uint32_t txSize)`
  - Sets the number of bytes to be transmitted by the UART.*
- `CyU3PReturnStatus_t CyU3PUartRxSetBlockXfer (uint32_t rxSize)`
  - Sets the number of bytes to be received by the UART.*
- `uint32_t CyU3PUartTransmitBytes (uint8_t *data_p, uint32_t count, CyU3PReturnStatus_t *status)`
  - Transmit data through the UART interface in register mode.*
- `uint32_t CyU3PUartReceiveBytes (uint8_t *data_p, uint32_t count, CyU3PReturnStatus_t *status)`
  - Receive data from the UART interface in register mode.*
- `void CyU3PRegisterUartCallBack (CyU3PUartIntrCb_t uartIntrCb)`
  - This function registers a callback function for notification of UART interrupts.*
- `CyU3PReturnStatus_t CyU3PUartSetTimeout (uint32_t readLoopCnt, uint32_t writeLoopCnt)`
  - Set the timeout for UART byte-mode transmit and receive operations.*

### 5.35.1 Detailed Description

The UART driver in FX3 firmware is responsible for handling data transfers through the UART interface on the device. This file defines the data structures and APIs for UART interface management.

## 5.35.2 Typedef Documentation

### 5.35.2.1 `typedef enum CyU3PUartBaudrate_t CyU3PUartBaudrate_t`

List of baud rates supported by the UART.

#### Description

This enumeration lists the various baud rate settings that are supported by the UART interface and driver implementation. The specific baud rates achieved will be close approximations of these standard values based on the clock frequencies that can be obtained on the FX3 hardware. The actual baud rate achieved for SYS\_CLK settings of 403.2MHz and 416MHz is also listed below.

See also

[CyU3PUartConfig\\_t](#)

### 5.35.2.2 `typedef struct CyU3PUartConfig_t CyU3PUartConfig_t`

Configuration parameters for the UART interface.

#### Description

This structure defines all of the configurable parameters for the UART interface such as baud rate, stop bits, parity bits etc. A pointer to this structure is passed in to the CyU3PUartSetConfig function to configure the UART interface.

The isDma member specifies whether the UART should be configured to transfer data one byte at a time, or in terms of larger blocks.

All of the parameters can be changed dynamically by calling the CyU3PUartSetConfig function repeatedly.

See also

[CyU3PUartBaudrate\\_t](#)  
[CyU3PUartStopBit\\_t](#)  
[CyU3PUartParity\\_t](#)  
[CyU3PUartSetConfig](#)

### 5.35.2.3 `typedef enum CyU3PUartError_t CyU3PUartError_t`

List of UART specific error/status codes.

#### Description

This type lists the various UART specific error/status codes that are sent to the event callback as event data associated with CY\_U3P\_UART\_ERROR\_EVT events.

See also

[CyU3PUartEvt\\_t](#)  
[CyU3PUartIntrCb\\_t](#)

### 5.35.2.4 `typedef enum CyU3PUartEvt_t CyU3PUartEvt_t`

List of UART related event types.

#### Description

This enumeration lists the various UART related event codes that are notified to the user application through an event callback.

See also

[CyU3PUartError\\_t](#)  
[CyU3PUartIntrCb\\_t](#)

### 5.35.2.5 `typedef void(* CyU3PUartIntrCb_t)(CyU3PUartEvt_t evt, CyU3PUartError_t error)`

Prototype of UART event callback function.

#### Description

This function type defines a callback to be called when UART interrupts are received. A function of this type can be registered with the UART driver as a callback function and will be called whenever an event of interest occurs.

The UART has to be configured for DMA mode of transfer for callbacks to be registered.

#### See also

[CyU3PUartEvt\\_t](#)  
[CyU3PUartError\\_t](#)  
[CyU3PRegisterUartCallBack](#)

### 5.35.2.6 `typedef enum CyU3PUartParity_t CyU3PUartParity_t`

List of parity settings supported by the UART interface.

#### Description

This enumeration lists the various parity settings that the UART interface can be configured to support.

#### See also

[CyU3PUartConfig\\_t](#)

### 5.35.2.7 `typedef enum CyU3PUartStopBit_t CyU3PUartStopBit_t`

List of number of stop bits to be used in UART communication.

#### Description

This enumeration lists the various number of stop bit settings that the UART interface can be configured to have.

#### See also

[CyU3PUartConfig\\_t](#)

## 5.35.3 Enumeration Type Documentation

### 5.35.3.1 `enum CyU3PUartBaudrate_t`

List of baud rates supported by the UART.

#### Description

This enumeration lists the various baud rate settings that are supported by the UART interface and driver implementation. The specific baud rates achieved will be close approximations of these standard values based on the clock frequencies that can be obtained on the FX3 hardware. The actual baud rate achieved for SYS\_CLK settings of 403.2MHz and 416MHz is also listed below.

#### See also

[CyU3PUartConfig\\_t](#)

#### Enumerator

**CY\_U3P\_UART\_BAUDRATE\_100** Baud: 100, Actual @403MHz: 100.00, @416MHz: 100.00,

**CY\_U3P\_UART\_BAUDRATE\_300** Baud: 300, Actual @403MHz: 300.00, @416MHz: 300.01,

**CY\_U3P\_UART\_BAUDRATE\_600** Baud: 600, Actual @403MHz: 600.00, @416MHz: 599.99

**CY\_U3P\_UART\_BAUDRATE\_1200** Baud: 1200, Actual @403MHz: 1200.00, @416MHz: 1200.01  
**CY\_U3P\_UART\_BAUDRATE\_2400** Baud: 2400, Actual @403MHz: 2400.00, @416MHz: 2399.96  
**CY\_U3P\_UART\_BAUDRATE\_4800** Baud: 4800, Actual @403MHz: 4800.00, @416MHz: 4800.15  
**CY\_U3P\_UART\_BAUDRATE\_9600** Baud: 9600, Actual @403MHz: 9600.00, @416MHz: 9599.41  
**CY\_U3P\_UART\_BAUDRATE\_10000** Baud: 10000, Actual @403MHz: 10000.00, @416MHz: 10000.00  
**CY\_U3P\_UART\_BAUDRATE\_14400** Baud: 14400, Actual @403MHz: 14400.00, @416MHz: 14400.44  
**CY\_U3P\_UART\_BAUDRATE\_19200** Baud: 19200, Actual @403MHz: 19200.00, @416MHz: 19202.36  
**CY\_U3P\_UART\_BAUDRATE\_38400** Baud: 38400, Actual @403MHz: 38385.38, @416MHz: 38404.73  
**CY\_U3P\_UART\_BAUDRATE\_50000** Baud: 50000, Actual @403MHz: 50000.00, @416MHz: 50000.00  
**CY\_U3P\_UART\_BAUDRATE\_57600** Baud: 57600, Actual @403MHz: 57600.00, @416MHz: 57585.83  
**CY\_U3P\_UART\_BAUDRATE\_75000** Baud: 75000, Actual @403MHz: 75000.00, @416MHz: 75036.08  
**CY\_U3P\_UART\_BAUDRATE\_100000** Baud: 100000, Actual @403MHz: 100000.00, @416MHz: 100000.00  
**CY\_U3P\_UART\_BAUDRATE\_115200** Baud: 115200, Actual @403MHz: 115068.49, @416MHz: 115299.33  
**CY\_U3P\_UART\_BAUDRATE\_153600** Baud: 153600, Actual @403MHz: 153658.54, @416MHz: 153392.33  
**CY\_U3P\_UART\_BAUDRATE\_200000** Baud: 200000, Actual @403MHz: 200000.00, @416MHz: 200000.00  
**CY\_U3P\_UART\_BAUDRATE\_225000** Baud: 225000, Actual @403MHz: 225000.00, @416MHz: 225108.23  
**CY\_U3P\_UART\_BAUDRATE\_230400** Baud: 230400, Actual @403MHz: 230136.99, @416MHz: 230088.50  
**CY\_U3P\_UART\_BAUDRATE\_300000** Baud: 300000, Actual @403MHz: 300000.00, @416MHz: 300578.03  
**CY\_U3P\_UART\_BAUDRATE\_400000** Baud: 400000, Actual @403MHz: 400000.00, @416MHz: 400000.00  
**CY\_U3P\_UART\_BAUDRATE\_460800** Baud: 460800, Actual @403MHz: 462385.32, @416MHz: 460176.99  
**CY\_U3P\_UART\_BAUDRATE\_500000** Baud: 500000, Actual @403MHz: 499009.90, @416MHz: 500000.00  
**CY\_U3P\_UART\_BAUDRATE\_750000** Baud: 750000, Actual @403MHz: 752238.81, @416MHz: 753623.19  
**CY\_U3P\_UART\_BAUDRATE\_921600** Baud: 921600, Actual @403MHz: 916363.64, @416MHz: 928571.43  
**CY\_U3P\_UART\_BAUDRATE\_1M** Baud: 1000000, Actual @403MHz: 1008000.00, @416MHz: 1000000.00  
**CY\_U3P\_UART\_BAUDRATE\_2M** Baud: 2000000, Actual @403MHz: 2016000.00, @416MHz: 2000000.00  
**CY\_U3P\_UART\_BAUDRATE\_3M** Baud: 3000000, Actual @403MHz: 2964705.88, @416MHz: 3058823.52  
**CY\_U3P\_UART\_BAUDRATE\_4M** Baud: 4000000, Actual @403MHz: 3876923.08, @416MHz: 4000000.00  
**CY\_U3P\_UART\_BAUDRATE\_4M608K** Baud: 4608000, Actual @403MHz: 4581818.18, @416MHz: 4727272.72

### 5.35.3.2 enum CyU3PUartError\_t

List of UART specific error/status codes.

#### Description

This type lists the various UART specific error/status codes that are sent to the event callback as event data associated with CY\_U3P\_UART\_ERROR\_EVT events.

#### See also

[CyU3PUartEvt\\_t](#)  
[CyU3PUartIntrCb\\_t](#)

#### Enumerator

**CY\_U3P\_UART\_ERROR\_NAK\_BYTE\_0** Missing stop bit.  
**CY\_U3P\_UART\_ERROR\_RX\_PARITY\_ERROR** RX parity error.  
**CY\_U3P\_UART\_ERROR\_TX\_OVERFLOW** Overflow of FIFO during transmit operation.  
**CY\_U3P\_UART\_ERROR\_RX\_UNDERFLOW** Underflow in FIFO during receive/read operation.  
**CY\_U3P\_UART\_ERROR\_RX\_OVERFLOW** Overflow of FIFO during receive operation.

### 5.35.3.3 enum CyU3PUartEvt\_t

List of UART related event types.

#### Description

This enumeration lists the various UART related event codes that are notified to the user application through an event callback.

#### See also

[CyU3PUartError\\_t](#)  
[CyU3PUartIntrCb\\_t](#)

#### Enumerator

**CY\_U3P\_UART\_EVENT\_RX\_DONE** UART data receive operation is complete.  
**CY\_U3P\_UART\_EVENT\_TX\_DONE** UART data transmit operation is complete.  
**CY\_U3P\_UART\_EVENT\_ERROR** An error has been detected.  
**CY\_U3P\_UART\_EVENT\_RX\_DATA** Data is available in receive FIFO.

### 5.35.3.4 enum CyU3PUartParity\_t

List of parity settings supported by the UART interface.

#### Description

This enumeration lists the various parity settings that the UART interface can be configured to support.

#### See also

[CyU3PUartConfig\\_t](#)

#### Enumerator

**CY\_U3P\_UART\_NO\_PARITY** No parity bits.  
**CY\_U3P\_UART EVEN\_PARITY** Even parity.  
**CY\_U3P\_UART ODD\_PARITY** Odd parity.  
**CY\_U3P\_UART\_NUM\_PARITY** Number of parity enumerations.

### 5.35.3.5 enum CyU3PUartStopBit\_t

List of number of stop bits to be used in UART communication.

#### Description

This enumeration lists the various number of stop bit settings that the UART interface can be configured to have.

#### See also

[CyU3PUartConfig\\_t](#)

#### Enumerator

**CY\_U3P\_UART\_ONE\_STOP\_BIT** 1 stop bit.

**CY\_U3P\_UART\_TWO\_STOP\_BIT** 2 stop bits.

## 5.35.4 Function Documentation

### 5.35.4.1 void CyU3PRegisterUartCallBack ( CyU3PUartIntrCb\_t uartIntrCb )

This function registers a callback function for notification of UART interrupts.

#### Description

This function registers a callback function that will be called for notification of UART interrupts. This can also be done by the CyU3PUartSetConfig API.

#### Return value

None

#### See also

[CyU3PUartIntrCb\\_t](#)

[CyU3PUartSetConfig](#)

#### Parameters

<i>uartIntrCb</i>	Callback function pointer.
-------------------	----------------------------

### 5.35.4.2 CyU3PReturnStatus\_t CyU3PUartDelInit ( void )

Stops the UART hardware block.

#### Description

This function disables and powers off the UART hardware block on the device.

#### Return value

CY\_U3P\_SUCCESS - if the de-init was successful.

CY\_U3P\_ERROR\_NOT\_STARTED - if UART had not been initialized.

#### See also

[CyU3PUartInit](#)

### 5.35.4.3 CyU3PReturnStatus\_t CyU3PUartInit ( void )

Starts the UART block on the FX3 device.

**Description**

This function powers up the UART hardware block on the device and should be the first UART related function called by the application.

**Return value**

CY\_U3P\_SUCCESS - if the init was successful.

CY\_U3P\_ERROR\_ALREADY\_STARTED - if the UART block had been previously initialized.

CY\_U3P\_ERROR\_NOT\_CONFIGURED - if UART was not enabled during IO configuration.

## See also

[CyU3PUartDelInit](#)  
[CyU3PUartSetConfig](#)  
[CyU3PUartTransmitBytes](#)  
[CyU3PUartReceiveBytes](#)  
[CyU3PUartTxSetBlockXfer](#)  
[CyU3PUartRxSetBlockXfer](#)

**5.35.4.4 uint32\_t CyU3PUartReceiveBytes ( uint8\_t \* data\_p, uint32\_t count, CyU3PReturnStatus\_t \* status )**

Receive data from the UART interface in register mode.

**Description**

This function is used to read the specified number of bytes from the UART in register mode. This function can only be used if the UART has been configured for register (non-DMA) transfer mode.

**Return value**

Number of bytes that are successfully received.

## See also

[CyU3PUartSetConfig](#)  
[CyU3PUartTransmitBytes](#)

## Parameters

<i>data_p</i>	Pointer to location where the data read is to be placed.
<i>count</i>	Number of bytes to be received.
<i>status</i>	Status returned from the operation.

**5.35.4.5 CyU3PReturnStatus\_t CyU3PUartRxSetBlockXfer ( uint32\_t rxSize )**

Sets the number of bytes to be received by the UART.

**Description**

This function sets the size of the desired data reception through the UART. The value 0xFFFFFFFFU can be used to specify infinite or indefinite data reception.

This function is to be used when the UART is configured for DMA mode of transfer. If this function is called when the UART is configured in register mode, it will return with an error.

**Return value**

CY\_U3P\_SUCCESS - if the transfer size was set successfully.

CY\_U3P\_ERROR\_NOT\_CONFIGURED - if UART was not configured for DMA mode.

CY\_U3P\_ERROR\_MUTEX\_FAILURE - Failed to get a mutex lock on the UART block.

## See also

[CyU3PUartSetConfig](#)  
[CyU3PUartTxSetBlockXfer](#)

## Parameters

<i>rxSize</i>	Desired transfer size.
---------------	------------------------

5.35.4.6 CyU3PReturnStatus\_t CyU3PUartSetConfig ( *CyU3PUartConfig\_t \* config, CyU3PUartIntrCb\_t cb* )

Sets the UART interface parameters.

**Description**

This function configures the UART block with the desired user parameters such as transfer mode, baud rate etc. This function should be called repeatedly to make any change to the set of configuration parameters.

Baudrate calculation: The FX3 UART supports baud rates between 100 Hz and 4 MHz. The UART block needs to be clocked at 16X the external baud rate. Since the block clock is derived from the SYS\_CLK frequency through integer and half dividers, it is only possible to set the baud rate to an approximation of the desired value.

See the documentation for the *CyU3PUartBaudrate\_t* for a list of actual baud rates that are achieved for various settings.

**Return value**

*CY\_U3P\_SUCCESS* - if the configuration was set successfully.  
*CY\_U3P\_ERROR\_NOT\_STARTED* - if UART was not initialized.  
*CY\_U3P\_ERROR\_NULL\_POINTER* - if a NULL pointer is passed.  
*CY\_U3P\_ERROR\_BAD\_ARGUMENT* - if any of the parameters are invalid.  
*CY\_U3P\_ERROR\_MUTEX\_FAILURE* - Failed to get a mutex lock on the UART block.

## See also

[CyU3PUartConfig\\_t](#)  
[CyU3PUartIntrCb\\_t](#)  
[CyU3PUartInit](#)  
[CyU3PUartTransmitBytes](#)  
[CyU3PUartReceiveBytes](#)  
[CyU3PUartTxSetBlockXfer](#)  
[CyU3PUartRxSetBlockXfer](#)

## Parameters

<i>config</i>	Pointer to structure containing config information.
<i>cb</i>	UART callback function to be registered.

5.35.4.7 CyU3PReturnStatus\_t CyU3PUartSetTimeout ( *uint32\_t readLoopCnt, uint32\_t writeLoopCnt* )

Set the timeout for UART byte-mode transmit and receive operations.

**Description**

The timeout setting for the *CyU3PUartTransmitBytes* and *CyU3PUartReceiveBytes* operations is based on the number of loop iterations used while waiting for the transfer completion. Each loop iteration will require two times the period of the UART clock. The default timeout is set to 0xFFFF and can be changed using this API.

**Note**

The API keeps the CPU spinning until the data transfer is completed, or the specified timeout period has elapsed.

Therefore, it is not desirable to set a very large timeout for these operations.

#### Return value

CY\_U3P\_SUCCESS if the timeout setting is updated. CY\_U3P\_ERROR\_NOT\_STARTED if the UART module is not enabled.

#### See also

[CyU3PUartTransmitBytes](#)  
[CyU3PUartReceiveBytes](#)

#### Parameters

<i>readLoopCnt</i>	Number of status checks to be performed during read operations.
<i>writeLoopCnt</i>	Number of status checks to be performed during write operations.

### 5.35.4.8 uint32\_t CyU3PUartTransmitBytes ( *uint8\_t \* data\_p*, *uint32\_t count*, *CyU3PReturnStatus\_t \* status* )

Transmit data through the UART interface in register mode.

#### Description

This function is used to transfer the specified number of bytes out through the UART in register mode. This function can only be used if the UART has been configured for register (non-DMA) transfer mode.

#### Return value

Number of bytes that are successfully transferred.

#### See also

[CyU3PUartSetConfig](#)  
[CyU3PUartReceiveBytes](#)

#### Parameters

<i>data_p</i>	Pointer to the data to be transferred.
<i>count</i>	Number of bytes to be transferred.
<i>status</i>	Status returned from the operation.

### 5.35.4.9 CyU3PReturnStatus\_t CyU3PUartTxSetBlockXfer ( *uint32\_t txSize* )

Sets the number of bytes to be transmitted by the UART.

#### Description

This function sets the size of the desired data transmission through the UART. The value 0xFFFFFFFFU can be used to specify infinite or indefinite data transmission.

This function is to be used when the UART is configured for DMA mode of transfer. If this function is called when the UART is configured in register mode, it will return with an error.

#### Return value

CY\_U3P\_SUCCESS - if the transfer size was set successfully.

CY\_U3P\_ERROR\_NOT\_CONFIGURED - if UART was not configured for DMA mode.

CY\_U3P\_ERROR\_MUTEX\_FAILURE - Failed to get a mutex lock on the UART block.

**See also**

[CyU3PUartSetConfig](#)  
[CyU3PUartRxSetBlockXfer](#)

**Parameters**

<code>txSize</code>	Desired transfer size.
---------------------	------------------------

## 5.36 firmware/u3p\_firmware/inc/cyu3usb.h File Reference

The USB device mode driver on the FX3 device is responsible for handling USB 3.0 and 2.0 connections, and providing the API hooks for configuring device operations.

```
#include "cyu3types.h"
#include "cyu3usbconst.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

### Data Structures

- struct [CyU3PEpConfig\\_t](#)  
*Endpoint configuration information.*

### Macros

- #define [CY\\_U3P\\_USB\\_REQUEST\\_TYPE\\_MASK](#) (0x000000FF)  
*Mask to get the bmRequestType field from the USB setup request.*
- #define [CY\\_U3P\\_USB\\_REQUEST\\_TYPE\\_POS](#) (0)  
*Position of the bmRequestType field in the USB setup request.*
- #define [CY\\_U3P\\_USB\\_REQUEST\\_MASK](#) (0x0000FF00)  
*Mask to get the bRequest field from the USB setup request.*
- #define [CY\\_U3P\\_USB\\_REQUEST\\_POS](#) (8)  
*Position of the bRequest field in the USB setup request.*
- #define [CY\\_U3P\\_USB\\_VALUE\\_MASK](#) (0xFFFF0000)  
*Mask to get the wValue field from the USB setup request.*
- #define [CY\\_U3P\\_USB\\_VALUE\\_POS](#) (16)  
*Position of the wValue field in the USB setup request.*
- #define [CY\\_U3P\\_USB\\_INDEX\\_MASK](#) (0x0000FFFF)  
*Mask to get the wIndex field from the USB setup request.*
- #define [CY\\_U3P\\_USB\\_INDEX\\_POS](#) (0)  
*Position of the wIndex field in the USB setup request.*
- #define [CY\\_U3P\\_USB\\_LENGTH\\_MASK](#) (0xFFFF0000)  
*Mask to get the wLength field from the USB setup request.*
- #define [CY\\_U3P\\_USB\\_LENGTH\\_POS](#) (16)  
*Position of the wLength field in the USB setup request.*
- #define [CY\\_U3P\\_MAX\\_STRING\\_DESC\\_INDEX](#) (16)  
*Number of string descriptors that are supported by the USB driver.*

## Typedefs

- **typedef enum CyU3PUsbEventType\_t CyU3PUsbEventType\_t**  
*List of USB related events that are raised by the USB manager.*
- **typedef enum CyU3PUsbMgrStates\_t CyU3PUsbMgrStates\_t**  
*List of USB device manager states.*
- **typedef enum CyU3PUSBSetDescType\_t CyU3PUSBSetDescType\_t**  
*Virtual descriptor types to be used to set descriptors.*
- **typedef enum CyU3PUSBSpeed\_t CyU3PUSBSpeed\_t**  
*List of supported USB connection speeds.*
- **typedef enum CyU3PUsbLinkPowerMode CyU3PUsbLinkPowerMode**  
*List of USB 3.0 link operating modes.*
- **typedef enum CyU3PUsbEpEvtType CyU3PUsbEpEvtType**  
*List of USB endpoint events.*
- **typedef void(\* CyU3PUsbEpEvtCb\_t) (CyU3PUsbEpEvtType evType, CyU3PUSBSpeed\_t usbSpeed, uint8\_t epNum)**  
*Callback function type used for notification of USB endpoint events.*
- **typedef void(\* CyU3PUSBEVENTCb\_t) (CyU3PUsbEventType\_t evtype, uint16\_t evdata)**  
*USB event callback type.*
- **typedef CyBool\_t(\* CyU3PUSBSetupCb\_t) (uint32\_t setupdat0, uint32\_t setupdat1)**  
*USB setup request handler type.*
- **typedef CyBool\_t(\* CyU3PUsbLPMReqCb\_t) (CyU3PUsbLinkPowerMode link\_mode)**  
*Event callback raised on host request to switch USB link to a low power state.*
- **typedef struct CyU3PEpConfig\_t CyU3PEpConfig\_t**  
*Endpoint configuration information.*
- **typedef enum CyU3PUsbDevProperty CyU3PUsbDevProperty**  
*List of USB device properties that can be queried.*

## Enumerations

- **enum CyU3PUsbEventType\_t {**  
**CY\_U3P\_USB\_EVENT\_CONNECT = 0, CY\_U3P\_USB\_EVENT\_DISCONNECT, CY\_U3P\_USB\_EVENT\_SUSPEND,**  
**CY\_U3P\_USB\_EVENT\_RESUME, CY\_U3P\_USB\_EVENT\_RESET, CY\_U3P\_USB\_EVENT\_SETCONF,**  
**CY\_U3P\_USB\_EVENT\_SPEED, CY\_U3P\_USB\_EVENT\_SETINTF,**  
**CY\_U3P\_USB\_EVENT\_SET\_SEL, CY\_U3P\_USB\_EVENT\_SOF\_ITP, CY\_U3P\_USB\_EVENT\_EP0\_STATUS\_CPLT,**  
**CY\_U3P\_USB\_EVENT\_VBUS\_VALID, CY\_U3P\_USB\_EVENT\_VBUS\_REMOVED, CY\_U3P\_USB\_EVENT\_HOST\_CONNECT,**  
**CY\_U3P\_USB\_EVENT\_HOST\_DISCONNECT, CY\_U3P\_USB\_EVENT\_OTG\_CHANGE, CY\_U3P\_USB\_EVENT\_OTG\_VBUS\_CHG,**  
**CY\_U3P\_USB\_EVENT\_OTG\_SRP, CY\_U3P\_USB\_EVENT\_EP\_UNDERRUN, CY\_U3P\_USB\_EVENT\_LNK\_RECOVERY,**  
**CY\_U3P\_USB\_EVENT\_USB3\_LNKFAIL, CY\_U3P\_USB\_EVENT\_SS\_COMP\_ENTRY, CY\_U3P\_USB\_EVENT\_SS\_COMP\_EXIT,**  
**CY\_U3P\_USB\_EVENT\_RESERVED\_1, CY\_U3P\_USB\_EVENT\_LMP\_EXCH\_FAIL }**
  - **enum CyU3PUsbMgrStates\_t {**  
**CY\_U3P\_USB\_INACTIVE = 0, CY\_U3P\_USB\_STARTED, CY\_U3P\_USB\_WAITING\_FOR\_DESC,**  
**CY\_U3P\_USB\_CONFIGURED, CY\_U3P\_USB\_VBUS\_WAIT, CY\_U3P\_USB\_CONNECTED,**  
**CY\_U3P\_USB\_ESTABLISHED, CY\_U3P\_USB\_MS\_ACTIVE,**  
**CY\_U3P\_USB\_MAX\_STATE }**
- List of USB device manager states.*



- `CyU3PReturnStatus_t CyU3PUsbSetDesc (CyU3PUSBSetDescType_t desc_type, uint8_t desc_index, uint8_t *desc)`  
*Register a USB descriptor with the driver.*
- `CyU3PReturnStatus_t CyU3PConnectState (CyBool_t connect, CyBool_t ssEnable)`  
*Enable / disable the USB connection.*
- `CyU3PReturnStatus_t CyU3PUsbStall (uint8_t ep, CyBool_t stall, CyBool_t toggle)`  
*Set or clear the stall status of an endpoint.*
- `CyU3PReturnStatus_t CyU3PUsbGetEpCfg (uint8_t ep, CyBool_t *isNak, CyBool_t *isStall)`  
*Retrieve the current state of the specified endpoint.*
- `CyU3PReturnStatus_t CyU3PUsbSetEpNak (uint8_t ep, CyBool_t nak)`  
*Force or clear the NAK status on the specified endpoint.*
- `CyBool_t CyU3PGetConnectState (void)`  
*Check whether the FX3 device is currently connected to a host.*
- `void CyU3PUsbAckSetup (void)`  
*Complete the status handshake of a USB control request.*
- `CyU3PReturnStatus_t CyU3PSetEpConfig (uint8_t ep, CyU3PEpConfig_t *epinfo)`  
*Configure a USB endpoint.*
- `CyU3PReturnStatus_t CyU3PSetEpPacketSize (uint8_t ep, uint16_t maxPktSize)`  
*Set the maximum packet size for a USB endpoint.*
- `CyU3PReturnStatus_t CyU3PUsbSetEpPktMode (uint8_t ep, CyBool_t pktMode)`  
*Select whether a OUT endpoint will function in packet (one buffer per packet) mode or not.*
- `CyU3PReturnStatus_t CyU3PUsbSendEP0Data (uint16_t count, uint8_t *buffer)`  
*Send data to the USB host via EP0-IN.*
- `CyU3PReturnStatus_t CyU3PUsbGetEP0Data (uint16_t count, uint8_t *buffer, uint16_t *readCount)`  
*Read data associated with a control-OUT transfer.*
- `CyU3PReturnStatus_t CyU3PUsbFlushEp (uint8_t ep)`  
*Clear (flush) all data buffers associated with a specified endpoint.*
- `CyU3PReturnStatus_t CyU3PUsbResetEp (uint8_t ep)`  
*Resets the specified endpoint.*
- `CyU3PReturnStatus_t CyU3PUsbMapStream (uint8_t ep, uint8_t socketNum, uint16_t streamId)`  
*Map a socket to stream of the specified endpoint.*
- `CyU3PReturnStatus_t CyU3PUsbChangeMapping (uint8_t ep, uint8_t socketNum, CyBool_t remap, uint16_t newstreamId, uint8_t newep)`  
*Map a socket to stream of the specified endpoint.*
- `CyU3PUSBSpeed_t CyU3PUsbGetSpeed (void)`  
*Get the connection speed at which USB is operating.*
- `CyU3PReturnStatus_t CyU3PUsbVBattEnable (CyBool_t enable)`  
*Configure USB block on FX3 to work off Vbatt power instead of Vbus.*
- `CyU3PReturnStatus_t CyU3PUsbGetDevProperty (CyU3PUsbDevProperty type, uint32_t *buf)`  
*Function that returns some properties of the USB device.*
- `CyU3PReturnStatus_t CyU3PUsbSendErdy (uint8_t ep, uint16_t bulkStream)`  
*Function to send an ERDY TP to a USB 3.0 host.*
- `CyU3PReturnStatus_t CyU3PUsbSendNrdy (uint8_t ep, uint16_t bulkStream)`  
*Function to send an NRDY TP to a USB 3.0 host.*
- `CyU3PReturnStatus_t CyU3PUsbSendAckTP (uint8_t ep, uint8_t nump, uint16_t bulkStream)`  
*Function to send an ACK TP to a USB 3.0 host.*
- `CyU3PReturnStatus_t CyU3PUsbGetLinkPowerState (CyU3PUsbLinkPowerMode *mode_p)`  
*Function to get the current power state of the USB 3.0 link.*
- `CyU3PReturnStatus_t CyU3PUsbSetLinkPowerState (CyU3PUsbLinkPowerMode link_mode)`  
*Function to request a device entry into one of the U0, U1 or U2 link power modes.*
- `CyU3PReturnStatus_t CyU3PUsbEnableITPEvent (CyBool_t enable)`

- Function to enable/disable notification of SOF/ITP events to the application.
- **CyU3PReturnStatus\_t CyU3PUsbDoRemoteWakeup (void)**
  - Trigger USB 2.0 Remote Wakeup signalling to the host.*
- **CyU3PReturnStatus\_t CyU3PUsbForceFullSpeed (CyBool\_t enable)**
  - Function to force the USB 2.0 device connection to full speed.*
- **CyU3PReturnStatus\_t CyU3PUsbSendDevNotification (uint8\_t notificationType, uint32\_t param0, uint32\_t param1)**
  - Function to send a DEV\_NOTIFICATION Transaction Packet to the host.*
- **CyU3PReturnStatus\_t CyU3PUsbLPMDisable (void)**
  - Function to prevent FX3 from entering U1/U2 states.*
- **CyU3PReturnStatus\_t CyU3PUsbLPMEnable (void)**
  - Function to re-enable automated handling of U1/U2 state entry.*
- **void CyU3PUsbSetBooterSwitch (CyBool\_t enable)**
  - Function to enable/disable switching control back to the FX3 2-stage bootloader.*
- **CyU3PReturnStatus\_t CyU3PUsbGetBooterVersion (uint8\_t \*major\_p, uint8\_t \*minor\_p, uint8\_t \*patch\_p)**
  - Function to check the version of the boot firmware that transferred control to this firmware image.*
- **CyU3PReturnStatus\_t CyU3PUsbJumpBackToBooter (uint32\_t address)**
  - Function to transfer control back to the FX3 2-stage bootloader.*
- **void CyU3PUsbInitEventLog (uint8\_t \*buffer, uint32\_t bufSize)**
  - Function to initiate logging of USB state changes into a circular buffer.*
- **uint16\_t CyU3PUsbGetEventLogIndex (void)**
  - Get the current event log buffer index.*
- **void CyU3PUsbEpPrepare (CyU3PUSBSpeed\_t curSpeed)**
  - Prepare all enabled USB device endpoints for data transfer.*
- **CyU3PReturnStatus\_t CyU3PUsbResetEndpointMemories (void)**
  - Reset and re-initialize the endpoint memory block on FX3.*
- **void CyU3PUsbEnableEPPPrefetch (void)**
  - Configure the USB DMA interface to continuously pre-fetch data.*
- **CyU3PReturnStatus\_t CyU3PUsbGetErrorCounts (uint16\_t \*phy\_err\_cnt, uint16\_t \*lnk\_err\_cnt)**
  - Function to get the number of USB 3.0 PHY and LINK error counts detected by FX3.*
- **CyU3PReturnStatus\_t CyU3PUsbEPSetBurstMode (uint8\_t ep, CyBool\_t burstEnable)**
  - Function to enable burst mode operation for a USB endpoint.*
- **CyU3PReturnStatus\_t CyU3PUsbSetTxSwing (uint32\_t swing)**
  - Set the Tx amplitude range for the USB 3.0 signals.*
- **CyU3PReturnStatus\_t CyU3PUsbSetTxDeemphasis (uint32\_t value)**
  - Set the Tx de-emphasis setting for the USB 3.0 signals.*
- **void CyU3PUsbSSCDDisable (void)**
  - Disable Spread-Spectrum Clocking support in the USB block.*
- **CyU3PReturnStatus\_t CyU3PUsbGetEpSeqNum (uint8\_t ep, uint8\_t \*seqnum\_p)**
  - Get the current sequence number for an endpoint.*
- **CyU3PReturnStatus\_t CyU3PUsbSetEpSeqNum (uint8\_t ep, uint8\_t seqnum)**
  - Set the active sequence number for an endpoint.*
- **CyU3PReturnStatus\_t CyU3PUsbControlVBusDetect (CyBool\_t enable, CyBool\_t useVbatt)**
  - Enable/Disable VBus detection in FX3 firmware.*
- **CyU3PReturnStatus\_t CyU3PUsbControlUsb2Support (CyBool\_t enable)**
  - Enable/Disable USB 2.0 device operation on FX3 device.*
- **CyU3PReturnStatus\_t CyU3PUsb2Resume (void)**
  - Resume USB 2.0 link from LPM-L1 suspend.*
- **CyU3PReturnStatus\_t CyU3PUsbEpSetPacketsPerBuffer (uint8\_t epNum, uint8\_t numPkts)**
  - Set the number of maximum sized packets that an OUT endpoint buffer can accommodate.*
- **CyU3PReturnStatus\_t CyU3PUsbSetEpSuspDisableMask (uint16\_t noSuspMask)**
  - Set the number of maximum sized packets that an OUT endpoint buffer can accommodate.*

*Select endpoints that should not be suspended to allow EP0 transfers to complete.*

- [CyBool\\_t CyU3PUibCheckConnection \(void\)](#)

*Function to check whether the USB connection is still active.*
- [void CyU3PUsbLinkComplianceControl \(CyBool\\_t isEnabled\)](#)

*Disable USB 3.0 electrical compliance mode support.*

### 5.36.1 Detailed Description

The USB device mode driver on the FX3 device is responsible for handling USB 3.0 and 2.0 connections, and providing the API hooks for configuring device operations.

### 5.36.2 Enumeration Modes

The FX3 USB driver supports two different modes of enumeration, one of which is optimized for good performance and simplified application code; whereas the other allows for greater flexibility.

#### Description

Two different modes of enumeration control are supported by the USB driver on the FX3 device.

1. **Fast enumeration\*\*:** In this case, all of the descriptors for all USB speeds of interest are registered with the USB driver through the CyU3PUsbSetDesc API call. The USB driver then uses this information to handle all of the standard USB setup requests. The driver is responsible for identifying the current connection speed and using the appropriate set of USB descriptors.

In this case, only a single configuration descriptor can be used because only one config descriptor for each connection speed can be registered with the USB driver. The application will also need to disconnect the USB link and re-enumerate every time it wants to change the descriptors.

1. **User enumeration\*\*:** In this case, all USB setup requests are forwarded to the user application logic through a callback function. It is the responsibility of the application code to handle these requests and to send the appropriate responses to the USB host. The application also has to track the current USB connection speed and use this to identify the specific descriptor to send up to the host.

In this case, the application can change the descriptor data at will and has maximum flexibility in terms of implementing multiple configurations and alternate settings.

#### Note

In either form of enumeration, any non-standard (Class or vendor specific) setup request will trigger a callback function. The user application is expected to handle these requests in all cases.

### 5.36.3 USB Device Mode Functions

The USB device APIs are used to configure the USB device functionality and to perform USB data transfers.

#### Description

The FX3 device supports programmable USB peripheral implementation at USB-SS, USB-HS and USB-FS speeds. The type of USB device to be implemented as well as the endpoint pipes to be used can be configured through a set of USB device mode APIs. The USB device mode APIs also provide the capability to manage the endpoint states such as STALL and NAK as well as to perform data transfers on the control endpoint (EP0). Data transfers on the other USB endpoints will be controlled through the DMA APIs.

### 5.36.4 Macro Definition Documentation

#### 5.36.4.1 #define CY\_U3P\_MAX\_STRING\_DESC\_INDEX (16)

Number of string descriptors that are supported by the USB driver.

##### Description

The USB driver is capable of storing pointers to and handling a number of string descriptors. This constant represents the number of strings that the driver is capable of handling.

##### See also

[CyU3PUsbSetDesc](#)

### 5.36.5 Typedef Documentation

#### 5.36.5.1 typedef struct CyU3PEpConfig\_t CyU3PEpConfig\_t

Endpoint configuration information.

##### Description

This structure holds all the properties of a USB endpoint. This structure is used to provide information about the desired configuration of various endpoints in the system.

##### See also

[CyU3PSetEpConfig](#)

#### 5.36.5.2 typedef enum CyU3PUsbDevProperty CyU3PUsbDevProperty

List of USB device properties that can be queried.

##### Description

This is the list of USB device properties that can be queried by the application through the CyU3PUsbGetDevProperty API. Each property value is specified as a 32 bit integer value.

##### See also

[CyU3PUsbGetDevProperty](#)

#### 5.36.5.3 typedef void(\* CyU3PUsbEpEvtCb\_t) (CyU3PUsbEpEvtType evType, CyU3PUSBSpeed\_t usbSpeed, uint8\_t epNum )

Callback function type used for notification of USB endpoint events.

##### Description

This callback function type is used by the firmware application to receive notifications of USB endpoint events of interest.

##### See also

[CyU3PUsbEpEvtType](#)  
[CyU3PUsbRegisterEpEvtCallback](#)

#### 5.36.5.4 typedef enum CyU3PUsbEpEvtType CyU3PUsbEpEvtType

List of USB endpoint events.

##### Description

The USB API can relay a set of endpoint related events to the firmware application for monitoring and control purposes. This enumerated data type lists the various endpoint events that can be generated by the API library.

See also

[CyU3PUsbEvtCb\\_t](#)  
[CyU3PUsbRegisterEpEvtCallback](#)  
[CyU3PUsbEpEvtControl](#)  
[CyU3PUsbSetXfer](#)

#### 5.36.5.5 `typedef void(* CyU3PUSBEvtCb_t)(CyU3PUsbEventType_t evtype, uint16_t evdata )`

USB event callback type.

##### Description

Type of callback function that should be registered with the USB driver to get notifications of USB specific events.

The evtype parameter specifies the type of event and evdata contains event specific data.

See also

[CyU3PUsbEventType\\_t](#)  
[CyU3PUsbRegisterEventCallback](#)

#### 5.36.5.6 `typedef enum CyU3PUsbEventType_t CyU3PUsbEventType_t`

List of USB related events that are raised by the USB manager.

##### Description

The USB manager or driver continually runs on a thread on the FX3 device and performs the necessary operations to handle various USB requests from the host. The USB driver can notify the user application about various USB specific events by calling a callback function that is registered with it. This type lists the various USB event types that are notified to the application logic.

See also

[CyU3PUSBEvtCb\\_t](#)  
[CyU3PUsbEnableITPEvent](#)

#### 5.36.5.7 `typedef enum CyU3PUsbLinkPowerMode CyU3PUsbLinkPowerMode`

List of USB 3.0 link operating modes.

##### Description

List of USB 3.0 link operating modes. Only the active and low power modes are listed here as the rest of the link modes are not visible to software.

See also

[CyU3PUsbLPMReqCb\\_t](#)

#### 5.36.5.8 `typedef CyBool_t(* CyU3PUsbLPMReqCb_t)(CyU3PUsbLinkPowerMode link_mode )`

Event callback raised on host request to switch USB link to a low power state.

##### Description

The USB 3.0 host may request the FX3 device to switch the USB link to a low power state when the host has no more activity to perform. This event callback is raised when a request from the host to switch the link to a U1/U2 state is received. The link\_mode parameter indicates the power state that the host is requesting for. The return value from this function call is expected to indicate whether the low power state entry request should be accepted. A return of CyTrue will cause FX3 to accept entry into U1/U2 state and a return of CyFalse will cause FX3 to reject the request.

See also

[CyU3PUsbLinkPowerMode](#)  
[CyU3PUsbRegisterLPMRequestCallback](#)

#### 5.36.5.9 `typedef enum CyU3PUsbMgrStates_t CyU3PUsbMgrStates_t`

List of USB device manager states.

##### Description

This USB driver (manager) implements a state machine that tracks the current state of the USB connection and uses this to identify the appropriate response to USB requests. This type lists the various possible states for the USB driver state machine. Some of these states are software defined and have no connection with any USB bus conditions.

#### 5.36.5.10 `typedef enum CyU3PUSBSetDescType_t CyU3PUSBSetDescType_t`

Virtual descriptor types to be used to set descriptors.

##### Description

The user application can register different device and configuration descriptors with the USB driver, to be used at various USB connection speeds. To support this, the `CyU3PUsbSetDesc` call accepts a descriptor type parameter that is different from the USB standard descriptor type value that is embedded in the descriptor itself. This enumerated type is to be used by the application to register various descriptors with the USB driver.

See also

[CyU3PUsbSetDesc](#)

#### 5.36.5.11 `typedef CyBool_t(* CyU3PUSBSetupCb_t)(uint32_t setupdat0, uint32_t setupdat1 )`

USB setup request handler type.

##### Description

Type of callback function that is invoked to handle USB setup requests. The function shall return `CyTrue` (1) if it has handled the setup request. If it is unable to handle this specific request, it shall return `CyFalse` and the USB driver will perform the default actions corresponding to the setup request.

The handling of each setup request will involve one and only one of the following API calls:

**CyU3PUsbSendEP0Data:** Called to perform any IN-data phase associated with the request. The status handshake for the request will be completed as soon as all of the data requested has been sent to the host.

**CyU3PUsbGetEP0Data:** Called to perform any OUT-data phase associated with the request. The length specified should match the transfer length specified by the host. The status handshake for the request will be completed as soon as all of the data has been received from the host.

**CyU3PUsbAckSetup:** Called to complete the status handshake (ACK) phase of a request that does not involve any data transfer.

**CyU3PUsbStall:** Called to stall EP0 to fail the setup request. The ep number can be specified as 0x00 or 0x80, and the API will take care of stalling the ep in the appropriate direction.

These API calls can be made from the Setup callback directly or in a delayed fashion in a bottom half that is scheduled from the Setup callback. In either case, the Setup callback should return `CyTrue` to let the API know that the default action (stalling EP0 for any non-standard request) should not be performed.

##### Note

Calling both `CyU3PUsbSendEP0Data/CyU3PUsbGetEP0Data` and `CyU3PUsbAckSetup` or calling `CyU3PUsbAckSetup` multiple times in response to a control request can result in errors due to the subsequent request being ACKed prematurely.

See also

[CyU3PUsbRegisterSetupCallback](#)  
[CyU3PUsbSendEP0Data](#)  
[CyU3PUsbGetEP0Data](#)  
[CyU3PUsbAckSetup](#)  
[CyU3PUsbStall](#)

## 5.36.6 Enumeration Type Documentation

### 5.36.6.1 enum CyU3PUsbDevProperty

List of USB device properties that can be queried.

#### Description

This is the list of USB device properties that can be queried by the application through the CyU3PUsbGetDevProperty API. Each property value is specified as a 32 bit integer value.

See also

[CyU3PUsbGetDevProperty](#)

#### Enumerator

**CY\_U3P\_USB\_PROP\_DEVADDR** The USB device address assigned to FX3. The 7 bit device address value is returned as the LS bits of a 32 bit word.

**CY\_U3P\_USB\_PROP\_FRAMECNT** USB 2.0 frame count value. Returned as a 32 bit unsigned integer value.

**CY\_U3P\_USB\_PROP\_LINKSTATE** Current state of the USB 3.0 Link. Returned as a 32 bit unsigned integer value. See CyU3PUsbLinkState\_t for state encoding values.

**CY\_U3P\_USB\_PROP\_ITPINFO** The Isochronous timestamp field from the last ITP received. Returned as a 32 bit unsigned integer value.

**CY\_U3P\_USB\_PROP\_SYS\_EXIT\_LAT** The System Exit Latency value specified by the USB host. The six byte SEL data is copied into the data buffer. The pointer passed in this case should correspond to a six byte (or longer) buffer.

### 5.36.6.2 enum CyU3PUsbEpEvtType

List of USB endpoint events.

#### Description

The USB API can relay a set of endpoint related events to the firmware application for monitoring and control purposes. This enumerated data type lists the various endpoint events that can be generated by the API library.

See also

[CyU3PUsbEpEvtCb\\_t](#)  
[CyU3PUsbRegisterEpEvtCallback](#)  
[CyU3PUsbEpEvtControl](#)  
[CyU3PUsbSetXfer](#)

#### Enumerator

**CYU3P\_USBEP\_NAK\_EVT** The endpoint sent NAK in response to a host transfer request. In the case of a USB 3.0 connection, this callback only indicates that the endpoint has gone into a flowcontrol state due to not being ready for data transfer. This does not necessarily mean that the device has sent an NRDY response to the USB host on that endpoint.

**CYU3P\_USBEP\_ZLP\_EVT** A Zero Length Packet was sent/received on the endpoint.

**CYU3P\_USBEP\_SLP\_EVT** A Short Length Packet was sent/received on the endpoint. Note that the SLP event will only occur on USB 2.0 connections if a non-zero transfer size has been set on the endpoint using CyU3PUsbSetXfer.

**CYU3P\_USBEP\_ISOERR\_EVT** Out of sequence ISO PIDs or ZLP received on ISO endpoint.

**CYU3P\_USBEP\_SS\_RETRY\_EVT** An ACK TP with the RETRY bit was received on the IN endpoint.

**CYU3P\_USBEP\_SS\_SEQERR\_EVT** Sequence number error detected during endpoint data transfer.

**CYU3P\_USBEP\_SS\_STREAMERR\_EVT** A Bulk-Stream protocol error occurred on the endpoint.

**CYU3P\_USBEP\_SS\_BTERM\_EVT** The USB 3.0 host prematurely terminated a burst transfer.

**CYU3P\_USBEP\_SS\_RESET\_EVT** USB 3.0 endpoint reset event. A stall recovery is required.

### 5.36.6.3 enum CyU3PUsbEventType\_t

List of USB related events that are raised by the USB manager.

#### Description

The USB manager or driver continually runs on a thread on the FX3 device and performs the necessary operations to handle various USB requests from the host. The USB driver can notify the user application about various USB specific events by calling a callback function that is registered with it. This type lists the various USB event types that are notified to the application logic.

#### See also

[CyU3PUSBEventCb\\_t](#)

[CyU3PUsbEnableITPEvent](#)

#### Enumerator

**CY\_U3P\_USB\_EVENT\_CONNECT** USB Connect event. The evData parameter is an integer value which indicates whether it is a 3.0 connection (evData = 1) or a 2.0 connection (evData = 0).

**CY\_U3P\_USB\_EVENT\_DISCONNECT** USB Disconnect event. The evData parameter is not used and will be NULL.

**CY\_U3P\_USB\_EVENT\_SUSPEND** USB Suspend event for both USB 2.0 and 3.0 connections. The evData parameter is not used and will be NULL.

**CY\_U3P\_USB\_EVENT\_RESUME** USB Resume event. The evData parameter is not used and will be NULL.

**CY\_U3P\_USB\_EVENT\_RESET** USB Reset event. The evData parameter is an integer value which indicates whether it is a 3.0 connection (evData = 1) or a 2.0 connection (evData = 0).

**CY\_U3P\_USB\_EVENT\_SETCONF** USB Set Configuration event. evData provides the configuration number that is selected by the host.

**CY\_U3P\_USB\_EVENT\_SPEED** USB Speed Change event. The evData parameter is not used and will always be set to 1.

**CY\_U3P\_USB\_EVENT\_SETINTF** USB Set Interface event. The evData parameter provides the interface number and the selected alternate setting values. Bits 7 - 0: LSB of wValue field from setup request. Bits 15 - 8: LSB of wIndex field from setup request.

**CY\_U3P\_USB\_EVENT\_SET\_SEL** USB 3.0 Set System Exit Latency event. The event data parameter will be zero. The CyU3PUsbGetDevProperty API can be used to get the SEL values specified by the host.

**CY\_U3P\_USB\_EVENT\_SOF\_ITP** SOF/ITP event notification. The evData parameter is not used and will be NULL. Since these events are very frequent, the event notification needs to be separately enabled using the [CyU3PUsbEnableITPEvent\(\)](#) API call.

**CY\_U3P\_USB\_EVENT\_EP0\_STAT\_CPLT** Event notifying completion of the status phase of a control request. This event is only generated for control requests that are handled by the user's application; and not for requests that are handled internally by the USB driver.

**CY\_U3P\_USB\_EVENT\_VBUS\_VALID** Vbus power detected on FX3's USB port.

**CY\_U3P\_USB\_EVENT\_VBUS\_REMOVED** Vbus power removed from FX3's USB port.

**CY\_U3P\_USB\_EVENT\_HOST\_CONNECT** USB host mode connect event.

**CY\_U3P\_USB\_EVENT\_HOST\_DISCONNECT** USB host mode disconnect event.

**CY\_U3P\_USB\_EVENT\_OTG\_CHANGE** USB OTG-ID Pin state change.

**CY\_U3P\_USB\_EVENT\_OTG\_VBUS\_CHG** VBus made valid by OTG host.

**CY\_U3P\_USB\_EVENT\_OTG\_SRП** SRP signalling detected from OTG device.

**CY\_U3P\_USB\_EVENT\_EP\_UNDERRUN** Indicates that a data underrun error has been detected on one of the USB endpoints. The event data will provide the endpoint number.

**CY\_U3P\_USB\_EVENT\_LNK\_RECOVERY** Indicates that the USB link has entered recovery. This event will not be raised if the recovery is entered as part of a transition from Ux to U0. This event is raised from the interrupt context, and the event handler should not invoke any blocking operations.

**CY\_U3P\_USB\_EVENT\_USB3\_LNKFAIL** Indicates that the USB 3.0 link from FX3 to the USB host has failed. This event is only generated if the USB 2.0 operation of FX3 is disabled; and should be used to trigger a USB 2.0 connection from the external controller.

**CY\_U3P\_USB\_EVENT\_SS\_COMP\_ENTRY** Indicates that the USB 3.0 link has entered the compliance state.

**CY\_U3P\_USB\_EVENT\_SS\_COMP\_EXIT** Indicates that the USB 3.0 link has exited the compliance state.

**CY\_U3P\_USB\_EVENT\_RESERVED\_1** Reserved event code. No handling required.

**CY\_U3P\_USB\_EVENT\_LMP\_EXCH\_FAIL** Indicates that USB 3.0 connection failed due to LPM handshake failure.

#### 5.36.6.4 enum CyU3PUsbLinkPowerMode

List of USB 3.0 link operating modes.

##### Description

List of USB 3.0 link operating modes. Only the active and low power modes are listed here as the rest of the link modes are not visible to software.

##### See also

[CyU3PUsbLPMReqCb\\_t](#)

##### Enumerator

**CyU3PUsbLPM\_U0** U0 (active) power state.

**CyU3PUsbLPM\_U1** U1 power state.

**CyU3PUsbLPM\_U2** U2 power state.

**CyU3PUsbLPM\_U3** U3 (suspend) power state.

**CyU3PUsbLPM\_COMP** Compliance state.

**CyU3PUsbLPM\_Unknown** The link is not in any of the Ux states. This normally happens while the link training is ongoing.

**CyU3PUsbLPM\_U0** U0 (active) power state.

**CyU3PUsbLPM\_U1** U1 power state.

**CyU3PUsbLPM\_U2** U2 power state.

**CyU3PUsbLPM\_U3** U3 (suspend) power state.

**CyU3PUsbLPM\_COMP** Compliance state.

**CyU3PUsbLPM\_Unknown** The link is not in any of the Ux states. This normally happens while the link training is ongoing.

### 5.36.6.5 enum CyU3PUsbMgrStates\_t

List of USB device manager states.

#### Description

This USB driver (manager) implements a state machine that tracks the current state of the USB connection and uses this to identify the appropriate response to USB requests. This type lists the various possible states for the USB driver state machine. Some of these states are software defined and have no connection with any USB bus conditions.

#### Enumerator

- CY\_U3P\_USB\_INACTIVE** USB module not started.
- CY\_U3P\_USB\_STARTED** USB module started and waiting for configuration.
- CY\_U3P\_USB\_WAITING\_FOR\_DESC** USB module waiting for a configuration descriptor.
- CY\_U3P\_USB\_CONFIGURED** USB module completely configured.
- CY\_U3P\_USB\_VBUS\_WAIT** USB module waiting for Vbus to connect to USB host.
- CY\_U3P\_USB\_CONNECTED** Waiting for USB connection.
- CY\_U3P\_USB\_ESTABLISHED** USB connection active.
- CY\_U3P\_USB\_MS\_ACTIVE** Mass storage function active.
- CY\_U3P\_USB\_MAX\_STATE** Sentinel state.

### 5.36.6.6 enum CyU3PUSBSetDescType\_t

Virtual descriptor types to be used to set descriptors.

#### Description

The user application can register different device and configuration descriptors with the USB driver, to be used at various USB connection speeds. To support this, the CyU3PUsbSetDesc call accepts a descriptor type parameter that is different from the USB standard descriptor type value that is embedded in the descriptor itself. This enumerated type is to be used by the application to register various descriptors with the USB driver.

#### See also

[CyU3PUsbSetDesc](#)

#### Enumerator

- CY\_U3P\_USB\_SET\_SS\_DEVICE\_DESCR** SuperSpeed (USB 3.0) device descriptor.
- CY\_U3P\_USB\_SET\_HS\_DEVICE\_DESCR** USB 2.0 device descriptor.
- CY\_U3P\_USB\_SET\_DEVQUAL\_DESCR** Device qualifier descriptor
- CY\_U3P\_USB\_SET\_FS\_CONFIG\_DESCR** Full Speed configuration descriptor.
- CY\_U3P\_USB\_SET\_HS\_CONFIG\_DESCR** High Speed configuration descriptor.
- CY\_U3P\_USB\_SET\_STRING\_DESCR** String descriptor.
- CY\_U3P\_USB\_SET\_SS\_CONFIG\_DESCR** SuperSpeed configuration descriptor.
- CY\_U3P\_USB\_SET\_SS\_BOS\_DESCR** BOS descriptor.
- CY\_U3P\_USB\_SET\_SS\_DEVICE\_DESCR** Device descriptor for SuperSpeed.
- CY\_U3P\_USB\_SET\_HS\_DEVICE\_DESCR** Device descriptor for Hi-Speed and Full Speed.
- CY\_U3P\_USB\_SET\_DEVQUAL\_DESCR** Device Qualifier descriptor.
- CY\_U3P\_USB\_SET\_FS\_CONFIG\_DESCR** Configuration descriptor for Full Speed.
- CY\_U3P\_USB\_SET\_HS\_CONFIG\_DESCR** Configuration descriptor for Hi-Speed.
- CY\_U3P\_USB\_SET\_STRING\_DESCR** String Descriptor
- CY\_U3P\_USB\_SET\_SS\_CONFIG\_DESCR** Configuration descriptor for SuperSpeed.
- CY\_U3P\_USB\_SET\_SS\_BOS\_DESCR** BOS descriptor.
- CY\_U3P\_USB\_SET\_OTG\_DESCR** OTG descriptor.

### 5.36.6.7 enum CyU3PUSBSpeed\_t

List of supported USB connection speeds.

#### Enumerator

- CY\_U3P\_NOT\_CONNECTED** USB device not connected.
- CY\_U3P\_FULL\_SPEED** USB full speed.
- CY\_U3P\_HIGH\_SPEED** High speed.
- CY\_U3P\_SUPER\_SPEED** Super speed.

## 5.36.7 Function Documentation

### 5.36.7.1 CyU3PReturnStatus\_t CyU3PConnectState ( CyBool\_t connect, CyBool\_t ssEnable )

Enable / disable the USB connection.

#### Description

This function is used to enable or disable the USB PHYs on the FX3 device and to control the connection to the USB host in that manner. Re-enumeration can be achieved by disabling and then enabling the USB connection.

#### Return value

- CY\_U3P\_SUCCESS - when the call is successful.
- CY\_U3P\_ERROR\_NOT\_STARTED - when the USB driver has not been started.
- CY\_U3P\_ERROR\_ALREADY\_STARTED - if the connection is already enabled.
- CY\_U3P\_ERROR\_NOT\_CONFIGURED - if all of the necessary configuration has not been completed.

#### See also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PUsbSetDesc](#)
- [CyU3PGetConnectState](#)

#### Parameters

<i>connect</i>	CyTrue to enable connection, CyFalse to disable connection.
<i>ssEnable</i>	Whether to enumerate as a SS device or FS/HS device

### 5.36.7.2 CyBool\_t CyU3PGetConnectState ( void )

Check whether the FX3 device is currently connected to a host.

#### Description

This function checks whether the the FX3 device is connected to a USB host. This depends on the detection of a valid Vbus input to the device.

#### Return value

CyTrue: The USB PHY on FX3 has been turned ON. CyFalse: The USB PHY on FX3 is turned OFF.

#### See also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PConnectState](#)

### 5.36.7.3 CyU3PReturnStatus\_t CyU3PSetEpConfig ( uint8\_t ep, CyU3PEpConfig\_t \* epinfo )

Configure a USB endpoint.

#### Description

The FX3 device has 30 user configurable endpoints (1-OUT to 15-OUT and 1-IN to 15-IN) which can be separately selected and configured with desired properties such as endpoint type, the maximum packet size, amount of desired buffering. For bulk endpoints under super-speed, the number of bulk streams supported on that endpoint also needs to be specified.

All of these endpoints are kept disabled by default. This function is used to enable and set the properties for a specified endpoint. Separate calls need to be made to enable and configure each endpoint that needs to be used. The same function can be called to disable the endpoints after use.

#### Note

Please note that a mult (isoPkts) setting of greater than 0 (multiple packets or bursts in a micro-frame) is only supported for endpoints 3 and 7 in either direction.

#### Return value

CY\_U3P\_SUCCESS - when the call is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - when the USB driver has not been started.

CY\_U3P\_NULL\_POINTER - when the epinfo pointer is NULL.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - when the packet size or the endpoint number is invalid.

CY\_U3P\_ERROR\_INVALID\_CONFIGURATION - when a non-zero isoPkts value is used with endpoints other than 3 and 7.

#### See also

[CyU3PEpConfig\\_t](#)

#### Parameters

ep	Endpoint number to be configured.
epinfo	Properties to associate with the endpoint.

### 5.36.7.4 CyU3PReturnStatus\_t CyU3PSetEpPacketSize ( uint8\_t ep, uint16\_t maxPktSize )

Set the maximum packet size for a USB endpoint.

#### Description

The DMA channels associated with USB endpoints on the FX3 device can be configured to have buffers of any user specified size (up to 64 KB). The FX3 device will allow multiple packets worth of data to be collected into a single DMA buffer as long as all of the packets are full packets. Any short packet arriving from USB will cause a DMA buffer to be wrapped up and committed to the consumer.

This API is used to set the maximum packet size that applies to the current endpoint. This needs to be set by the application based on the active USB connection speed, if there is a need to collect multiple data packets into a buffer. By default, this is set to the maximum packet size computed based on the user specified [CyU3PEpConfig\\_t.pcktSize](#) value, the USB connection speed and endpoint type.

#### Note

Please note that the maximum packet size will be automatically adjusted to the connection speed when the U↔SB connection is first detected. If this value is to be over-ridden, this should be done after the enumeration is complete. One way to do this is by calling this function on receiving a SET\_CONFIGURATION setup callback or CY\_U3P\_USB\_EVENT\_SETCONF event.

#### Return value

CY\_U3P\_SUCCESS - when the call is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the endpoint or size specified is invalid.

**See also**

[CyU3PSetEpConfig](#)  
[CyU3PUsbSetEpPktMode](#)

**Parameters**

<i>ep</i>	Endpoint number to be configured.
<i>maxPktSize</i>	Maximum packet size for endpoint in bytes.

**5.36.7.5 CyBool\_t CyU3PUibCheckConnection ( void )**

Function to check whether the USB connection is still active.

**Description**

This function checks whether the USB connection is active by looking for SOF or ITP events. This will only work if the user ensures that the USB link is not in a low power mode.

**Return value**

CyTrue if the connection is active. CyFalse if the connection is not active.

**5.36.7.6 CyU3PReturnStatus\_t CyU3PUsb2Resume ( void )**

Resume USB 2.0 link from LPM-L1 suspend.

**Description**

The USB host can request the USB Hi-Speed link to be placed in the LPM-L1 state during device operation. The FX3 device can initiate an exit from the L1 state using resume signaling, when it is ready with data. This function is used to initiate L1 exit from the FX3 side.

**Return value**

CY\_U3P\_SUCCESS if the resume is initiated, or if the link is already in L0 state.

CY\_U3P\_ERROR\_NOT\_STARTED if the USB block has not been started.

CY\_U3P\_ERROR\_NOT\_CONFIGURED if the USB connection has not been enabled.

CY\_U3P\_ERROR\_OPERN\_DISABLED if a USB 3.0 connection is active.

**See also**

[CyU3PUsbLPMDisable](#)

**5.36.7.7 void CyU3PUsbAckSetup ( void )**

Complete the status handshake of a USB control request.

**Description**

This function is used to complete the status handshake of a USB control request that does not involve any data transfer. If there is a need for OUT or IN data transfers to process the control request, the CyU3PUsbGetEP0Data and CyU3PUsbSendEP0Data calls should be used instead.

This function should only be used if a positive ACK is to be sent to the USB host. To indicate an error condition, the CyU3PUsbStall call should be used to stall the endpoint EP0-OUT or EP0-IN.

**Return value**

None

See also

[CyU3PUSBSetupCb\\_t](#)  
[CyU3PUsbRegisterSetupCallback](#)  
[CyU3PUsbSendEP0Data](#)  
[CyU3PUsbGetEP0Data](#)

### 5.36.7.8 CyU3PReturnStatus\_t CyU3PUsbChangeMapping ( uint8\_t ep, uint8\_t socketNum, CyBool\_t remap, uint16\_t newstreamId, uint8\_t newep )

Map a socket to stream of the specified endpoint.

#### Description

This function is used to map the stream of the specified endpoint to the socket passed as a parameter, this can be done if the socket is not already mapped.

#### Return value

CY\_U3P\_SUCCESS - when the call is successful.  
CY\_U3P\_ERROR\_NOT\_STARTED - when the USB driver has not been started.  
CY\_U3P\_ERROR\_BAD\_ARGUMENT - if invalid parameter is passed to function.

See also

[CyU3PSetEpConfig](#)  
[CyU3PUsbMapStream](#)

#### Parameters

<i>ep</i>	Endpoint number to which the stream was mapped
<i>socketNum</i>	Socket number to which the stream was mapped
<i>remap</i>	CyTrue: if remapping is to be done
<i>newstreamId</i>	Stream id of the new stream to be mapped to the socket
<i>newep</i>	Endpoint number of the new stream to be mapped to the socket

### 5.36.7.9 CyU3PReturnStatus\_t CyU3PUsbControlUsb2Support ( CyBool\_t enable )

Enable/Disable USB 2.0 device operation on FX3 device.

#### Description

By default, the FX3 device is configured to attempt both USB 3.0 and 2.0 connections to a USB host. The USB driver in the firmware automatically attempts a hi-speed or full speed connection, if the SuperSpeed connection fails to go through link training.

Some designs may use the FX3 device in USB 3.0 mode alone, and require that an external controller be used in USB 2.0 mode. This API can be used to modify the driver behavior to suppress the USB 2.0 device operation on FX3. USB events are provided to notify the user that the 3.0 connection has failed, and a 2.0 connection needs to be attempted. The CyU3PConnectState API should be called repeatedly to retry the USB 3.0 connection.

#### Return value

CY\_U3P\_SUCCESS if the configuration is updated properly.  
CY\_U3P\_ERROR\_NOT\_STARTED if the USB block has not been started.  
CY\_U3P\_ERROR\_INVALID\_SEQUENCE if the USB connection has already been enabled.

#### Parameters

<i>enable</i>	Whether USB 2.0 support on FX3 is enabled (default = enabled).
---------------	--

### 5.36.7.10 CyU3PReturnStatus\_t CyU3PUsbControlVBusDetect ( CyBool\_t enable, CyBool\_t useVbatt )

Enable/Disable VBus detection in FX3 firmware.

#### Description

When the FX3 device is functioning as a USB device, it performs Vbus detection by default; and will connect to the host only when VBus voltage is seen to be above 4.1 V. This API is used to configure the FX3 device to ignore the VBus input.

In some cases, the VBus voltage may be connected to the VBatt input of FX3 through a regulator. The firmware can be configured to automatically detect the VBatt input and enable/disable the USB connectivity accordingly. The useVbatt parameter can be set to select this mode of operation.

If the VBus signal from the USB connector is not connected to FX3 or connected to a GPIO, the user is expected to enable/disable the USB connection at appropriate times. In this case; both the enable and useVbatt parameters should be set to CyFalse, and the CyU3PConnectState API should be called directly by the user application.

#### Note

The USB block on the FX3 device draws power from the VBus supply by default. If the voltage source connected on the VBus pin is unable to provide enough power for this, the user should use the CyU3PUsbVBattEnable API to have the device draw power from VBatt instead.

#### Return value

CY\_U3P\_SUCCESS if the configuration is updated properly.

CY\_U3P\_ERROR\_NOT\_STARTED if the USB block is not started.

CY\_U3P\_ERROR\_INVALID\_SEQUENCE if the USB connection is already enabled.

#### See also

[CyU3PConnectState](#)

[CyU3PUsbVBattEnable](#)

#### Parameters

<i>enable</i>	Whether VBus detection in firmware is enabled or disabled.
<i>useVbatt</i>	Whether VBatt should be used as an indication of VBus validity. This parameter is only relevant when the enable parameter is set to false.

### 5.36.7.11 CyU3PReturnStatus\_t CyU3PUsbDoRemoteWakeup ( void )

Trigger USB 2.0 Remote Wakeup signalling to the host.

#### Description

Self powered USB 2.0 devices which have been suspended can notify the host that they would like to be active again, by using remote wakeup signalling. This API call can be used to trigger remote wakeup signalling on the USB 2.0 pins of the FX3 device.

#### Return value

CY\_U3P\_SUCCESS - if the remote wakeup signalling was successful.

CY\_U3P\_ERROR\_NOT\_STARTED - if the USB driver has not been started.

CY\_U3P\_ERROR\_OPERN\_DISABLED - if the USB connection is not 2.0, is not suspended or remote wakeup is disabled.

### 5.36.7.12 void CyU3PUsbEnableEPPPrefetch ( void )

Configure the USB DMA interface to continuously pre-fetch data.

#### Description

Applications that use multiple USB IN endpoints and have high data rate may run into data corruption errors due to

an underrun of data on the RAM to USB endpoint path. This API is used to configure the USB DMA interface to pre-fetch the data from RAM more aggressively, so as to prevent this kind of data corruption.

Receiving a CY\_U3P\_USB\_EVENT\_EP\_UNDERRUN event from the USB driver is an indication that this API should be called by the application.

**Return value**

None

**See also**

[CY\\_U3P\\_USB\\_EVENT\\_EP\\_UNDERRUN](#)

### 5.36.7.13 CyU3PReturnStatus\_t CyU3PUsbEnableITPEvent ( *CyBool\_t enable* )

Function to enable/disable notification of SOF/ITP events to the application.

**Description**

Start Of Frame (SOF) and Isochronous Timestamp Packet (ITP) packets are sent by a USB host to connected devices on a periodic basis (every 125 us for high speed and super speed USB connections). While some applications may require notifications when these packets are received, they may cause unnecessary overhead in many other cases. This function is used to enable/disable notifications of SOF/ITP events from the FX3 firmware library to the application. These events are kept disabled by default and are sent through the regular USB event callback function.

**Note**

The SOF/ITP event notification is sent to the application in interrupt context so as to reduce latencies. Performing time consuming operations in these callbacks is therefore not recommended.

**Return value**

CY\_U3P\_SUCCESS - if the event change was successful.

CY\_U3P\_ERROR\_NOT\_STARTED - if the USB driver has not been started.

**See also**

[CyU3PUsbEventType\\_t](#)  
[CyU3PUSBEventCb\\_t](#)

**Parameters**

<i>enable</i>	Whether SOF/ITP event notification should be enabled.
---------------	---

### 5.36.7.14 CyU3PReturnStatus\_t CyU3PUsbEpEvtControl ( *uint8\_t epNum, uint32\_t eventMask* )

Enable or disable USB endpoint specific interrupts.

**Description**

The CyU3PUsbRegisterEpEvtCallback function is used to register a handler for and to enable endpoint specific events. A limitation of the function is that it enables the same events on all endpoints specified. There may be cases where the user is interested in different events on different endpoints. The CyU3PUsbEpEvtControl allows the user to selectively enable/disable events on specific endpoints.

All events corresponding to the bits set in eventMask will be enabled and all other events will be disabled for the endpoint epNum.

**Return value**

CY\_U3P\_SUCCESS if the endpoint events are updated as requested.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the endpoint specified is invalid or has not been enabled.

CY\_U3P\_ERROR\_NOT\_STARTED if the USB block is not enabled, or an EP event callback is not registered.

See also

[CyU3PUsbRegisterEpEvtCallback](#)  
[CyU3PUsbEpEvtType](#)

#### Parameters

<i>epNum</i>	Endpoint to be updated.
<i>eventMask</i>	Bit mask representing the events to be enabled. See <a href="#">CyU3PUsbEpEvtType</a> for the bit definitions.

### 5.36.7.15 void CyU3PUsbEpPrepare ( [CyU3PUSBSpeed\\_t curSpeed](#) )

Prepare all enabled USB device endpoints for data transfer.

#### Description

This function prepares all of the enabled USB endpoints on the FX3 device for data transfer at the current link operating speed. This sets the maximum packet size for the endpoint based on the *curSpeed* parameter, and clears the sequence numbers, data toggled and any STALL conditions associated with these endpoints. This function also sets up the endpoint memory block on the FX3 device to remove the possibility of data underruns.

This API is called internally by the USB driver in response to USB connect, reset and SET\_CONFIGURATION events; and does not generally need to be called directly by the user application.

#### Note

The caller of this function is expected to determine the current link operating speed, and then call this API with the appropriate parameter.

#### Return value

None

See also

[CyU3PUsbGetSpeed](#)  
[CyU3PSetEpPacketSize](#)  
[CyU3PUsbStall](#)

#### Parameters

<i>curSpeed</i>	Current USB connection speed.
-----------------	-------------------------------

### 5.36.7.16 [CyU3PReturnStatus\\_t CyU3PUsbEPSetBurstMode](#) ( [uint8\\_t ep](#), [CyBool\\_t burstEnable](#) )

Function to enable burst mode operation for a USB endpoint.

#### Description

Under normal operation, the FX3 device will only send data from one DMA buffer as part of one burst transaction on an USB IN endpoint. In some cases, this constraint may limit the data transfer performance achieved. This limitation can be fixed by configuring the endpoints for burst mode, where the device will combine data from multiple DMA buffers into a single burst transaction. This API is used to enable/disable burst mode on an USB endpoint.

#### Note

Setting up burst mode is not recommended for endpoints which are likely to transfer a significant number of ZLPs or short packets. This should ideally be used for cases where most of the packets transferred will be full packets.

#### Return value

[CY\\_U3P\\_SUCCESS](#) if the configuration was updated as required.

[CY\\_U3P\\_ERROR\\_BAD\\_ARGUMENT](#) if the endpoint specified is invalid.

### Parameters

<i>ep</i>	The endpoint to be configured.
<i>burstEnable</i>	Whether burst mode is to be enabled or not.

### 5.36.7.17 CyU3PReturnStatus\_t CyU3PUsbEpSetPacketsPerBuffer ( uint8\_t epNum, uint8\_t numPkts )

Set the number of maximum sized packets that an OUT endpoint buffer can accommodate.

#### Description

Some USB endpoints (isochronous or interrupt) may have maximum packet sizes that are not aligned to 512 or 1024 bytes. When one such OUT endpoint makes use of DMA buffers that can hold multiple packets worth of data, the space availability may not be calculated correctly by the FX3 device.

For example, if there is an ISOCHRONOUS endpoint with maximum packet size of 128 bytes and DMA buffers of 384 bytes are used with this endpoint; each buffer can actually accommodate 3 packets of data. However, FX3 will calculate this as 2 packets only and may end up dropping the third packet if all three packets are received in a single micro-frame.

In such cases, the count of packets that can be accommodated in one DMA buffer can be statically programmed by the user. This API allows the user to set this configuration. The value specified here should be calculated based on the DMA buffer size and the maximum packet size for the endpoint. This should be called once the SET\_CONFIGURATION or SET\_INTERFACE request that enables the interface has been received.

#### Return value

CY\_U3P\_SUCCESS if the configuration is updated properly. CY\_U3P\_ERROR\_NOT\_STARTED if the USB block has not been started. CY\_U3P\_ERROR\_BAD\_ARGUMENT if the endpoint specified is not valid, or has not been enabled.

#### See also

[CyU3PSetEpConfig](#)  
[CyU3PSetEpPacketSize](#)

### Parameters

<i>epNum</i>	Endpoint to be updated. Should be a non-zero OUT endpoint.
<i>numPkts</i>	Number of full packets that a DMA buffer can hold. Should be in the 0-31 range. Setting numPkts to 0 will cause FX3 to revert to the default buffer size calculation.

### 5.36.7.18 CyU3PReturnStatus\_t CyU3PUsbFlushEp ( uint8\_t ep )

Clear (flush) all data buffers associated with a specified endpoint.

#### Description

This function is used to clear the contents of all data buffers associated with a specified endpoint. This functionality is typically used to get rid of stale data when handling a USB bus reset or recovering an error/stall condition on the endpoint.

#### Return value

CY\_U3P\_SUCCESS - when the call is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - when the USB driver has not been started.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - if invalid parameter is passed to function.

#### See also

[CyU3PUsbStall](#)  
[CyU3PUsbResetEp](#)  
[CyU3PUsbResetEndpointMemories](#)

**Parameters**

<i>ep</i>	Endpoint number to be cleared.
-----------	--------------------------------

**5.36.7.19 CyU3PReturnStatus\_t CyU3PUsbForceFullSpeed ( CyBool\_t enable )**

Function to force the USB 2.0 device connection to full speed.

**Description**

When the FX3 is functioning as a USB device, the speed selection between super speed and other (2.0) speeds can be done through the CyU3PConnectState API call. On a 2.0 connection, the device uses the highest speed that the host supports; and this will be high speed in most cases. This API can be used to force the device to connect as a full-speed device instead of a high-speed device, and also to re-enable high speed operation.

**Note**

This API only prevents FX3 from connecting as a high speed device. If the ssEnable parameter to the [CyU3PConnectState\(\)](#) API is set to True, the device may still connect as a super-speed device.

**Return value**

CY\_U3P\_SUCCESS - if the control operation was completed successfully.

CY\_U3P\_ERROR\_NOT\_STARTED - if the USB driver has not been started.

CY\_U3P\_ERROR\_OPERN\_DISABLED - if the USB connection has already been enabled.

**Parameters**

<i>enable</i>	Whether to enable/disable the forced full speed operation.
---------------	--

**5.36.7.20 CyU3PReturnStatus\_t CyU3PUsbGetBooterVersion ( uint8\_t \* major\_p, uint8\_t \* minor\_p, uint8\_t \* patch\_p )**

Function to check the version of the boot firmware that transferred control to this firmware image.

**Description**

The full FX3 firmware image could be loaded directly by the FX3 ROM boot-loader, or by a firmware application using the boot firmware library. This API can be used to fetch the version number of the boot firmware library that loaded this firmware image.

**Return value**

CY\_U3P\_SUCCESS if the boot firmware version could be identified and returned.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if valid pointers are not provided for the Returns.

CY\_U3P\_ERROR\_FAILURE if the boot firmware version could not be identified. This can happen if the firmware application was loaded directly by the boot loader, or if the noReenum option was not used.

**Parameters**

<i>major_p</i>	Return parameter to be filled with major version of boot firmware.
<i>minor_p</i>	Return parameter to be filled with minor version of boot firmware.
<i>patch_p</i>	Return parameter to be filled with patch level of boot firmware.

**5.36.7.21 CyU3PReturnStatus\_t CyU3PUsbGetDevProperty ( CyU3PUsbDevProperty type, uint32\_t \* buf )**

Function that returns some properties of the USB device.

**Description**

This function is used to retrieve properties of the USB device such as Device address, current ITP timestamp or frame number etc.

**Return value**

CY\_U3P\_SUCCESS - when the call is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - when the USB driver has not been started.

CY\_U3P\_ERROR\_NOT\_CONFIGURED - when the USB device connection is not active.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - when the property type being queried is invalid.

**See also**

[CyU3PUsbDevProperty](#)

**Parameters**

<i>type</i>	The type of property to be queried.
<i>buf</i>	Buffer into which the property value is copied. The format of the data in the buffer depends on the property being queried. See CyU3PUsbDevProperty for details on how the property data is returned.

**5.36.7.22 CyU3PReturnStatus\_t CyU3PUsbGetEP0Data ( uint16\_t count, uint8\_t \* buffer, uint16\_t \* readCount )**

Read data associated with a control-OUT transfer.

**Description**

This function is used to get the OUT data associated with a USB control transfer. The caller is responsible for ensuring that all of the data being sent by the host is retrieved. If the caller is not able to do this, the control request should be stalled using the CyU3PUsbStall API.

Multiple calls of this function can be made to fetch all of the data associated with a single control transfer, as long as each of the partial calls fetches an integral number of full data packets from the host.

If the control request is to be failed with a STALL handshake, the stall call has to be made before all of the OUT data has been read. The request will be completed with a positive ACK as soon as all of the OUT data has been received by the device.

**Return value**

CY\_U3P\_SUCCESS - when the call is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - when the USB driver has not been started.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the buffer passed is NULL or a DMA failure occurs.

CY\_U3P\_XFER\_CANCELLED - is returned if the transaction has already been cancelled.

CY\_U3P\_ERROR\_TIMEOUT - if the scheduled data transfer is not completed within 5 seconds.

CY\_U3P\_ERROR\_ABORTED - if the data transfer is aborted due to a USB reset or another control transfer.

CY\_U3P\_ERROR\_DMA\_FAILURE - if the data transfer fails due to a DMA error.

**See also**

[CyU3PUsbSendEP0Data](#)

[CyU3PUsbAckSetup](#)

[CyU3PUsbStall](#)

**Parameters**

<i>count</i>	The length of data to be read in bytes. This should not be greater than the size requested by the host. While all of the data sent by the host needs to be read out completely, it is possible to break the read into multiple CyU3PUsbGetEP0Data API calls. In such a case, the count should be a multiple of the EP0 max. packet size (64 bytes for USB 2.0 and 512 bytes for USB 3.0).
<i>buffer</i>	Pointer to buffer where the data should be placed.
<i>readCount</i>	Output parameter which will be filled with the actual size of data read.

### 5.36.7.23 CyU3PReturnStatus\_t CyU3PUsbGetEpCfg ( uint8\_t ep, CyBool\_t \* isNak, CyBool\_t \* isStall )

Retrieve the current state of the specified endpoint.

#### Description

This function retrieves the current NAK and STALL status of the specified endpoint. The isNak return value will be CyTrue if the endpoint is forced to NAK all requests. The isStall return value will be CyTrue if the endpoint is currently stalled.

#### Return value

CY\_U3P\_SUCCESS - when the call is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - when the USB driver has not been started.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the EP specified is not valid.

#### See also

[CyU3PUsbSetEpNak](#)

[CyU3PUsbStall](#)

#### Parameters

<i>ep</i>	Endpoint number to query.
<i>isNak</i>	Return parameter which will be filled with the NAK status.
<i>isStall</i>	Return parameter which will be filled with the STALL status.

### 5.36.7.24 CyU3PReturnStatus\_t CyU3PUsbGetEpSeqNum ( uint8\_t ep, uint8\_t \* seqnum\_p )

Get the current sequence number for an endpoint.

#### Description

This function is used to query the current sequence number for a USB endpoint. This API is only valid while the device is functioning at USB 3.0.

#### Return value

CY\_U3P\_SUCCESS if the query was successful and the sequence number is returned.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the ep or the seqnum\_p parameter is invalid.

CY\_U3P\_ERROR\_INVALID\_SEQUENCE if the USB link is not at SuperSpeed.

#### See also

[CyU3PUsbSetEpSeqNum](#)

#### Parameters

<i>ep</i>	Endpoint to be queried.
<i>seqnum_p</i>	Return parameter to be filled with sequence number.

### 5.36.7.25 CyU3PReturnStatus\_t CyU3PUsbGetErrorCounts ( uint16\_t \* phy\_err\_cnt, uint16\_t \* lnk\_err\_cnt )

Function to get the number of USB 3.0 PHY and LINK error counts detected by FX3.

#### Description

The FX3 device keeps track of the number of USB 3.0 PHY and LINK errors encountered during device operation. This API can be used to query the number of PHY and LINK errors detected since the last query was completed.

Both error counts are cleared to zero whenever a query is performed.

The PHY errors tracked by FX3 are:

- 8b/10b Decode errors.
- Elastic buffer overflow/underflow errors.
- CRC errors.
- Training sequence errors.
- PHY lock loss.

The LINK errors tracked by FX3 are:

- Header packet ACK timeout.
- Link credit timeout.
- Missing LGOOD\_x / LCRD\_x error.
- Tx/Rx header sequence number errors.
- Link power management timeout (missing LAU/LXU).
- Link header sequence number / credit advertisement timeout.
- Link header or LGO\_x received before sequence number / credit advertisement.

Please note that both error counters saturate at a value of 0xFFFF.

#### **Return value**

- CY\_U3P\_SUCCESS if the query API is successful.
- CY\_U3P\_ERROR\_BAD\_ARGUMENT if valid pointers are not provided.
- CY\_U3P\_ERROR\_INVALID\_SEQUENCE if the 3.0 connection is not active

#### **Parameters**

<i>phy_err_cnt</i>	Return parameter to be filled with phy error count.
<i>lnk_err_cnt</i>	Return parameter to be filled with link error count.

### 5.36.7.26 uint16\_t CyU3PUsbGetEventLogIndex ( void )

Get the current event log buffer index.

#### **Description**

This function is used to get the current write location within the USB event log buffer. This can be used to identify the most recent values that have been added to the buffer.

#### **Return value**

Current buffer index

#### **See also**

[CyU3PUsbInitEventLog](#)

### 5.36.7.27 CyU3PReturnStatus\_t CyU3PUsbGetLinkPowerState ( CyU3PUsbLinkPowerMode \* mode\_p )

Function to get the current power state of the USB 3.0 link.

#### **Description**

This function is used to get the current power state of the USB 3.0 link. The actual link state will be returned through the mode\_p parameter if the link is in any of the Ux states. If the link is in any of the other LTSSM states, CyU3PUsbLPM\_Unknown will be returned. An error will be returned if the USB 3.0 connection is not enabled.

#### **Return value**

- CY\_U3P\_SUCCESS - the link state has been successfully retrieved.
- CY\_U3P\_ERROR\_NOT\_STARTED - if the USB driver has not been started.
- CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the USB connection is not active.
- CY\_U3P\_ERROR\_OPERN\_DISABLED - if the USB connection is not in 3.0 mode.
- CY\_U3P\_ERROR\_BAD\_ARGUMENT - if a null pointer is passed in as the mode\_p parameter.

See also

[CyU3PUsbLinkPowerMode](#)  
[CyU3PUsbSetLinkPowerState](#)

Parameters

<i>mode</i> — <i>p</i>	Return parameter that will be filled in with the current power state.
------------------------------	---

### 5.36.7.28 CyU3PUSBSpeed\_t CyU3PUsbGetSpeed ( void )

Get the connection speed at which USB is operating.

**Description**

This function is used to get the operating speed of the USB connection.

**Return value**

Current USB connection speed.

See also

[CyU3PUSBSpeed\\_t](#) [CyU3PConnectState](#) [CyU3PGetConnectState](#)

### 5.36.7.29 void CyU3PUsbInitEventLog ( uint8\_t \* buffer, uint32\_t bufSize )

Function to initiate logging of USB state changes into a circular buffer.

**Description**

While debugging USB connection/transfer failures with FX3, it is useful to have a detailed log of the USB related state changes and events. This function is used to register a memory buffer into which the state change information will be logged by the USB driver. The buffer would be treated as a circular buffer into which the data is continuously logged.

**Return value**

None

See also

[CyU3PUsbAddToEventLog](#)

Parameters

<i>buffer</i>	Pointer to memory buffer into which events are to be logged.
<i>bufSize</i>	Size of memory buffer in bytes.

### 5.36.7.30 CyBool\_t CyU3PUsbIsStarted ( void )

This function returns whether the USB device module has been started.

**Description**

Since there can be various modes of USB operations this API returns whether CyU3PUsbStart was invoked.

**Return value**

CyTrue if the USB module has been started.

CyFalse if the USB module is not running.

#### See also

[CyU3PUsbStart](#)  
[CyU3PUsbStop](#)

### 5.36.7.31 CyU3PReturnStatus\_t CyU3PUsbJumpBackToBooter ( uint32\_t address )

Function to transfer control back to the FX3 2-stage bootloader.

#### Description

This function is used to transfer control back to the FX3 2-stage bootloader. The caller is expected to do cleanup of modules other than USB prior to this call. D-Cache needs to be cleaned before calling this function. The entry address for the booter firmware can be obtained by using the verbose option of the elf2img converter tool.

It is expected that all Serial Peripheral blocks are turned OFF before this function is called. Only the GPIO block can be left ON, if the booter had previously been configured to leave the GPIO block ON.

#### Return value

CY\_U3P\_ERROR\_INVALID\_SEQUENCE - If this function is called before the serial peripherals are turned off.

CY\_U3P\_ERROR\_OPERN\_DISABLED - If the FX3 2-stage booter doesn't support switching back.

Otherwise this is a non-returning call.

#### See also

[CyU3PUsbSetBooterSwitch](#)

#### Parameters

<code>address</code>	Address of the FX3 2-stage bootloader's entry point.
----------------------	--

### 5.36.7.32 void CyU3PUsbLinkComplianceControl ( CyBool\_t isEnabled )

Disable USB 3.0 electrical compliance mode support.

#### Description

The USB spec requires USB devices to check for USB 3.0 support on the upstream device/hub by checking for a valid termination on the SSTX (far side RX) pins. The spec defines a valid termination impedance of under 30 Ohms and an invalid (no USB 3.0 support) termination impedance above 25 KOhms.

FX3 has a receiver detection threshold of approximately 150 Ohms, and will enter USB 3.0 electrical compliance test mode if a host/hub offers impedance under 150 Ohms while USB 3.0 is disabled. This behavior can be modified by calling this API to disable compliance test mode support. If compliance mode is disabled, FX3 will enter USB 2.0 link mode if the Polling.LFPS handshake times out.

#### Return value

None

#### Parameters

<code>isEnabled</code>	Whether link compliance test mode is enabled.
------------------------	---

### 5.36.7.33 CyU3PReturnStatus\_t CyU3PUsbLPMDisable ( void )

Function to prevent FX3 from entering U1/U2 states.

#### Description

The USB driver in FX3 firmware runs a state machine that determines whether entry to U1/U2 low power states is to be allowed. At most times, this decision is made automatically by the FX3 device itself based on whether it has any pending packets to go out. At other times, the decision is made by the firmware based on whether a ForceLINKPM\_ACCEPT command has been received, and also the amount of time that has elapsed since the last exit from U1/U2.

This function allows the user to override the state machine, and ensure that entry to U1/U2 states will be systematically denied by FX3 until the CyU3PUsbLPMEnable call is made.

This call also disables the acceptance of LPM-L1 entry requests when FX3 is functioning as a Hi-Speed or a full speed device.

#### Note

Indiscriminate use of this function can result in USB compliance test failures. Please ensure that the LPM functionality is enabled every time an USB connect or reset event is received. This can be disabled again after ensuring that the device is functioning in a performance critical mode.

#### Return value

CY\_U3P\_SUCCESS - if the operation is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - if the USB driver has not been started.

CY\_U3P\_ERROR\_OPERN\_DISABLED - if the USB connection has not been enabled.

#### See also

[CyU3PUsbLPMEnable](#)

### 5.36.7.34 CyU3PReturnStatus\_t CyU3PUsbLPMEnable ( void )

Function to re-enable automated handling of U1/U2 state entry.

#### Description

This function is used to re-enable the USB driver state machine that governs the handling of U1/U2 requests from the USB host. This function removes the override that is specified using the CyU3PUsbLPMDisable call.

This function also enables the acceptance of LPM-L1 entry requests when FX3 is functioning as a Hi-Speed or a full speed device.

#### Note

It is expected that this call is made every time the application receives a USB connect or reset event, if a CyU3PUsbLPMDisable call has been made previously.

#### Return value

CY\_U3P\_SUCCESS - if the operation is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - if the USB driver has not been started.

CY\_U3P\_ERROR\_OPERN\_DISABLED - if the USB connection has not been enabled.

#### See also

[CyU3PUsbLPMDisable](#)

### 5.36.7.35 CyU3PReturnStatus\_t CyU3PUsbMapStream ( uint8\_t ep, uint8\_t socketNum, uint16\_t streamId )

Map a socket to stream of the specified endpoint.

#### Description

This function is used to map the stream of the specified endpoint to the socket passed as a parameter, this can be done if the socket is not already mapped.

**Return value**

CY\_U3P\_SUCCESS - when the call is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - when the USB driver has not been started.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - if invalid parameter is passed to function.

**See also**

[CyU3PSetEpConfig](#)  
[CyU3PUsbChangeMapping](#)

**Parameters**

<i>ep</i>	Endpoint number for which the stream is to be mapped.
<i>socketNum</i>	Socket number for mapping the stream.
<i>streamId</i>	StreamId to be mapped.

5.36.7.36 void CyU3PUsbRegisterEpEvtCallback ( *CyU3PUsbEpEvtCb\_t cbFunc*, *uint32\_t eventMask*, *uint16\_t outEpMask*, *uint16\_t inEpMask* )

Register a callback function for notification of USB endpoint events.

**Description**

This function is used to register a callback function that will be invoked for notification of USB endpoint events during device operation.

**Return value**

None

**See also**

[CyU3PUsbEpEvtCb\\_t](#)  
[CyU3PUsbEpEvtType](#)  
[CyU3PUsbEpEvtControl](#)

**Parameters**

<i>cbFunc</i>	Callback function pointer.
<i>eventMask</i>	Bitmap variable representing the events that should be enabled.
<i>outEpMask</i>	Bitmap variable representing the OUT endpoints whose events are to be enabled. Bit 1 represents EP 1-OUT, 2 represents EP 2-OUT and so on.
<i>inEpMask</i>	Bitmap variable representing the IN endpoints whose events are to be enabled.

5.36.7.37 void CyU3PUsbRegisterEventCallback ( *CyU3PUSBEventCb\_t callback* )

Register a USB event callback function.

**Description**

This function registers a USB event callback function with the USB driver. This function will be invoked by the driver every time an USB event of interest happens.

**Return value**

None

## See also

[CyU3PUSBEVENTCb\\_t](#)  
[CyU3PUsbEventType\\_t](#)  
[CyU3PUsbStart](#)

## Parameters

<i>callback</i>	Event callback function pointer.
-----------------	----------------------------------

5.36.7.38 void CyU3PUsbRegisterLPMRequestCallback ( [CyU3PUsbLPMReqCb\\_t cb](#) )

Register a USB 3.0 LPM request handler callback.

**Description**

This function is used to register a callback that handles Link Power state requests from the USB host. In the absence of a registered callback, all U1/U2 LPM entry requests will be failed by the FX3 USB driver.

**Return value**

None

## See also

[CyU3PUsbLPMReqCb\\_t](#)  
[CyU3PUsbSetLinkPowerState](#)  
[CyU3PUsbGetLinkPowerState](#)  
[CyU3PUsbLPMDisable](#)  
[CyU3PUsbLPMEnable](#)

## Parameters

<i>cb</i>	Callback function pointer.
-----------	----------------------------

5.36.7.39 void CyU3PUsbRegisterSetupCallback ( [CyU3PUSBSetupCb\\_t callback](#), [CyBool\\_t fastEnum](#) )

Register a USB setup request handler.

**Description**

This function is used to register a USB setup request handler with the USB driver. The fastEnum parameter specifies whether this setup handler should be used only for unknown setup requests or for all USB setup requests.

**Return value**

None

## See also

[CyU3PUSBSetupCb\\_t](#)  
[CyU3PUsbStart](#)

## Parameters

<i>callback</i>	Setup request handler function.
<i>fastEnum</i>	Select fast enumeration mode by setting CyTrue.

### 5.36.7.40 CyU3PReturnStatus\_t CyU3PUsbResetEndpointMemories ( void )

Reset and re-initialize the endpoint memory block on FX3.

#### Description

Some transfer error conditions can leave the endpoint memories on FX3 in a bad state. This API is used to reset and re-initialize the memory block, so that USB data transfers can resume. This API should only be used when the application is recovering from an error condition, such as a transfer stall due to backflow from the USB host.

Please note that this API leaves the sequence numbers of all endpoints unaffected. If this is being done as part of a EP reset handling, the sequence number should be explicitly cleared using the CyU3PUsbStall API.

#### Return value

CY\_U3P\_SUCCESS in case of successful reset and re-initialization.

CY\_U3P\_ERROR\_NOT\_STARTED if the USB block has not been started.

### 5.36.7.41 CyU3PReturnStatus\_t CyU3PUsbResetEp ( uint8\_t ep )

Resets the specified endpoint.

#### Description

This function is used to reset USB endpoints while functioning at Super speed. The reset clears error conditions on the endpoint logic and prepares the endpoint for data transfer.

This function does nothing if called while in a USB 2.0 connection.

#### Return value

CY\_U3P\_SUCCESS - when the call is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - when the endpoint number is invalid.

#### See also

[CyU3PUsbStall](#)

[CyU3PUsbFlushEp](#)

[CyU3PUsbResetEndpointMemories](#)

#### Parameters

<i>ep</i>	Endpoint number to be cleared.
-----------	--------------------------------

### 5.36.7.42 CyU3PReturnStatus\_t CyU3PUsbSendAckTP ( uint8\_t ep, uint8\_t numP, uint16\_t bulkStream )

Function to send an ACK TP to a USB 3.0 host.

#### Description

This function allows the firmware to generate an ACK TP for the specified endpoint. This function is only provided for debug and testing purposes, and should be used with caution.

#### Return value

CY\_U3P\_SUCCESS - when the call is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - if the USB driver has not been started.

CY\_U3P\_ERROR\_OPERN\_DISABLED - if the USB connection is currently not in USB 3.0 mode.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the endpoint specified is invalid.

#### See also

[CyU3PUsbSendErdy](#)

**Parameters**

<i>ep</i>	Endpoint for which the ACK TP is to be generated. Bit 7 of the endpoint indicates direction.
<i>nump</i>	NUMP (Number of Packets) value to associate with the TP. Please refer to USB specification for meaning and interpretation.
<i>bulkStream</i>	Stream number for which the ACK TP is to be generated. Is only valid for stream enabled bulk endpoints and is ignored otherwise.

5.36.7.43 **CyU3PReturnStatus\_t CyU3PUsbSendDevNotification ( uint8\_t *notificationType*, uint32\_t *param0*, uint32\_t *param1* )**

Function to send a DEV\_NOTIFICATION Transaction Packet to the host.

**Description**

This API allows the user to send DEV\_NOTIFICATION Transaction packets to the USB 3.0 host. The Notification Type and Notification Type Specific fields are accepted as parameters. None of the parameters are validated by the API, so the caller needs to ensure validity of these values.

**Note**

This API has to be called by the user after ensuring that the link is in U0 state. If the link is in U1/U2 and a TP has to be sent, the CyU3PUsbSetLinkPowerState API should be called first to trigger a recovery to U0.

**Return value**

CY\_U3P\_SUCCESS - if the DEV\_NOTIFICATION TP was successfully send to the host.

CY\_U3P\_ERROR\_NOT\_STARTED - if the USB driver has not been started.

CY\_U3P\_ERROR\_OPERN\_DISABLED - if the USB connection is not in Super Speed mode or if the link is not in U0 state.

**Parameters**

<i>notificationType</i>	Notification type - No validation is performed by the API.
<i>param0</i>	Notification Type Specific Data - DWORD1[31:8].
<i>param1</i>	Notification Type Specific Data - DWORD2[31:0].

5.36.7.44 **CyU3PReturnStatus\_t CyU3PUsbSendEP0Data ( uint16\_t *count*, uint8\_t \* *buffer* )**

Send data to the USB host via EP0-IN.

**Description**

This function is used to respond to USB control requests with an associated IN data transfer phase. The data in the buffer will be sent to the host in multiple packets of appropriate size as per the USB connection speed. Multiple calls of this function can be made to respond to a single control request as long as each call sends an integral number of full packets to the host. Any send call that results in a short packet will terminate the control transfer.

If the data in the buffer ends in a full packet, a zero length packet (ZLP) should be sent to the host to terminate the control transfer.

**Return value**

CY\_U3P\_SUCCESS - when the call is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - when the USB driver has not been started.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the buffer passed is NULL or a DMA failure occurs.

CY\_U3P\_XFER\_CANCELLED - is returned if the transaction has already been cancelled.

CY\_U3P\_ERROR\_TIMEOUT - if the scheduled data transfer is not completed within 5 seconds.

CY\_U3P\_ERROR\_ABORTED - if the data transfer is aborted due to a USB reset or another control transfer.

CY\_U3P\_ERROR\_DMA\_FAILURE - if the data transfer fails due to a DMA error.

See also

[CyU3PUSBSetupCb\\_t](#)  
[CyU3PUsbGetEP0Data](#)  
[CyU3PUsbAckSetup](#)  
[CyU3PUsbStall](#)

Parameters

<i>count</i>	The amount of data in bytes.
<i>buffer</i>	Pointer to buffer containing the data.

#### 5.36.7.45 CyU3PReturnStatus\_t CyU3PUsbSendErdy ( uint8\_t ep, uint16\_t bulkStream )

Function to send an ERDY TP to a USB 3.0 host.

##### Description

This function allows the firmware to generate an ERDY TP for a specified endpoint. Under normal operation, ERDY TPs are automatically generated by the FX3 device as and when required. This function is provided to support exceptional cases where the firmware application needs to generate a specific ERDY TP.

##### Return value

CY\_U3P\_SUCCESS - when the call is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - if the USB driver has not been started.  
 CY\_U3P\_ERROR\_OPERN\_DISABLED - if the USB connection is currently not in USB 3.0 mode.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the endpoint specified is invalid.

See also

[CyU3PUsbSendNrdy](#)

Parameters

<i>ep</i>	Endpoint for which the ERDY TP is to be generated. Bit 7 of the endpoint indicates direction.
<i>bulkStream</i>	Stream number for which the ERDY TP is to be generated. Is only valid for stream enabled bulk endpoints and is ignored otherwise.

#### 5.36.7.46 CyU3PReturnStatus\_t CyU3PUsbSendNrdy ( uint8\_t ep, uint16\_t bulkStream )

Function to send an NRDY TP to a USB 3.0 host.

##### Description

This function allows the firmware to generate an NRDY TP for a specified endpoint. Under normal operation, NRDY TPs are automatically generated by the FX3 device as and when required. This function is provided to support exceptional cases where the firmware application needs to generate a specific NRDY TP.

##### Return value

CY\_U3P\_SUCCESS - when the call is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - if the USB driver has not been started.  
 CY\_U3P\_ERROR\_OPERN\_DISABLED - if the USB connection is currently not in USB 3.0 mode.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the endpoint specified is invalid.

See also

[CyU3PUsbSendErdy](#)

**Parameters**

<i>ep</i>	Endpoint for which the NRDY TP is to be generated. Bit 7 of the endpoint indicates direction.
<i>bulkStream</i>	Stream number for which the NRDY TP is to be generated. Is only valid for stream enabled bulk endpoints and is ignored otherwise.

5.36.7.47 void CyU3PUsbSetBooterSwitch ( *CyBool\_t enable* )

Function to enable/disable switching control back to the FX3 2-stage bootloader.

**Description**

This function is used to enable/disable the switching of control back to the FX3 2-stage bootloader. The function is expected to be called prior to [CyU3PUsbJumpBackToBooter\(\)](#).

**Return value**

None

**See also**

[CyU3PUsbJumpBackToBooter](#)  
[CyU3PUsbStart](#)

**Parameters**

<i>enable</i>	CyTrue - Enable switching to booter; CyFalse - Disable switching to booter.
---------------	---

5.36.7.48 CyU3PReturnStatus\_t CyU3PUsbSetDesc ( *CyU3PUSBSetDescType\_t desc\_type*, *uint8\_t desc\_index*, *uint8\_t \* desc* )

Register a USB descriptor with the driver.

**Description**

This function is used to register a USB descriptor with the USB driver. The driver is capable of remembering one descriptor each of the various supported types as well as up to 16 different string descriptors.

The driver only stores the descriptor pointers that are passed in to this function, and does not make copies of the descriptors. The caller therefore should not free up these descriptor buffers while the USB driver is active.

**Return value**

CY\_U3P\_SUCCESS - when the call is successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - when the USB driver has not been started.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - when the buffer pointer is invalid.  
 CY\_U3P\_ERROR\_BAD\_DESCRIPTOR\_TYPE - When the descriptor type is not valid.  
 CY\_U3P\_ERROR\_BAD\_INDEX - if the string descriptor index exceeds 16.

**See also**

[CyU3PUSBSetDescType\\_t](#)  
[CyU3PUsbRegisterSetupCallback](#)

**Parameters**

<i>desc_type</i>	Type of descriptor to register.
<i>desc_index</i>	Descriptor index: Used only for string descriptors.
<i>desc</i>	Pointer to buffer containing the descriptor.

### 5.36.7.49 CyU3PReturnStatus\_t CyU3PUsbSetEpNak ( uint8\_t *ep*, CyBool\_t *nak* )

Force or clear the NAK status on the specified endpoint.

#### Description

All bulk and interrupt endpoints on the FX3 device can be forced to NAK any host request while the corresponding function driver is not prepared for data transfers. This function is used to force the specified endpoint to NAK all requests, or to clear the NAK status and allow data transfers to proceed.

#### Return value

CY\_U3P\_SUCCESS - when the call is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - when the USB driver has not been started.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the EP specified is not valid.

#### See also

[CyU3PUsbGetEpCfg](#)

#### Parameters

<i>ep</i>	Endpoint to be updated.
<i>nak</i>	CyTrue to force NAK, CyFalse to clear NAK.

### 5.36.7.50 CyU3PReturnStatus\_t CyU3PUsbSetEpPktMode ( uint8\_t *ep*, CyBool\_t *pktMode* )

Select whether a OUT endpoint will function in packet (one buffer per packet) mode or not.

#### Description

USB OUT endpoints on the FX3 device are setup by default to collect multiple full data packets into a single DMA buffer (size permitting). The DMA buffer into which data is received is wrapped up and made available to the consumer only if it is filled up, or if a short packet is received on the endpoint. The endpoint can also be configured for packet mode, where each DMA buffer will only hold one packet worth of data (regardless of the actual packet size) and gets wrapped up as soon as any data packet is received.

This API is used to switch the USB OUT endpoint between packet mode and transfer mode as required. This API only operates on non-control OUT endpoints.

#### See also

[CyU3PSetEpPacketSize](#)

#### Return value

CY\_U3P\_SUCCESS if the endpoint mode was updated as requested.

CY\_U3P\_ERROR\_NOT\_STARTED if the USB driver has not been started.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the endpoint specified is invalid.

#### Parameters

<i>ep</i>	Endpoint number to be configured.
<i>pktMode</i>	Whether the endpoint will function in packet mode or not.

### 5.36.7.51 CyU3PReturnStatus\_t CyU3PUsbSetEpSeqNum ( uint8\_t *ep*, uint8\_t *seqnum* )

Set the active sequence number for an endpoint.

**Description**

This function is used to set the active sequence number for a USB 3.0 endpoint to a desired value.

**Return value**

CY\_U3P\_SUCCESS if the update operation is successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the ep or seqnum parameter is invalid.

CY\_U3P\_ERROR\_INVALID\_SEQUENCE if the USB link is not at SuperSpeed.

**See also**

[CyU3PUsbGetEpSeqNum](#)

**Parameters**

<i>ep</i>	Endpoint to be updated.
<i>seqnum</i>	Sequence number to be selected.

**5.36.7.52 CyU3PReturnStatus\_t CyU3PUsbSetEpSuspDisableMask ( uint16\_t noSuspMask )**

Select endpoints that should not be suspended to allow EP0 transfers to complete.

**Description**

When FX3 is operating in High Speed mode, other IN endpoints are suspended while any EP0-IN data transfers are being handled. This scheme is implemented as a work-around to ensure that no data corruption occurs. However, this work-around can affect the on-time data transfer on periodic endpoints. This API can be used to specify that some IN endpoints should be exempted from the implementation of this suspend work-around.

**Note**

This list of endpoints for whom DMA suspend is disabled is cleared when FX3 goes through a USB bus reset. Therefore, the setting needs to be done afresh every time the device goes through SET\_CONFIGURATION.

**Return value**

CY\_U3P\_SUCCESS if the configuration is updated. CY\_U3P\_ERROR\_NOT\_STARTED if the USB block has not been started.

**See also**

[CyU3PUsbSendEP0Data](#)

**Parameters**

<i>noSuspMask</i>	Bitmask representing IN endpoints that should not be suspended while handling EP0-IN data transfers.
-------------------	--

**5.36.7.53 CyU3PReturnStatus\_t CyU3PUsbSetLinkPowerState ( CyU3PUsbLinkPowerMode link\_mode )**

Function to request a device entry into one of the U0, U1 or U2 link power modes.

**Description**

This function is used to request the FX3 USB driver to place the USB 3.0 link in a desired (U0, U1 or U2) power state. The U3 power state is not supported because U3 entry can only be triggered by the host. The request to move into U1 or U2 will only be allowed while the link is currently in the U0 state. The request to move into U0 will be allowed while the link is in the U1, U2 or U3 states.

**Return value**

CY\_U3P\_SUCCESS - when the requested link power state switch request has been initiated.

CY\_U3P\_ERROR\_NOT\_STARTED - if the USB driver has not been started.

CY\_U3P\_ERROR\_NOT\_CONFIGURED - if the USB connection is not active.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - if the link state specified is invalid.

CY\_U3P\_ERROR\_OPERN\_DISABLED - if the USB connection is not in USB 3.0 mode or if the current link power state does not allow this transition.

#### See also

[CyU3PUsbLinkPowerMode](#)  
[CyU3PUsbGetLinkPowerState](#)  
[CyU3PUsbLPMEnable](#)  
[CyU3PUsbLPMDisable](#)

#### Parameters

<i>link_mode</i>	Desired link power state.
------------------	---------------------------

### 5.36.7.54 CyU3PReturnStatus\_t CyU3PUsbSetTxDeemphasis ( uint32\_t value )

Set the Tx de-emphasis setting for the USB 3.0 signals.

#### Description

This API sets the Tx de-emphasis level used by FX3 on the USB 3.0 interface. Please use this API with caution. This API is expected to be called before calling the [CyU3PConnectState\(\)](#) API to enable USB connections.

#### Return value

CY\_U3P\_SUCCESS if the setting is updated as expected.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the value specified is out of range.

#### Parameters

<i>value</i>	TX De-emphasis value. Should be less than 0x1F. Default value is 0x11.
--------------	--

### 5.36.7.55 CyU3PReturnStatus\_t CyU3PUsbSetTxSwing ( uint32\_t swing )

Set the Tx amplitude range for the USB 3.0 signals.

#### Description

This API sets the Tx amplitude used by FX3 on the USB 3.0 interface. Please use this API with caution. The device has only been tested to work properly under the default swing setting of 0.9V (swing value set to 90). This API is expected to be called before calling the [CyU3PConnectState\(\)](#) API to enable USB connections.

#### Return value

CY\_U3P\_SUCCESS if the setting is updated as expected.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the swing value specified is out of range.

#### Parameters

<i>swing</i>	TX Amplitude swing in 10 mV units. Should be less than 1.28V.
--------------	---

### 5.36.7.56 CyU3PReturnStatus\_t CyU3PUsbSetXfer ( uint8\_t ep, uint16\_t count )

Set the expected data transfer size on a USB 2.0 endpoint.

**Description**

This function is used to define an expected transfer size on a USB 2.0 endpoint (does not work for USB 3.0). The hardware will only generate the SLP event if a short packet is received on the endpoint while it has a non-zero transfer count.

**Return value**

None

**Parameters**

<i>ep</i>	Endpoint number and direction.
<i>count</i>	Expected transfer size. Please note that the transfer count is only used to generate the SLP event on the endpoint, and does not affect the ability of the endpoint to actually transfer more data.

**5.36.7.57 void CyU3PUsbSSCDisable( void )**

Disable Spread-Spectrum Clocking support in the USB block.

**Description**

The USB 3.0 block on FX3 has Spread-Spectrum Clocking support enabled by default. This API is used to disable SSC support on the FX3 device.

**5.36.7.58 CyU3PReturnStatus\_t CyU3PUsbStall( uint8\_t ep, CyBool\_t stall, CyBool\_t toggle )**

Set or clear the stall status of an endpoint.

**Description**

This function is to set or clear the stall status of a given endpoint. This function is to be used in response to SET\_FEATURE and CLEAR\_FEATURE requests from the host as well as for interface specific error handling. When the stall condition is being cleared, the data toggles for the endpoint can also be cleared. While an option is provided to leave the data toggles unmodified, this should only be used under specific conditions as recommended by Cypress.

**Return value**

CY\_U3P\_SUCCESS - when the call is successful.

CY\_U3P\_ERROR\_NOT\_STARTED - when the USB driver has not been started.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - when the endpoint is invalid.

**See also**

[CyU3PSetEpConfig](#)  
[CyU3PUsbGetEpCfg](#)

**Parameters**

<i>ep</i>	Endpoint number to be modified.
<i>stall</i>	CyTrue: Set the stall condition, CyFalse: Clear the stall
<i>toggle</i>	CyTrue: Clear the data toggles in a Clear Stall call. Note that the toggle parameter is ignored when the stall parameter is set to CyTrue.

**5.36.7.59 CyU3PReturnStatus\_t CyU3PUsbStart( void )**

Start the USB driver.

**Description**

This function is used to start the USB device mode driver in the FX3 device and also to create the DMA channels

required for the control endpoint.

**Return value**

CY\_U3P\_SUCCESS - when the call is successful.  
CY\_U3P\_ERROR\_ALREADY\_STARTED - when the USB driver has already been started.  
CY\_U3P\_ERROR\_CHANNEL\_CREATE\_FAILED - when the DMA channel creation for the control endpoint fails.  
CY\_U3P\_ERROR\_NO\_REENUM\_REQUIRED - if the USB block has been left running by the boot firmware image..

See also

[CyU3PUsbStop](#)  
[CyU3PUsbIsStarted](#)  
[CyU3PUsbVBattEnable](#)  
[CyU3PUsbRegisterEventCallback](#)  
[CyU3PUsbRegisterSetupCallback](#)  
[CyU3PUsbRegisterEpEvtCallback](#)  
[CyU3PUsbRegisterLPMRequestCallback](#)  
[CyU3PUsbSetDesc](#)  
[CyU3PConnectState](#)

### 5.36.7.60 CyU3PReturnStatus\_t CyU3PUsbStop ( void )

Stop the USB driver.

**Description**

This function stops the USB driver on the FX3 device. It also frees up the DMA channels created for the control endpoint.

**Return value**

CY\_U3P\_SUCCESS - when the call is successful.  
CY\_U3P\_ERROR\_NOT\_STARTED - when the USB driver has not been started.  
Other DMA error codes - when the control endpoint DMA channel destroy fails.

See also

[CyU3PUsbStart](#)  
[CyU3PUsbIsStarted](#)

### 5.36.7.61 CyU3PReturnStatus\_t CyU3PUsbVBattEnable ( CyBool\_t enable )

Configure USB block on FX3 to work off Vbatt power instead of Vbus.

**Description**

The USB block on the FX3 device can be configured to work off Vbus power or Vbatt power, with the Vbus power being the default setting. This function is used to enable/disable the Vbatt power input to the USB block.

This API needs to be called before the CyU3PConnectState API is called.

**Return value**

CY\_U3P\_SUCCESS - when the call is successful.  
CY\_U3P\_ERROR\_NOT\_STARTED - when the USB driver has not been started.  
CY\_U3P\_ERROR\_INVALID\_SEQUENCE - when the API is called after CyU3PConnectState has been called.

See also

[CyU3PUsbStart](#)  
[CyU3PConnectState](#)

**Parameters**

<code>enable</code>	CyTrue: Work off Vbatt, CyFalse: Work off Vbus.
---------------------	---

**5.37 firmware/u3p\_firmware/inc/cyu3usbconst.h File Reference**

This file defines constants that are derived from the USB specifications, for the use of the USB driver and user firmware.

```
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

**Typedefs**

- **typedef enum CyU3PUsbSetupCmds CyU3PUsbSetupCmds**  
*Standard device request codes.*
- **typedef enum CyU3PUsbEpType\_t CyU3PUsbEpType\_t**  
*Enumeration of the endpoint types.*
- **typedef enum CyU3PUsbDescType CyU3PUsbDescType**  
*Enumeration of descriptor types.*
- **typedef enum CyU3PUsbDevCapType CyU3PUsbDevCapType**  
*Device capability type codes.*
- **typedef enum CyU3PUsb3PacketType CyU3PUsb3PacketType**  
*USB 3.0 packet type codes.*
- **typedef enum CyU3PUsb3TpSubType CyU3PUsb3TpSubType**  
*USB 3.0 transaction packet sub type codes.*
- **typedef enum CyU3PUsbFeatureSelector CyU3PUsbFeatureSelector**  
*List of USB Feature selector codes.*
- **typedef enum CyU3PUsb2TestModes CyU3PUsb2TestModes**  
*List of USB 2.0 test modes.*
- **typedef enum CyU3PUsbLinkState\_t CyU3PUsbLinkState\_t**  
*Link state machine states.*

**Enumerations**

- **enum CyU3PUsbSetupCmds {**  
**CY\_U3P\_USB\_SC\_GET\_STATUS** = 0x00, **CY\_U3P\_USB\_SC\_CLEAR\_FEATURE**, **CY\_U3P\_USB\_SC\_RESERVED**, **CY\_U3P\_USB\_SC\_SET\_FEATURE**,  
**CY\_U3P\_USB\_SC\_SET\_ADDRESS** = 0x05, **CY\_U3P\_USB\_SC\_GET\_DESCRIPTOR**, **CY\_U3P\_USB\_SC\_SET\_DESCRIPTOR**, **CY\_U3P\_USB\_SC\_GET\_CONFIGURATION**,  
**CY\_U3P\_USB\_SC\_SET\_CONFIGURATION**, **CY\_U3P\_USB\_SC\_GET\_INTERFACE**, **CY\_U3P\_USB\_SC\_SET\_INTERFACE**, **CY\_U3P\_USB\_SC\_SYNC\_FRAME**,  
**CY\_U3P\_USB\_SC\_SET\_SEL** = 0x30, **CY\_U3P\_USB\_SC\_SET\_ISOC\_DELAY** }  
*Standard device request codes.*
- **enum CyU3PUsbEpType\_t { CY\_U3P\_USB\_EP\_CONTROL** = 0, **CY\_U3P\_USB\_EP\_ISO** = 1, **CY\_U3P\_USB\_EP\_BULK** = 2, **CY\_U3P\_USB\_EP\_INTR** = 3 }  
*Enumeration of the endpoint types.*

- enum `CyU3PUsbDescType` {
 `CY_U3P_USB_DEVICE_DESCR` = 0x01, `CY_U3P_USB_CONFIG_DESCR`, `CY_U3P_USB_STRING_DESCRIPTOR`, `CY_U3P_USB_INTRFC_DESCRIPTOR`,
 `CY_U3P_USB_ENDPNT_DESCRIPTOR`, `CY_U3P_USB_DEVQUAL_DESCRIPTOR`, `CY_U3P_USB_OTHERSPEED_DESCRIPTOR`, `CY_U3P_USB_INTRFC_POWER_DESCRIPTOR`,
 `CY_U3P_BOS_DESCRIPTOR` = 0x0F, `CY_U3P_DEVICE_CAPB_DESCRIPTOR`, `CY_U3P_USB_HID_DESCRIPTOR` = 0x21,
 `CY_U3P_USB_REPORT_DESCRIPTOR`,
 `CY_U3P_SS_EP_COMPN_DESCRIPTOR` = 0x30, `CY_U3P_USB_DEVICE_DESCRIPTOR` = 0x01, `CY_U3P_USB_CONFIGURATION_DESCRIPTOR`, `CY_U3P_USB_STRING_DESCRIPTOR`,
 `CY_U3P_USB_INTRFC_DESCRIPTOR`, `CY_U3P_USB_ENDPNT_DESCRIPTOR`, `CY_U3P_USB_DEVQUAL_DESCRIPTOR`, `CY_U3P_USB_OTHERSPEED_DESCRIPTOR`,
 `CY_U3P_USB_INTRFC_POWER_DESCRIPTOR`, `CY_U3P_USB_OTG_DESCRIPTOR`, `CY_U3P_BOS_DESCRIPTOR` = 0x0F,
 `CY_U3P_DEVICE_CAPB_DESCRIPTOR`,
 `CY_U3P_USB_HID_DESCRIPTOR` = 0x21, `CY_U3P_USB_REPORT_DESCRIPTOR`, `CY_U3P_SS_EP_COMPN_DESCRIPTOR` = 0x30 }

*Enumeration of descriptor types.*

- enum `CyU3PUsbDevCapType` { `CY_U3P_WIRELESS_USB_CAPB_TYPE` = 0x01, `CY_U3P_USB2_EXTENSION_CAPB_TYPE`, `CY_U3P_SS_USB_CAPB_TYPE`, `CY_U3P_CONTAINER_ID_CAPB_TYPE` }

*Device capability type codes.*

- enum `CyU3PUsb3PacketType` { `CY_U3P_USB3_PACK_TYPE_LMP` = 0x00, `CY_U3P_USB3_PACK_TYPE_TE`, `CY_U3P_USB3_PACK_TYPE_TP` = 0x04, `CY_U3P_USB3_PACK_TYPE_DPH` = 0x08, `CY_U3P_USB3_PACK_TYPE_ITP` = 0x0C }

*USB 3.0 packet type codes.*

- enum `CyU3PUsb3TpSubType` {
 `CY_U3P_USB3_TP_SUBTYPE_RES` = 0, `CY_U3P_USB3_TP_SUBTYPE_ACK`, `CY_U3P_USB3_TP_STATUS`, `CY_U3P_USB3_TP_ERDY`,
 `CY_U3P_USB3_TP_SUBTYPE_STATUS`, `CY_U3P_USB3_TP_SUBTYPE_STALL`, `CY_U3P_USB3_TP_NOTICE`, `CY_U3P_USB3_TP_PING`,
 `CY_U3P_USB3_TP_PINGRSP` }

*USB 3.0 transaction packet sub type codes.*

- enum `CyU3PUsbFeatureSelector` {
 `CY_U3P_USBX_FS_EP_HALT` = 0, `CY_U3P_USB2_FS_REMOTE_WAKE` = 1, `CY_U3P_USB2_FS_TEST_MODE` = 2, `CY_U3P_USB2_OTG_B_HNP_ENABLE` = 3,
 `CY_U3P_USB2_OTG_A_HNP_SUPPORT` = 4, `CY_U3P_USB3_FS_U1_ENABLE` = 48, `CY_U3P_USB3_FS_U2_ENABLE` = 49, `CY_U3P_USB3_FS_LTM_ENABLE` = 50 }

*List of USB Feature selector codes.*

- enum `CyU3PUsb2TestModes` {
 `CY_U3P_USB2_TEST_NONE` = 0, `CY_U3P_USB2_TEST_J`, `CY_U3P_USB2_TEST_K`, `CY_U3P_USB2_TEST_SE0_NAK`,
 `CY_U3P_USB2_TEST_PACKET`, `CY_U3P_USB2_TEST_FORCE_EN` }

*List of USB 2.0 test modes.*

- enum `CyU3PUsbLinkState_t` {
 `CY_U3P_UIB_LNK_STATE_SSDISABLED` = 0x00, `CY_U3P_UIB_LNK_STATE_RXDETECT_RES` = 0x01,
 `CY_U3P_UIB_LNK_STATE_RXDETECT_ACT` = 0x02, `CY_U3P_UIB_LNK_STATE_RXDETECT_QUT` = 0x03,
 `CY_U3P_UIB_LNK_STATE_SSINACT_QUT` = 0x04, `CY_U3P_UIB_LNK_STATE_SSINACT_DET` = 0x05,
 `CY_U3P_UIB_LNK_STATE_POLLING_LFPS` = 0x08, `CY_U3P_UIB_LNK_STATE_POLLING_RxEQ` = 0x09,
 `CY_U3P_UIB_LNK_STATE_POLLING_ACT` = 0x0A, `CY_U3P_UIB_LNK_STATE_POLLING_IDLE` = 0x0C,
 `CY_U3P_UIB_LNK_STATE_U0` = 0x10, `CY_U3P_UIB_LNK_STATE_U1` = 0x11,
 `CY_U3P_UIB_LNK_STATE_U2` = 0x12, `CY_U3P_UIB_LNK_STATE_U3` = 0x13, `CY_U3P_UIB_LNK_STATE_COMP` = 0x17,
 `CY_U3P_UIB_LNK_STATE_RECov_ACT` = 0x18, `CY_U3P_UIB_LNK_STATE_RECov_Cnfg` = 0x19, `CY_U3P_UIB_LNK_STATE_RECov_Idle` = 0x1A }

*Link state machine states.*

### 5.37.1 Detailed Description

This file defines constants that are derived from the USB specifications, for the use of the USB driver and user firmware.

### 5.37.2 Typedef Documentation

#### 5.37.2.1 `typedef enum CyU3PUsb2TestModes CyU3PUsb2TestModes`

List of USB 2.0 test modes.

##### Description

This enumeration lists the various USB 2.0 test modes defined in the specification.

#### 5.37.2.2 `typedef enum CyU3PUsb3PacketType CyU3PUsb3PacketType`

USB 3.0 packet type codes.

##### Description

The following are the various USB 3.0 packet types.

#### 5.37.2.3 `typedef enum CyU3PUsb3TpSubType CyU3PUsb3TpSubType`

USB 3.0 transaction packet sub type codes.

##### Description

The following are the various packet sub-types transmitted during SuperSpeed operation.

#### 5.37.2.4 `typedef enum CyU3PUsbDescType CyU3PUsbDescType`

Enumeration of descriptor types.

##### Description

A USB device is identified by the descriptors provided. The following are among the standard descriptors defined by the USB specification.

#### 5.37.2.5 `typedef enum CyU3PUsbDevCapType CyU3PUsbDevCapType`

Device capability type codes.

##### Description

The following are the various extended capabilities of the USB device.

#### 5.37.2.6 `typedef enum CyU3PUsbEpType_t CyU3PUsbEpType_t`

Enumeration of the endpoint types.

##### Description

There are four types of endpoints. This defines the behaviour of the endpoint. The control endpoint is a compulsory for any device whereas the other endpoints are used as per device requirement.

#### 5.37.2.7 `typedef enum CyU3PUsbFeatureSelector CyU3PUsbFeatureSelector`

List of USB Feature selector codes.

**Description**

The following are the various features that can be selected using the SetFeature request or cleared using Clear← Feature setup request.

**5.37.2.8 typedef enum CyU3PUsbLinkState\_t CyU3PUsbLinkState\_t**

Link state machine states.

**Description**

The following are the USB 3.0 link states of interest to FX3 firmware.

**5.37.2.9 typedef enum CyU3PUsbSetupCmds CyU3PUsbSetupCmds**

Standard device request codes.

**Description**

These are the various standard requests received from the USB host. The device is expected to respond to all these requests.

**5.37.3 Enumeration Type Documentation****5.37.3.1 enum CyU3PUsb2TestModes**

List of USB 2.0 test modes.

**Description**

This enumeration lists the various USB 2.0 test modes defined in the specification.

Enumerator

- CY\_U3P\_USB2\_TEST\_NONE** Reserved.
- CY\_U3P\_USB2\_TEST\_J** Test\_J mode.
- CY\_U3P\_USB2\_TEST\_K** Test\_K mode.
- CY\_U3P\_USB2\_TEST\_SE0\_NAK** Test\_SE0\_NAK mode.
- CY\_U3P\_USB2\_TEST\_PACKET** Test\_Packet mode.
- CY\_U3P\_USB2\_TEST\_FORCE\_EN** Test\_Force\_Enable mode.

**5.37.3.2 enum CyU3PUsb3PacketType**

USB 3.0 packet type codes.

**Description**

The following are the various USB 3.0 packet types.

Enumerator

- CY\_U3P\_USB3\_PACK\_TYPE\_LMP** Link Management Packet.
- CY\_U3P\_USB3\_PACK\_TYPE\_TP** Transaction Packet.
- CY\_U3P\_USB3\_PACK\_TYPE\_DPH** Data Packet Header.
- CY\_U3P\_USB3\_PACK\_TYPE\_ITP** Isochronous Timestamp Packet.

### 5.37.3.3 enum CyU3PUsb3TpSubType

USB 3.0 transaction packet sub type codes.

#### Description

The following are the various packet sub-types transmitted during SuperSpeed operation.

#### Enumerator

**CY\_U3P\_USB3\_TP\_SUBTYPE\_RES** Reserved.  
**CY\_U3P\_USB3\_TP\_SUBTYPE\_ACK** ACK TP.  
**CY\_U3P\_USB3\_TP\_SUBTYPE\_NRDY** NRDY TP.  
**CY\_U3P\_USB3\_TP\_SUBTYPE\_ERDY** ERDY TP.  
**CY\_U3P\_USB3\_TP\_SUBTYPE\_STATUS** STATUS TP.  
**CY\_U3P\_USB3\_TP\_SUBTYPE\_STALL** STALL TP.  
**CY\_U3P\_USB3\_TP\_SUBTYPE\_NOTICE** DEV\_NOTIFICATION TP.  
**CY\_U3P\_USB3\_TP\_SUBTYPE\_PING** PING TP.  
**CY\_U3P\_USB3\_TP\_SUBTYPE\_PINGRSP** PING RESPONSE TP.

### 5.37.3.4 enum CyU3PUsbDescType

Enumeration of descriptor types.

#### Description

A USB device is identified by the descriptors provided. The following are among the standard descriptors defined by the USB specification.

#### Enumerator

**CY\_U3P\_USB\_DEVICE\_DESCR** Super Speed Device descr  
**CY\_U3P\_USB\_CONFIG\_DESCR** Configuration  
**CY\_U3P\_USB\_STRING\_DESCR** String  
**CY\_U3P\_USB\_INTRFC\_DESCR** Interface  
**CY\_U3P\_USB\_ENDPNT\_DESCR** End Point  
**CY\_U3P\_USB\_DEVQUAL\_DESCR** Device Qualifier  
**CY\_U3P\_USB\_OTHERSPEED\_DESCR** Other Speed Configuration  
**CY\_U3P\_USB\_INTRFC\_POWER\_DESCR** Interface power descriptor  
**CY\_U3P\_BOS\_DESCR** BOS descriptor  
**CY\_U3P\_DEVICE\_CAPB\_DESCR** Device Capability descriptor  
**CY\_U3P\_USB\_HID\_DESCR** HID descriptor  
**CY\_U3P\_USB\_REPORT\_DESCR** Report descriptor  
**CY\_U3P\_SS\_EP\_COMPN\_DESCR** End Point companion descriptor  
**CY\_U3P\_USB\_DEVICE\_DESCR** Device descriptor  
**CY\_U3P\_USB\_CONFIG\_DESCR** Configuration descriptor  
**CY\_U3P\_USB\_STRING\_DESCR** String descriptor  
**CY\_U3P\_USB\_INTRFC\_DESCR** Interface descriptor  
**CY\_U3P\_USB\_ENDPNT\_DESCR** Endpoint descriptor  
**CY\_U3P\_USB\_DEVQUAL\_DESCR** Device Qualifier descriptor  
**CY\_U3P\_USB\_OTHERSPEED\_DESCR** Other Speed Configuration descriptor  
**CY\_U3P\_USB\_INTRFC\_POWER\_DESCR** Interface power descriptor

**CY\_U3P\_USB\_OTG\_DESCR** OTG descriptor  
**CY\_U3P\_BOS\_DESCR** BOS descriptor  
**CY\_U3P\_DEVICE\_CAPB\_DESCR** Device Capability descriptor  
**CY\_U3P\_USB\_HID\_DESCR** HID descriptor  
**CY\_U3P\_USB\_REPORT\_DESCR** Report descriptor  
**CY\_U3P\_SS\_EP\_COMPN\_DESCR** Endpoint companion descriptor

### 5.37.3.5 enum CyU3PUusbDevCapType

Device capability type codes.

#### Description

The following are the various extended capabilities of the USB device.

Enumerator

**CY\_U3P\_WIRELESS\_USB\_CAPB\_TYPE** Wireless USB specific device level capabilities.  
**CY\_U3P\_USB2\_EXTN\_CAPB\_TYPE** USB 2.0 extension descriptor.  
**CY\_U3P\_SS\_USB\_CAPB\_TYPE** Super speed USB specific device level capabilities.  
**CY\_U3P\_CONTAINER\_ID\_CAPB\_TYPE** Unique ID used to identify the instance across all operating modes.

### 5.37.3.6 enum CyU3PUusbEpType\_t

Enumeration of the endpoint types.

#### Description

There are four types of endpoints. This defines the behaviour of the endpoint. The control endpoint is a compulsory for any device whereas the other endpoints are used as per device requirement.

Enumerator

**CY\_U3P\_USB\_EP\_CONTROL** Control Endpoint Type  
**CY\_U3P\_USB\_EP\_ISO** Isochronous Endpoint Type  
**CY\_U3P\_USB\_EP\_BULK** Bulk Endpoint Type  
**CY\_U3P\_USB\_EP\_INTR** Interrupt Endpoint Type

### 5.37.3.7 enum CyU3PUusbFeatureSelector

List of USB Feature selector codes.

#### Description

The following are the various features that can be selected using the SetFeature request or cleared using Clear← Feature setup request.

Enumerator

**CY\_U3P\_USBX\_FS\_EP\_HALT** USB Endpoint HALT feature. Sets or clears EP stall.  
**CY\_U3P\_USB2\_FS\_REMOTE\_WAKE** USB 2.0 Remote Wakeup.  
**CY\_U3P\_USB2\_FS\_TEST\_MODE** USB 2.0 Test mode.  
**CY\_U3P\_USB2\_OTG\_B\_HNP\_ENABLE** USB 2.0 OTG HNP enable signal to B-device.  
**CY\_U3P\_USB2\_OTG\_A\_HNP\_SUPPORT** USB 2.0 OTG HNP supported indication to B-device.  
**CY\_U3P\_USB3\_FS\_U1\_ENABLE** USB 3.0 U1 Enable.  
**CY\_U3P\_USB3\_FS\_U2\_ENABLE** USB 3.0 U2 Enable.  
**CY\_U3P\_USB3\_FS\_LTM\_ENABLE** USB 3.0 LTM Enable.

5.37.3.8 enum CyU3PUsbLinkState\_t

## 5.38 firmware/u3p\_firmware/inc/cyu3usbhost.h File Reference

The FX3 device supports programmable USB host implementation for a single USB host port at USB-HS, USB-FS and USB-LS speeds. The control pipe as well as the data pipes to be used can be configured through a set of USB host mode APIs. The USB host mode APIs also provide the capability to manage the host port.

```
#include <cyu3types.h>
#include <cyu3usbconst.h>
#include <cyu3externcstart.h>
#include <cyu3externcend.h>
```

### Data Structures

- struct [CyU3PUsbHostEpConfig\\_t](#)  
*Host mode endpoint configuration structure.*
- struct [CyU3PUsbHostConfig\\_t](#)  
*USB host mode configuration information.*

### Macros

- #define [CY\\_U3P\\_USB\\_HOST\\_EPS\\_EPNUM\\_POS](#) (0)  
*Position of endpoint number field in the status word.*
- #define [CY\\_U3P\\_USB\\_HOST\\_EPS\\_EPNUM\\_MASK](#) (0x0000000F)  
*Mask for the endpoint number field in the status word.*
- #define [CY\\_U3P\\_USB\\_HOST\\_EPS\\_EPDIR](#) (0x00000010)  
*Endpoint direction bit (1 - OUT, 0 - IN).*
- #define [CY\\_U3P\\_USB\\_HOST\\_EPS\\_ACTIVE](#) (0x00000020)  
*Endpoint active bit. Indicates whether the EP is still active.*
- #define [CY\\_U3P\\_USB\\_HOST\\_EPS\\_HALT](#) (0x00000040)  
*Endpoint halt bit. Set if the endpoint is halted (stalled).*
- #define [CY\\_U3P\\_USB\\_HOST\\_EPS\\_OVER\\_UNDER\\_RUN](#) (0x00000080)  
*This bit is set if an overrun or underrun has happened on this endpoint.*
- #define [CY\\_U3P\\_USB\\_HOST\\_EPS\\_BABBLE](#) (0x00000100)  
*This bit is set if a babble was detected during the transfer.*
- #define [CY\\_U3P\\_USB\\_HOST\\_EPS\\_XACT\\_ERROR](#) (0x00000200)  
*This bit is set if there was a transfer error.*
- #define [CY\\_U3P\\_USB\\_HOST\\_EPS\\_PING](#) (0x00000400)  
*This bit is set if there was a PING packet.*
- #define [CY\\_U3P\\_USB\\_HOST\\_EPS\\_PHY\\_ERROR](#) (0x00000800)  
*This bit is set if there was a PHY error during the transfer.*
- #define [CY\\_U3P\\_USB\\_HOST\\_EPS\\_PID\\_ERROR](#) (0x00001000)  
*This bit is set if there was a PID error during the transfer.*
- #define [CY\\_U3P\\_USB\\_HOST\\_EPS\\_TIMEOUT\\_ERROR](#) (0x00002000)  
*This bit is set if there was a timeout error during the transfer.*
- #define [CY\\_U3P\\_USB\\_HOST\\_EPS\\_ISO\\_ORUN\\_ERROR](#) (0x00004000)  
*This bit is set if there was an ISO overrun error during the transfer.*
- #define [CY\\_U3P\\_USB\\_HOST\\_EPS\\_IOC\\_INT](#) (0x00008000)  
*This bit is set if there was an IOC interrupt.*
- #define [CY\\_U3P\\_USB\\_HOST\\_EPS\\_BYTE\\_COUNT\\_POS](#) (16)  
*Position of the remaining byte count field.*
- #define [CY\\_U3P\\_USB\\_HOST\\_EPS\\_BYTE\\_COUNT\\_MASK](#) (0xFFFF0000)

- `#define CY_U3P_USB_HOST_PORT_STAT_CONNECTED (0x0001)`

*This bit is set if a peripheral is connected downstream.*
- `#define CY_U3P_USB_HOST_PORT_STAT_ENABLED (0x0002)`

*This bit is set if the host port has been enabled by the user.*
- `#define CY_U3P_USB_HOST_PORT_STAT_ACTIVE (0x0003)`

*Mask to identify whether the port is active. A port is active if it is enabled and connected to a downstream peripheral.*
- `#define CY_U3P_USB_HOST_PORT_STAT_SUSPENDED (0x0004)`

*This bit is set if the port has been suspended.*

## Typedefs

- `typedef enum CyU3PUsbHostEpXferType_t CyU3PUsbHostEpXferType_t`

*Mode of data transfer.*
- `typedef enum CyU3PUsbHostOpSpeed_t CyU3PUsbHostOpSpeed_t`

*Speed of operation for the USB host port.*
- `typedef enum CyU3PUsbHostEventType_t CyU3PUsbHostEventType_t`

*List of USB host mode events.*
- `typedef uint32_t CyU3PUsbHostEpStatus_t`

*Host mode endpoint status.*
- `typedef uint16_t CyU3PUsbHostPortStatus_t`

*Host mode port status information.*
- `typedef struct CyU3PUsbHostEpConfig_t CyU3PUsbHostEpConfig_t`

*Host mode endpoint configuration structure.*
- `typedef void(* CyU3PUsbHostEventCb_t) (CyU3PUsbHostEventType_t evType, uint32_t evData)`

*Host mode event callback function.*
- `typedef void(* CyU3PUsbHostXferCb_t) (uint8_t ep, CyU3PUsbHostEpStatus_t epStatus)`

*Host mode endpoint transfer complete callback function.*
- `typedef struct CyU3PUsbHostConfig_t CyU3PUsbHostConfig_t`

*USB host mode configuration information.*

## Enumerations

- `enum CyU3PUsbHostEpXferType_t { CY_U3P_USB_HOST_EPXFER_NORMAL = 0, CY_U3P_USB_HOST_EPXFER_SETUP_OUT_DATA, CY_U3P_USB_HOST_EPXFER_SETUP_IN_DATA, CY_U3P_USB_HOST_EPXFER_SETUP_NO_DATA }`

*Mode of data transfer.*
- `enum CyU3PUsbHostOpSpeed_t { CY_U3P_USB_HOST_LOW_SPEED = 0, CY_U3P_USB_HOST_FULL_SPEED, CY_U3P_USB_HOST_HIGH_SPEED }`

*Speed of operation for the USB host port.*
- `enum CyU3PUsbHostEventType_t { CY_U3P_USB_HOST_EVENT_CONNECT = 0, CY_U3P_USB_HOST_EVENT_DISCONNECT }`

*List of USB host mode events.*

## Functions

- `CyBool_t CyU3PUsbHostIsStarted (void)`

*Check whether the USB host stack has been started.*
- `Cy3PReturnStatus_t CyU3PUsbHostStart (CyU3PUsbHostConfig_t *hostCfg)`

*This function initializes the USB 2.0 host stack.*

- `CyU3PReturnStatus_t CyU3PUsbHostStop (void)`  
*This function de-initializes the USB host stack.*
- `CyU3PReturnStatus_t CyU3PUsbHostPortEnable (void)`  
*This function enables the USB host port on the FX3 device.*
- `CyU3PReturnStatus_t CyU3PUsbHostPortDisable (void)`  
*Disable the USB host port.*
- `CyU3PReturnStatus_t CyU3PUsbHostGetPortStatus (CyU3PUsbHostPortStatus_t *portStatus, CyU3PUsbHostOpSpeed_t *portSpeed)`  
*This function retrieves the current port status.*
- `CyU3PReturnStatus_t CyU3PUsbHostPortReset (void)`  
*This function resets the USB host port.*
- `CyU3PReturnStatus_t CyU3PUsbHostPortSuspend (void)`  
*Suspend the USB host port.*
- `CyU3PReturnStatus_t CyU3PUsbHostPortResume (void)`  
*Resume the previously suspended USB host port.*
- `CyU3PReturnStatus_t CyU3PUsbHostGetFrameNumber (uint32_t *frameNumber)`  
*Get the current frame number to be used.*
- `CyU3PReturnStatus_t CyU3PUsbHostGetDeviceAddress (uint8_t *devAddr)`  
*Get the current downstream peripheral address.*
- `CyU3PReturnStatus_t CyU3PUsbHostSetDeviceAddress (uint8_t devAddr)`  
*Set (update) the downstream peripheral address.*
- `CyU3PReturnStatus_t CyU3PUsbHostEpAdd (uint8_t ep, CyU3PUsbHostEpConfig_t *cfg)`  
*Add an endpoint to the scheduler.*
- `CyU3PReturnStatus_t CyU3PUsbHostEpRemove (uint8_t ep)`  
*Removes an endpoint from the scheduler.*
- `CyU3PReturnStatus_t CyU3PUsbHostEpReset (uint8_t ep)`  
*Reset an endpoint.*
- `CyU3PReturnStatus_t CyU3PUsbHostSendSetupRqt (uint8_t *setupPkt, uint8_t *buf_p)`  
*Perform a USB control (EP0) transfer.*
- `CyU3PReturnStatus_t CyU3PUsbHostEpSetXfer (uint8_t ep, CyU3PUsbHostEpXferType_t type, uint32_t count)`  
*Start a non-EP0 data transfer.*
- `CyU3PReturnStatus_t CyU3PUsbHostEp0BeginXfer (void)`  
*Kick off a low level control endpoint transfer.*
- `CyU3PReturnStatus_t CyU3PUsbHostEpAbort (uint8_t ep)`  
*Abort pending transfers on the selected endpoint.*
- `CyU3PReturnStatus_t CyU3PUsbHostEpWaitForCompletion (uint8_t ep, CyU3PUsbHostEpStatus_t *ep_Status, uint32_t waitOption)`  
*Wait for the current endpoint transfer to complete.*

### 5.38.1 Detailed Description

The FX3 device supports programmable USB host implementation for a single USB host port at USB-HS, USB-FS and USB-LS speeds. The control pipe as well as the data pipes to be used can be configured through a set of USB host mode APIs. The USB host mode APIs also provide the capability to manage the host port.

## 5.38.2 Typedef Documentation

### 5.38.2.1 `typedef struct CyU3PUsbHostConfig_t CyU3PUsbHostConfig_t`

USB host mode configuration information.

#### Description

The FX3 host mode driver takes the following configuration parameters, which are set when starting the stack. These settings cannot be changed dynamically while the stack is active.

#### See also

[CyU3PUsbHostStart](#)

### 5.38.2.2 `typedef struct CyU3PUsbHostEpConfig_t CyU3PUsbHostEpConfig_t`

Host mode endpoint configuration structure.

#### Description

The structure holds the information for configuring an endpoint when the FX3 is acting as a USB host.

#### See also

[CyU3PUsbHostEpAdd](#)

### 5.38.2.3 `typedef uint32_t CyU3PUsbHostEpStatus_t`

Host mode endpoint status.

#### Description

At the end of each data transfer, the transfer scheduler on the FX3 device returns a 32-bit status value. This status value has a number of fields that report the status of the data transfer.

#### See also

[CyU3PUsbHostXferCb\\_t](#)

### 5.38.2.4 `typedef enum CyU3PUsbHostEpXferType_t CyU3PUsbHostEpXferType_t`

Mode of data transfer.

#### Description

This type lists the various data transfer modes for USB endpoints on the FX3 host. The NORMAL mode is to be used for all endpoints other than the control endpoint. For the control (EP0) endpoint, the mode needs to be set on a per-transfer basis depending on the type of the control request.

#### See also

[CyU3PUsbHostEpSetXfer](#)

### 5.38.2.5 `typedef void(* CyU3PUsbHostEventCb_t)(CyU3PUsbHostEventType_t evType, uint32_t evData )`

Host mode event callback function.

#### Description

The event callback function is used to notify the user application about peripheral connect and disconnect events.

See also

[CyU3PUsbHostStart](#)

### 5.38.2.6 `typedef enum CyU3PUsbHostEventType_t CyU3PUsbHostEventType_t`

List of USB host mode events.

#### Description

This type lists the possible USB host mode related events that are reported to the application via the event callback function.

See also

[CyU3PUsbHostEventCb\\_t](#)

### 5.38.2.7 `typedef enum CyU3PUsbHostOpSpeed_t CyU3PUsbHostOpSpeed_t`

Speed of operation for the USB host port.

#### Description

The USB host on the FX3 device supports Low, Full and Hi-Speed operation. This type defines named constants for each of these connection types.

See also

[CyU3PUsbHostGetPortStatus](#)

### 5.38.2.8 `typedef uint16_t CyU3PUsbHostPortStatus_t`

Host mode port status information.

#### Description

The port status returned by the library is a 16-bit value with multiple fields.

See also

[CyU3PUsbHostGetPortStatus](#)

### 5.38.2.9 `typedef void(* CyU3PUsbHostXferCb_t)(uint8_t ep, CyU3PUsbHostEpStatus_t epStatus )`

Host mode endpoint transfer complete callback function.

#### Description

The transfer callback function is registered when the USB host stack is started, and is used by the driver to notify the application about transfer completion.

See also

[CyU3PUsbHostStart](#)  
[CyU3PUsbHostEpSetXfer](#)

## 5.38.3 Enumeration Type Documentation

### 5.38.3.1 `enum CyU3PUsbHostEpXferType_t`

Mode of data transfer.

**Description**

This type lists the various data transfer modes for USB endpoints on the FX3 host. The NORMAL mode is to be used for all endpoints other than the control endpoint. For the control (EP0) endpoint, the mode needs to be set on a per-transfer basis depending on the type of the control request.

**See also**

[CyU3PUsbHostEpSetXfer](#)

**Enumerator**

**CY\_U3P\_USB\_HOST\_EPXFER\_NORMAL** Normal transfer. All non-EP0 transfers.

**CY\_U3P\_USB\_HOST\_EPXFER\_SETUP\_OUT\_DATA** EP0 setup packet with OUT data phase.

**CY\_U3P\_USB\_HOST\_EPXFER\_SETUP\_IN\_DATA** EP0 setup packet with IN data phase.

**CY\_U3P\_USB\_HOST\_EPXFER\_SETUP\_NO\_DATA** EP0 setup packet with no data phase.

### 5.38.3.2 enum CyU3PUsbHostEventType\_t

List of USB host mode events.

**Description**

This type lists the possible USB host mode related events that are reported to the application via the event callback function.

**See also**

[CyU3PUsbHostEventCb\\_t](#)

**Enumerator**

**CY\_U3P\_USB\_HOST\_EVENT\_CONNECT** USB Connect event.

**CY\_U3P\_USB\_HOST\_EVENT\_DISCONNECT** USB Disconnect event.

### 5.38.3.3 enum CyU3PUsbHostOpSpeed\_t

Speed of operation for the USB host port.

**Description**

The USB host on the FX3 device supports Low, Full and Hi-Speed operation. This type defines named constants for each of these connection types.

**See also**

[CyU3PUsbHostGetPortStatus](#)

**Enumerator**

**CY\_U3P\_USB\_HOST\_LOW\_SPEED** Host port is operating in low speed mode.

**CY\_U3P\_USB\_HOST\_FULL\_SPEED** Host port is operating in full speed mode.

**CY\_U3P\_USB\_HOST\_HIGH\_SPEED** Host port is operating in high speed mode.

## 5.38.4 Function Documentation

### 5.38.4.1 CyU3PReturnStatus\_t CyU3PUsbHostEp0BeginXfer( void )

Kick off a low level control endpoint transfer.

**Description**

This function enables the endpoint and starts the transfer for EP0. The setup packet must be queued on EP0 egress socket before this function is called.

**Return value**

CY\_U3P\_SUCCESS - The call was successful.

CY\_U3P\_ERROR\_NOT\_STARTED - The port is not enabled.

CY\_U3P\_ERROR\_INVALID\_SEQUENCE - The endpoint is not configured or the transfer is not setup.

**See also**

[CyU3PUsbHostEpSetXfer](#)  
[CyU3PUsbHostEpAbort](#)

**5.38.4.2 CyU3PReturnStatus\_t CyU3PUsbHostEpAbort ( uint8\_t ep )**

Abort pending transfers on the selected endpoint.

**Description**

This function aborts any ongoing data transfer on the selected endpoint and deactivates it. The DMA channels are not touched by the API, and they need to be separately reset.

**Return value**

CY\_U3P\_SUCCESS - The call was successful.

CY\_U3P\_ERROR\_NOT\_STARTED - The port is not enabled.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - The endpoint number is invalid.

CY\_U3P\_ERROR\_NOT\_CONFIGURED - The endpoint is not added.

**See also**

[CyU3PUsbHostEpSetXfer](#)  
[CyU3PUsbHostEp0BeginXfer](#)

**Parameters**

<i>ep</i>	Endpoint to abort.
-----------	--------------------

**5.38.4.3 CyU3PReturnStatus\_t CyU3PUsbHostEpAdd ( uint8\_t ep, CyU3PUsbHostEpConfig\_t \* cfg )**

Add an endpoint to the scheduler.

**Description**

The USB host block on the FX3 maintains a schedule based on which data transfers on various endpoints are initiated. The firmware application should identify the active set of endpoints on the downstream peripheral, and add the corresponding endpoints to the execution schedule.

The schedule parameters that are passed to this function depend on the values reported by the peripheral in the endpoint descriptors.

**Return value**

CY\_U3P\_SUCCESS - The call was successful.

CY\_U3P\_ERROR\_NULL\_POINTER - The input pointer was NULL.

CY\_U3P\_ERROR\_NOT\_STARTED - The port is not enabled.

CY\_U3P\_ERROR\_ALREADY\_STARTED - The endpoint is already added.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - One or more of the input parameter fields is invalid.

See also

[CyU3PUsbHostEpRemove](#)

Parameters

<i>ep</i>	Endpoint to be added to the transfer schedule.
<i>cfg</i>	Endpoint configuration parameters.

#### 5.38.4.4 CyU3PReturnStatus\_t CyU3PUsbHostEpRemove ( uint8\_t ep )

Removes an endpoint from the scheduler.

**Description**

This function can be called to remove an endpoint from the transfer schedule, once it is no longer in use by the application.

**Return value**

CY\_U3P\_SUCCESS - The call was successful.  
CY\_U3P\_ERROR\_BAD\_ARGUMENT - The endpoint number is invalid.  
CY\_U3P\_ERROR\_NOT\_STARTED - The port is not enabled.  
CY\_U3P\_ERROR\_NOT\_CONFIGURED - The endpoint is not yet added.

See also

[CyU3PUsbHostEpAdd](#)

Parameters

<i>ep</i>	Endpoint to be removed from scheduler.
-----------	--

#### 5.38.4.5 CyU3PReturnStatus\_t CyU3PUsbHostEpReset ( uint8\_t ep )

Reset an endpoint.

**Description**

This function flushes the internal buffers and resets the data toggles for an endpoint. This should only be called when there is no active transfer on the endpoint.

**Return value**

CY\_U3P\_SUCCESS - The call was successful.  
CY\_U3P\_ERROR\_NOT\_STARTED - The port is not enabled.  
CY\_U3P\_ERROR\_BAD\_ARGUMENT - The endpoint number is invalid.  
CY\_U3P\_ERROR\_NOT\_CONFIGURED - The endpoint is not added.  
CY\_U3P\_ERROR\_INVALID\_SEQUENCE - The endpoint is active.

See also

[CyU3PUsbHostEpAdd](#)

Parameters

<i>ep</i>	Endpoint to be reset.
-----------	-----------------------

5.38.4.6 `CyU3PReturnStatus_t CyU3PUsbHostEpSetXfer ( uint8_t ep, CyU3PUsbHostEpXferType_t type, uint32_t count )`

Start a non-EP0 data transfer.

#### Description

This function will enable the endpoint and setup the data transfer for all EPs except for EP0. This can also be used for EP0 when the low level control flag has been set to true.

In the case of a low level EP0 transfer, the 8 byte setup packet needs to be queued on the EP0 egress DMA channel. If there is a data phase, that also needs to be queued on the EP0 ingress or egress DMA channel. Once the DMA channels have been setup, the EP0 transfer can be kicked off by calling the CyU3PUsbHostEp0BeginXfer API.

#### Return value

`CY_U3P_SUCCESS` - The call was successful.  
`CY_U3P_ERROR_NOT_STARTED` - The port is not enabled.  
`CY_U3P_ERROR_BAD_ARGUMENT` - One or more input parameters is invalid.  
`CY_U3P_ERROR_NOT_CONFIGURED` - The endpoint is not added.  
`CY_U3P_ERROR_INVALID_SEQUENCE` - The endpoint is already active.

#### See also

[CyU3PUsbHostEp0BeginXfer](#)  
[CyU3PUsbHostEpAbort](#)

#### Parameters

<code>ep</code>	Endpoint to configure.
<code>type</code>	Type of transfer. This is meaningful only for EP0.
<code>count</code>	Size of data to be transferred.

5.38.4.7 `CyU3PReturnStatus_t CyU3PUsbHostEpWaitForCompletion ( uint8_t ep, CyU3PUsbHostEpStatus_t *epStatus, uint32_t waitOption )`

Wait for the current endpoint transfer to complete.

#### Description

The function waits for the transfer to complete or get aborted. If this does not happen within the timeout specified, it returns `CY_U3P_ERROR_TIMEOUT`.

A timeout error does not necessarily mean that it has failed, and it is possible that another call to this API returns successfully.

#### Return value

`CY_U3P_SUCCESS` - The call was successful.  
`CY_U3P_ERROR_NOT_STARTED` - The port is not enabled.  
`CY_U3P_ERROR_BAD_ARGUMENT` - The endpoint number is invalid.  
`CY_U3P_ERROR_NOT_CONFIGURED` - The endpoint is not added.  
`CY_U3P_ERROR_INVALID_SEQUENCE` - No active transfer.  
`CY_U3P_ERROR_TIMEOUT` - The transfer is not complete at the end of the specified timeout duration.  
`CY_U3P_ERROR_STALLED` - The transfer was stalled by the USB peripheral.

#### See also

[CyU3PUsbHostEpSetXfer](#)  
[CyU3PUsbHostEp0BeginXfer](#)

**Parameters**

<i>ep</i>	Endpoint to wait on.
<i>epStatus</i>	Output parameter to be filled in with transfer status.
<i>waitOption</i>	Timeout duration to wait for.

**5.38.4.8 CyU3PReturnStatus\_t CyU3PUsbHostGetDeviceAddress ( uint8\_t \* *devAddr* )**

Get the current downstream peripheral address.

**Description**

This function returns the device address that has been assigned by FX3 to the downstream peripheral. A return value of zero indicates that the peripheral has been disconnected.

**Return value**

CY\_U3P\_SUCCESS - The call was successful.

CY\_U3P\_ERROR\_NULL\_POINTER - The pointer provided is NULL.

CY\_U3P\_ERROR\_NOT\_STARTED - The host port is not enabled.

**See also**

[CyU3PUsbHostSetDeviceAddress](#)

**Parameters**

<i>devAddr</i>	Pointer to load the current device address.
----------------	---

**5.38.4.9 CyU3PReturnStatus\_t CyU3PUsbHostGetFrameNumber ( uint32\_t \* *frameNumber* )**

Get the current frame number to be used.

**Description**

This function gets the current USB frame number. The frame number query is not synchronized with the scheduler and only returns the number at the time of the API call.

**Return value**

CY\_U3P\_SUCCESS - The call was successful.

CY\_U3P\_ERROR\_NULL\_POINTER - The input pointer was NULL.

CY\_U3P\_ERROR\_NOT\_STARTED - The host port is not enabled.

**See also**

[CyU3PUsbHostStart](#)  
[CyU3PUsbHostPortEnable](#)

**Parameters**

<i>frameNumber</i>	Pointer to load the frame number.
--------------------	-----------------------------------

**5.38.4.10 CyU3PReturnStatus\_t CyU3PUsbHostGetPortStatus ( CyU3PUsbHostPortStatus\_t \* *portStatus*, CyU3PUsbHostOpSpeed\_t \* *portSpeed* )**

This function retrieves the current port status.

**Description**

This function retrieves the current state and speed of operation of the USB host port on the FX3 device.

**Return value**

CY\_U3P\_SUCCESS - The call was successful.

CY\_U3P\_ERROR\_NOT\_STARTED - The host stack is not running.

**See also**

[CyU3PUsbHostPortEnable](#)  
[CyU3PUsbHostPortDisable](#)

**5.38.4.11 CyBool\_t CyU3PUsbHostIsStarted( void )**

Check whether the USB host stack has been started.

**Description**

This function is used to check whether the USB host stack has been started.

**Return value**

CyTrue - USB host module started.

CyFalse - USB host module not started.

**See also**

[CyU3PUsbHostStart](#)  
[CyU3PUsbHostStop](#)

**5.38.4.12 CyU3PReturnStatus\_t CyU3PUsbHostPortDisable( void )**

Disable the USB host port.

**Description**

This function disables the USB host port on the FX3 device. The device will still be able to detect a newly connected peripheral after the port is disabled.

**Return value**

CY\_U3P\_SUCCESS - The call was successful.

CY\_U3P\_ERROR\_NOT\_ENABLED - The port is not enabled.

**See also**

[CyU3PUsbHostPortEnable](#)  
[CyU3PUsbHostGetPortStatus](#)

**5.38.4.13 CyU3PReturnStatus\_t CyU3PUsbHostPortEnable( void )**

This function enables the USB host port on the FX3 device.

**Description**

The USB host port on the FX3 is kept disabled when the host stack is started. The driver uses the event callback to notify that a downstream peripheral has been connected, and then the application can call this API to enable the host port.

This function enables the port and sets it to the correct speed of operation. When the down-stream peripheral is disconnected, the port automatically gets disabled.

The FX3 device has only a single usb host port which can support a single peripheral device. Hubs are not supported.

**Return value**

CY\_U3P\_SUCCESS - The call was successful.  
CY\_U3P\_ERROR\_NOT\_STARTED - The host stack is not running.  
CY\_U3P\_ERROR\_ALREADY\_STARTED - The port is already enabled.  
CY\_U3P\_ERROR\_FAILURE - No downstream peripheral attached.

**See also**

[CyU3PUsbHostPortDisable](#)  
[CyU3PUsbHostGetPortStatus](#)

#### 5.38.4.14 CyU3PReturnStatus\_t CyU3PUsbHostPortReset( void )

This function resets the USB host port.

**Description**

This function resets the USB host port so that the peripheral connected can be re-enumerated. Calling this function is equivalent to a port disable call followed by a port enable call.

**Return value**

CY\_U3P\_SUCCESS - The call was successful.  
CY\_U3P\_ERROR\_NOT\_STARTED - The host stack is not running.

**See also**

[CyU3PUsbHostPortEnable](#)  
[CyU3PUsbHostPortDisable](#)  
[CyU3PUsbHostGetPortStatus](#)

#### 5.38.4.15 CyU3PReturnStatus\_t CyU3PUsbHostPortResume( void )

Resume the previously suspended USB host port.

**Description**

This function resumes the USB host port, after it was previously suspended through the CyU3PUsbHostPortSuspend API.

**Return value**

CY\_U3P\_SUCCESS - The call was successful.  
CY\_U3P\_ERROR\_NOT\_STARTED - The port is not enabled or device got disconnected.  
CY\_U3P\_ERROR\_INVALID\_SEQUENCE - The port is not suspended.

**See also**

[CyU3PUsbHostPortSuspend](#)  
[CyU3PUsbHostGetPortStatus](#)

#### 5.38.4.16 CyU3PReturnStatus\_t CyU3PUsbHostPortSuspend( void )

Suspend the USB host port.

**Description**

This function suspends the USB host port. Please note that the FX3 device does not support remote wakeup and the port resumption has to be done by the firmware application.

**Return value**

CY\_U3P\_SUCCESS - The call was successful.  
CY\_U3P\_ERROR\_NOT\_STARTED - The port is not active.  
CY\_U3P\_ERROR\_INVALID\_SEQUENCE - There is an active data transfer.

**See also**

[CyU3PUsbHostPortResume](#)  
[CyU3PUsbHostGetPortStatus](#)

**5.38.4.17 CyU3PReturnStatus\_t CyU3PUsbHostSendSetupRqt ( uint8\_t \* setupPkt, uint8\_t \* buf\_p )**

Perform a USB control (EP0) transfer.

**Description**

This function performs all of the operations associated with a USB control (EP0) transfer. This is only valid when the ep0LowLevelControl setting is false.

The API initiates the transfer of setup packet, but does not wait for completion. The setup, data and status phases of the transfer will be handled by the driver; and the transfer complete callback shall be called when all of these are done.

**Return value**

CY\_U3P\_SUCCESS - The call was successful.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - EP0 Low level control enabled.  
 CY\_U3P\_ERROR\_NOT\_STARTED - The port is not enabled.  
 CY\_U3P\_ERROR\_INVALID\_SEQUENCE - The endpoint is already active.  
 CY\_U3P\_ERROR\_NOT\_CONFIGURED - The endpoint is not added.  
 CY\_U3P\_ERROR\_DMA\_FAILURE - Error in setting internal DMA channels.

**See also**

[CyU3PUsbHostStart](#)  
[CyU3PUsbHostEpAbort](#)

**Parameters**

<i>setupPkt</i>	Pointer to buffer containing the 8 byte setup packet.
<i>buf_p</i>	Buffer for the data phase. The buffer should be big enough to handle the complete data phase, and should already contain the data in the case of an OUT data transfer..

**5.38.4.18 CyU3PReturnStatus\_t CyU3PUsbHostSetDeviceAddress ( uint8\_t devAddr )**

Set (update) the downstream peripheral address.

**Description**

This function sets (updates) the device address that should be used by the FX3 to talk to the downstream peripheral. This address is initialized to zero whenever the port is enabled. The application should set the address after completing a SET\_ADDRESS command successfully.

**Return value**

CY\_U3P\_SUCCESS - The call was successful.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - The address parameter is invalid.  
 CY\_U3P\_ERROR\_NOT\_STARTED - The host port is not enabled.

**See also**

[CyU3PUsbHostGetDeviceAddress](#)

**Parameters**

<i>devAddr</i>	Device address to be set.
----------------	---------------------------

### 5.38.4.19 CyU3PReturnStatus\_t CyU3PUsbHostStart ( CyU3PUsbHostConfig\_t \* hostCfg )

This function initializes the USB 2.0 host stack.

#### Description

This function enables the USB block and configures it to function as a host. The configuration parameters cannot be changed unless the stack is stopped and re-started.

#### Return value

CY\_U3P\_SUCCESS - The call was successful.

CY\_U3P\_ERROR\_NOT\_SUPPORTED - if the current FX3 device does not support the USB 2.0 host.

CY\_U3P\_ERROR\_NULL\_POINTER - If NULL pointer is passed in as parameter.

CY\_U3P\_ERROR\_ALREADY\_STARTED - The host stack is already running.

CY\_U3P\_ERROR\_INVALID\_SEQUENCE - FX3 is in the wrong OTG mode or USB device stack is running.

#### See also

[CyU3PUsbHostConfig\\_t](#)

[CyU3PUsbHostStop](#)

[CyU3PUsbHostIsStarted](#)

#### Parameters

<i>hostCfg</i>	Pointer to the host configuration information.
----------------	--

### 5.38.4.20 CyU3PReturnStatus\_t CyU3PUsbHostStop ( void )

This function de-initializes the USB host stack.

#### Description

This function disables the USB host mode operation and de-initializes the USB host stack.

#### Return value

CY\_U3P\_SUCCESS - The call was successful.

CY\_U3P\_ERROR\_NOT\_STARTED - The host mode stack is not running.

#### See also

[CyU3PUsbHostStart](#)

[CyU3PUsbHostIsStarted](#)

## 5.39 firmware/u3p\_firmware/inc/cyu3usbottg.h File Reference

The FX3 device supports USB 2.0 On-The-Go (OTG) functionality which allows the device to identify whether it should function as a USB host or a peripheral. This file defines the data types and APIs provided by the USB driver on FX3 for OTG management.

```
#include <cyu3types.h>
#include <cyu3externcstart.h>
#include <cyu3externcend.h>
```

## Data Structures

- struct [CyU3POtgConfig\\_t](#)  
*OTG configuration information.*

## Macros

- `#define CY_U3P_OTG_SR_P_MAX_REPEAT_INTERVAL (10000)`  
*Maximum value in ms that can be used for the SRP repeat interval.*

## Typedefs

- `typedef enum CyU3POtgMode_t CyU3POtgMode_t`  
*OTG modes of operation.*
- `typedef enum CyU3POtgChargerDetectMode_t CyU3POtgChargerDetectMode_t`  
*Various charger detection methods supported.*
- `typedef enum CyU3POtgEvent_t CyU3POtgEvent_t`  
*List of OTG events.*
- `typedef enum CyU3POtgPeripheralType_t CyU3POtgPeripheralType_t`  
*List of OTG peripheral types.*
- `typedef void(* CyU3POtgEventCallback_t) (CyU3POtgEvent_t event, uint32_t input)`  
*OTG event callback function.*
- `typedef struct CyU3POtgConfig_t CyU3POtgConfig_t`  
*OTG configuration information.*

## Enumerations

- `enum CyU3POtgMode_t { CY_U3P_OTG_MODE_DEVICE_ONLY = 0, CY_U3P_OTG_MODE_HOST_ONLY, CY_U3P_OTG_MODE_OTG, CY_U3P_OTG_MODE_CARKIT_PPRT, CY_U3P_OTG_MODE_CARKIT_UART, CY_U3P_OTG_NUM_MODES }`  
*OTG modes of operation.*
- `enum CyU3POtgChargerDetectMode_t { CY_U3P_OTG_CHARGER_DETECT_ACA_MODE = 0, CY_U3P_OTG_CHARGER_DETECT_MOT_EMU, CY_U3P_OTG_CHARGER_DETECT_NUM_MODES }`  
*Various charger detection methods supported.*
- `enum CyU3POtgEvent_t { CY_U3P_OTG_PERIPHERAL_CHANGE = 0, CY_U3P_OTG_SR_P_DETECT, CY_U3P_OTG_VBUS_VALID_CHANGE }`  
*List of OTG events.*
- `enum CyU3POtgPeripheralType_t { CY_U3P_OTG_TYPE_DISABLED = 0, CY_U3P_OTG_TYPE_A_CABLE, CY_U3P_OTG_TYPE_B_CABLE, CY_U3P_OTG_TYPE_ACA_A_CHG, CY_U3P_OTG_TYPE_ACA_B_CHG, CY_U3P_OTG_TYPE_ACA_C_CHG, CY_U3P_OTG_TYPE_MOT_MPX200, CY_U3P_OTG_TYPE_MOT_CHG, CY_U3P_OTG_TYPE_MOT_MID, CY_U3P_OTG_TYPE_MOT_FAST }`  
*List of OTG peripheral types.*

## Functions

- `CyBool_t CyU3POtgIsStarted (void)`  
*Check whether the OTG module has been started.*
- `CyU3POtgPeripheralType_t CyU3POtgGetPeripheralType (void)`  
*Identify the type of USB peripheral attached to FX3.*
- `CyU3POtgMode_t CyU3POtgGetMode (void)`  
*Retrieve the currently selected OTG operating mode.*
- `CyBool_t CyU3POtgIsDeviceMode (void)`  
*Check whether USB device (peripheral) mode operation is permitted.*
- `CyBool_t CyU3POtgIsHostMode (void)`

- [CyU3PReturnStatus\\_t CyU3POtgStart \(CyU3POtgConfig\\_t \\*cfg\)](#)

*Check whether USB host mode operation is permitted.*
- [CyU3PReturnStatus\\_t CyU3POtgStop \(void\)](#)

*Initialize the OTG module.*
- [CyU3PReturnStatus\\_t CyU3POtgSrpStart \(uint32\\_t repeatInterval\)](#)

*Disable the OTG module.*
- [CyU3PReturnStatus\\_t CyU3POtgSrpAbort \(void\)](#)

*Initiate an SRP request.*
- [CyU3PReturnStatus\\_t CyU3POtgRequestHnp \(CyBool\\_t isEnabled\)](#)

*Abort SRP request.*
- [CyU3PReturnStatus\\_t CyU3POtgHnpEnable \(CyBool\\_t isEnabled\)](#)

*Set the HNP request bit in the device status word.*
- [CyU3PReturnStatus\\_t CyU3POtgHnpEnabled \(void\)](#)

*Initiate a HNP role change.*
- [CyBool\\_t CyU3POtgIsHnpEnabled \(void\)](#)

*Check if a role reversal mode is active or not.*
- [CyBool\\_t CyU3POtgIsVBusValid \(void\)](#)

*Check if a valid VBus is available.*

### 5.39.1 Detailed Description

The FX3 device supports USB 2.0 On-The-Go (OTG) functionality which allows the device to identify whether it should function as a USB host or a peripheral. This file defines the data types and APIs provided by the USB driver on FX3 for OTG management.

#### Description

When the FX3 USB port is in the device mode of operation, it can function as per USB 3.0 specification, and function at SuperSpeed, Hi-Speed or full speed depending upon the host capabilities.

When configured as a USB host, the port supports USB 2.0 operations at Hi-Speed, full speed or low speed according to the peripheral capabilities. In host mode of operation, only a single peripheral can be attached at a time and HUB devices are not supported.

The OTG port on FX3 is also capable of ACA charger detection based on the USB Battery Charging specification 1.1 or Motorola EMU specification depending on the selected configuration. The peripheral type detection is based on the ID pin state. The OTG port supports D+ pulsing based SRP and also supports Host Negotiation Protocol (HNP).

### 5.39.2 Typedef Documentation

#### 5.39.2.1 [typedef enum CyU3POtgChargerDetectMode\\_t CyU3POtgChargerDetectMode\\_t](#)

Various charger detection methods supported.

#### Description

The FX3 device can detect chargers based on standard ACA requirements as well as Motorola EMU (enhanced mini USB) requirements. This enumeration lists the various charger detect modes available on FX3.

Charger detection is based on the state of the OTG ID pin and so is available only in CY\_U3P\_OTG\_MODE\_OTG mode of operation.

#### See also

[CyU3POtgMode\\_t](#)  
[CyU3POtgConfig\\_t](#)

### 5.39.2.2 `typedef struct CyU3POtgConfig_t CyU3POtgConfig_t`

OTG configuration information.

#### Description

This structure encapsulates all of the OTG configuration information, and is taken in as an argument by the CyU3POtgStart function.

#### See also

[CyU3POtgMode\\_t](#)  
[CyU3POtgEventCallback\\_t](#)  
[CyU3POtgStart](#)

### 5.39.2.3 `typedef enum CyU3POtgEvent_t CyU3POtgEvent_t`

List of OTG events.

#### Description

The enumeration lists the various OTG events that the USB driver provides to the user application.

#### See also

[CyU3POtgEventCallback\\_t](#)  
[CyU3POtgPeripheralType\\_t](#)

### 5.39.2.4 `typedef void(* CyU3POtgEventCallback_t)(CyU3POtgEvent_t event, uint32_t input )`

OTG event callback function.

#### Description

This type defines the prototype of the event callback function that is called by the USB driver to notify the application about OTG events.

The input to the callback is dependant on the actual event. For the CY\_U3P\_OTG\_PERIPHERAL\_CHANGE event, this specifies the type of peripheral detected (CyU3POtgPeripheralType\_t). For the VBus change event, it specifies the status of VBus.

#### See also

[CyU3POtgConfig\\_t](#)  
[CyU3POtgEvent\\_t](#)  
[CyU3POtgPeripheralType\\_t](#)

### 5.39.2.5 `typedef enum CyU3POtgMode_t CyU3POtgMode_t`

OTG modes of operation.

#### Description

The FX3 device has a single USB port which can function in multiple modes. It can act as a USB 2.0 host or as a USB 3.0 device. It can also route the USB 2.0 lines (D+/D-) to the UART block for car-kit mode of operation.

This enumeration lists the various modes of operation allowed for the device. The default mode of operation is CY\_U3P\_OTG\_MODE\_DEVICE\_ONLY.

#### See also

[CyU3POtgConfig\\_t](#)

### 5.39.2.6 `typedef enum CyU3POtgPeripheralType_t CyU3POtgPeripheralType_t`

List of OTG peripheral types.

#### Description

This enumeration lists the various types of OTG peripherals that can be detected by the FX3.

#### See also

[CyU3POtgEventCallback\\_t](#)  
[CyU3POtgEvent\\_t](#)

### 5.39.3 Enumeration Type Documentation

#### 5.39.3.1 `enum CyU3POtgChargerDetectMode_t`

Various charger detection methods supported.

#### Description

The FX3 device can detect chargers based on standard ACA requirements as well as Motorola EMU (enhanced mini USB) requirements. This enumeration lists the various charger detect modes available on FX3.

Charger detection is based on the state of the OTG ID pin and so is available only in CY\_U3P\_OTG\_MODE\_OTG mode of operation.

#### See also

[CyU3POtgMode\\_t](#)  
[CyU3POtgConfig\\_t](#)

#### Enumerator

**CY\_U3P\_OTG\_CHARGER\_DETECT\_ACA\_MODE** Charger detection is based on standard ACA charger mode. This is the default mode.

**CY\_U3P\_OTG\_CHARGER\_DETECT\_MOT\_EMU** Charger detection is based on Motorola Enhanced Mini USB (EMU) requirements.

**CY\_U3P\_OTG\_CHARGER\_DETECT\_NUM\_MODES** Number of charger detection modes.

#### 5.39.3.2 `enum CyU3POtgEvent_t`

List of OTG events.

#### Description

The enumeration lists the various OTG events that the USB driver provides to the user application.

#### See also

[CyU3POtgEventCallback\\_t](#)  
[CyU3POtgPeripheralType\\_t](#)

#### Enumerator

**CY\_U3P\_OTG\_PERIPHERAL\_CHANGE** The OTG peripheral attached to FX3 has changed (removed or new device connected). The parameter to the event callback identifies the type of peripheral attached.

**CY\_U3P\_OTG\_SR\_P\_DETECT** The remote device has initiated an SRP request. On receiving this request, the user is expected to turn on the VBus.

**CY\_U3P\_OTG\_VBUS\_VALID\_CHANGE** Notifies that VBus state has changed. The parameter is CyTrue if VBus is active and CyFalse if VBus is not active.

### 5.39.3.3 enum CyU3POtgMode\_t

OTG modes of operation.

#### Description

The FX3 device has a single USB port which can function in multiple modes. It can act as a USB 2.0 host or as a USB 3.0 device. It can also route the USB 2.0 lines (D+/D-) to the UART block for car-kit mode of operation.

This enumeration lists the various modes of operation allowed for the device. The default mode of operation is CY\_U3P\_OTG\_MODE\_DEVICE\_ONLY.

#### See also

[CyU3POtgConfig\\_t](#)

#### Enumerator

**CY\_U3P\_OTG\_MODE\_DEVICE\_ONLY** USB port acts in device only mode. The ID pin value is ignored, and charger detection is disabled.

**CY\_U3P\_OTG\_MODE\_HOST\_ONLY** USB port acts in host only mode. The ID pin value is ignored, and charger detection is disabled.

**CY\_U3P\_OTG\_MODE\_OTG** USB port acts in OTG mode and identifies the mode of operation based on the state of the ID pin. This mode also enables charger detection based on the ID pin.

**CY\_U3P\_OTG\_MODE\_CARKIT\_PPORT** The D+/D- lines are routed to the P-Port pads PIB\_CTL11 and PIB\_CTL12. Charger detection is disabled.

**CY\_U3P\_OTG\_MODE\_CARKIT\_UART** The D+/D- lines are routed to the FX3 UART lines. FX3 UART will not function in this mode. Charger detection is disabled.

**CY\_U3P\_OTG\_NUM\_MODES** Number of OTG modes.

### 5.39.3.4 enum CyU3POtgPeripheralType\_t

List of OTG peripheral types.

#### Description

This enumeration lists the various types of OTG peripherals that can be detected by the FX3.

#### See also

[CyU3POtgEventCallback\\_t](#)

[CyU3POtgEvent\\_t](#)

#### Enumerator

**CY\_U3P\_OTG\_TYPE\_DISABLED** Device type not identified because the OTG mode detection is disabled.

**CY\_U3P\_OTG\_TYPE\_A\_CABLE** OTG A-type peripheral cable connected to FX3. FX3 is expected to behave as an OTG host. This mode is specified based on the ID pin state alone, and does not guarantee that a device has been connected. The FX3 device can either enable the VBus at this stage or wait for the peripheral to initiate an SRP request.

**CY\_U3P\_OTG\_TYPE\_B\_CABLE** OTG B-type peripheral cable connected to FX3. FX3 is expected to act as an OTG device by default. The FX3 device is expected to wait for the VBus to be valid. If this does not happen, then CyU3POtgSrpStart API can be invoked for initiating SRP.

**CY\_U3P\_OTG\_TYPE ACA\_A\_CHG** ACA RID\_A\_CHG charger. FX3 is expected to behave as OTG host. VBus is already available from the charger.

**CY\_U3P\_OTG\_TYPE ACA\_B\_CHG** ACA RID\_B\_CHG charger. FX3 can charge and initiate SRP. The remote host is not asserting VBus or is absent.

**CY\_U3P\_OTG\_TYPE ACA\_C\_CHG** ACA RID\_C\_CHG charger. FX3 device can charge and can connect but cannot initiate SRP as VBus is already asserted by the remote host.

**CY\_U3P\_OTG\_TYPE\_MOT\_MPX200** Motorola MPX.200 VPA  
**CY\_U3P\_OTG\_TYPE\_MOT\_CHG** Motorola non-intelligent charger.  
**CY\_U3P\_OTG\_TYPE\_MOT\_MID** Motorola mid rate charger.  
**CY\_U3P\_OTG\_TYPE\_MOT\_FAST** Motorola fast charger.

### 5.39.4 Function Documentation

#### 5.39.4.1 CyU3POtgMode\_t CyU3POtgGetMode ( void )

Retrieve the currently selected OTG operating mode.

##### Description

This function retrieves the active OTG operation mode.

##### Return value

OTG mode selected at the time CyU3POtgStart call.

##### See also

[CyU3POtgMode\\_t](#)  
[CyU3POtgStart](#)

#### 5.39.4.2 CyU3POtgPeripheralType\_t CyU3POtgGetPeripheralType ( void )

Identify the type of USB peripheral attached to FX3.

##### Description

This function returns the type of USB peripheral attached to FX3. This function can return the correct value only if there is a valid VBus or VBatt.

##### Return value

Type of peripheral attached (CyU3POtgPeripheralType\_t)

##### See also

[CyU3POtgPeripheralType\\_t](#)

#### 5.39.4.3 CyU3PReturnStatus\_t CyU3POtgHnpEnable ( CyBool\_t isEnabled )

Initiate a HNP role change.

##### Description

This API initiates a OTG role change. This should only be called when both the host and device mode USB stacks in FX3 firmware are disabled.

If the isEnabled parameter is true, the following sequence is performed:

If this API is called when in 'A' session, the API assumes that the remote device wants to get host role and will allow subsequent CyU3PUsbStart call to go through. If this API is called when in 'B' session, the API assumes that the remote host wants to relinquish control and will allow subsequent CyU3PUsbHostStart call to go through. It should be noted that the previous configuration has to be stopped before invoking this call.

If the isEnabled parameter is false, the API allows the devices to revert to their original roles. The previous configuration has to be stopped before calling this API.

The following sequence should be followed when FX3 is default USB host:

1. The FX3 hosts sends down the SetFeature request for a\_hnp\_support indicating to the remote device that the host can do a role change. This is required for only legacy peripherals.

2. Remote devices request for an HNP via the session request bit.
3. Finish / abort all on-going transfers.
4. FX3 host sends down the SetFeature request for b\_hnp\_enable. This is to indicate to the remote device that the host is ready to initiate a role change.
5. The [CyU3PUsbHostPortSuspend\(\)](#) API should be used to suspend the port.
6. Once the port is suspended, FX3 should wait for the remote device to get disconnected (CY\_U3P\_USB\_HOST\_EVENT\_DISCONNECT host event). If this does not happen within the spec defined time, then the host can either resume host mode operation or it can end the session.
7. If the remote device got disconnected, FX3 should then call [CyU3PUsbHostStop\(\)](#) to stop the host mode of operation.
8. Enable role change by calling [CyU3POtgHnpEnable](#) (CyTrue).
9. Call [CyU3PUsbStart\(\)](#) to start the device mode stack.

The following sequence should be followed when FX3 is default USB device:

1. When FX3 requires a role change, it should first respond to a OTG GetStatus request with the session request flag set. When using fast enumeration mode, this can be done using [CyU3POtgRequestHnp](#) (CyTrue) call.
2. FX3 should wait until the remote host enables HNP by issuing SetFeature request for b\_hnp\_enable.
3. Once the SetFeature request is received, FX3 should wait for the bus to be suspended within the spec defined time. If this does not happen, then the device can either disconnect and end session or it can remain connected until the remote host resumes operation.
4. If the bus is suspended, then disconnect from the USB bus by calling [CyU3PConnectState](#) (CyFalse, CyFalse).
5. Now FX3 should stop the device mode stack by cleaning up all DMA channels and then finally calling [CyU3PUsbStop\(\)](#).
6. FX3 should then start the role change by calling [CyU3POtgHnpEnable](#) (CyTrue).
7. FX3 should then start the host mode operation by calling [CyU3PUsbHostStart\(\)](#).

#### Note

As HNP is role reversal, it has to be explicitly disabled. This only gets disabled if the user calls [CyU3POtgStop\(\)](#) or [CyU3POtgHnpEnable\(CyFalse\)](#); or if the OTG peripheral is disconnected.

#### Return value

[CY\\_U3P\\_SUCCESS](#) - The call was successful.  
[CY\\_U3P\\_ERROR\\_NOT\\_SUPPORTED](#) - Not in OTG mode.  
[CY\\_U3P\\_ERROR\\_NOT\\_STARTED](#) - The module is not started.  
[CY\\_U3P\\_ERROR\\_INVALID\\_SEQUENCE](#) - Device or host stack is still active.

**See also**

[CyU3POtgStart](#)  
[CyU3POtgRequestHnp](#)  
[CyU3POtgStop](#)  
[CyU3PUsbStart](#)  
[CyU3PUsbStop](#)  
[CyU3PUsbHostStart](#)  
[CyU3PUsbHostStop](#)  
[CyU3PUsbHostPortSuspend](#)

**Parameters**

<i>isEnable</i>	Whether to initiate or reverse the HNP role change.
-----------------	---

**5.39.4.4 CyBool\_t CyU3POtgIsDeviceMode( void )**

Check whether USB device (peripheral) mode operation is permitted.

**Description**

This function determines the mode of OTG operation and checks whether the FX3 can initiate the device (peripheral) mode of operation.

**Return value**

CyTrue - Device mode of operation is allowed.  
CyFalse - Device mode of operation is not allowed.

**See also**

[CyU3POtgStart](#)  
[CyU3PUsbStart](#)

**5.39.4.5 CyBool\_t CyU3POtgIsHnpEnabled( void )**

Check if a role reversal mode is active or not.

**Description**

This API checks whether HNP role reversal is currently requested by the user.

**Return value**

CyTrue - HNP role change is active.  
CyFalse - HNP role change is not active.

**See also**

[CyU3POtgStart](#)  
[CyU3POtgEventCallback\\_t](#)

**5.39.4.6 CyBool\_t CyU3POtgIsHostMode( void )**

Check whether USB host mode operation is permitted.

**Description**

This function determines the mode of OTG operation and checks whether the FX3 can initiate host mode operation.

**Return value**

CyTrue - Host mode of operation is allowed.  
CyFalse - Host mode of operation is not allowed.

See also

[CyU3POtgStart](#)  
[CyU3PUsbHostStart](#)

#### 5.39.4.7 CyBool\_t CyU3POtgIsStarted ( void )

Check whether the OTG module has been started.

##### Description

This API can be used to check whether the OTG module in the FX3 firmware has been started.

##### Return value

CyTrue - OTG module has been started.

CyFalse - OTG module is not running.

See also

[CyU3POtgStart](#)  
[CyU3POtgStop](#)

#### 5.39.4.8 CyBool\_t CyU3POtgIsVBusValid ( void )

Check if a valid VBus is available.

##### Description

The API can be used to determine the state of the VBus. Since the USB module can function properly only with the VBus enabled, this can be used to determine when to start the device / host stacks.

Notification of VBus state change can be received through the registered OTG event callback function as well.

##### Return value

CyTrue - VBus is valid.

CyFalse - A valid VBus is not available.

See also

[CyU3POtgStart](#)  
[CyU3POtgEventCallback\\_t](#)

#### 5.39.4.9 CyU3PReturnStatus\_t CyU3POtgRequestHnp ( CyBool\_t isEnabled )

Set the HNP request bit in the device status word.

##### Description

This API is valid only in device mode of operation. To initiate HNP, the device need to set the session request bit in the device status word. If USB control requests are being handled in user callback, this needs to be handled by the user.

Since this is a role change request, the flag is not cleared until CyU3PUsbStop or CyU3POtgRequestHnp (CyFalse) is called, or there is an OTG peripheral change. In case of a change in the OTG peripheral attached, the library will automatically clear the session request flag.

##### Return value

CY\_U3P\_SUCCESS - The call was successful. CY\_U3P\_ERROR\_NOT\_SUPPORTED - Not in OTG mode. CY\_U3P\_ERROR\_NOT\_STARTED - Device stack is not started.

See also

[CyU3POtgStart](#)  
[CyU3POtgHnpEnable](#)

**Parameters**

<i>isEnable</i>	Whether to set or clear the host request flag.
-----------------	--

**5.39.4.10 CyU3PReturnStatus\_t CyU3POtgSrpAbort( void )**

Abort SRP request.

**Description**

The USB driver repeats the SRP request until VBus is detected. This function instructs the driver to abort the periodic SRP requests.

**Return value**

CY\_U3P\_SUCCESS - The call was successful.

CY\_U3P\_ERROR\_NOT\_STARTED - The module is not started.

CY\_U3P\_ERROR\_INVALID\_SEQUENCE - Wrong mode or host stack is still running.

**See also**

[CyU3POtgStart](#)

[CyU3POtgSrpStart](#)

**5.39.4.11 CyU3PReturnStatus\_t CyU3POtgSrpStart( uint32\_t repeatInterval )**

Initiate an SRP request.

**Description**

This API is valid only when the FX3 is functioning as a USB 2.0 device, and is used to start the SRP request. This is expected to cause the USB host to enable the VBus voltage.

The CY\_U3P\_OTG\_VBUS\_VALID\_CHANGE event is sent when a valid VBus voltage is detected. The OTG module will automatically repeat the SRP request until the VBus voltage is valid, or until CyU3POtgSrpAbort() has been called.

The SRP repetition interval in milli-seconds is set through a parameter to this API.

**Return value**

CY\_U3P\_SUCCESS - The call was successful.

CY\_U3P\_ERROR\_NOT\_STARTED - The module is not yet started.

CY\_U3P\_ERROR\_BAD\_ARGUMENT - Input parameter is invalid.

CY\_U3P\_ERROR\_INVALID\_SEQUENCE - Host / device stack is still active or not in the correct mode.

**See also**

[CyU3POtgStart](#)

[CyU3POtgSrpAbort](#)

**Parameters**

<i>repeatInterval</i>	SRP repeat interval in ms. The valid range is from 500 ms to 10s.
-----------------------	---

**5.39.4.12 CyU3PReturnStatus\_t CyU3POtgStart( CyU3POtgConfig\_t \* cfg )**

Initialize the OTG module.

**Description**

The function initializes the OTG functionality in the FX3 USB block. At this point, the device is ready to detect any device/host/charger connection based on the state of the ID pin. Once the mode of operation is detected by the USB driver and notified through the OTG event callback, the corresponding start API needs to be invoked.

Carkit mode allows the USB 2.0 D+ and D- lines to be routed to either P-Port IOs or the UART TX/RX IO lines. These IOs cannot be used for the normal function when the carkit mode is active. Also, USB connections cannot be enabled when the carkit mode is active. The carkit mode can be disabled by invoking the CyU3POtgStop API.

**Return value**

CY\_U3P\_SUCCESS - when the configuration was successful.  
 CY\_U3P\_ERROR\_NOT\_SUPPORTED - if the FX3 device in use does not support the OTG feature.  
 CY\_U3P\_ERROR\_NULL\_POINTER - if the input parameter is NULL.  
 CY\_U3P\_ERROR\_ALREADY\_STARTED - the module was already started.  
 CY\_U3P\_ERROR\_INVALID\_SEQUENCE - the CY\_U3P\_OTG\_MODE\_DEVICE\_ONLY mode was already started.  
 CY\_U3P\_ERROR\_BAD\_ARGUMENT - some input parameters are invalid.

**See also**

[CyU3POtgConfig\\_t](#)  
[CyU3POtgStop](#)  
[CyU3POtgIsStarted](#)  
[CyU3PUsbStart](#)  
[CyU3PUsbHostStart](#)

**Parameters**

<i>cfg</i>	OTG configuration information.
------------	--------------------------------

**5.39.4.13 CyU3PReturnStatus\_t CyU3POtgStop( void )**

Disable the OTG module.

**Description**

This function disables OTG mode detection on the FX3 device. This expects that the USB host/device mode operation has been shut down by calling the corresponding stop function.

**Return value**

CY\_U3P\_SUCCESS - The API call was successful.  
 CY\_U3P\_ERROR\_NOT\_STARTED - The module has not been started yet.  
 CY\_U3P\_ERROR\_INVALID\_SEQUENCE - The device or host stack is running.

**See also**

[CyU3PUsbStop](#)  
[CyU3PUsbHostStop](#)  
[CyU3POtgStart](#)  
[CyU3POtgIsStarted](#)

**5.40 firmware/u3p\_firmware/inc/cyu3utils.h File Reference**

Utility functions provided by the FX3 library.

```
#include "cyu3types.h"
#include "cyfx3_api.h"
#include "assert.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

## Macros

- `#define CY_U3P_MIN(a, b) (((a) > (b)) ? (b) : (a))`  
*Find the minimum of two numbers.*
- `#define CY_U3P_MAX(a, b) (((a) > (b)) ? (a) : (b))`  
*Find the maximum of two numbers.*
- `#define CY_U3P_MAKELWORD(u, l) ((uint16_t)((u) << 8) | (l)))`  
*Create a word (16-bit) from two 8-bit numbers.*
- `#define CY_U3P_GET_LSB(w) ((uint8_t)((w) & UINT8_MAX))`  
*Get the LS byte from a 16-bit number.*
- `#define CY_U3P_GET_LSW(d) ((uint16_t)((d) & UINT16_MAX))`  
*Get the LS word from a 32-bit number.*
- `#define CY_U3P_GET_MSB(w) ((uint8_t)((w) >> 8))`  
*Get the MS byte from a 16-bit number.*
- `#define CY_U3P_GET_MSW(d) ((uint16_t)((d) >> 16))`  
*Get the MS word from a 32-bit number.*
- `#define CY_U3P_MAKEDWORD(b3, b2, b1, b0)`  
*Create a double word (32-bit) from four 8-bit numbers.*
- `#define CY_U3P_DWORD_GET_BYTE0(d) ((uint8_t)((d) & 0xFF))`  
*Retrieves byte 0 from a 32 bit number.*
- `#define CY_U3P_DWORD_GET_BYTE1(d) ((uint8_t)((d) >> 8) & 0xFF))`  
*Retrieves byte 1 from a 32 bit number.*
- `#define CY_U3P_DWORD_GET_BYTE2(d) ((uint8_t)((d) >> 16) & 0xFF))`  
*Retrieves byte 2 from a 32 bit number.*
- `#define CY_U3P_DWORD_GET_BYTE3(d) ((uint8_t)((d) >> 24) & 0xFF))`  
*Retrieves byte 3 from a 32 bit number.*
- `#define CY_U3P_MAKEDWORD1(uw, lw) ((uint32_t)((((uint32_t)(uw) << 16) | ((uint32_t)(lw))))))`  
*Join two 16-bit words into a 32 bit double word.*
- `#define CyU3PAssert(cond) assert(cond)`  
*Assert function used to log errors when expected conditions are not satisfied.*

## Functions

- `void CyU3PMemCopy32 (uint32_t *dest, uint32_t *src, uint32_t count)`  
*Copy data 32-bits at a time from one memory location to another.*
- `void CyU3PBusyWait (uint16_t usWait)`  
*Delay function based on busy spinning in a loop.*
- `CyU3PReturnStatus_t CyU3PComputeChecksum (uint32_t *buffer, uint32_t length, uint32_t *chkSum)`  
*Compute a checksum over a user specified data buffer.*
- `CyU3PReturnStatus_t CyU3PReadDeviceRegisters (uint32_t *regAddr, uint8_t numRegs, uint32_t *dataBuf)`  
*Function to read one or more FX3 device registers.*
- `CyU3PReturnStatus_t CyU3PWriteDeviceRegisters (uint32_t *regAddr, uint8_t numRegs, uint32_t *dataBuf)`  
*Function to write one or more FX3 device registers.*

### 5.40.1 Detailed Description

Utility functions provided by the FX3 library.

#### Description

The utility APIs are generic functions that are provided as helpers for the FX3 APIs.

### 5.40.2 Macro Definition Documentation

#### 5.40.2.1 #define CY\_U3P\_MAKEDWORD( b3, b2, b1, b0 )

##### Value:

```
((uint32_t) (((uint32_t)(b3)) << 24) | (((uint32_t)(b2)) << 16) | \
((uint32_t)(b1)) << 8) | ((uint32_t)(b0)))
```

Create a double word (32-bit) from four 8-bit numbers.

### 5.40.3 Function Documentation

#### 5.40.3.1 void CyU3PBusyWait ( uint16\_t usWait )

Delay function based on busy spinning in a loop.

##### Description

This function is used to insert small delays (of the order of micro-seconds) into the firmware application. The delay is implemented using a busy spin loop and can be used anywhere. This API should not be used for large delays as other lower or same priority threads will not be able to run during this.

##### Return value

None

##### See also

[CyU3PThreadSleep](#)

##### Parameters

<i>usWait</i>	Delay duration in micro-seconds.
---------------	----------------------------------

#### 5.40.3.2 CyU3PReturnStatus\_t CyU3PComputeChecksum ( uint32\_t \* buffer, uint32\_t length, uint32\_t \* chkSum )

Compute a checksum over a user specified data buffer.

##### Description

This function computes the binary sum of all values in a user specified data buffer and can be used as a simple checksum to verify data consistency. This checksum API is used by the boot-loader to determine the checksum as well.

##### Return value

CY\_U3P\_SUCCESS if the checksum is successfully computed.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the parameters provided are erroneous.

##### Parameters

<i>buffer</i>	Pointer to data buffer on which to calculate the checksum.
<i>length</i>	Length of data buffer on which to calculate the checksum.

**Parameters**

<i>chkSum</i>	Pointer to buffer that will be filled with the checksum.
---------------	--

5.40.3.3 void CyU3PMemCopy32 ( *uint32\_t \* dest, uint32\_t \* src, uint32\_t count* )

Copy data 32-bits at a time from one memory location to another.

**Description**

This is a memcpy equivalent function. This requires that the addresses provided are four byte aligned. Since the ARM core is 32-bit, this API does a faster copy of data than the 1 byte equivalent (CyU3PMemCopy). This API is also used by the firmware library, and handles cases where the source and destination buffers are overlapping.

**Return value**

None

**See also**

[CyU3PMemCopy](#)

**Parameters**

<i>dest</i>	Pointer to destination buffer.
<i>src</i>	Pointer to source buffer.
<i>count</i>	Size of the buffer (in words) to be copied.

5.40.3.4 CyU3PReturnStatus\_t CyU3PReadDeviceRegisters ( *uvint32\_t \* regAddr, uint8\_t numRegs, uint32\_t \* dataBuf* )

Function to read one or more FX3 device registers.

**Description**

The FX3 device hardware implements a number of Control and Status registers that govern the behavior of and report the current status of each of the blocks. This function is used to read one or more contiguous registers from the register space of the FX3 device.

**Return value**

CY\_U3P\_SUCCESS if the register read(s) is/are successful.

CY\_U3P\_ERROR\_BAD\_ARGUMENT if the arguments passed in are erroneous.

**See also**

[CyU3PWriteDeviceRegisters](#)

**Parameters**

<i>regAddr</i>	Address of first register to be read.
<i>numRegs</i>	Number of registers to be read.
<i>dataBuf</i>	Pointer to data buffer into which the registers are to be read.

---

5.40.3.5 **CyU3PReturnStatus\_t CyU3PWriteDeviceRegisters ( *uint32\_t \* regAddr, uint8\_t numRegs, uint32\_t \* dataBuf* )**

Function to write one or more FX3 device registers.

#### Description

This function is used to write one or more contiguous registers from the register space of the FX3 device.

#### Note

Use this function with caution and preferably under guidance from Cypress support personnel. The function does not implement any validity/side effect checks on the values being written into the registers.

#### Return value

**CY\_U3P\_SUCCESS** if the register write(s) is/are successful.

**CY\_U3P\_ERROR\_BAD\_ARGUMENT** if the arguments passed in are erroneous.

#### See also

[CyU3PReadDeviceRegisters](#)

#### Parameters

<i>regAddr</i>	Address of first register to be written.
<i>numRegs</i>	Number of registers to be written.
<i>dataBuf</i>	Pointer to data buffer containing data to be written to the registers.

## 5.41 firmware/u3p\_firmware/inc/cyu3vic.h File Reference

Internal functions for managing the VIC and interrupts on the FX3 device.

```
#include "cyu3types.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

#### Enumerations

- enum **CyU3PVicVector\_t** {
 CY\_U3P\_VIC\_GCTL\_CORE\_VECTOR = 0, CY\_U3P\_VIC\_SWI\_VECTOR, CY\_U3P\_VIC\_DEBUG\_RX\_VECTOR, CY\_U3P\_VIC\_DEBUG\_TX\_VECTOR, CY\_U3P\_VIC\_WDT\_VECTOR, CY\_U3P\_VIC\_BIAS\_CORRECT\_VECTOR, CY\_U3P\_VIC\_PIB\_DMA\_VECTOR, CY\_U3P\_VIC\_PIB\_CORE\_VECTOR, CY\_U3P\_VIC\_UIB\_DMA\_VECTOR, CY\_U3P\_VIC\_UIB\_CORE\_VECTOR, CY\_U3P\_VIC\_UIB\_CONTROL\_VECTOR, CY\_U3P\_VIC\_SIB\_DMA\_VECTOR, CY\_U3P\_VIC\_SIB0\_CORE\_VECTOR, CY\_U3P\_VIC\_SIB1\_CORE\_VECTOR, CY\_U3P\_VIC\_RESERVE\_D\_15\_VECTOR, CY\_U3P\_VIC\_I2C\_CORE\_VECTOR, CY\_U3P\_VIC\_I2S\_CORE\_VECTOR, CY\_U3P\_VIC\_SPI\_CORE\_VECTOR, CY\_U3P\_VIC\_UART\_CORE\_VECTOR, CY\_U3P\_VIC\_LPP\_DMA\_VECTOR, CY\_U3P\_VIC\_GCTL\_PWR\_VECTOR, CY\_U3P\_VIC\_NUM\_VECTORS }

*Vector numbers for different interrupt sources.*

#### Functions

- void [CyU3PVicEnableInt](#) (*uint32\_t vectorNum*)

- void [CyU3PVicDisableInt](#) (uint32\_t vectorNum)
 

*Enable the interrupt for the specified vector number.*
- void [CyU3PVicClearInt](#) (void)
 

*Disable the interrupt for the specified vector number.*
- uint32\_t [CyU3PVicGetIntStatus](#) (void)
 

*Clear any interrupt that has occurred.*
- uint32\_t [CyU3PVicIRQGetStatus](#) (void)
 

*Get the raw interrupt status.*
- void [CyU3PVicIntSetPriority](#) (uint32\_t vectorNum, uint32\_t priority)
 

*Set the priority level of the interrupt vector.*
- uint32\_t [CyU3PVicIntGetPriority](#) (uint32\_t vectorNum)
 

*Get the priority level of the interrupt vector.*
- void [CyU3PVicInit](#) (void)
 

*The function initializes the VIC.*
- void [CyU3PVicSetupIntVectors](#) (void)
 

*The function initializes the interrupt vector table.*
- uint32\_t [CyU3PVicDisableAllInterrupts](#) (void)
 

*This function disables all FX3 interrupts at the VIC level.*
- void [CyU3PVicEnableInterrupts](#) (uint32\_t mask)
 

*This function enables the specified interrupts at the VIC level.*

### 5.41.1 Detailed Description

Internal functions for managing the VIC and interrupts on the FX3 device.

### 5.41.2 Enumeration Type Documentation

#### 5.41.2.1 enum CyU3PVicVector\_t

Vector numbers for different interrupt sources.

##### Description

The following enumeration lists the interrupt vector numbers on the FX3 device.

##### Enumerator

- CY\_U3P\_VIC\_GCTL\_CORE\_VECTOR** 0: Low power mode entry/exit interrupt.
- CY\_U3P\_VIC\_SWI\_VECTOR** 1: Software interrupt.
- CY\_U3P\_VIC\_DEBUG\_RX\_VECTOR** 2: Unused debug vector.
- CY\_U3P\_VIC\_DEBUG\_TX\_VECTOR** 3: Unused debug vector.
- CY\_U3P\_VIC\_WDT\_VECTOR** 4: Watchdog timer interrupt.
- CY\_U3P\_VIC\_BIAS\_CORRECT\_VECTOR** 5: Timer for PVT Bias correction.
- CY\_U3P\_VIC\_PIB\_DMA\_VECTOR** 6: GPIO (PIB) DMA interrupt.
- CY\_U3P\_VIC\_PIB\_CORE\_VECTOR** 7: GPIO (PIB) Core interrupt.
- CY\_U3P\_VIC\_UIB\_DMA\_VECTOR** 8: USB DMA interrupt.
- CY\_U3P\_VIC\_UIB\_CORE\_VECTOR** 9: USB core interrupt.
- CY\_U3P\_VIC\_UIB\_CONTROL\_VECTOR** 10: Unused interrupt vector.
- CY\_U3P\_VIC\_SIB\_DMA\_VECTOR** 11: Storage port DMA interrupt (FX3S only).
- CY\_U3P\_VIC\_SIB0\_CORE\_VECTOR** 12: Storage port 0 core interrupt (FX3S only).

**CY\_U3P\_VIC\_SIB1\_CORE\_VECTOR** 13: Storage port 1 core interrupt (FX3S only).  
**CY\_U3P\_VIC\_RESERVED\_15\_VECTOR** 14: Unused interrupt vector.  
**CY\_U3P\_VIC\_I2C\_CORE\_VECTOR** 15: I2C block interrupt.  
**CY\_U3P\_VIC\_I2S\_CORE\_VECTOR** 16: I2S block interrupt.  
**CY\_U3P\_VIC\_SPI\_CORE\_VECTOR** 17: SPI block interrupt.  
**CY\_U3P\_VIC\_UART\_CORE\_VECTOR** 18: UART block interrupt.  
**CY\_U3P\_VIC\_GPIO\_CORE\_VECTOR** 19: GPIO block interrupt.  
**CY\_U3P\_VIC\_LPP\_DMA\_VECTOR** 20: Serial peripheral (I2C, I2S, SPI and UART) DMA interrupt.  
**CY\_U3P\_VIC\_GCTL\_PWR\_VECTOR** 21: VBus detect interrupt.  
**CY\_U3P\_VIC\_NUM\_VECTORS** Number of valid FX3 interrupt vectors.

### 5.41.3 Function Documentation

#### 5.41.3.1 void CyU3PVicClearInt( void )

Clear any interrupt that has occurred.

##### Description

This function marks the current interrupt cleared at the VIC level, so that the VIC can raise the next pending interrupt. The actual cause of the interrupt has to be cleared or masked out before doing this.

This function should not be used directly by the user application.

##### Return value

None

##### See also

[CyU3PVicEnableInt](#)  
[CyU3PVicDisableInt](#)

#### 5.41.3.2 uint32\_t CyU3PVicDisableAllInterrupts( void )

This function disables all FX3 interrupts at the VIC level.

##### Description

This function can be used to disable all FX3 interrupts at the VIC level. The function returns a mask that represent the interrupts that were enabled before this function was called. It is expected that this mask would be used to re-enable the interrupts using the CyU3PVicEnableInterrupts function.

##### Return value

A mask that represents the interrupts that were enabled before this call.

##### See also

[CyU3PVicEnableInterrupts](#)

#### 5.41.3.3 void CyU3PVicDisableInt( uint32\_t vectorNum )

Disable the interrupt for the specified vector number.

##### Description

Disable the interrupt with the specified vector number. All FX3 interrupts are enabled/disabled at appropriate times by the corresponding drivers in the FX3 firmware. This function should not be directly used by the user application.

##### Return value

None

## See also

[CyU3PVicEnableInt](#)  
[CyU3PVicClearInt](#)

## Parameters

<i>vectorNum</i>	Interrupt vector number (0 - CY_U3P_VIC_NUM_VECTORS) to be disabled.
------------------	--

5.41.3.4 void CyU3PVicEnableInt ( *uint32\_t vectorNum* )

Enable the interrupt for the specified vector number.

**Description**

Enable the interrupt with the specified vector number. All FX3 interrupts are enabled/disabled at appropriate times by the corresponding drivers in the FX3 firmware. This function should not be directly used by the user application.

**Return value**

None

## See also

[CyU3PVicDisableInt](#)  
[CyU3PVicClearInt](#)

## Parameters

<i>vectorNum</i>	Vector number to be enabled (0 - CY_U3P_VIC_NUM_VECTORS).
------------------	---

5.41.3.5 void CyU3PVicEnableInterrupts ( *uint32\_t mask* )

This function enables the specified interrupts at the VIC level.

**Description**

This function can be used to re-enable interrupts that were previously disabled through the CyU3PVicDisableAllInterrupts function. These two functions can be used together to save and restore interrupt states across critical sections of code.

**Return value**

None

## See also

[CyU3PVicDisableAllInterrupts](#)

## Parameters

<i>mask</i>	Bit-mask representing interrupts to be enabled.
-------------	---

5.41.3.6 void CyU3PVicInit ( *void* )

The function initializes the VIC.

**Description**

This function initializes the PL192 Vectored Interrupt Controller on the FX3 device and sets up the interrupt vector addresses for all FX3 device interrupts.

This function is called internally during device initialization and should not be called directly.

**Return value**

None

**5.41.3.7 uint32\_t CyU3PVicIntGetPriority ( uint32\_t *vectorNum* )**

Get the priority level of the interrupt vector.

**Description**

Returns the currently programmed priority level for the specified interrupt vector. 0 is the highest priority level, and 15 is the lowest priority level.

**Return value**

The priority level for the specified interrupt vector.

**See also**

[CyU3PVicIntSetPriority](#)

**Parameters**

<i>vectorNum</i>	Interrupt vector number.
------------------	--------------------------

**5.41.3.8 uint32\_t CyU3PVicIntGetStatus ( void )**

Get the raw interrupt status.

**Description**

Get a bit vector that represents the current status of all FX3 device interrupts. The bit vector will report an interrupt even if it is currently disabled at the VIC level.

**Return value**

Bit vector reporting the current status of all FX3 interrupts.

**See also**

[CyU3PVicIRQGetStatus](#)

**5.41.3.9 void CyU3PVicIntSetPriority ( uint32\_t *vectorNum*, uint32\_t *priority* )**

Set the priority level of the interrupt vector.

**Description**

Set the priority level for a specified interrupt vector. This function is not used in the library and is not expected to be called. Interrupts in FX3 firmware are split into two groups: a high priority group which is not pre-emptable; and a low priority group that is pre-emptable.

**Return value**

None

**See also**

[CyU3PVicIntGetPriority](#)

**Parameters**

<i>vectorNum</i>	Interrupt vector number (0 - CY_U3P_VIC_NUM_VECTORS).
<i>priority</i>	Priority level to be set (0 - 15)

5.41.3.10 `uint32_t CyU3PVicIRQGetStatus( void )`

Get the IRQ interrupt status.

**Description**

This function returns a bit vector that reports the interrupt vectors that are both active and enabled.

**Return value**

Bit vector representing the active and enabled interrupts.

**See also**

[CyU3PVicIntGetStatus](#)

5.41.3.11 `void CyU3PVicSetupIntVectors( void )`

The function initializes the interrupt vector table.

**Description**

This function sets up the interrupt vector address table for the FX3 device, and is called internally by the library during device initialization.

**Return value**

None

**See also**

[CyU3PVicInit](#)

# Index

ALPHA\_CX3\_START\_SCK0  
    cyu3mipicsi.h, 323

ALPHA\_CX3\_START\_SCK1  
    cyu3mipicsi.h, 323

actBuffer  
    CyFx3BootDmaSockRegs\_t, 27

actChain  
    CyFx3BootDmaSockRegs\_t, 27

actSize  
    CyFx3BootDmaSockRegs\_t, 27

actSync  
    CyFx3BootDmaSockRegs\_t, 27

activeConsIndex  
    CyU3PDmaChannel, 45  
    CyU3PDmaMultiChannel, 53

activeDscr  
    CyU3PDmaSocket\_t, 58

activePartition  
    CyU3PSibGlobalData, 84

activeProdIndex  
    CyU3PDmaChannel, 45  
    CyU3PDmaMultiChannel, 53

activeUnitId  
    CyU3PSibCtxt, 80

addrCIS  
    CyU3PSdioCardRegs, 79

alloc\_id  
    MemBlockInfo, 95

alloc\_size  
    MemBlockInfo, 95

bIdx0  
    CyFx3BootUsbEp0Pkt\_t, 37

bIdx1  
    CyFx3BootUsbEp0Pkt\_t, 37

bReq  
    CyFx3BootUsbEp0Pkt\_t, 37

bVal0  
    CyFx3BootUsbEp0Pkt\_t, 37

bVal1  
    CyFx3BootUsbEp0Pkt\_t, 37

baudRate  
    CyFx3BootUartConfig\_t, 36  
    CyU3PUartConfig\_t, 91

bitRate  
    CyFx3BootI2cConfig\_t, 30  
    CyU3PI2cConfig\_t, 68

blkLen  
    CyU3PCardCtxt, 39  
    CyU3PSibDevInfo, 82

blockSize  
    CyU3PSibLunInfo, 87

bmReqType  
    CyFx3BootUsbEp0Pkt\_t, 37

buffer  
    CyFx3BootDmaDescriptor\_t, 25  
    CyFx3BootI2cPreamble\_t, 31  
    CyU3PDmaBuffer\_t, 43  
    CyU3PDmaDescriptor\_t, 51  
    CyU3PI2cPreamble\_t, 69

buffer\_p  
    CyU3PDmaCBInput\_t, 44

bufferCount  
    CyU3PDmaMultiChannel, 53

burstLen  
    CyFx3BootUsbEpConfig\_t, 38  
    CyU3PEpConfig\_t, 61

busTimeout  
    CyFx3BootI2cConfig\_t, 30  
    CyU3PI2cConfig\_t, 68

busWidth  
    CyU3PCardCtxt, 39  
    CyU3PSibDevInfo, 82

CCCRVersion  
    CyU3PSdioCardRegs, 79

CX3\_ALPHA\_START  
    cyu3mipicsi.h, 323

CX3\_FULL\_BUFFER\_IN\_SCK0  
    cyu3mipicsi.h, 324

CX3\_FULL\_BUFFER\_IN\_SCK1  
    cyu3mipicsi.h, 324

CX3\_IDLE\_SCK0  
    cyu3mipicsi.h, 324

CX3\_IDLE\_SCK1  
    cyu3mipicsi.h, 324

CX3\_IDLE  
    cyu3mipicsi.h, 324

CX3\_NUMBER\_OF\_STATES  
    cyu3mipicsi.h, 324

CX3\_PARTIAL\_BUFFER\_IN\_SCK0  
    cyu3mipicsi.h, 324

CX3\_PARTIAL\_BUFFER\_IN\_SCK1  
    cyu3mipicsi.h, 324

CX3\_PUSH\_DATA\_SCK0  
    cyu3mipicsi.h, 324

CX3\_PUSH\_DATA\_SCK1  
    cyu3mipicsi.h, 324

CX3\_PUSH\_DATA\_TO\_SCK0  
    cyu3mipicsi.h, 324

CX3\_PUSH\_DATA\_TO\_SCK1  
  cyu3mipicsi.h, 324

CX3\_START\_SCK0  
  cyu3mipicsi.h, 325

CX3\_START\_SCK1  
  cyu3mipicsi.h, 325

CX3\_START  
  cyu3mipicsi.h, 325

CX3\_WAIT\_FOR\_FRAME\_START\_SCK0  
  cyu3mipicsi.h, 325

CX3\_WAIT\_FOR\_FRAME\_START\_SCK1  
  cyu3mipicsi.h, 325

CX3\_WAIT\_FOR\_FRAME\_START  
  cyu3mipicsi.h, 325

CX3\_WAIT\_FULL\_SCK0\_NEXT\_SCK1  
  cyu3mipicsi.h, 325

CX3\_WAIT\_FULL\_SCK1\_NEXT\_SCK0  
  cyu3mipicsi.h, 325

CX3\_WAIT\_TO\_FILL\_SCK0  
  cyu3mipicsi.h, 325

CX3\_WAIT\_TO\_FILL\_SCK1  
  cyu3mipicsi.h, 325

CY\_DMA\_CPU\_SOCKET\_CONS  
  cyfx3dma.h, 110

CY\_DMA\_CPU\_SOCKET\_PROD  
  cyfx3dma.h, 110

CY\_DMA\_LPP\_SOCKET\_I2C\_CONS  
  cyfx3dma.h, 108

CY\_DMA\_LPP\_SOCKET\_I2C\_PROD  
  cyfx3dma.h, 108

CY\_DMA\_LPP\_SOCKET\_I2S\_LEFT  
  cyfx3dma.h, 108

CY\_DMA\_LPP\_SOCKET\_I2S\_RIGHT  
  cyfx3dma.h, 108

CY\_DMA\_LPP\_SOCKET\_SPI\_CONS  
  cyfx3dma.h, 108

CY\_DMA\_LPP\_SOCKET\_SPI\_PROD  
  cyfx3dma.h, 108

CY\_DMA\_LPP\_SOCKET\_UART\_CONS  
  cyfx3dma.h, 108

CY\_DMA\_LPP\_SOCKET\_UART\_PROD  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_0  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_1  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_10  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_11  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_12  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_13  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_14  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_15  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_16  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_17  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_18  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_19  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_2  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_20  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_21  
  cyfx3dma.h, 109

CY\_DMA\_PIB\_SOCKET\_22  
  cyfx3dma.h, 109

CY\_DMA\_PIB\_SOCKET\_23  
  cyfx3dma.h, 109

CY\_DMA\_PIB\_SOCKET\_24  
  cyfx3dma.h, 109

CY\_DMA\_PIB\_SOCKET\_25  
  cyfx3dma.h, 109

CY\_DMA\_PIB\_SOCKET\_26  
  cyfx3dma.h, 109

CY\_DMA\_PIB\_SOCKET\_27  
  cyfx3dma.h, 109

CY\_DMA\_PIB\_SOCKET\_28  
  cyfx3dma.h, 109

CY\_DMA\_PIB\_SOCKET\_29  
  cyfx3dma.h, 109

CY\_DMA\_PIB\_SOCKET\_3  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_30  
  cyfx3dma.h, 109

CY\_DMA\_PIB\_SOCKET\_31  
  cyfx3dma.h, 109

CY\_DMA\_PIB\_SOCKET\_4  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_5  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_6  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_7  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_8  
  cyfx3dma.h, 108

CY\_DMA\_PIB\_SOCKET\_9  
  cyfx3dma.h, 108

CY\_DMA\_SIB\_SOCKET\_0  
  cyfx3dma.h, 109

CY\_DMA\_SIB\_SOCKET\_1  
  cyfx3dma.h, 109

CY\_DMA\_SIB\_SOCKET\_2  
  cyfx3dma.h, 109

CY\_DMA\_SIB\_SOCKET\_3  
  cyfx3dma.h, 109

CY\_DMA\_SIB\_SOCKET\_4  
  cyfx3dma.h, 109

CY\_DMA\_SIB\_SOCKET\_5  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_CONS\_0  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_CONS\_1  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_CONS\_10  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_CONS\_11  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_CONS\_12  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_CONS\_13  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_CONS\_14  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_CONS\_15  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_CONS\_2  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_CONS\_3  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_CONS\_4  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_CONS\_5  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_CONS\_6  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_CONS\_7  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_CONS\_8  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_CONS\_9  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_PROD\_0  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_PROD\_1  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_PROD\_10  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_PROD\_11  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_PROD\_12  
    cyfx3dma.h, 110

CY\_DMA\_UIB\_SOCKET\_PROD\_13  
    cyfx3dma.h, 110

CY\_DMA\_UIB\_SOCKET\_PROD\_14  
    cyfx3dma.h, 110

CY\_DMA\_UIB\_SOCKET\_PROD\_15  
    cyfx3dma.h, 110

CY\_DMA\_UIB\_SOCKET\_PROD\_2  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_PROD\_3  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_PROD\_4  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_PROD\_5  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_PROD\_6  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_PROD\_7  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_PROD\_8  
    cyfx3dma.h, 109

CY\_DMA\_UIB\_SOCKET\_PROD\_9  
    cyfx3dma.h, 109

CY\_FX3\_BOOT\_ERROR\_ABORTED  
    cyfx3error.h, 115

CY\_FX3\_BOOT\_ERROR\_ALREADY\_STARTED  
    cyfx3error.h, 115

CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT  
    cyfx3error.h, 115

CY\_FX3\_BOOT\_ERROR\_BAD\_DESCRIPTOR\_TYPE  
    cyfx3error.h, 115

CY\_FX3\_BOOT\_ERROR\_FAILURE  
    cyfx3error.h, 116

CY\_FX3\_BOOT\_ERROR\_I2C  
    cyfx3error.h, 116

CY\_FX3\_BOOT\_ERROR\_INVALID\_DMA\_ADDR  
    cyfx3error.h, 115

CY\_FX3\_BOOT\_ERROR\_MEMORY\_ERROR  
    cyfx3error.h, 115

CY\_FX3\_BOOT\_ERROR\_NO\_REENUM\_REQUIRED  
    cyfx3error.h, 115

CY\_FX3\_BOOT\_ERROR\_NOT\_CONFIGURED  
    cyfx3error.h, 115

CY\_FX3\_BOOT\_ERROR\_NOT\_STARTED  
    cyfx3error.h, 115

CY\_FX3\_BOOT\_ERROR\_NOT\_SUPPORTED  
    cyfx3error.h, 115

CY\_FX3\_BOOT\_ERROR\_NULL\_POINTER  
    cyfx3error.h, 115

CY\_FX3\_BOOT\_ERROR\_TIMEOUT  
    cyfx3error.h, 115

CY\_FX3\_BOOT\_ERROR\_XFER\_FAILURE  
    cyfx3error.h, 115

CY\_FX3\_BOOT\_FULL\_SPEED  
    cyfx3usb.h, 156

CY\_FX3\_BOOT\_GPIO\_INTR\_BOTH\_EDGE  
    cyfx3gpio.h, 118

CY\_FX3\_BOOT\_GPIO\_INTR\_HIGH\_LEVEL  
    cyfx3gpio.h, 118

CY\_FX3\_BOOT\_GPIO\_INTR\_LOW\_LEVEL  
    cyfx3gpio.h, 118

CY\_FX3\_BOOT\_GPIO\_INTR\_NEG\_EDGE  
    cyfx3gpio.h, 118

CY\_FX3\_BOOT\_GPIO\_INTR\_POS\_EDGE  
    cyfx3gpio.h, 118

CY\_FX3\_BOOT\_GPIO\_NO\_INTR  
    cyfx3gpio.h, 118

CY\_FX3\_BOOT\_HIGH\_SPEED  
    cyfx3usb.h, 156

CY\_FX3\_BOOT\_NOT\_CONNECTED  
    cyfx3usb.h, 156

CY\_FX3\_BOOT\_NUM\_CLK\_SRC  
    cyfx3device.h, 100

CY_FX3_BOOT_SPI_NUM_SSN_CTRL cyfx3spi.h, 139	CY_FX3_BOOT_UART_TWO_STOP_BIT cyfx3uart.h, 147
CY_FX3_BOOT_SPI_NUM_SSN_LAG_LEAD cyfx3spi.h, 140	CY_FX3_BOOT_USB_COMPLIANCE cyfx3usb.h, 156
CY_FX3_BOOT_SPI_SSN_CTRL_FW cyfx3spi.h, 139	CY_FX3_BOOT_USB_CONNECT cyfx3usb.h, 156
CY_FX3_BOOT_SPI_SSN_CTRL_HW_CPHA_BASED cyfx3spi.h, 139	CY_FX3_BOOT_USB_DISCONNECT cyfx3usb.h, 156
CY_FX3_BOOT_SPI_SSN_CTRL_HW_EACH_WORD cyfx3spi.h, 139	CY_FX3_BOOT_USB_EP_BULK cyfx3usb.h, 155
CY_FX3_BOOT_SPI_SSN_CTRL_HW_END_OF_XFER cyfx3spi.h, 139	CY_FX3_BOOT_USB_EP_CONTROL cyfx3usb.h, 155
CY_FX3_BOOT_SPI_SSN_CTRL_NONE cyfx3spi.h, 139	CY_FX3_BOOT_USB_EP_INTR cyfx3usb.h, 155
CY_FX3_BOOT_SPI_SSN_LAG_LEAD_HALF_CLK cyfx3spi.h, 140	CY_FX3_BOOT_USB_EP_ISO cyfx3usb.h, 155
CY_FX3_BOOT_SPI_SSN_LAG_LEAD_ONE_CLK cyfx3spi.h, 140	CY_FX3_BOOT_USB_IN_SS_DISCONNECT cyfx3usb.h, 156
CY_FX3_BOOT_SPI_SSN_LAG_LEAD_ONE_HALF_CLK cyfx3spi.h, 140	CY_FX3_BOOT_USB_RESET cyfx3usb.h, 156
CY_FX3_BOOT_SPI_SSN_LAG_LEAD_ZERO_CLK cyfx3spi.h, 140	CY_FX3_BOOT_USB_RESUME cyfx3usb.h, 156
CY_FX3_BOOT_SUCCESS cyfx3error.h, 115	CY_FX3_BOOT_USB_SUSPEND cyfx3usb.h, 156
CY_FX3_BOOT_SUPER_SPEED cyfx3usb.h, 156	CY_FX3_DMA_GETDSCR_BY_INDEX cyfx3dma.h, 106
CY_FX3_BOOT_SYS_CLK_BY_16 cyfx3device.h, 100	CY_FX3_DMA_LPP_SOCKCNT cyfx3dma.h, 106
CY_FX3_BOOT_SYS_CLK_BY_2 cyfx3device.h, 100	CY_FX3_DMA_MAX_DSCR_INDEX cyfx3dma.h, 106
CY_FX3_BOOT_SYS_CLK_BY_4 cyfx3device.h, 100	CY_FX3_DMA_MIN_DSCR_INDEX cyfx3dma.h, 106
CY_FX3_BOOT_SYS_CLK cyfx3device.h, 100	CY_FX3_DMA_PIB_SOCKCNT cyfx3dma.h, 106
CY_FX3_BOOT_UART_BAUDRATE_115200 cyfx3uart.h, 147	CY_FX3_DMA_USB_IN_SOCKCNT cyfx3dma.h, 106
CY_FX3_BOOT_UART_BAUDRATE_19200 cyfx3uart.h, 146	CY_FX3_DMA_USB_OUT_SOCKCNT cyfx3dma.h, 106
CY_FX3_BOOT_UART_BAUDRATE_38400 cyfx3uart.h, 147	CY_FX3_GPIO_IO_MODE_NONE cyfx3gpio.h, 118
CY_FX3_BOOT_UART_BAUDRATE_4800 cyfx3uart.h, 146	CY_FX3_GPIO_IO_MODE_WPD cyfx3gpio.h, 118
CY_FX3_BOOT_UART_BAUDRATE_57600 cyfx3uart.h, 147	CY_FX3_GPIO_IO_MODE_WPU cyfx3gpio.h, 118
CY_FX3_BOOT_UART_BAUDRATE_9600 cyfx3uart.h, 146	CY_FX3_LPP_DMA_DSCR_INDEX cyfx3dma.h, 106
CY_FX3_BOOT_UART EVEN_PARITY cyfx3uart.h, 147	CY_FX3_PIB_DMA_DSCR_INDEX cyfx3dma.h, 106
CY_FX3_BOOT_UART_NO_PARITY cyfx3uart.h, 147	CY_FX3_USB_DMA_DSCR_INDEX cyfx3dma.h, 107
CY_FX3_BOOT_UART_NUM_PARITY cyfx3uart.h, 147	CY_FX3_USB_MAX_STRING_DESC_INDEX cyfx3usb.h, 154
CY_FX3_BOOT_UART_ODD_PARITY cyfx3uart.h, 147	CY_U2P_I2S_NUM_PAD_MODES cyu3i2s.h, 303
CY_FX3_BOOT_UART_ONE_STOP_BIT cyfx3uart.h, 147	CY_U3P_BOOT_PARTITION_NUM_BLKS cyu3sibpp.h, 440
	CY_U3P_BOS_DESCR cyfx3usb.h, 156, 157

cyu3usbconst.h, 542, 543  
CY\_U3P\_CARD\_BUS\_WIDTH\_1\_BIT  
    cyu3cardmgr.h, 188  
CY\_U3P\_CARD\_BUS\_WIDTH\_4\_BIT  
    cyu3cardmgr.h, 188  
CY\_U3P\_CARD\_BUS\_WIDTH\_8\_BIT  
    cyu3cardmgr.h, 188  
CY\_U3P\_CONTAINER\_ID\_CAPB\_TYPE  
    cyu3usbconst.h, 543  
CY\_U3P\_CPU\_IP\_BLOCK\_ID  
    cyu3socket.h, 447  
CY\_U3P\_CPU\_SOCKET\_CONS  
    cyu3dma.h, 225  
CY\_U3P\_CPU\_SOCKET\_PROD  
    cyu3dma.h, 225  
CY\_U3P\_CSI\_DF\_RAW10  
    cyu3mipicsi.h, 329  
CY\_U3P\_CSI\_DF\_RAW12  
    cyu3mipicsi.h, 329  
CY\_U3P\_CSI\_DF\_RAW14  
    cyu3mipicsi.h, 329  
CY\_U3P\_CSI\_DF\_RAW8  
    cyu3mipicsi.h, 329  
CY\_U3P\_CSI\_DF\_RGB565\_0  
    cyu3mipicsi.h, 329  
CY\_U3P\_CSI\_DF\_RGB565\_1  
    cyu3mipicsi.h, 329  
CY\_U3P\_CSI\_DF\_RGB565\_2  
    cyu3mipicsi.h, 329  
CY\_U3P\_CSI\_DF\_RGB666\_0  
    cyu3mipicsi.h, 329  
CY\_U3P\_CSI\_DF\_RGB666\_1  
    cyu3mipicsi.h, 329  
CY\_U3P\_CSI\_DF\_RGB888  
    cyu3mipicsi.h, 329  
CY\_U3P\_CSI\_DF\_YUV422\_10  
    cyu3mipicsi.h, 329  
CY\_U3P\_CSI\_DF\_YUV422\_8\_0  
    cyu3mipicsi.h, 329  
CY\_U3P\_CSI\_DF\_YUV422\_8\_1  
    cyu3mipicsi.h, 329  
CY\_U3P\_CSI\_DF\_YUV422\_8\_2  
    cyu3mipicsi.h, 329  
CY\_U3P\_CSI\_HARD\_RST  
    cyu3mipicsi.h, 331  
CY\_U3P\_CSI\_IO\_XRES  
    cyu3mipicsi.h, 331  
CY\_U3P\_CSI\_IO\_XSHUTDOWN  
    cyu3mipicsi.h, 331  
CY\_U3P\_CSI\_PLL\_CLK\_DIV\_2  
    cyu3mipicsi.h, 330  
CY\_U3P\_CSI\_PLL\_CLK\_DIV\_4  
    cyu3mipicsi.h, 330  
CY\_U3P\_CSI\_PLL\_CLK\_DIV\_8  
    cyu3mipicsi.h, 330  
CY\_U3P\_CSI\_PLL\_CLK\_DIV\_INVALID  
    cyu3mipicsi.h, 330  
CY\_U3P\_CSI\_PLL\_FRS\_125\_250M  
    cyu3mipicsi.h, 330  
CY\_U3P\_CSI\_PLL\_FRS\_250\_500M  
    cyu3mipicsi.h, 330  
CY\_U3P\_CSI\_PLL\_FRS\_500\_1000M  
    cyu3mipicsi.h, 330  
CY\_U3P\_CSI\_PLL\_FRS\_63\_125M  
    cyu3mipicsi.h, 330  
CY\_U3P\_CSI\_SOFT\_RST  
    cyu3mipicsi.h, 331  
CY\_U3P\_DEVICE\_CAPB\_DESCR  
    cyfx3usb.h, 156, 157  
    cyu3usbconst.h, 542, 543  
CY\_U3P\_DMA\_ABORTED  
    cyu3dma.h, 225  
CY\_U3P\_DMA\_ACTIVE  
    cyu3dma.h, 225  
CY\_U3P\_DMA\_CB\_ABORTED  
    cyu3dma.h, 221  
CY\_U3P\_DMA\_CB\_CONS\_EVENT  
    cyu3dma.h, 221  
CY\_U3P\_DMA\_CB\_CONS\_SUSP  
    cyu3dma.h, 221  
CY\_U3P\_DMA\_CB\_ERROR  
    cyu3dma.h, 221  
CY\_U3P\_DMA\_CB\_PROD\_EVENT  
    cyu3dma.h, 221  
CY\_U3P\_DMA\_CB\_PROD\_SUSP  
    cyu3dma.h, 221  
CY\_U3P\_DMA\_CB\_RECV\_CPLT  
    cyu3dma.h, 221  
CY\_U3P\_DMA\_CB\_SEND\_CPLT  
    cyu3dma.h, 221  
CY\_U3P\_DMA\_CB\_XFER\_CPLT  
    cyu3dma.h, 221  
CY\_U3P\_DMA\_CONFIGURED  
    cyu3dma.h, 225  
CY\_U3P\_DMA\_CONS\_OVERRIDE  
    cyu3dma.h, 225  
CY\_U3P\_DMA\_ERROR  
    cyu3dma.h, 225  
CY\_U3P\_DMA\_IN\_COMPLETION  
    cyu3dma.h, 225  
CY\_U3P\_DMA\_MODE\_BUFFER  
    cyu3dma.h, 221  
CY\_U3P\_DMA\_MODE\_BYTE  
    cyu3dma.h, 221  
CY\_U3P\_DMA\_NOT\_CONFIGURED  
    cyu3dma.h, 225  
CY\_U3P\_DMA\_NUM\_MODES  
    cyu3dma.h, 221  
CY\_U3P\_DMA\_NUM\_SINGLE\_TYPES  
    cyu3dma.h, 226  
CY\_U3P\_DMA\_NUM\_STATES  
    cyu3dma.h, 225  
CY\_U3P\_DMA\_NUM\_TYPES  
    cyu3dma.h, 222  
CY\_U3P\_DMA\_PROD\_OVERRIDE  
    cyu3dma.h, 225

CY\_U3P\_DMA\_RECV\_COMPLETED  
cyu3dma.h, 226

CY\_U3P\_DMA\_SCK\_SUSP\_CONS\_PARTIAL\_BUF  
cyu3dma.h, 222

CY\_U3P\_DMA\_SCK\_SUSP\_CUR\_BUF  
cyu3dma.h, 222

CY\_U3P\_DMA\_SCK\_SUSP\_EOP  
cyu3dma.h, 222

CY\_U3P\_DMA\_SCK\_SUSP\_NONE  
cyu3dma.h, 222

CY\_U3P\_DMA\_SEND\_COMPLETED  
cyu3dma.h, 226

CY\_U3P\_DMA\_TYPE\_AUTO\_MANY\_TO\_ONE  
cyu3dma.h, 222

CY\_U3P\_DMA\_TYPE\_AUTO\_ONE\_TO\_MANY  
cyu3dma.h, 222

CY\_U3P\_DMA\_TYPE\_AUTO\_SIGNAL  
cyu3dma.h, 226

CY\_U3P\_DMA\_TYPE\_AUTO  
cyu3dma.h, 226

CY\_U3P\_DMA\_TYPE\_MANUAL\_IN  
cyu3dma.h, 226

CY\_U3P\_DMA\_TYPE\_MANUAL\_MANY\_TO\_ONE  
cyu3dma.h, 222

CY\_U3P\_DMA\_TYPE\_MANUAL\_ONE\_TO\_MANY  
cyu3dma.h, 222

CY\_U3P\_DMA\_TYPE\_MANUAL\_OUT  
cyu3dma.h, 226

CY\_U3P\_DMA\_TYPE\_MANUAL  
cyu3dma.h, 226

CY\_U3P\_DMA\_TYPE\_MULTICAST  
cyu3dma.h, 222

CY\_U3P\_XFER\_COMPLETED  
cyu3dma.h, 225

CY\_U3P\_DS\_FULL\_STRENGTH  
cyu3system.h, 469

CY\_U3P\_DS\_HALF\_STRENGTH  
cyu3system.h, 469

CY\_U3P\_DS\_QUARTER\_STRENGTH  
cyu3system.h, 469

CY\_U3P\_DS\_THREE\_QUARTER\_STRENGTH  
cyu3system.h, 469

CY\_U3P\_ERROR\_ABORTED  
cyu3error.h, 255

CY\_U3P\_ERROR\_ACTIVATE\_FAILED  
cyu3error.h, 255

CY\_U3P\_ERROR\_ALREADY\_PARTITIONED  
cyu3error.h, 256

CY\_U3P\_ERROR\_ALREADY\_STARTED  
cyu3error.h, 255

CY\_U3P\_ERROR\_BAD\_ARGUMENT  
cyu3error.h, 255

CY\_U3P\_ERROR\_BAD\_CMD\_ARG  
cyu3error.h, 256

CY\_U3P\_ERROR\_BAD\_DESCRIPTOR\_TYPE  
cyu3error.h, 255

CY\_U3P\_ERROR\_BAD\_ENUM\_METHOD  
cyu3error.h, 255

CY\_U3P\_ERROR\_BAD\_EVENT\_GRP  
cyu3error.h, 254

CY\_U3P\_ERROR\_BAD\_INDEX  
cyu3error.h, 255

CY\_U3P\_ERROR\_BAD\_MUTEX  
cyu3error.h, 255

CY\_U3P\_ERROR\_BAD\_OPTION  
cyu3error.h, 254

CY\_U3P\_ERROR\_BAD\_PARTITION  
cyu3error.h, 256

CY\_U3P\_ERROR\_BAD\_POINTER  
cyu3error.h, 254

CY\_U3P\_ERROR\_BAD\_POOL  
cyu3error.h, 254

CY\_U3P\_ERROR\_BAD\_PRIORITY  
cyu3error.h, 255

CY\_U3P\_ERROR\_BAD\_QUEUE  
cyu3error.h, 254

CY\_U3P\_ERROR\_BAD\_SEMAPHORE  
cyu3error.h, 254

CY\_U3P\_ERROR\_BAD\_SIZE  
cyu3error.h, 254

CY\_U3P\_ERROR\_BAD\_THREAD  
cyu3error.h, 255

CY\_U3P\_ERROR\_BAD\_THRESHOLD  
cyu3error.h, 255

CY\_U3P\_ERROR\_BAD\_TICK  
cyu3error.h, 255

CY\_U3P\_ERROR\_BAD\_TIMER  
cyu3error.h, 255

CY\_U3P\_ERROR\_BLOCK\_FAILURE  
cyu3error.h, 255

CY\_U3P\_ERROR\_CARD\_FORCE\_ERASE  
cyu3error.h, 256

CY\_U3P\_ERROR\_CARD\_LOCK\_FAILURE  
cyu3error.h, 256

CY\_U3P\_ERROR\_CARD\_LOCKED  
cyu3error.h, 256

CY\_U3P\_ERROR\_CARD\_NOT\_ACTIVE  
cyu3error.h, 256

CY\_U3P\_ERROR\_CARD\_UNHEALTHY  
cyu3error.h, 256

CY\_U3P\_ERROR\_CARD\_WRONG\_RESPONSE  
cyu3error.h, 256

CY\_U3P\_ERROR\_CARD\_WRONG\_STATE  
cyu3error.h, 256

CY\_U3P\_ERROR\_CHANNEL\_CREATE\_FAILED  
cyu3error.h, 255

CY\_U3P\_ERROR\_CHANNEL\_DESTROY\_FAILED  
cyu3error.h, 255

CY\_U3P\_ERROR\_CMD\_NOT\_SUPPORTED  
cyu3error.h, 256

CY\_U3P\_ERROR\_CRC  
cyu3error.h, 256

CY\_U3P\_ERROR\_DELETE\_FAILED  
cyu3error.h, 255

CY\_U3P\_ERROR\_DELETED  
cyu3error.h, 254

CY_U3P_ERROR_DEVICE_BUSY	CY_U3P_ERROR_NOT_STARTED
cyu3error.h, 256	cyu3error.h, 255
CY_U3P_ERROR_DMA_FAILURE	CY_U3P_ERROR_NOT_SUPPORTED
cyu3error.h, 255	cyu3error.h, 255
CY_U3P_ERROR_FAILURE	CY_U3P_ERROR_NULL_POINTER
cyu3error.h, 255	cyu3error.h, 255
CY_U3P_ERROR_FEATURE_NOT_ENABLED	CY_U3P_ERROR_OPERN_DISABLED
cyu3error.h, 255	cyu3error.h, 256
CY_U3P_ERROR_ILLEGAL_CMD	CY_U3P_ERROR_QUEUE_EMPTY
cyu3error.h, 256	cyu3error.h, 254
CY_U3P_ERROR_INHERIT_FAILED	CY_U3P_ERROR_QUEUE_FULL
cyu3error.h, 255	cyu3error.h, 254
CY_U3P_ERROR_INVALID_ADDR	CY_U3P_ERROR_READ_WRITE_ABORTED
cyu3error.h, 256	cyu3error.h, 256
CY_U3P_ERROR_INVALID_BLOCKSIZE	CY_U3P_ERROR_RESUME_FAILED
cyu3error.h, 256	cyu3error.h, 255
CY_U3P_ERROR_INVALID_CALLER	CY_U3P_ERROR_SDIO_UNKNOWN
cyu3error.h, 255	cyu3error.h, 256
CY_U3P_ERROR_INVALID_CONFIGURATION	CY_U3P_ERROR_SEMGET_FAILED
cyu3error.h, 255	cyu3error.h, 255
CY_U3P_ERROR_INVALID_DEV	CY_U3P_ERROR_SIB_INIT
cyu3error.h, 256	cyu3error.h, 256
CY_U3P_ERROR_INVALID_FUNCTION	CY_U3P_ERROR_STALLED
cyu3error.h, 256	cyu3error.h, 255
CY_U3P_ERROR_INVALID_SEQUENCE	CY_U3P_ERROR_STANDBY_FAILED
cyu3error.h, 255	cyu3error.h, 255
CY_U3P_ERROR_INVALID_UNIT	CY_U3P_ERROR_SUSPEND_FAILED
cyu3error.h, 256	cyu3error.h, 255
CY_U3P_ERROR_INVALID_VOLTAGE_RANGE	CY_U3P_ERROR_SUSPEND_LIFTED
cyu3error.h, 255	cyu3error.h, 255
CY_U3P_ERROR_INVALID_WAIT	CY_U3P_ERROR_TIMEOUT
cyu3error.h, 254	cyu3error.h, 255
CY_U3P_ERROR_IO_ABORTED	CY_U3P_ERROR_TUPLE_NOT_FOUND
cyu3error.h, 256	cyu3error.h, 256
CY_U3P_ERROR_IO_SUSPENDED	CY_U3P_ERROR_UNINITIALIZED_FUNCTION
cyu3error.h, 256	cyu3error.h, 256
CY_U3P_ERROR_LOST_ARBITRATION	CY_U3P_ERROR_UNSUPPORTED_CARD
cyu3error.h, 255	cyu3error.h, 256
CY_U3P_ERROR_MEDIA_FAILURE	CY_U3P_ERROR_WAIT_ABORT_FAILED
cyu3error.h, 256	cyu3error.h, 255
CY_U3P_ERROR_MEMORY_ERROR	CY_U3P_ERROR_WAIT_ABORTED
cyu3error.h, 255	cyu3error.h, 255
CY_U3P_ERROR_MUTEX_FAILURE	CY_U3P_ERROR_WRITE_PROTECTED
cyu3error.h, 255	cyu3error.h, 256
CY_U3P_ERROR_MUTEX_PUT_FAILED	CY_U3P_ERROR_XFER_CANCELLED
cyu3error.h, 255	cyu3error.h, 255
CY_U3P_ERROR_NO_EVENTS	CY_U3P_FULL_SPEED
cyu3error.h, 254	cyu3usb.h, 512
CY_U3P_ERROR_NO_METADATA	CY_U3P_GPIO_INTR_BOTH_EDGE
cyu3error.h, 256	cyu3gpio.h, 277
CY_U3P_ERROR_NO_REENUM_REQUIRED	CY_U3P_GPIO_INTR_HIGH_LEVEL
cyu3error.h, 256	cyu3gpio.h, 277
CY_U3P_ERROR_NOT_CONFIGURED	CY_U3P_GPIO_INTR_LOW_LEVEL
cyu3error.h, 255	cyu3gpio.h, 277
CY_U3P_ERROR_NOT_IDLE	CY_U3P_GPIO_INTR_NEG_EDGE
cyu3error.h, 255	cyu3gpio.h, 277
CY_U3P_ERROR_NOT_PARTITIONED	CY_U3P_GPIO_INTR_POS_EDGE
cyu3error.h, 256	cyu3gpio.h, 277

CY\_U3P\_GPIO\_INTR\_TIMER\_THRES  
cyu3gpio.h, 277

CY\_U3P\_GPIO\_INTR\_TIMER\_ZERO  
cyu3gpio.h, 277

CY\_U3P\_GPIO\_IO\_MODE\_NONE  
cyu3lpp.h, 310

CY\_U3P\_GPIO\_IO\_MODE\_WPD  
cyu3lpp.h, 310

CY\_U3P\_GPIO\_IO\_MODE\_WPU  
cyu3lpp.h, 310

CY\_U3P\_GPIO\_MODE\_MEASURE\_ANY\_ONCE  
cyu3gpio.h, 277

CY\_U3P\_GPIO\_MODE\_MEASURE\_ANY  
cyu3gpio.h, 277

CY\_U3P\_GPIO\_MODE\_MEASURE\_HIGH\_ONCE  
cyu3gpio.h, 276

CY\_U3P\_GPIO\_MODE\_MEASURE\_HIGH  
cyu3gpio.h, 276

CY\_U3P\_GPIO\_MODE\_MEASURE\_LOW\_ONCE  
cyu3gpio.h, 276

CY\_U3P\_GPIO\_MODE\_MEASURE\_LOW  
cyu3gpio.h, 276

CY\_U3P\_GPIO\_MODE\_MEASURE\_NEG\_ONCE  
cyu3gpio.h, 277

CY\_U3P\_GPIO\_MODE\_MEASURE\_NEG  
cyu3gpio.h, 276

CY\_U3P\_GPIO\_MODE\_MEASURE\_POS\_ONCE  
cyu3gpio.h, 277

CY\_U3P\_GPIO\_MODE\_MEASURE\_POS  
cyu3gpio.h, 276

CY\_U3P\_GPIO\_MODE\_PULSE\_NOW  
cyu3gpio.h, 276

CY\_U3P\_GPIO\_MODE\_PULSE  
cyu3gpio.h, 276

CY\_U3P\_GPIO\_MODE\_PWM  
cyu3gpio.h, 276

CY\_U3P\_GPIO\_MODE\_SAMPLE\_NOW  
cyu3gpio.h, 276

CY\_U3P\_GPIO\_MODE\_STATIC  
cyu3gpio.h, 276

CY\_U3P\_GPIO\_MODE\_TOGGLE  
cyu3gpio.h, 276

CY\_U3P\_GPIO\_NO\_INTR  
cyu3gpio.h, 277

CY\_U3P\_GPIO\_SIMPLE\_DIV\_BY\_16  
cyu3lpp.h, 310

CY\_U3P\_GPIO\_SIMPLE\_DIV\_BY\_2  
cyu3lpp.h, 310

CY\_U3P\_GPIO\_SIMPLE\_DIV\_BY\_4  
cyu3lpp.h, 310

CY\_U3P\_GPIO\_SIMPLE\_DIV\_BY\_64  
cyu3lpp.h, 310

CY\_U3P\_GPIO\_SIMPLE\_NUM\_DIV  
cyu3lpp.h, 310

CY\_U3P\_GPIO\_TIMER\_ANY\_EDGE  
cyu3gpio.h, 278

CY\_U3P\_GPIO\_TIMER\_HIGH\_FREQ  
cyu3gpio.h, 278

CY\_U3P\_GPIO\_TIMER\_LOW\_FREQ  
cyu3gpio.h, 278

CY\_U3P\_GPIO\_TIMER\_NEG\_EDGE  
cyu3gpio.h, 278

CY\_U3P\_GPIO\_TIMER\_POS\_EDGE  
cyu3gpio.h, 278

CY\_U3P\_GPIO\_TIMER\_RESERVED  
cyu3gpio.h, 278

CY\_U3P\_GPIO\_TIMER\_SHUTDOWN  
cyu3gpio.h, 278

CY\_U3P\_GPIO\_TIMER\_STANDBY\_FREQ  
cyu3gpio.h, 278

CY\_U3P\_HIGH\_SPEED  
cyu3usb.h, 512

CY\_U3P\_I2C\_ERROR\_NAK\_BYTE\_0  
cyu3i2c.h, 291

CY\_U3P\_I2C\_ERROR\_NAK\_BYTE\_1  
cyu3i2c.h, 291

CY\_U3P\_I2C\_ERROR\_NAK\_BYTE\_2  
cyu3i2c.h, 291

CY\_U3P\_I2C\_ERROR\_NAK\_BYTE\_3  
cyu3i2c.h, 291

CY\_U3P\_I2C\_ERROR\_NAK\_BYTE\_4  
cyu3i2c.h, 291

CY\_U3P\_I2C\_ERROR\_NAK\_BYTE\_5  
cyu3i2c.h, 291

CY\_U3P\_I2C\_ERROR\_NAK\_BYTE\_6  
cyu3i2c.h, 291

CY\_U3P\_I2C\_ERROR\_NAK\_BYTE\_7  
cyu3i2c.h, 291

CY\_U3P\_I2C\_ERROR\_NAK\_DATA  
cyu3i2c.h, 291

CY\_U3P\_I2C\_ERROR\_NAK\_RX\_OVERFLOW  
cyu3i2c.h, 291

CY\_U3P\_I2C\_ERROR\_NAK\_RX\_UNDERFLOW  
cyu3i2c.h, 291

CY\_U3P\_I2C\_ERROR\_NAK\_TX\_OVERFLOW  
cyu3i2c.h, 291

CY\_U3P\_I2C\_ERROR\_NAK\_TX\_UNDERFLOW  
cyu3i2c.h, 291

CY\_U3P\_I2C\_ERROR\_PREAMBLE\_EXIT\_NACK\_A←CK  
cyu3i2c.h, 291

CY\_U3P\_I2C\_ERROR\_PREAMBLE\_EXIT  
cyu3i2c.h, 291

CY\_U3P\_I2C\_EVENT\_ERROR  
cyu3i2c.h, 292

CY\_U3P\_I2C\_EVENT\_LOST\_ARBITRATION  
cyu3i2c.h, 292

CY\_U3P\_I2C\_EVENT\_RX\_DONE  
cyu3i2c.h, 292

CY\_U3P\_I2C\_EVENT\_TIMEOUT  
cyu3i2c.h, 292

CY\_U3P\_I2C\_EVENT\_TX\_DONE  
cyu3i2c.h, 292

CY\_U3P\_I2S\_ERROR\_LTX\_OVERFLOW  
cyu3i2s.h, 302

CY\_U3P\_I2S\_ERROR\_LTX\_UNDERFLOW  
cyu3i2s.h, 302

cyu3i2s.h, 302  
CY\_U3P\_I2S\_ERROR\_RTX\_OVERFLOW  
  cyu3i2s.h, 302  
CY\_U3P\_I2S\_ERROR\_RTX\_UNDERFLOW  
  cyu3i2s.h, 302  
CY\_U3P\_I2S\_EVENT\_ERROR  
  cyu3i2s.h, 302  
CY\_U3P\_I2S\_EVENT\_PAUSED  
  cyu3i2s.h, 302  
CY\_U3P\_I2S\_EVENT\_TXL\_DONE  
  cyu3i2s.h, 302  
CY\_U3P\_I2S\_EVENT\_TXR\_DONE  
  cyu3i2s.h, 302  
CY\_U3P\_I2S\_NUM\_BIT\_WIDTH  
  cyu3i2s.h, 303  
CY\_U3P\_I2S\_PAD\_MODE\_CONTINUOUS  
  cyu3i2s.h, 303  
CY\_U3P\_I2S\_PAD\_MODE\_LEFT\_JUSTIFIED  
  cyu3i2s.h, 302  
CY\_U3P\_I2S\_PAD\_MODE\_NORMAL  
  cyu3i2s.h, 302  
CY\_U3P\_I2S\_PAD\_MODE\_RESERVED  
  cyu3i2s.h, 303  
CY\_U3P\_I2S\_PAD\_MODE\_RIGHT\_JUSTIFIED  
  cyu3i2s.h, 302  
CY\_U3P\_I2S\_SAMPLE\_RATE\_16KHz  
  cyu3i2s.h, 303  
CY\_U3P\_I2S\_SAMPLE\_RATE\_192KHz  
  cyu3i2s.h, 303  
CY\_U3P\_I2S\_SAMPLE\_RATE\_32KHz  
  cyu3i2s.h, 303  
CY\_U3P\_I2S\_SAMPLE\_RATE\_44\_1KHz  
  cyu3i2s.h, 303  
CY\_U3P\_I2S\_SAMPLE\_RATE\_48KHz  
  cyu3i2s.h, 303  
CY\_U3P\_I2S\_SAMPLE\_RATE\_8KHz  
  cyu3i2s.h, 303  
CY\_U3P\_I2S\_SAMPLE\_RATE\_96KHz  
  cyu3i2s.h, 303  
CY\_U3P\_I2S\_WIDTH\_16\_BIT  
  cyu3i2s.h, 303  
CY\_U3P\_I2S\_WIDTH\_18\_BIT  
  cyu3i2s.h, 303  
CY\_U3P\_I2S\_WIDTH\_24\_BIT  
  cyu3i2s.h, 303  
CY\_U3P\_I2S\_WIDTH\_32\_BIT  
  cyu3i2s.h, 303  
CY\_U3P\_I2S\_WIDTH\_8\_BIT  
  cyu3i2s.h, 303  
CY\_U3P\_IO\_MATRIX\_LPP\_DEFAULT  
  cyfx3\_api.h, 171  
CY\_U3P\_IO\_MATRIX\_LPP\_I2S\_ONLY  
  cyfx3\_api.h, 171  
CY\_U3P\_IO\_MATRIX\_LPP\_NONE  
  cyfx3\_api.h, 171  
CY\_U3P\_IO\_MATRIX\_LPP\_SPI\_ONLY  
  cyfx3\_api.h, 171  
CY\_U3P\_IO\_MATRIX\_LPP\_UART\_ONLY  
  cyfx3\_api.h, 171  
CY\_U3P\_LPP\_GPIO  
  cyu3system.h, 469  
CY\_U3P\_LPP\_I2C  
  cyu3system.h, 469  
CY\_U3P\_LPP\_I2S  
  cyu3system.h, 469  
CY\_U3P\_LPP\_IP\_BLOCK\_ID  
  cyu3socket.h, 447  
CY\_U3P\_LPP\_SOCKET\_I2C\_CONS  
  cyu3dma.h, 223  
CY\_U3P\_LPP\_SOCKET\_I2C\_PROD  
  cyu3dma.h, 223  
CY\_U3P\_LPP\_SOCKET\_I2S\_LEFT  
  cyu3dma.h, 223  
CY\_U3P\_LPP\_SOCKET\_I2S\_RIGHT  
  cyu3dma.h, 223  
CY\_U3P\_LPP\_SOCKET\_SPI\_CONS  
  cyu3dma.h, 223  
CY\_U3P\_LPP\_SOCKET\_SPI\_PROD  
  cyu3dma.h, 223  
CY\_U3P\_LPP\_SOCKET\_UART\_CONS  
  cyu3dma.h, 223  
CY\_U3P\_LPP\_SOCKET\_UART\_PROD  
  cyu3dma.h, 223  
CY\_U3P\_LPP\_SPI  
  cyu3system.h, 469  
CY\_U3P\_LPP\_UART  
  cyu3system.h, 469  
CY\_U3P\_MAKEDWORD  
  cyu3utils.h, 571  
CY\_U3P\_MAX\_STRING\_DESC\_INDEX  
  cyu3usb.h, 504  
CY\_U3P\_MIPICSI\_BUS\_16  
  cyu3mipicsi.h, 328  
CY\_U3P\_MIPICSI\_BUS\_24  
  cyu3mipicsi.h, 328  
CY\_U3P\_MIPICSI\_BUS\_8  
  cyu3mipicsi.h, 328  
CY\_U3P\_MIPICSI\_I2C\_100KHZ  
  cyu3mipicsi.h, 330  
CY\_U3P\_MIPICSI\_I2C\_400KHZ  
  cyu3mipicsi.h, 330  
CY\_U3P\_MMCF\_BUS\_TEST  
  cyu3cardmgr.h, 189  
CY\_U3P\_MMCF\_SW\_PARTCFG\_BOOT1\_PARAM  
  cyu3cardmgr.h, 186  
CY\_U3P\_MMCF\_SW\_PARTCFG\_BOOT2\_PARAM  
  cyu3cardmgr.h, 186  
CY\_U3P\_MMCF\_SW\_PARTCFG\_USER\_PARAM  
  cyu3cardmgr.h, 186  
CY\_U3P\_NOT\_CONNECTED  
  cyu3usb.h, 512  
CY\_U3P\_NUM\_CLK\_SRC  
  cyu3system.h, 470  
CY\_U3P\_NUM\_IP\_BLOCK\_ID  
  cyu3socket.h, 447  
CY\_U3P\_OTG\_CHARGER\_DETECT\_ACA\_MODE

cyu3usb0tg.h, 562  
**CY\_U3P\_OTG\_CHARGER\_DETECT\_MOT\_EMU**  
 cyu3usb0tg.h, 562  
**CY\_U3P\_OTG\_CHARGER\_DETECT\_NUM\_MODES**  
 cyu3usb0tg.h, 562  
**CY\_U3P\_OTG\_MODE\_CARKIT\_PPORt**  
 cyu3usb0tg.h, 563  
**CY\_U3P\_OTG\_MODE\_CARKIT\_UART**  
 cyu3usb0tg.h, 563  
**CY\_U3P\_OTG\_MODE\_DEVICE\_ONLY**  
 cyu3usb0tg.h, 563  
**CY\_U3P\_OTG\_MODE\_HOST\_ONLY**  
 cyu3usb0tg.h, 563  
**CY\_U3P\_OTG\_MODE\_OTG**  
 cyu3usb0tg.h, 563  
**CY\_U3P\_OTG\_NUM\_MODES**  
 cyu3usb0tg.h, 563  
**CY\_U3P\_OTG\_PERIPHERAL\_CHANGE**  
 cyu3usb0tg.h, 562  
**CY\_U3P\_OTG\_SRp\_DETECT**  
 cyu3usb0tg.h, 562  
**CY\_U3P\_OTG\_TYPE\_A\_CABLE**  
 cyu3usb0tg.h, 563  
**CY\_U3P\_OTG\_TYPE\_ACA\_A\_CHG**  
 cyu3usb0tg.h, 563  
**CY\_U3P\_OTG\_TYPE\_ACA\_B\_CHG**  
 cyu3usb0tg.h, 563  
**CY\_U3P\_OTG\_TYPE\_ACA\_C\_CHG**  
 cyu3usb0tg.h, 563  
**CY\_U3P\_OTG\_TYPE\_B\_CABLE**  
 cyu3usb0tg.h, 563  
**CY\_U3P\_OTG\_TYPE\_DISABLED**  
 cyu3usb0tg.h, 563  
**CY\_U3P\_OTG\_TYPE\_MOT\_CHG**  
 cyu3usb0tg.h, 564  
**CY\_U3P\_OTG\_TYPE\_MOT\_FAST**  
 cyu3usb0tg.h, 564  
**CY\_U3P\_OTG\_TYPE\_MOT\_MID**  
 cyu3usb0tg.h, 564  
**CY\_U3P\_OTG\_TYPE\_MOT\_MPX200**  
 cyu3usb0tg.h, 563  
**CY\_U3P\_OTG\_VBUS\_VALID\_CHANGE**  
 cyu3usb0tg.h, 562  
**CY\_U3P\_PARTITION\_TYPE\_AP\_BOOT**  
 cyu3sibpp.h, 440  
**CY\_U3P\_PARTITION\_TYPE\_BENICIA\_BOOT**  
 cyu3sibpp.h, 440  
**CY\_U3P\_PARTITION\_TYPE\_DATA\_AREA**  
 cyu3sibpp.h, 440  
**CY\_U3P\_PIB\_IP\_BLOCK\_ID**  
 cyu3socket.h, 447  
**CY\_U3P\_PIB\_SOCKET\_0**  
 cyu3dma.h, 223  
**CY\_U3P\_PIB\_SOCKET\_1**  
 cyu3dma.h, 223  
**CY\_U3P\_PIB\_SOCKET\_10**  
 cyu3dma.h, 223  
**CY\_U3P\_PIB\_SOCKET\_11**  
 cyu3dma.h, 223  
**CY\_U3P\_PIB\_SOCKET\_12**  
 cyu3dma.h, 223  
**CY\_U3P\_PIB\_SOCKET\_13**  
 cyu3dma.h, 223  
**CY\_U3P\_PIB\_SOCKET\_14**  
 cyu3dma.h, 223  
**CY\_U3P\_PIB\_SOCKET\_15**  
 cyu3dma.h, 223  
**CY\_U3P\_PIB\_SOCKET\_16**  
 cyu3dma.h, 224  
**CY\_U3P\_PIB\_SOCKET\_17**  
 cyu3dma.h, 224  
**CY\_U3P\_PIB\_SOCKET\_18**  
 cyu3dma.h, 224  
**CY\_U3P\_PIB\_SOCKET\_19**  
 cyu3dma.h, 224  
**CY\_U3P\_PIB\_SOCKET\_2**  
 cyu3dma.h, 223  
**CY\_U3P\_PIB\_SOCKET\_20**  
 cyu3dma.h, 224  
**CY\_U3P\_PIB\_SOCKET\_21**  
 cyu3dma.h, 224  
**CY\_U3P\_PIB\_SOCKET\_22**  
 cyu3dma.h, 224  
**CY\_U3P\_PIB\_SOCKET\_23**  
 cyu3dma.h, 224  
**CY\_U3P\_PIB\_SOCKET\_24**  
 cyu3dma.h, 224  
**CY\_U3P\_PIB\_SOCKET\_25**  
 cyu3dma.h, 224  
**CY\_U3P\_PIB\_SOCKET\_26**  
 cyu3dma.h, 224  
**CY\_U3P\_PIB\_SOCKET\_27**  
 cyu3dma.h, 224  
**CY\_U3P\_PIB\_SOCKET\_28**  
 cyu3dma.h, 224  
**CY\_U3P\_PIB\_SOCKET\_29**  
 cyu3dma.h, 224  
**CY\_U3P\_PIB\_SOCKET\_3**  
 cyu3dma.h, 223  
**CY\_U3P\_PIB\_SOCKET\_30**  
 cyu3dma.h, 224  
**CY\_U3P\_PIB\_SOCKET\_31**  
 cyu3dma.h, 224  
**CY\_U3P\_PIB\_SOCKET\_32**  
 cyu3dma.h, 223  
**CY\_U3P\_PIB\_SOCKET\_4**  
 cyu3dma.h, 223  
**CY\_U3P\_PIB\_SOCKET\_5**  
 cyu3dma.h, 223  
**CY\_U3P\_PIB\_SOCKET\_6**  
 cyu3dma.h, 223  
**CY\_U3P\_PIB\_SOCKET\_7**  
 cyu3dma.h, 223  
**CY\_U3P\_PIB\_SOCKET\_8**  
 cyu3dma.h, 223  
**CY\_U3P\_PIB\_SOCKET\_9**  
 cyu3dma.h, 223  
**CY\_U3P\_PMMC\_BOOT**

cyu3pib.h, 406  
CY\_U3P\_PMMC\_BUSTEST  
    cyu3pib.h, 406  
CY\_U3P\_PMMC\_DISCONNECT  
    cyu3pib.h, 406  
CY\_U3P\_PMMC\_IDENTIFICATION  
    cyu3pib.h, 406  
CY\_U3P\_PMMC\_IDLE  
    cyu3pib.h, 406  
CY\_U3P\_PMMC\_INACTIVE  
    cyu3pib.h, 406  
CY\_U3P\_PMMC\_PGM  
    cyu3pib.h, 406  
CY\_U3P\_PMMC\_PREBOOT  
    cyu3pib.h, 406  
CY\_U3P\_PMMC\_PREIDLE  
    cyu3pib.h, 406  
CY\_U3P\_PMMC\_READY  
    cyu3pib.h, 406  
CY\_U3P\_PMMC\_RECVDATA  
    cyu3pib.h, 406  
CY\_U3P\_PMMC\_SENDDATA  
    cyu3pib.h, 406  
CY\_U3P\_PMMC\_SLEEP  
    cyu3pib.h, 406  
CY\_U3P\_PMMC\_STANDBY  
    cyu3pib.h, 406  
CY\_U3P\_PMMC\_TRANS  
    cyu3pib.h, 406  
CY\_U3P\_PMMC\_WAITIRQ  
    cyu3pib.h, 406  
CY\_U3P\_SD\_CARD\_VER\_1\_X  
    cyu3cardmgr.h, 189  
CY\_U3P\_SD\_CARD\_VER\_2\_0  
    cyu3cardmgr.h, 189  
CY\_U3P\_SD\_CARD\_VER\_3\_0  
    cyu3cardmgr.h, 189  
CY\_U3P\_SD\_MMC\_DATA  
    cyu3cardmgr.h, 189  
CY\_U3P\_SD\_MMC\_DISCONNECT  
    cyu3cardmgr.h, 189  
CY\_U3P\_SD\_MMC\_IDENTIFICATION  
    cyu3cardmgr.h, 189  
CY\_U3P\_SD\_MMC\_IDLE  
    cyu3cardmgr.h, 189  
CY\_U3P\_SD\_MMC\_PROGRAMMING  
    cyu3cardmgr.h, 189  
CY\_U3P\_SD\_MMC\_READY  
    cyu3cardmgr.h, 189  
CY\_U3P\_SD\_MMC\_RECEIVE  
    cyu3cardmgr.h, 189  
CY\_U3P\_SD\_MMC\_STANDBY  
    cyu3cardmgr.h, 189  
CY\_U3P\_SD\_MMC\_TRANSFER  
    cyu3cardmgr.h, 189  
CY\_U3P\_SD\_REG\_CID  
    cyu3sib.h, 418  
CY\_U3P\_SD\_REG\_CSD  
    cyu3sib.h, 418  
CY\_U3P\_SD\_REG\_OCR  
    cyu3sib.h, 418  
CY\_U3P\_SD\_SW\_HIGHSP\_PARAM  
    cyu3cardmgr.h, 187  
CY\_U3P\_SD\_SW\_QUERY\_FUNCTIONS  
    cyu3cardmgr.h, 187  
CY\_U3P\_SD\_SW\_UHS1\_PARAM  
    cyu3cardmgr.h, 187  
CY\_U3P\_SDIO\_CARD\_CAPABILITY\_4BLS  
    cyu3cardmgr\_fx3s.h, 196  
CY\_U3P\_SDIO\_CARD\_CAPABILITY\_E4MI  
    cyu3cardmgr\_fx3s.h, 196  
CY\_U3P\_SDIO\_CARD\_CAPABILITY\_LSC  
    cyu3cardmgr\_fx3s.h, 196  
CY\_U3P\_SDIO\_CARD\_CAPABILITY\_S4MI  
    cyu3cardmgr\_fx3s.h, 196  
CY\_U3P\_SDIO\_CARD\_CAPABILITY\_SBS  
    cyu3cardmgr\_fx3s.h, 197  
CY\_U3P\_SDIO\_CARD\_CAPABILITY\_SDC  
    cyu3cardmgr\_fx3s.h, 197  
CY\_U3P\_SDIO\_CARD\_CAPABILITY\_SMB  
    cyu3cardmgr\_fx3s.h, 197  
CY\_U3P\_SDIO\_CARD\_CAPABILITY\_SRW  
    cyu3cardmgr\_fx3s.h, 197  
CY\_U3P\_SDIO\_CCCR\_Version\_1\_00  
    cyu3cardmgr\_fx3s.h, 197  
CY\_U3P\_SDIO\_CCCR\_Version\_1\_10  
    cyu3cardmgr\_fx3s.h, 197  
CY\_U3P\_SDIO\_CCCR\_Version\_2\_00  
    cyu3cardmgr\_fx3s.h, 197  
CY\_U3P\_SDIO\_CCCR\_Version\_3\_00  
    cyu3cardmgr\_fx3s.h, 197  
CY\_U3P\_SDIO\_CHECK\_INT\_ENABLE\_REG  
    cyu3cardmgr\_fx3s.h, 197  
CY\_U3P\_SDIO\_CIA\_FUNCTION  
    cyu3cardmgr\_fx3s.h, 197  
CY\_U3P\_SDIO\_CISTPL\_END  
    cyu3cardmgr\_fx3s.h, 197  
CY\_U3P\_SDIO\_CISTPL\_FUNCE  
    cyu3cardmgr\_fx3s.h, 197  
CY\_U3P\_SDIO\_CISTPL\_MANFID  
    cyu3cardmgr\_fx3s.h, 198  
CY\_U3P\_SDIO\_CISTPL\_NULL  
    cyu3cardmgr\_fx3s.h, 198  
CY\_U3P\_SDIO\_DISABLE\_INT  
    cyu3cardmgr\_fx3s.h, 198  
CY\_U3P\_SDIO\_EAI  
    cyu3cardmgr\_fx3s.h, 198  
CY\_U3P\_SDIO\_ENABLE\_ASYNC\_INT  
    cyu3cardmgr\_fx3s.h, 198  
CY\_U3P\_SDIO\_ENABLE\_HIGH\_SPEED  
    cyu3cardmgr\_fx3s.h, 198  
CY\_U3P\_SDIO\_ENABLE\_INT  
    cyu3cardmgr\_fx3s.h, 198  
CY\_U3P\_SDIO\_FULL\_SPEED  
    cyu3cardmgr\_fx3s.h, 198  
CY\_U3P\_SDIO\_HIGH\_SPEED

cyu3cardmgr\_fx3s.h, 198  
**CY\_U3P\_SDIO\_INT\_MASTER**  
 cyu3cardmgr\_fx3s.h, 198  
**CY\_U3P\_SDIO\_INTFC\_BT\_A\_AMP**  
 cyu3cardmgr\_fx3s.h, 198  
**CY\_U3P\_SDIO\_INTFC\_BT\_A**  
 cyu3cardmgr\_fx3s.h, 198  
**CY\_U3P\_SDIO\_INTFC\_BT\_B**  
 cyu3cardmgr\_fx3s.h, 199  
**CY\_U3P\_SDIO\_INTFC\_CAM**  
 cyu3cardmgr\_fx3s.h, 199  
**CY\_U3P\_SDIO\_INTFC\_EMBD\_ATA**  
 cyu3cardmgr\_fx3s.h, 199  
**CY\_U3P\_SDIO\_INTFC\_GPS**  
 cyu3cardmgr\_fx3s.h, 199  
**CY\_U3P\_SDIO\_INTFC\_NONE**  
 cyu3cardmgr\_fx3s.h, 199  
**CY\_U3P\_SDIO\_INTFC\_PHS**  
 cyu3cardmgr\_fx3s.h, 199  
**CY\_U3P\_SDIO\_INTFC\_UART**  
 cyu3cardmgr\_fx3s.h, 199  
**CY\_U3P\_SDIO\_INTFC\_WLAN**  
 cyu3cardmgr\_fx3s.h, 199  
**CY\_U3P\_SDIO\_LOW\_SPEED**  
 cyu3cardmgr\_fx3s.h, 199  
**CY\_U3P\_SDIO\_READ\_AFTER\_WRITE**  
 cyu3cardmgr\_fx3s.h, 199  
**CY\_U3P\_SDIO\_REG\_BUS\_INTERFACE\_CONTROL**  
 cyu3cardmgr\_fx3s.h, 199  
**CY\_U3P\_SDIO\_REG\_BUS\_SUSPEND**  
 cyu3cardmgr\_fx3s.h, 199  
**CY\_U3P\_SDIO\_REG\_CARD\_CAPABILITY**  
 cyu3cardmgr\_fx3s.h, 200  
**CY\_U3P\_SDIO\_REG\_CCCR\_HIGH\_SPEED**  
 cyu3cardmgr\_fx3s.h, 200  
**CY\_U3P\_SDIO\_REG\_CCCR\_REVISION**  
 cyu3cardmgr\_fx3s.h, 200  
**CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D0**  
 cyu3cardmgr\_fx3s.h, 200  
**CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D1**  
 cyu3cardmgr\_fx3s.h, 200  
**CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D2**  
 cyu3cardmgr\_fx3s.h, 200  
**CY\_U3P\_SDIO\_REG\_DRIVER\_STRENGTH**  
 cyu3cardmgr\_fx3s.h, 200  
**CY\_U3P\_SDIO\_REG\_EXEC\_FLAGS**  
 cyu3cardmgr\_fx3s.h, 200  
**CY\_U3P\_SDIO\_REG\_FBR\_CIS\_PTR\_D1**  
 cyu3cardmgr\_fx3s.h, 200  
**CY\_U3P\_SDIO\_REG\_FBR\_CIS\_PTR\_D2**  
 cyu3cardmgr\_fx3s.h, 200  
**CY\_U3P\_SDIO\_REG\_FBR\_CIS\_PTR\_DO**  
 cyu3cardmgr\_fx3s.h, 200  
**CY\_U3P\_SDIO\_REG\_FBR\_CSA\_PTR\_D1**  
 cyu3cardmgr\_fx3s.h, 200  
**CY\_U3P\_SDIO\_REG\_FBR\_CSA\_PTR\_D2**  
 cyu3cardmgr\_fx3s.h, 201  
**CY\_U3P\_SDIO\_REG\_FBR\_CSA\_PTR\_DO**  
 cyu3cardmgr\_fx3s.h, 201  
**CY\_U3P\_SDIO\_REG\_FBR\_DATA\_ACCESS\_WINDOW**  
 cyu3cardmgr\_fx3s.h, 201  
**CY\_U3P\_SDIO\_REG\_FBR\_EXT\_INTERFACE\_CODE**  
 cyu3cardmgr\_fx3s.h, 201  
**CY\_U3P\_SDIO\_REG\_FBR\_INTERFACE\_CODE**  
 cyu3cardmgr\_fx3s.h, 201  
**CY\_U3P\_SDIO\_REG\_FBR\_IO\_BLOCKSIZE\_D0**  
 cyu3cardmgr\_fx3s.h, 201  
**CY\_U3P\_SDIO\_REG\_FBR\_IO\_BLOCKSIZE\_D1**  
 cyu3cardmgr\_fx3s.h, 201  
**CY\_U3P\_SDIO\_REG\_FUNCTION\_BLOCKSIZE\_D0**  
 cyu3cardmgr\_fx3s.h, 201  
**CY\_U3P\_SDIO\_REG\_FUNCTION\_BLOCKSIZE\_D1**  
 cyu3cardmgr\_fx3s.h, 201  
**CY\_U3P\_SDIO\_REG\_FUNCTION\_SELECT**  
 cyu3cardmgr\_fx3s.h, 201  
**CY\_U3P\_SDIO\_REG\_INTERRUPT\_EXTENSION**  
 cyu3cardmgr\_fx3s.h, 201  
**CY\_U3P\_SDIO\_REG\_IO\_ABORT**  
 cyu3cardmgr\_fx3s.h, 202  
**CY\_U3P\_SDIO\_REG\_IO\_ENABLE**  
 cyu3cardmgr\_fx3s.h, 202  
**CY\_U3P\_SDIO\_REG\_IO\_INTR\_ENABLE**  
 cyu3cardmgr\_fx3s.h, 202  
**CY\_U3P\_SDIO\_REG\_IO\_INTR\_PENDING**  
 cyu3cardmgr\_fx3s.h, 202  
**CY\_U3P\_SDIO\_REG\_IO\_READY**  
 cyu3cardmgr\_fx3s.h, 202  
**CY\_U3P\_SDIO\_REG\_POWER\_CONTROL**  
 cyu3cardmgr\_fx3s.h, 202  
**CY\_U3P\_SDIO\_REG\_READY\_FLAGS**  
 cyu3cardmgr\_fx3s.h, 202  
**CY\_U3P\_SDIO\_REG\_SD\_SPEC\_REVISION**  
 cyu3cardmgr\_fx3s.h, 202  
**CY\_U3P\_SDIO\_REG\_UHS\_I\_SUPPORT**  
 cyu3cardmgr\_fx3s.h, 202  
**CY\_U3P\_SDIO\_RESET**  
 cyu3cardmgr\_fx3s.h, 202  
**CY\_U3P\_SDIO\_SAI**  
 cyu3cardmgr\_fx3s.h, 202  
**CY\_U3P\_SDIO\_SD\_Version\_1\_00**  
 cyu3cardmgr\_fx3s.h, 202  
**CY\_U3P\_SDIO\_SD\_Version\_1\_10**  
 cyu3cardmgr\_fx3s.h, 203  
**CY\_U3P\_SDIO\_SD\_Version\_2\_00**  
 cyu3cardmgr\_fx3s.h, 203  
**CY\_U3P\_SDIO\_SD\_Version\_3\_00**  
 cyu3cardmgr\_fx3s.h, 203  
**CY\_U3P\_SDIO\_SDDR50\_SPEED**  
 cyu3cardmgr\_fx3s.h, 203  
**CY\_U3P\_SDIO\_SSDDR104\_SPEED**  
 cyu3cardmgr\_fx3s.h, 203  
**CY\_U3P\_SDIO\_SSDDR12\_SPEED**  
 cyu3cardmgr\_fx3s.h, 203

CY\_U3P\_SDIO\_SSDR25\_SPEED  
    cyu3cardmgr\_fx3s.h, 203  
CY\_U3P\_SDIO\_SSDR50\_SPEED  
    cyu3cardmgr\_fx3s.h, 203  
CY\_U3P\_SDIO\_SUPPORT\_HIGH\_SPEED  
    cyu3cardmgr\_fx3s.h, 203  
CY\_U3P\_SDIO\_UHS\_SDDR50  
    cyu3cardmgr\_fx3s.h, 203  
CY\_U3P\_SDIO\_UHS\_SSDR104  
    cyu3cardmgr\_fx3s.h, 203  
CY\_U3P\_SDIO\_UHS\_SSDR50  
    cyu3cardmgr\_fx3s.h, 203  
CY\_U3P\_SDIO\_Version\_1\_00  
    cyu3cardmgr\_fx3s.h, 204  
CY\_U3P\_SDIO\_Version\_1\_10  
    cyu3cardmgr\_fx3s.h, 204  
CY\_U3P\_SDIO\_Version\_1\_20  
    cyu3cardmgr\_fx3s.h, 204  
CY\_U3P\_SDIO\_Version\_2\_00  
    cyu3cardmgr\_fx3s.h, 204  
CY\_U3P\_SDIO\_Version\_3\_00  
    cyu3cardmgr\_fx3s.h, 204  
CY\_U3P\_SIB\_DETECT\_DAT\_3  
    cyu3sib.h, 418  
CY\_U3P\_SIB\_DETECT\_GPIO  
    cyu3sib.h, 418  
CY\_U3P\_SIB\_DETECT\_NONE  
    cyu3sib.h, 418  
CY\_U3P\_SIB\_DEV\_MMC  
    cyu3sib.h, 418  
CY\_U3P\_SIB\_DEV\_NONE  
    cyu3sib.h, 418  
CY\_U3P\_SIB\_DEV\_PARTITION\_0  
    cyu3sibpp.h, 442  
CY\_U3P\_SIB\_DEV\_PARTITION\_1  
    cyu3sibpp.h, 442  
CY\_U3P\_SIB\_DEV\_PARTITION\_2  
    cyu3sibpp.h, 442  
CY\_U3P\_SIB\_DEV\_PARTITION\_3  
    cyu3sibpp.h, 442  
CY\_U3P\_SIB\_DEV\_SDIO\_COMBO  
    cyu3sib.h, 418  
CY\_U3P\_SIB\_DEV\_SDIO  
    cyu3sib.h, 418  
CY\_U3P\_SIB\_DEV\_SD  
    cyu3sib.h, 418  
CY\_U3P\_SIB\_ERASE\_SECURE  
    cyu3sib.h, 418  
CY\_U3P\_SIB\_ERASE\_STANDARD  
    cyu3sib.h, 418  
CY\_U3P\_SIB\_ERASE\_TRIM\_STEP1  
    cyu3sib.h, 419  
CY\_U3P\_SIB\_ERASE\_TRIM\_STEP2  
    cyu3sib.h, 419  
CY\_U3P\_SIB\_EVENT\_ABORT  
    cyu3sib.h, 419  
CY\_U3P\_SIB\_EVENT\_DATA\_ERROR  
    cyu3sib.h, 419  
CY\_U3P\_SIB\_EVENT\_INSERT  
    cyu3sib.h, 419  
CY\_U3P\_SIB\_EVENT\_RELEASE  
    cyu3sib.h, 419  
CY\_U3P\_SIB\_EVENT\_REMOVE  
    cyu3sib.h, 419  
CY\_U3P\_SIB\_EVENT\_SDIO\_INTR  
    cyu3sib.h, 419  
CY\_U3P\_SIB\_EVENT\_XFER\_CPLT  
    cyu3sib.h, 419  
CY\_U3P\_SIB\_EVT\_ABORT  
    cyu3sibpp.h, 441  
CY\_U3P\_SIB\_EVT\_DRVENTRY  
    cyu3sibpp.h, 441  
CY\_U3P\_SIB\_EVT\_PORT\_0  
    cyu3sibpp.h, 441  
CY\_U3P\_SIB\_EVT\_PORT\_1  
    cyu3sibpp.h, 441  
CY\_U3P\_SIB\_EVT\_PORT\_POS  
    cyu3sibpp.h, 441  
CY\_U3P\_SIB\_FREQ\_104MHZ  
    cyu3sib.h, 420  
CY\_U3P\_SIB\_FREQ\_20MHZ  
    cyu3sib.h, 420  
CY\_U3P\_SIB\_FREQ\_26MHZ  
    cyu3sib.h, 420  
CY\_U3P\_SIB\_FREQ\_400KHZ  
    cyu3sib.h, 420  
CY\_U3P\_SIB\_FREQ\_52MHZ  
    cyu3sib.h, 420  
CY\_U3P\_SIB\_IP\_BLOCK\_ID  
    cyu3socket.h, 447  
CY\_U3P\_SIB\_LOCATION\_BOOT1  
    cyu3sib.h, 419  
CY\_U3P\_SIB\_LOCATION\_BOOT2  
    cyu3sib.h, 419  
CY\_U3P\_SIB\_LOCATION\_USER  
    cyu3sib.h, 419  
CY\_U3P\_SIB\_LUN\_BOOT  
    cyu3sib.h, 420  
CY\_U3P\_SIB\_LUN\_DATA  
    cyu3sib.h, 420  
CY\_U3P\_SIB\_LUN\_RSVD  
    cyu3sib.h, 420  
CY\_U3P\_SIB\_NUM\_PARTITIONS  
    cyu3sibpp.h, 442  
CY\_U3P\_SIB\_NUM\_PORTS  
    cyu3sib.h, 420  
CY\_U3P\_SIB\_PORT\_0  
    cyu3sib.h, 420  
CY\_U3P\_SIB\_PORT\_1  
    cyu3sib.h, 420  
CY\_U3P\_SIB\_SD\_CARD\_ID\_400  
    cyu3cardmgr.h, 188  
CY\_U3P\_SIB\_SD\_REG\_CID  
    cyu3cardmgr.h, 189  
CY\_U3P\_SIB\_SD\_REG\_CSD  
    cyu3cardmgr.h, 189

CY\_U3P\_SIB\_SD\_REG\_OCR  
cyu3cardmgr.h, 189

CY\_U3P\_SIB\_SDHC\_DS\_25  
cyu3cardmgr.h, 188

CY\_U3P\_SIB\_SDHC\_HS\_50  
cyu3cardmgr.h, 188

CY\_U3P\_SIB\_SDSC\_25  
cyu3cardmgr.h, 188

CY\_U3P\_SIB\_SOCKET0  
cyu3sibpp.h, 443

CY\_U3P\_SIB\_SOCKET1  
cyu3sibpp.h, 443

CY\_U3P\_SIB\_SOCKET2  
cyu3sibpp.h, 443

CY\_U3P\_SIB\_SOCKET3  
cyu3sibpp.h, 443

CY\_U3P\_SIB\_SOCKET4  
cyu3sibpp.h, 443

CY\_U3P\_SIB\_SOCKET5  
cyu3sibpp.h, 443

CY\_U3P\_SIB\_SOCKET\_0  
cyu3dma.h, 224

CY\_U3P\_SIB\_SOCKET\_1  
cyu3dma.h, 224

CY\_U3P\_SIB\_SOCKET\_2  
cyu3dma.h, 224

CY\_U3P\_SIB\_SOCKET\_3  
cyu3dma.h, 224

CY\_U3P\_SIB\_SOCKET\_4  
cyu3dma.h, 224

CY\_U3P\_SIB\_SOCKET\_5  
cyu3dma.h, 224

CY\_U3P\_SIB\_SOCKET\_6  
cyu3sibpp.h, 443

CY\_U3P\_SIB\_SOCKET\_7  
cyu3sibpp.h, 443

CY\_U3P\_SIB\_STACK\_SIZE  
cyu3sibpp.h, 441

CY\_U3P\_SIB\_THREAD\_PRIORITY  
cyu3sibpp.h, 441

CY\_U3P\_SIB\_TIMER0\_EVT  
cyu3sibpp.h, 441

CY\_U3P\_SIB\_TIMER1\_EVT  
cyu3sibpp.h, 441

CY\_U3P\_SIB\_UHS\_I\_DDR50  
cyu3cardmgr.h, 188

CY\_U3P\_SIB\_UHS\_I\_DS  
cyu3cardmgr.h, 188

CY\_U3P\_SIB\_UHS\_I\_HS  
cyu3cardmgr.h, 188

CY\_U3P\_SIB\_UHS\_I\_SDR104  
cyu3cardmgr.h, 188

CY\_U3P\_SIB\_UHS\_I\_SDR12  
cyu3cardmgr.h, 188

CY\_U3P\_SIB\_UHS\_I\_SDR25  
cyu3cardmgr.h, 188

CY\_U3P\_SIB\_UHS\_I\_SDR50  
cyu3cardmgr.h, 188

CY\_U3P\_SIB\_VOLTAGE\_LOW  
cyu3sib.h, 419

CY\_U3P\_SIB\_VOLTAGE\_NORMAL  
cyu3sib.h, 419

CY\_U3P\_SPI\_ERROR\_NONE  
cyu3spi.h, 456

CY\_U3P\_SPI\_ERROR\_RX\_UNDERFLOW  
cyu3spi.h, 456

CY\_U3P\_SPI\_ERROR\_TX\_OVERFLOW  
cyu3spi.h, 456

CY\_U3P\_SPI\_EVENT\_ERROR  
cyu3spi.h, 456

CY\_U3P\_SPI\_EVENT\_RX\_DONE  
cyu3spi.h, 456

CY\_U3P\_SPI\_EVENT\_TX\_DONE  
cyu3spi.h, 456

CY\_U3P\_SPI\_NUM\_SSN\_CTRL  
cyu3spi.h, 457

CY\_U3P\_SPI\_NUM\_SSN\_LAG\_LEAD  
cyu3spi.h, 457

CY\_U3P\_SPI\_SSN\_CTRL\_FW  
cyu3spi.h, 457

CY\_U3P\_SPI\_SSN\_CTRL\_HW\_CPHA\_BASED  
cyu3spi.h, 457

CY\_U3P\_SPI\_SSN\_CTRL\_HW\_EACH\_WORD  
cyu3spi.h, 457

CY\_U3P\_SPI\_SSN\_CTRL\_HW\_END\_OF\_XFER  
cyu3spi.h, 457

CY\_U3P\_SPI\_SSN\_CTRL\_NONE  
cyu3spi.h, 457

CY\_U3P\_SPI\_SSN\_LAG\_LEAD\_HALF\_CLK  
cyu3spi.h, 457

CY\_U3P\_SPI\_SSN\_LAG\_LEAD\_ONE\_CLK  
cyu3spi.h, 457

CY\_U3P\_SPI\_SSN\_LAG\_LEAD\_ONE\_HALF\_CLK  
cyu3spi.h, 457

CY\_U3P\_SPI\_SSN\_LAG\_LEAD\_ZERO\_CLK  
cyu3spi.h, 457

CY\_U3P\_SPORT\_1BIT  
cyfx3\_api.h, 173

CY\_U3P\_SPORT\_4BIT  
cyfx3\_api.h, 173

CY\_U3P\_SPORT\_8BIT  
cyfx3\_api.h, 173

CY\_U3P\_SPORT\_INACTIVE  
cyfx3\_api.h, 173

CY\_U3P\_SS\_EP\_COMPN\_DESCR  
cyfx3usb.h, 156, 157

CY\_U3P\_SS\_USB\_CAPB\_TYPE  
cyu3usbconst.h, 542, 543

CY\_U3P\_SUCCESS  
cyu3error.h, 254

CY\_U3P\_SUPER\_SPEED  
cyu3usb.h, 512

CY\_U3P\_SYS\_CLK\_BY\_16  
cyu3system.h, 470

CY\_U3P\_SYS\_CLK\_BY\_2

cyu3system.h, 470  
CY\_U3P\_SYS\_CLK\_BY\_4  
    cyu3system.h, 470  
CY\_U3P\_SYS\_CLK  
    cyu3system.h, 470  
CY\_U3P\_THREAD\_ID\_DEBUG  
    cyu3system.h, 470  
CY\_U3P\_THREAD\_ID\_DMA  
    cyu3system.h, 470  
CY\_U3P\_THREAD\_ID\_INT  
    cyu3system.h, 470  
CY\_U3P\_THREAD\_ID\_LIB\_MAX  
    cyu3system.h, 470  
CY\_U3P\_THREAD\_ID\_LPP  
    cyu3system.h, 470  
CY\_U3P\_THREAD\_ID\_PIB  
    cyu3system.h, 470  
CY\_U3P\_THREAD\_ID\_SYSTEM  
    cyu3system.h, 470  
CY\_U3P\_THREAD\_ID\_UIB  
    cyu3system.h, 470  
CY\_U3P\_UART\_BAUDRATE\_100  
    cyu3uart.h, 492  
CY\_U3P\_UART\_BAUDRATE\_10000  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_100000  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_115200  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_1200  
    cyu3uart.h, 492  
CY\_U3P\_UART\_BAUDRATE\_14400  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_153600  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_19200  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_1M  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_200000  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_225000  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_230400  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_2400  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_2M  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_300  
    cyu3uart.h, 492  
CY\_U3P\_UART\_BAUDRATE\_300000  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_38400  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_3M  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_400000  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_460800  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_4800  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_4M608K  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_4M  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_50000  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_500000  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_57600  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_600  
    cyu3uart.h, 492  
CY\_U3P\_UART\_BAUDRATE\_75000  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_750000  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_921600  
    cyu3uart.h, 493  
CY\_U3P\_UART\_BAUDRATE\_9600  
    cyu3uart.h, 493  
CY\_U3P\_UART\_ERROR\_NAK\_BYTE\_0  
    cyu3uart.h, 494  
CY\_U3P\_UART\_ERROR\_RX\_OVERFLOW  
    cyu3uart.h, 494  
CY\_U3P\_UART\_ERROR\_RX\_PARITY\_ERROR  
    cyu3uart.h, 494  
CY\_U3P\_UART\_ERROR\_RX\_UNDERFLOW  
    cyu3uart.h, 494  
CY\_U3P\_UART\_ERROR\_TX\_OVERFLOW  
    cyu3uart.h, 494  
CY\_U3P\_UART\_EVEN\_PARITY  
    cyu3uart.h, 494  
CY\_U3P\_UART\_EVENT\_ERROR  
    cyu3uart.h, 494  
CY\_U3P\_UART\_EVENT\_RX\_DATA  
    cyu3uart.h, 494  
CY\_U3P\_UART\_EVENT\_RX\_DONE  
    cyu3uart.h, 494  
CY\_U3P\_UART\_EVENT\_TX\_DONE  
    cyu3uart.h, 494  
CY\_U3P\_UART\_NO\_PARITY  
    cyu3uart.h, 494  
CY\_U3P\_UART\_NUM\_PARITY  
    cyu3uart.h, 494  
CY\_U3P\_UART\_ODD\_PARITY  
    cyu3uart.h, 494  
CY\_U3P\_UART\_ONE\_STOP\_BIT  
    cyu3uart.h, 495  
CY\_U3P\_UART\_TWO\_STOP\_BIT  
    cyu3uart.h, 495  
CY\_U3P\_UIB\_IP\_BLOCK\_ID  
    cyu3socket.h, 447  
CY\_U3P\_UIB\_LNK\_STATE\_COMP

cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_LNK\_STATE\_POLLING\_ACT**  
 cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_LNK\_STATE\_POLLING\_IDLE**  
 cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_LNK\_STATE\_POLLING\_LFPS**  
 cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_LNK\_STATE\_POLLING\_RxEQ**  
 cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_LNK\_STATE\_RECov\_ACT**  
 cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_LNK\_STATE\_RECov\_CNFG**  
 cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_LNK\_STATE\_RECov\_IDLE**  
 cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_LNK\_STATE\_RXDETECT\_ACT**  
 cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_LNK\_STATE\_RXDETECT\_QUT**  
 cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_LNK\_STATE\_RXDETECT\_RES**  
 cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_LNK\_STATE\_SS DISABLED**  
 cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_LNK\_STATE\_SSINACT\_DET**  
 cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_LNK\_STATE\_SSINACT\_QUT**  
 cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_LNK\_STATE\_U0**  
 cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_LNK\_STATE\_U1**  
 cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_LNK\_STATE\_U2**  
 cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_LNK\_STATE\_U3**  
 cyu3usbconst.h, 544  
**CY\_U3P\_UIB\_SOCKET\_CONS\_0**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_CONS\_1**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_CONS\_10**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_CONS\_11**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_CONS\_12**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_CONS\_13**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_CONS\_14**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_CONS\_15**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_CONS\_2**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_CONS\_3**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_CONS\_4**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_CONS\_5**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_CONS\_6**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_CONS\_7**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_CONS\_8**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_CONS\_9**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_PROD\_0**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_PROD\_1**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_PROD\_10**  
 cyu3dma.h, 225  
**CY\_U3P\_UIB\_SOCKET\_PROD\_11**  
 cyu3dma.h, 225  
**CY\_U3P\_UIB\_SOCKET\_PROD\_12**  
 cyu3dma.h, 225  
**CY\_U3P\_UIB\_SOCKET\_PROD\_13**  
 cyu3dma.h, 225  
**CY\_U3P\_UIB\_SOCKET\_PROD\_14**  
 cyu3dma.h, 225  
**CY\_U3P\_UIB\_SOCKET\_PROD\_15**  
 cyu3dma.h, 225  
**CY\_U3P\_UIB\_SOCKET\_PROD\_2**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_PROD\_3**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_PROD\_4**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_PROD\_5**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_PROD\_6**  
 cyu3dma.h, 224  
**CY\_U3P\_UIB\_SOCKET\_PROD\_7**  
 cyu3dma.h, 225  
**CY\_U3P\_UIB\_SOCKET\_PROD\_8**  
 cyu3dma.h, 225  
**CY\_U3P\_UIB\_SOCKET\_PROD\_9**  
 cyu3dma.h, 225  
**CY\_U3P\_UIBIN\_IP\_BLOCK\_ID**  
 cyu3socket.h, 447  
**CY\_U3P\_USB2\_EXTN\_CAPB\_TYPE**  
 cyu3usbconst.h, 543  
**CY\_U3P\_USB2\_FS\_REMOTE\_WAKE**  
 cyu3usbconst.h, 543  
**CY\_U3P\_USB2\_FS\_TEST\_MODE**  
 cyu3usbconst.h, 543  
**CY\_U3P\_USB2\_OTG\_A\_HNP\_SUPPORT**  
 cyu3usbconst.h, 543  
**CY\_U3P\_USB2\_OTG\_B\_HNP\_ENABLE**  
 cyu3usbconst.h, 543  
**CY\_U3P\_USB2\_TEST\_FORCE\_EN**  
 cyu3usbconst.h, 541  
**CY\_U3P\_USB2\_TEST\_NONE**  
 cyu3usbconst.h, 541  
**CY\_U3P\_USB2\_TEST\_PACKET**

cyu3usbconst.h, 541  
CY\_U3P\_USB2\_TEST\_SE0\_NAK  
    cyu3usbconst.h, 541  
CY\_U3P\_USB2\_TEST\_J  
    cyu3usbconst.h, 541  
CY\_U3P\_USB2\_TEST\_K  
    cyu3usbconst.h, 541  
CY\_U3P\_USB3\_FS\_LTM\_ENABLE  
    cyu3usbconst.h, 543  
CY\_U3P\_USB3\_FS\_U1\_ENABLE  
    cyu3usbconst.h, 543  
CY\_U3P\_USB3\_FS\_U2\_ENABLE  
    cyu3usbconst.h, 543  
CY\_U3P\_USB3\_PACK\_TYPE\_DPH  
    cyu3usbconst.h, 541  
CY\_U3P\_USB3\_PACK\_TYPE\_ITP  
    cyu3usbconst.h, 541  
CY\_U3P\_USB3\_PACK\_TYPE\_LMP  
    cyu3usbconst.h, 541  
CY\_U3P\_USB3\_PACK\_TYPE\_TP  
    cyu3usbconst.h, 541  
CY\_U3P\_USB3\_TP\_SUBTYPE\_ACK  
    cyu3usbconst.h, 542  
CY\_U3P\_USB3\_TP\_SUBTYPE\_ERDY  
    cyu3usbconst.h, 542  
CY\_U3P\_USB3\_TP\_SUBTYPE\_NOTICE  
    cyu3usbconst.h, 542  
CY\_U3P\_USB3\_TP\_SUBTYPE\_NRDY  
    cyu3usbconst.h, 542  
CY\_U3P\_USB3\_TP\_SUBTYPE\_PINGRSP  
    cyu3usbconst.h, 542  
CY\_U3P\_USB3\_TP\_SUBTYPE\_PING  
    cyu3usbconst.h, 542  
CY\_U3P\_USB3\_TP\_SUBTYPE\_RES  
    cyu3usbconst.h, 542  
CY\_U3P\_USB3\_TP\_SUBTYPE\_STALL  
    cyu3usbconst.h, 542  
CY\_U3P\_USB3\_TP\_SUBTYPE\_STATUS  
    cyu3usbconst.h, 542  
CY\_U3P\_USB\_CONFIG\_DESCR  
    cyfx3usb.h, 156  
    cyu3usbconst.h, 542  
CY\_U3P\_USB\_CONFIGURED  
    cyu3usb.h, 511  
CY\_U3P\_USB\_CONNECTED  
    cyu3usb.h, 511  
CY\_U3P\_USB\_DEVICE\_DESCR  
    cyfx3usb.h, 156  
    cyu3usbconst.h, 542  
CY\_U3P\_USB\_DEVQUAL\_DESCR  
    cyfx3usb.h, 156  
    cyu3usbconst.h, 542  
CY\_U3P\_USB\_ENDPNT\_DESCR  
    cyfx3usb.h, 156  
    cyu3usbconst.h, 542  
CY\_U3P\_USB\_EP\_BULK  
    cyu3usbconst.h, 543  
CY\_U3P\_USB\_EP\_CONTROL  
    cyu3usbconst.h, 543  
CY\_U3P\_USB\_EP\_INTR  
    cyu3usbconst.h, 543  
CY\_U3P\_USB\_EP\_ISO  
    cyu3usbconst.h, 543  
CY\_U3P\_USB\_ESTABLISHED  
    cyu3usb.h, 511  
CY\_U3P\_USB\_EVENT\_CONNECT  
    cyu3usb.h, 509  
CY\_U3P\_USB\_EVENT\_DISCONNECT  
    cyu3usb.h, 509  
CY\_U3P\_USB\_EVENT\_EP\_STAT\_CPLT  
    cyu3usb.h, 509  
CY\_U3P\_USB\_EVENT\_EP\_UNDERRUN  
    cyu3usb.h, 510  
CY\_U3P\_USB\_EVENT\_HOST\_CONNECT  
    cyu3usb.h, 510  
CY\_U3P\_USB\_EVENT\_HOST\_DISCONNECT  
    cyu3usb.h, 510  
CY\_U3P\_USB\_EVENT\_LMP\_EXCH\_FAIL  
    cyu3usb.h, 510  
CY\_U3P\_USB\_EVENT\_LNK\_RECOVERY  
    cyu3usb.h, 510  
CY\_U3P\_USB\_EVENT\_OTG\_CHANGE  
    cyu3usb.h, 510  
CY\_U3P\_USB\_EVENT\_OTG\_SRП  
    cyu3usb.h, 510  
CY\_U3P\_USB\_EVENT\_OTG\_VBUS\_CHG  
    cyu3usb.h, 510  
CY\_U3P\_USB\_EVENT\_RESERVED\_1  
    cyu3usb.h, 510  
CY\_U3P\_USB\_EVENT\_RESET  
    cyu3usb.h, 509  
CY\_U3P\_USB\_EVENT\_RESUME  
    cyu3usb.h, 509  
CY\_U3P\_USB\_EVENT\_SET\_SEL  
    cyu3usb.h, 509  
CY\_U3P\_USB\_EVENT\_SETCONF  
    cyu3usb.h, 509  
CY\_U3P\_USB\_EVENT\_SETINTF  
    cyu3usb.h, 509  
CY\_U3P\_USB\_EVENT\_SOF\_ITP  
    cyu3usb.h, 509  
CY\_U3P\_USB\_EVENT\_SPEED  
    cyu3usb.h, 509  
CY\_U3P\_USB\_EVENT\_SS\_COMP\_ENTRY  
    cyu3usb.h, 510  
CY\_U3P\_USB\_EVENT\_SS\_COMP\_EXIT  
    cyu3usb.h, 510  
CY\_U3P\_USB\_EVENT\_SUSPEND  
    cyu3usb.h, 509  
CY\_U3P\_USB\_EVENT\_USB3\_LNKFAIL  
    cyu3usb.h, 510  
CY\_U3P\_USB\_EVENT\_VBUS\_REMOVED  
    cyu3usb.h, 509  
CY\_U3P\_USB\_EVENT\_VBUS\_VALID  
    cyu3usb.h, 509  
CY\_U3P\_USB\_HID\_DESCR

cyfx3usb.h, 156, 157  
 cyu3usbconst.h, 542, 543  
**CY\_U3P\_USB\_HOST\_EPXFER\_NORMAL**  
 cyu3usbhost.h, 550  
**CY\_U3P\_USB\_HOST\_EPXFER\_SETUP\_IN\_DATA**  
 cyu3usbhost.h, 550  
**CY\_U3P\_USB\_HOST\_EPXFER\_SETUP\_NO\_DATA**  
 cyu3usbhost.h, 550  
**CY\_U3P\_USB\_HOST\_EPXFER\_SETUP\_OUT\_DATA**  
 cyu3usbhost.h, 550  
**CY\_U3P\_USB\_HOST\_EVENT\_CONNECT**  
 cyu3usbhost.h, 550  
**CY\_U3P\_USB\_HOST\_EVENT\_DISCONNECT**  
 cyu3usbhost.h, 550  
**CY\_U3P\_USB\_HOST\_FULL\_SPEED**  
 cyu3usbhost.h, 550  
**CY\_U3P\_USB\_HOST\_HIGH\_SPEED**  
 cyu3usbhost.h, 550  
**CY\_U3P\_USB\_HOST\_LOW\_SPEED**  
 cyu3usbhost.h, 550  
**CY\_U3P\_USB\_INACTIVE**  
 cyu3usb.h, 511  
**CY\_U3P\_USB\_INTRFC\_DESCR**  
 cyfx3usb.h, 156  
 cyu3usbconst.h, 542  
**CY\_U3P\_USB\_INTRFC\_POWER\_DESCR**  
 cyfx3usb.h, 156, 157  
 cyu3usbconst.h, 542  
**CY\_U3P\_USB\_MAX\_STATE**  
 cyu3usb.h, 511  
**CY\_U3P\_USB\_MS\_ACTIVE**  
 cyu3usb.h, 511  
**CY\_U3P\_USB\_OTG\_DESCR**  
 cyfx3usb.h, 157  
 cyu3usbconst.h, 542  
**CY\_U3P\_USB\_OTHERSPEED\_DESCR**  
 cyfx3usb.h, 156  
 cyu3usbconst.h, 542  
**CY\_U3P\_USB\_PROP\_DEVADDR**  
 cyu3usb.h, 508  
**CY\_U3P\_USB\_PROP\_FRAMECNT**  
 cyu3usb.h, 508  
**CY\_U3P\_USB\_PROP\_ITPINFO**  
 cyu3usb.h, 508  
**CY\_U3P\_USB\_PROP\_LINKSTATE**  
 cyu3usb.h, 508  
**CY\_U3P\_USB\_PROP\_SYS\_EXIT\_LAT**  
 cyu3usb.h, 508  
**CY\_U3P\_USB\_REPORT\_DESCR**  
 cyfx3usb.h, 156, 157  
 cyu3usbconst.h, 542, 543  
**CY\_U3P\_USB\_SC\_CLEAR\_FEATURE**  
 cyu3usbconst.h, 544  
**CY\_U3P\_USB\_SC\_GET\_CONFIGURATION**  
 cyu3usbconst.h, 544  
**CY\_U3P\_USB\_SC\_GET\_DESCRIPTOR**  
 cyu3usbconst.h, 544  
**CY\_U3P\_USB\_SC\_GET\_INTERFACE**  
 cyu3usbconst.h, 544  
**CY\_U3P\_USB\_SC\_GET\_STATUS**  
 cyu3usbconst.h, 544  
**CY\_U3P\_USB\_SC\_RESERVED**  
 cyu3usbconst.h, 544  
**CY\_U3P\_USB\_SC\_SET\_ADDRESS**  
 cyu3usbconst.h, 544  
**CY\_U3P\_USB\_SC\_SET\_CONFIGURATION**  
 cyu3usbconst.h, 544  
**CY\_U3P\_USB\_SC\_SET\_DESCRIPTOR**  
 cyu3usbconst.h, 544  
**CY\_U3P\_USB\_SC\_SET\_FEATURE**  
 cyu3usbconst.h, 544  
**CY\_U3P\_USB\_SC\_SET\_INTERFACE**  
 cyu3usbconst.h, 544  
**CY\_U3P\_USB\_SC\_SET\_ISOC\_DELAY**  
 cyu3usbconst.h, 544  
**CY\_U3P\_USB\_SC\_SET\_SEL**  
 cyu3usbconst.h, 544  
**CY\_U3P\_USB\_SC\_SYNC\_FRAME**  
 cyu3usbconst.h, 544  
**CY\_U3P\_USB\_SET\_DEVQUAL\_DESCR**  
 cyfx3usb.h, 157, 158  
 cyu3usb.h, 511  
**CY\_U3P\_USB\_SET\_FS\_CONFIG\_DESCR**  
 cyfx3usb.h, 157, 158  
 cyu3usb.h, 511  
**CY\_U3P\_USB\_SET\_HS\_CONFIG\_DESCR**  
 cyfx3usb.h, 157, 158  
 cyu3usb.h, 511  
**CY\_U3P\_USB\_SET\_HS\_DEVICE\_DESCR**  
 cyfx3usb.h, 157, 158  
 cyu3usb.h, 511  
**CY\_U3P\_USB\_SET\_OTG\_DESCR**  
 cyfx3usb.h, 158  
 cyu3usb.h, 511  
**CY\_U3P\_USB\_SET\_SS\_BOS\_DESCR**  
 cyfx3usb.h, 158  
 cyu3usb.h, 511  
**CY\_U3P\_USB\_SET\_SS\_CONFIG\_DESCR**  
 cyfx3usb.h, 158  
 cyu3usb.h, 511  
**CY\_U3P\_USB\_SET\_SS\_DEVICE\_DESCR**  
 cyfx3usb.h, 157, 158  
 cyu3usb.h, 511  
**CY\_U3P\_USB\_SET\_STRING\_DESCR**  
 cyfx3usb.h, 158  
 cyu3usb.h, 511  
**CY\_U3P\_USB\_STARTED**  
 cyu3usb.h, 511  
**CY\_U3P\_USB\_STRING\_DESCR**  
 cyfx3usb.h, 156  
 cyu3usbconst.h, 542  
**CY\_U3P\_USB\_VBUS\_WAIT**  
 cyu3usb.h, 511  
**CY\_U3P\_USB\_WAITING\_FOR\_DESC**  
 cyu3usb.h, 511  
**CY\_U3P\_USBX\_FS\_EP\_HALT**

cyu3usbconst.h, 543  
CY\_U3P\_VIC\_BIAS\_CORRECT\_VECTOR  
  cyu3vic.h, 574  
CY\_U3P\_VIC\_DEBUG\_RX\_VECTOR  
  cyu3vic.h, 574  
CY\_U3P\_VIC\_DEBUG\_TX\_VECTOR  
  cyu3vic.h, 574  
CY\_U3P\_VIC\_GCTL\_CORE\_VECTOR  
  cyu3vic.h, 574  
CY\_U3P\_VIC\_GCTL\_PWR\_VECTOR  
  cyu3vic.h, 575  
CY\_U3P\_VIC\_GPIO\_CORE\_VECTOR  
  cyu3vic.h, 575  
CY\_U3P\_VIC\_I2C\_CORE\_VECTOR  
  cyu3vic.h, 575  
CY\_U3P\_VIC\_I2S\_CORE\_VECTOR  
  cyu3vic.h, 575  
CY\_U3P\_VIC\_LPP\_DMA\_VECTOR  
  cyu3vic.h, 575  
CY\_U3P\_VIC\_NUM\_VECTORS  
  cyu3vic.h, 575  
CY\_U3P\_VIC\_PIB\_CORE\_VECTOR  
  cyu3vic.h, 574  
CY\_U3P\_VIC\_PIB\_DMA\_VECTOR  
  cyu3vic.h, 574  
CY\_U3P\_VIC\_RESERVED\_15\_VECTOR  
  cyu3vic.h, 575  
CY\_U3P\_VIC\_SIB0\_CORE\_VECTOR  
  cyu3vic.h, 574  
CY\_U3P\_VIC\_SIB1\_CORE\_VECTOR  
  cyu3vic.h, 574  
CY\_U3P\_VIC\_SIB\_DMA\_VECTOR  
  cyu3vic.h, 574  
CY\_U3P\_VIC\_SPI\_CORE\_VECTOR  
  cyu3vic.h, 575  
CY\_U3P\_VIC\_SWI\_VECTOR  
  cyu3vic.h, 574  
CY\_U3P\_VIC\_UART\_CORE\_VECTOR  
  cyu3vic.h, 575  
CY\_U3P\_VIC\_UIB\_CONTROL\_VECTOR  
  cyu3vic.h, 574  
CY\_U3P\_VIC\_UIB\_CORE\_VECTOR  
  cyu3vic.h, 574  
CY\_U3P\_VIC\_UIB\_DMA\_VECTOR  
  cyu3vic.h, 574  
CY\_U3P\_VIC\_WDT\_VECTOR  
  cyu3vic.h, 574  
CY\_U3P\_WIRELESS\_USB\_CAPB\_TYPE  
  cyu3usbconst.h, 543  
CYFX3\_GPIF\_COUNTER\_ADDRESS  
  cyfx3pib.h, 131  
CYFX3\_GPIF\_COUNTER\_CONTROL  
  cyfx3pib.h, 131  
CYFX3\_GPIF\_COUNTER\_DATA  
  cyfx3pib.h, 131  
CYFX3\_PMMC\_CMD12\_STOP  
  cyfx3pib.h, 131  
CYFX3\_PMMC\_CMD15\_INACTIVE  
  cyfx3pib.h, 131  
CYFX3\_PMMC\_CMD5\_AWAKE  
  cyfx3pib.h, 131  
CYFX3\_PMMC\_CMD5\_SLEEP  
  cyfx3pib.h, 131  
CYFX3\_PMMC\_CMD6\_SWITCH  
  cyfx3pib.h, 131  
CYFX3\_PMMC\_CMD7\_SELECT  
  cyfx3pib.h, 131  
CYFX3\_PMMC\_GOIDLE\_CMD  
  cyfx3pib.h, 131  
CYFX3\_PMMC\_SOCKET\_NOT\_READY  
  cyfx3pib.h, 131  
CYPART\_LASTDEV  
  cyfx3\_api.h, 172  
CYPART\_USB2011  
  cyfx3\_api.h, 172  
CYPART\_USB2013  
  cyfx3\_api.h, 172  
CYPART\_USB2014  
  cyfx3\_api.h, 172  
CYPART\_USB2023  
  cyfx3\_api.h, 172  
CYPART\_USB2024  
  cyfx3\_api.h, 172  
CYPART\_USB2025  
  cyfx3\_api.h, 172  
CYPART\_USB2031  
  cyfx3\_api.h, 172  
CYPART\_USB2032  
  cyfx3\_api.h, 172  
CYPART\_USB2033  
  cyfx3\_api.h, 172  
CYPART\_USB2034  
  cyfx3\_api.h, 172  
CYPART\_USB2035  
  cyfx3\_api.h, 171  
CYPART\_USB2064  
  cyfx3\_api.h, 172  
CYPART\_USB3011  
  cyfx3\_api.h, 171  
CYPART\_USB3012  
  cyfx3\_api.h, 171  
CYPART\_USB3013  
  cyfx3\_api.h, 171  
CYPART\_USB3014  
  cyfx3\_api.h, 171  
CYPART\_USB3021  
  cyfx3\_api.h, 172  
CYPART\_USB3023  
  cyfx3\_api.h, 172  
CYPART\_USB3024  
  cyfx3\_api.h, 172  
CYPART\_USB3025  
  cyfx3\_api.h, 172  
CYPART\_USB3031  
  cyfx3\_api.h, 171  
CYPART\_USB3032

cyfx3\_api.h, 171  
CYPART\_USB3033  
    cyfx3\_api.h, 171  
CYPART\_USB3034  
    cyfx3\_api.h, 171  
CYPART\_USB3035  
    cyfx3\_api.h, 171

cyu3gpif.h, 262  
CYU3P\_GPIF\_OP\_THR1\_READY  
    cyu3gpif.h, 262  
CYU3P\_GPIF\_OP\_THR2\_PART  
    cyu3gpif.h, 262  
CYU3P\_GPIF\_OP\_THR2\_READY  
    cyu3gpif.h, 262  
CYU3P\_GPIF\_OP\_THR3\_PART  
    cyu3gpif.h, 262  
CYU3P\_GPIF\_OP\_THR3\_READY  
    cyu3gpif.h, 262  
CYU3P\_ICACHE\_EN\_MASK  
    cyu3mmu.h, 342  
CYU3P\_ITCM\_BASE\_ADDR  
    cyu3mmu.h, 342  
CYU3P\_ITCM\_SIZE  
    cyu3mmu.h, 342  
CYU3P\_ITCM\_SZ\_EN  
    cyu3mmu.h, 342  
CYU3P\_MMIO\_BASE\_ADDR  
    cyu3mmu.h, 342  
CYU3P\_MMIO\_SIZE  
    cyu3mmu.h, 342  
CYU3P\_MMU\_EN\_MASK  
    cyu3mmu.h, 342  
CYU3P\_PIB\_CLK\_PHASE\_00  
    cyu3pib.h, 402  
CYU3P\_PIB\_CLK\_PHASE\_01  
    cyu3pib.h, 402  
CYU3P\_PIB\_CLK\_PHASE\_02  
    cyu3pib.h, 402  
CYU3P\_PIB\_CLK\_PHASE\_03  
    cyu3pib.h, 402  
CYU3P\_PIB\_CLK\_PHASE\_04  
    cyu3pib.h, 402  
CYU3P\_PIB\_CLK\_PHASE\_05  
    cyu3pib.h, 402  
CYU3P\_PIB\_CLK\_PHASE\_06  
    cyu3pib.h, 402  
CYU3P\_PIB\_CLK\_PHASE\_07  
    cyu3pib.h, 402  
CYU3P\_PIB\_CLK\_PHASE\_08  
    cyu3pib.h, 402  
CYU3P\_PIB\_CLK\_PHASE\_09  
    cyu3pib.h, 402  
CYU3P\_PIB\_CLK\_PHASE\_10  
    cyu3pib.h, 402  
CYU3P\_PIB\_CLK\_PHASE\_11  
    cyu3pib.h, 402  
CYU3P\_PIB\_CLK\_PHASE\_12  
    cyu3pib.h, 402  
CYU3P\_PIB\_CLK\_PHASE\_13  
    cyu3pib.h, 402  
CYU3P\_PIB\_CLK\_PHASE\_14  
    cyu3pib.h, 402  
CYU3P\_PIB\_CLK\_PHASE\_15  
    cyu3pib.h, 402  
CYU3P\_PIB\_DLL\_MASTER\_SLAVE  
    cyu3pib.h, 403  
CYU3P\_PIB\_DLL\_MASTER  
    cyu3pib.h, 403  
CYU3P\_PIB\_DLL\_SLAVE  
    cyu3pib.h, 403  
CYU3P\_PIB\_ERR\_NONE  
    cyu3pib.h, 403  
CYU3P\_PIB\_ERR\_THR0\_ADAP\_OVERRUN  
    cyu3pib.h, 404  
CYU3P\_PIB\_ERR\_THR0\_ADAP\_UNDERRUN  
    cyu3pib.h, 404  
CYU3P\_PIB\_ERR\_THR0\_DIRECTION  
    cyu3pib.h, 403  
CYU3P\_PIB\_ERR\_THR0\_RD\_BURST  
    cyu3pib.h, 404  
CYU3P\_PIB\_ERR\_THR0\_RD\_FORCE\_END  
    cyu3pib.h, 404  
CYU3P\_PIB\_ERR\_THR0\_RD\_UNDERRUN  
    cyu3pib.h, 403  
CYU3P\_PIB\_ERR\_THR0\_SCK\_INACTIVE  
    cyu3pib.h, 404  
CYU3P\_PIB\_ERR\_THR0\_WR\_OVERRUN  
    cyu3pib.h, 403  
CYU3P\_PIB\_ERR\_THR1\_ADAP\_OVERRUN  
    cyu3pib.h, 404  
CYU3P\_PIB\_ERR\_THR1\_ADAP\_UNDERRUN  
    cyu3pib.h, 404  
CYU3P\_PIB\_ERR\_THR1\_DIRECTION  
    cyu3pib.h, 403  
CYU3P\_PIB\_ERR\_THR1\_RD\_BURST  
    cyu3pib.h, 404  
CYU3P\_PIB\_ERR\_THR1\_RD\_FORCE\_END  
    cyu3pib.h, 404  
CYU3P\_PIB\_ERR\_THR1\_RD\_UNDERRUN  
    cyu3pib.h, 403  
CYU3P\_PIB\_ERR\_THR1\_SCK\_INACTIVE  
    cyu3pib.h, 404  
CYU3P\_PIB\_ERR\_THR1\_WR\_OVERRUN  
    cyu3pib.h, 403  
CYU3P\_PIB\_ERR\_THR2\_ADAP\_OVERRUN  
    cyu3pib.h, 404  
CYU3P\_PIB\_ERR\_THR2\_ADAP\_UNDERRUN  
    cyu3pib.h, 404  
CYU3P\_PIB\_ERR\_THR2\_DIRECTION  
    cyu3pib.h, 403  
CYU3P\_PIB\_ERR\_THR2\_RD\_BURST  
    cyu3pib.h, 404  
CYU3P\_PIB\_ERR\_THR2\_RD\_FORCE\_END  
    cyu3pib.h, 404  
CYU3P\_PIB\_ERR\_THR2\_RD\_UNDERRUN  
    cyu3pib.h, 403  
CYU3P\_PIB\_ERR\_THR2\_SCK\_INACTIVE  
    cyu3pib.h, 404  
CYU3P\_PIB\_ERR\_THR2\_WR\_OVERRUN  
    cyu3pib.h, 403  
CYU3P\_PIB\_ERR\_THR3\_ADAP\_OVERRUN  
    cyu3pib.h, 404  
CYU3P\_PIB\_ERR\_THR3\_ADAP\_UNDERRUN

cyu3pib.h, 404  
**CYU3P\_PIB\_ERR\_THR3\_DIRECTION**  
 cyu3pib.h, 403  
**CYU3P\_PIB\_ERR\_THR3\_RD\_BURST**  
 cyu3pib.h, 404  
**CYU3P\_PIB\_ERR\_THR3\_RD\_FORCE\_END**  
 cyu3pib.h, 404  
**CYU3P\_PIB\_ERR\_THR3\_RD\_UNDERRUN**  
 cyu3pib.h, 403  
**CYU3P\_PIB\_ERR\_THR3\_SCK\_INACTIVE**  
 cyu3pib.h, 404  
**CYU3P\_PIB\_ERR\_THR3\_WR\_OVERRUN**  
 cyu3pib.h, 403  
**CYU3P\_PIB\_INTR\_DLL\_UPDATE**  
 cyu3pib.h, 405  
**CYU3P\_PIB\_INTR\_ERROR**  
 cyu3pib.h, 405  
**CYU3P\_PIB\_INTR\_PPCONFIG**  
 cyu3pib.h, 405  
**CYU3P\_PMMC\_CMD12\_STOP**  
 cyu3pib.h, 405  
**CYU3P\_PMMC\_CMD15\_INACTIVE**  
 cyu3pib.h, 405  
**CYU3P\_PMMC\_CMD5\_AWAKE**  
 cyu3pib.h, 405  
**CYU3P\_PMMC\_CMD5\_SLEEP**  
 cyu3pib.h, 405  
**CYU3P\_PMMC\_CMD6\_SWITCH**  
 cyu3pib.h, 405  
**CYU3P\_PMMC\_CMD7\_SELECT**  
 cyu3pib.h, 405  
**CYU3P\_PMMC\_DIRECT\_READ**  
 cyu3pib.h, 405  
**CYU3P\_PMMC\_DIRECT\_WRITE**  
 cyu3pib.h, 405  
**CYU3P\_PMMC\_GOIDLE\_CMD**  
 cyu3pib.h, 405  
**CYU3P\_PMMC\_SOCKET\_NOT\_READY**  
 cyu3pib.h, 405  
**CYU3P\_ROM\_BASE\_ADDR**  
 cyu3mmu.h, 342  
**CYU3P\_ROM\_SIZE**  
 cyu3mmu.h, 342  
**CYU3P\_SIB\_INT\_READ\_SOCKET**  
 cyu3sibpp.h, 441  
**CYU3P\_SIB\_INT\_WRITE\_SOCKET**  
 cyu3sibpp.h, 441  
**CYU3P\_SYSMEM\_BASE\_ADDR**  
 cyu3mmu.h, 342  
**CYU3P\_SYSMEM\_SIZE**  
 cyu3mmu.h, 342  
**CYU3P\_TCMREG\_ADDRESS\_MASK**  
 cyu3mmu.h, 342  
**CYU3P\_USBEP\_ISOERR\_EVT**  
 cyu3usb.h, 509  
**CYU3P\_USBEP\_NAK\_EVT**  
 cyu3usb.h, 508  
**CYU3P\_USBEP\_SLP\_EVT**  
 cyu3usb.h, 508  
**CYU3P\_USBEP\_SS\_BTERM\_EVT**  
 cyu3usb.h, 509  
**CYU3P\_USBEP\_SS\_RESET\_EVT**  
 cyu3usb.h, 509  
**CYU3P\_USBEP\_SS\_RETRY\_EVT**  
 cyu3usb.h, 509  
**CYU3P\_USBEP\_SS\_SEQERR\_EVT**  
 cyu3usb.h, 509  
**CYU3P\_USBEP\_SS\_STREAMERR\_EVT**  
 cyu3usb.h, 509  
**CYU3P\_USBEP\_ZLP\_EVT**  
 cyu3usb.h, 508  
**CYU3P\_VIC\_BASE\_ADDR**  
 cyu3mmu.h, 343  
**CYU3P\_VIC\_SIZE**  
 cyu3mmu.h, 343  
**cardCapability**  
 CyU3PSdioCardRegs, 79  
**cardDetType**  
 CyU3PSibIntfParams, 85  
**cardInitDelay**  
 CyU3PSibIntfParams, 85  
**cardRCA**  
 CyU3PCardCtxt, 40  
**cardSpeed**  
 CyU3PCardCtxt, 40  
 CyU3PSdioCardRegs, 79  
**cardType**  
 CyU3PCardCtxt, 40  
 CyU3PSibDevInfo, 82  
**cardVer**  
 CyU3PCardCtxt, 40  
**cb**  
 CyU3PDmaChannel, 45  
 CyU3PDmaChannelConfig\_t, 49  
 CyU3PDmaMultiChannel, 53  
 CyU3PDmaMultiChannelConfig\_t, 56  
 CyU3POtgConfig\_t, 77  
**ccc**  
 CyU3PCardCtxt, 40  
 CyU3PSibDevInfo, 82  
**chain**  
 CyFx3BootDmaDescriptor\_t, 25  
 CyU3PDmaDescriptor\_t, 51  
**chargerMode**  
 CyU3POtgConfig\_t, 77  
**cidRegData**  
 CyU3PCardCtxt, 40  
**clkDiv**  
 CyFx3BootPibClock\_t, 33  
 CyU3PPibClock\_t, 78  
**clkRate**  
 CyU3PCardCtxt, 40  
 CyU3PSibDevInfo, 82  
**clkSrc**  
 CyFx3BootPibClock\_t, 33  
 CyU3PGpioClock\_t, 64

CyU3PPibClock\_t, 78  
CyU3PSysClockConfig\_t, 90  
clock  
    CyFx3BootSpiConfig\_t, 34  
    CyU3PSpiConfig\_t, 88  
commitConsIndex  
    CyU3PDmaChannel, 45  
    CyU3PDmaMultiChannel, 53  
commitProdIndex  
    CyU3PDmaChannel, 45  
    CyU3PDmaMultiChannel, 53  
consDisabled  
    CyU3PDmaMultiChannel, 53  
consHeader  
    CyU3PDmaChannel, 46  
    CyU3PDmaChannelConfig\_t, 49  
    CyU3PDmaMultiChannel, 53  
    CyU3PDmaMultiChannelConfig\_t, 56  
consSckId  
    CyU3PDmaChannel, 46  
    CyU3PDmaChannelConfig\_t, 49  
    CyU3PDmaMultiChannel, 53  
    CyU3PDmaMultiChannelConfig\_t, 56  
consSusp  
    CyU3PDmaChannel, 46  
    CyU3PDmaMultiChannel, 53  
count  
    CyU3PDmaBuffer\_t, 43  
    CyU3PDmaChannel, 46  
    CyU3PDmaChannelConfig\_t, 49  
    CyU3PDmaMultiChannel, 53  
    CyU3PDmaMultiChannelConfig\_t, 57  
cpha  
    CyFx3BootSpiConfig\_t, 34  
    CyU3PSpiConfig\_t, 88  
cpol  
    CyFx3BootSpiConfig\_t, 34  
    CyU3PSpiConfig\_t, 88  
cpuClkDiv  
    CyU3PSysClockConfig\_t, 90  
crcErrCnt  
    CyU3PMipicsiErrorCounts\_t, 76  
csdRegData  
    CyU3PCardCtxt, 40  
csiRxClkDiv  
    CyU3PMipicsiCfg\_t, 74  
ctlErrCnt  
    CyU3PMipicsiErrorCounts\_t, 76  
ctrlMask  
    CyFx3BootI2cPreamble\_t, 31  
    CyU3PI2cPreamble\_t, 70  
currentConsIndex  
    CyU3PDmaChannel, 46  
    CyU3PDmaMultiChannel, 53  
currentProdIndex  
    CyU3PDmaChannel, 46  
    CyU3PDmaMultiChannel, 53  
CyBool\_t  
    cyu3types.h, 488  
CyFalse  
    cyu3types.h, 488  
CyFx3BootBusyWait  
    cyfx3utils.h, 165  
CyFx3BootDeviceConfigureIMatrix  
    cyfx3device.h, 100  
CyFx3BootDeviceInit  
    cyfx3device.h, 101  
CyFx3BootDeviceReset  
    cyfx3device.h, 101  
CyFx3BootDmaCallback\_t  
    cyfx3dma.h, 107  
CyFx3BootDmaClearSockInterrupts  
    cyfx3dma.h, 110  
CyFx3BootDmaDescriptor\_t, 25  
    buffer, 25  
    chain, 25  
    cyfx3dma.h, 107  
    size, 25  
    sync, 25  
CyFx3BootDmaDisableSocket  
    cyfx3dma.h, 110  
CyFx3BootDmaEnableSocket  
    cyfx3dma.h, 110  
CyFx3BootDmaGetDscrConfig  
    cyfx3dma.h, 111  
CyFx3BootDmaGetSockInterrupts  
    cyfx3dma.h, 112  
CyFx3BootDmaGetSocketConfig  
    cyfx3dma.h, 111  
CyFx3BootDmaRegisterCallback  
    cyfx3dma.h, 112  
CyFx3BootDmaSendSocketEvent  
    cyfx3dma.h, 112  
CyFx3BootDmaSetDscrConfig  
    cyfx3dma.h, 113  
CyFx3BootDmaSetSocketConfig  
    cyfx3dma.h, 113  
CyFx3BootDmaSockId\_t  
    cyfx3dma.h, 107, 108  
CyFx3BootDmaSockRegs\_t, 27  
    actBuffer, 27  
    actChain, 27  
    actSize, 27  
    actSync, 27  
    cyfx3dma.h, 107  
    dscrChain, 28  
    intr, 28  
    intrMask, 28  
    sckEvent, 28  
    status, 28  
    unused19, 28  
    unused2, 28  
    xferCount, 28  
    xferSize, 28  
CyFx3BootDmaSocket\_t, 26  
    cyfx3dma.h, 107

dscrChain, 26  
 intr, 26  
 intrMask, 26  
 status, 26  
 xferCount, 26  
 xferSize, 26  
**CyFx3BootDmaWrapSocket**  
 cyfx3dma.h, 114  
**CyFx3BootErrorCode\_t**  
 cyfx3error.h, 115  
**CyFx3BootGetPartNumber**  
 cyfx3device.h, 101  
**CyFx3BootGpifControlSWInput**  
 cyfx3pib.h, 131  
**CyFx3BootGpifDisable**  
 cyfx3pib.h, 132  
**CyFx3BootGpifGetState**  
 cyfx3pib.h, 132  
**CyFx3BootGpifInitCounter**  
 cyfx3pib.h, 132  
**CyFx3BootGpifIntrCb\_t**  
 cyfx3pib.h, 129  
**CyFx3BootGpifLoad**  
 cyfx3pib.h, 133  
**CyFx3BootGpifRegisterCallback**  
 cyfx3pib.h, 133  
**CyFx3BootGpifSMStart**  
 cyfx3pib.h, 133  
**CyFx3BootGpifSMSwitch**  
 cyfx3pib.h, 134  
**CyFx3BootGpifSocketConfigure**  
 cyfx3pib.h, 134  
**CyFx3BootGpioDelInit**  
 cyfx3gpio.h, 118  
**CyFx3BootGpioDisable**  
 cyfx3gpio.h, 119  
**CyFx3BootGpioGetValue**  
 cyfx3gpio.h, 119  
**CyFx3BootGpioInit**  
 cyfx3gpio.h, 119  
**CyFx3BootGpioIntrMode\_t**  
 cyfx3gpio.h, 117, 118  
**CyFx3BootGpiolMode\_t**  
 cyfx3gpio.h, 117, 118  
**CyFx3BootGpioOverride**  
 cyfx3device.h, 101  
**CyFx3BootGpioRestore**  
 cyfx3device.h, 102  
**CyFx3BootGpioSetIoMode**  
 cyfx3gpio.h, 120  
**CyFx3BootGpioSetSimpleConfig**  
 cyfx3gpio.h, 120  
**CyFx3BootGpioSetValue**  
 cyfx3gpio.h, 120  
**CyFx3BootGpioSimpleConfig\_t**, 28  
 cyfx3gpio.h, 117  
 driveHighEn, 29  
 driveLowEn, 29  
 inputEn, 29  
 intrMode, 29  
 outValue, 29  
**CyFx3BootI2cConfig\_t**, 29  
 bitRate, 30  
 busTimeout, 30  
 cyfx3i2c.h, 122  
 dmaTimeout, 30  
 isDma, 30  
**CyFx3BootI2cDeInit**  
 cyfx3i2c.h, 123  
**CyFx3BootI2cDmaXferData**  
 cyfx3i2c.h, 123  
**CyFx3BootI2cInit**  
 cyfx3i2c.h, 124  
**CyFx3BootI2cPreamble\_t**, 30  
 buffer, 31  
 ctrlMask, 31  
 cyfx3i2c.h, 122  
 length, 31  
**CyFx3BootI2cReceiveBytes**  
 cyfx3i2c.h, 124  
**CyFx3BootI2cSendCommand**  
 cyfx3i2c.h, 125  
**CyFx3BootI2cSetConfig**  
 cyfx3i2c.h, 125  
**CyFx3BootI2cSetTimeout**  
 cyfx3i2c.h, 126  
**CyFx3BootI2cTransmitBytes**  
 cyfx3i2c.h, 126  
**CyFx3BootI2cWaitForAck**  
 cyfx3i2c.h, 126  
**CyFx3BootIoMatrixConfig\_t**, 32  
 cyfx3device.h, 99  
 gpioSimpleEn, 32  
 isDQ32Bit, 32  
 useI2C, 32  
 useI2S, 32  
 useSpi, 33  
 useUart, 33  
**CyFx3BootJumpToProgramEntry**  
 cyfx3device.h, 102  
**CyFx3BootMemcpy**  
 cyfx3utils.h, 165  
**CyFx3BootMemSet**  
 cyfx3utils.h, 166  
**CyFx3BootPMMCEvent\_t**  
 cyfx3pib.h, 129, 131  
**CyFx3BootPMMCIIntrCb\_t**  
 cyfx3pib.h, 129  
**CyFx3BootPibClock\_t**, 33  
 clkDiv, 33  
 clkSrc, 33  
 cyfx3pib.h, 129  
 isDIIEnable, 33  
 isHalfDiv, 34  
**CyFx3BootPibDeinit**  
 cyfx3pib.h, 135

CyFx3BootPibDmaXferData  
    cyfx3pib.h, 135  
CyFx3BootPibHandleEvents  
    cyfx3pib.h, 136  
CyFx3BootPibInit  
    cyfx3pib.h, 136  
CyFx3BootPibRegisterMmcCallback  
    cyfx3pib.h, 136  
CyFx3BootRegisterSetupCallback  
    cyfx3usb.h, 158  
CyFx3BootRetainGpioState  
    cyfx3device.h, 103  
CyFx3BootSNPrintf  
    cyfx3utils.h, 166  
CyFx3BootSpiConfig\_t, 34  
    clock, 34  
    cpha, 34  
    cpol, 34  
    cyfx3spi.h, 138  
    isLsbFirst, 35  
    lagTime, 35  
    leadTime, 35  
    ssnCtrl, 35  
    ssnPol, 35  
    wordLen, 35  
CyFx3BootSpiDeInit  
    cyfx3spi.h, 140  
CyFx3BootSpiDisableBlockXfer  
    cyfx3spi.h, 140  
CyFx3BootSpiDmaXferData  
    cyfx3spi.h, 140  
CyFx3BootSpiInit  
    cyfx3spi.h, 141  
CyFx3BootSpiReceiveWords  
    cyfx3spi.h, 141  
CyFx3BootSpiSetBlockXfer  
    cyfx3spi.h, 142  
CyFx3BootSpiSetConfig  
    cyfx3spi.h, 142  
CyFx3BootSpiSetSsnLine  
    cyfx3spi.h, 143  
CyFx3BootSpiSetTimeout  
    cyfx3spi.h, 143  
CyFx3BootSpiSsnCtrl\_t  
    cyfx3spi.h, 138, 139  
CyFx3BootSpiSsnLagLead\_t  
    cyfx3spi.h, 139  
CyFx3BootSpiTransmitWords  
    cyfx3spi.h, 143  
CyFx3BootSysClockSrc\_t  
    cyfx3device.h, 99, 100  
CyFx3BootUSBEVENTCb\_t  
    cyfx3usb.h, 154  
CyFx3BootUSBSetupCb\_t  
    cyfx3usb.h, 154  
CyFx3BootUartBaudrate\_t  
    cyfx3uart.h, 145, 146  
CyFx3BootUartConfig\_t, 35  
    baudRate, 36  
    cyfx3uart.h, 145  
    flowCtrl, 36  
    isDma, 36  
    parity, 36  
    rxEnable, 36  
    stopBit, 36  
    txEnable, 36  
CyFx3BootUartDeInit  
    cyfx3uart.h, 147  
CyFx3BootUartDmaXferData  
    cyfx3uart.h, 147  
CyFx3BootUartInit  
    cyfx3uart.h, 148  
CyFx3BootUartParity\_t  
    cyfx3uart.h, 146, 147  
CyFx3BootUartPrintMessage  
    cyfx3uart.h, 148  
CyFx3BootUartReceiveBytes  
    cyfx3uart.h, 149  
CyFx3BootUartRxSetBlockXfer  
    cyfx3uart.h, 149  
CyFx3BootUartSetConfig  
    cyfx3uart.h, 149  
CyFx3BootUartSetTimeout  
    cyfx3uart.h, 150  
CyFx3BootUartStopBit\_t  
    cyfx3uart.h, 146, 147  
CyFx3BootUartTransmitBytes  
    cyfx3uart.h, 150  
CyFx3BootUartTxSetBlockXfer  
    cyfx3uart.h, 150  
CyFx3BootUsbAckSetup  
    cyfx3usb.h, 158  
CyFx3BootUsbCheckUsb3Disconnect  
    cyfx3usb.h, 158  
CyFx3BootUsbConnect  
    cyfx3usb.h, 159  
CyFx3BootUsbDmaXferData  
    cyfx3usb.h, 159  
CyFx3BootUsbEp0Pkt\_t, 37  
    blIdx0, 37  
    blIdx1, 37  
    bReq, 37  
    bVal0, 37  
    bVal1, 37  
    bmReqType, 37  
    cyfx3usb.h, 154  
    pData, 37  
    wLen, 38  
CyFx3BootUsbEp0StatusCheck  
    cyfx3usb.h, 159  
CyFx3BootUsbEpConfig\_t, 38  
    burstLen, 38  
    cyfx3usb.h, 154  
    enable, 38  
    epType, 38  
    isoPkts, 38

pcktSize, 38  
streams, 39  
CyFx3BootUsbEpType\_t  
  cyfx3usb.h, 155  
CyFx3BootUsbEventType\_t  
  cyfx3usb.h, 155  
CyFx3BootUsbGetDesc  
  cyfx3usb.h, 160  
CyFx3BootUsbGetEpCfg  
  cyfx3usb.h, 160  
CyFx3BootUsbGetLinkPowerState  
  cyfx3usb.h, 160  
CyFx3BootUsbGetSpeed  
  cyfx3usb.h, 160  
CyFx3BootUsbHandleEvents  
  cyfx3usb.h, 161  
CyFx3BootUsbLPMDisable  
  cyfx3usb.h, 161  
CyFx3BootUsbLPMEnable  
  cyfx3usb.h, 161  
CyFx3BootUsbSendCompliancePatterns  
  cyfx3usb.h, 161  
CyFx3BootUsbSetClrFeature  
  cyfx3usb.h, 162  
CyFx3BootUsbSetDesc  
  cyfx3usb.h, 162  
CyFx3BootUsbSetEpConfig  
  cyfx3usb.h, 162  
CyFx3BootUsbSetLinkPowerState  
  cyfx3usb.h, 163  
CyFx3BootUsbSigResume  
  cyfx3usb.h, 163  
CyFx3BootUsbSpeed\_t  
  cyfx3usb.h, 156  
CyFx3BootUsbStall  
  cyfx3usb.h, 163  
CyFx3BootUsbStart  
  cyfx3usb.h, 164  
CyFx3BootUsbVBattEnable  
  cyfx3usb.h, 164  
CyFx3BootWatchdogClear  
  cyfx3device.h, 103  
CyFx3BootWatchdogConfigure  
  cyfx3device.h, 103  
CyFx3BusyWait  
  cyfx3\_api.h, 173  
CyFx3DevClearSwInterrupt  
  cyfx3\_api.h, 173  
CyFx3DevGetMipiLaneCount  
  cyfx3\_api.h, 173  
CyFx3DevIOConfigure  
  cyfx3\_api.h, 174  
CyFx3DevIODisableSib0  
  cyfx3\_api.h, 174  
CyFx3DevIODisableSib1  
  cyfx3\_api.h, 174  
CyFx3DevIOIsGpio  
  cyfx3\_api.h, 174  
CyFx3DevIOIsI2cConfigured  
  cyfx3\_api.h, 175  
CyFx3DevIOIsI2sConfigured  
  cyfx3\_api.h, 175  
CyFx3DevIOIsSib0Configured  
  cyfx3\_api.h, 175  
CyFx3DevIOIsSib1BitWide  
  cyfx3\_api.h, 175  
CyFx3DevIOIsSib1Configured  
  cyfx3\_api.h, 176  
CyFx3DevIOIsSib8BitWide  
  cyfx3\_api.h, 176  
CyFx3DevIOIsSpiConfigured  
  cyfx3\_api.h, 176  
CyFx3DevIOIsUartConfigured  
  cyfx3\_api.h, 176  
CyFx3DevIOSelectGpio  
  cyfx3\_api.h, 176  
CyFx3DevIdentifyPart  
  cyfx3\_api.h, 173  
CyFx3DevInitPageTables  
  cyfx3\_api.h, 174  
CyFx3DevIsGpif32Supported  
  cyfx3\_api.h, 177  
CyFx3DevIsGpifSupported  
  cyfx3\_api.h, 177  
CyFx3DevIsI2sSupported  
  cyfx3\_api.h, 177  
CyFx3DevIsMipicsiSupported  
  cyfx3\_api.h, 177  
CyFx3DevIsOtgSupported  
  cyfx3\_api.h, 177  
CyFx3DevIsRam512Supported  
  cyfx3\_api.h, 178  
CyFx3DevIsSib0Supported  
  cyfx3\_api.h, 178  
CyFx3DevIsSib1Supported  
  cyfx3\_api.h, 178  
CyFx3DevIsUsb3Supported  
  cyfx3\_api.h, 178  
CyFx3GpifCounterType  
  cyfx3pib.h, 130, 131  
CyFx3LppIsOn  
  cyfx3\_api.h, 178  
CyFx3PartNumber\_t  
  cyfx3device.h, 100  
CyFx3PibDIIEnable  
  cyfx3\_api.h, 179  
CyFx3PibGetDIIStatus  
  cyfx3\_api.h, 179  
CyFx3PibIsOn  
  cyfx3\_api.h, 179  
CyFx3PibPowerOff  
  cyfx3\_api.h, 179  
CyFx3PibPowerOn  
  cyfx3\_api.h, 179  
CyFx3SetCpuFreq  
  cyfx3\_api.h, 180

CyFx3SibPowerOff  
    cyfx3\_api.h, 180  
CyFx3SibPowerOn  
    cyfx3\_api.h, 180  
CyFx3Usb2PhySetup  
    cyfx3\_api.h, 180  
CyFx3Usb3LnkRelaxHpTimeout  
    cyfx3\_api.h, 181  
CyFx3Usb3LnkSetup  
    cyfx3\_api.h, 181  
CyFx3Usb3SendTP  
    cyfx3\_api.h, 181  
CyFx3UsbDmaPrefetchEnable  
    cyfx3\_api.h, 181  
CyFx3UsbIsOn  
    cyfx3\_api.h, 181  
CyFx3UsbPowerOn  
    cyfx3\_api.h, 182  
CyFx3UsbWritePhyReg  
    cyfx3\_api.h, 182  
CyFxApplicationDefine  
    cyu3system.h, 471  
CyTrue  
    cyu3types.h, 488  
CyU3PAbortHandler  
    cyu3os.h, 361  
CyU3PApplicationDefine  
    cyu3os.h, 361  
CyU3PBlockAlloc  
    cyu3os.h, 361  
CyU3PBlockFree  
    cyu3os.h, 362  
CyU3PBlockId\_t  
    cyu3socket.h, 446, 447  
CyU3PBlockPool  
    cyu3os.h, 357  
CyU3PBlockPoolCreate  
    cyu3os.h, 362  
CyU3PBlockPoolDestroy  
    cyu3os.h, 362  
CyU3PBufCorruptionCheck  
    cyu3os.h, 363  
CyU3PBufEnableChecks  
    cyu3os.h, 363  
CyU3PBufGetActiveList  
    cyu3os.h, 364  
CyU3PBufGetCounts  
    cyu3os.h, 364  
CyU3PBusyWait  
    cyu3utils.h, 571  
CyU3PByteAlloc  
    cyu3os.h, 364  
CyU3PByteFree  
    cyu3os.h, 364  
CyU3PBytePool  
    cyu3os.h, 358  
CyU3PBytePoolCreate  
    cyu3os.h, 365  
CyU3PBytePoolDestroy  
    cyu3os.h, 365  
CyU3PCardBusWidth\_t  
    cyu3cardmgr.h, 188  
CyU3PCardCtxt, 39  
    blkLen, 39  
    busWidth, 39  
    cardRCA, 40  
    cardSpeed, 40  
    cardType, 40  
    cardVer, 40  
    ccc, 40  
    cidRegData, 40  
    clkRate, 40  
    csdRegData, 40  
    dataTimeOut, 40  
    ddrMode, 40  
    eraseSize, 40  
    highCapacity, 40  
    locked, 41  
    numBlks, 41  
    ocrRegData, 41  
    opVoltage, 41  
    removable, 41  
    uhslOpMode, 41  
    writeable, 41  
CyU3PCardCtxt\_t  
    cyu3cardmgr.h, 188  
CyU3PCardMgrCheckStatus  
    cyu3cardmgr.h, 189  
CyU3PCardMgrCompleteSDInit  
    cyu3cardmgr.h, 190  
CyU3PCardMgrContinueReadWrite  
    cyu3cardmgr.h, 190  
CyU3PCardMgrDelInit  
    cyu3cardmgr.h, 190  
CyU3PCardMgrGetCSD  
    cyu3cardmgr.h, 191  
CyU3PCardMgrInit  
    cyu3cardmgr.h, 191  
CyU3PCardMgrReadExtCsd  
    cyu3cardmgr.h, 191  
CyU3PCardMgrSendCmd  
    cyu3cardmgr.h, 192  
CyU3PCardMgrSetClockFreq  
    cyu3cardmgr.h, 192  
CyU3PCardMgrSetupRead  
    cyu3cardmgr.h, 192  
CyU3PCardMgrSetupWrite  
    cyu3cardmgr.h, 193  
CyU3PCardMgrStopTransfer  
    cyu3cardmgr.h, 193  
CyU3PCardMgrWaitForInterrupt  
    cyu3cardmgr.h, 193  
CyU3PCardOpMode\_t  
    cyu3cardmgr.h, 188  
CyU3PComputeChecksum  
    cyu3utils.h, 571

CyU3PConnectState  
     cyu3usb.h, 512

CyU3PCx3DeviceReset  
     cyu3mipicsi.h, 331

CyU3PDebugDelInit  
     cyu3system.h, 471

CyU3PDebugDisable  
     cyu3system.h, 471

CyU3PDebugEnable  
     cyu3system.h, 471

CyU3PDebugInit  
     cyu3system.h, 472

CyU3PDebugLog  
     cyu3system.h, 472

CyU3PDebugLog\_t, 41  
     cyu3system.h, 467  
     msg, 42  
     param, 42  
     priority, 42  
     threadId, 42

CyU3PDebugLogClear  
     cyu3system.h, 473

CyU3PDebugLogFlush  
     cyu3system.h, 473

CyU3PDebugPreamble  
     cyu3system.h, 473

CyU3PDebugPrint  
     cyu3system.h, 474

CyU3PDebugSetTimeout  
     cyu3system.h, 474

CyU3PDebugSetTraceLevel  
     cyu3system.h, 475

CyU3PDebugStringPrint  
     cyu3system.h, 475

CyU3PDebugSysMemDelInit  
     cyu3system.h, 476

CyU3PDebugSysMemInit  
     cyu3system.h, 476

CyU3PDeviceCacheControl  
     cyu3system.h, 476

CyU3PDeviceConfigureOMatrix  
     cyu3system.h, 477

CyU3PDeviceGetCpuLoad  
     cyu3system.h, 478

CyU3PDeviceGetDriverLoad  
     cyu3system.h, 478

CyU3PDeviceGetPartNumber  
     cyu3system.h, 478

CyU3PDeviceGetSysClkFreq  
     cyu3system.h, 479

CyU3PDeviceGetThreadLoad  
     cyu3system.h, 479

CyU3PDeviceGpioOverride  
     cyu3system.h, 479

CyU3PDeviceGpioRestore  
     cyu3system.h, 480

CyU3PDeviceInit  
     cyu3system.h, 480

CyU3PDeviceReset  
     cyu3system.h, 480

CyU3PDmaBuffer\_t, 42  
     buffer, 43  
     count, 43  
     cyu3dma.h, 216  
     size, 43  
     status, 43

CyU3PDmaBufferAlloc  
     cyu3os.h, 366

CyU3PDmaBufferDelInit  
     cyu3os.h, 366

CyU3PDmaBufferFree  
     cyu3os.h, 366

CyU3PDmaBufferInit  
     cyu3os.h, 367

CyU3PDmaCBInput\_t, 43  
     buffer\_p, 44  
     cyu3dma.h, 217

CyU3PDmaCallback\_t  
     cyu3dma.h, 216

CyU3PDmaCbType\_t  
     cyu3dma.h, 217, 220

CyU3PDmaChannel, 44  
     activeConsIndex, 45  
     activeProdIndex, 45  
     cb, 45  
     commitConsIndex, 45  
     commitProdIndex, 45  
     consHeader, 46  
     consSckId, 46  
     consSusp, 46  
     count, 46  
     currentConsIndex, 46  
     currentProdIndex, 46  
     discardCount, 46  
     dmaMode, 46  
     endSig, 46  
     firstConsIndex, 46  
     firstProdIndex, 46  
     flags, 46  
     isDmaHandleDCache, 47  
     lock, 47  
     notification, 47  
     overrideDscrIndex, 47  
     prodAvailCount, 47  
     prodFooter, 47  
     prodHeader, 47  
     prodSckId, 47  
     prodSusp, 47  
     size, 47  
     startSig, 47  
     state, 47  
     type, 48  
     usbConsSusp, 48  
     xferSize, 48

CyU3PDmaChannelAbort  
     cyu3dma.h, 226

CyU3PDmaChannelCacheControl  
  cyu3dma.h, 227

CyU3PDmaChannelCommitBuffer  
  cyu3dma.h, 227

CyU3PDmaChannelConfig\_t, 48

- cb, 49
- consHeader, 49
- consSckId, 49
- count, 49
- cyu3dma.h, 217
- dmaMode, 49
- notification, 49
- prodAvailCount, 49
- prodFooter, 50
- prodHeader, 50
- prodSckId, 50
- size, 50

CyU3PDmaChannelCreate  
  cyu3dma.h, 228

CyU3PDmaChannelDestroy  
  cyu3dma.h, 229

CyU3PDmaChannelDiscardBuffer  
  cyu3dma.h, 229

CyU3PDmaChannelGetBuffer  
  cyu3dma.h, 230

CyU3PDmaChannelGetHandle  
  cyu3dma.h, 231

CyU3PDmaChannelGetStatus  
  cyu3dma.h, 231

CyU3PDmaChannelIsValid  
  cyu3dma.h, 232

CyU3PDmaChannelReset  
  cyu3dma.h, 232

CyU3PDmaChannelResume  
  cyu3dma.h, 232

CyU3PDmaChannelResumeUsbConsumer  
  cyu3dma.h, 233

CyU3PDmaChannelSetSuspend  
  cyu3dma.h, 233

CyU3PDmaChannelSetWrapUp  
  cyu3dma.h, 235

CyU3PDmaChannelSetXfer  
  cyu3dma.h, 236

CyU3PDmaChannelSetupRecvBuffer  
  cyu3dma.h, 234

CyU3PDmaChannelSetupSendBuffer  
  cyu3dma.h, 235

CyU3PDmaChannelSuspendUsbConsumer  
  cyu3dma.h, 236

CyU3PDmaChannelUpdateMode  
  cyu3dma.h, 237

CyU3PDmaChannelWaitForCompletion  
  cyu3dma.h, 237

CyU3PDmaChannelWaitForRecvBuffer  
  cyu3dma.h, 238

CyU3PDmaDescriptor\_t, 50

- buffer, 51
- chain, 51

  cyu3descriptor.h, 206

  size, 51

  sync, 51

CyU3PDmaDscrChainCreate  
  cyu3descriptor.h, 206

CyU3PDmaDscrChainDestroy  
  cyu3descriptor.h, 207

CyU3PDmaDscrGet  
  cyu3descriptor.h, 207

CyU3PDmaDscrGetConfig  
  cyu3descriptor.h, 207

CyU3PDmaDscrGetFreeCount  
  cyu3descriptor.h, 208

CyU3PDmaDscrListCreate  
  cyu3descriptor.h, 208

CyU3PDmaDscrListDestroy  
  cyu3descriptor.h, 208

CyU3PDmaDscrPut  
  cyu3descriptor.h, 209

CyU3PDmaDscrSetConfig  
  cyu3descriptor.h, 209

CyU3PDmaEnableMulticast  
  cyu3dma.h, 239

CyU3PDmaGetSckId  
  cyu3socket.h, 446

CyU3PDmaMode\_t  
  cyu3dma.h, 218, 221

CyU3PDmaMultiCallback\_t  
  cyu3dma.h, 218

CyU3PDmaMultiChannel, 51

- activeConsIndex, 53
- activeProdIndex, 53
- bufferCount, 53
- cb, 53
- commitConsIndex, 53
- commitProdIndex, 53
- consDisabled, 53
- consHeader, 53
- consSckId, 53
- consSusp, 53
- count, 53
- currentConsIndex, 53
- currentProdIndex, 53
- discardCount, 54
- dmaMode, 54
- endSig, 54
- firstConsIndex, 54
- firstProdIndex, 54
- flags, 54
- isDmaHandleDCache, 54
- lock, 54
- notification, 54
- overrideDscrIndex, 54
- prodAvailCount, 54
- prodFooter, 54
- prodHeader, 55
- prodSckId, 55
- prodSusp, 55

size, 55  
 startSig, 55  
 state, 55  
 type, 55  
 usbConsSusp, 55  
 validSckCount, 55  
 xferSize, 55  
**CyU3PDmaMultiChannelAbort**  
 cyu3dma.h, 240  
**CyU3PDmaMultiChannelCacheControl**  
 cyu3dma.h, 240  
**CyU3PDmaMultiChannelCommitBuffer**  
 cyu3dma.h, 241  
**CyU3PDmaMultiChannelConfig\_t**, 56  
 cb, 56  
 consHeader, 56  
 consSckId, 56  
 count, 57  
 cyu3dma.h, 218  
 dmaMode, 57  
 notification, 57  
 prodAvailCount, 57  
 prodFooter, 57  
 prodHeader, 57  
 prodSckId, 57  
 size, 57  
 validSckCount, 57  
**CyU3PDmaMultiChannelCreate**  
 cyu3dma.h, 241  
**CyU3PDmaMultiChannelDestroy**  
 cyu3dma.h, 242  
**CyU3PDmaMultiChannelDiscardBuffer**  
 cyu3dma.h, 242  
**CyU3PDmaMultiChannelGetBuffer**  
 cyu3dma.h, 243  
**CyU3PDmaMultiChannelGetHandle**  
 cyu3dma.h, 244  
**CyU3PDmaMultiChannelGetStatus**  
 cyu3dma.h, 244  
**CyU3PDmaMultiChannelsValid**  
 cyu3dma.h, 245  
**CyU3PDmaMultiChannelReset**  
 cyu3dma.h, 245  
**CyU3PDmaMultiChannelResume**  
 cyu3dma.h, 246  
**CyU3PDmaMultiChannelResumeUsbConsumer**  
 cyu3dma.h, 246  
**CyU3PDmaMultiChannelSetSuspend**  
 cyu3dma.h, 247  
**CyU3PDmaMultiChannelSetWrapUp**  
 cyu3dma.h, 248  
**CyU3PDmaMultiChannelSetXfer**  
 cyu3dma.h, 249  
**CyU3PDmaMultiChannelSetupRecvBuffer**  
 cyu3dma.h, 247  
**CyU3PDmaMultiChannelSetupSendBuffer**  
 cyu3dma.h, 248  
**CyU3PDmaMultiChannelSuspendUsbConsumer**  
 cyu3dma.h, 250  
**CyU3PDmaMultiChannelUpdateMode**  
 cyu3dma.h, 250  
**CyU3PDmaMultiChannelWaitForCompletion**  
 cyu3dma.h, 251  
**CyU3PDmaMultiChannelWaitForRecvBuffer**  
 cyu3dma.h, 251  
**CyU3PDmaMultiType\_t**  
 cyu3dma.h, 219, 221  
**CyU3PDmaMulticastDisableConsumer**  
 cyu3dma.h, 239  
**CyU3PDmaMulticastSocketSelect**  
 cyu3dma.h, 239  
**CyU3PDmaSckSuspType\_t**  
 cyu3dma.h, 219, 222  
**CyU3PDmaSocket\_t**, 58  
 activeDscr, 58  
 cyu3socket.h, 446  
 dscrChain, 58  
 intr, 58  
 intrMask, 58  
 sckEvent, 58  
 status, 59  
 unused19, 59  
 unused2, 59  
 xferCount, 59  
 xferSize, 59  
**CyU3PDmaSocketCallback\_t**  
 cyu3socket.h, 446  
**CyU3PDmaSocketClearInterrupts**  
 cyu3socket.h, 447  
**CyU3PDmaSocketConfig\_t**, 59  
 cyu3socket.h, 447  
 dscrChain, 60  
 intr, 60  
 intrMask, 60  
 status, 60  
 xferCount, 60  
 xferSize, 60  
**CyU3PDmaSocketDisable**  
 cyu3socket.h, 448  
**CyU3PDmaSocketEnable**  
 cyu3socket.h, 448  
**CyU3PDmaSocketGetConfig**  
 cyu3socket.h, 448  
**CyU3PDmaSocketId\_t**  
 cyu3dma.h, 219, 223  
**CyU3PDmaSocketIsValid**  
 cyu3socket.h, 449  
**CyU3PDmaSocketIsValidConsumer**  
 cyu3socket.h, 449  
**CyU3PDmaSocketIsValidProducer**  
 cyu3socket.h, 450  
**CyU3PDmaSocketRegisterCallback**  
 cyu3socket.h, 450  
**CyU3PDmaSocketSendEvent**  
 cyu3socket.h, 450  
**CyU3PDmaSocketSetConfig**

cyu3socket.h, 451  
CyU3PDmaSocketSetWrapUp  
    cyu3socket.h, 451  
CyU3PDmaState\_t  
    cyu3dma.h, 220, 225  
CyU3PDmaType\_t  
    cyu3dma.h, 220, 226  
CyU3PDmaUpdateSocketResume  
    cyu3socket.h, 452  
CyU3PDmaUpdateSocketSuspendOption  
    cyu3socket.h, 452  
CyU3PDmaUsbInEpGetChannel  
    cyu3dma.h, 252  
CyU3PDmaUsbInEpGetMultiChannel  
    cyu3dma.h, 252  
CyU3PDriveStrengthState\_t  
    cyu3system.h, 467, 469  
CyU3PEpConfig\_t, 60  
    burstLen, 61  
    cyu3usb.h, 505  
    enable, 61  
    epType, 61  
    isoPkts, 61  
    pkctSize, 61  
    streams, 61  
CyU3PErrorCode\_t  
    cyu3error.h, 254  
CyU3PEvent  
    cyu3os.h, 358  
CyU3PEventCreate  
    cyu3os.h, 367  
CyU3PEventDestroy  
    cyu3os.h, 368  
CyU3PEventGet  
    cyu3os.h, 368  
CyU3PEventPerfGet  
    cyu3os.h, 369  
CyU3PEventSet  
    cyu3os.h, 369  
CyU3PEventSetActiveGpio  
    cyu3os.h, 369  
CyU3PEventSystemPerfGet  
    cyu3os.h, 370  
CyU3PFiqContextRestore  
    cyu3os.h, 370  
CyU3PFiqContextSave  
    cyu3os.h, 370  
CyU3PFirmwareEntry  
    cyu3system.h, 481  
CyU3PFreeHeaps  
    cyu3os.h, 371  
CyU3PGetConnectState  
    cyu3usb.h, 512  
CyU3PGetTime  
    cyu3os.h, 371  
CyU3PGpifComparatorType  
    cyu3gpif.h, 259, 261  
CyU3PGpifConfig\_t, 61  
    cyfx3pib.h, 130  
    cyu3gpif.h, 259  
    functionCount, 62  
    functionData, 62  
    regCount, 62  
    regData, 62  
    stateCount, 62  
    stateData, 63  
    statePosition, 63  
CyU3PGpifConfigure  
    cyu3gpif.h, 262  
CyU3PGpifControlSWInput  
    cyu3gpif.h, 263  
CyU3PGpifDisable  
    cyu3gpif.h, 263  
CyU3PGpifErrorType  
    cyu3pib.h, 398, 401  
CyU3PGpifEventCb\_t  
    cyu3gpif.h, 259  
CyU3PGpifEventType  
    cyu3gpif.h, 260, 261  
CyU3PGpifGetSMState  
    cyu3gpif.h, 264  
CyU3PGpifInitAddrCounter  
    cyu3gpif.h, 264  
CyU3PGpifInitComparator  
    cyu3gpif.h, 264  
CyU3PGpifInitCtrlCounter  
    cyu3gpif.h, 265  
CyU3PGpifInitDataCounter  
    cyu3gpif.h, 265  
CyU3PGpifInitTransFunctions  
    cyu3gpif.h, 265  
CyU3PGpifLoad  
    cyu3gpif.h, 266  
CyU3PGpifOutput\_t  
    cyu3gpif.h, 260, 261  
CyU3PGpifOutputConfigure  
    cyu3gpif.h, 266  
CyU3PGpifReadDataWords  
    cyu3gpif.h, 267  
CyU3PGpifRegisterCallback  
    cyu3gpif.h, 267  
CyU3PGpifRegisterConfig  
    cyu3gpif.h, 268  
CyU3PGpifRegisterSMIntrCallback  
    cyu3gpif.h, 268  
CyU3PGpifSMControl  
    cyu3gpif.h, 269  
CyU3PGpifSMIntrCb\_t  
    cyu3gpif.h, 260  
CyU3PGpifSMStart  
    cyu3gpif.h, 269  
CyU3PGpifSMSwitch  
    cyu3gpif.h, 269  
CyU3PGpifSocketConfigure  
    cyu3gpif.h, 270  
CyU3PGpifWaveData, 63

cyfx3pib.h, 130  
 cyu3gpif.h, 260  
 leftData, 63  
 rightData, 63  
 CyU3PGpifWaveformLoad  
     cyu3gpif.h, 271  
 CyU3PGpifWriteDataWords  
     cyu3gpif.h, 271  
 CyU3PGpioClock\_t, 64  
     clkSrc, 64  
     cyu3lpp.h, 309  
     fastClkDiv, 64  
     halfDiv, 64  
     simpleDiv, 64  
     slowClkDiv, 65  
 CyU3PGpioComplexConfig\_t, 65  
     cyu3gpio.h, 274  
     driveHighEn, 66  
     driveLowEn, 66  
     inputEn, 66  
     intrMode, 66  
     outValue, 66  
     period, 66  
     pinMode, 66  
     threshold, 66  
     timer, 66  
     timerMode, 66  
 CyU3PGpioComplexGetThreshold  
     cyu3gpio.h, 278  
 CyU3PGpioComplexMeasureOnce  
     cyu3gpio.h, 278  
 CyU3PGpioComplexMode\_t  
     cyu3gpio.h, 274, 276  
 CyU3PGpioComplexPulse  
     cyu3gpio.h, 279  
 CyU3PGpioComplexPulseNow  
     cyu3gpio.h, 279  
 CyU3PGpioComplexSampleNow  
     cyu3gpio.h, 280  
 CyU3PGpioComplexUpdate  
     cyu3gpio.h, 280  
 CyU3PGpioComplexWaitForCompletion  
     cyu3gpio.h, 281  
 CyU3PGpioDeInit  
     cyu3gpio.h, 282  
 CyU3PGpioDisable  
     cyu3gpio.h, 282  
 CyU3PGpioGetIOValues  
     cyu3gpio.h, 282  
 CyU3PGpioGetValue  
     cyu3gpio.h, 283  
 CyU3PGpioInit  
     cyu3gpio.h, 283  
 CyU3PGpioIntrCb\_t  
     cyu3gpio.h, 274  
 CyU3PGpioIntrMode\_t  
     cyu3gpio.h, 275, 277  
 CyU3PGpioloMode\_t  
     cyu3lpp.h, 309, 310  
 CyU3PGpioSetClock  
     cyu3lpp.h, 310  
 CyU3PGpioSetComplexConfig  
     cyu3gpio.h, 284  
 CyU3PGpioSetIoMode  
     cyu3lpp.h, 312  
 CyU3PGpioSetSimpleConfig  
     cyu3gpio.h, 285  
 CyU3PGpioSetValue  
     cyu3gpio.h, 285  
 CyU3PGpioSimpleClkDiv\_t  
     cyu3lpp.h, 309, 310  
 CyU3PGpioSimpleConfig\_t, 66  
     cyu3gpio.h, 275  
     driveHighEn, 67  
     driveLowEn, 67  
     inputEn, 67  
     intrMode, 67  
     outValue, 67  
 CyU3PGpioSimpleGetValue  
     cyu3gpio.h, 286  
 CyU3PGpioSimpleSetValue  
     cyu3gpio.h, 286  
 CyU3PGpioStopClock  
     cyu3lpp.h, 312  
 CyU3PGpioTimerMode\_t  
     cyu3gpio.h, 275, 277  
 CyU3PI2cConfig\_t, 68  
     bitRate, 68  
     busTimeout, 68  
     cyu3i2c.h, 289  
     dmaTimeout, 68  
     isDma, 68  
 CyU3PI2cDeInit  
     cyu3i2c.h, 292  
 CyU3PI2cError\_t  
     cyu3i2c.h, 289, 291  
 CyU3PI2cEvt\_t  
     cyu3i2c.h, 290, 291  
 CyU3PI2cGetErrorCode  
     cyu3i2c.h, 292  
 CyU3PI2cInit  
     cyu3i2c.h, 293  
 CyU3PI2cIntrCb\_t  
     cyu3i2c.h, 290  
 CyU3PI2cPreamble\_t, 69  
     buffer, 69  
     ctrlMask, 70  
     cyu3i2c.h, 290  
     length, 70  
 CyU3PI2cReceiveBytes  
     cyu3i2c.h, 293  
 CyU3PI2cSendCommand  
     cyu3i2c.h, 294  
 CyU3PI2cSendStopCondition  
     cyu3i2c.h, 294  
 CyU3PI2cSetClock

cyu3lpp.h, 312  
CyU3PI2cSetConfig  
    cyu3i2c.h, 295  
CyU3PI2cSetTimeout  
    cyu3i2c.h, 295  
CyU3PI2cStopClock  
    cyu3lpp.h, 313  
CyU3PI2cTransmitBytes  
    cyu3i2c.h, 296  
CyU3PI2cWaitForAck  
    cyu3i2c.h, 297  
CyU3PI2cWaitForBlockXfer  
    cyu3i2c.h, 297  
CyU3PI2sConfig\_t, 70  
    cyu3i2s.h, 300  
    isDma, 70  
    isLsbFirst, 70  
    isMono, 70  
    padMode, 71  
    sampleRate, 71  
    sampleWidth, 71  
CyU3PI2sDelInit  
    cyu3i2s.h, 303  
CyU3PI2sEnableExternalMclk  
    cyu3i2s.h, 304  
CyU3PI2sError\_t  
    cyu3i2s.h, 300, 302  
CyU3PI2sEvt\_t  
    cyu3i2s.h, 300, 302  
CyU3PI2sInit  
    cyu3i2s.h, 304  
CyU3PI2sIntrCb\_t  
    cyu3i2s.h, 301  
CyU3PI2sPadMode\_t  
    cyu3i2s.h, 301, 302  
CyU3PI2sSampleRate\_t  
    cyu3i2s.h, 301, 303  
CyU3PI2sSampleWidth\_t  
    cyu3i2s.h, 301, 303  
CyU3PI2sSetClock  
    cyu3lpp.h, 313  
CyU3PI2sSetConfig  
    cyu3i2s.h, 304  
CyU3PI2sSetMute  
    cyu3i2s.h, 305  
CyU3PI2sSetPause  
    cyu3i2s.h, 305  
CyU3PI2sStopClock  
    cyu3lpp.h, 313  
CyU3PI2sTransmitBytes  
    cyu3i2s.h, 306  
CyU3PInitPageTable  
    cyu3mmu.h, 343  
CyU3PloMatrixConfig\_t, 71  
    cyfx3\_api.h, 169  
    gpioComplexEn, 72  
    gpioSimpleEn, 72  
    isDQ32Bit, 72  
    lppMode, 72  
    s0Mode, 72  
    s1Mode, 72  
    useI2C, 72  
    useI2S, 72  
    useSpi, 72  
    useUart, 72  
CyU3PloMatrixLppMode\_t  
    cyfx3\_api.h, 169, 170  
CyU3PIrqContextRestore  
    cyu3os.h, 371  
CyU3PIrqContextSave  
    cyu3os.h, 371  
CyU3PIrqNestingStart  
    cyu3os.h, 372  
CyU3PIrqNestingStop  
    cyu3os.h, 372  
CyU3PIrqVectoredContextSave  
    cyu3os.h, 372  
CyU3PIsGpioComplexIOConfigured  
    cyu3system.h, 481  
CyU3PIsGpioSimpleIOConfigured  
    cyu3system.h, 481  
CyU3PIsGpioValid  
    cyu3system.h, 482  
CyU3PIsLppIOConfigured  
    cyu3system.h, 482  
CyU3PJJumpToAddress  
    cyu3system.h, 483  
CyU3PKernelEntry  
    cyu3os.h, 373  
CyU3PLppDelInit  
    cyu3lpp.h, 314  
CyU3PLppGpioBlockIsOn  
    cyu3lpp.h, 314  
CyU3PLppInit  
    cyu3lpp.h, 314  
CyU3PLppInterruptHandler  
    cyu3lpp.h, 309  
CyU3PLppModule\_t  
    cyu3system.h, 467, 469  
CyU3PMbox, 73  
    cyu3mbox.h, 318  
    w0, 73  
    w1, 73  
CyU3PMboxCb\_t  
    cyu3mbox.h, 318  
CyU3PMboxDelInit  
    cyu3mbox.h, 319  
CyU3PMboxInit  
    cyu3mbox.h, 319  
CyU3PMboxRead  
    cyu3mbox.h, 319  
CyU3PMboxReset  
    cyu3mbox.h, 319  
CyU3PMboxWait  
    cyu3mbox.h, 320  
CyU3PMboxWrite

cyu3mbox.h, 320  
 CyU3PMemAlloc  
     cyu3os.h, 373  
 CyU3PMemCmp  
     cyu3os.h, 373  
 CyU3PMemCopy  
     cyu3os.h, 374  
 CyU3PMemCopy32  
     cyu3utils.h, 572  
 CyU3PMemCorruptCallback  
     cyu3os.h, 358  
 CyU3PMemCorruptionCheck  
     cyu3os.h, 374  
 CyU3PMemEnableChecks  
     cyu3os.h, 374  
 CyU3PMemFree  
     cyu3os.h, 375  
 CyU3PMemGetActiveList  
     cyu3os.h, 375  
 CyU3PMemGetCounts  
     cyu3os.h, 375  
 CyU3PMemInit  
     cyu3os.h, 375  
 CyU3PMemSet  
     cyu3os.h, 376  
 CyU3PMipicsiBusWidth\_t  
     cyu3mipicsi.h, 326, 328  
 CyU3PMipicsiCfg\_t, 73  
     csiRxClkDiv, 74  
     cyu3mipicsi.h, 326  
     dataFormat, 74  
     fifoDelay, 74  
     hResolution, 74  
     mClkCtl, 74  
     mClkRefDiv, 74  
     numDataLanes, 74  
     parClkDiv, 75  
     pllFbd, 75  
     pllFrs, 75  
     pllPrd, 75  
 CyU3PMipicsiCheckBlockActive  
     cyu3mipicsi.h, 332  
 CyU3PMipicsiDataFormat\_t  
     cyu3mipicsi.h, 326, 328  
 CyU3PMipicsiDeInit  
     cyu3mipicsi.h, 332  
 CyU3PMipicsiErrorCounts\_t, 75  
     crcErrCnt, 76  
     ctlErrCnt, 76  
     cyu3mipicsi.h, 327  
     eidErrCnt, 76  
     frmErrCnt, 76  
     mdlErrCnt, 76  
     recSyncErrCnt, 76  
     recrErrCnt, 76  
     unrSyncErrCnt, 76  
     unrcErrCnt, 76  
 CyU3PMipicsiGetErrors  
     cyu3mipicsi.h, 332  
 CyU3PMipicsiGpifLoad  
     cyu3mipicsi.h, 333  
 CyU3PMipicsi2cFreq\_t  
     cyu3mipicsi.h, 327, 329  
 CyU3PMipicsiInit  
     cyu3mipicsi.h, 333  
 CyU3PMipicsiInitializeGPIO  
     cyu3mipicsi.h, 334  
 CyU3PMipicsiInitializeI2c  
     cyu3mipicsi.h, 334  
 CyU3PMipicsiInitializePIB  
     cyu3mipicsi.h, 335  
 CyU3PMipicsiPllClkDiv\_t  
     cyu3mipicsi.h, 327, 330  
 CyU3PMipicsiPllClkFrs\_t  
     cyu3mipicsi.h, 327, 330  
 CyU3PMipicsiQueryIntfParams  
     cyu3mipicsi.h, 335  
 CyU3PMipicsiReset  
     cyu3mipicsi.h, 336  
 CyU3PMipicsiReset\_t  
     cyu3mipicsi.h, 327, 330  
 CyU3PMipicsiSensorIo\_t  
     cyu3mipicsi.h, 328, 331  
 CyU3PMipicsiSetIntfParams  
     cyu3mipicsi.h, 336  
 CyU3PMipicsiSetPhyTimeDelay  
     cyu3mipicsi.h, 337  
 CyU3PMipicsiSetSensorControl  
     cyu3mipicsi.h, 338  
 CyU3PMipicsiSleep  
     cyu3mipicsi.h, 338  
 CyU3PMipicsiWakeUp  
     cyu3mipicsi.h, 338  
 CyU3PMutex  
     cyu3os.h, 358  
 CyU3PMutexCreate  
     cyu3os.h, 376  
 CyU3PMutexDestroy  
     cyu3os.h, 377  
 CyU3PMutexGet  
     cyu3os.h, 377  
 CyU3PMutexPerfGet  
     cyu3os.h, 377  
 CyU3PMutexPut  
     cyu3os.h, 378  
 CyU3PMutexSetActivityGpio  
     cyu3os.h, 378  
 CyU3PMutexSystemPerfGet  
     cyu3os.h, 379  
 CyU3POsTimerHandler  
     cyu3os.h, 379  
 CyU3POsTimerInit  
     cyu3os.h, 379  
 CyU3POtgChargerDetectMode\_t  
     cyu3usb0t.h, 560, 562  
 CyU3POtgConfig\_t, 76

cb, 77  
chargerMode, 77  
cyu3usbotg.h, 560  
otgMode, 77  
CyU3POtgEvent\_t  
    cyu3usbotg.h, 561, 562  
CyU3POtgEventCallback\_t  
    cyu3usbotg.h, 561  
CyU3POtgGetMode  
    cyu3usbotg.h, 564  
CyU3POtgGetPeripheralType  
    cyu3usbotg.h, 564  
CyU3POtgHnpEnable  
    cyu3usbotg.h, 564  
CyU3POtgIsDeviceMode  
    cyu3usbotg.h, 566  
CyU3POtgIsHnpEnabled  
    cyu3usbotg.h, 566  
CyU3POtgIsHostMode  
    cyu3usbotg.h, 566  
CyU3POtgIsStarted  
    cyu3usbotg.h, 567  
CyU3POtgIsVBusValid  
    cyu3usbotg.h, 567  
CyU3POtgMode\_t  
    cyu3usbotg.h, 561, 562  
CyU3POtgPeripheralType\_t  
    cyu3usbotg.h, 561, 563  
CyU3POtgRequestHnp  
    cyu3usbotg.h, 567  
CyU3POtgSrpAbort  
    cyu3usbotg.h, 568  
CyU3POtgSrpStart  
    cyu3usbotg.h, 568  
CyU3POtgStart  
    cyu3usbotg.h, 568  
CyU3POtgStop  
    cyu3usbotg.h, 569  
CyU3PPartNumber\_t  
    cyfx3\_api.h, 170, 171  
CyU3PPibClock\_t, 77  
    clkDiv, 78  
    clkSrc, 78  
    cyu3pib.h, 399  
    isDIIEnable, 78  
    isHalfDiv, 78  
CyU3PPibClockPhase\_t  
    cyu3pib.h, 399, 402  
CyU3PPibDeInit  
    cyu3pib.h, 406  
CyU3PPibDIIConfigure  
    cyu3pib.h, 406  
CyU3PPibDIIMode\_t  
    cyu3pib.h, 399, 402  
CyU3PPibErrorType  
    cyu3pib.h, 400, 403  
CyU3PPibInit  
    cyu3pib.h, 407  
CyU3PPibIntrCb\_t  
    cyu3pib.h, 400  
CyU3PPibIntrType  
    cyu3pib.h, 400, 404  
CyU3PPibRegisterCallback  
    cyu3pib.h, 407  
CyU3PPibSelectIntSources  
    cyu3pib.h, 408  
CyU3PPibSelectMmcSlaveMode  
    cyu3pib.h, 408  
CyU3PPibSetInterruptPriority  
    cyu3pib.h, 409  
CyU3PPibSetPmmcBusyMode  
    cyu3pib.h, 409  
CyU3PPibSetSocketEop  
    cyu3pib.h, 409  
CyU3PPmmcEnableDirectAccess  
    cyu3pib.h, 410  
CyU3PPmmcEventType  
    cyu3pib.h, 400, 405  
CyU3PPmmcIntrCb\_t  
    cyu3pib.h, 401  
CyU3PPmmcRegisterCallback  
    cyu3pib.h, 410  
CyU3PPmmcState  
    cyu3pib.h, 401, 405  
CyU3PPPrefetchHandler  
    cyu3os.h, 379  
CyU3PQueue  
    cyu3os.h, 359  
CyU3PQueueCreate  
    cyu3os.h, 380  
CyU3PQueueDestroy  
    cyu3os.h, 380  
CyU3PQueueFlush  
    cyu3os.h, 381  
CyU3PQueuePerfGet  
    cyu3os.h, 381  
CyU3PQueuePrioritySend  
    cyu3os.h, 382  
CyU3PQueueReceive  
    cyu3os.h, 382  
CyU3PQueueSend  
    cyu3os.h, 383  
CyU3PQueueSystemPerfGet  
    cyu3os.h, 383  
CyU3PReadDeviceRegisters  
    cyu3utils.h, 572  
CyU3PRegisterGpioCallBack  
    cyu3gpio.h, 287  
CyU3PRegisterI2cCallBack  
    cyu3i2c.h, 298  
CyU3PRegisterI2sCallBack  
    cyu3i2s.h, 306  
CyU3PRegisterSpiCallBack  
    cyu3spi.h, 458  
CyU3PRegisterUartCallBack  
    cyu3uart.h, 495

CyU3PReturnStatus\_t  
     cyu3types.h, 488

CyU3PSDCardVer\_t  
     cyu3cardmgr.h, 188

CyU3PSPortMode\_t  
     cyfx3\_api.h, 170, 172

CyU3PSdMmcStates\_t  
     cyu3cardmgr.h, 189

CyU3PSdioAbortFunctionIO  
     cyu3sib.h, 420

CyU3PSdioByteReadWrite  
     cyu3sib.h, 421

CyU3PSdioCardRegs, 78  
     addrCIS, 79  
     CCCRVersion, 79  
     cardCapability, 79  
     cardSpeed, 79  
     fn0BlockSize, 79  
     isMemoryPresent, 79  
     manufacturerId, 79  
     manufacturerInfo, 79  
     numberOfFunctions, 79  
     sdioVersion, 79  
     supportsAsyncIntr, 80  
     uhsSupport, 80

CyU3PSdioCardRegs\_t  
     cyu3sib.h, 415

CyU3PSdioCardReset  
     cyu3sib.h, 421

CyU3PSdioDirectReadWrite  
     cyu3sib.h, 422

CyU3PSdioExtendedReadWrite  
     cyu3sib.h, 422

CyU3PSdioGetBlockSize  
     cyu3sib.h, 423

CyU3PSdioGetCISAddress  
     cyu3sib.h, 424

CyU3PSdioGetTuples  
     cyu3sib.h, 424

CyU3PSdioInterruptControl  
     cyu3sib.h, 425

CyU3PSdioQueryCard  
     cyu3sib.h, 425

CyU3PSdioReadWaitEnable  
     cyu3sib.h, 426

CyU3PSdioSetBlockSize  
     cyu3sib.h, 426

CyU3PSdioSuspendResumeFunction  
     cyu3sib.h, 427

CyU3PSSemaphore  
     cyu3os.h, 359

CyU3PSSemaphoreCreate  
     cyu3os.h, 383

CyU3PSSemaphoreDestroy  
     cyu3os.h, 384

CyU3PSSemaphoreGet  
     cyu3os.h, 384

CyU3PSSemaphorePut  
     cyu3os.h, 385

CyU3PSSemaphoreSetActivityGpio  
     cyu3os.h, 385

CyU3PSetEpConfig  
     cyu3usb.h, 512

CyU3PSetEpPacketSize  
     cyu3usb.h, 513

CyU3PSetGpioDriveStrength  
     cyu3lpp.h, 315

CyU3PSetI2cDriveStrength  
     cyu3lpp.h, 315

CyU3PSetPportDriveStrength  
     cyu3pib.h, 410

CyU3PSetSerialIoDriveStrength  
     cyu3system.h, 483

CyU3PSetTime  
     cyu3os.h, 386

CyU3PSibAbortRequest  
     cyu3sib.h, 427

CyU3PSibCardDetect  
     cyu3sib.h, 415, 418

CyU3PSibClearIntr  
     cyu3cardmgr.h, 187

CyU3PSibCommitReadWrite  
     cyu3sib.h, 427

CyU3PSibConvertAddr  
     cyu3cardmgr.h, 187

CyU3PSibCtxt, 80  
     activeUnitId, 80  
     inUse, 80  
     isRead, 80  
     mutexLock, 81  
     numBootLuns, 81  
     numUserLuns, 81  
     partition, 81  
     status, 81  
     writeTimer, 81  
     writeTimerCbk, 81

CyU3PSibDeInit  
     cyu3sib.h, 428

CyU3PSibDevInfo, 81  
     blkLen, 82  
     busWidth, 82  
     cardType, 82  
     ccc, 82  
     clkRate, 82  
     ddrMode, 82  
     eraseSize, 82  
     locked, 82  
     numBlks, 83  
     numUnits, 83  
     opVoltage, 83  
     removable, 83  
     writeable, 83

CyU3PSibDevInfo\_t  
     cyu3sib.h, 415

CyU3PSibDevPartition  
     cyu3sibpp.h, 442

CyU3PSibDevRegType  
  cyu3sib.h, 416, 418

CyU3PSibDevType  
  cyu3sib.h, 416, 418

CyU3PSibDisableCoreIntr  
  cyu3sibpp.h, 441

CyU3PSibEnableCoreIntr  
  cyu3sibpp.h, 442

CyU3PSibEraseBlocks  
  cyu3sib.h, 428

CyU3PSibEraseMode  
  cyu3sib.h, 416, 418

CyU3PSibEventType  
  cyu3sib.h, 416, 419

CyU3PSibEvtCbk\_t  
  cyu3sib.h, 416

CyU3PSibForceErase  
  cyu3sib.h, 429

CyU3PSibGetCSD  
  cyu3sib.h, 429

CyU3PSibGetCardStatus  
  cyu3sib.h, 429

CyU3PSibGetMMCExtCsd  
  cyu3sib.h, 430

CyU3PSibGlobalData, 83  
  activePartition, 84  
  cyu3sibpp.h, 442  
  isActive, 84  
  nextWrAddress, 84  
  openWrSize, 84  
  partConfig, 84  
  s0Enabled, 84  
  s1Enabled, 84  
  sibDmaChannel, 84  
  sibEvtCbk, 84  
  wrCommitPending, 84  
  wrCommitSize, 84

CyU3PSibInit  
  cyu3sib.h, 430

CyU3PSibInitCard  
  cyu3sibpp.h, 443

CyU3PSibIntfParams, 85  
  cardDetType, 85  
  cardInitDelay, 85  
  lowVoltage, 85  
  lvGpioState, 85  
  maxFreq, 85  
  resetGpio, 86  
  rstActHigh, 86  
  useDdr, 86  
  voltageSwGpio, 86  
  writeProtEnable, 86

CyU3PSibIntfParams\_t  
  cyu3sib.h, 416

CyU3PSibIntfVoltage  
  cyu3sib.h, 416, 419

CyU3PSibLockUnlockCard  
  cyu3sib.h, 431

CyU3PSibLunInfo, 86  
  blockSize, 87  
  location, 87  
  numBlocks, 87  
  startAddr, 87  
  type, 87  
  valid, 87  
  writeable, 87

CyU3PSibLunInfo\_t  
  cyu3sib.h, 417

CyU3PSibLunLocation  
  cyu3sib.h, 417, 419

CyU3PSibLunType  
  cyu3sib.h, 417, 419

CyU3PSibOpFreq  
  cyu3sib.h, 417, 420

CyU3PSibPartitionStorage  
  cyu3sib.h, 431

CyU3PSibPortId  
  cyu3sib.h, 417, 420

CyU3PSibQueryDevice  
  cyu3sib.h, 432

CyU3PSibQueryUnit  
  cyu3sib.h, 432

CyU3PSibReadRegister  
  cyu3sib.h, 433

CyU3PSibReadWriteRequest  
  cyu3sib.h, 433

CyU3PSibRegisterCbk  
  cyu3sib.h, 434

CyU3PSibRemovePartitions  
  cyu3sib.h, 434

CyU3PSibRemovePasswd  
  cyu3sib.h, 434

CyU3PSibResetSibCtrlr  
  cyu3cardmgr.h, 187

CyU3PSibSDRegs\_t  
  cyu3cardmgr.h, 189

CyU3PSibSendSwitchCommand  
  cyu3sib.h, 435

CyU3PSibSetActiveSocket  
  cyu3cardmgr.h, 187

CyU3PSibSetBlockLen  
  cyu3sib.h, 435

CyU3PSibSetIntfParams  
  cyu3sib.h, 436

CyU3PSibSetPasswd  
  cyu3sib.h, 436

CyU3PSibSetWriteCommitSize  
  cyu3sib.h, 436

CyU3PSibSocketId\_t  
  cyu3sibpp.h, 442

CyU3PSibStart  
  cyu3sib.h, 437

CyU3PSibStop  
  cyu3sib.h, 437

CyU3PSibUpdateLunInfo  
  cyu3sibpp.h, 443

CyU3PSibVendorAccess  
     cyu3sib.h, 437

CyU3PSibWriteTimerModify  
     cyu3sib.h, 438

CyU3PSpiConfig\_t, 87  
     clock, 88  
     cpha, 88  
     cpol, 88  
     cyu3spi.h, 454  
     isLsbFirst, 88  
     lagTime, 88  
     leadTime, 88  
     ssnCtrl, 88  
     ssnPol, 88  
     wordLen, 89

CyU3PSpiDelInit  
     cyu3spi.h, 458

CyU3PSpiDisableBlockXfer  
     cyu3spi.h, 458

CyU3PSpiError\_t  
     cyu3spi.h, 455, 456

CyU3PSpiEvt\_t  
     cyu3spi.h, 455, 456

CyU3PSpiInit  
     cyu3spi.h, 459

CyU3PSpiIntrCb\_t  
     cyu3spi.h, 455

CyU3PSpiReceiveWords  
     cyu3spi.h, 459

CyU3PSpiSetBlockXfer  
     cyu3spi.h, 460

CyU3PSpiSetClock  
     cyu3lpp.h, 316

CyU3PSpiSetConfig  
     cyu3spi.h, 460

CyU3PSpiSetSsnLine  
     cyu3spi.h, 461

CyU3PSpiSetTimeout  
     cyu3spi.h, 461

CyU3PSpiSsnCtrl\_t  
     cyu3spi.h, 455, 456

CyU3PSpiSsnLagLead\_t  
     cyu3spi.h, 456, 457

CyU3PSpiStopClock  
     cyu3lpp.h, 316

CyU3PSpiTransferWords  
     cyu3spi.h, 462

CyU3PSpiTransmitWords  
     cyu3spi.h, 462

CyU3PSpiWaitForBlockXfer  
     cyu3spi.h, 463

CyU3PSysBarrierSync  
     cyu3mmu.h, 343

CyU3PSysCacheDRegion  
     cyu3mmu.h, 343

CyU3PSysCacheICache  
     cyu3mmu.h, 343

CyU3PSysCheckStandbyParam  
     cyu3system.h, 483

CyU3PSysCheckSuspendParams  
     cyu3system.h, 483

CyU3PSysCleanDCache  
     cyu3mmu.h, 344

CyU3PSysCleanDRegion  
     cyu3mmu.h, 344

CyU3PSysClearDCache  
     cyu3mmu.h, 344

CyU3PSysClearDRegion  
     cyu3mmu.h, 344

CyU3PSysClockConfig\_t, 89  
     clkSrc, 90  
     cpuClkDiv, 90  
     cyu3system.h, 468  
     dmaClkDiv, 90  
     mmioClkDiv, 90  
     setSysClk400, 90  
     useStandbyClk, 90

CyU3PSysClockSrc\_t  
     cyu3system.h, 468, 469

CyU3PSysDisableCacheMMU  
     cyu3mmu.h, 345

CyU3PSysDisableDCache  
     cyu3mmu.h, 345

CyU3PSysDisableICache  
     cyu3mmu.h, 345

CyU3PSysDisableMMU  
     cyu3mmu.h, 346

CyU3PSysEnableCacheMMU  
     cyu3mmu.h, 346

CyU3PSysEnableDCache  
     cyu3mmu.h, 346

CyU3PSysEnableICache  
     cyu3mmu.h, 346

CyU3PSysEnableMMU  
     cyu3mmu.h, 347

CyU3PSysEnterStandbyMode  
     cyu3system.h, 484

CyU3PSysEnterSuspendMode  
     cyu3system.h, 484

CyU3PSysFlushCaches  
     cyu3mmu.h, 347

CyU3PSysFlushDCache  
     cyu3mmu.h, 347

CyU3PSysFlushDRegion  
     cyu3mmu.h, 347

CyU3PSysFlushICache  
     cyu3mmu.h, 348

CyU3PSysFlushIRegion  
     cyu3mmu.h, 348

CyU3PSysFlushTLBEntry  
     cyu3mmu.h, 348

CyU3PSysGetApiVersion  
     cyu3system.h, 485

CyU3PSysInitTCMs  
     cyu3mmu.h, 348

CyU3PSysLoadTLB

cyu3mmu.h, 349  
CyU3PSysLockTLBEntry  
    cyu3mmu.h, 349  
CyU3PSysSendEnterSuspendStatus  
    cyu3system.h, 486  
CyU3PSysThreadId\_t  
    cyu3system.h, 468, 470  
CyU3PSysWatchDogClear  
    cyu3system.h, 486  
CyU3PSysWatchDogConfigure  
    cyu3system.h, 486  
CyU3PSystemRegisterDriver  
    cyu3system.h, 486  
CyU3PThread  
    cyu3os.h, 359  
CyU3PThreadCreate  
    cyu3os.h, 386  
CyU3PThreadDestroy  
    cyu3os.h, 387  
CyU3PThreadEntry\_t  
    cyu3os.h, 360  
CyU3PThreadIdentify  
    cyu3os.h, 387  
CyU3PThreadInfoGet  
    cyu3os.h, 387  
CyU3PThreadPerfGet  
    cyu3os.h, 388  
CyU3PThreadPriorityChange  
    cyu3os.h, 388  
CyU3PThreadRelinquish  
    cyu3os.h, 389  
CyU3PThreadResume  
    cyu3os.h, 389  
CyU3PThreadSetActivityGpio  
    cyu3os.h, 389  
CyU3PThreadSleep  
    cyu3os.h, 390  
CyU3PThreadSuspend  
    cyu3os.h, 390  
CyU3PThreadSystemPerfGet  
    cyu3os.h, 391  
CyU3PThreadWaitAbort  
    cyu3os.h, 391  
CyU3PTimer  
    cyu3os.h, 360  
CyU3PTimerCb\_t  
    cyu3os.h, 360  
CyU3PTimerCreate  
    cyu3os.h, 392  
CyU3PTimerDestroy  
    cyu3os.h, 392  
CyU3PTimerModify  
    cyu3os.h, 393  
CyU3PTimerPerfGet  
    cyu3os.h, 393  
CyU3PTimerStart  
    cyu3os.h, 394  
CyU3PTimerStop  
    cyu3os.h, 394  
CyU3PTimerSystemPerfGet  
    cyu3os.h, 394  
CyU3PToolChainInit  
    cyu3system.h, 487  
CyU3PUSBEventCb\_t  
    cyu3usb.h, 506  
CyU3PUSBSetDescType\_t  
    cyfx3usb.h, 155, 157  
    cyu3usb.h, 507, 511  
CyU3PUSBSetupCb\_t  
    cyu3usb.h, 507  
CyU3PUSBSpeed\_t  
    cyu3usb.h, 511  
CyU3PUartBaudrate\_t  
    cyu3uart.h, 491, 492  
CyU3PUartConfig\_t, 90  
    baudRate, 91  
    cyu3uart.h, 491  
    flowCtrl, 91  
    isDma, 91  
    parity, 91  
    rxEnable, 91  
    stopBit, 91  
    txEnable, 91  
CyU3PUartDeinit  
    cyu3uart.h, 495  
CyU3PUartError\_t  
    cyu3uart.h, 491, 493  
CyU3PUartEvt\_t  
    cyu3uart.h, 491, 494  
CyU3PUartInit  
    cyu3uart.h, 495  
CyU3PUartIntrCb\_t  
    cyu3uart.h, 491  
CyU3PUartParity\_t  
    cyu3uart.h, 492, 494  
CyU3PUartReceiveBytes  
    cyu3uart.h, 496  
CyU3PUartRxSetBlockXfer  
    cyu3uart.h, 496  
CyU3PUartSetClock  
    cyu3lpp.h, 316  
CyU3PUartSetConfig  
    cyu3uart.h, 497  
CyU3PUartSetTimeout  
    cyu3uart.h, 497  
CyU3PUartStopBit\_t  
    cyu3uart.h, 492, 494  
CyU3PUartStopClock  
    cyu3lpp.h, 317  
CyU3PUartTransmitBytes  
    cyu3uart.h, 498  
CyU3PUartTxSetBlockXfer  
    cyu3uart.h, 498  
CyU3PUibCheckConnection  
    cyu3usb.h, 514  
CyU3PUndefHandler

cyu3os.h, 395  
 CyU3PUsb2Resume  
     cyu3usb.h, 514  
 CyU3PUsb2TestModes  
     cyu3usbconst.h, 540, 541  
 CyU3PUsb3PacketType  
     cyu3usbconst.h, 540, 541  
 CyU3PUsb3TpSubType  
     cyu3usbconst.h, 540, 541  
 CyU3PUsbAckSetup  
     cyu3usb.h, 514  
 CyU3PUsbChangeMapping  
     cyu3usb.h, 515  
 CyU3PUsbControlUsb2Support  
     cyu3usb.h, 515  
 CyU3PUsbControlVBusDetect  
     cyu3usb.h, 516  
 CyU3PUsbDescType  
     cyfx3usb.h, 155, 156  
     cyu3usbconst.h, 540, 542  
 CyU3PUsbDescrPtrs, 92  
     cyfx3usb.h, 155  
     usbConfigDesc\_p, 92  
     usbDevDesc\_p, 92  
     usbDevQualDesc\_p, 92  
     usbFSConfigDesc\_p, 92  
     usbHSConfigDesc\_p, 92  
     usbOtherSpeedConfigDesc\_p, 92  
     usbSSBOSDesc\_p, 92  
     usbSSConfigDesc\_p, 93  
     usbSSDevDesc\_p, 93  
     usbStringDesc\_p, 93  
 CyU3PUsbDevCapType  
     cyu3usbconst.h, 540, 543  
 CyU3PUsbDevProperty  
     cyu3usb.h, 505, 508  
 CyU3PUsbDoRemoteWakeups  
     cyu3usb.h, 516  
 CyU3PUsbEPSetBurstMode  
     cyu3usb.h, 518  
 CyU3PUsbEnableEPPrefetch  
     cyu3usb.h, 516  
 CyU3PUsbEnableITPEvent  
     cyu3usb.h, 517  
 CyU3PUsbEpEvtCb\_t  
     cyu3usb.h, 505  
 CyU3PUsbEpEvtControl  
     cyu3usb.h, 517  
 CyU3PUsbEpEvtType  
     cyu3usb.h, 505, 508  
 CyU3PUsbEpPrepare  
     cyu3usb.h, 518  
 CyU3PUsbEpSetPacketsPerBuffer  
     cyu3usb.h, 519  
 CyU3PUsbEpType\_t  
     cyu3usbconst.h, 540, 543  
 CyU3PUsbEventType\_t  
     cyu3usb.h, 506, 509  
 CyU3PUsbFeatureSelector  
     cyu3usbconst.h, 540, 543  
 CyU3PUsbFlushEp  
     cyu3usb.h, 519  
 CyU3PUsbForceFullSpeed  
     cyu3usb.h, 520  
 CyU3PUsbGetBooterVersion  
     cyu3usb.h, 520  
 CyU3PUsbGetDevProperty  
     cyu3usb.h, 520  
 CyU3PUsbGetEP0Data  
     cyu3usb.h, 521  
 CyU3PUsbGetEpCfg  
     cyu3usb.h, 522  
 CyU3PUsbGetEpSeqNum  
     cyu3usb.h, 522  
 CyU3PUsbGetErrorCounts  
     cyu3usb.h, 522  
 CyU3PUsbGetEventLogIndex  
     cyu3usb.h, 523  
 CyU3PUsbGetLinkPowerState  
     cyu3usb.h, 523  
 CyU3PUsbGetSpeed  
     cyu3usb.h, 524  
 CyU3PUsbHostConfig\_t, 93  
     cyu3usbhost.h, 548  
     ep0LowLevelControl, 93  
     eventCb, 93  
     xferCb, 94  
 CyU3PUsbHostEp0BeginXfer  
     cyu3usbhost.h, 550  
 CyU3PUsbHostEpAbort  
     cyu3usbhost.h, 551  
 CyU3PUsbHostEpAdd  
     cyu3usbhost.h, 551  
 CyU3PUsbHostEpConfig\_t, 94  
     cyu3usbhost.h, 548  
     fullPktSize, 94  
     isStreamMode, 94  
     maxPktSize, 94  
     mult, 95  
     pollingRate, 95  
     type, 95  
 CyU3PUsbHostEpRemove  
     cyu3usbhost.h, 552  
 CyU3PUsbHostEpReset  
     cyu3usbhost.h, 552  
 CyU3PUsbHostEpSetXfer  
     cyu3usbhost.h, 552  
 CyU3PUsbHostEpStatus\_t  
     cyu3usbhost.h, 548  
 CyU3PUsbHostEpWaitForCompletion  
     cyu3usbhost.h, 553  
 CyU3PUsbHostEpXferType\_t  
     cyu3usbhost.h, 548, 549  
 CyU3PUsbHostEventCb\_t  
     cyu3usbhost.h, 548  
 CyU3PUsbHostEventType\_t

cyu3usbhost.h, 549, 550  
CyU3PUsbHostGetDeviceAddress  
    cyu3usbhost.h, 554  
CyU3PUsbHostGetFrameNumber  
    cyu3usbhost.h, 554  
CyU3PUsbHostGetPortStatus  
    cyu3usbhost.h, 554  
CyU3PUsbHostIsStarted  
    cyu3usbhost.h, 555  
CyU3PUsbHostOpSpeed\_t  
    cyu3usbhost.h, 549, 550  
CyU3PUsbHostPortDisable  
    cyu3usbhost.h, 555  
CyU3PUsbHostPortEnable  
    cyu3usbhost.h, 555  
CyU3PUsbHostPortReset  
    cyu3usbhost.h, 556  
CyU3PUsbHostPortResume  
    cyu3usbhost.h, 556  
CyU3PUsbHostPortStatus\_t  
    cyu3usbhost.h, 549  
CyU3PUsbHostPortSuspend  
    cyu3usbhost.h, 556  
CyU3PUsbHostSendSetupRqt  
    cyu3usbhost.h, 557  
CyU3PUsbHostSetDeviceAddress  
    cyu3usbhost.h, 557  
CyU3PUsbHostStart  
    cyu3usbhost.h, 558  
CyU3PUsbHostStop  
    cyu3usbhost.h, 558  
CyU3PUsbHostXferCb\_t  
    cyu3usbhost.h, 549  
CyU3PUsbInitEventLog  
    cyu3usb.h, 524  
CyU3PUsbIsStarted  
    cyu3usb.h, 524  
CyU3PUsbJumpBackToBooter  
    cyu3usb.h, 525  
CyU3PUsbLPM\_COMP  
    cyfx3usb.h, 157  
    cyu3usb.h, 510  
CyU3PUsbLPM\_U0  
    cyfx3usb.h, 157  
    cyu3usb.h, 510  
CyU3PUsbLPM\_U1  
    cyfx3usb.h, 157  
    cyu3usb.h, 510  
CyU3PUsbLPM\_U2  
    cyfx3usb.h, 157  
    cyu3usb.h, 510  
CyU3PUsbLPM\_U3  
    cyfx3usb.h, 157  
    cyu3usb.h, 510  
CyU3PUsbLPM\_Unknown  
    cyfx3usb.h, 157  
    cyu3usb.h, 510  
CyU3PUsbLPMDisable  
    cyu3usb.h, 525  
CyU3PUsbLPMEnable  
    cyu3usb.h, 526  
CyU3PUsbLPMReqCb\_t  
    cyu3usb.h, 506  
CyU3PUsbLinkComplianceControl  
    cyu3usb.h, 525  
CyU3PUsbLinkPowerMode  
    cyfx3usb.h, 155, 157  
    cyu3usb.h, 506, 510  
CyU3PUsbLinkState\_t  
    cyu3usbconst.h, 541, 543  
CyU3PUsbMapStream  
    cyu3usb.h, 526  
CyU3PUsbMgrStates\_t  
    cyu3usb.h, 507, 510  
CyU3PUsbRegisterEpEvtCallback  
    cyu3usb.h, 527  
CyU3PUsbRegisterEventCallback  
    cyu3usb.h, 527  
CyU3PUsbRegisterLPMRequestCallback  
    cyu3usb.h, 528  
CyU3PUsbRegisterSetupCallback  
    cyu3usb.h, 528  
CyU3PUsbResetEndpointMemories  
    cyu3usb.h, 528  
CyU3PUsbResetEp  
    cyu3usb.h, 529  
CyU3PUsbSSCDisable  
    cyu3usb.h, 536  
CyU3PUsbSendAckTP  
    cyu3usb.h, 529  
CyU3PUsbSendDevNotification  
    cyu3usb.h, 530  
CyU3PUsbSendEP0Data  
    cyu3usb.h, 530  
CyU3PUsbSendErdy  
    cyu3usb.h, 531  
CyU3PUsbSendNrdy  
    cyu3usb.h, 531  
CyU3PUsbSetBooterSwitch  
    cyu3usb.h, 532  
CyU3PUsbSetDesc  
    cyu3usb.h, 532  
CyU3PUsbSetEpNak  
    cyu3usb.h, 533  
CyU3PUsbSetEpPktMode  
    cyu3usb.h, 533  
CyU3PUsbSetEpSeqNum  
    cyu3usb.h, 533  
CyU3PUsbSetEpSuspDisableMask  
    cyu3usb.h, 534  
CyU3PUsbSetLinkPowerState  
    cyu3usb.h, 534  
CyU3PUsbSetTxDeemphasis  
    cyu3usb.h, 535  
CyU3PUsbSetTxSwing  
    cyu3usb.h, 535

CyU3PUsbSetXfer  
     cyu3usb.h, 535

CyU3PUsbSetupCmds  
     cyu3usbconst.h, 541, 544

CyU3PUsbStall  
     cyu3usb.h, 536

CyU3PUsbStart  
     cyu3usb.h, 536

CyU3PUsbStop  
     cyu3usb.h, 537

CyU3PUsbVBattEnable  
     cyu3usb.h, 537

CyU3PVicClearInt  
     cyu3vic.h, 575

CyU3PVicDisableAllInterrupts  
     cyu3vic.h, 575

CyU3PVicDisableInt  
     cyu3vic.h, 575

CyU3PVicEnableInt  
     cyu3vic.h, 576

CyU3PVicEnableInterrupts  
     cyu3vic.h, 576

CyU3PVicIRQGetStatus  
     cyu3vic.h, 578

CyU3PVicInit  
     cyu3vic.h, 576

CyU3PVicIntGetPriority  
     cyu3vic.h, 577

CyU3PVicIntGetStatus  
     cyu3vic.h, 577

CyU3PVicIntSetPriority  
     cyu3vic.h, 577

CyU3PVicSetupIntVectors  
     cyu3vic.h, 578

CyU3PVicVector\_t  
     cyu3vic.h, 574

CyU3PWriteDeviceRegisters  
     cyu3utils.h, 572

cyfx3\_api.h  
     CY\_U3P\_IO\_MATRIX\_LPP\_DEFAULT, 171  
     CY\_U3P\_IO\_MATRIX\_LPP\_I2S\_ONLY, 171  
     CY\_U3P\_IO\_MATRIX\_LPP\_NONE, 171  
     CY\_U3P\_IO\_MATRIX\_LPP\_SPI\_ONLY, 171  
     CY\_U3P\_IO\_MATRIX\_LPP\_UART\_ONLY, 171  
     CY\_U3P\_SPORT\_1BIT, 173  
     CY\_U3P\_SPORT\_4BIT, 173  
     CY\_U3P\_SPORT\_8BIT, 173  
     CY\_U3P\_SPORT\_INACTIVE, 173

CYPART\_LASTDEV, 172

CYPART\_USB2011, 172

CYPART\_USB2013, 172

CYPART\_USB2014, 172

CYPART\_USB2023, 172

CYPART\_USB2024, 172

CYPART\_USB2025, 172

CYPART\_USB2031, 172

CYPART\_USB2032, 172

CYPART\_USB2033, 172

CYPART\_USB2034, 172

CYPART\_USB2035, 171

CYPART\_USB2064, 172

CYPART\_USB3011, 171

CYPART\_USB3012, 171

CYPART\_USB3013, 171

CYPART\_USB3014, 171

CYPART\_USB3021, 172

CYPART\_USB3023, 172

CYPART\_USB3024, 172

CYPART\_USB3025, 172

CYPART\_USB3031, 171

CYPART\_USB3032, 171

CYPART\_USB3033, 171

CYPART\_USB3034, 171

CYPART\_USB3035, 171

CYPART\_USB3061, 172

CYPART\_USB3062, 172

CYPART\_USB3063, 172

CYPART\_USB3064, 172

CYPART\_USB3065, 172

CYPART\_USB3075, 172

CYPART\_WB0163, 171

CYPART\_WB0263, 171

CyFx3BusyWait, 173

CyFx3DevClearSwInterrupt, 173

CyFx3DevGetMipiLaneCount, 173

CyFx3DevIOConfigure, 174

CyFx3DevIODisableSib0, 174

CyFx3DevIODisableSib1, 174

CyFx3DevIOIsGpio, 174

CyFx3DevIOIsI2cConfigured, 175

CyFx3DevIOIsI2sConfigured, 175

CyFx3DevIOIsSib0Configured, 175

CyFx3DevIOIsSib1BitWide, 175

CyFx3DevIOIsSib1Configured, 176

CyFx3DevIOIsSib8BitWide, 176

CyFx3DevIOIsSpiConfigured, 176

CyFx3DevIOIsUartConfigured, 176

CyFx3DevIOSelectGpio, 176

CyFx3DevIdentifyPart, 173

CyFx3DevInitPageTables, 174

CyFx3DevsGpif32Supported, 177

CyFx3DevsGpifSupported, 177

CyFx3DevsI2sSupported, 177

CyFx3DevsMipicsiSupported, 177

CyFx3DevsOtgSupported, 177

CyFx3DevsRam512Supported, 178

CyFx3DevsSib0Supported, 178

CyFx3DevsSib1Supported, 178

CyFx3DevsUsb3Supported, 178

CyFx3LppIsOn, 178

CyFx3PibDIIEnable, 179

CyFx3PibGetDIIStatus, 179

CyFx3PibIsOn, 179

CyFx3PibPowerOff, 179

CyFx3PibPowerOn, 179

CyFx3SetCpuFreq, 180

CyFx3SibPowerOff, 180  
CyFx3SibPowerOn, 180  
CyFx3Usb2PhySetup, 180  
CyFx3Usb3LnkRelaxHpTimeout, 181  
CyFx3Usb3LnkSetup, 181  
CyFx3Usb3SendTP, 181  
CyFx3UsbDmaPrefetchEnable, 181  
CyFx3UsbIsOn, 181  
CyFx3UsbPowerOn, 182  
CyFx3UsbWritePhyReg, 182  
CyU3PloMatrixConfig\_t, 169  
CyU3PloMatrixLppMode\_t, 169, 170  
CyU3PPartNumber\_t, 170, 171  
CyU3PSPortMode\_t, 170, 172  
**cyfx3device.h**  
CY\_FX3\_BOOT\_NUM\_CLK\_SRC, 100  
CY\_FX3\_BOOT\_SYS\_CLK\_BY\_16, 100  
CY\_FX3\_BOOT\_SYS\_CLK\_BY\_2, 100  
CY\_FX3\_BOOT\_SYS\_CLK\_BY\_4, 100  
CY\_FX3\_BOOT\_SYS\_CLK, 100  
CyFx3BootDeviceConfigureOMatrix, 100  
CyFx3BootDeviceInit, 101  
CyFx3BootDeviceReset, 101  
CyFx3BootGetPartNumber, 101  
CyFx3BootGpioOverride, 101  
CyFx3BootGpioRestore, 102  
CyFx3BootIloMatrixConfig\_t, 99  
CyFx3BootJumpToProgramEntry, 102  
CyFx3BootRetainGpioState, 103  
CyFx3BootSysClockSrc\_t, 99, 100  
CyFx3BootWatchdogClear, 103  
CyFx3BootWatchdogConfigure, 103  
CyFx3PartNumber\_t, 100  
**cyfx3dma.h**  
CY\_DMA\_CPU\_SOCKET\_CONS, 110  
CY\_DMA\_CPU\_SOCKET\_PROD, 110  
CY\_DMA\_LPP\_SOCKET\_I2C\_CONS, 108  
CY\_DMA\_LPP\_SOCKET\_I2C\_PROD, 108  
CY\_DMA\_LPP\_SOCKET\_I2S\_LEFT, 108  
CY\_DMA\_LPP\_SOCKET\_I2S\_RIGHT, 108  
CY\_DMA\_LPP\_SOCKET\_SPI\_CONS, 108  
CY\_DMA\_LPP\_SOCKET\_SPI\_PROD, 108  
CY\_DMA\_LPP\_SOCKET\_UART\_CONS, 108  
CY\_DMA\_LPP\_SOCKET\_UART\_PROD, 108  
CY\_DMA\_PIB\_SOCKET\_0, 108  
CY\_DMA\_PIB\_SOCKET\_1, 108  
CY\_DMA\_PIB\_SOCKET\_10, 108  
CY\_DMA\_PIB\_SOCKET\_11, 108  
CY\_DMA\_PIB\_SOCKET\_12, 108  
CY\_DMA\_PIB\_SOCKET\_13, 108  
CY\_DMA\_PIB\_SOCKET\_14, 108  
CY\_DMA\_PIB\_SOCKET\_15, 108  
CY\_DMA\_PIB\_SOCKET\_16, 108  
CY\_DMA\_PIB\_SOCKET\_17, 108  
CY\_DMA\_PIB\_SOCKET\_18, 108  
CY\_DMA\_PIB\_SOCKET\_19, 108  
CY\_DMA\_PIB\_SOCKET\_2, 108  
CY\_DMA\_PIB\_SOCKET\_20, 108  
CY\_DMA\_PIB\_SOCKET\_21, 109  
CY\_DMA\_PIB\_SOCKET\_22, 109  
CY\_DMA\_PIB\_SOCKET\_23, 109  
CY\_DMA\_PIB\_SOCKET\_24, 109  
CY\_DMA\_PIB\_SOCKET\_25, 109  
CY\_DMA\_PIB\_SOCKET\_26, 109  
CY\_DMA\_PIB\_SOCKET\_27, 109  
CY\_DMA\_PIB\_SOCKET\_28, 109  
CY\_DMA\_PIB\_SOCKET\_29, 109  
CY\_DMA\_PIB\_SOCKET\_3, 108  
CY\_DMA\_PIB\_SOCKET\_30, 109  
CY\_DMA\_PIB\_SOCKET\_31, 109  
CY\_DMA\_PIB\_SOCKET\_4, 108  
CY\_DMA\_PIB\_SOCKET\_5, 108  
CY\_DMA\_PIB\_SOCKET\_6, 108  
CY\_DMA\_PIB\_SOCKET\_7, 108  
CY\_DMA\_PIB\_SOCKET\_8, 108  
CY\_DMA\_PIB\_SOCKET\_9, 108  
CY\_DMA\_SIB\_SOCKET\_0, 109  
CY\_DMA\_SIB\_SOCKET\_1, 109  
CY\_DMA\_SIB\_SOCKET\_2, 109  
CY\_DMA\_SIB\_SOCKET\_3, 109  
CY\_DMA\_SIB\_SOCKET\_4, 109  
CY\_DMA\_SIB\_SOCKET\_5, 109  
CY\_DMA\_UIB\_SOCKET\_CONS\_0, 109  
CY\_DMA\_UIB\_SOCKET\_CONS\_1, 109  
CY\_DMA\_UIB\_SOCKET\_CONS\_10, 109  
CY\_DMA\_UIB\_SOCKET\_CONS\_11, 109  
CY\_DMA\_UIB\_SOCKET\_CONS\_12, 109  
CY\_DMA\_UIB\_SOCKET\_CONS\_13, 109  
CY\_DMA\_UIB\_SOCKET\_CONS\_14, 109  
CY\_DMA\_UIB\_SOCKET\_CONS\_15, 109  
CY\_DMA\_UIB\_SOCKET\_CONS\_2, 109  
CY\_DMA\_UIB\_SOCKET\_CONS\_3, 109  
CY\_DMA\_UIB\_SOCKET\_CONS\_4, 109  
CY\_DMA\_UIB\_SOCKET\_CONS\_5, 109  
CY\_DMA\_UIB\_SOCKET\_CONS\_6, 109  
CY\_DMA\_UIB\_SOCKET\_CONS\_7, 109  
CY\_DMA\_UIB\_SOCKET\_CONS\_8, 109  
CY\_DMA\_UIB\_SOCKET\_CONS\_9, 109  
CY\_DMA\_UIB\_SOCKET\_PROD\_0, 109  
CY\_DMA\_UIB\_SOCKET\_PROD\_1, 109  
CY\_DMA\_UIB\_SOCKET\_PROD\_10, 109  
CY\_DMA\_UIB\_SOCKET\_PROD\_11, 109  
CY\_DMA\_UIB\_SOCKET\_PROD\_12, 110  
CY\_DMA\_UIB\_SOCKET\_PROD\_13, 110  
CY\_DMA\_UIB\_SOCKET\_PROD\_14, 110  
CY\_DMA\_UIB\_SOCKET\_PROD\_15, 110  
CY\_DMA\_UIB\_SOCKET\_PROD\_2, 109  
CY\_DMA\_UIB\_SOCKET\_PROD\_3, 109  
CY\_DMA\_UIB\_SOCKET\_PROD\_4, 109  
CY\_DMA\_UIB\_SOCKET\_PROD\_5, 109  
CY\_DMA\_UIB\_SOCKET\_PROD\_6, 109  
CY\_DMA\_UIB\_SOCKET\_PROD\_7, 109  
CY\_DMA\_UIB\_SOCKET\_PROD\_8, 109  
CY\_DMA\_UIB\_SOCKET\_PROD\_9, 109  
CY\_FX3\_DMA\_GETDSCR\_BY\_INDEX, 106  
CY\_FX3\_DMA\_LPP SOCKCNT, 106

CY\_FX3\_DMA\_MAX\_DSCR\_INDEX, 106  
CY\_FX3\_DMA\_MIN\_DSCR\_INDEX, 106  
CY\_FX3\_DMA\_PIB\_SOCKCNT, 106  
CY\_FX3\_DMA\_USB\_IN\_SOCKCNT, 106  
CY\_FX3\_DMA\_USB\_OUT\_SOCKCNT, 106  
CY\_FX3\_LPP\_DMA\_DSCR\_INDEX, 106  
CY\_FX3\_PIB\_DMA\_DSCR\_INDEX, 106  
CY\_FX3\_USB\_DMA\_DSCR\_INDEX, 107  
CyFx3BootDmaCallback\_t, 107  
CyFx3BootDmaClearSockInterrupts, 110  
CyFx3BootDmaDescriptor\_t, 107  
CyFx3BootDmaDisableSocket, 110  
CyFx3BootDmaEnableSocket, 110  
CyFx3BootDmaGetDscrConfig, 111  
CyFx3BootDmaGetSockInterrupts, 112  
CyFx3BootDmaGetSocketConfig, 111  
CyFx3BootDmaRegisterCallback, 112  
CyFx3BootDmaSendSocketEvent, 112  
CyFx3BootDmaSetDscrConfig, 113  
CyFx3BootDmaSetSocketConfig, 113  
CyFx3BootDmaSockId\_t, 107, 108  
CyFx3BootDmaSockRegs\_t, 107  
CyFx3BootDmaSocket\_t, 107  
CyFx3BootDmaWrapSocket, 114

cyfx3error.h

CY\_FX3\_BOOT\_ERROR\_ABORTED, 115  
CY\_FX3\_BOOT\_ERROR\_ALREADY\_STARTED,  
    115  
CY\_FX3\_BOOT\_ERROR\_BAD\_ARGUMENT, 115  
CY\_FX3\_BOOT\_ERROR\_BAD\_DESCRIPTOR←  
    \_TYPE, 115  
CY\_FX3\_BOOT\_ERROR\_FAILURE, 116  
CY\_FX3\_BOOT\_ERROR\_I2C, 116  
CY\_FX3\_BOOT\_ERROR\_INVALID\_DMA\_ADDR,  
    115  
CY\_FX3\_BOOT\_ERROR\_MEMORY\_ERROR,  
    115  
CY\_FX3\_BOOT\_ERROR\_NO\_REENUM\_REQ←  
    UIRED, 115  
CY\_FX3\_BOOT\_ERROR\_NOT\_CONFIGURED,  
    115  
CY\_FX3\_BOOT\_ERROR\_NOT\_STARTED,  
    CY\_FX3\_BOOT\_ERROR\_BADIsFx3BootDmaSockRegs\_t,CY\_FX3\_BOOT\_ERROR\_NOT\_STARTED,  
    CY\_FX3\_BOOT\_ERROR\_NOT\_STARTED,  
    CY\_BOOT\_ERROR\_NOT\_STARTED,  
    CY\_BOOT\_ERROR\_NOT\_STARTED,  
    CyFx3BootDmaSockRegs5g 0 0 1 RG 19rg 0 gpio 114CY8BOOT\_ERROR\_NOT\_STARTED,CY8BOOT\_ERROR\_NOT\_STARTED,CY8BO  
CY8BOOT\_ERROR\_NOT\_STARTED,  
Y8BOOT\_ERROR\_NOERROR\_NOERROR\_NO 231.75.9290.195,  
OOT\_ERROR\_NOT\_STARTE09,  
ERROR\_NOT\_STARTE09,CY9BOOT\_ERROR\_NOT\_STARTE08,  
OR\_NOT\_STARTE09,  
NOT\_STARTE09,  
ockId\_t,  
ockId\_t,  
OOT\_ERROR\_NOT\_STARTE08,  
STARTE09,  
egs5g 0 0 09,  
OT\_STARTE09,  
RTE09,  
8,

CY\_FX3\_BOOT\_SPI\_NUM\_SSN\_LAG\_LEAD,  
    140  
CY\_FX3\_BOOT\_SPI\_SSN\_CTRL\_FW, 139  
CY\_FX3\_BOOT\_SPI\_SSN\_CTRL\_HW\_CPHA←  
    BASED, 139  
CY\_FX3\_BOOT\_SPI\_SSN\_CTRL\_HW\_EACH←  
    WORD, 139  
CY\_FX3\_BOOT\_SPI\_SSN\_CTRL\_HW\_END\_O←  
    F\_XFER, 139  
CY\_FX3\_BOOT\_SPI\_SSN\_CTRL\_NONE, 139  
CY\_FX3\_BOOT\_SPI\_SSN\_LAG\_LEAD\_HALF←  
    CLK, 140  
CY\_FX3\_BOOT\_SPI\_SSN\_LAG\_LEAD\_ONE←  
    CLK, 140  
CY\_FX3\_BOOT\_SPI\_SSN\_LAG\_LEAD\_ONE←  
    HALF\_CLK, 140  
CY\_FX3\_BOOT\_SPI\_SSN\_LAG\_LEAD\_ZERO←  
    CLK, 140  
CyFx3BootSpiConfig\_t, 138  
CyFx3BootSpiDeInit, 140  
CyFx3BootSpiDisableBlockXfer, 140  
CyFx3BootSpiDmaXferData, 140  
CyFx3BootSpiInit, 141  
CyFx3BootSpiReceiveWords, 141  
CyFx3BootSpiSetBlockXfer, 142  
CyFx3BootSpiSetConfig, 142  
CyFx3BootSpiSetSsnLine, 143  
CyFx3BootSpiSetTimeout, 143  
CyFx3BootSpiSsnCtrl\_t, 138, 139  
CyFx3BootSpiSsnLagLead\_t, 139  
CyFx3BootSpiTransmitWords, 143  
cyfx3uart.h  
    CY\_FX3\_BOOT\_UART\_BAUDRATE\_115200, 147  
    CY\_FX3\_BOOT\_UART\_BAUDRATE\_19200, 146  
    CY\_FX3\_BOOT\_UART\_BAUDRATE\_38400, 147  
    CY\_FX3\_BOOT\_UART\_BAUDRATE\_4800, 146  
    CY\_FX3\_BOOT\_UART\_BAUDRATE\_57600, 147  
    CY\_FX3\_BOOT\_UART\_BAUDRATE\_9600, 146  
    CY\_FX3\_BOOT\_UART EVEN\_PARITY, 147  
    CY\_FX3\_BOOT\_UART\_NO\_PARITY, 147  
    CY\_FX3\_BOOT\_UART\_NUM\_PARITY, 147  
    CY\_FX3\_BOOT\_UART\_ODD\_PARITY, 147  
    CY\_FX3\_BOOT\_UART\_ONE\_STOP\_BIT, 147  
    CY\_FX3\_BOOT\_UART\_TWO\_STOP\_BIT, 147  
CyFx3BootUartBaudrate\_t, 145, 146  
CyFx3BootUartConfig\_t, 145  
CyFx3BootUartDelInit, 147  
CyFx3BootUartDmaXferData, 147  
CyFx3BootUartInit, 148  
CyFx3BootUartParity\_t, 146, 147  
CyFx3BootUartPrintMessage, 148  
CyFx3BootUartReceiveBytes, 149  
CyFx3BootUartRxSetBlockXfer, 149  
CyFx3BootUartSetConfig, 149  
CyFx3BootUartSetTimeout, 150  
CyFx3BootUartStopBit\_t, 146, 147  
CyFx3BootUartTransmitBytes, 150  
CyFx3BootUartTxSetBlockXfer, 150  
cyfx3usb.h  
    CY\_FX3\_BOOT\_FULL\_SPEED, 156  
    CY\_FX3\_BOOT\_HIGH\_SPEED, 156  
    CY\_FX3\_BOOT\_NOT\_CONNECTED, 156  
    CY\_FX3\_BOOT\_SUPER\_SPEED, 156  
    CY\_FX3\_BOOT\_USB\_COMPLIANCE, 156  
    CY\_FX3\_BOOT\_USB\_CONNECT, 156  
    CY\_FX3\_BOOT\_USB\_DISCONNECT, 156  
    CY\_FX3\_BOOT\_USB\_EP\_BULK, 155  
    CY\_FX3\_BOOT\_USB\_EP\_CONTROL, 155  
    CY\_FX3\_BOOT\_USB\_EP\_INTR, 155  
    CY\_FX3\_BOOT\_USB\_EP\_ISO, 155  
    CY\_FX3\_BOOT\_USB\_IN\_SS\_DISCONNECT,  
        156  
    CY\_FX3\_BOOT\_USB\_RESET, 156  
    CY\_FX3\_BOOT\_USB\_RESUME, 156  
    CY\_FX3\_BOOT\_USB\_SUSPEND, 156  
    CY\_FX3\_USB\_MAX\_STRING\_DESC\_INDEX,  
        154  
    CY\_U3P\_BOS\_DESCR, 156, 157  
    CY\_U3P\_DEVICE\_CAPB\_DESCR, 156, 157  
    CY\_U3P\_SS\_EP\_COMPN\_DESCR, 156, 157  
    CY\_U3P\_USB\_CONFIG\_DESCR, 156  
    CY\_U3P\_USB\_DEVICE\_DESCR, 156  
    CY\_U3P\_USB\_DEVQUAL\_DESCR, 156  
    CY\_U3P\_USB\_ENDPNT\_DESCR, 156  
    CY\_U3P\_USB\_HID\_DESCR, 156, 157  
    CY\_U3P\_USB\_INTRFC\_DESCR, 156  
    CY\_U3P\_USB\_INTRFC\_POWER\_DESCR, 156,  
        157  
    CY\_U3P\_USB\_OTG\_DESCR, 157  
    CY\_U3P\_USB\_OTHERSPEED\_DESCR, 156  
    CY\_U3P\_USB\_REPORT\_DESCR, 156, 157  
    CY\_U3P\_USB\_SET\_DEVQUAL\_DESCR, 157,  
        158  
    CY\_U3P\_USB\_SET\_FS\_CONFIG\_DESCR, 157,  
        158  
    CY\_U3P\_USB\_SET\_HS\_CONFIG\_DESCR, 157,  
        158  
    CY\_U3P\_USB\_SET\_HS\_DEVICE\_DESCR, 157,  
        158  
    CY\_U3P\_USB\_SET\_OTG\_DESCR, 158  
    CY\_U3P\_USB\_SET\_SS\_BOS\_DESCR, 158  
    CY\_U3P\_USB\_SET\_SS\_CONFIG\_DESCR, 158  
    CY\_U3P\_USB\_SET\_SS\_DEVICE\_DESCR, 157,  
        158  
    CY\_U3P\_USB\_SET\_STRING\_DESCR, 158  
    CY\_U3P\_USB\_STRING\_DESCR, 156  
    CyFx3BootRegisterSetupCallback, 158  
    CyFx3BootUSBEventCb\_t, 154  
    CyFx3BootUSBSetupCb\_t, 154  
    CyFx3BootUsbAckSetup, 158  
    CyFx3BootUsbCheckUsb3Disconnect, 158  
    CyFx3BootUsbConnect, 159  
    CyFx3BootUsbDmaXferData, 159  
    CyFx3BootUsbEp0Pk\_t, 154  
    CyFx3BootUsbEp0StatusCheck, 159  
    CyFx3BootUsbEpConfig\_t, 154



CY\_U3P\_SDIO\_DISABLE\_INT, 198  
CY\_U3P\_SDIO\_EAI, 198  
CY\_U3P\_SDIO\_ENABLE\_ASYNC\_INT, 198  
CY\_U3P\_SDIO\_ENABLE\_HIGH\_SPEED, 198  
CY\_U3P\_SDIO\_ENABLE\_INT, 198  
CY\_U3P\_SDIO\_FULL\_SPEED, 198  
CY\_U3P\_SDIO\_HIGH\_SPEED, 198  
CY\_U3P\_SDIO\_INT\_MASTER, 198  
CY\_U3P\_SDIO\_INTC\_BT\_A\_AMP, 198  
CY\_U3P\_SDIO\_INTC\_BT\_A, 198  
CY\_U3P\_SDIO\_INTC\_BT\_B, 199  
CY\_U3P\_SDIO\_INTC\_CAM, 199  
CY\_U3P\_SDIO\_INTC\_EMBD\_ATA, 199  
CY\_U3P\_SDIO\_INTC\_GPS, 199  
CY\_U3P\_SDIO\_INTC\_NONE, 199  
CY\_U3P\_SDIO\_INTC\_PHS, 199  
CY\_U3P\_SDIO\_INTC\_UART, 199  
CY\_U3P\_SDIO\_INTC\_WLAN, 199  
CY\_U3P\_SDIO\_LOW\_SPEED, 199  
CY\_U3P\_SDIO\_READ\_AFTER\_WRITE, 199  
CY\_U3P\_SDIO\_REG\_BUS\_INTERFACE\_CON-  
TROL, 199  
CY\_U3P\_SDIO\_REG\_BUS\_SUSPEND, 199  
CY\_U3P\_SDIO\_REG\_CARD\_CAPABILITY, 200  
CY\_U3P\_SDIO\_REG\_CCCR\_HIGH\_SPEED, 200  
CY\_U3P\_SDIO\_REG\_CCCR\_REVISION, 200  
CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D0, 200  
CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D1, 200  
CY\_U3P\_SDIO\_REG\_CIS\_PTR\_D2, 200  
CY\_U3P\_SDIO\_REG\_DRIVER\_STRENGTH, 200  
CY\_U3P\_SDIO\_REG\_EXEC\_FLAGS, 200  
CY\_U3P\_SDIO\_REG\_FBR\_CIS\_PTR\_D1, 200  
CY\_U3P\_SDIO\_REG\_FBR\_CIS\_PTR\_D2, 200  
CY\_U3P\_SDIO\_REG\_FBR\_CIS\_PTR\_DO, 200  
CY\_U3P\_SDIO\_REG\_FBR\_CSA\_PTR\_D1, 200  
CY\_U3P\_SDIO\_REG\_FBR\_CSA\_PTR\_D2, 201  
CY\_U3P\_SDIO\_REG\_FBR\_CSA\_PTR\_DO, 201  
CY\_U3P\_SDIO\_REG\_FBR\_DATA\_ACCESS-  
WINDOW, 201  
CY\_U3P\_SDIO\_REG\_FBR\_EXT\_INTERFACE-  
CODE, 201  
CY\_U3P\_SDIO\_REG\_FBR\_INTERFACE\_CODE,  
201  
CY\_U3P\_SDIO\_REG\_FBR\_IO\_BLOCKSIZE\_D0,  
201  
CY\_U3P\_SDIO\_REG\_FBR\_IO\_BLOCKSIZE\_D1,  
201  
CY\_U3P\_SDIO\_REG\_FBR\_POWER\_SELECT,  
201  
CY\_U3P\_SDIO\_REG\_FUNCTION\_BLOCKSIZE-  
\_D0, 201  
CY\_U3P\_SDIO\_REG\_FUNCTION\_BLOCKSIZE-  
\_D1, 201  
CY\_U3P\_SDIO\_REG\_FUNCTION\_SELECT, 201  
CY\_U3P\_SDIO\_REG\_INTERRUPT\_EXTENSION,  
201  
CY\_U3P\_SDIO\_REG\_IO\_ABORT, 202  
CY\_U3P\_SDIO\_REG\_IO\_ENABLE, 202  
CY\_U3P\_SDIO\_REG\_IO\_INTR\_ENABLE, 202  
CY\_U3P\_SDIO\_REG\_IO\_INTR\_PENDING, 202  
CY\_U3P\_SDIO\_REG\_IO\_READY, 202  
CY\_U3P\_SDIO\_REG\_POWER\_CONTROL, 202  
CY\_U3P\_SDIO\_REG\_READY\_FLAGS, 202  
CY\_U3P\_SDIO\_REG\_SD\_SPEC\_REVISION, 202  
CY\_U3P\_SDIO\_REG\_UHS\_I\_SUPPORT, 202  
CY\_U3P\_SDIO\_RESET, 202  
CY\_U3P\_SDIO\_SAI, 202  
CY\_U3P\_SDIO\_SD\_Version\_1\_00, 202  
CY\_U3P\_SDIO\_SD\_Version\_1\_10, 203  
CY\_U3P\_SDIO\_SD\_Version\_2\_00, 203  
CY\_U3P\_SDIO\_SD\_Version\_3\_00, 203  
CY\_U3P\_SDIO\_SDDR50\_SPEED, 203  
CY\_U3P\_SDIO\_SDDR104\_SPEED, 203  
CY\_U3P\_SDIO\_SDDR12\_SPEED, 203  
CY\_U3P\_SDIO\_SDDR25\_SPEED, 203  
CY\_U3P\_SDIO\_SDDR50\_SPEED, 203  
CY\_U3P\_SDIO\_SUPPORT\_HIGH\_SPEED, 203  
CY\_U3P\_SDIO\_UHS\_SDDR50, 203  
CY\_U3P\_SDIO\_UHS\_SDDR104, 203  
CY\_U3P\_SDIO\_UHS\_SDDR50, 203  
CY\_U3P\_SDIO\_Version\_1\_00, 204  
CY\_U3P\_SDIO\_Version\_1\_10, 204  
CY\_U3P\_SDIO\_Version\_1\_20, 204  
CY\_U3P\_SDIO\_Version\_2\_00, 204  
CY\_U3P\_SDIO\_Version\_3\_00, 204  
cyu3descriptor.h  
CyU3PDmaDescriptor\_t, 206  
CyU3PDmaDscrChainCreate, 206  
CyU3PDmaDscrChainDestroy, 207  
CyU3PDmaDscrGet, 207  
CyU3PDmaDscrGetConfig, 207  
CyU3PDmaDscrGetFreeCount, 208  
CyU3PDmaDscrListCreate, 208  
CyU3PDmaDscrListDestroy, 208  
CyU3PDmaDscrPut, 209  
CyU3PDmaDscrSetConfig, 209  
gIDmaDescriptor, 210  
cyu3dma.h  
CY\_U3P\_CPU\_SOCKET\_CONS, 225  
CY\_U3P\_CPU\_SOCKET\_PROD, 225  
CY\_U3P\_DMA\_ABORTED, 225  
CY\_U3P\_DMA\_ACTIVE, 225  
CY\_U3P\_DMA\_CB\_ABORTED, 221  
CY\_U3P\_DMA\_CB\_CONS\_EVENT, 221  
CY\_U3P\_DMA\_CB\_CONS\_SUSP, 221  
CY\_U3P\_DMA\_CB\_ERROR, 221  
CY\_U3P\_DMA\_CB\_PROD\_EVENT, 221  
CY\_U3P\_DMA\_CB\_PROD\_SUSP, 221  
CY\_U3P\_DMA\_CB\_RECV\_CPLT, 221  
CY\_U3P\_DMA\_CB\_SEND\_CPLT, 221  
CY\_U3P\_DMA\_CB\_XFER\_CPLT, 221  
CY\_U3P\_DMA\_CONFIGURED, 225  
CY\_U3P\_DMA\_CONS\_OVERRIDE, 225  
CY\_U3P\_DMA\_ERROR, 225  
CY\_U3P\_DMA\_IN\_COMPLETION, 225  
CY\_U3P\_DMA\_MODE\_BUFFER, 221

CY\_U3P\_DMA\_MODE\_BYTE, 221  
 CY\_U3P\_DMA\_NOT\_CONFIGURED, 225  
 CY\_U3P\_DMA\_NUM\_MODES, 221  
 CY\_U3P\_DMA\_NUM\_SINGLE\_TYPES, 226  
 CY\_U3P\_DMA\_NUM\_STATES, 225  
 CY\_U3P\_DMA\_NUM\_TYPES, 222  
 CY\_U3P\_DMA\_PROD\_OVERRIDE, 225  
 CY\_U3P\_DMA\_RECV\_COMPLETED, 226  
 CY\_U3P\_DMA\_SCK\_SUSP\_CONS\_PARTIAL←  
     BUF, 222  
 CY\_U3P\_DMA\_SCK\_SUSP\_CUR\_BUF, 222  
 CY\_U3P\_DMA\_SCK\_SUSP\_EOP, 222  
 CY\_U3P\_DMA\_SCK\_SUSP\_NONE, 222  
 CY\_U3P\_DMA\_SEND\_COMPLETED, 226  
 CY\_U3P\_DMA\_TYPE\_AUTO\_MANY\_TO\_ONE,  
     222  
 CY\_U3P\_DMA\_TYPE\_AUTO\_ONE\_TO\_MANY,  
     222  
 CY\_U3P\_DMA\_TYPE\_AUTO\_SIGNAL, 226  
 CY\_U3P\_DMA\_TYPE\_AUTO, 226  
 CY\_U3P\_DMA\_TYPE\_MANUAL\_IN, 226  
 CY\_U3P\_DMA\_TYPE\_MANUAL\_MANY\_TO\_O←  
     NE, 222  
 CY\_U3P\_DMA\_TYPE\_MANUAL\_ONE\_TO\_MA←  
     NY, 222  
 CY\_U3P\_DMA\_TYPE\_MANUAL\_OUT, 226  
 CY\_U3P\_DMA\_TYPE\_MANUAL, 226  
 CY\_U3P\_DMA\_TYPE\_MULTICAST, 222  
 CY\_U3P\_DMA\_XFER\_COMPLETED, 225  
 CY\_U3P\_LPP\_SOCKET\_I2C\_CONS, 223  
 CY\_U3P\_LPP\_SOCKET\_I2C\_PROD, 223  
 CY\_U3P\_LPP\_SOCKET\_I2S\_LEFT, 223  
 CY\_U3P\_LPP\_SOCKET\_I2S\_RIGHT, 223  
 CY\_U3P\_LPP\_SOCKET\_SPI\_CONS, 223  
 CY\_U3P\_LPP\_SOCKET\_SPI\_PROD, 223  
 CY\_U3P\_LPP\_SOCKET\_UART\_CONS, 223  
 CY\_U3P\_LPP\_SOCKET\_UART\_PROD, 223  
 CY\_U3P\_PIB\_SOCKET\_0, 223  
 CY\_U3P\_PIB\_SOCKET\_1, 223  
 CY\_U3P\_PIB\_SOCKET\_10, 223  
 CY\_U3P\_PIB\_SOCKET\_11, 223  
 CY\_U3P\_PIB\_SOCKET\_12, 223  
 CY\_U3P\_PIB\_SOCKET\_13, 223  
 CY\_U3P\_PIB\_SOCKET\_14, 223  
 CY\_U3P\_PIB\_SOCKET\_15, 223  
 CY\_U3P\_PIB\_SOCKET\_16, 224  
 CY\_U3P\_PIB\_SOCKET\_17, 224  
 CY\_U3P\_PIB\_SOCKET\_18, 224  
 CY\_U3P\_PIB\_SOCKET\_19, 224  
 CY\_U3P\_PIB\_SOCKET\_2, 223  
 CY\_U3P\_PIB\_SOCKET\_20, 224  
 CY\_U3P\_PIB\_SOCKET\_21, 224  
 CY\_U3P\_PIB\_SOCKET\_22, 224  
 CY\_U3P\_PIB\_SOCKET\_23, 224  
 CY\_U3P\_PIB\_SOCKET\_24, 224  
 CY\_U3P\_PIB\_SOCKET\_25, 224  
 CY\_U3P\_PIB\_SOCKET\_26, 224  
 CY\_U3P\_PIB\_SOCKET\_27, 224  
 CY\_U3P\_PIB\_SOCKET\_28, 224  
 CY\_U3P\_PIB\_SOCKET\_29, 224  
 CY\_U3P\_PIB\_SOCKET\_3, 223  
 CY\_U3P\_PIB\_SOCKET\_30, 224  
 CY\_U3P\_PIB\_SOCKET\_31, 224  
 CY\_U3P\_PIB\_SOCKET\_4, 223  
 CY\_U3P\_PIB\_SOCKET\_5, 223  
 CY\_U3P\_PIB\_SOCKET\_6, 223  
 CY\_U3P\_PIB\_SOCKET\_7, 223  
 CY\_U3P\_PIB\_SOCKET\_8, 223  
 CY\_U3P\_PIB\_SOCKET\_9, 223  
 CY\_U3P\_SIB\_SOCKET\_0, 224  
 CY\_U3P\_SIB\_SOCKET\_1, 224  
 CY\_U3P\_SIB\_SOCKET\_2, 224  
 CY\_U3P\_SIB\_SOCKET\_3, 224  
 CY\_U3P\_SIB\_SOCKET\_4, 224  
 CY\_U3P\_SIB\_SOCKET\_5, 224  
 CY\_U3P\_UIB\_SOCKET\_CONS\_0, 224  
 CY\_U3P\_UIB\_SOCKET\_CONS\_1, 224  
 CY\_U3P\_UIB\_SOCKET\_CONS\_10, 224  
 CY\_U3P\_UIB\_SOCKET\_CONS\_11, 224  
 CY\_U3P\_UIB\_SOCKET\_CONS\_12, 224  
 CY\_U3P\_UIB\_SOCKET\_CONS\_13, 224  
 CY\_U3P\_UIB\_SOCKET\_CONS\_14, 224  
 CY\_U3P\_UIB\_SOCKET\_CONS\_15, 224  
 CY\_U3P\_UIB\_SOCKET\_CONS\_2, 224  
 CY\_U3P\_UIB\_SOCKET\_CONS\_3, 224  
 CY\_U3P\_UIB\_SOCKET\_CONS\_4, 224  
 CY\_U3P\_UIB\_SOCKET\_CONS\_5, 224  
 CY\_U3P\_UIB\_SOCKET\_CONS\_6, 224  
 CY\_U3P\_UIB\_SOCKET\_CONS\_7, 224  
 CY\_U3P\_UIB\_SOCKET\_CONS\_8, 224  
 CY\_U3P\_UIB\_SOCKET\_CONS\_9, 224  
 CY\_U3P\_UIB\_SOCKET\_PROD\_0, 224  
 CY\_U3P\_UIB\_SOCKET\_PROD\_1, 224  
 CY\_U3P\_UIB\_SOCKET\_PROD\_10, 225  
 CY\_U3P\_UIB\_SOCKET\_PROD\_11, 225  
 CY\_U3P\_UIB\_SOCKET\_PROD\_12, 225  
 CY\_U3P\_UIB\_SOCKET\_PROD\_13, 225  
 CY\_U3P\_UIB\_SOCKET\_PROD\_14, 225  
 CY\_U3P\_UIB\_SOCKET\_PROD\_15, 225  
 CY\_U3P\_UIB\_SOCKET\_PROD\_2, 224  
 CY\_U3P\_UIB\_SOCKET\_PROD\_3, 224  
 CY\_U3P\_UIB\_SOCKET\_PROD\_4, 224  
 CY\_U3P\_UIB\_SOCKET\_PROD\_5, 224  
 CY\_U3P\_UIB\_SOCKET\_PROD\_6, 224  
 CY\_U3P\_UIB\_SOCKET\_PROD\_7, 225  
 CY\_U3P\_UIB\_SOCKET\_PROD\_8, 225  
 CY\_U3P\_UIB\_SOCKET\_PROD\_9, 225  
 CyU3PDmaBuffer\_t, 216  
 CyU3PDmaCBInput\_t, 217  
 CyU3PDmaCallback\_t, 216  
 CyU3PDmaCbType\_t, 217, 220  
 CyU3PDmaChannelAbort, 226  
 CyU3PDmaChannelCacheControl, 227  
 CyU3PDmaChannelCommitBuffer, 227  
 CyU3PDmaChannelConfig\_t, 217  
 CyU3PDmaChannelCreate, 228

CyU3PDmaChannelDestroy, 229  
CyU3PDmaChannelDiscardBuffer, 229  
CyU3PDmaChannelGetBuffer, 230  
CyU3PDmaChannelGetHandle, 231  
CyU3PDmaChannelGetStatus, 231  
CyU3PDmaChannelIsValid, 232  
CyU3PDmaChannelReset, 232  
CyU3PDmaChannelResume, 232  
CyU3PDmaChannelResumeUsbConsumer, 233  
CyU3PDmaChannelSetSuspend, 233  
CyU3PDmaChannelSetWrapUp, 235  
CyU3PDmaChannelSetXfer, 236  
CyU3PDmaChannelSetupRecvBuffer, 234  
CyU3PDmaChannelSetupSendBuffer, 235  
CyU3PDmaChannelSuspendUsbConsumer, 236  
CyU3PDmaChannelUpdateMode, 237  
CyU3PDmaChannelWaitForCompletion, 237  
CyU3PDmaChannelWaitForRecvBuffer, 238  
CyU3PDmaEnableMulticast, 239  
CyU3PDmaMode\_t, 218, 221  
CyU3PDmaMultiCallback\_t, 218  
CyU3PDmaMultiChannelAbort, 240  
CyU3PDmaMultiChannelCacheControl, 240  
CyU3PDmaMultiChannelCommitBuffer, 241  
CyU3PDmaMultiChannelConfig\_t, 218  
CyU3PDmaMultiChannelCreate, 241  
CyU3PDmaMultiChannelDestroy, 242  
CyU3PDmaMultiChannelDiscardBuffer, 242  
CyU3PDmaMultiChannelGetBuffer, 243  
CyU3PDmaMultiChannelGetHandle, 244  
CyU3PDmaMultiChannelGetStatus, 244  
CyU3PDmaMultiChannelIsValid, 245  
CyU3PDmaMultiChannelReset, 245  
CyU3PDmaMultiChannelResume, 246  
CyU3PDmaMultiChannelResumeUsbConsumer, 246  
CyU3PDmaMultiChannelSetSuspend, 247  
CyU3PDmaMultiChannelSetWrapUp, 248  
CyU3PDmaMultiChannelSetXfer, 249  
CyU3PDmaMultiChannelSetupRecvBuffer, 247  
CyU3PDmaMultiChannelSetupSendBuffer, 248  
CyU3PDmaMultiChannelSuspendUsbConsumer, 250  
CyU3PDmaMultiChannelUpdateMode, 250  
CyU3PDmaMultiChannelWaitForCompletion, 251  
CyU3PDmaMultiChannelWaitForRecvBuffer, 251  
CyU3PDmaMultiType\_t, 219, 221  
CyU3PDmaMulticastDisableConsumer, 239  
CyU3PDmaMulticastSocketSelect, 239  
CyU3PDmaSckSuspType\_t, 219, 222  
CyU3PDmaSocketId\_t, 219, 223  
CyU3PDmaState\_t, 220, 225  
CyU3PDmaType\_t, 220, 226  
CyU3PDmaUsbInEpGetChannel, 252  
CyU3PDmaUsbInEpGetMultiChannel, 252  
cyu3error.h  
    CY\_U3P\_ERROR\_ABORTED, 255  
    CY\_U3P\_ERROR\_ACTIVATE\_FAILED, 255  
CY\_U3P\_ERROR\_ALREADY\_PARTITIONED, 256  
CY\_U3P\_ERROR\_ALREADY\_STARTED, 255  
CY\_U3P\_ERROR\_BAD\_ARGUMENT, 255  
CY\_U3P\_ERROR\_BAD\_CMD\_ARG, 256  
CY\_U3P\_ERROR\_BAD\_DESCRIPTOR\_TYPE, 255  
CY\_U3P\_ERROR\_BAD\_ENUM\_METHOD, 255  
CY\_U3P\_ERROR\_BAD\_EVENT\_GRP, 254  
CY\_U3P\_ERROR\_BAD\_INDEX, 255  
CY\_U3P\_ERROR\_BAD\_MUTEX, 255  
CY\_U3P\_ERROR\_BAD\_OPTION, 254  
CY\_U3P\_ERROR\_BAD\_PARTITION, 256  
CY\_U3P\_ERROR\_BAD\_POINTER, 254  
CY\_U3P\_ERROR\_BAD\_POOL, 254  
CY\_U3P\_ERROR\_BAD\_PRIORITY, 255  
CY\_U3P\_ERROR\_BAD\_QUEUE, 254  
CY\_U3P\_ERROR\_BAD\_SEMAPHORE, 254  
CY\_U3P\_ERROR\_BAD\_SIZE, 254  
CY\_U3P\_ERROR\_BAD\_THREAD, 255  
CY\_U3P\_ERROR\_BAD\_THRESHOLD, 255  
CY\_U3P\_ERROR\_BAD\_TICK, 255  
CY\_U3P\_ERROR\_BAD\_TIMER, 255  
CY\_U3P\_ERROR\_BLOCK\_FAILURE, 255  
CY\_U3P\_ERROR\_CARD\_FORCE\_ERASE, 256  
CY\_U3P\_ERROR\_CARD\_LOCK\_FAILURE, 256  
CY\_U3P\_ERROR\_CARD\_LOCKED, 256  
CY\_U3P\_ERROR\_CARD\_NOT\_ACTIVE, 256  
CY\_U3P\_ERROR\_CARD\_UNHEALTHY, 256  
CY\_U3P\_ERROR\_CARD\_WRONG\_RESPONSE, 256  
CY\_U3P\_ERROR\_CARD\_WRONG\_STATE, 256  
CY\_U3P\_ERROR\_CHANNEL\_CREATE\_FAILED, 255  
CY\_U3P\_ERROR\_CHANNEL\_DESTROY\_FAILED, 255  
CY\_U3P\_ERROR\_CMD\_NOT\_SUPPORTED, 256  
CY\_U3P\_ERROR\_CRC, 256  
CY\_U3P\_ERROR\_DELETE\_FAILED, 255  
CY\_U3P\_ERROR\_DELETED, 254  
CY\_U3P\_ERROR\_DEVICE\_BUSY, 256  
CY\_U3P\_ERROR\_DMA\_FAILURE, 255  
CY\_U3P\_ERROR\_FAILURE, 255  
CY\_U3P\_ERROR\_FEATURE\_NOT\_ENABLED, 255  
CY\_U3P\_ERROR\_ILLEGAL\_CMD, 256  
CY\_U3P\_ERROR\_INHERIT\_FAILED, 255  
CY\_U3P\_ERROR\_INVALID\_ADDR, 256  
CY\_U3P\_ERROR\_INVALID\_BLOCKSIZE, 256  
CY\_U3P\_ERROR\_INVALID\_CALLER, 255  
CY\_U3P\_ERROR\_INVALID\_CONFIGURATION, 255  
CY\_U3P\_ERROR\_INVALID\_DEV, 256  
CY\_U3P\_ERROR\_INVALID\_FUNCTION, 256  
CY\_U3P\_ERROR\_INVALID\_SEQUENCE, 255  
CY\_U3P\_ERROR\_INVALID\_UNIT, 256  
CY\_U3P\_ERROR\_INVALID\_VOLTAGE\_RANGE, 255

CY\_U3P\_ERROR\_INVALID\_WAIT, 254  
CY\_U3P\_ERROR\_IO\_ABORTED, 256  
CY\_U3P\_ERROR\_IO\_SUSPENDED, 256  
CY\_U3P\_ERROR\_LOST\_ARBITRATION, 255  
CY\_U3P\_ERROR\_MEDIA\_FAILURE, 256  
CY\_U3P\_ERROR\_MEMORY\_ERROR, 255  
CY\_U3P\_ERROR\_MUTEX\_FAILURE, 255  
CY\_U3P\_ERROR\_MUTEX\_PUT\_FAILED, 255  
CY\_U3P\_ERROR\_NO\_EVENTS, 254  
CY\_U3P\_ERROR\_NO\_METADATA, 256  
CY\_U3P\_ERROR\_NO\_REENUM\_REQUIRED,  
  256  
CY\_U3P\_ERROR\_NOT\_CONFIGURED, 255  
CY\_U3P\_ERROR\_NOT\_IDLE, 255  
CY\_U3P\_ERROR\_NOT\_PARTITIONED, 256  
CY\_U3P\_ERROR\_NOT\_STARTED, 255  
CY\_U3P\_ERROR\_NOT\_SUPPORTED, 255  
CY\_U3P\_ERROR\_NULL\_POINTER, 255  
CY\_U3P\_ERROR\_OPERN\_DISABLED, 256  
CY\_U3P\_ERROR\_QUEUE\_EMPTY, 254  
CY\_U3P\_ERROR\_QUEUE\_FULL, 254  
CY\_U3P\_ERROR\_READ\_WRITE\_ABORTED,  
  256  
CY\_U3P\_ERROR\_RESUME\_FAILED, 255  
CY\_U3P\_ERROR\_SDIO\_UNKNOWN, 256  
CY\_U3P\_ERROR\_SEMGET\_FAILED, 255  
CY\_U3P\_ERROR\_SIB\_INIT, 256  
CY\_U3P\_ERROR\_STALLED, 255  
CY\_U3P\_ERROR\_STANDBY\_FAILED, 255  
CY\_U3P\_ERROR\_SUSPEND\_FAILED, 255  
CY\_U3P\_ERROR\_SUSPEND\_LIFTED, 255  
CY\_U3P\_ERROR\_TIMEOUT, 255  
CY\_U3P\_ERROR\_TUPLE\_NOT\_FOUND, 256  
CY\_U3P\_ERROR\_UNINITIALIZED\_FUNCTION,  
  256  
CY\_U3P\_ERROR\_UNSUPPORTED\_CARD, 256  
CY\_U3P\_ERROR\_WAIT\_ABORT\_FAILED, 255  
CY\_U3P\_ERROR\_WAIT\_ABORTED, 255  
CY\_U3P\_ERROR\_WRITE\_PROTECTED, 256  
CY\_U3P\_ERROR\_XFER\_CANCELLED, 255  
CY\_U3P\_SUCCESS, 254  
CyU3PErrorCode\_t, 254  
cyu3gpif.h  
  CYU3P\_GPIF\_COMP\_ADDR, 261  
  CYU3P\_GPIF\_COMP\_CTRL, 261  
  CYU3P\_GPIF\_COMP\_DATA, 261  
  CYU3P\_GPIF\_EVT\_ADDR\_COMP, 261  
  CYU3P\_GPIF\_EVT\_ADDR\_COUNTER, 261  
  CYU3P\_GPIF\_EVT\_CRC\_ERROR, 261  
  CYU3P\_GPIF\_EVT\_CTRL\_COMP, 261  
  CYU3P\_GPIF\_EVT\_CTRL\_COUNTER, 261  
  CYU3P\_GPIF\_EVT\_DATA\_COMP, 261  
  CYU3P\_GPIF\_EVT\_DATA\_COUNTER, 261  
  CYU3P\_GPIF\_EVT\_END\_STATE, 261  
  CYU3P\_GPIF\_EVT\_SM\_INTERRUPT, 261  
  CYU3P\_GPIF\_EVT\_SWITCH\_TIMEOUT, 261  
  CYU3P\_GPIF\_OP\_ALPHA0, 262  
  CYU3P\_GPIF\_OP\_ALPHA1, 262  
CYU3P\_GPIF\_OP\_ALPHA2, 262  
CYU3P\_GPIF\_OP\_ALPHA3, 262  
CYU3P\_GPIF\_OP\_BETA0, 262  
CYU3P\_GPIF\_OP\_BETA1, 262  
CYU3P\_GPIF\_OP\_BETA2, 262  
CYU3P\_GPIF\_OP\_BETA3, 262  
CYU3P\_GPIF\_OP\_DMA\_READY, 262  
CYU3P\_GPIF\_OP\_PARTIAL, 262  
CYU3P\_GPIF\_OP\_PPDRQ, 262  
CYU3P\_GPIF\_OP\_THR0\_PART, 262  
CYU3P\_GPIF\_OP\_THR0\_READY, 262  
CYU3P\_GPIF\_OP\_THR1\_PART, 262  
CYU3P\_GPIF\_OP\_THR1\_READY, 262  
CYU3P\_GPIF\_OP\_THR2\_PART, 262  
CYU3P\_GPIF\_OP\_THR2\_READY, 262  
CYU3P\_GPIF\_OP\_THR3\_PART, 262  
CYU3P\_GPIF\_OP\_THR3\_READY, 262  
CyU3PGpifComparatorType, 259, 261  
CyU3PGpifConfig\_t, 259  
CyU3PGpifConfigure, 262  
CyU3PGpifControlSWInput, 263  
CyU3PGpifDisable, 263  
CyU3PGpifEventCb\_t, 259  
CyU3PGpifEventType, 260, 261  
CyU3PGpifGetSMState, 264  
CyU3PGpifInitAddrCounter, 264  
CyU3PGpifInitComparator, 264  
CyU3PGpifInitCtrlCounter, 265  
CyU3PGpifInitDataCounter, 265  
CyU3PGpifInitTransFunctions, 265  
CyU3PGpifLoad, 266  
CyU3PGpifOutput\_t, 260, 261  
CyU3PGpifOutputConfigure, 266  
CyU3PGpifReadDataWords, 267  
CyU3PGpifRegisterCallback, 267  
CyU3PGpifRegisterConfig, 268  
CyU3PGpifRegisterSMIntrCallback, 268  
CyU3PGpifSMControl, 269  
CyU3PGpifSMIntrCb\_t, 260  
CyU3PGpifSMStart, 269  
CyU3PGpifSMSwitch, 269  
CyU3PGpifSocketConfigure, 270  
CyU3PGpifWaveData, 260  
CyU3PGpifWaveformLoad, 271  
CyU3PGpifWriteDataWords, 271  
cyu3gpio.h  
  CY\_U3P\_GPIO\_INTR\_BOTH\_EDGE, 277  
  CY\_U3P\_GPIO\_INTR\_HIGH\_LEVEL, 277  
  CY\_U3P\_GPIO\_INTR\_LOW\_LEVEL, 277  
  CY\_U3P\_GPIO\_INTR\_NEG\_EDGE, 277  
  CY\_U3P\_GPIO\_INTR\_POS\_EDGE, 277  
  CY\_U3P\_GPIO\_INTR\_TIMER\_THRES, 277  
  CY\_U3P\_GPIO\_INTR\_TIMER\_ZERO, 277  
  CY\_U3P\_GPIO\_MODE\_MEASURE\_ANY\_ONCE,  
    277  
  CY\_U3P\_GPIO\_MODE\_MEASURE\_ANY, 277  
  CY\_U3P\_GPIO\_MODE\_MEASURE\_HIGH\_ON←  
    CE, 276

CY\_U3P\_GPIO\_MODE\_MEASURE\_HIGH, 276  
CY\_U3P\_GPIO\_MODE\_MEASURE\_LOW\_ON↔CE, 276  
CY\_U3P\_GPIO\_MODE\_MEASURE\_LOW, 276  
CY\_U3P\_GPIO\_MODE\_MEASURE\_NEG\_ONCE, 277  
CY\_U3P\_GPIO\_MODE\_MEASURE\_NEG, 276  
CY\_U3P\_GPIO\_MODE\_MEASURE\_POS\_ONCE, 277  
CY\_U3P\_GPIO\_MODE\_MEASURE\_POS, 276  
CY\_U3P\_GPIO\_MODE\_PULSE\_NOW, 276  
CY\_U3P\_GPIO\_MODE\_PULSE, 276  
CY\_U3P\_GPIO\_MODE\_PWM, 276  
CY\_U3P\_GPIO\_MODE\_SAMPLE\_NOW, 276  
CY\_U3P\_GPIO\_MODE\_STATIC, 276  
CY\_U3P\_GPIO\_MODE\_TOGGLE, 276  
CY\_U3P\_GPIO\_NO\_INTR, 277  
CY\_U3P\_GPIO\_TIMER\_ANY\_EDGE, 278  
CY\_U3P\_GPIO\_TIMER\_HIGH\_FREQ, 278  
CY\_U3P\_GPIO\_TIMER\_LOW\_FREQ, 278  
CY\_U3P\_GPIO\_TIMER\_NEG\_EDGE, 278  
CY\_U3P\_GPIO\_TIMER\_POS\_EDGE, 278  
CY\_U3P\_GPIO\_TIMER\_RESERVED, 278  
CY\_U3P\_GPIO\_TIMER\_SHUTDOWN, 278  
CY\_U3P\_GPIO\_TIMER\_STANDBY\_FREQ, 278  
CyU3PGpioComplexConfig\_t, 274  
CyU3PGpioComplexGetThreshold, 278  
CyU3PGpioComplexMeasureOnce, 278  
CyU3PGpioComplexMode\_t, 274, 276  
CyU3PGpioComplexPulse, 279  
CyU3PGpioComplexPulseNow, 279  
CyU3PGpioComplexSampleNow, 280  
CyU3PGpioComplexUpdate, 280  
CyU3PGpioComplexWaitForCompletion, 281  
CyU3PGpioDelInit, 282  
CyU3PGpioDisable, 282  
CyU3PGpioGetIValues, 282  
CyU3PGpioGetValue, 283  
CyU3PGpioInit, 283  
CyU3PGpioIntrCb\_t, 274  
CyU3PGpioIntrMode\_t, 275, 277  
CyU3PGpioSetComplexConfig, 284  
CyU3PGpioSetSimpleConfig, 285  
CyU3PGpioSetValue, 285  
CyU3PGpioSimpleConfig\_t, 275  
CyU3PGpioSimpleGetValue, 286  
CyU3PGpioSimpleSetValue, 286  
CyU3PGpioTimerMode\_t, 275, 277  
CyU3PRegisterGpioCallBack, 287  
cyu3i2s.h  
CY\_U2P\_I2S\_NUM\_PAD\_MODES, 303  
CY\_U3P\_I2S\_ERROR\_LTX\_OVERFLOW, 302  
CY\_U3P\_I2S\_ERROR\_LTX\_UNDERFLOW, 302  
CY\_U3P\_I2S\_ERROR\_RTX\_OVERFLOW, 302  
CY\_U3P\_I2S\_ERROR\_RTX\_UNDERFLOW, 302  
CY\_U3P\_I2S\_EVENT\_ERROR, 302  
CY\_U3P\_I2S\_EVENT\_PAUSED, 302  
CY\_U3P\_I2S\_EVENT\_TXL\_DONE, 302  
CY\_U3P\_I2S\_EVENT\_TXR\_DONE, 302  
CY\_U3P\_I2S\_NUM\_BIT\_WIDTH, 303  
CY\_U3P\_I2S\_PAD\_MODE\_CONTINUOUS, 303  
CY\_U3P\_I2S\_PAD\_MODE\_LEFT\_JUSTIFIED, 302  
CY\_U3P\_I2S\_PAD\_MODE\_NORMAL, 302  
CY\_U3P\_I2S\_PAD\_MODE\_RESERVED, 303  
CY\_U3P\_I2S\_PAD\_MODE\_RIGHT\_JUSTIFIED, 302  
CY\_U3P\_I2S\_SAMPLE\_RATE\_16KHz, 303  
CY\_U3P\_I2S\_SAMPLE\_RATE\_192KHz, 303  
CY\_U3P\_I2S\_SAMPLE\_RATE\_32KHz, 303  
CY\_U3P\_I2S\_SAMPLE\_RATE\_44\_1KHz, 303  
CY\_U3P\_I2S\_SAMPLE\_RATE\_48KHz, 303  
CY\_U3P\_I2S\_SAMPLE\_RATE\_8KHz, 303

CY\_U3P\_I2S\_SAMPLE\_RATE\_96KHz, 303  
CY\_U3P\_I2S\_WIDTH\_16\_BIT, 303  
CY\_U3P\_I2S\_WIDTH\_18\_BIT, 303  
CY\_U3P\_I2S\_WIDTH\_24\_BIT, 303  
CY\_U3P\_I2S\_WIDTH\_32\_BIT, 303  
CY\_U3P\_I2S\_WIDTH\_8\_BIT, 303  
CyU3PI2sConfig\_t, 300  
CyU3PI2sDeInit, 303  
CyU3PI2sEnableExternalMclk, 304  
CyU3PI2sError\_t, 300, 302  
CyU3PI2sEvt\_t, 300, 302  
CyU3PI2sInit, 304  
CyU3PI2sIntrCb\_t, 301  
CyU3PI2sPadMode\_t, 301, 302  
CyU3PI2sSampleRate\_t, 301, 303  
CyU3PI2sSampleWidth\_t, 301, 303  
CyU3PI2sSetConfig, 304  
CyU3PI2sSetMute, 305  
CyU3PI2sSetPause, 305  
CyU3PI2sTransmitBytes, 306  
CyU3PRegisterI2sCallBack, 306

cyu3lpp.h  
CY\_U3P\_GPIO\_IO\_MODE\_NONE, 310  
CY\_U3P\_GPIO\_IO\_MODE\_WPD, 310  
CY\_U3P\_GPIO\_IO\_MODE\_WPU, 310  
CY\_U3P\_GPIO\_SIMPLE\_DIV\_BY\_16, 310  
CY\_U3P\_GPIO\_SIMPLE\_DIV\_BY\_2, 310  
CY\_U3P\_GPIO\_SIMPLE\_DIV\_BY\_4, 310  
CY\_U3P\_GPIO\_SIMPLE\_DIV\_BY\_64, 310  
CY\_U3P\_GPIO\_SIMPLE\_NUM\_DIV, 310  
CyU3PGpioClock\_t, 309  
CyU3PGpioloMode\_t, 309, 310  
CyU3PGpioSetClock, 310  
CyU3PGpioSetIoMode, 312  
CyU3PGpioSimpleClkDiv\_t, 309, 310  
CyU3PGpioStopClock, 312  
CyU3PI2cSetClock, 312  
CyU3PI2cStopClock, 313  
CyU3PI2sSetClock, 313  
CyU3PI2sStopClock, 313  
CyU3PLppDeInit, 314  
CyU3PLppGpioBlockIsOn, 314  
CyU3PLppInit, 314  
CyU3PLppInterruptHandler, 309  
CyU3PSetGpioDriveStrength, 315  
CyU3PSetI2cDriveStrength, 315  
CyU3PSpiSetClock, 316  
CyU3PSpiStopClock, 316  
CyU3PUartSetClock, 316  
CyU3PUartStopClock, 317

cyu3mbox.h  
CyU3PMbox, 318  
CyU3PMboxCb\_t, 318  
CyU3PMboxDeInit, 319  
CyU3PMboxInit, 319  
CyU3PMboxRead, 319  
CyU3PMboxReset, 319  
CyU3PMboxWait, 320

CyU3PMboxWrite, 320  
gIMBoxCb, 320  
cyu3mipicsi.h  
ALPHA\_CX3\_START\_SCK0, 323  
ALPHA\_CX3\_START\_SCK1, 323  
CX3\_ALPHA\_START, 323  
CX3\_FULL\_BUFFER\_IN\_SCK0, 324  
CX3\_FULL\_BUFFER\_IN\_SCK1, 324  
CX3\_IDLE\_SCK0, 324  
CX3\_IDLE\_SCK1, 324  
CX3\_IDLE, 324  
CX3\_NUMBER\_OF\_STATES, 324  
CX3\_PARTIAL\_BUFFER\_IN\_SCK0, 324  
CX3\_PARTIAL\_BUFFER\_IN\_SCK1, 324  
CX3\_PUSH\_DATA\_SCK0, 324  
CX3\_PUSH\_DATA\_SCK1, 324  
CX3\_PUSH\_DATA\_TO\_SCK0, 324  
CX3\_PUSH\_DATA\_TO\_SCK1, 324  
CX3\_START\_SCK0, 325  
CX3\_START\_SCK1, 325  
CX3\_START, 325  
CX3\_WAIT\_FOR\_FRAME\_START\_SCK0, 325  
CX3\_WAIT\_FOR\_FRAME\_START\_SCK1, 325  
CX3\_WAIT\_FOR\_FRAME\_START, 325  
CX3\_WAIT\_FULL\_SCK0\_NEXT\_SCK1, 325  
CX3\_WAIT\_FULL\_SCK1\_NEXT\_SCK0, 325  
CX3\_WAIT\_TO\_FILL\_SCK0, 325  
CX3\_WAIT\_TO\_FILL\_SCK1, 325  
CY\_U3P\_CSI\_DF\_RAW10, 329  
CY\_U3P\_CSI\_DF\_RAW12, 329  
CY\_U3P\_CSI\_DF\_RAW14, 329  
CY\_U3P\_CSI\_DF\_RAW8, 329  
CY\_U3P\_CSI\_DF\_RGB565\_0, 329  
CY\_U3P\_CSI\_DF\_RGB565\_1, 329  
CY\_U3P\_CSI\_DF\_RGB565\_2, 329  
CY\_U3P\_CSI\_DF\_RGB666\_0, 329  
CY\_U3P\_CSI\_DF\_RGB666\_1, 329  
CY\_U3P\_CSI\_DF\_RGB888, 329  
CY\_U3P\_CSI\_DF\_YUV422\_10, 329  
CY\_U3P\_CSI\_DF\_YUV422\_8\_0, 329  
CY\_U3P\_CSI\_DF\_YUV422\_8\_1, 329  
CY\_U3P\_CSI\_DF\_YUV422\_8\_2, 329  
CY\_U3P\_CSI\_HARD\_RST, 331  
CY\_U3P\_CSI\_IO\_XRES, 331  
CY\_U3P\_CSI\_IO\_XSHUTDOWN, 331  
CY\_U3P\_CSI\_PLL\_CLK\_DIV\_2, 330  
CY\_U3P\_CSI\_PLL\_CLK\_DIV\_4, 330  
CY\_U3P\_CSI\_PLL\_CLK\_DIV\_8, 330  
CY\_U3P\_CSI\_PLL\_CLK\_DIV\_INVALID, 330  
CY\_U3P\_CSI\_PLL\_FRS\_125\_250M, 330  
CY\_U3P\_CSI\_PLL\_FRS\_250\_500M, 330  
CY\_U3P\_CSI\_PLL\_FRS\_500\_1000M, 330  
CY\_U3P\_CSI\_PLL\_FRS\_63\_125M, 330  
CY\_U3P\_CSI\_SOFT\_RST, 331  
CY\_U3P\_MIPICSI\_BUS\_16, 328  
CY\_U3P\_MIPICSI\_BUS\_24, 328  
CY\_U3P\_MIPICSI\_BUS\_8, 328  
CY\_U3P\_MIPICSI\_I2C\_100KHZ, 330

CY\_U3P\_MIPICSI\_I2C\_400KHZ, 330  
CyU3PCx3DeviceReset, 331  
CyU3PMipicsiBusWidth\_t, 326, 328  
CyU3PMipicsiCfg\_t, 326  
CyU3PMipicsiCheckBlockActive, 332  
CyU3PMipicsiDataFormat\_t, 326, 328  
CyU3PMipicsiDeInit, 332  
CyU3PMipicsiErrorCounts\_t, 327  
CyU3PMipicsiGetErrors, 332  
CyU3PMipicsiGpifLoad, 333  
CyU3PMipicsi2cFreq\_t, 327, 329  
CyU3PMipicsiInit, 333  
CyU3PMipicsiInitializeGPIO, 334  
CyU3PMipicsiInitializeI2c, 334  
CyU3PMipicsiInitializePIB, 335  
CyU3PMipicsiPIIClkDiv\_t, 327, 330  
CyU3PMipicsiPIIClkFrs\_t, 327, 330  
CyU3PMipicsiQueryIntfParams, 335  
CyU3PMipicsiReset, 336  
CyU3PMipicsiReset\_t, 327, 330  
CyU3PMipicsiSensorlo\_t, 328, 331  
CyU3PMipicsiSetIntfParams, 336  
CyU3PMipicsiSetPhyTimeDelay, 337  
CyU3PMipicsiSetSensorControl, 338  
CyU3PMipicsiSleep, 338  
CyU3PMipicsiWakeup, 338  
FW\_WAIT\_SCK0, 325  
FW\_WAIT\_SCK1, 325

cyu3mmu.h

CYU3P\_CACHE\_LINE\_SZ, 341  
CYU3P\_CACHE\_MMU\_EN\_MASK, 341  
CYU3P\_CACHE\_NWAYS, 341  
CYU3P\_CACHE\_REPLACEMENT\_MASK, 341  
CYU3P\_CACHE\_SIZE, 341  
CYU3P\_DCACHE\_EN\_MASK, 341  
CYU3P\_DTCM\_BASE\_ADDR, 341  
CYU3P\_DTCM\_SIZE, 341  
CYU3P\_DTCM\_SZ\_EN, 341  
CYU3P\_GCTL\_PAGE\_TABLE\_ADDR, 341  
CYU3P\_ICACHE\_EN\_MASK, 342  
CYU3P\_ITCM\_BASE\_ADDR, 342  
CYU3P\_ITCM\_SIZE, 342  
CYU3P\_ITCM\_SZ\_EN, 342  
CYU3P\_MMIO\_BASE\_ADDR, 342  
CYU3P\_MMIO\_SIZE, 342  
CYU3P\_MMU\_EN\_MASK, 342  
CYU3P\_ROM\_BASE\_ADDR, 342  
CYU3P\_ROM\_SIZE, 342  
CYU3P\_SYSMEM\_BASE\_ADDR, 342  
CYU3P\_SYSMEM\_SIZE, 342  
CYU3P\_TCMREG\_ADDRESS\_MASK, 342  
CYU3P\_VIC\_BASE\_ADDR, 343  
CYU3P\_VIC\_SIZE, 343  
CyU3PInitPageTable, 343  
CyU3PSysBarrierSync, 343  
CyU3PSysCacheDRegion, 343  
CyU3PSysCachelRegion, 343  
CyU3PSysCleanDCache, 344  
CyU3PSysCleanDRegion, 344  
CyU3PSysClearDCache, 344  
CyU3PSysClearDRegion, 344  
CyU3PSysDisableCacheMMU, 345  
CyU3PSysDisableDCache, 345  
CyU3PSysDisableICache, 345  
CyU3PSysDisableMMU, 346  
CyU3PSysEnableCacheMMU, 346  
CyU3PSysEnableDCache, 346  
CyU3PSysEnableICache, 346  
CyU3PSysEnableMMU, 347  
CyU3PSysFlushCaches, 347  
CyU3PSysFlushDCache, 347  
CyU3PSysFlushDRegion, 347  
CyU3PSysFlushICache, 348  
CyU3PSysFlushIRegion, 348  
CyU3PSysFlushTLBEntry, 348  
CyU3PSysInitTCMs, 348  
CyU3PSysLoadTLB, 349  
CyU3PSysLockTLBEntry, 349

## cyu3os.h

CyU3PAbortHandler, 361  
CyU3PApplicationDefine, 361  
CyU3PBlockAlloc, 361  
CyU3PBlockFree, 362  
CyU3PBlockPool, 357  
CyU3PBlockPoolCreate, 362  
CyU3PBlockPoolDestroy, 362  
CyU3PBufCorruptionCheck, 363  
CyU3PBufEnableChecks, 363  
CyU3PBufGetActiveList, 364  
CyU3PBufGetCounts, 364  
CyU3PByteAlloc, 364  
CyU3PByteFree, 364  
CyU3PBytePool, 358  
CyU3PBytePoolCreate, 365  
CyU3PBytePoolDestroy, 365  
CyU3PDmaBufferAlloc, 366  
CyU3PDmaBufferDeInit, 366  
CyU3PDmaBufferFree, 366  
CyU3PDmaBufferInit, 367  
CyU3PEvent, 358  
CyU3PEventCreate, 367  
CyU3PEventDestroy, 368  
CyU3PEventGet, 368  
CyU3PEventPerfGet, 369  
CyU3PEventSet, 369  
CyU3PEventSetActivityGpio, 369  
CyU3PEventSystemPerfGet, 370  
CyU3PFiqContextRestore, 370  
CyU3PFiqContextSave, 370  
CyU3PFreeHeaps, 371  
CyU3PGetTime, 371  
CyU3PIrqContextRestore, 371  
CyU3PIrqContextSave, 371  
CyU3PIrqNestingStart, 372  
CyU3PIrqNestingStop, 372  
CyU3PIrqVectoredContextSave, 372

CyU3PKernelEntry, 373  
CyU3PMemAlloc, 373  
CyU3PMemCmp, 373  
CyU3PMemCopy, 374  
CyU3PMemCorruptCallback, 358  
CyU3PMemCorruptionCheck, 374  
CyU3PMemEnableChecks, 374  
CyU3PMemFree, 375  
CyU3PMemGetActiveList, 375  
CyU3PMemGetCounts, 375  
CyU3PMemInit, 375  
CyU3PMemSet, 376  
CyU3PMutex, 358  
CyU3PMutexCreate, 376  
CyU3PMutexDestroy, 377  
CyU3PMutexGet, 377  
CyU3PMutexPerfGet, 377  
CyU3PMutexPut, 378  
CyU3PMutexSetActivityGpio, 378  
CyU3PMutexSystemPerfGet, 379  
CyU3POsTimerHandler, 379  
CyU3POsTimerInit, 379  
CyU3PPrefetchHandler, 379  
CyU3PQueue, 359  
CyU3PQueueCreate, 380  
CyU3PQueueDestroy, 380  
CyU3PQueueFlush, 381  
CyU3PQueuePerfGet, 381  
CyU3PQueuePrioritySend, 382  
CyU3PQueueReceive, 382  
CyU3PQueueSend, 383  
CyU3PQueueSystemPerfGet, 383  
CyU3PSemaphore, 359  
CyU3PSemaphoreCreate, 383  
CyU3PSemaphoreDestroy, 384  
CyU3PSemaphoreGet, 384  
CyU3PSemaphorePut, 385  
CyU3PSemaphoreSetActivityGpio, 385  
CyU3PSetTime, 386  
CyU3PThread, 359  
CyU3PThreadCreate, 386  
CyU3PThreadDestroy, *enadCreate*,

CYU3P\_PIB\_ERR\_THR0\_RD\_BURST, 404  
CYU3P\_PIB\_ERR\_THR0\_RD\_FORCE\_END, 404  
CYU3P\_PIB\_ERR\_THR0\_RD\_UNDERRUN, 403  
CYU3P\_PIB\_ERR\_THR0\_SCK\_INACTIVE, 404  
CYU3P\_PIB\_ERR\_THR0\_WR\_OVERRUN, 403  
CYU3P\_PIB\_ERR\_THR1\_ADAP\_OVERRUN, 404  
CYU3P\_PIB\_ERR\_THR1\_ADAP\_UNDERRUN, 404  
CYU3P\_PIB\_ERR\_THR1\_DIRECTION, 403  
CYU3P\_PIB\_ERR\_THR1\_RD\_BURST, 404  
CYU3P\_PIB\_ERR\_THR1\_RD\_FORCE\_END, 404  
CYU3P\_PIB\_ERR\_THR1\_RD\_UNDERRUN, 403  
CYU3P\_PIB\_ERR\_THR1\_SCK\_INACTIVE, 404  
CYU3P\_PIB\_ERR\_THR1\_WR\_OVERRUN, 403  
CYU3P\_PIB\_ERR\_THR2\_ADAP\_OVERRUN, 404  
CYU3P\_PIB\_ERR\_THR2\_ADAP\_UNDERRUN, 404  
CYU3P\_PIB\_ERR\_THR2\_DIRECTION, 403  
CYU3P\_PIB\_ERR\_THR2\_RD\_BURST, 404  
CYU3P\_PIB\_ERR\_THR2\_RD\_FORCE\_END, 404  
CYU3P\_PIB\_ERR\_THR2\_RD\_UNDERRUN, 403  
CYU3P\_PIB\_ERR\_THR2\_SCK\_INACTIVE, 404  
CYU3P\_PIB\_ERR\_THR2\_WR\_OVERRUN, 403  
CYU3P\_PIB\_ERR\_THR3\_ADAP\_OVERRUN, 404  
CYU3P\_PIB\_ERR\_THR3\_ADAP\_UNDERRUN, 404  
CYU3P\_PIB\_ERR\_THR3\_DIRECTION, 403  
CYU3P\_PIB\_ERR\_THR3\_RD\_BURST, 404  
CYU3P\_PIB\_ERR\_THR3\_RD\_FORCE\_END, 404  
CYU3P\_PIB\_ERR\_THR3\_RD\_UNDERRUN, 403  
CYU3P\_PIB\_ERR\_THR3\_SCK\_INACTIVE, 404  
CYU3P\_PIB\_ERR\_THR3\_WR\_OVERRUN, 403  
CYU3P\_PIB\_INTR\_DLL\_UPDATE, 405  
CYU3P\_PIB\_INTR\_ERROR, 405  
CYU3P\_PIB\_INTR\_PPCONFIG, 405  
CYU3P\_PMMC\_CMD12\_STOP, 405  
CYU3P\_PMMC\_CMD15\_INACTIVE, 405  
CYU3P\_PMMC\_CMD5\_AWAKE, 405  
CYU3P\_PMMC\_CMD5\_SLEEP, 405  
CYU3P\_PMMC\_CMD6\_SWITCH, 405  
CYU3P\_PMMC\_CMD7\_SELECT, 405  
CYU3P\_PMMC\_DIRECT\_READ, 405  
CYU3P\_PMMC\_DIRECT\_WRITE, 405  
CYU3P\_PMMC\_GOIDLE\_CMD, 405  
CYU3P\_PMMC\_SOCKET\_NOT\_READY, 405  
CyU3PGpifErrorType, 398, 401  
CyU3PPibClock\_t, 399  
CyU3PPibClockPhase\_t, 399, 402  
CyU3PPibDeInit, 406  
CyU3PPibDIIConfigure, 406  
CyU3PPibDIIMode\_t, 399, 402  
CyU3PPibErrorType, 400, 403  
CyU3PPibInit, 407  
CyU3PPibIntrCb\_t, 400  
CyU3PPibIntrType, 400, 404  
CyU3PPibRegisterCallback, 407  
CyU3PPibSelectIntSources, 408  
CyU3PPibSelectMmcSlaveMode, 408  
CyU3PPibSetInterruptPriority, 409  
CyU3PPibSetPmmcBusyMode, 409  
CyU3PPibSetSocketEop, 409  
CyU3PPmmcEnableDirectAccess, 410  
CyU3PPmmcEventType, 400, 405  
CyU3PPmmcIntrCb\_t, 401  
CyU3PPmmcRegisterCallback, 410  
CyU3PPmmcState, 401, 405  
CyU3PSetPportDriveStrength, 410  
cyu3sib.h  
CY\_U3P\_SD\_REG\_CID, 418  
CY\_U3P\_SD\_REG\_CSD, 418  
CY\_U3P\_SD\_REG\_OCR, 418  
CY\_U3P\_SIB\_DETECT\_DAT\_3, 418  
CY\_U3P\_SIB\_DETECT\_GPIO, 418  
CY\_U3P\_SIB\_DETECT\_NONE, 418  
CY\_U3P\_SIB\_DEV\_MMIC, 418  
CY\_U3P\_SIB\_DEV\_NONE, 418  
CY\_U3P\_SIB\_DEV\_SDIO\_COMBO, 418  
CY\_U3P\_SIB\_DEV\_SDIO, 418  
CY\_U3P\_SIB\_DEV\_SD, 418  
CY\_U3P\_SIB\_ERASE\_SECURE, 418  
CY\_U3P\_SIB\_ERASE\_STANDARD, 418  
CY\_U3P\_SIB\_ERASE\_TRIM\_STEP1, 419  
CY\_U3P\_SIB\_ERASE\_TRIM\_STEP2, 419  
CY\_U3P\_SIB\_EVENT\_ABORT, 419  
CY\_U3P\_SIB\_EVENT\_DATA\_ERROR, 419  
CY\_U3P\_SIB\_EVENT\_INSERT, 419  
CY\_U3P\_SIB\_EVENT\_RELEASE, 419  
CY\_U3P\_SIB\_EVENT\_REMOVE, 419  
CY\_U3P\_SIB\_EVENT\_SDIO\_INTR, 419  
CY\_U3P\_SIB\_EVENT\_XFER\_CPLT, 419  
CY\_U3P\_SIB\_FREQ\_104MHZ, 420  
CY\_U3P\_SIB\_FREQ\_20MHZ, 420  
CY\_U3P\_SIB\_FREQ\_26MHZ, 420  
CY\_U3P\_SIB\_FREQ\_400KHZ, 420  
CY\_U3P\_SIB\_FREQ\_52MHZ, 420  
CY\_U3P\_SIB\_LOCATION\_BOOT1, 419  
CY\_U3P\_SIB\_LOCATION\_BOOT2, 419  
CY\_U3P\_SIB\_LOCATION\_USER, 419  
CY\_U3P\_SIB\_LUN\_BOOT, 420  
CY\_U3P\_SIB\_LUN\_DATA, 420  
CY\_U3P\_SIB\_LUN\_RSVD, 420  
CY\_U3P\_SIB\_NUM\_PORTS, 420  
CY\_U3P\_SIB\_PORT\_0, 420  
CY\_U3P\_SIB\_PORT\_1, 420  
CY\_U3P\_SIB\_VOLTAGE\_LOW, 419  
CY\_U3P\_SIB\_VOLTAGE\_NORMAL, 419  
CyU3PSdioAbortFunctionIO, 420  
CyU3PSdioByteReadWrite, 421  
CyU3PSdioCardRegs\_t, 415  
CyU3PSdioCardReset, 421  
CyU3PSdioDirectReadWrite, 422  
CyU3PSdioExtendedReadWrite, 422  
CyU3PSdioGetBlockSize, 423  
CyU3PSdioGetCISAddress, 424  
CyU3PSdioGetTuples, 424  
CyU3PSdioInterruptControl, 425

CyU3PSdioQueryCard, 425  
 CyU3PSdioReadWaitEnable, 426  
 CyU3PSdioSetBlockSize, 426  
 CyU3PSdioSuspendResumeFunction, 427  
 CyU3PSibAbortRequest, 427  
 CyU3PSibCardDetect, 415, 418  
 CyU3PSibCommitReadWrite, 427  
 CyU3PSibDeInit, 428  
 CyU3PSibDevInfo\_t, 415  
 CyU3PSibDevRegType, 416, 418  
 CyU3PSibDevType, 416, 418  
 CyU3PSibEraseBlocks, 428  
 CyU3PSibEraseMode, 416, 418  
 CyU3PSibEventType, 416, 419  
 CyU3PSibEvtCbk\_t, 416  
 CyU3PSibForceErase, 429  
 CyU3PSibGetCSD, 429  
 CyU3PSibGetCardStatus, 429  
 CyU3PSibGetMMCExtCsd, 430  
 CyU3PSibInit, 430  
 CyU3PSibIntfParams\_t, 416  
 CyU3PSibIntfVoltage, 416, 419  
 CyU3PSibLockUnlockCard, 431  
 CyU3PSibLunInfo\_t, 417  
 CyU3PSibLunLocation, 417, 419  
 CyU3PSibLunType, 417, 419  
 CyU3PSibOpFreq, 417, 420  
 CyU3PSibPartitionStorage, 431  
 CyU3PSibPortId, 417, 420  
 CyU3PSibQueryDevice, 432  
 CyU3PSibQueryUnit, 432  
 CyU3PSibReadRegister, 433  
 CyU3PSibReadWriteRequest, 433  
 CyU3PSibRegisterCbk, 434  
 CyU3PSibRemovePartitions, 434  
 CyU3PSibRemovePasswd, 434  
 CyU3PSibSendSwitchCommand, 435  
 CyU3PSibSetBlockLen, 435  
 CyU3PSibSetIntfParams, 436  
 CyU3PSibSetPasswd, 436  
 CyU3PSibSetWriteCommitSize, 436  
 CyU3PSibStart, 437  
 CyU3PSibStop, 437  
 CyU3PSibVendorAccess, 437  
 CyU3PSibWriteTimerModify, 438  
**cyu3sibpp.h**  
 CY\_U3P\_BOOT\_PARTITION\_NUM\_BLKS, 440  
 CY\_U3P\_PARTITION\_TYPE\_AP\_BOOT, 440  
 CY\_U3P\_PARTITION\_TYPE\_BENICIA\_BOOT,  
     440  
 CY\_U3P\_PARTITION\_TYPE\_DATA\_AREA, 440  
 CY\_U3P\_SIB\_DEV\_PARTITION\_0, 442  
 CY\_U3P\_SIB\_DEV\_PARTITION\_1, 442  
 CY\_U3P\_SIB\_DEV\_PARTITION\_2, 442  
 CY\_U3P\_SIB\_DEV\_PARTITION\_3, 442  
 CY\_U3P\_SIB\_EVT\_ABORT, 441  
 CY\_U3P\_SIB\_EVT\_DRVENTRY, 441  
 CY\_U3P\_SIB\_EVT\_PORT\_0, 441  
 CY\_U3P\_SIB\_EVT\_PORT\_1, 441  
 CY\_U3P\_SIB\_EVT\_PORT\_POS, 441  
 CY\_U3P\_SIB\_NUM\_PARTITIONS, 442  
 CY\_U3P\_SIB\_SOCKET0, 443  
 CY\_U3P\_SIB\_SOCKET1, 443  
 CY\_U3P\_SIB\_SOCKET2, 443  
 CY\_U3P\_SIB\_SOCKET3, 443  
 CY\_U3P\_SIB\_SOCKET4, 443  
 CY\_U3P\_SIB\_SOCKET5, 443  
 CY\_U3P\_SIB\_SOCKET\_6, 443  
 CY\_U3P\_SIB\_SOCKET\_7, 443  
 CY\_U3P\_SIB\_STACK\_SIZE, 441  
 CY\_U3P\_SIB\_THREAD\_PRIORITY, 441  
 CY\_U3P\_SIB\_TIMER0\_EVT, 441  
 CY\_U3P\_SIB\_TIMER1\_EVT, 441  
 CYU3P\_SIB\_INT\_READ\_SOCKET, 441  
 CYU3P\_SIB\_INT\_WRITE\_SOCKET, 441  
 CyU3PSibDevPartition, 442  
 CyU3PSibDisableCoreIntr, 441  
 CyU3PSibEnableCoreIntr, 442  
 CyU3PSibGlobalData, 442  
 CyU3PSibInitCard, 443  
 CyU3PSibSocketId\_t, 442  
 CyU3PSibUpdateLunInfo, 443  
 gISibCtx, 443  
 gISibDeviceInfo, 443  
 gISibEvent, 444  
 gISibEvtCbk, 444  
 gISibIntfParams, 444  
 gISibLunInfo, 444  
 gISibThread, 444  
**cyu3socket.h**  
 CY\_U3P\_CPU\_IP\_BLOCK\_ID, 447  
 CY\_U3P\_LPP\_IP\_BLOCK\_ID, 447  
 CY\_U3P\_NUM\_IP\_BLOCK\_ID, 447  
 CY\_U3P\_PIB\_IP\_BLOCK\_ID, 447  
 CY\_U3P\_SIB\_IP\_BLOCK\_ID, 447  
 CY\_U3P\_UIB\_IP\_BLOCK\_ID, 447  
 CY\_U3P\_UIBIN\_IP\_BLOCK\_ID, 447  
 CyU3PBlockId\_t, 446, 447  
 CyU3PDmaGetSckId, 446  
 CyU3PDmaSocket\_t, 446  
 CyU3PDmaSocketCallback\_t, 446  
 CyU3PDmaSocketClearInterrupts, 447  
 CyU3PDmaSocketConfig\_t, 447  
 CyU3PDmaSocketDisable, 448  
 CyU3PDmaSocketEnable, 448  
 CyU3PDmaSocketGetConfig, 448  
 CyU3PDmaSocketIsValid, 449  
 CyU3PDmaSocketIsValidConsumer, 449  
 CyU3PDmaSocketIsValidProducer, 450  
 CyU3PDmaSocketRegisterCallback, 450  
 CyU3PDmaSocketSendEvent, 450  
 CyU3PDmaSocketSetConfig, 451  
 CyU3PDmaSocketSetWrapUp, 451  
 CyU3PDmaUpdateSocketResume, 452  
 CyU3PDmaUpdateSocketSuspendOption, 452  
**cyu3spi.h**

CY\_U3P\_SPI\_ERROR\_NONE, 456  
CY\_U3P\_SPI\_ERROR\_RX\_UNDERFLOW, 456  
CY\_U3P\_SPI\_ERROR\_TX\_OVERFLOW, 456  
CY\_U3P\_SPI\_EVENT\_ERROR, 456  
CY\_U3P\_SPI\_EVENT\_RX\_DONE, 456  
CY\_U3P\_SPI\_EVENT\_TX\_DONE, 456  
CY\_U3P\_SPI\_NUM\_SSN\_CTRL, 457  
CY\_U3P\_SPI\_NUM\_SSN\_LAG LEAD, 457  
CY\_U3P\_SPI\_SSN\_CTRL\_FW, 457  
CY\_U3P\_SPI\_SSN\_CTRL\_HW\_CPHA\_BASED,  
    457  
CY\_U3P\_SPI\_SSN\_CTRL\_HW\_EACH\_WORD,  
    457  
CY\_U3P\_SPI\_SSN\_CTRL\_HW\_END\_OF\_XFER,  
    457  
CY\_U3P\_SPI\_SSN\_CTRL\_NONE, 457  
CY\_U3P\_SPI\_SSN\_LAG LEAD\_HALF\_CLK, 457  
CY\_U3P\_SPI\_SSN\_LAG LEAD\_ONE\_CLK, 457  
CY\_U3P\_SPI\_SSN\_LAG LEAD\_ONE\_HALF\_C  
    LK, 457  
CY\_U3P\_SPI\_SSN\_LAG LEAD\_ZERO\_CLK, 457  
CyU3PRegisterSpiCallBack, 458  
CyU3PSpiConfig\_t, 454  
CyU3PSpiDeInit, 458  
CyU3PSpiDisableBlockXfer, 458  
CyU3PSpiError\_t, 455, 456  
CyU3PSpiEvt\_t, 455, 456  
CyU3PSpiInit, 459  
CyU3PSpiIntrCb\_t, 455  
CyU3PSpiReceiveWords, 459  
CyU3PSpiSetBlockXfer, 460  
CyU3PSpiSetConfig, 460  
CyU3PSpiSetSsnLine, 461  
CyU3PSpiSetTimeout, 461  
CyU3PSpiSsnCtrl\_t, 455, 456  
CyU3PSpiSsnLagLead\_t, 456, 457  
CyU3PSpiTransferWords, 462  
CyU3PSpiTransmitWords, 462  
CyU3PSpiWaitForBlockXfer, 463  
cyu3system.h  
    CY\_U3P\_DS\_FULL\_STRENGTH, 469  
    CY\_U3P\_DS\_HALF\_STRENGTH, 469  
    CY\_U3P\_DS\_QUARTER\_STRENGTH, 469  
    CY\_U3P\_DS\_THREE\_QUARTER\_STRENGTH,  
        469  
    CY\_U3P\_LPP\_GPIO, 469  
    CY\_U3P\_LPP\_I2C, 469  
    CY\_U3P\_LPP\_I2S, 469  
    CY\_U3P\_LPP\_SPI, 469  
    CY\_U3P\_LPP\_UART, 469  
    CY\_U3P\_NUM\_CLK\_SRC, 470  
    CY\_U3P\_SYS\_CLK\_BY\_16, 470  
    CY\_U3P\_SYS\_CLK\_BY\_2, 470  
    CY\_U3P\_SYS\_CLK\_BY\_4, 470  
    CY\_U3P\_SYS\_CLK, 470  
    CY\_U3P\_THREAD\_ID\_DEBUG, 470  
    CY\_U3P\_THREAD\_ID\_DMA, 470  
    CY\_U3P\_THREAD\_ID\_INT, 470  
            CY\_U3P\_THREAD\_ID\_LIB\_MAX, 470  
            CY\_U3P\_THREAD\_ID\_LPP, 470  
            CY\_U3P\_THREAD\_ID\_PIB, 470  
            CY\_U3P\_THREAD\_ID\_SYSTEM, 470  
            CY\_U3P\_THREAD\_ID\_UIB, 470  
            CyFxApplicationDefine, 471  
            CyU3PDebugDeInit, 471  
            CyU3PDebugDisable, 471  
            CyU3PDebugEnable, 471  
            CyU3PDebugInit, 472  
            CyU3PDebugLog, 472  
            CyU3PDebugLog\_t, 467  
            CyU3PDebugLogClear, 473  
            CyU3PDebugLogFlush, 473  
            CyU3PDebugPreamble, 473  
            CyU3PDebugPrint, 474  
            CyU3PDebugSetTimeout, 474  
            CyU3PDebugSetTraceLevel, 475  
            CyU3PDebugStringPrint, 475  
            CyU3PDebugSysMemDeInit, 476  
            CyU3PDebugSysMemInit, 476  
            CyU3PDeviceCacheControl, 476  
            CyU3PDeviceConfigureIOMatrix, 477  
            CyU3PDeviceGetCpuLoad, 478  
            CyU3PDeviceGetDriverLoad, 478  
            CyU3PDeviceGetPartNumber, 478  
            CyU3PDeviceGetSysClkFreq, 479  
            CyU3PDeviceGetThreadLoad, 479  
            CyU3PDeviceGpioOverride, 479  
            CyU3PDeviceGpioRestore, 480  
            CyU3PDeviceInit, 480  
            CyU3PDeviceReset, 480  
            CyU3PDriveStrengthState\_t, 467, 469  
            CyU3PFirmwareEntry, 481  
            CyU3PIsGpioComplexIOConfigured, 481  
            CyU3PIsGpioSimpleIOConfigured, 481  
            CyU3PIsGpioValid, 482  
            CyU3PIsLppIOConfigured, 482  
            CyU3PJumpToAddress, 483  
            CyU3PLppModule\_t, 467, 469  
            CyU3PSetSerialIoDriveStrength, 483  
            CyU3PSysCheckStandbyParam, 483  
            CyU3PSysCheckSuspendParams, 483  
            CyU3PSysClockConfig\_t, 468  
            CyU3PSysClockSrc\_t, 468, 469  
            CyU3PSysEnterStandbyMode, 484  
            CyU3PSysEnterSuspendMode, 484  
            CyU3PSysGetApiVersion, 485  
            CyU3PSysSendEnterSuspendStatus, 486  
            CyU3PSysThreadId\_t, 468, 470  
            CyU3PSysWatchDogClear, 486  
            CyU3PSysWatchDogConfigure, 486  
            CyU3PSystemRegisterDriver, 486  
            CyU3PToolChainInit, 487  
cyu3types.h  
    CyBool\_t, 488  
    CyFalse, 488  
    CyTrue, 488

CyU3PReturnStatus\_t, 488  
 uint16\_t, 488  
 uint32\_t, 488  
 uint8\_t, 488  
 cyu3uart.h  
   CY\_U3P\_UART\_BAUDRATE\_100, 492  
   CY\_U3P\_UART\_BAUDRATE\_10000, 493  
   CY\_U3P\_UART\_BAUDRATE\_100000, 493  
   CY\_U3P\_UART\_BAUDRATE\_115200, 493  
   CY\_U3P\_UART\_BAUDRATE\_1200, 492  
   CY\_U3P\_UART\_BAUDRATE\_14400, 493  
   CY\_U3P\_UART\_BAUDRATE\_153600, 493  
   CY\_U3P\_UART\_BAUDRATE\_19200, 493  
   CY\_U3P\_UART\_BAUDRATE\_1M, 493  
   CY\_U3P\_UART\_BAUDRATE\_200000, 493  
   CY\_U3P\_UART\_BAUDRATE\_225000, 493  
   CY\_U3P\_UART\_BAUDRATE\_230400, 493  
   CY\_U3P\_UART\_BAUDRATE\_2400, 493  
   CY\_U3P\_UART\_BAUDRATE\_2M, 493  
   CY\_U3P\_UART\_BAUDRATE\_300, 492  
   CY\_U3P\_UART\_BAUDRATE\_300000, 493  
   CY\_U3P\_UART\_BAUDRATE\_38400, 493  
   CY\_U3P\_UART\_BAUDRATE\_3M, 493  
   CY\_U3P\_UART\_BAUDRATE\_400000, 493  
   CY\_U3P\_UART\_BAUDRATE\_460800, 493  
   CY\_U3P\_UART\_BAUDRATE\_4800, 493  
   CY\_U3P\_UART\_BAUDRATE\_4M608K, 493  
   CY\_U3P\_UART\_BAUDRATE\_4M, 493  
   CY\_U3P\_UART\_BAUDRATE\_50000, 493  
   CY\_U3P\_UART\_BAUDRATE\_500000, 493  
   CY\_U3P\_UART\_BAUDRATE\_57600, 493  
   CY\_U3P\_UART\_BAUDRATE\_600, 492  
   CY\_U3P\_UART\_BAUDRATE\_75000, 493  
   CY\_U3P\_UART\_BAUDRATE\_750000, 493  
   CY\_U3P\_UART\_BAUDRATE\_921600, 493  
   CY\_U3P\_UART\_BAUDRATE\_9600, 493  
   CY\_U3P\_UART\_ERROR\_NAK\_BYTE\_0, 494  
   CY\_U3P\_UART\_ERROR\_RX\_OVERFLOW, 494  
   CY\_U3P\_UART\_ERROR\_RX\_PARITY\_ERROR,  
     494  
   CY\_U3P\_UART\_ERROR\_RX\_UNDERFLOW, 494  
   CY\_U3P\_UART\_ERROR\_TX\_OVERFLOW, 494  
   CY\_U3P\_UART\_EVENT\_ERROR, 494  
   CY\_U3P\_UART\_EVENT\_RX\_DATA, 494  
   CY\_U3P\_UART\_EVENT\_RX\_DONE, 494  
   CY\_U3P\_UART\_EVENT\_TX\_DONE, 494  
   CY\_U3P\_UART\_NO\_PARITY, 494  
   CY\_U3P\_UART\_NUM\_PARITY, 494  
   CY\_U3P\_UART\_ODD\_PARITY, 494  
   CY\_U3P\_UART\_ONE\_STOP\_BIT, 495  
   CY\_U3P\_UART\_TWO\_STOP\_BIT, 495  
 CyU3PRegisterUartCallBack, 495  
 CyU3PUartBaudrate\_t, 491, 492  
 CyU3PUartConfig\_t, 491  
 CyU3PUartDelinit, 495  
 CyU3PUartError\_t, 491, 493  
 CyU3PUartEvt\_t, 491, 494  
 CyU3PUartInit, 495  
 CyU3PUartIntrCb\_t, 491  
 CyU3PUartParity\_t, 492, 494  
 CyU3PUartReceiveBytes, 496  
 CyU3PUartRxSetBlockXfer, 496  
 CyU3PUartSetConfig, 497  
 CyU3PUartSetTimeout, 497  
 CyU3PUartStopBit\_t, 492, 494  
 CyU3PUartTransmitBytes, 498  
 CyU3PUartTxSetBlockXfer, 498  
 cyu3usb.h  
   CY\_U3P\_FULL\_SPEED, 512  
   CY\_U3P\_HIGH\_SPEED, 512  
   CY\_U3P\_MAX\_STRING\_DESC\_INDEX, 504  
   CY\_U3P\_NOT\_CONNECTED, 512  
   CY\_U3P\_SUPER\_SPEED, 512  
   CY\_U3P\_USB\_CONFIGURED, 511  
   CY\_U3P\_USB\_CONNECTED, 511  
   CY\_U3P\_USB\_ESTABLISHED, 511  
   CY\_U3P\_USB\_EVENT\_CONNECT, 509  
   CY\_U3P\_USB\_EVENT\_DISCONNECT, 509  
   CY\_U3P\_USB\_EVENT\_EP0\_STAT\_CPLT, 509  
   CY\_U3P\_USB\_EVENT\_EP\_UNDERRUN, 510  
   CY\_U3P\_USB\_EVENT\_HOST\_CONNECT, 510  
   CY\_U3P\_USB\_EVENT\_HOST\_DISCONNECT,  
     510  
   CY\_U3P\_USB\_EVENT\_LMP\_EXCH\_FAIL, 510  
   CY\_U3P\_USB\_EVENT\_LNK\_RECOVERY, 510  
   CY\_U3P\_USB\_EVENT\_OTG\_CHANGE, 510  
   CY\_U3P\_USB\_EVENT\_OTG\_SR\_P, 510  
   CY\_U3P\_USB\_EVENT\_OTG\_VBUS\_CHG, 510  
   CY\_U3P\_USB\_EVENT\_RESERVED\_1, 510  
   CY\_U3P\_USB\_EVENT\_RESET, 509  
   CY\_U3P\_USB\_EVENT\_RESUME, 509  
   CY\_U3P\_USB\_EVENT\_SET\_SEL, 509  
   CY\_U3P\_USB\_EVENT\_SETCONF, 509  
   CY\_U3P\_USB\_EVENT\_SETINTF, 509  
   CY\_U3P\_USB\_EVENT\_SOF\_ITP, 509  
   CY\_U3P\_USB\_EVENT\_SPEED, 509  
   CY\_U3P\_USB\_EVENT\_SS\_COMP\_ENTRY, 510  
   CY\_U3P\_USB\_EVENT\_SS\_COMP\_EXIT, 510  
   CY\_U3P\_USB\_EVENT\_SUSPEND, 509  
   CY\_U3P\_USB\_EVENT\_USB3\_LNKFAIL, 510  
   CY\_U3P\_USB\_EVENT\_VBUS\_REMOVED, 509  
   CY\_U3P\_USB\_EVENT\_VBUS\_VALID, 509  
   CY\_U3P\_USB\_INACTIVE, 511  
   CY\_U3P\_USB\_MAX\_STATE, 511  
   CY\_U3P\_USB\_MS\_ACTIVE, 511  
   CY\_U3P\_USB\_PROP\_DEVADDR, 508  
   CY\_U3P\_USB\_PROP\_FRAMECNT, 508  
   CY\_U3P\_USB\_PROP\_ITPINFO, 508  
   CY\_U3P\_USB\_PROP\_LINKSTATE, 508  
   CY\_U3P\_USB\_PROP\_SYS\_EXIT\_LAT, 508  
   CY\_U3P\_USB\_SET\_DEVQUAL\_DESCR, 511  
   CY\_U3P\_USB\_SET\_FS\_CONFIG\_DESCR, 511  
   CY\_U3P\_USB\_SET\_HS\_CONFIG\_DESCR, 511  
   CY\_U3P\_USB\_SET\_HS\_DEVICE\_DESCR, 511  
   CY\_U3P\_USB\_SET\_OTG\_DESCR, 511

CY\_U3P\_USB\_SET\_SS\_BOS\_DESCR, 511  
CY\_U3P\_USB\_SET\_SS\_CONFIG\_DESCR, 511  
CY\_U3P\_USB\_SET\_SS\_DEVICE\_DESCR, 511  
CY\_U3P\_USB\_SET\_STRING\_DESCR, 511  
CY\_U3P\_USB\_STARTED, 511  
CY\_U3P\_USB\_VBUS\_WAIT, 511  
CY\_U3P\_USB\_WAITING\_FOR\_DESC, 511  
CYU3P\_USBEP\_ISOERR\_EVT, 509  
CYU3P\_USBEP\_NAK\_EVT, 508  
CYU3P\_USBEP\_SLP\_EVT, 508  
CYU3P\_USBEP\_SS\_BTERM\_EVT, 509  
CYU3P\_USBEP\_SS\_RESET\_EVT, 509  
CYU3P\_USBEP\_SS\_RETRY\_EVT, 509  
CYU3P\_USBEP\_SS\_SEQERR\_EVT, 509  
CYU3P\_USBEP\_SS\_STREAMERR\_EVT, 509  
CYU3P\_USBEP\_ZLP\_EVT, 508  
CyU3PConnectState, 512  
CyU3PEpConfig\_t, 505  
CyU3PGetConnectState, 512  
CyU3PSetEpConfig, 512  
CyU3PSetEpPacketSize, 513  
CyU3PUSBEventCb\_t, 506  
CyU3PUSBSetDescType\_t, 507, 511  
CyU3PUSBSetupCb\_t, 507  
CyU3PUSBspeed\_t, 511  
CyU3PUibCheckConnection, 514  
CyU3PUsb2Resume, 514  
CyU3PUsbAckSetup, 514  
CyU3PUsbChangeMapping, 515  
CyU3PUsbControlUsb2Support, 515  
CyU3PUsbControlVBusDetect, 516  
CyU3PUsbDevProperty, 505, 508  
CyU3PUsbDoRemoteWakeup, 516  
CyU3PUsbEPSetBurstMode, 518  
CyU3PUsbEnableEPPPrefetch, 516  
CyU3PUsbEnableITPEvent, 517  
CyU3PUsbEpEvtCb\_t, 505  
CyU3PUsbEpEvtControl, 517  
CyU3PUsbEpEvtType, 505, 508  
CyU3PUsbEpPrepare, 518  
CyU3PUsbEpSetPacketsPerBuffer, 519  
CyU3PUsbEventType\_t, 506, 509  
CyU3PUsbFlushEp, 519  
CyU3PUsbForceFullSpeed, 520  
CyU3PUsbGetBooterVersion, 520  
CyU3PUsbGetDevProperty, 520  
CyU3PUsbGetEP0Data, 521  
CyU3PUsbGetEpCfg, 522  
CyU3PUsbGetEpSeqNum, 522  
CyU3PUsbGetErrorCounts, 522  
CyU3PUsbGetEventLogIndex, 523  
CyU3PUsbGetLinkPowerState, 523  
CyU3PUsbGetSpeed, 524  
CyU3PUsbInitEventLog, 524  
CyU3PUsbIsStarted, 524  
CyU3PUsbJumpBackToBooster, 525  
CyU3PUsbLPM\_COMP, 510  
CyU3PUsbLPM\_U0, 510  
CyU3PUsbLPM\_U1, 510  
CyU3PUsbLPM\_U2, 510  
CyU3PUsbLPM\_U3, 510  
CyU3PUsbLPM\_Unknown, 510  
CyU3PUsbLPMDisable, 525  
CyU3PUsbLPMEnable, 526  
CyU3PUsbLPMReqCb\_t, 506  
CyU3PUsbLinkComplianceControl, 525  
CyU3PUsbLinkPowerMode, 506, 510  
CyU3PUsbMapStream, 526  
CyU3PUsbMgrStates\_t, 507, 510  
CyU3PUsbRegisterEpEvtCallback, 527  
CyU3PUsbRegisterEventCallback, 527  
CyU3PUsbRegisterLPMRequestCallback, 528  
CyU3PUsbRegisterSetupCallback, 528  
CyU3PUsbResetEndpointMemories, 528  
CyU3PUsbResetEp, 529  
CyU3PUsbSSCDisable, 536  
CyU3PUsbSendAckTP, 529  
CyU3PUsbSendDevNotification, 530  
CyU3PUsbSendEP0Data, 530  
CyU3PUsbSendErdy, 531  
CyU3PUsbSendNrdy, 531  
CyU3PUsbSetBooterSwitch, 532  
CyU3PUsbSetDesc, 532  
CyU3PUsbSetEpNak, 533  
CyU3PUsbSetEpPktMode, 533  
CyU3PUsbSetEpSeqNum, 533  
CyU3PUsbSetEpSuspDisableMask, 534  
CyU3PUsbSetLinkPowerState, 534  
CyU3PUsbSetTxDeemphasis, 535  
CyU3PUsbSetTxSwing, 535  
CyU3PUsbSetXfer, 535  
CyU3PUsbStall, 536  
CyU3PUsbStart, 536  
CyU3PUsbStop, 537  
CyU3PUsbVBattEnable, 537  
cyu3usbconst.h  
CY\_U3P\_BOS\_DESCR, 542, 543  
CY\_U3P\_CONTAINER\_ID\_CAPB\_TYPE, 543  
CY\_U3P\_DEVICE\_CAPB\_DESCR, 542, 543  
CY\_U3P\_SS\_EP\_COMPN\_DESCR, 542, 543  
CY\_U3P\_SS\_USB\_CAPB\_TYPE, 543  
CY\_U3P\_UIB\_LNK\_STATE\_COMP, 544  
CY\_U3P\_UIB\_LNK\_STATE\_POLLING\_ACT, 544  
CY\_U3P\_UIB\_LNK\_STATE\_POLLING\_IDLE, 544  
CY\_U3P\_UIB\_LNK\_STATE\_POLLING\_LFPS, 544  
CY\_U3P\_UIB\_LNK\_STATE\_POLLING\_RXEQ,  
544  
CY\_U3P\_UIB\_LNK\_STATE\_RECov\_ACT, 544  
CY\_U3P\_UIB\_LNK\_STATE\_RECov\_Cnfg, 544  
CY\_U3P\_UIB\_LNK\_STATE\_RECov\_Idle, 544  
CY\_U3P\_UIB\_LNK\_STATE\_RXDETECT\_ACT,  
544  
CY\_U3P\_UIB\_LNK\_STATE\_RXDETECT\_Qut,  
544  
CY\_U3P\_UIB\_LNK\_STATE\_RXDETECT\_Res,  
544

CY\_U3P\_UIB\_LNK\_STATE\_SSDISABLED, 544  
 CY\_U3P\_UIB\_LNK\_STATE\_SSINACT\_DET, 544  
 CY\_U3P\_UIB\_LNK\_STATE\_SSINACT\_QUT, 544  
 CY\_U3P\_UIB\_LNK\_STATE\_U0, 544  
 CY\_U3P\_UIB\_LNK\_STATE\_U1, 544  
 CY\_U3P\_UIB\_LNK\_STATE\_U2, 544  
 CY\_U3P\_UIB\_LNK\_STATE\_U3, 544  
 CY\_U3P\_USB2\_EXTN\_CAPB\_TYPE, 543  
 CY\_U3P\_USB2\_FS\_REMOTE\_WAKE, 543  
 CY\_U3P\_USB2\_FS\_TEST\_MODE, 543  
 CY\_U3P\_USB2\_OTG\_A\_HNP\_SUPPORT, 543  
 CY\_U3P\_USB2\_OTG\_B\_HNP\_ENABLE, 543  
 CY\_U3P\_USB2\_TEST\_FORCE\_EN, 541  
 CY\_U3P\_USB2\_TEST\_NONE, 541  
 CY\_U3P\_USB2\_TEST\_PACKET, 541  
 CY\_U3P\_USB2\_TEST\_SE0\_NAK, 541  
 CY\_U3P\_USB2\_TEST\_J, 541  
 CY\_U3P\_USB2\_TEST\_K, 541  
 CY\_U3P\_USB3\_FS\_LTM\_ENABLE, 543  
 CY\_U3P\_USB3\_FS\_U1\_ENABLE, 543  
 CY\_U3P\_USB3\_FS\_U2\_ENABLE, 543  
 CY\_U3P\_USB3\_PACK\_TYPE\_DPH, 541  
 CY\_U3P\_USB3\_PACK\_TYPE\_ITP, 541  
 CY\_U3P\_USB3\_PACK\_TYPE\_LMP, 541  
 CY\_U3P\_USB3\_PACK\_TYPE\_TP, 541  
 CY\_U3P\_USB3\_TP\_SUBTYPE\_ACK, 542  
 CY\_U3P\_USB3\_TP\_SUBTYPE\_ERDY, 542  
 CY\_U3P\_USB3\_TP\_SUBTYPE\_NOTICE, 542  
 CY\_U3P\_USB3\_TP\_SUBTYPE\_NRDY, 542  
 CY\_U3P\_USB3\_TP\_SUBTYPE\_PINGRSP, 542  
 CY\_U3P\_USB3\_TP\_SUBTYPE\_PING, 542  
 CY\_U3P\_USB3\_TP\_SUBTYPE\_RES, 542  
 CY\_U3P\_USB3\_TP\_SUBTYPE\_STALL, 542  
 CY\_U3P\_USB3\_TP\_SUBTYPE\_STATUS, 542  
 CY\_U3P\_USB\_CONFIG\_DESCR, 542  
 CY\_U3P\_USB\_DEVICE\_DESCR, 542  
 CY\_U3P\_USB\_DEVQUAL\_DESCR, 542  
 CY\_U3P\_USB\_ENDPNT\_DESCR, 542  
 CY\_U3P\_USB\_EP\_BULK, 543  
 CY\_U3P\_USB\_EP\_CONTROL, 543  
 CY\_U3P\_USB\_EP\_INTR, 543  
 CY\_U3P\_USB\_EP\_ISO, 543  
 CY\_U3P\_USB\_HID\_DESCR, 542, 543  
 CY\_U3P\_USB\_INTRFC\_DESCR, 542  
 CY\_U3P\_USB\_INTRFC\_POWER\_DESCR, 542  
 CY\_U3P\_USB\_OTG\_DESCR, 542  
 CY\_U3P\_USB\_OTHERSPEED\_DESCR, 542  
 CY\_U3P\_USB\_REPORT\_DESCR, 542, 543  
 CY\_U3P\_USB\_SC\_CLEAR\_FEATURE, 544  
 CY\_U3P\_USB\_SC\_GET\_CONFIGURATION, 544  
 CY\_U3P\_USB\_SC\_GET\_DESCRIPTOR, 544  
 CY\_U3P\_USB\_SC\_GET\_INTERFACE, 544  
 CY\_U3P\_USB\_SC\_GET\_STATUS, 544  
 CY\_U3P\_USB\_SC\_RESERVED, 544  
 CY\_U3P\_USB\_SC\_SET\_ADDRESS, 544  
 CY\_U3P\_USB\_SC\_SET\_CONFIGURATION, 544  
 CY\_U3P\_USB\_SC\_SET\_DESCRIPTOR, 544  
 CY\_U3P\_USB\_SC\_SET\_FEATURE, 544  
 CY\_U3P\_USB\_SC\_SET\_INTERFACE, 544  
 CY\_U3P\_USB\_SC\_SET\_ISOC\_DELAY, 544  
 CY\_U3P\_USB\_SC\_SET\_SEL, 544  
 CY\_U3P\_USB\_SC\_SYNC\_FRAME, 544  
 CY\_U3P\_USB\_STRING\_DESCR, 542  
 CY\_U3P\_USBX\_FS\_EP\_HALT, 543  
 CY\_U3P\_WIRELESS\_USB\_CAPB\_TYPE, 543  
 CyU3PUsb2TestModes, 540, 541  
 CyU3PUsb3PacketType, 540, 541  
 CyU3PUsb3TpSubType, 540, 541  
 CyU3PUsbDescType, 540, 542  
 CyU3PUsbDevCapType, 540, 543  
 CyU3PUsbEpType\_t, 540, 543  
 CyU3PUsbFeatureSelector, 540, 543  
 CyU3PUsbLinkState\_t, 541, 543  
 CyU3PUsbSetupCmds, 541, 544  
 cyu3usbhost.h  
 CY\_U3P\_USB\_HOST\_EPXFER\_NORMAL, 550  
 CY\_U3P\_USB\_HOST\_EPXFER\_SETUP\_IN\_DATA, 550  
 CY\_U3P\_USB\_HOST\_EPXFER\_SETUP\_NO\_DATA, 550  
 CY\_U3P\_USB\_HOST\_EPXFER\_SETUP\_OUT\_DATA, 550  
 CY\_U3P\_USB\_HOST\_EVENT\_CONNECT, 550  
 CY\_U3P\_USB\_HOST\_EVENT\_DISCONNECT, 550  
 CY\_U3P\_USB\_HOST\_FULL\_SPEED, 550  
 CY\_U3P\_USB\_HOST\_HIGH\_SPEED, 550  
 CY\_U3P\_USB\_HOST\_LOW\_SPEED, 550  
 CyU3PUsbHostConfig\_t, 548  
 CyU3PUsbHostEp0BeginXfer, 550  
 CyU3PUsbHostEpAbort, 551  
 CyU3PUsbHostEpAdd, 551  
 CyU3PUsbHostEpConfig\_t, 548  
 CyU3PUsbHostEpRemove, 552  
 CyU3PUsbHostEpReset, 552  
 CyU3PUsbHostEpSetXfer, 552  
 CyU3PUsbHostEpStatus\_t, 548  
 CyU3PUsbHostEpWaitForCompletion, 553  
 CyU3PUsbHostEpXferType\_t, 548, 549  
 CyU3PUsbHostEventCb\_t, 548  
 CyU3PUsbHostEventType\_t, 549, 550  
 CyU3PUsbHostGetDeviceAddress, 554  
 CyU3PUsbHostGetFrameNumber, 554  
 CyU3PUsbHostGetPortStatus, 554  
 CyU3PUsbHostIsStarted, 555  
 CyU3PUsbHostOpSpeed\_t, 549, 550  
 CyU3PUsbHostPortDisable, 555  
 CyU3PUsbHostPortEnable, 555  
 CyU3PUsbHostPortReset, 556  
 CyU3PUsbHostPortResume, 556  
 CyU3PUsbHostPortStatus\_t, 549  
 CyU3PUsbHostPortSuspend, 556  
 CyU3PUsbHostSendSetupRqt, 557  
 CyU3PUsbHostSetDeviceAddress, 557  
 CyU3PUsbHostStart, 558  
 CyU3PUsbHostStop, 558

CyU3PUsbHostXferCb\_t, 549  
cyu3usbott.h  
CY\_U3P\_OTG\_CHARGER\_DETECT\_ACA\_MODE, 562  
CY\_U3P\_OTG\_CHARGER\_DETECT\_MOT\_EMU, 562  
CY\_U3P\_OTG\_CHARGER\_DETECT\_NUM\_MODES, 562  
CY\_U3P\_OTG\_MODE\_CARKIT\_PPOT, 563  
CY\_U3P\_OTG\_MODE\_CARKIT\_UART, 563  
CY\_U3P\_OTG\_MODE\_DEVICE\_ONLY, 563  
CY\_U3P\_OTG\_MODE\_HOST\_ONLY, 563  
CY\_U3P\_OTG\_MODE\_OTG, 563  
CY\_U3P\_OTG\_NUM\_MODES, 563  
CY\_U3P\_OTG\_PERIPHERAL\_CHANGE, 562  
CY\_U3P\_OTG\_SRPP\_DETECT, 562  
CY\_U3P\_OTG\_TYPE\_A\_CABLE, 563  
CY\_U3P\_OTG\_TYPE ACA\_A\_CHG, 563  
CY\_U3P\_OTG\_TYPE ACA\_B\_CHG, 563  
CY\_U3P\_OTG\_TYPE ACA\_C\_CHG, 563  
CY\_U3P\_OTG\_TYPE\_B\_CABLE, 563  
CY\_U3P\_OTG\_TYPE\_DISABLED, 563  
CY\_U3P\_OTG\_TYPE\_MOT\_CHG, 564  
CY\_U3P\_OTG\_TYPE\_MOT\_FAST, 564  
CY\_U3P\_OTG\_TYPE\_MOT\_MID, 564  
CY\_U3P\_OTG\_TYPE\_MOT\_MPX200, 563  
CY\_U3P\_OTG\_VBUS\_VALID\_CHANGE, 562  
CyU3POtgChargerDetectMode\_t, 560, 562  
CyU3POtgConfig\_t, 560  
CyU3POtgEvent\_t, 561, 562  
CyU3POtgEventCallback\_t, 561  
CyU3POtgGetMode, 564  
CyU3POtgGetPeripheralType, 564  
CyU3POtgHnpEnable, 564  
CyU3POtgIsDeviceMode, 566  
CyU3POtgIsHnpEnabled, 566  
CyU3POtgIsHostMode, 566  
CyU3POtgIsStarted, 567  
CyU3POtgIsVBusValid, 567  
CyU3POtgMode\_t, 561, 562  
CyU3POtgPeripheralType\_t, 561, 563  
CyU3POtgRequestHnp, 567  
CyU3POtgSrpAbort, 568  
CyU3POtgSrpStart, 568  
CyU3POtgStart, 568  
CyU3POtgStop, 569  
cyu3utils.h  
CY\_U3P\_MAKEDWORD, 571  
CyU3PBusyWait, 571  
CyU3PComputeChecksum, 571  
CyU3PMemCopy32, 572  
CyU3PReadDeviceRegisters, 572  
CyU3PWriteDeviceRegisters, 572  
cyu3vic.h  
CY\_U3P\_VIC\_BIAS\_CORRECT\_VECTOR, 574  
CY\_U3P\_VIC\_DEBUG\_RX\_VECTOR, 574  
CY\_U3P\_VIC\_DEBUG\_TX\_VECTOR, 574  
CY\_U3P\_VIC\_GCTL\_CORE\_VECTOR, 574  
CY\_U3P\_VIC\_GCTL\_PWR\_VECTOR, 575  
CY\_U3P\_VIC\_GPIO\_CORE\_VECTOR, 575  
CY\_U3P\_VIC\_I2C\_CORE\_VECTOR, 575  
CY\_U3P\_VIC\_I2S\_CORE\_VECTOR, 575  
CY\_U3P\_VIC\_LPP\_DMA\_VECTOR, 575  
CY\_U3P\_VIC\_NUM\_VECTORS, 575  
CY\_U3P\_VIC\_PIB\_CORE\_VECTOR, 574  
CY\_U3P\_VIC\_PIB\_DMA\_VECTOR, 574  
CY\_U3P\_VIC\_RESERVED\_15\_VECTOR, 575  
CY\_U3P\_VIC\_SIB0\_CORE\_VECTOR, 574  
CY\_U3P\_VIC\_SIB1\_CORE\_VECTOR, 574  
CY\_U3P\_VIC\_SIB\_DMA\_VECTOR, 574  
CY\_U3P\_VIC\_SPI\_CORE\_VECTOR, 575  
CY\_U3P\_VIC\_SWI\_VECTOR, 574  
CY\_U3P\_VIC\_UART\_CORE\_VECTOR, 575  
CY\_U3P\_VIC\_UIB\_CONTROL\_VECTOR, 574  
CY\_U3P\_VIC\_UIB\_CORE\_VECTOR, 574  
CY\_U3P\_VIC\_UIB\_DMA\_VECTOR, 574  
CY\_U3P\_VIC\_WDT\_VECTOR, 574  
CyU3PVicClearInt, 575  
CyU3PVicDisableAllInterrupts, 575  
CyU3PVicDisableInt, 575  
CyU3PVicEnableInt, 576  
CyU3PVicEnableInterrupts, 576  
CyU3PVicIRQGetStatus, 578  
CyU3PVicInit, 576  
CyU3PVicIntGetPriority, 577  
CyU3PVicIntGetStatus, 577  
CyU3PVicIntSetPriority, 577  
CyU3PVicSetupIntVectors, 578  
CyU3PVicVector\_t, 574  
  
dataFormat  
CyU3PMipicsiCfg\_t, 74  
dataTimeOut  
CyU3PCardCtxt, 40  
ddrMode  
CyU3PCardCtxt, 40  
CyU3PSibDevInfo, 82  
discardCount  
CyU3PDmaChannel, 46  
CyU3PDmaMultiChannel, 54  
dmaClkDiv  
CyU3PSysClockConfig\_t, 90  
dmaMode  
CyU3PDmaChannel, 46  
CyU3PDmaChannelConfig\_t, 49  
CyU3PDmaMultiChannel, 54  
CyU3PDmaMultiChannelConfig\_t, 57  
dmaTimeout  
CyFx3BootI2cConfig\_t, 30  
CyU3PI2cConfig\_t, 68  
driveHighEn  
CyFx3BootGpioSimpleConfig\_t, 29  
CyU3PGpioComplexConfig\_t, 66  
CyU3PGpioSimpleConfig\_t, 67  
driveLowEn  
CyFx3BootGpioSimpleConfig\_t, 29  
CyU3PGpioComplexConfig\_t, 66

CyU3PGpioSimpleConfig\_t, 67  
 dscrChain  
     CyFx3BootDmaSockRegs\_t, 28  
     CyFx3BootDmaSocket\_t, 26  
     CyU3PDmaSocket\_t, 58  
     CyU3PDmaSocketConfig\_t, 60  
  
 eidErrCnt  
     CyU3PMipicsiErrorCounts\_t, 76  
  
 enable  
     CyFx3BootUsbEpConfig\_t, 38  
     CyU3PEpConfig\_t, 61  
  
 endSig  
     CyU3PDmaChannel, 46  
     CyU3PDmaMultiChannel, 54  
  
 ep0LowLevelControl  
     CyU3PUusbHostConfig\_t, 93  
  
 epType  
     CyFx3BootUsbEpConfig\_t, 38  
     CyU3PEpConfig\_t, 61  
  
 eraseSize  
     CyU3PCardCtxt, 40  
     CyU3PSibDevInfo, 82  
  
 eventCb  
     CyU3PUusbHostConfig\_t, 93  
  
 FW\_WAIT\_SCK0  
     cyu3mipicsi.h, 325  
  
 FW\_WAIT\_SCK1  
     cyu3mipicsi.h, 325  
  
 fastClkDiv  
     CyU3PGpioClock\_t, 64  
  
 fifoDelay  
     CyU3PMipicsiCfg\_t, 74  
  
 firmware/boot\_fw/include/cyfx3device.h, 97  
  
 firmware/boot\_fw/include/cyfx3dma.h, 103  
  
 firmware/boot\_fw/include/cyfx3error.h, 114  
  
 firmware/boot\_fw/include/cyfx3gpio.h, 116  
  
 firmware/boot\_fw/include/cyfx3i2c.h, 121  
  
 firmware/boot\_fw/include/cyfx3i2c.h, 121  
  
 firmware/boot\_fw/include/cyfx3spi.h, 127  
  
 firmware/boot\_fw/include/cyfx3spi.h, 137  
  
 firmware/boot\_fw/include/cyfx3uart.h, 144  
  
 firmware/boot\_fw/include/cyfx3usb.h, 151  
  
 firmware/boot\_fw/include/cyfx3utils.h, 165  
  
 firmware/u3p\_firmware/inc/cyfx3\_api.h, 166  
  
 firmware/u3p\_firmware/inc/cyfxapidesc.h, 182  
  
 firmware/u3p\_firmware/inc/cyfxversion.h, 182  
  
 firmware/u3p\_firmware/inc/cyu3cardmgr.h, 183  
  
 firmware/u3p\_firmware/inc/cyu3cardmgr\_fx3s.h, 194  
  
 firmware/u3p\_firmware/inc/cyu3descriptor.h, 204  
  
 firmware/u3p\_firmware/inc/cyu3dma.h, 210  
  
 firmware/u3p\_firmware/inc/cyu3error.h, 253  
  
 firmware/u3p\_firmware/inc/cyu3gpif.h, 256  
  
 firmware/u3p\_firmware/inc/cyu3gpio.h, 272  
  
 firmware/u3p\_firmware/inc/cyu3i2c.h, 287  
  
 firmware/u3p\_firmware/inc/cyu3i2s.h, 298  
  
 firmware/u3p\_firmware/inc/cyu3lpp.h, 307  
  
 firmware/u3p\_firmware/inc/cyu3mbox.h, 317  
  
 firmware/u3p\_firmware/inc/cyu3mipicsi.h, 320  
  
                         firmware/u3p\_firmware/inc/cyu3mmu.h, 339  
                         firmware/u3p\_firmware/inc/cyu3os.h, 349  
                         firmware/u3p\_firmware/inc/cyu3pib.h, 395  
                         firmware/u3p\_firmware/inc/cyu3sib.h, 411  
                         firmware/u3p\_firmware/inc/cyu3sibpp.h, 439  
                         firmware/u3p\_firmware/inc/cyu3socket.h, 444  
                         firmware/u3p\_firmware/inc/cyu3spi.h, 453  
                         firmware/u3p\_firmware/inc/cyu3system.h, 463  
                         firmware/u3p\_firmware/inc/cyu3types.h, 487  
                         firmware/u3p\_firmware/inc/cyu3uart.h, 489  
                         firmware/u3p\_firmware/inc/cyu3usb.h, 499  
                         firmware/u3p\_firmware/inc/cyu3usbconst.h, 538  
                         firmware/u3p\_firmware/inc/cyu3usbhost.h, 545  
                         firmware/u3p\_firmware/inc/cyu3usbottg.h, 558  
                         firmware/u3p\_firmware/inc/cyu3utils.h, 569  
                         firmware/u3p\_firmware/inc/cyu3vic.h, 573  
  
 firstConsIndex  
     CyU3PDmaChannel, 46  
     CyU3PDmaMultiChannel, 54  
  
 firstProdIndex  
     CyU3PDmaChannel, 46  
     CyU3PDmaMultiChannel, 54  
  
 flags  
     CyU3PDmaChannel, 46  
     CyU3PDmaMultiChannel, 54  
  
 flowCtrl  
     CyFx3BootUartConfig\_t, 36  
     CyU3PUartConfig\_t, 91  
  
 fn0BlockSize  
     CyU3PSdioCardRegs, 79  
  
 frmErrCnt  
     CyU3PMipicsiErrorCounts\_t, 76  
  
 fullPktSize  
     CyU3PUusbHostEpConfig\_t, 94  
  
 functionCount  
     CyU3PGpifConfig\_t, 62  
  
 functionData  
     CyU3PGpifConfig\_t, 62  
  
 glDmaDescriptor  
     cyu3descriptor.h, 210  
  
 gIMBoxCb  
     cyu3mbox.h, 320  
  
 gISibCtx  
     cyu3sibpp.h, 443  
  
 gISibDevInfo  
     cyu3sibpp.h, 443  
  
 gISibEvent  
     cyu3sibpp.h, 444  
  
 gISibEvtCbk  
     cyu3sibpp.h, 444  
  
 gISibIntfParams  
     cyu3sibpp.h, 444  
  
 gISibLunInfo  
     cyu3sibpp.h, 444  
  
 gISibThread  
     cyu3sibpp.h, 444  
  
 gpioComplexEn  
     CyU3PIoMatrixConfig\_t, 72

gpioSimpleEn  
    CyFx3BootIoMatrixConfig\_t, 32  
    CyU3PloMatrixConfig\_t, 72

hResolution  
    CyU3PMipicsiCfg\_t, 74

halfDiv  
    CyU3PGpioClock\_t, 64

highCapacity  
    CyU3PCardCtxt, 40

inUse  
    CyU3PSibCtxt, 80

inputEn  
    CyFx3BootGpioSimpleConfig\_t, 29  
    CyU3PGpioComplexConfig\_t, 66  
    CyU3PGpioSimpleConfig\_t, 67

intr  
    CyFx3BootDmaSockRegs\_t, 28  
    CyFx3BootDmaSocket\_t, 26  
    CyU3PDmaSocket\_t, 58  
    CyU3PDmaSocketConfig\_t, 60

intrMask  
    CyFx3BootDmaSockRegs\_t, 28  
    CyFx3BootDmaSocket\_t, 26  
    CyU3PDmaSocket\_t, 58  
    CyU3PDmaSocketConfig\_t, 60

intrMode  
    CyFx3BootGpioSimpleConfig\_t, 29  
    CyU3PGpioComplexConfig\_t, 66  
    CyU3PGpioSimpleConfig\_t, 67

isActive  
    CyU3PSibGlobalData, 84

isDQ32Bit  
    CyFx3BootIoMatrixConfig\_t, 32  
    CyU3PloMatrixConfig\_t, 72

isDIIEnable  
    CyFx3BootPibClock\_t, 33  
    CyU3PPibClock\_t, 78

isDma  
    CyFx3BootI2cConfig\_t, 30  
    CyFx3BootUartConfig\_t, 36  
    CyU3PI2cConfig\_t, 68  
    CyU3PI2sConfig\_t, 70  
    CyU3PUuartConfig\_t, 91

isDmaHandleDCache  
    CyU3PDmaChannel, 47  
    CyU3PDmaMultiChannel, 54

isHalfDiv  
    CyFx3BootPibClock\_t, 34  
    CyU3PPibClock\_t, 78

isLsbFirst  
    CyFx3BootSpiConfig\_t, 35  
    CyU3PI2sConfig\_t, 70  
    CyU3PSpiConfig\_t, 88

isMemoryPresent  
    CyU3PSdioCardRegs, 79

isMono  
    CyU3PI2sConfig\_t, 70

isRead  
    CyU3PSibCtxt, 80

isStreamMode  
    CyU3PUsbHostEpConfig\_t, 94

isoPkts  
    CyFx3BootUsbEpConfig\_t, 38  
    CyU3PEpConfig\_t, 61

lagTime  
    CyFx3BootSpiConfig\_t, 35  
    CyU3PSpiConfig\_t, 88

leadTime  
    CyFx3BootSpiConfig\_t, 35  
    CyU3PSpiConfig\_t, 88

leftData  
    CyU3PGpifWaveData, 63

length  
    CyFx3BootI2cPreamble\_t, 31  
    CyU3PI2cPreamble\_t, 70

location  
    CyU3PSibLunInfo, 87

lock  
    CyU3PDmaChannel, 47  
    CyU3PDmaMultiChannel, 54

locked  
    CyU3PCardCtxt, 41  
    CyU3PSibDevInfo, 82

lowVoltage  
    CyU3PSibIntfParams, 85

lppMode  
    CyU3PloMatrixConfig\_t, 72

lvGpioState  
    CyU3PSibIntfParams, 85

mClkCtl  
    CyU3PMipicsiCfg\_t, 74

mClkRefDiv  
    CyU3PMipicsiCfg\_t, 74

manufacturerId  
    CyU3PSdioCardRegs, 79

manufacturerInfo  
    CyU3PSdioCardRegs, 79

maxFreq  
    CyU3PSibIntfParams, 85

maxPktSize  
    CyU3PUsbHostEpConfig\_t, 94

mdlErrCnt  
    CyU3PMipicsiErrorCounts\_t, 76

MemBlockInfo, 95  
    alloc\_id, 95  
    alloc\_size, 95  
    cyu3os.h, 360  
    next\_blk, 96  
    pad, 96  
    prev\_blk, 96  
    start\_sig, 96

mmioClkDiv  
    CyU3PSysClockConfig\_t, 90

msg

CyU3PDebugLog\_t, 42  
 mult  
   CyU3PUsbHostEpConfig\_t, 95  
 mutexLock  
   CyU3PSibCtx, 81  
  
 next\_blk  
   MemBlockInfo, 96  
 nextWrAddress  
   CyU3PSibGlobalData, 84  
 notification  
   CyU3PDmaChannel, 47  
   CyU3PDmaChannelConfig\_t, 49  
   CyU3PDmaMultiChannel, 54  
   CyU3PDmaMultiChannelConfig\_t, 57  
 numBlks  
   CyU3PCardCtxt, 41  
   CyU3PSibDevInfo, 83  
 numBlocks  
   CyU3PSibLunInfo, 87  
 numBootLuns  
   CyU3PSibCtx, 81  
 numDataLanes  
   CyU3PMipicsiCfg\_t, 74  
 numUnits  
   CyU3PSibDevInfo, 83  
 numUserLuns  
   CyU3PSibCtx, 81  
 numberOfFunctions  
   CyU3PSdioCardRegs, 79  
  
 ocrRegData  
   CyU3PCardCtxt, 41  
 opVoltage  
   CyU3PCardCtxt, 41  
   CyU3PSibDevInfo, 83  
 openWrSize  
   CyU3PSibGlobalData, 84  
 otgMode  
   CyU3POtgConfig\_t, 77  
 outValue  
   CyFx3BootGpioSimpleConfig\_t, 29  
   CyU3PGpioComplexConfig\_t, 66  
   CyU3PGpioSimpleConfig\_t, 67  
 overrideDscrIndex  
   CyU3PDmaChannel, 47  
   CyU3PDmaMultiChannel, 54  
  
 pData  
   CyFx3BootUsbEp0Pkt\_t, 37  
 pad  
   MemBlockInfo, 96  
 padMode  
   CyU3PI2sConfig\_t, 71  
 parClkDiv  
   CyU3PMipicsiCfg\_t, 75  
 param  
   CyU3PDebugLog\_t, 42  
 parity

CyFx3BootUartConfig\_t, 36  
 CyU3PUartConfig\_t, 91  
 partConfig  
   CyU3PSibGlobalData, 84  
 partition  
   CyU3PSibCtx, 81  
 pktsSize  
   CyFx3BootUsbEpConfig\_t, 38  
   CyU3PEpConfig\_t, 61  
 period  
   CyU3PGpioComplexConfig\_t, 66  
 pinMode  
   CyU3PGpioComplexConfig\_t, 66  
 plIFbd  
   CyU3PMipicsiCfg\_t, 75  
 plIFrs  
   CyU3PMipicsiCfg\_t, 75  
 plIPrd  
   CyU3PMipicsiCfg\_t, 75  
 pollingRate  
   CyU3PUsbHostEpConfig\_t, 95  
 prev\_blk  
   MemBlockInfo, 96  
 priority  
   CyU3PDebugLog\_t, 42  
 prodAvailCount  
   CyU3PDmaChannel, 47  
   CyU3PDmaChannelConfig\_t, 49  
   CyU3PDmaMultiChannel, 54  
   CyU3PDmaMultiChannelConfig\_t, 57  
 prodFooter  
   CyU3PDmaChannel, 47  
   CyU3PDmaChannelConfig\_t, 50  
   CyU3PDmaMultiChannel, 54  
   CyU3PDmaMultiChannelConfig\_t, 57  
 prodHeader  
   CyU3PDmaChannel, 47  
   CyU3PDmaChannelConfig\_t, 50  
   CyU3PDmaMultiChannel, 55  
   CyU3PDmaMultiChannelConfig\_t, 57  
 prodSckId  
   CyU3PDmaChannel, 47  
   CyU3PDmaChannelConfig\_t, 50  
   CyU3PDmaMultiChannel, 55  
   CyU3PDmaMultiChannelConfig\_t, 57  
 prodSusp  
   CyU3PDmaChannel, 47  
   CyU3PDmaMultiChannel, 55  
  
 recSyncErrCnt  
   CyU3PMipicsiErrorCounts\_t, 76  
 recrErrCnt  
   CyU3PMipicsiErrorCounts\_t, 76  
 regCount  
   CyU3PGpifConfig\_t, 62  
 regData  
   CyU3PGpifConfig\_t, 62  
 removable  
   CyU3PCardCtxt, 41

CyU3PSibDevInfo, 83  
resetGpio  
    CyU3PSibIntfParams, 86  
rightData  
    CyU3PGpifWaveData, 63  
rstActHigh  
    CyU3PSibIntfParams, 86  
rxEnable  
    CyFx3BootUartConfig\_t, 36  
    CyU3PUuartConfig\_t, 91  
  
s0Enabled  
    CyU3PSibGlobalData, 84  
s0Mode  
    CyU3PloMatrixConfig\_t, 72  
s1Enabled  
    CyU3PSibGlobalData, 84  
s1Mode  
    CyU3PloMatrixConfig\_t, 72  
sampleRate  
    CyU3PI2sConfig\_t, 71  
sampleWidth  
    CyU3PI2sConfig\_t, 71  
sckEvent  
    CyFx3BootDmaSockRegs\_t, 28  
    CyU3PDmaSocket\_t, 58  
sdioVersion  
    CyU3PSdioCardRegs, 79  
setSysClk400  
    CyU3PSysClockConfig\_t, 90  
sibDmaChannel  
    CyU3PSibGlobalData, 84  
sibEvtCbk  
    CyU3PSibGlobalData, 84  
simpleDiv  
    CyU3PGpioClock\_t, 64  
size  
    CyFx3BootDmaDescriptor\_t, 25  
    CyU3PDmaBuffer\_t, 43  
    CyU3PDmaChannel, 47  
    CyU3PDmaChannelConfig\_t, 50  
    CyU3PDmaDescriptor\_t, 51  
    CyU3PDmaMultiChannel, 55  
    CyU3PDmaMultiChannelConfig\_t, 57  
slowClkDiv  
    CyU3PGpioClock\_t, 65  
ssnCtrl  
    CyFx3BootSpiConfig\_t, 35  
    CyU3PSpiConfig\_t, 88  
ssnPol  
    CyFx3BootSpiConfig\_t, 35  
    CyU3PSpiConfig\_t, 88  
start\_sig  
    MemBlockInfo, 96  
startAddr  
    CyU3PSibLunInfo, 87  
startSig  
    CyU3PDmaChannel, 47  
    CyU3PDmaMultiChannel, 55  
  
state  
    CyU3PDmaChannel, 47  
    CyU3PDmaMultiChannel, 55  
stateCount  
    CyU3PGpifConfig\_t, 62  
stateData  
    CyU3PGpifConfig\_t, 63  
statePosition  
    CyU3PGpifConfig\_t, 63  
status  
    CyFx3BootDmaSockRegs\_t, 28  
    CyFx3BootDmaSocket\_t, 26  
    CyU3PDmaBuffer\_t, 43  
    CyU3PDmaSocket\_t, 59  
    CyU3PDmaSocketConfig\_t, 60  
    CyU3PSibCtxt, 81  
stopBit  
    CyFx3BootUartConfig\_t, 36  
    CyU3PUuartConfig\_t, 91  
streams  
    CyFx3BootUsbEpConfig\_t, 39  
    CyU3PEpConfig\_t, 61  
supportsAsyncIntr  
    CyU3PSdioCardRegs, 80  
sync  
    CyFx3BootDmaDescriptor\_t, 25  
    CyU3PDmaDescriptor\_t, 51  
  
threadId  
    CyU3PDebugLog\_t, 42  
threshold  
    CyU3PGpioComplexConfig\_t, 66  
timer  
    CyU3PGpioComplexConfig\_t, 66  
timerMode  
    CyU3PGpioComplexConfig\_t, 66  
txEnable  
    CyFx3BootUartConfig\_t, 36  
    CyU3PUuartConfig\_t, 91  
type  
    CyU3PDmaChannel, 48  
    CyU3PDmaMultiChannel, 55  
    CyU3PSibLunInfo, 87  
    CyU3PUusbHostEpConfig\_t, 95  
  
uhslOpMode  
    CyU3PCardCtxt, 41  
uhsSupport  
    CyU3PSdioCardRegs, 80  
unrSyncErrCnt  
    CyU3PMipicsiErrorCounts\_t, 76  
unrcErrCnt  
    CyU3PMipicsiErrorCounts\_t, 76  
unused19  
    CyFx3BootDmaSockRegs\_t, 28  
    CyU3PDmaSocket\_t, 59  
unused2  
    CyFx3BootDmaSockRegs\_t, 28  
    CyU3PDmaSocket\_t, 59

usbConfigDesc\_p  
     CyU3PUsbDescrPtrs, 92

usbConsSusp  
     CyU3PDmaChannel, 48  
     CyU3PDmaMultiChannel, 55

usbDevDesc\_p  
     CyU3PUsbDescrPtrs, 92

usbDevQualDesc\_p  
     CyU3PUsbDescrPtrs, 92

usbFSConfigDesc\_p  
     CyU3PUsbDescrPtrs, 92

usbHSConfigDesc\_p  
     CyU3PUsbDescrPtrs, 92

usbOtherSpeedConfigDesc\_p  
     CyU3PUsbDescrPtrs, 92

usbSSBOSDesc\_p  
     CyU3PUsbDescrPtrs, 92

usbSSConfigDesc\_p  
     CyU3PUsbDescrPtrs, 93

usbSSDevDesc\_p  
     CyU3PUsbDescrPtrs, 93

usbStringDesc\_p  
     CyU3PUsbDescrPtrs, 93

useDdr  
     CyU3PSibIntfParams, 86

useI2C  
     CyFx3BootIoMatrixConfig\_t, 32  
     CyU3PIoMatrixConfig\_t, 72

useI2S  
     CyFx3BootIoMatrixConfig\_t, 32  
     CyU3PIoMatrixConfig\_t, 72

useSpi  
     CyFx3BootIoMatrixConfig\_t, 33  
     CyU3PIoMatrixConfig\_t, 72

useStandbyClk  
     CyU3PSysClockConfig\_t, 90

useUart  
     CyFx3BootIoMatrixConfig\_t, 33  
     CyU3PIoMatrixConfig\_t, 72

uvint16\_t  
     cyu3types.h, 488

uvint32\_t  
     cyu3types.h, 488

uvint8\_t  
     cyu3types.h, 488

valid  
     CyU3PSibLunInfo, 87

validSckCount  
     CyU3PDmaMultiChannel, 55  
     CyU3PDmaMultiChannelConfig\_t, 57

voltageSwGpio  
     CyU3PSibIntfParams, 86

w0  
     CyU3PMbox, 73

w1  
     CyU3PMbox, 73

wLen

    CyFx3BootUsbEp0Pkt\_t, 38

wordLen  
     CyFx3BootSpiConfig\_t, 35  
     CyU3PSpiConfig\_t, 89

wrCommitPending  
     CyU3PSibGlobalData, 84

wrCommitSize  
     CyU3PSibGlobalData, 84

writeProtEnable  
     CyU3PSibIntfParams, 86

writeTimer  
     CyU3PSibCtx, 81

writeTimerCbk  
     CyU3PSibCtx, 81

writeable  
     CyU3PCardCtx, 41  
     CyU3PSibDevInfo, 83  
     CyU3PSibLunInfo, 87

xferCb  
     CyU3PUsbHostConfig\_t, 94

xferCount  
     CyFx3BootDmaSockRegs\_t, 28  
     CyFx3BootDmaSocket\_t, 26  
     CyU3PDmaSocket\_t, 59  
     CyU3PDmaSocketConfig\_t, 60

xferSize  
     CyFx3BootDmaSockRegs\_t, 28  
     CyFx3BootDmaSocket\_t, 26  
     CyU3PDmaChannel, 48  
     CyU3PDmaMultiChannel, 55  
     CyU3PDmaSocket\_t, 59  
     CyU3PDmaSocketConfig\_t, 60