



FX3 SDK

Firmware API Guide

Revision 1.2.1

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 © 2010-2012 Cypress Semiconductor Corporation. All rights reserved.

EZ-USB, FX3 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 installation.

Table of Contents

FX3 Device Configuration	1
Device Configuration Data Types	1
CyU3PSysClockSrc_t	2
CyU3PPartNumber_t	3
CyU3PIoMatrixLppMode_t	4
CyU3PDriveStrengthState_t	4
CyU3PIoMatrixConfig_t	5
CyU3PSysClockConfig_t	6
CyU3PLppInterruptHandler	7
Device Configuration Functions	8
CyU3PFirmwareEntry	9
CyU3PToolChainInit	9
CyU3PDeviceInit	10
CyU3PDeviceCacheControl	11
CyU3PDeviceConfigureIOMatrix	12
CyU3PSetSerialIoDriveStrength	13
CyU3PDeviceGetSysClkFreq	14
CyFxApplicationDefine	15
CyU3PDeviceReset	15
CyU3PSysGetApiVersion	16
CyU3PSysWatchDogConfigure	17
CyU3PSysWatchDogClear	18
CyU3PSysEnterSuspendMode	18
CyU3PIsLppIOConfigured	19
CyU3PLppGpioBlocksOn	20
CyU3PLppInit	20
CyU3PLppDeInit	21
CyU3PDeviceGetPartNumber	22
CyU3PUsbGetBooterVersion	23
Exceptions and Interrupts	25
Exception Handler Functions	25
CyU3PAbortHandler	25
CyU3PPrefetchHandler	26
CyU3PUndefinedHandler	27
Interrupt Handling	27

CyU3PvicEnableInterrupts	28
CyU3PvicDisableAllInterrupts	29
Cache and Memory Management	31
Cache Manager Functions	31
CyU3PSysBarrierSync	32
CyU3PSysEnableICache	32
CyU3PSysDisableICache	33
CyU3PSysEnableDCache	33
CyU3PSysDisableDCache	34
CyU3PSysEnableCacheMMU	34
CyU3PSysDisableCacheMMU	35
CyU3PSysFlushICache	35
CyU3PSysFlushDCache	36
CyU3PSysFlushCaches	36
CyU3PSysFlushIRegion	37
CyU3PSysFlushDRegion	37
CyU3PSysCleanDCache	38
CyU3PSysCleanDRegion	39
CyU3PSysClearDCache	39
CyU3PSysClearDRegion	40
CyU3PSysCacheIRegion	40
CyU3PSysCacheDRegion	41
MMU Control Functions	42
CyU3PSysEnableMMU	42
CyU3PSysDisableMMU	43
CyU3PSysInitTCMs	43
CyU3PSysLockTLBEntry	44
CyU3PSysLoadTLB	44
CyU3PSysFlushTLBEntry	45
CyU3PInitPageTable	45
Embedded Real Time OS	47
RTOS Constants	47
RTOS Data Types	48
CyU3PThread	49
CyU3PThreadEntry_t	50
CyU3PThreadStackErrorHandler_t	50
CyU3PBlockPool	51

CyU3PBytePool	51
CyU3PEvent	52
CyU3PMutex	52
CyU3PQueue	53
CyU3PSemaphore	53
CyU3PTimer	54
CyU3PTimerCb_t	54
RTOS Functions	55
Kernel Functions	55
Memory Functions	67
Thread Functions	83
Message Queue Functions	98
Semaphore Functions	104
Mutex functions	109
Event Flag Functions	114
Timer Functions	119
DMA Management	129
DMA Socket	129
DMA Buffer	130
DMA Descriptor	130
DMA Channel	130
DMA Data Types	132
CY_U3P_DMA_DSCR_COUNT	135
CY_U3P_DMA_DSCR_SIZE	135
CY_U3P_DMA_DSCR0_LOCATION	135
CY_U3P_DMA_LPP_MIN_CONS_SCK	136
CY_U3P_DMA_LPP_MAX_CONS_SCK	136
CY_U3P_DMA_LPP_MIN_PROD_SCK	136
CY_U3P_DMA_LPP_MAX_PROD_SCK	137
CY_U3P_DMA_LPP_NUM_SCK	137
CY_U3P_DMA_PIB_MIN_CONS_SCK	137
CY_U3P_DMA_PIB_MAX_CONS_SCK	138
CY_U3P_DMA_PIB_MIN_PROD_SCK	138
CY_U3P_DMA_PIB_MAX_PROD_SCK	139
CY_U3P_DMA_PIB_NUM_SCK	139
CY_U3P_DMA_UIB_MIN_CONS_SCK	139
CY_U3P_DMA_UIB_MAX_CONS_SCK	140
CY_U3P_DMA_UIB_NUM_SCK	140

CY_U3P_DMA_UIBIN_MIN_PROD_SCK	140
CY_U3P_DMA_UIBIN_MAX_PROD_SCK	141
CY_U3P_DMA_UIBIN_NUM_SCK	141
CY_U3P_DMA_CPU_NUM_SCK	141
CY_U3P_IP_BLOCK_POS	142
CY_U3P_IP_BLOCK_MASK	142
CY_U3P_DMA_SCK_MASK	142
CY_U3P_DMA_SCK_ID_MASK	143
CyU3PDmaGetSckNum	143
CyU3PDmaGetIpNum	143
CyU3PDmaGetSckId	144
CY_U3P_DMA_MAX_BUFFER_SIZE	144
CY_U3P_DMA_BUFFER_AREA_BASE	144
CY_U3P_DMA_BUFFER_AREA_LIMIT	145
CY_U3P_DMA_BUFFER_MARKER	145
CY_U3P_DMA_BUFFER_EOP	145
CY_U3P_DMA_BUFFER_ERROR	146
CY_U3P_DMA_BUFFER_OCCUPIED	146
CY_U3P_DMA_BUFFER_STATUS_MASK	146
CY_U3P_DMA_BUFFER_STATUS_WRITE_MASK	147
CY_U3P_DMA_MAX_AVAIL_COUNT	147
CY_U3P_DMA_MAX_MULTI_SCK_COUNT	147
CY_U3P_DMA_MIN_MULTI_SCK_COUNT	148
CyU3PDmaSocketId_t	148
CyU3PDmaType_t	152
CyU3PDmaMultiType_t	152
CyU3PDmaState_t	153
CyU3PDmaMode_t	155
CyU3PDmaCbType_t	156
CyU3PDmaCBInput_t	157
CyU3PDmaSckSuspType_t	157
CyU3PDmaCallback_t	159
CyU3PDmaMultiCallback_t	159
CyU3PDmaSocketCallback_t	160
CyU3PBlockId_t	160
CyU3PDmaChannel	161
CyU3PDmaChannelConfig_t	164
CyU3PDmaMultiChannel	165
CyU3PDmaMultiChannelConfig_t	168

CyU3PDmaBuffer_t	170
CyU3PDmaDescriptor_t	171
CyU3PDmaSocketConfig_t	172
CyU3PDmaSocket_t	173
DMA Functions	174
Buffer Functions	175
Descriptor Functions	179
Socket Functions	189
Channel Functions	202
Multi-Channel Functions	240
USB Management	279
USB Constants	279
CY_U3P_USB_STANDARD_RQT	280
CY_U3P_USB_CLASS_RQT	281
CY_U3P_USB_VENDOR_RQT	281
CY_U3P_USB_GS_DEVICE	281
CY_U3P_USB_GS_ENDPOINT	282
CY_U3P_USB_GS_INTERFACE	282
CY_U3P_USB_TARGET_DEVICE	282
CY_U3P_USB_TARGET_INTF	283
CY_U3P_USB_TARGET_ENDPT	283
CY_U3P_USB_TARGET_OTHER	283
CY_FX3_USB_MAX_STRING_DESC_INDEX	284
CyU3PUsbEpType_t	284
CyU3PUsbSetupCmds	285
CyU3PUsbDescType	286
CyU3PUsbDevCapType	287
CyU3PUsb3PacketType	287
CyU3PUsb3TpSubType	288
CyU3PUsbFeatureSelector	289
CY_U3P_USB_TARGET_MASK	290
CY_U3P_USB_TYPE_MASK	290
CY_U3P_USB3_TP_DEVADDR_POS	290
CY_U3P_USB3_TP_EPNUM_POS	291
CY_U3P_USB_RESERVED_RQT	291
CY_U3P_USB_OTG_STATUS_SELECTOR	291
CyU3PUsbLinkState_t	292
USB OTG Management	293

USB OTG Data Types	293
USB OTG Mode Functions	302
USB Device Management	317
Enumeration Modes	318
USB Device Data Types	318
USB Device Mode Functions	344
USB Host Management	412
USB Host Data Types	413
USB Host Mode Functions	444
P-port Management	465
Mailbox Handler	465
Mailbox Data Types	465
Mailbox Functions	468
PIB Interface Manager	474
P-port Data Types	474
P-port Functions	485
GPIF-II Management	491
GPIF Configuration	491
GPIF Resources	492
GPIF State Machine Control	492
GPIF Data Types	493
CYU3P_GPIF_NUM_STATES	494
CYU3P_GPIF_INVALID_STATE	494
CYU3P_GPIF_NUM_TRANS_FNS	494
CYU3P_PIB_MAX_BURST_SETTING	495
CYU3P_PIB_SOCKET_COUNT	495
CYU3P_PIB_THREAD_COUNT	495
CyU3PGpifEventCb_t	496
CyU3PGpifComparatorType	496
CyU3PGpifEventType	497
CyU3PGpifOutput_t	498
CyU3PGpifConfig_t	499
CyU3PGpifWaveData	500
GPIF Functions	500
CyU3PGpifLoad	501
CyU3PGpifConfigure	502
CyU3PGpifInitTransFunctions	503

CyU3PGpifWaveformLoad	504
CyU3PGpifOutputConfigure	505
CyU3PGpifRegisterConfig	506
CyU3PGpifRegisterCallback	506
CyU3PGpifSMStart	507
CyU3PGpifGetSMState	508
CyU3PGpifSMControl	509
CyU3PGpifSMSwitch	509
CyU3PGpifDisable	510
CyU3PGpifReadDataWords	511
CyU3PGpifWriteDataWords	512
CyU3PGpifControlSWInput	513
CyU3PGpifInitAddrCounter	513
CyU3PGpifInitDataCounter	514
CyU3PGpifInitCtrlCounter	515
CyU3PGpifInitComparator	515
CyU3PGpifSocketConfigure	516
Serial Peripheral Interfaces	519
UART Interface	520
UART data types	520
UART Functions	531
I2C Interface	545
I2C Data Types	546
I2C Functions	553
SPI Interface	573
SPI data types	574
SPI Functions	582
I2S Interface	599
I2S data types	600
I2S Functions	607
GPIO Interface	619
GPIO data types	620
GPIO Functions	632
Logging Support	675
Logging Data Types	675
CyU3PSysThreadId_t	676
CyU3PDebugLog_t	677

Logging Functions	677
CyU3PDebugInit	678
CyU3PDebugDeInit	679
CyU3PDebugSysMemInit	679
CyU3PDebugSysMemDeInit	680
CyU3PDebugPrint	681
CyU3PDebugPreamble	682
CyU3PDebugSetTraceLevel	683
CyU3PDebugLog	683
CyU3PDebugLogFlush	684
CyU3PDebugLogClear	685
CyU3PDebugEnable	685
CyU3PDebugDisable	686
Error Codes	689
CyU3PReturnStatus_t	689
CyU3PErrorCode_t	689
Utility Functions	693
CY_U3P_MAX	693
CY_U3P_MIN	694
CY_U3P_GET_LSB	694
CY_U3P_GET_MSB	694
CY_U3P_MAKEWORD	695
CY_U3P_DWORD_GET_BYTE0	695
CY_U3P_DWORD_GET_BYTE1	696
CY_U3P_DWORD_GET_BYTE2	696
CY_U3P_DWORD_GET_BYTE3	696
CY_U3P_MAKEDWORD	697
CyU3PAssert	697
CyU3PBusyWait	698
CyU3PMemCopy32	698
CyU3PComputeChecksum	699
CyU3PReadDeviceRegisters	700
CyU3PWriteDeviceRegisters	701
FX3 Boot APIs	703
FX3 Boot Device Interface	703
FX3 Boot Device Data Types	704

FX3 Boot Device Functions	708
FX3 Boot USB Interface	716
FX3 Boot USB Data Types	716
FX3 Boot USB Functions	726
FX3 Boot UART Interface	745
FX3 Boot UART Data Types	745
FX3 Boot UART Functions	751
FX3 Boot SPI Interface	759
FX3 Boot SPI Data Types	760
FX3 Boot SPI Functions	765
FX3 Boot I2C Interface	774
FX3 Boot I2C Data Types	775
FX3 Boot I2C Functions	780
FX3 Boot GPIO Interface	788
FX3 Boot GPIO Data Types	789
FX3 Boot GPIO Functions	791
Index	a

1 FX3 Device Configuration

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 phy and controller along with a 32-bit micro-controller (ARM926EJ-S) for powerful data processing and for building custom applications.

It has an ingenious inter-port DMA architecture which enables data transfers greater than 400 MBps. An integrated USB 2.0 OTG controller enables applications that need dual role usage scenarios. There is 512 KB of on-chip SRAM for code and data.

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

The FX3 device has a number of 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.

Please refer to the FX3 device datasheet for the complete list of supported modes on each of the device interfaces.

The system initialization sequence has to perform the following:

- FX3 Device initialization
- IO configuration
- RTOS startup





Topics

Topic	Description
Device Configuration Data Types	This section documents the data types used for configuring the FX3 device. The device configuration mainly consists of setting the required clock frequencies and initializing the IO configuration. The following data types are used to specify this information.
Device Configuration Functions	This section documents the functions used to configure and initialize the FX3 device.

1.1 Device Configuration Data Types

This section documents the data types used for configuring the FX3 device. The device configuration mainly consists of setting the required clock frequencies and initializing the IO configuration. The following data types are used to specify this information.



Enumerations

Enumeration	Description
 CyU3PSysClockSrc_t	Clock source for a peripheral block.
 CyU3PPartNumber_t	Enumeration of EZ-USB FX3 part numbers.
 CyU3PIoMatrixLppMode_t	Defines the enumerations for LPP IO line configurations.
 CyU3PDriveStrengthState_t	IO Drive Strength configuration for P port, I2C, GPIO, SPI, UART and I2S.



Group

[FX3 Device Configuration](#)

Legend

	Enumeration
	Structure

Structures

Structure	Description
 CyU3PIoMatrixConfig_t	Defines the IO matrix configuration parameters.
 CyU3PSysClockConfig_t	The clock divider information for CPU, DMA and MMIO.

Types

Type	Description
CyU3PLppInterruptHandler	Prototype of serial peripheral interrupt handler function.

1.1.1 CyU3PSysClockSrc_t

Clock source for a peripheral block.

The peripheral blocks can take various clock values based on the system clock supplied. This is all derived from the SYS_CLK_PLL supplied to the device. The SYS_CLK frequency depends on the input clock fed to the device. Also the FSLC pins should be configured correctly. If the input clock to the device is 19.2 MHz or 38.4MHz then this value is 403.2MHz and if the input clock is 26MHz or 52MHz this value is 416MHz.

C++

```
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  
};
```

Group

[Device Configuration Data Types](#)

See Also

- [CyU3PSysClockConfig_t](#)
- [CyU3PDeviceGetSysClkFreq](#)

Members

Members	Description
CY_U3P_SYS_CLK_BY_16 = 0	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.
CY_U3P_NUM_CLK_SRC	Number of clock source enumerations.

File

cyu3system.h

1.1.2 CyU3PPartNumber_t

Enumeration of EZ-USB FX3 part numbers.

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.

C++

```
enum CyU3PPartNumber_t {
    CYPART_USB3014 = 0,
    CYPART_USB3012,
    CYPART_USB3013,
    CYPART_USB3011
};
```

Group

[Device Configuration Data Types](#)

See Also

- [CyU3PDeviceGetPartNumber](#)

Members

Members	Description
CYPART_USB3014 = 0	CYUSB3014: 512 KB RAM; GPIF can be 32 bit; UART, SPI and I2S supported. OTG and USB host mode supported.
CYPART_USB3012	CYUSB3012: 256 KB RAM; GPIF can be 32 bit; UART, SPI and I2S supported.
CYPART_USB3013	CYUSB3013: 512 KB RAM; GPIF supports 16 bit; no UART, SPI or I2S.
CYPART_USB3011	CYUSB3011: 256 KB RAM; GPIF supports 16 bit; no UART, SPI or I2S.

File

cyu3system.h

1.1.3 CyU3PIoMatrixLppMode_t

Defines the enumerations for LPP IO line configurations.

The default mode should be selected when all peripherals need to be accessed. The low performance peripheral can be relocated if single LPP peripheral is chosen. Refer to the datasheet for more details on the IOs used in various configuration. When GPIF 32-bit is used, this should always be CY_U3P_IO_MATRIX_LPP_DEFAULT. I2C can be enabled in all modes and is not considered in this enumeration.

C++

```
enum CyU3PIoMatrixLppMode_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  
};
```

Group

[Device Configuration Data Types](#)

See Also

- [CyU3PDeviceConfigureIOMatrix](#)

Members

Members	Description
CY_U3P_IO_MATRIX_LPP_DEFAULT = 0	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.

File

cyu3system.h

1.1.4 CyU3PDriveStrengthState_t

IO Drive Strength configuration for P port, I2C, GPIO, SPI, UART and I2S.

The drive strength for P port and all the other serial interfaces (I2C, GPIO, SPI, UART and I2S) can be configured to various values. This enumeration defines the possible configurations that can be set.

C++

```
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
};
```

Group

[Device Configuration Data Types](#)

See Also

- [CyU3PSetPportDriveStrength](#)
- [CyU3PSetI2cDriveStrength](#)
- [CyU3PSetGpioDriveStrength](#)
- [CyU3PSetSerialIoDriveStrength](#)

Members

Members	Description
CY_U3P_DS_QUARTER_STRENGTH = 0	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

File

cyu3system.h

1.1.5 CyU3PIoMatrixConfig_t

Defines the IO matrix configuration parameters.

This structure defines all the IO configuration parameters that are required to be set at startup.

C++

```
struct CyU3PIoMatrixConfig_t {
    CyBool_t isDQ32Bit;
    CyBool_t useUart;
    CyBool_t useI2C;
    CyBool_t useI2S;
    CyBool_t useSpi;
    CyU3PIoMatrixLppMode\_t lppMode;
    uint32_t gpioSimpleEn[2];
    uint32_t gpioComplexEn[2];
};
```

Group

Device Configuration Data Types

See Also

- [CyU3PloMatrixLppMode_t](#)
- [CyU3PDeviceConfigureIOMatrix](#)

Members

Members	Description
<code>CyBool_t isDQ32Bit;</code>	CyTrue: The GPIF bus width is 32 bit, CyFalse: The GPIF bus width is 16 bit
<code>CyBool_t useUart;</code>	CyTrue: The UART interface is to be used, CyFalse: The UART interface is not to be used
<code>CyBool_t useI2C;</code>	CyTrue: The I2C interface is to be used, CyFalse: The I2C interface is not to be used
<code>CyBool_t useI2S;</code>	CyTrue: The I2S interface is to be used, CyFalse: The I2S interface is not to be used
<code>CyBool_t useSpi;</code>	CyTrue: The SPI interface is to be used, CyFalse: The SPI interface is not to be used
<code>CyU3PloMatrixLppMode_t lppMode;</code>	LPP IO configuration to be used.
<code>uint32_t gpioSimpleEn[2];</code>	Bitmap variable that identifies pins that should be configured as simple GPIOs.
<code>uint32_t gpioComplexEn[2];</code>	Bitmap variable that identifies pins that should be configured as complex GPIOs.

File

cyu3system.h

1.1.6 CyU3PSysClockConfig_t

The clock divider information for CPU, DMA and MMIO.

This structure holds information to set the clock divider for CPU, DMA and MMIO. The DMA and MMIO clocks are derived from CPU clock. There is an additional condition: DMA clock = N * MMIO clock, where N is a non-zero 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 generates a clock from the main oscillator clock.

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.

C++

```
struct CyU3PSysClockConfig_t {
    CyBool_t setSysClk400;
    uint8_t cpuClkDiv;
    uint8_t dmaClkDiv;
    uint8_t mmioClkDiv;
    CyBool_t useStandbyClk;
    CyU3PSysClockSrc\_t clkSrc;
};
```

Group

[Device Configuration Data Types](#)

See Also

- [CyU3PSysClockSrc_t](#)
- [CyU3PDeviceInit](#)
- [CyU3PDeviceGetSysClkFreq](#)

Members

Members	Description
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.
uint8_t cpuClkDiv;	CPU clock divider from clkSrc. Valid value ranges from 2 - 16.
uint8_t dmaClkDiv;	DMA clock divider from CPU clock. Valid value ranges from 2 - 16.
uint8_t mmioClkDiv;	MMIO clock divider from CPU clock. Valid value ranges from 2 - 16.
CyBool_t useStandbyClk;	Whether the standby clock is supplied
CyU3PSysClockSrc_t clkSrc;	Clock source for CPU clocking.

File

cyu3system.h

1.1.7 CyU3PLppInterruptHandler

Prototype of serial peripheral interrupt handler function.

Each serial peripheral (I2C, I2S, SPI, UART and GPIO) on the FX3 device has some 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 [CyU3PLppInit](#) function.

C++

```
typedef void (* CyU3PLppInterruptHandler)(void);
```

Group

[Device Configuration Data Types](#)

See Also

- [CyU3PLppInit](#)

File

cyu3lpp.h

1.2 Device Configuration Functions

This section documents the functions used to configure and initialize the FX3 device.


Functions

Function	Description
CyU3PFirmwareEntry	This is the entry routine for the firmware.
CyU3PToolChainInit	This is the normal entry routine provided by the tool-chain.
CyU3PDeviceInit	This function initializes the device.
CyU3PDeviceCacheControl	This function initializes the CPU caches.
CyU3PDeviceConfigureIOMatrix	Configures the IO matrix for the device.
CyU3PSetSerialIoDriveStrength	Set the IO drive strength for UART, SPI and I2S interfaces.
CyU3PDeviceGetSysClkFreq	This function returns the current SYS_CLK frequency.
CyFxApplicationDefine	This is the FX3 application definition function.
CyU3PDeviceReset	This function resets the FX3 device.
CyU3PSysGetApiVersion	This function returns the API version.
CyU3PSysWatchDogConfigure	Configure the watchdog reset control.
CyU3PSysWatchDogClear	Clear the watchdog timer to prevent a device reset.
CyU3PSysEnterSuspendMode	Puts the device to the low power state.
CyU3PIsLppIOConfigured	Check whether a specific serial peripheral has been enabled in the IO Matrix.
CyU3PLppGpioBlockIsOn	Function to check if the boot firmware has left the GPIO block powered ON.
CyU3PLppInit	This function registers the specified peripheral block as active.
CyU3PLppDeInit	Function that registers that a specified peripheral block has been made inactive.
CyU3PDeviceGetPartNumber	This function is used to get the part number of the FX3 part in use.
CyU3PUsbGetBooterVersion	Function to check the version of the boot firmware that transferred control to this firmware image.

Group

[FX3 Device Configuration](#)

Legend

	Method
---	--------

1.2.1 CyU3PFirmwareEntry

This is the entry routine for the firmware.

The tool-chain is expected to make this routine as the entry routine. The function shall be defined inside the library. This routine shall invoke the [CyU3PToolChainInit](#) routine after completing the device setup. This function sets up the stack and does device specific initializations. This function should not be explicitly invoked.

Returns

- None

C++

```
void CyU3PFirmwareEntry(
    void
);
```

Group

[Device Configuration Functions](#)

See Also

- [CyU3PToolChainInit](#)

File

cyu3system.h

1.2.2 CyU3PToolChainInit

This is the normal entry routine provided by the tool-chain.

This routine should provide a jump to the normal entry routine provided by the tool-chain (This would be main for most tool-chains). This function (main) is expected to be implemented by the user firmware.

If the user application requires a heap (uses malloc / new), it should be initialized here.

Returns

- None

C++

```
void CyU3PToolChainInit(
    void
);
```

Group

[Device Configuration Functions](#)

See Also

- [CyU3PFirmwareEntry](#)

File

cyu3system.h

1.2.3 CyU3PDeviceInit

This function initializes the device.

The function is expected to be invoked as the first call from the main () function. This function configures the various system clocks. This also initializes the VIC and exception vectors. It should be invoked only once.

Parameters

Parameters	Description
<code>CyU3PSysClockConfig_t * clkCfg</code>	The clock configuration for CPU, DMA and MMIO. For default configuration, pass in NULL as parameter. This will set CPU divider to 2 (~200MHz), DMA and MMIO dividers to 2 (~100MHz); and assume 32KHz standby clock is supplied. These are the maximum frequencies supported by the FX3 device.

Returns

- CY_U3P_SUCCESS - If the call is successful
- CY_U3P_ERROR_BAD_ARGUMENT - if the parameters are invalid

C++

```
CyU3PReturnStatus_t CyU3PDeviceInit(  
    CyU3PSysClockConfig\_t * clkCfg  
) ;
```

Group

[Device Configuration Functions](#)

See Also

- [CyU3PSysClockConfig_t](#)

File

cyu3system.h

1.2.4 CyU3PDeviceCacheControl

This function initializes the CPU caches.

The function is expected to be invoked immediately after the [CyU3PDeviceInit](#) call. This function should be called only once. The function controls the cache handling in the system. By default all 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 multiple. This is taken care of by the [CyU3PDmaBufferAlloc](#) 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.

Parameters

Parameters	Description
<code>CyBool_t isICacheEnable</code>	Whether to enable the CPU I-cache.
<code>CyBool_t isDCacheEnable</code>	Whether to enable the CPU D-cache.
<code>CyBool_t isDmaHandleDCache</code>	Whether the DMA APIs should internally take care of cache handling. The APIs can take care of simple use cases where the DMA buffers are accessed only through the DMA APIs. Set <code>CyTrue</code> if this feature needs to be enabled. If this is not the case, or if the user has better cache handling mechanism, then set this parameter to <code>CyFalse</code> .

Returns

- `CY_U3P_SUCCESS` - If the call is successful
- `CY_U3P_ERROR_BAD_ARGUMENT` - If the wrong clock values are used

C++

```
CyU3PReturnStatus_t CyU3PDeviceCacheControl(
    CyBool_t isICacheEnable,
    CyBool_t isDCacheEnable,
    CyBool_t isDmaHandleDCache
);
```

Group

[Device Configuration Functions](#)

See Also

- [CyU3PDeviceInit](#)

File

cyu3system.h

1.2.5 CyU3PDeviceConfigureIOMatrix

Configures the IO matrix for the device.

The FX3 device has 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 to as a GPIF interface. The mode selection for this interface should be based on the means of connecting the external device / processor. Some pins are not used depending upon the configuration used. The free pins can be used as GPIOs. If 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 [CyU3PDeviceGpioOverride](#) call. If 32 bit GPIF interface is used all of DQ[31:0] is reserved for GPIF interface.

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 is 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 GPIO.

Please refer to the FX3 device data sheet for the complete list of supported modes on each of the device interfaces.

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 left tri-stated. 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 $(1 \ll \text{IO number})$.

There are only eight complex GPIOs possible. The restriction is applied based on % 8. So only one of 0, 8, 16, ... can be configured as complex GPIO. Similarly only one of 1, 9, 17 ... can be configured as complex GPIO.

This function is recommended to be called after the [CyU3PDeviceInit](#) call from the main () function. IO matrix is not recommended to be dynamically changed. This API can be invoked when the peripherals affected are disabled. If the API is invoked after initialization, then it is expected that the external devices attached are also capable of this dynamic reconfiguration. This function provides error checks but is not completely fool proof. The actual IO configuration selected should make sure that there are no conflicts. For example, CTL[12:5] are allowed as GPIOs by this function. But if the GPIF configuration is using any of these lines and these lines are also configured as GPIO, then the IO will behave as the GPIOs and not as part of GPIF. So care should be taken during hardware design.

None of the IOs are expected to be used before this call. Even if the configuration to be used is same as default, the call still has to be made to initialize the IOs. The default configuration for p-port will depend on the boot-mode used. If p-port boot is selected, then the boot-loader will configure it accordingly. GPIF will be left configured at 16-bit mode, the LPP mode will be CY_U3P_IO_MATRIX_LPP_DEFAULT and none of the GPIOs will be enabled.

Parameters

Parameters	Description
CyU3PIoMatrixConfig_t * cfg_p	Pointer to Configuration parameters.

Returns

- 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 FX3 part in use does not support any of the selected features

C++

```
CyU3PReturnStatus\_t CyU3PDeviceConfigureIOMatrix(  
    CyU3PIoMatrixConfig\_t * cfg_p  
);
```

Group

[Device Configuration Functions](#)

See Also

- [CyU3PIoMatrixConfig_t](#)
- [CyU3PDeviceGpioOverride](#)
- [CyU3PDeviceGpioRestore](#)

File

cyu3system.h

1.2.6 CyU3PSetSerialIoDriveStrength

Set the IO drive strength for UART, SPI and I2S interfaces.

The default IO drive strength for the interfaces are set to CY_U3P_DS_THREE_QUARTER_STRENGTH. I2C has a separate API for setting the drive strength. Refer to [CyU3PSetI2cDriveStrength\(\)](#) API.

Parameters

Parameters	Description
CyU3PDriveStrengthState_t driveStrength	Serial IO drive strength

Returns

- CY_U3P_SUCCESS - If the operation is successful
- CY_U3P_ERROR_BAD_ARGUMENT - If the Drive strength requested is invalid

C++

```
CyU3PReturnStatus\_t CyU3PSetSerialIoDriveStrength(  
    CyU3PDriveStrengthState\_t driveStrength  
);
```

Group

[Device Configuration Functions](#)

See Also

- [CyU3PDriveStrengthState_t](#)
- [CyU3PSetI2cDriveStrength](#)

File

cyu3system.h

1.2.7 CyU3PDeviceGetSysClkFreq

This function returns the current SYS_CLK frequency.

The function can be called only after the [CyU3PDeviceInit](#) call has been invoked. The SYS_CLK frequency depends on the input clock fed to the device. Also the FSLC pins should be configured correctly. If the input clock to the device is 19.2 MHz or 38.4MHz then this value is 403.2MHz and if the input clock is 26MHz or 52MHz this value is 416MHz.

Parameters

Parameters	Description
uint32_t * freq	Pointer to return the current SYS_CLK frequency.

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PDeviceGetSysClkFreq(  
    uint32_t * freq  
);
```

Group

[Device Configuration Functions](#)

See Also

- [CyU3PSysClockConfig_t](#)
- [CyU3PFirmwareEntry](#)

File

cyu3system.h

1.2.8 CyFxApplicationDefine

This is the FX3 application definition function.

The FX3 application RTOS primitives are created in this function. This function needs to be defined by the FX3 application firmware. This is invoked from the system module after the device and all the modules are initialized. No APIs are expected to be invoked from this function. The application threads and other required OS primitives can be created here. At-least one thread must be created. This function should not be explicitly invoked.

Returns

- None

C++

```
void CyFxApplicationDefine(
    void
);
```

Group

[Device Configuration Functions](#)

See Also

- [CyU3PKernelEntry](#)

File

cyu3system.h

1.2.9 CyU3PDeviceReset

This function resets the FX3 device.

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, then it should be noted that the firmware should be capable of initializing the global variables. Also if the scatter load option is used, then the firmware should be capable of skipping this step as the image is already loaded.

Parameters

Parameters	Description
CyBool_t isWarmReset	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 FX3 needs to be performed again.

Returns

None as this function does not return.

C++

```
void CyU3PDeviceReset(  
    CyBool_t isWarmReset  
);
```

Group

[Device Configuration Functions](#)

File

cyu3system.h

1.2.10 CyU3PSysGetApiVersion

This function returns the API version.

The function returns the version number placed in cyfxversion.h at the time the API library was built. This can be cross referenced by the application. Any parameter can take a NULL pointer if that field is not required.

Return values

- CY_U3P_SUCCESS - If the call is successful
- CY_U3P_ERROR_INVALID_CONFIGURATION - If the library configuration is invalid

Parameters

Parameters	Description
uint16_t * majorVersion	Major version number for the release
uint16_t * minorVersion	Minor version number for the release
uint16_t * patchNumer	Patch version for the release
uint16_t * buildNumer	The build number for the release

C++

```
CyU3PReturnStatus_t CyU3PSysGetApiVersion(  
    uint16_t * majorVersion,  
    uint16_t * minorVersion,  
    uint16_t * patchNumer,
```

```
uint16_t * buildNumer
);
```

Group

[Device Configuration Functions](#)

See Also

- None

File

cyu3system.h

1.2.11 CyU3PSysWatchDogConfigure

Configure the watchdog reset control.

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.

Parameters

Parameters	Description
CyBool_t enable	Whether the watch dog should be enabled.
uint32_t period	Period in milliseconds. Is valid only for enable calls.

Returns

None

C++

```
void CyU3PSysWatchDogConfigure(
    CyBool_t enable,
    uint32_t period
);
```

Group

[Device Configuration Functions](#)

See Also

- [CyU3PSysWatchDogClear](#)

File

cyu3system.h

1.2.12 CyU3PSysWatchDogClear

Clear the watchdog timer to prevent a device reset.

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.

Returns

None

C++

```
void CyU3PSysWatchDogClear(  
    void  
);
```

Group

[Device Configuration Functions](#)

See Also

- [CyU3PSysWatchDogConfigure](#)

File

cyu3system.h

1.2.13 CyU3PSysEnterSuspendMode

Puts the device to the low power state.

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.

CY_U3P_SYS_PPORT_WAKEUP_SRC_EN, CY_U3P_SYS_USB_WAKEUP_SRC_EN and CY_U3P_SYS_UART_WAKEUP_SRC_EN are the various wakeup sources.

Parameters

Parameters	Description
uint16_t wakeupFlags	Bit mask representing the wakeup sources that are allowed to bring FX3 out of suspend mode.
uint16_t polarity	Polarity 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.
uint16_t * wakeupSource	Output parameter indicating the sources responsible for waking the FX3 from the Suspend mode.

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PSysEnterSuspendMode(
    uint16_t wakeupFlags,
    uint16_t polarity,
    uint16_t * wakeupSource
);
```

Group

[Device Configuration Functions](#)

File

cyu3system.h

1.2.14 CyU3PIsLppIOConfigured

Check whether a specific serial peripheral has been enabled in the IO Matrix.

This function checks whether a specific serial peripheral interface has been enabled in the active IO matrix.

Parameters

Parameters	Description
CyU3PLppModule_t lppModule	Serial interface to be queried.

Returns

- CyTrue if specified lpp is enabled, CyFalse otherwise.

C++

```
CyBool_t CyU3PIsLppIOConfigured(
    CyU3PLppModule_t lppModule
);
```

Group

[Device Configuration Functions](#)

See Also

- [CyU3PDeviceConfigureIOMatrix](#)

- [CyU3PLppModule_t](#)

File

cyu3system.h

1.2.15 CyU3PLppGpioBlockIsOn

Function to check if the boot firmware has left the GPIO block powered ON.

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.

C++

```
CyBool_t CyU3PLppGpioBlockIsOn(  
    void  
);
```

Group

[Device Configuration Functions](#)

Notes

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.

See Also

- [CyU3PDeviceConfigureIOMatrix](#)

Return Values

- CyTrue if the boot firmware has left GPIO on, CyFalse otherwise.

File

cyu3lpp.h

1.2.16 CyU3PLppInit

This function registers the specified peripheral block as active.

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.

Parameters

Parameters	Description
CyU3PLppModule_t lppModule	The peripheral block being initialized.
CyU3PLppInterruptHandler intrHandler	Interrupt handler function for the peripheral block.

Returns

- CY_U3P_SUCCESS - If the call is successful.
- CY_U3P_ERROR_ALREADY_STARTED - The block is already initialized.
- CY_U3P_ERROR_NOT_SUPPORTED - If the serial peripheral block is not supported by the FX3 part.
- CY_U3P_ERROR_INVALID_SEQUENCE - If the block init is being called without turning on the corresponding clock.

C++

```
CyU3PReturnStatus_t CyU3PLppInit(
    CyU3PLppModule_t lppModule,
    CyU3PLppInterruptHandler intrHandler
);
```

Group

[Device Configuration Functions](#)

See Also

- [CyU3PLppDeInit](#)
- [CyU3PUartSetClock](#)
- [CyU3PI2cSetClock](#)
- [CyU3PI2sSetClock](#)
- [CyU3PSpiSetClock](#)

File

cyu3lpp.h

1.2.17 CyU3PLppDeInit

Function that registers that a specified peripheral block has been made inactive.

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.

Parameters

Parameters	Description
<code>CyU3PLppModule_t</code> <code>lppModule</code>	The peripheral block being de-initialized.

Returns

- `CY_U3P_SUCCESS` - If the call is successful.
- `CY_U3P_ERROR_NOT_SUPPORTED` - If the serial peripheral selected is not supported by the FX3 device
- `CY_U3P_ERROR_NOT_STARTED` - If the serial peripheral being stopped has not been started.

C++

```
CyU3PReturnStatus\_t CyU3PLppDeInit(  
    CyU3PLppModule_t lppModule  
);
```

Group

[Device Configuration Functions](#)

See Also

- [CyU3PLppInit](#)

File

`cyu3lpp.h`

1.2.18 CyU3PDeviceGetPartNumber

This function is used to get the part number of the FX3 part in use.

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.

Returns

Part number of the FX3 device in use.

C++

```
CyU3PPartNumber\_t CyU3PDeviceGetPartNumber(  
    void  
);
```

Group

[Device Configuration Functions](#)

See Also

- [CyU3PPartNumber_t](#)

File

cyu3system.h

1.2.19 CyU3PUsbGetBooterVersion

Function to check the version of the boot firmware that transferred control to this firmware image.

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.

Parameters

Parameters	Description
uint8_t * major_p	Return parameter to be filled with major version of boot firmware.
uint8_t * minor_p	Return parameter to be filled with minor version of boot firmware.
uint8_t * patch_p	Return parameter to be filled with patch level of boot firmware.

Returns

- 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 return values.
- 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.

C++

```
CyU3PReturnStatus\_t CyU3PUsbGetBooterVersion(
    uint8_t * major_p,
    uint8_t * minor_p,
    uint8_t * patch_p
);
```

Group

[Device Configuration Functions](#)

File

cyu3usb.h

2 Exceptions and Interrupts

This section describes the operating modes, exceptions and interrupts in the FX3 firmware.

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 firmware makes use of the full ARM instruction set.

Topics




Topic	Description
Exception Handler Functions	This section provides information on ARM Exception handlers in the FX3 firmware.
Interrupt Handling	This section provides information on FX3 device interrupts.

2.1 Exception Handler Functions

This section provides information on ARM Exception handlers in the FX3 firmware.

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 cyfctx.c file, and these can be modified by the users to match their requirements (for example, reset the FX3 device and restart operation).

Functions

Function	Description
 CyU3PAbortHandler	The abort error exception handler.
 CyU3PPrefetchHandler	The pre-fetch error exception handler.
 CyU3PUndefinedHandler	The undefined instruction exception handler.

Group

[Exceptions and Interrupts](#)

Legend

	Method
---	--------

2.1.1 CyU3PAbortHandler

The abort error exception handler.

This function gets invoked when the ARM CPU encounters an data pre-fetch abort error. Since there are no virtual memory use case, this is an unknown memory access error. This does not occur as a part of normal operation sequence. This is a fatal error for this platform.

Returns

None

C++

```
void CyU3PAbortHandler(  
    void  
);
```

Group

[Exception Handler Functions](#)

See Also

- [CyU3PUndefinedHandler](#)
- [CyU3PPrefetchHandler](#)

File

cyu3os.h

2.1.2 CyU3PPrefetchHandler

The pre-fetch error exception handler.

This function gets invoked when the ARM CPU encounters an instruction pre-fetch error. Since there are no virtual memory use case, this is an unknown memory access error. This does not occur as a part of normal operation sequence. This is a fatal error for this platform.

Returns

None

C++

```
void CyU3PPrefetchHandler(  
    void  
);
```

Group

[Exception Handler Functions](#)

See Also

- [CyU3PUndefinedHandler](#)

- [CyU3PAbortHandler](#)

File

cyu3os.h

2.1.3 CyU3PUndefinedHandler

The undefined instruction exception handler.

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 does not occur as a part of normal operation sequence. This is a fatal error.

Returns

None

C++

```
void CyU3PUndefinedHandler(  
    void  
);
```

Group

[Exception Handler Functions](#)

See Also

- [CyU3PPrefetchHandler](#)
- [CyU3PAbortHandler](#)

File

cyu3os.h

2.2 Interrupt Handling

This section provides information on FX3 device interrupts.

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.


Functions

Function	Description
 CyU3PvicEnableInterrupts	This function enables the specified interrupts at the VIC level.
 CyU3PvicDisableAllInterrupts	This function disables all FX3 interrupts at the VIC level.

Group

[Exceptions and Interrupts](#)

Legend

	Method
---	--------

2.2.1 CyU3PvicEnableInterrupts

This function enables the specified interrupts at the VIC level.

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.

Parameters

Parameters	Description
uint32_t mask	Bit-mask representing interrupts to be enabled.

Returns

None

C++

```
void CyU3PvicEnableInterrupts(  
    uint32_t mask  
);
```

Group

[Interrupt Handling](#)

See Also

- [CyU3PvicDisableAllInterrupts](#)

File

cyu3vic.h

2.2.2 CyU3PvicDisableAllInterrupts

This function disables all FX3 interrupts at the VIC level.

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.

Returns

A uint32_t mask that represents the interrupts that were enabled.

C++

```
uint32_t CyU3PvicDisableAllInterrupts(  
    void  
);
```

Group

[Interrupt Handling](#)

See Also

- [CyU3PvicEnableInterrupts](#)

File

cyu3vic.h

3 Cache and Memory Management

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.

Topics

Topic	Description
Cache Manager Functions	This section documents the I-Cache and D-Cache management functions.
MMU Control Functions	This section documents the functions provided to initialize and manage the MMU on the ARM core.

3.1 Cache Manager Functions

This section documents the I-Cache and D-Cache management functions.

Functions

Function	Description
CyU3PSysBarrierSync	Memory barrier synchronization function.
CyU3PSysEnableICache	Enable the instruction cache.
CyU3PSysDisableICache	Disable the instruction cache.
CyU3PSysEnableDCache	Enable the Data Cache.
CyU3PSysDisableDCache	Disable the data cache.
CyU3PSysEnableCacheMMU	Enable the Caches and the MMU.
CyU3PSysDisableCacheMMU	Disable the caches and MMU.
CyU3PSysFlushICache	Flush the I-Cache.
CyU3PSysFlushDCache	Flush the D-Cache.
CyU3PSysFlushCaches	Flush both I and D Caches.
CyU3PSysFlushIRegion	Flush a code region from the I-Cache.
CyU3PSysFlushDRegion	Flush a data region from the D-Cache.
CyU3PSysCleanDCache	Clean the entire D-Cache.
CyU3PSysCleanDRegion	Clean a data region from the D-Cache.
CyU3PSysClearDCache	Clean and Flush the entire D-Cache.
CyU3PSysClearDRegion	Clean and flush a data region from the D-Cache.
CyU3PSysCacheIRegion	Function to lock a code region into the I-Cache.
CyU3PSysCacheDRegion	Function to lock a data region into the D-Cache.

Group

[Cache and Memory Management](#)

Legend

	Method
---	--------

3.1.1 CyU3PSysBarrierSync

Memory barrier synchronization function.

Function that ensures that all write buffers to memory have been drained. This call can be used as a synchronization barrier.

Returns

None

C++

```
void CyU3PSysBarrierSync(  
    void  
);
```

Group

[Cache Manager Functions](#)

File

cyu3mmu.h

3.1.2 CyU3PSysEnableICache

Enable the instruction cache.

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 functions is available for debug purposes and is used by the library.

Returns

None

C++

```
void CyU3PSysEnableICache(  
    void  
);
```

Group

[Cache Manager Functions](#)

File

cyu3mmu.h

3.1.3 CyU3PSysDisableICache

Disable the instruction cache.

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 disabled by default.

Returns

None

C++

```
void CyU3PSysDisableICache(
    void
);
```

Group

[Cache Manager Functions](#)

File

cyu3mmu.h

3.1.4 CyU3PSysEnableDCache

Enable the Data Cache.

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. The functions is available for debug purposes and is used by the library.

Returns

None

C++

```
void CyU3PSysEnableDCache(
    void
);
```

```
);
```

Group

[Cache Manager Functions](#)

File

cyu3mmu.h

3.1.5 CyU3PSysDisableDCache

Disable the data cache.

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 disabled by default.

Returns

None

C++

```
void CyU3PSysDisableDCache(  
    void  
);
```

Group

[Cache Manager Functions](#)

File

cyu3mmu.h

3.1.6 CyU3PSysEnableCacheMMU

Enable the Caches and the MMU.

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.

Returns

None

C++

```
void CyU3PSysEnableCacheMMU(  
    void  
);
```

```
void  
) ;
```

Group

[Cache Manager Functions](#)

File

cyu3mmu.h

3.1.7 CyU3PSysDisableCacheMMU

Disable the caches and MMU.

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.

Returns

None

C++

```
void CyU3PSysDisableCacheMMU(  
    void  
) ;
```

Group

[Cache Manager Functions](#)

File

cyu3mmu.h

3.1.8 CyU3PSysFlushICache

Flush the I-Cache.

Function to flush the Instruction Cache.

Returns

None

C++

```
void CyU3PSysFlushICache(  
    void  
) ;
```

Group

[Cache Manager Functions](#)

File

cyu3mmu.h

3.1.9 CyU3PSysFlushDCache

Flush the D-Cache.

Function to flush the Data Cache. The caller should ensure that the cache has been cleaned before calling this function.

Returns

None

C++

```
void CyU3PSysFlushDCache(  
    void  
);
```

Group

[Cache Manager Functions](#)

File

cyu3mmu.h

3.1.10 CyU3PSysFlushCaches

Flush both I and D Caches.

Function to flush both the Instruction and Data Caches. The caller is responsible for ensuring that the Data cache is clean.

Returns

None

C++

```
void CyU3PSysFlushCaches(  
    void  
);
```


Group[Cache Manager Functions](#)**File**

cyu3mmu.h

3.1.11 CyU3PSysFlushIRegion

Flush a code region from the I-Cache.

Function to remove a specified code region from the Instruction Cache.

Parameters

Parameters	Description
uint32_t * addr	Start address of the region to flush.
uint32_t len	Length of the region in bytes.

Returns

None

C++

```
void CyU3PSysFlushIRegion(  
    uint32_t * addr,  
    uint32_t len  
);
```

Group[Cache Manager Functions](#)**File**

cyu3mmu.h

3.1.12 CyU3PSysFlushDRegion

Flush a data region from the D-Cache.

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.

Parameters

Parameters	Description
uint32_t * addr	Start address of the region to clean.
uint32_t len	Length of the region in bytes.

Returns

None

C++

```
void CyU3PSysFlushDRegion(  
    uint32_t * addr,  
    uint32_t len  
);
```

Group

[Cache Manager Functions](#)

File

cyu3mmu.h

3.1.13 CyU3PSysCleanDCache

Clean the entire D-Cache.

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.

Returns

None

C++

```
void CyU3PSysCleanDCache(  
    void  
);
```

Group

[Cache Manager Functions](#)

File

cyu3mmu.h

3.1.14 CyU3PSysCleanDRegion

Clean a data region from the D-Cache.

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.

Parameters

Parameters	Description
uint32_t * addr	Start address of the region to clean.
uint32_t len	Length of the region in bytes.

Returns

None

C++

```
void CyU3PSysCleanDRegion(  
    uint32_t * addr,  
    uint32_t len  
);
```

Group

[Cache Manager Functions](#)

File

cyu3mmu.h

3.1.15 CyU3PSysClearDCache

Clean and Flush the entire D-Cache.

Function to clean and flush the entire Data cache using the test/clean command.

Returns

None

C++

```
void CyU3PSysClearDCache(  
    void  
);
```

Group

[Cache Manager Functions](#)

File

cyu3mmu.h

3.1.16 CyU3PSysClearDRegion

Clean and flush a data region from the D-Cache.

Function to clean and flush a specified data region from the Data Cache.

Parameters

Parameters	Description
uint32_t * addr	Start address of the region to flush.
uint32_t len	Length of the region in bytes.

Returns

None

C++

```
void CyU3PSysClearDRegion(  
    uint32_t * addr,  
    uint32_t len  
);
```

Group

[Cache Manager Functions](#)

File

cyu3mmu.h

3.1.17 CyU3PSysCacheIRegion

Function to lock a code region into the I-Cache.

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.

Parameters

Parameters	Description
uint32_t * addr	Start address of the region to cache.
uint32_t len	Length of the region in bytes.

Returns

The way index into which the region was locked.

C++

```
uint32_t CyU3PSysCacheIRegion(
    uint32_t * addr,
    uint32_t len
);
```

Group

[Cache Manager Functions](#)

File

cyu3mmu.h

3.1.18 CyU3PSysCacheDRegion

Function to lock a data region into the D-Cache.

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.

Parameters

Parameters	Description
uint32_t * addr	Start address of the region to cache.
uint32_t len	Length of the region in bytes.

Returns

The way index into which the region was locked.

C++

```
uint32_t CyU3PSysCachedRegion(
    uint32_t * addr,
    uint32_t len
);
```

Group

[Cache Manager Functions](#)








File

cyu3mmu.h

3.2 MMU Control Functions

This section documents the functions provided to initialize and manage the MMU on the ARM core.

Functions

Function	Description
 CyU3PSysEnableMMU	Enable the MMU.
 CyU3PSysDisableMMU	Disable the MMU.
 CyU3PSysInitTCMs	Function to initialize the TCM regions.
 CyU3PSysLockTLBEntry	Function to lock the TLB entry for a page walk.
 CyU3PSysLoadTLB	Function to lock all valid page walks into TLB.
 CyU3PSysFlushTLBEntry	Invalidate the TLB entry corresponding to a specified MVA.
 CyU3PInitPageTable	Function to create the page tables for the device memory map.

Group

[Cache and Memory Management](#)

Legend

	Method
---	--------

3.2.1 CyU3PSysEnableMMU

Enable the MMU.

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.

Returns

None

C++

```
void CyU3PSysEnableMMU(  
    void  
);
```

Group

[MMU Control Functions](#)

File

cyu3mmu.h

3.2.2 CyU3PSysDisableMMU

Disable the MMU.

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.

Returns

None

C++

```
void CyU3PSysDisableMMU(
    void
);
```

Group

[MMU Control Functions](#)

File

cyu3mmu.h

3.2.3 CyU3PSysInitTCMs

Function to initialize the TCM regions.

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.

Returns

None

C++

```
void CyU3PSysInitTCMs(
    void
);
```

```
);
```

Group

[MMU Control Functions](#)

File

cyu3mmu.h

3.2.4 CyU3PSysLockTLBEntry

Function to lock the TLB entry for a page walk.

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.

Returns

None

C++

```
void CyU3PSysLockTLBEntry(  
    uint32_t * addr  
);
```

Group

[MMU Control Functions](#)

File

cyu3mmu.h

3.2.5 CyU3PSysLoadTLB

Function to lock all valid page walks into TLB.

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.

Returns

None

C++

```
void CyU3PSysLoadTLB(  
    void
```



```
);
```

Group[MMU Control Functions](#)**File**

cyu3mmu.h

3.2.6 CyU3PSysFlushTLBEntry

Invalidate the TLB entry corresponding to a specified MVA.

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.

Parameters

Parameters	Description
uint32_t * addr	Address whose TLB entry is to be invalidated.

Returns

None

C++

```
void CyU3PSysFlushTLBEntry(  
    uint32_t * addr  
);
```

Group[MMU Control Functions](#)**File**

cyu3mmu.h

3.2.7 CyU3PInitPageTable

Function to create the page tables for the device memory map.

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.

Returns

None

C++

```
void CyU3PInitPageTable(  
    void  
);
```

Group

[MMU Control Functions](#)

File

cyu3mmu.h

4 Embedded Real Time OS

The FX3 firmware framework makes use of a Embedded Real-Time Operating System. The drivers for various peripheral blocks in the platform are typically 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.

The framework provides hooks for the application logic to configure the device behavior and to perform data transfers through it. The application logic itself can be implemented across multiple threads and make use of all of the OS services that are used by the Cypress provided drivers.

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.

The ThreadX services are not directly exposed by the firmware framework. This is to ensure that the application logic is independent of the OS used and need not be changed to accommodate a future changes in the embedded OS. The OS services are made available through a set of platform specific wrappers that are placed around them.

Topics

Topic	Description
RTOS Constants	The RTOS wrappers used by the FX3 firmware library define and make use of the following constants.
RTOS Data Types	The RTOS wrappers used by the FX3 firmware library for data types.
RTOS Functions	This section documents the functions that are provided as part of the Embedded RTOS services.

4.1 RTOS Constants

The RTOS wrappers used by the FX3 firmware library define and make use of the following constants.

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 getOption / setOption 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 getOption 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 getOption / setOption 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.

Group

[Embedded Real Time OS](#)

4.2 RTOS Data Types

The RTOS wrappers used by the FX3 firmware library for data types.

Group

[Embedded Real Time OS](#)

Types

Type	Description
CyU3PThread	Thread structure.
CyU3PThreadEntry_t	Thread entry function type.
CyU3PThreadStackErrorHandler_t	Type of thread stack overflow handler function.
CyU3PBlockPool	Block pool structure.
CyU3PBytePool	Data Types
CyU3PEvent	Event structure.
CyU3PMutex	Mutex structure.
CyU3PQueue	Queue structure.
CyU3PSemaphore	Semaphore structure.
CyU3PTimer	Timer structure.
CyU3PTimerCb_t	Type of callback function called on timer expiration.

4.2.1 CyU3PThread

Thread structure.

This is the RTOS defined thread structure for RTOS internal use. The pointer to this structure is used as handle for all RTOS APIs.

C++

```
typedef struct CyU3PThread CyU3PThread;
```

Group

[RTOS Data Types](#)

See Also

- [CyU3PThreadCreate](#)
- [CyU3PThreadDestroy](#)
- [CyU3PThreadIdentify](#)
- [CyU3PThreadInfoGet](#)
- [CyU3PThreadSleep](#)
- [CyU3PThreadSuspend](#)
- [CyU3PThreadResume](#)
- [CyU3PThreadWaitAbort](#)
- [CyU3PThreadRelinquish](#)
- [CyU3PThreadPriorityChange](#)

- [CyU3PThreadStackErrorNotify](#)

File

cyu3os.h

4.2.2 CyU3PThreadEntry_t

Thread entry function type.

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.

C++

```
typedef void (* CyU3PThreadEntry_t)(uint32_t threadArg);
```

Group

[RTOS Data Types](#)

See Also

- [CyU3PThread](#)
- [CyU3PThreadCreate](#)

File

cyu3os.h

4.2.3 CyU3PThreadStackErrorHandler_t

Type of thread stack overflow handler function.

This type represents the callback function that will be called when the RTOS detects a stack overflow condition on one of the user created threads.

C++

```
typedef void (* CyU3PThreadStackErrorHandler_t)(CyU3PThread *thread_p);
```

Group

[RTOS Data Types](#)

See Also

- [CyU3PThreadStackErrorNotify](#)

File

cyu3os.h

4.2.4 CyU3PBlockPool

Block pool structure.

This is the RTOS defined block pool structure for RTOS internal use. The pointer to this structure is used as handle for all RTOS APIs.

C++

```
typedef struct CyU3PBlockPool CyU3PBlockPool;
```

Group

[RTOS Data Types](#)

See Also

- [CyU3PBlockPoolCreate](#)
- [CyU3PBlockPoolDestroy](#)
- [CyU3PBlockAlloc](#)
- [CyU3PBlockFree](#)

File

cyu3os.h

4.2.5 CyU3PBytePool

Data Types

C++

```
typedef struct CyU3PBytePool CyU3PBytePool;
```

Group

[RTOS Data Types](#)

File

cyu3os.h

4.2.6 CyU3PEvent

Event structure.

This is the RTOS defined event structure for RTOS internal use. The pointer to this structure is used as handle for all RTOS APIs.

C++

```
typedef struct CyU3PEvent CyU3PEvent;
```

Group

[RTOS Data Types](#)

See Also

- [CyU3PEventCreate](#)
- [CyU3PEventDestroy](#)
- [CyU3PEventSet](#)
- [CyU3PEventGet](#)

File

cyu3os.h

4.2.7 CyU3PMutex

Mutex structure.

This is the RTOS defined mutex structure for RTOS internal use. The pointer to this structure is used as handle for all RTOS APIs.

C++

```
typedef struct CyU3PMutex CyU3PMutex;
```

Group

[RTOS Data Types](#)

See Also

- [CyU3PMutexCreate](#)
- [CyU3PMutexDestroy](#)
- [CyU3PMutexPut](#)
- [CyU3PMutexGet](#)

File

cyu3os.h

4.2.8 CyU3PQueue

Queue structure.

This is the RTOS defined queue structure for RTOS internal use. The pointer to this structure is used as handle for all RTOS APIs.

C++

```
typedef struct CyU3PQueue CyU3PQueue;
```

Group

[RTOS Data Types](#)

See Also

- [CyU3PQueueCreate](#)
- [CyU3PQueueDestroy](#)
- [CyU3PQueueSend](#)
- [CyU3PQueuePrioritySend](#)
- [CyU3PQueueReceive](#)
- [CyU3PQueueFlush](#)

File

cyu3os.h

4.2.9 CyU3PSemaphore

Semaphore structure.

This is the RTOS defined semaphore structure for RTOS internal use. The pointer to this structure is used as handle for all RTOS APIs.

C++

```
typedef struct CyU3PSemaphore CyU3PSemaphore;
```

Group

[RTOS Data Types](#)

See Also

- [CyU3PSemaphoreCreate](#)
- [CyU3PSemaphoreDestroy](#)
- [CyU3PSemaphoreGet](#)
- [CyU3PSemaphorePut](#)

File

cyu3os.h

4.2.10 CyU3PTimer

Timer structure.

This is the RTOS defined timer structure for RTOS internal use. The pointer to this structure is used as handle for all RTOS APIs.

C++

```
typedef struct CyU3PTimer CyU3PTimer;
```

Group

[RTOS Data Types](#)

See Also

- [CyU3PTimerCreate](#)
- [CyU3PTimerDestroy](#)
- [CyU3PTimerStart](#)
- [CyU3PTimerStop](#)
- [CyU3PTimerModify](#)

File

cyu3os.h

4.2.11 CyU3PTimerCb_t

Type of callback function called on timer expiration.

Type of callback functions that are registered as part of timer creation and are invoked when the specified time has elapsed. The timerArg argument is a context input that is specified by the user when creating the timer.

C++

```
typedef void (* CyU3PTimerCb_t)(uint32_t timerArg);
```

Group

[RTOS Data Types](#)

See Also

- [CyU3PTimer](#)
- [CyU3PTimerCreate](#)

File

cyu3os.h

4.3 RTOS Functions

This section documents the functions that are provided as part of the Embedded RTOS services.

Group

[Embedded Real Time OS](#)












Topics

Topic	Description
Kernel Functions	The following functions represent the interface between the RTOS kernel and the firmware application code.
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.
Thread Functions	This section documents the thread functions that are provided as part of the FX3 firmware.
Message Queue Functions	This section documents the message queue functions that are provided as part of the FX3 firmware.
Semaphore Functions	This section documents the semaphore functions supported by the FX3 firmware.
Mutex functions	This section documents the mutex functions that are provided as part of the FX3 firmware.
Event Flag Functions	This section documents the event flag functions provided by the FX3 firmware.
Timer Functions	This section documents the application timer functions that are provided by the FX3 firmware.

4.3.1 Kernel Functions

The following functions represent the interface between the RTOS kernel and the firmware application code.


Functions

Function	Description
 CyU3PIrqContextSave	This function saves the active thread context when entering the IRQ context.
 CyU3PIrqVectoredContextSave	This function saves the thread context when entering the IRQ context.
 CyU3PIrqContextRestore	This function restores the thread context after handling an IRQ interrupt.
 CyU3PFiqContextSave	This function saves the active thread context when entering the FIQ context.
 CyU3PFiqContextRestore	This function restores the thread context after handling an FIQ interrupt.
 CyU3PIrqNestingStart	This function switches the ARM mode from IRQ to SYS to allow nesting of interrupts.
 CyU3PIrqNestingStop	This function switches the ARM mode from SYS to IRQ at the end of a interrupt handler.
 CyU3PKernelEntry	RTOS kernel entry function.
 CyU3PApplicationDefine	The driver specific OS primitives and threads shall be created in this function.
 CyU3POsTimerHandler	The OS scheduler.
 CyU3POsTimerInit	Initialize the OS scheduler timer.

Group

[RTOS Functions](#)

Legend

	Method
---	--------

4.3.1.1 CyU3PIrqContextSave

This function saves the active thread context when entering the IRQ context.

This function shall be invoked by the IRQ handlers as the first call and saves the current thread state before handling the IRQ. This function should be used if the device does not have a VIC. The [CyU3PIrqVectoredContextSave](#) function should be used if the device includes a VIC.

Returns

None

C++

```
void CyU3PIrqContextSave(  
    void  
);
```

Group

[Kernel Functions](#)

See Also

- [CyU3PIrqContextRestore](#)
- [CyU3PIrqVectoredContextSave](#)
- [CyU3PFiqContextSave](#)

File

cyu3os.h

4.3.1.2 CyU3PIrqVectoredContextSave

This function saves the thread context when entering the IRQ context.

This shall be invoked by the IRQ handlers as the first call and saves the current thread state before handling the IRQ. This function should be used if the device uses a VIC and has vectored interrupt handlers.

Returns

None

C++

```
void CyU3PIrqVectoredContextSave(  
    void  
);
```

Group

[Kernel Functions](#)

See Also

- [CyU3PIrqContextRestore](#)
- [CyU3PIrqContextSave](#)
- [CyU3PFiqContextSave](#)

File

cyu3os.h

4.3.1.3 CyU3PIrqContextRestore

This function restores the thread context after handling an IRQ interrupt.

This shall be invoked by the IRQ handlers as the last call. This is a non returning function and restores control to the new thread according to the scheduling mechanism of the RTOS. This function can be used in devices with or without a VIC.

Returns

None

C++

```
void CyU3PIrqContextRestore(  
    void  
);
```

Group

[Kernel Functions](#)

See Also

- [CyU3PIrqContextSave](#)
- [CyU3PIrqVectoredContextSave](#)

File

cyu3os.h

4.3.1.4 CyU3PFiqContextSave

This function saves the active thread context when entering the FIQ context.

This shall be invoked by the FIQ handler as the first call and saves the current thread state before handling the FIQ.

Returns

None

C++

```
void CyU3PFiqContextSave(  
    void  
);
```

Group

[Kernel Functions](#)

See Also

- [CyU3PFiqContextRestore](#)

File

cyu3os.h

4.3.1.5 CyU3PFiqContextRestore

This function restores the thread context after handling an FIQ interrupt.

This shall be invoked by the FIQ handlers as the last call. This is a non-returning call and restores the thread state according to the scheduling mechanism of the RTOS.

Returns

None

C++

```
void CyU3PFiqContextRestore(
    void
);
```

Group

[Kernel Functions](#)

See Also

- [CyU3PFiqContextSave](#)

File

cyu3os.h

4.3.1.6 CyU3PIrqNestingStart

This function switches the ARM mode from IRQ to SYS to allow nesting of interrupts.

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.

Returns

None

C++

```
void CyU3PIrqNestingStart(  
    void  
);
```

Group

[Kernel Functions](#)

See Also

- [CyU3PIrqNestingStop](#)

File

cyu3os.h

4.3.1.7 CyU3PIrqNestingStop

This function switches the ARM mode from SYS to IRQ at the end of an interrupt handler.

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.

Returns

None

C++

```
void CyU3PIrqNestingStop(  
    void  
);
```

Group

[Kernel Functions](#)

See Also

- [CyU3PIrqNestingStart](#)

File

cyu3os.h

4.3.1.8 CyU3PKernelEntry

RTOS kernel entry function.

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.

Returns

None - Function is a non returning function.

C++

```
void CyU3PKernelEntry(  
    void  
);
```

Group

[Kernel Functions](#)

See Also

- [CyU3PApplicationDefine](#)

File

cyu3os.h

4.3.1.9 CyU3PApplicationDefine

The driver specific OS primitives and threads shall be created in this function.

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.

Returns

None

C++

```
void CyU3PApplicationDefine(
    void
);
```

Group

[Kernel Functions](#)

See Also

- [CyU3PKernelEntry](#)

File

cyu3os.h

4.3.1.10 CyU3POsTimerHandler

The OS scheduler.

This function needs to be implemented in the RTOS kernel and this function shall be invoked each time the OS timer count value overflows. The timer interrupt handler is implemented as part of the firmware library and shall invoke this function.

Returns

None

C++

```
void CyU3POsTimerHandler(  
    void  
);
```

Group

[Kernel Functions](#)

File

cyu3os.h

4.3.1.11 CyU3POsTimerInit

Initialize the OS scheduler timer.

This function is implemented by the firmware library and is invoked from the kernel during initialization. The function is not expected to be invoked by the FX3 application unless the OS timer tick interval needs to be modified. It is recommended to run the OS timer tick at 1ms (default) interval for fast and accurate response. If the timer tick needs to be modified, the API can be invoked only after the RTOS has been initialized.

The OS_TIMER_TICK shall be defined by this function and all wait options will be calculated based on this. By default, the OS timer tick is configured to be 1ms. The timer interval value specified can 1ms to 1000ms. Any other value will set the tick timer to 1ms.

The clock is derived off the 32KHz standby clock. The actual time interval can have an error of upto +/- 4%. The time interval is accurate only as long as the interrupts are running.

Parameters

Parameters	Description
uint16_t intervalMs	OS Timer tick interval in millisecond.

Returns

None

C++

```
void CyU3POsTimerInit(
    uint16_t intervalMs
);
```

Group

[Kernel Functions](#)

File

cyu3os.h

4.3.2 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.

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 if 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 code expects that the following functions are provided by the application environment for use as a general purpose heap.

- [CyU3PMemInit](#)
- [CyU3PMemAlloc](#)
- [CyU3PMemFree](#)
- [CyU3PDmaBufferInit](#)
- [CyU3PDmaBufferDelnit](#)
- [CyU3PDmaBufferAlloc](#)
- [CyU3PDmaBufferFree](#)
- [CyU3PFreeHeaps](#)

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.

Functions

Function	Description
 CyU3PBytePoolCreate	This function creates and initializes a memory byte pool.
 CyU3PBytePoolDestroy	De-initialize and free up a memory byte pool.
 CyU3PByteAlloc	Allocate memory from a byte pool.
 CyU3PByteFree	Frees memory allocated by the CyU3PByteAlloc function.
 CyU3PBlockPoolCreate	Creates and initializes a memory block pool.
 CyU3PBlockPoolDestroy	De-initialize a block memory pool.
 CyU3PBlockAlloc	Allocate a memory buffer from a block pool.
 CyU3PBlockFree	Frees a memory buffer allocated from a block pool.
 CyU3PMemInit	Initialize the custom heap manager for OS specific allocation calls
 CyU3PFreeHeaps	Free up the custom heap manager and the buffer allocator.
 CyU3PMemAlloc	Allocate memory from the dynamic memory pool.
 CyU3PMemFree	Free memory allocated from the dynamic memory pool.
 CyU3PMemCmp	Compare the contents of two memory buffers.
 CyU3PMemSet	Fill a region of memory with a specified value.
 CyU3PMemCopy	Copy data from one memory location to another.

Group

[RTOS Functions](#)

Legend

	Method
---	--------

4.3.2.1 CyU3PBytePoolCreate

This function creates and initializes a memory byte pool.

This function is a wrapper around the byte pool service provided by the RTOS and creates a fixed size general purpose heap structure from which memory buffers of various sizes can be allocated. The size and location of the memory region to be used as the heap storage are passed in as parameters.

Parameters

Parameters	Description
CyU3PBytePool * pool_p	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.
void * poolStart	Pointer to memory region provided for the byte pool. This address needs to be 16 byte aligned.
uint32_t poolSize	Size of the memory region provided for the byte pool. This size needs to be a multiple of 16 bytes.

Returns

CY_U3P_SUCCESS if pool was successfully initialized. Other RTOS specific error codes in case of failure.

C++

```
uint32_t CyU3PBytePoolCreate(
    CyU3PBytePool * pool_p,
    void * poolStart,
    uint32_t poolSize
);
```

Group

[Memory Functions](#)

See Also

- [CyU3PBytePoolDestroy](#)
- [CyU3PByteAlloc](#)
- [CyU3PBlockFree](#)

File

cyu3os.h

4.3.2.2 CyU3PBytePoolDestroy

De-initialize and free up a memory byte pool.

This function cleans up the previously created byte pool pointed to by the pool_p structure. Memory allocated for the pool can be reused after this call returns.

Parameters

Parameters	Description
CyU3PBytePool * pool_p	Handle to the byte pool to be freed up

Returns

CY_U3P_SUCCESS on success, other RTOS error codes on failure.

C++

```
uint32_t CyU3PBytePoolDestroy(  
    CyU3PBytePool * pool_p  
);
```

Group

[Memory Functions](#)

See Also

- [CyU3PBytePoolCreate](#)
- [CyU3PByteAlloc](#)
- [CyU3PBlockFree](#)

File

cyu3os.h

4.3.2.3 CyU3PByteAlloc

Allocate memory from a byte pool.

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.

Parameters

Parameters	Description
CyU3PBytePool * pool_p	Handle to the byte pool to allocate memory from.
void ** mem_p	Output variable that points to the allocated memory buffer.
uint32_t memSize	Size of memory buffer required in bytes.
uint32_t waitOption	Timeout for memory allocation operation in terms of OS timer ticks. Can be set to CYU3P_NO_WAIT or CYU3P_WAIT_FOREVER.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PByteAlloc(
    CyU3PBytePool * pool_p,
    void ** mem_p,
    uint32_t memSize,
    uint32_t waitOption
);
```

Group

[Memory Functions](#)

See Also

- [CyU3PBytePoolCreate](#)
- [CyU3PBytePoolDestroy](#)
- [CyU3PByteFree](#)

File

cyu3os.h

4.3.2.4 CyU3PByteFree

Frees memory allocated by the [CyU3PByteAlloc](#) function.

This function frees memory allocated from a byte pool using the [CyU3PByteAlloc](#) function. This function can also be invoked from an interrupt context.

Parameters

Parameters	Description
<code>void * mem_p</code>	Pointer to memory buffer to be freed

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PByteFree(  
    void * mem_p  
);
```

Group

[Memory Functions](#)

See Also

- [CyU3PBytePoolCreate](#)
- [CyU3PBytePoolDestroy](#)
- [CyU3PByteAlloc](#)

File

cyu3os.h

4.3.2.5 CyU3PBlockPoolCreate

Creates and initializes a memory block pool.

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.

Parameters

Parameters	Description
CyU3PBlockPool * pool_p	Handle to block pool structure. The caller needs to allocate the structure, and this function only initializes the fields of the structure.
uint32_t blockSize	Size of memory blocks that this pool will manage. The block size needs to be a multiple of 16 bytes.
void * poolStart	Pointer to memory region provided for the block pool.
uint32_t poolSize	Size of memory region provided for the block pool.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PBlockPoolCreate(
    CyU3PBlockPool * pool_p,
    uint32_t blockSize,
    void * poolStart,
    uint32_t poolSize
);
```

Group

[Memory Functions](#)

See Also

- [CyU3PBlockPoolDestroy](#)
- [CyU3PBlockAlloc](#)
- [CyU3PBlockFree](#)

File

cyu3os.h

4.3.2.6 CyU3PBlockPoolDestroy

De-initialize a block memory pool.

This function de-initializes a memory block pool. The memory region used by the block pool can be re-used after this function returns.

Parameters

Parameters	Description
<code>CyU3PBlockPool * pool_p</code>	Handle to the block pool to be destroyed.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PBlockPoolDestroy(  
    CyU3PBlockPool * pool_p  
);
```

Group

[Memory Functions](#)

See Also

- [CyU3PBlockPoolCreate](#)
- [CyU3PBlockAlloc](#)
- [CyU3PBlockFree](#)

File

cyu3os.h

4.3.2.7 CyU3PBlockAlloc

Allocate a memory buffer from a block pool.

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.

Parameters

Parameters	Description
CyU3PBlockPool * pool_p	Handle to the memory block pool.
void ** block_p	Output variable that will be filled with a pointer to the allocated buffer.
uint32_t waitOption	Timeout duration in timer ticks.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PBlockAlloc(
    CyU3PBlockPool * pool_p,
    void ** block_p,
    uint32_t waitOption
);
```

Group

[Memory Functions](#)

See Also

- [CyU3PBlockPoolCreate](#)
- [CyU3PBlockPoolDestroy](#)
- [CyU3PBlockFree](#)

File

cyu3os.h

4.3.2.8 CyU3PBlockFree

Frees a memory buffer allocated from a block pool.

This function frees a memory buffer allocated through the [CyU3PBlockAlloc](#) call.

Parameters

Parameters	Description
void * block_p	Pointer to buffer to be freed.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PBlockFree(  
    void * block_p  
);
```

Group

[Memory Functions](#)

See Also

- [CyU3PBlockPoolCreate](#)
- [CyU3PBlockPoolDestroy](#)
- [CyU3PBlockAlloc](#)

File

cyu3os.h

4.3.2.9 CyU3PMemInit

Initialize the custom heap manager for OS specific allocation calls

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. User application must not invoke this call.

Returns

None.

C++

```
void CyU3PMemInit(
    void
);
```

Group

[Memory Functions](#)

See Also

- [CyU3PMemAlloc](#)
- [CyU3PMemFree](#)

File

cyu3os.h

4.3.2.10 CyU3PFreeHeaps

Free up the custom heap manager and the buffer allocator.

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.

Returns

None

C++

```
void CyU3PFreeHeaps(  
    void  
);
```

Group

[Memory Functions](#)

See Also

- [CyU3PMemInit](#)
- [CyU3PDmaBufferInit](#)

File

cyu3os.h

4.3.2.11 CyU3PMemAlloc

Allocate memory from the dynamic memory pool.

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.

Parameters

Parameters	Description
uint32_t size	The size of the buffer to be allocated in bytes.

Returns

Pointer to the allocated buffer, or NULL in case of alloc failure.

C++

```
void * CyU3PMemAlloc(
    uint32_t size
);
```

Group

[Memory Functions](#)

See Also

- [CyU3PMemInit](#)
- [CyU3PMemFree](#)

File

cyu3os.h

4.3.2.12 CyU3PMemFree

Free memory allocated from the dynamic memory pool.

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.

Parameters

Parameters	Description
<code>void * mem_p</code>	Pointer to memory buffer to be freed.

Returns

None

C++

```
void CyU3PMemFree(  
    void * mem_p  
);
```

Group

[Memory Functions](#)

See Also

- [CyU3PMemInit](#)
- [CyU3PMemAlloc](#)

File

`cyu3os.h`

4.3.2.13 CyU3PMemCmp

Compare the contents of two memory buffers.

This is a memcmp equivalent function that does a byte by byte comparison of two memory buffers. This function is provided and used by the firmware library.

Parameters

Parameters	Description
const void* s1	Pointer to first memory buffer.
const void* s2	Pointer to second memory buffer.
uint32_t n	Size of buffers to compare in bytes.

Returns

Zero : s1 == s2 Negative: s1 < s2 Positive: s1 > s2

C++

```
int32_t CyU3PMemCmp(
    const void* s1,
    const void* s2,
    uint32_t n
);
```

Group

[Memory Functions](#)

File

cyu3os.h

4.3.2.14 CyU3PMemSet

Fill a region of memory with a specified value.

This function is a memset equivalent and is used by the firmware library code. It can also be used by the application firmware.

Parameters

Parameters	Description
uint8_t * ptr	Pointer to the buffer to be filled.
uint8_t data	Value to fill the buffer with.
uint32_t count	Size of the buffer in bytes.

Returns

None

C++

```
void CyU3PMemSet(  
    uint8_t * ptr,  
    uint8_t data,  
    uint32_t count  
);
```

Group

[Memory Functions](#)

File

cyu3os.h

4.3.2.15 CyU3PMemCopy

Copy data from one memory location to another.

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.

Parameters

Parameters	Description
uint8_t * dest	Pointer to destination buffer.
uint8_t * src	Pointer to source buffer.
uint32_t count	Size of the buffer to be copied.

Returns

None

C++

```
void CyU3PMemCopy(
    uint8_t * dest,
    uint8_t * src,
    uint32_t count
);
```

Group

[Memory Functions](#)

File

cyu3os.h

4.3.3 Thread Functions












This section documents the thread functions that are provided as part of the FX3 firmware.

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.

Pre-emptive scheduling is typically used in the firmware, and time slices are typically not used. This means that thread switches will only happen when the active thread is blocked.


Functions

Function	Description
 CyU3PThreadCreate	This function creates a new thread.
 CyU3PThreadDestroy	Free up and remove a thread from the RTOS scheduler.
 CyU3PThreadIdentify	Get the thread structure corresponding to the current thread.
 CyU3PThreadInfoGet	Extract information regarding a specified thread.
 CyU3PThreadPriorityChange	Change the priority of a thread.
 CyU3PThreadSuspend	Suspends the specified thread.
 CyU3PThreadResume	Resume operation of a thread that was previously suspended.
 CyU3PThreadSleep	Puts a thread to sleep for the specified timer ticks.
 CyU3PThreadRelinquish	Relinquish control to the OS scheduler.
 CyU3PThreadWaitAbort	Returns a thread to ready state by aborting all waits on the thread.
 CyU3PThreadStackErrorNotify	Registers a callback function that will be notified of thread stack overflows.

Group

[RTOS Functions](#)

Legend

	Method
---	--------

4.3.3.1 CyU3PThreadCreate

This function creates a new thread.

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 are expected to be created in the [CyFxAApplicationDefine](#) function. This function is expected to be implemented in the application specific code and shall be invoked by the library after all the device modules have been initialized.

The application threads should take only priority values from 7 to 15. The higher priorities (0 - 6) are reserved for the driver threads.

Parameters

Parameters	Description
CyU3PThread * thread_p	Pointer to the thread structure. Memory for this structure has to be allocated by the caller.
char * threadName	Name string to associate with the thread. All threads should be named with the "Thread_Number:Description" convention.
CyU3PThreadEntry_t entryFn	Thread entry function.
uint32_t entryInput	Parameter to be passed into the thread entry function.
void * stackStart	Start address of the thread stack.
uint32_t stackSize	Size of the thread stack in bytes.
uint32_t priority	Priority to be assigned to this thread.
uint32_t preemptThreshold	Threshold value for thread pre-emption. Only threads with higher priorities than this value will be allowed to pre-empt this thread.
uint32_t timeSlice	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.
uint32_t autoStart	Whether this thread should be suspended or started immediately. Can be set to CYU3P_AUTO_START or CYU3P_DONT_START.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
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
);
```

Group

[Thread Functions](#)

See Also

- [CyU3PThreadDestroy](#)
- [CyU3PThreadIdentify](#)
- [CyU3PThreadInfoGet](#)
- [CyU3PThreadPriorityChange](#)
- [CyU3PThreadRelinquish](#)
- [CyU3PThreadSleep](#)
- [CyU3PThreadSuspend](#)
- [CyU3PThreadResume](#)
- [CyU3PThreadWaitAbort](#)
- [CyU3PThreadStackErrorNotify](#)

File

`cyu3os.h`

4.3.3.2 CyU3PThreadDestroy

Free up and remove a thread from the RTOS scheduler.

This function removes a previously created thread from the scheduler, and frees up the resources associated with it.

Parameters

Parameters	Description
CyU3PThread * thread_ptr	Pointer to the thread structure.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PThreadDestroy(
    CyU3PThread * thread_ptr
);
```

Group

[Thread Functions](#)

See Also

- [CyU3PThreadCreate](#)
- [CyU3PThreadIdentify](#)
- [CyU3PThreadInfoGet](#)
- [CyU3PThreadPriorityChange](#)
- [CyU3PThreadRelinquish](#)
- [CyU3PThreadSleep](#)
- [CyU3PThreadSuspend](#)
- [CyU3PThreadResume](#)
- [CyU3PThreadWaitAbort](#)
- [CyU3PThreadStackErrorNotify](#)

File

cyu3os.h

4.3.3.3 CyU3PThreadIdentify

Get the thread structure corresponding to the current thread.

This function returns a pointer to the thread structure corresponding to the active thread, or NULL if called from interrupt context.

Returns

Pointer to the thread structure of the currently running thread

C++

```
CyU3PThread * CyU3PThreadIdentify(  
    void  
);
```

Group

[Thread Functions](#)

See Also

- [CyU3PThreadCreate](#)
- [CyU3PThreadDestroy](#)
- [CyU3PThreadInfoGet](#)
- [CyU3PThreadPriorityChange](#)
- [CyU3PThreadRelinquish](#)
- [CyU3PThreadSleep](#)
- [CyU3PThreadSuspend](#)
- [CyU3PThreadResume](#)
- [CyU3PThreadWaitAbort](#)
- [CyU3PThreadStackErrorNotify](#)

File

cyu3os.h

4.3.3.4 CyU3PThreadInfoGet

Extract information regarding a specified thread.

This function is used to extract information about a thread whose pointer is provided. This function is used by the debug mechanism in the firmware library to extract information about the source thread which is queuing log messages.

The output variables can be set to NULL if the corresponding return value is not required.

Parameters

Parameters	Description
CyU3PThread * thread_p	Pointer to thread to be queried.
uint8_t ** name_p	Output variable to be filled with the thread's name string.
uint32_t * priority	Output variable to be filled with the thread priority.
uint32_t * preemptionThreshold	Output variable to be filled with the pre-emption threshold.
uint32_t * timeSlice	Output variable to be filled with the time slice value.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PThreadInfoGet(
    CyU3PThread * thread_p,
    uint8_t ** name_p,
    uint32_t * priority,
    uint32_t * preemptionThreshold,
    uint32_t * timeSlice
);
```

Group

[Thread Functions](#)

See Also

- [CyU3PThreadCreate](#)
- [CyU3PThreadDestroy](#)
- [CyU3PThreadIdentify](#)
- [CyU3PThreadPriorityChange](#)
- [CyU3PThreadRelinquish](#)
- [CyU3PThreadSleep](#)
- [CyU3PThreadSuspend](#)
- [CyU3PThreadResume](#)
- [CyU3PThreadWaitAbort](#)
- [CyU3PThreadStackErrorNotify](#)

File

cyu3os.h

4.3.3.5 CyU3PThreadPriorityChange

Change the priority of a thread.

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.

Parameters

Parameters	Description
CyU3PThread * thread_p	Pointer to the thread structure.
uint32_t newPriority	The new priority value to be set.
uint32_t * oldPriority	Pointer to a uint32_t to get the previous priority as return value.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PThreadPriorityChange(
    CyU3PThread * thread_p,
    uint32_t newPriority,
    uint32_t * oldPriority
);
```

Group

[Thread Functions](#)

See Also

- [CyU3PThreadCreate](#)
- [CyU3PThreadDestroy](#)
- [CyU3PThreadIdentify](#)
- [CyU3PThreadInfoGet](#)
- [CyU3PThreadRelinquish](#)
- [CyU3PThreadSleep](#)
- [CyU3PThreadSuspend](#)
- [CyU3PThreadResume](#)
- [CyU3PThreadWaitAbort](#)
- [CyU3PThreadStackErrorNotify](#)

File

cyu3os.h

4.3.3.6 CyU3PThreadSuspend

Suspends the specified thread.

This function is used to suspend a thread that is in the ready state, and can be called from any thread.

Parameters

Parameters	Description
CyU3PThread * thread_p	Pointer to the thread to be suspended.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PThreadSuspend(  
    CyU3PThread * thread_p  
);
```

Group

[Thread Functions](#)

See Also

- [CyU3PThreadCreate](#)
- [CyU3PThreadDestroy](#)
- [CyU3PThreadIdentify](#)
- [CyU3PThreadInfoGet](#)
- [CyU3PThreadPriorityChange](#)
- [CyU3PThreadRelinquish](#)
- [CyU3PThreadSleep](#)
- [CyU3PThreadResume](#)
- [CyU3PThreadWaitAbort](#)
- [CyU3PThreadStackErrorNotify](#)

File

cyu3os.h

4.3.3.7 CyU3PThreadResume

Resume operation of a thread that was previously suspended.

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.

Parameters

Parameters	Description
CyU3PThread * thread_p	Pointer to thread to be resumed.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PThreadResume(
    CyU3PThread * thread_p
);
```

Group

[Thread Functions](#)

See Also

- [CyU3PThreadCreate](#)
- [CyU3PThreadDestroy](#)
- [CyU3PThreadIdentify](#)
- [CyU3PThreadInfoGet](#)
- [CyU3PThreadPriorityChange](#)
- [CyU3PThreadRelinquish](#)
- [CyU3PThreadSleep](#)
- [CyU3PThreadSuspend](#)
- [CyU3PThreadWaitAbort](#)
- [CyU3PThreadStackErrorNotify](#)

File

cyu3os.h

4.3.3.8 CyU3PThreadSleep

Puts a thread to sleep for the specified timer ticks.

This function puts the current thread to sleep for the specified number of timer ticks.

Parameters

Parameters	Description
uint32_t timerTicks	Number of timer ticks to sleep.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PThreadSleep(  
    uint32_t timerTicks  
);
```

Group

[Thread Functions](#)

See Also

- [CyU3PThreadCreate](#)
- [CyU3PThreadDestroy](#)
- [CyU3PThreadIdentify](#)
- [CyU3PThreadInfoGet](#)
- [CyU3PThreadPriorityChange](#)
- [CyU3PThreadRelinquish](#)
- [CyU3PThreadSuspend](#)
- [CyU3PThreadResume](#)
- [CyU3PThreadWaitAbort](#)
- [CyU3PThreadStackErrorNotify](#)

File

cyu3os.h

4.3.3.9 CyU3PThreadRelinquish

Relinquish control to the OS scheduler.

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 be regain control if there are no other ready threads with the same priority level.

This function is useful in threads which take very little time for operation and can let other threads to function within the same timer tick.

Returns

None

C++

```
void CyU3PThreadRelinquish(
    void
);
```

Group

[Thread Functions](#)

See Also

- [CyU3PThreadCreate](#)
- [CyU3PThreadDestroy](#)
- [CyU3PThreadIdentify](#)
- [CyU3PThreadInfoGet](#)
- [CyU3PThreadPriorityChange](#)
- [CyU3PThreadSleep](#)
- [CyU3PThreadSuspend](#)
- [CyU3PThreadResume](#)
- [CyU3PThreadWaitAbort](#)
- [CyU3PThreadStackErrorNotify](#)

File

cyu3os.h

4.3.3.10 CyU3PThreadWaitAbort

Returns a thread to ready state by aborting all waits on the thread.

This function is used to restore a thread to ready state by aborting any waits that the thread is performing on Queues, Mutexes or Events. The wait operations will return with an error code that indicates that they were aborted.

Parameters

Parameters	Description
CyU3PThread * thread_p	Pointer to the thread to restore.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PThreadWaitAbort(  
    CyU3PThread * thread_p  
);
```

Group

[Thread Functions](#)

See Also

- [CyU3PThreadCreate](#)
- [CyU3PThreadDestroy](#)
- [CyU3PThreadIdentify](#)
- [CyU3PThreadInfoGet](#)
- [CyU3PThreadPriorityChange](#)
- [CyU3PThreadRelinquish](#)
- [CyU3PThreadSleep](#)
- [CyU3PThreadSuspend](#)
- [CyU3PThreadResume](#)
- [CyU3PThreadStackErrorNotify](#)

File

cyu3os.h

4.3.3.11 CyU3PThreadStackErrorNotify

Registers a callback function that will be notified of thread stack overflows.

This function is only provided in debug builds of the firmware and allows the application to register a callback function that will be called to notify when any thread in the system has encountered a stack overflow.

This API must be called before any user thread is created. It is recommended to invoke this API from [CyFxApplicationDefine\(\)](#)

Parameters

Parameters	Description
<code>CyU3PThreadStackErrorHandler_t errorHandler</code>	Pointer to thread stack error handler function.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PThreadStackErrorNotify(
    CyU3PThreadStackErrorHandler\_t errorHandler
);
```

Group

[Thread Functions](#)

See Also

- [CyU3PThreadCreate](#)
- [CyU3PThreadDestroy](#)
- [CyU3PThreadIdentify](#)
- [CyU3PThreadInfoGet](#)
- [CyU3PThreadPriorityChange](#)
- [CyU3PThreadRelinquish](#)
- [CyU3PThreadSleep](#)
- [CyU3PThreadSuspend](#)
- [CyU3PThreadResume](#)
- [CyU3PThreadWaitAbort](#)

File







cyu3os.h

4.3.4 Message Queue Functions

This section documents the message queue functions that are provided as part of the FX3 firmware.

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.


Functions

Function	Description
 CyU3PQueueCreate	Create a message queue.
 CyU3PQueueDestroy	Free up a previously created message queue.
 CyU3PQueueSend	Queue a new message on the specified message queue.
 CyU3PQueuePrioritySend	Add a new message at the head of a message queue.
 CyU3PQueueReceive	Receive a message from a message queue.
 CyU3PQueueFlush	Flushes all messages on a queue.

Group

[RTOS Functions](#)

Legend

	Method
---	--------

4.3.4.1 CyU3PQueueCreate

Create a message queue.

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.

Parameters

Parameters	Description
CyU3PQueue * queue_p	Pointer to the queue structure. This should be allocated by the caller and will be initialized by the queue create function.
uint32_t messageSize	Size of each message in 4 byte words. Allowed values are from 1 to 16 (4 bytes to 64 bytes).
void * queueStart	Pointer to memory buffer to be used for message storage.
uint32_t queueSize	Total size of the queue in bytes.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PQueueCreate(
    CyU3PQueue * queue_p,
    uint32_t messageSize,
    void * queueStart,
    uint32_t queueSize
);
```

Group

[Message Queue Functions](#)

See Also

- [CyU3PQueueDestroy](#)
- [CyU3PQueueSend](#)
- [CyU3PQueuePrioritySend](#)
- [CyU3PQueueReceive](#)
- [CyU3PQueueFlush](#)

File

cyu3os.h

4.3.4.2 CyU3PQueueDestroy

Free up a previously created message queue.

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.

Parameters

Parameters	Description
CyU3PQueue * queue_p	Pointer to the queue to be destroyed.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PQueueDestroy(  
    CyU3PQueue * queue_p  
);
```

Group

[Message Queue Functions](#)

See Also

- [CyU3PQueueCreate](#)
- [CyU3PQueueSend](#)
- [CyU3PQueuePrioritySend](#)
- [CyU3PQueueReceive](#)
- [CyU3PQueueFlush](#)

File

cyu3os.h

4.3.4.3 CyU3PQueueSend

Queue a new message on the specified message queue.

This function waits on and adds a new message on to the specified message queue. The amount of time to wait to try to enqueue the message 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.

Parameters

Parameters	Description
CyU3PQueue * queue_p	Queue to add a new message to.
void * src_p	Pointer to buffer containing message.
uint32_t waitOption	Timeout value to wait on the queue.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PQueueSend(
    CyU3PQueue * queue_p,
    void * src_p,
    uint32_t waitOption
);
```

Group

[Message Queue Functions](#)

See Also

- [CyU3PQueueCreate](#)
- [CyU3PQueueDestroy](#)
- [CyU3PQueuePrioritySend](#)
- [CyU3PQueueReceive](#)
- [CyU3PQueueFlush](#)

File

cyu3os.h

4.3.4.4 CyU3PQueuePrioritySend

Add a new message at the head of a message queue.

This function is used to send a high priority message to a message queue which will be placed at the head of the queue. The duration to wait for a free location on the queue needs to be specified through the waitOption parameter.

Parameters

Parameters	Description
CyU3PQueue * queue_p	Pointer to the message queue structure.
void * src_p	Pointer to message buffer.
uint32_t waitOption	Timeout duration in timer ticks.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PQueuePrioritySend(  
    CyU3PQueue * queue_p,  
    void * src_p,  
    uint32_t waitOption  
);
```

Group

[Message Queue Functions](#)

See Also

- [CyU3PQueueCreate](#)
- [CyU3PQueueDestroy](#)
- [CyU3PQueueSend](#)
- [CyU3PQueueReceive](#)
- [CyU3PQueueFlush](#)

File

cyu3os.h

4.3.4.5 CyU3PQueueReceive

Receive a message from a message queue.

This function waits on and tries to receive a message from the specified message queue. The message from the head of the queue will be returned when the queue is non-empty. The timeout duration should be set to CYU3P_NO_WAIT if this call is being made from an interrupt handler.

Parameters

Parameters	Description
CyU3PQueue * queue_p	Pointer to the message queue.
void * dest_p	Pointer to buffer where the message should be copied.
uint32_t waitOption	Timeout duration in timer ticks.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PQueueReceive(
    CyU3PQueue * queue_p,
    void * dest_p,
    uint32_t waitOption
);
```

Group

[Message Queue Functions](#)

See Also

- [CyU3PQueueCreate](#)
- [CyU3PQueueDestroy](#)
- [CyU3PQueueSend](#)
- [CyU3PQueuePrioritySend](#)
- [CyU3PQueueFlush](#)

File

cyu3os.h

4.3.4.6 CyU3PQueueFlush

Flushes all messages on a queue.

The function removes all waiting messages on the specified queue.

Parameters

Parameters	Description
CyU3PQueue * queue_p	Pointer to the queue to be flushed.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PQueueFlush(  
    CyU3PQueue * queue_p  
);
```

Group

[Message Queue Functions](#)

See Also

- [CyU3PQueueCreate](#)
- [CyU3PQueueDestroy](#)
- [CyU3PQueueSend](#)
- [CyU3PQueuePrioritySend](#)
- [CyU3PQueueReceive](#)

File


cyu3os.h




4.3.5 Semaphore Functions

This section documents the semaphore functions supported by the FX3 firmware.

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.

Functions

Function	Description
 CyU3PSemaphoreCreate	Create a semaphore object.

 CyU3PSemaphoreDestroy	Destroy a semaphore object.
 CyU3PSemaphoreGet	Get an instance from the specified counting semaphore.
 CyU3PSemaphorePut	Release an instance of the specified counting semaphore.

Group

[RTOS Functions](#)

Legend

	Method
---	--------

4.3.5.1 CyU3PSemaphoreCreate

Create a semaphore object.

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.

Parameters

Parameters	Description
CyU3PSemaphore * semaphore_p	Pointer to semaphore to be initialized.
uint32_t initialCount	Initial count to associate with semaphore.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PSemaphoreCreate(  
    CyU3PSemaphore * semaphore_p,  
    uint32_t initialCount  
);
```

Group

[Semaphore Functions](#)

See Also

- [CyU3PSemaphoreDestroy](#)
- [CyU3PSemaphoreGet](#)
- [CyU3PSemaphorePut](#)

File

cyu3os.h

4.3.5.2 CyU3PSemaphoreDestroy

Destroy a semaphore object.

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.

Parameters

Parameters	Description
CyU3PSemaphore * semaphore_p	Pointer to semaphore to be destroyed.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PSemaphoreDestroy(
    CyU3PSemaphore * semaphore_p
);
```

Group

[Semaphore Functions](#)

See Also

- [CyU3PSemaphoreCreate](#)
- [CyU3PSemaphoreGet](#)
- [CyU3PSemaphorePut](#)

File

cyu3os.h

4.3.5.3 CyU3PSemaphoreGet

Get an instance from the specified counting semaphore.

This function is used to get an instance (i.e., decrement the count by one) from the specified counting semaphore. The maximum interval to wait for is specified using the waitOption parameter.

Parameters

Parameters	Description
CyU3PSemaphore * semaphore_p	Pointer to semaphore to get.
uint32_t waitOption	Timeout duration in timer ticks.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PSemaphoreGet(  
    CyU3PSemaphore * semaphore_p,  
    uint32_t waitOption  
);
```

Group

[Semaphore Functions](#)

See Also

- [CyU3PSemaphoreCreate](#)
- [CyU3PSemaphoreDestroy](#)
- [CyU3PSemaphorePut](#)

File

cyu3os.h

4.3.5.4 CyU3PSemaphorePut

Release an instance of the specified counting semaphore.

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.

Parameters

Parameters	Description
CyU3PSemaphore * semaphore_p	Pointer to semaphore to put.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PSemaphorePut(
    CyU3PSemaphore * semaphore_p
);
```

Group

[Semaphore Functions](#)

See Also

- [CyU3PSemaphoreCreate](#)
- [CyU3PSemaphoreDestroy](#)
- [CyU3PSemaphoreGet](#)

File



cyu3os.h



4.3.6 Mutex functions

This section documents the mutex functions that are provided as part of the FX3 firmware.

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.

Functions

Function	Description
 CyU3PMutexCreate	Create a mutex variable.
 CyU3PMutexDestroy	Destroy a mutex variable.

 CyU3PMutexGet	Get the lock on a mutex variable.
 CyU3PMutexPut	Release the lock on a mutex variable.

Group

[RTOS Functions](#)

Legend

	Method
---	--------

4.3.6.1 CyU3PMutexCreate

Create a mutex variable.

This function creates a mutex variable. The mutex data structure has to be allocated by the caller, and will be initialized by the function.

Parameters

Parameters	Description
CyU3PMutex * mutex_p	Pointer to Mutex data structure to be initialized.
uint32_t priorityInherit	Whether priority inheritance should be allowed for this mutex. Allowed values for this parameter are CYU3P_NO_INHERIT and CYU3P_INHERIT.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PMutexCreate(
    CyU3PMutex * mutex_p,
    uint32_t priorityInherit
);
```

Group

[Mutex functions](#)

See Also

- [CyU3PMutexDestroy](#)
- [CyU3PMutexGet](#)
- [CyU3PMutexPut](#)

File

cyu3os.h

4.3.6.2 CyU3PMutexDestroy

Destroy a mutex variable.

This function destroys a mutex that was created using the [CyU3PMutexCreate](#) API.

Parameters

Parameters	Description
<code>CyU3PMutex * mutex_p</code>	Pointer to mutex to be destroyed.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PMutexDestroy(  
    CyU3PMutex * mutex_p  
);
```

Group

[Mutex functions](#)

See Also

- [CyU3PMutexCreate](#)
- [CyU3PMutexGet](#)
- [CyU3PMutexPut](#)

File

cyu3os.h

4.3.6.3 CyU3PMutexGet

Get the lock on a mutex variable.

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.

Parameters

Parameters	Description
CyU3PMutex * mutex_p	Pointer to mutex to be acquired.
uint32_t waitOption	Timeout duration in timer ticks.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PMutexGet(
    CyU3PMutex * mutex_p,
    uint32_t waitOption
);
```

Group

[Mutex functions](#)

See Also

- [CyU3PMutexCreate](#)
- [CyU3PMutexDestroy](#)
- [CyU3PMutexPut](#)

File

cyu3os.h

4.3.6.4 CyU3PMutexPut

Release the lock on a mutex variable.

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.

Parameters

Parameters	Description
CyU3PMutex * mutex_p	Pointer to mutex to be released.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PMutexPut(  
    CyU3PMutex * mutex_p  
);
```

Group

[Mutex functions](#)

See Also

- [CyU3PMutexCreate](#)
- [CyU3PMutexDestroy](#)
- [CyU3PMutexGet](#)

File

cyu3os.h

4.3.7 Event Flag Functions





This section documents the event flag functions provided by the FX3 firmware.

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.

Functions

Function	Description
 CyU3PEventCreate	Create an event flag group.
 CyU3PEventDestroy	Destroy an event flag group.
 CyU3PEventGet	Wait for or get the status of an event flag group.
 CyU3PEventSet	Update one or more flags in an event group.

Group

[RTOS Functions](#)

Legend

	Method
---	--------

4.3.7.1 CyU3PEventCreate

Create an event flag group.

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.

Parameters

Parameters	Description
CyU3PEvent * event_p	Pointer to event structure to be initialized.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PEventCreate(  
    CyU3PEvent * event_p  
);
```

Group

[Event Flag Functions](#)

See Also

- [CyU3PEventDestroy](#)
- [CyU3PEventSet](#)
- [CyU3PEventGet](#)

File

cyu3os.h

4.3.7.2 CyU3PEventDestroy

Destroy an event flag group.

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.

Parameters

Parameters	Description
CyU3PEvent * event_p	Pointer to event group to be destroyed.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PEventDestroy(
    CyU3PEvent * event_p
);
```

Group

[Event Flag Functions](#)

See Also

- [CyU3PEventCreate](#)
- [CyU3PEventSet](#)
- [CyU3PEventGet](#)

File

cyu3os.h

4.3.7.3 CyU3PEventGet

Wait for or get the status of an event flag group.

This function is used to get the current status of the flags in an 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.

Parameters

Parameters	Description
CyU3PEvent * event_p	Pointer to event group to be queried.
uint32_t rqtFlag	Bit-mask that selects event flags of interest. All bit positions corresponding to events of interest should be set to 1, and others should be cleared.
uint32_t getOption	Type of operation to perform. <ul style="list-style-type: none">• 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.
uint32_t * flag_p	Output parameter filled in with the state of the flags.
uint32_t waitOption	Timeout duration in timer ticks.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PEventGet(  
    CyU3PEvent * event_p,  
    uint32_t rqtFlag,  
    uint32_t getOption,  
    uint32_t * flag_p,  
    uint32_t waitOption  
);
```

Group

[Event Flag Functions](#)

See Also

- [CyU3PEventCreate](#)
- [CyU3PEventDestroy](#)
- [CyU3PEventSet](#)

File

cyu3os.h

4.3.7.4 CyU3PEventSet

Update one or more flags in an event group.

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.

Parameters

Parameters	Description
CyU3PEvent * event_p	Pointer to event group to update.
uint32_t rqtFlag	Bit-mask that selects event flags of interest.
uint32_t setOption	Type of set operation to perform. <ul style="list-style-type: none"> CYU3P_EVENT_AND: The rqtFlag will be ANDed with the current flags. CYU3P_EVENT_OR: The rqtFlag will be ORed with the current flags.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PEventSet(
    CyU3PEvent * event_p,
    uint32_t rqtFlag,
    uint32_t setOption
);
```

Group

[Event Flag Functions](#)

See Also

- [CyU3PEventCreate](#)
- [CyU3PEventDestroy](#)
- [CyU3PEventGet](#)

File

cyu3os.h








4.3.8 Timer Functions

This section documents the application timer functions that are provided by the FX3 firmware.

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.

Functions

Function	Description
 CyU3PTimerCreate	Create an application timer.
 CyU3PTimerDestroy	Destroy an application timer object.
 CyU3PTimerStart	Start an application timer.
 CyU3PTimerStop	Stop operation of an application timer.
 CyU3PTimerModify	Modify parameters of an application timer.
 CyU3PGetTime	Get time from reset in terms of OS timer ticks.
 CyU3PSetTime	Update the timer tick count.

Group

[RTOS Functions](#)

Legend

	Method
---	--------

4.3.8.1 CyU3PTimerCreate

Create an application timer.

This function creates an application timer than can be configured as a one-shot timer or as a auto-reload timer.

Parameters

Parameters	Description
CyU3PTimer * timer_p	Pointer to the timer structure to be initialized.
CyU3PTimerCb_t expirationFunction	Pointer to callback function called on timer expiration.
uint32_t expirationInput	Parameter to be passed to the callback function.
uint32_t initialTicks	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.
uint32_t rescheduleTicks	The reload value for the timer. If set to zero, the timer will be a one-shot timer.
uint32_t timerOption	Timer start control. <ul style="list-style-type: none"> CYU3P_AUTO_ACTIVATE: Start the timer immediately. CYU3P_NO_ACTIVATE: Wait until timer is started explicitly.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PTimerCreate(
    CyU3PTimer * timer_p,
    CyU3PTimerCb_t expirationFunction,
    uint32_t expirationInput,
    uint32_t initialTicks,
    uint32_t rescheduleTicks,
    uint32_t timerOption
);
```

Group

[Timer Functions](#)

See Also

- [CyU3PTimerCb_t](#)
- [CyU3PTimerDestroy](#)
- [CyU3PTimerStart](#)
- [CyU3PTimerStop](#)
- [CyU3PTimerModify](#)
- [CyU3PGetTime](#)
- [CyU3PSetTime](#)

File

cyu3os.h

4.3.8.2 CyU3PTimerDestroy

Destroy an application timer object.

This function destroys and application timer object.

Parameters

Parameters	Description
CyU3PTimer * timer_p	Pointer to the timer to be destroyed.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PTimerDestroy(
    CyU3PTimer * timer_p
);
```

Group

[Timer Functions](#)

See Also

- [CyU3PTimerCreate](#)
- [CyU3PTimerStart](#)
- [CyU3PTimerStop](#)
- [CyU3PTimerModify](#)
- [CyU3PGetTime](#)
- [CyU3PSetTime](#)

File

cyu3os.h

4.3.8.3 CyU3PTimerStart

Start an application timer.

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.

Parameters

Parameters	Description
CyU3PTimer * timer_p	Timer to be started.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PTimerStart(  
    CyU3PTimer * timer_p  
);
```

Group

[Timer Functions](#)

See Also

- [CyU3PTimerCreate](#)
- [CyU3PTimerDestroy](#)
- [CyU3PTimerStop](#)
- [CyU3PTimerModify](#)
- [CyU3PGetTime](#)
- [CyU3PSetTime](#)

File

cyu3os.h

4.3.8.4 CyU3PTimerStop

Stop operation of an application timer.

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.

Parameters

Parameters	Description
CyU3PTimer * timer_p	Pointer to timer to be stopped.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PTimerStop(
    CyU3PTimer * timer_p
);
```

Group

[Timer Functions](#)

See Also

- [CyU3PTimerCreate](#)
- [CyU3PTimerDestroy](#)
- [CyU3PTimerStart](#)
- [CyU3PTimerModify](#)
- [CyU3PGetTime](#)
- [CyU3PSetTime](#)

File

cyu3os.h

4.3.8.5 CyU3PTimerModify

Modify parameters of an application timer.

This function is used to modify the periodicity of an application timer and can be called only after stopping the timer.

Parameters

Parameters	Description
CyU3PTimer * timer_p	Pointer to the timer.
uint32_t initialTicks	Initial count to be set for the timer.
uint32_t rescheduleTicks	Reload count for the timer.

Returns

CY_U3P_SUCCESS (0) on success. Other error codes on failure.

C++

```
uint32_t CyU3PTimerModify(  
    CyU3PTimer * timer_p,  
    uint32_t initialTicks,  
    uint32_t rescheduleTicks  
);
```

Group

[Timer Functions](#)

See Also

- [CyU3PTimerCreate](#)
- [CyU3PTimerDestroy](#)
- [CyU3PTimerStart](#)
- [CyU3PTimerStop](#)
- [CyU3PGetTime](#)
- [CyU3PSetTime](#)

File

cyu3os.h

4.3.8.6 CyU3PGetTime

Get time from reset in terms of OS timer ticks.

The function returns relative tick count value.

Returns

Current OS timer tick count. Starts from zero on reset.

C++

```
uint32_t CyU3PGetTime(  
    void  
);
```

Group

[Timer Functions](#)

See Also

- [CyU3PTimerCreate](#)
- [CyU3PTimerDestroy](#)
- [CyU3PTimerStart](#)
- [CyU3PTimerStop](#)
- [CyU3PTimerModify](#)
- [CyU3PSetTime](#)

File

cyu3os.h

4.3.8.7 CyU3PSetTime

Update the timer tick count.

This function modifies the timer tick count that is started at system reset. This function should be called when the application gets defined as this affects all OS primitives.

Parameters

Parameters	Description
uint32_t newTime	New timer tick value to set.

Returns

None.

C++

```
void CyU3PSetTime(  
    uint32_t newTime  
);
```

Group

[Timer Functions](#)

See Also

- [CyU3PTimerCreate](#)
- [CyU3PTimerDestroy](#)
- [CyU3PTimerStart](#)
- [CyU3PTimerStop](#)
- [CyU3PTimerModify](#)
- [CyU3PGetTime](#)

File

cyu3os.h

5 DMA Management

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

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.

The DMA manager definition makes use of the following concepts.

- [DMA Socket](#)
- [DMA Buffer](#)
- [DMA Descriptor](#)
- [DMA Channel](#)

Topics

Topic	Description
DMA Socket	A DMA socket is a FX3 device construct that maps to one end of a data path into or out of the device.
DMA Buffer	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.
DMA Descriptor	A DMA descriptor is a data structure that binds the buffers and sockets corresponding to a data flow.
DMA Channel	A DMA channel is a software construct that encapsulates all of the hardware elements that are used in a single data flow.
DMA Data Types	This section documents the data types that are defined and used as part of the DMA manager module.
DMA Functions	This section documents the functions that are provided as part of the DMA manager module in the FX3 firmware library.

5.1 DMA Socket

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

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.

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

Group

[DMA Management](#)

5.2 DMA Buffer

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.

All data flows through the FX3 device pass 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.

Group

[DMA Management](#)

5.3 DMA Descriptor

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

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.

Group

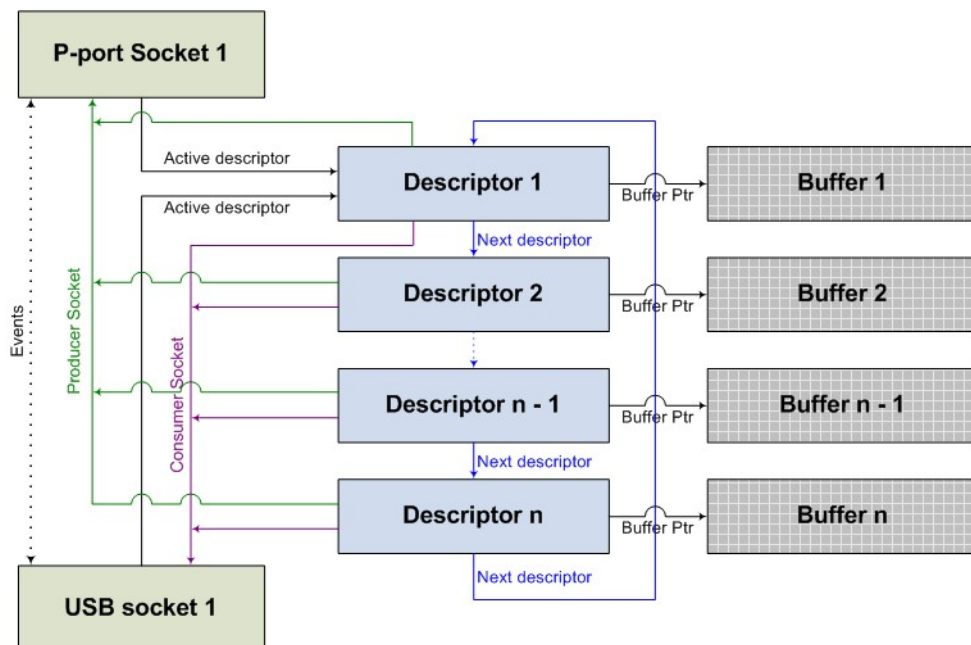
[DMA Management](#)

5.4 DMA Channel

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

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 device.

Resources used in a DMA channel and their relationships



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.

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.

Auto Channel with Signaling

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.

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 then 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.

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.

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 (Interleaved)

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 have different channel types such as auto or manual based on their operating model.

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 all consumers. This channel provides RAID1 functionality. The channel type has only manual configuration.

Channel States

A DMA channel (single or multi) transitions through the DMA channel states during its operation. After a DMA channel has been created, it is in the CY_U3P_DMA_CONFIGURED state. In the normal mode of operation, a SetXfer (single or multi channel) is allowed from this state. For override modes, SetupSendBuffer and SetupRecvBuffer (single or multi channel) are allowed from the CY_U3P_DMA_CONFIGURED state.

After the DMA operations are completed, the channel returns to the CY_U3P_DMA_CONFIGURED state. In the case of any error, the channel will transition to one of the error states. A ChannelReset (single or multi channel) will bring the channel back to the CY_U3P_DMA_CONFIGURED state.



Group







[DMA Management](#)

5.5 DMA Data Types

This section documents the data types that are defined and used as part of the DMA manager module.

Enumerations



Enumeration	Description
 CyU3PDmaSocketId_t	DMA socket IDs for all sockets in the device.
 CyU3PDmaType_t	Various DMA single channel types.

 CyU3PDmaMultiType_t	Various DMA multichannel types.
 CyU3PDmaState_t	Different DMA channel states.
 CyU3PDmaMode_t	Different DMA transfer modes.
 CyU3PDmaCbType_t	DMA channel callback input values.
 CyU3PDmaSckSuspType_t	DMA socket suspend options.
 CyU3PBlockId_t	DMA IP block IDs.

Group

DMA Management

Legend









	Enumeration
	Union

Macros

Macro	Description
CY_U3P_DMA_DSCR_COUNT	This is the number of descriptor in the pool. This value should not be modified.
CY_U3P_DMA_DSCR_SIZE	This is the size of the descriptor in memory. This value should not be modified.
CY_U3P_DMA_DSCR0_LOCATION	This is the location of the descriptor pool. These are special memory locations used for hardware configuration and this value must not be modified.
CY_U3P_DMA_LPP_MIN_CONS_SCK	The min consumer socket number for Serial IOs.
CY_U3P_DMA_LPP_MAX_CONS_SCK	The max consumer socket number for Serial IOs.
CY_U3P_DMA_LPP_MIN_PROD_SCK	The min producer socket number for Serial IOs.
CY_U3P_DMA_LPP_MAX_PROD_SCK	The max producer socket number for Serial IOs.
CY_U3P_DMA_LPP_NUM_SCK	The number of sockets for Serial IOs.
CY_U3P_DMA_PIB_MIN_CONS_SCK	The min consumer socket number for GPIF-II.
CY_U3P_DMA_PIB_MAX_CONS_SCK	The max consumer socket number for GPIF-II.
CY_U3P_DMA_PIB_MIN_PROD_SCK	The min producer socket number for GPIF-II.
CY_U3P_DMA_PIB_MAX_PROD_SCK	The max producer socket number for GPIF-II.
CY_U3P_DMA_PIB_NUM_SCK	The number of sockets for GPIF-II.
CY_U3P_DMA_UIB_MIN_CONS_SCK	The min consumer socket number for USB egress EPs.
CY_U3P_DMA_UIB_MAX_CONS_SCK	The max consumer socket number for USB egress EPs.
CY_U3P_DMA_UIB_NUM_SCK	The number of sockets for USB egress EPs.
CY_U3P_DMA_UIBIN_MIN_PROD_SCK	The min producer socket number for USB ingress EPs.
CY_U3P_DMA_UIBIN_MAX_PROD_SCK	The max producer socket number for USB ingress EPs.
CY_U3P_DMA_UIBIN_NUM_SCK	The number of sockets for USB ingress EPs.
CY_U3P_DMA_CPU_NUM_SCK	The number of sockets for CPU.
CY_U3P_IP_BLOCK_POS	The ip block position.
CY_U3P_IP_BLOCK_MASK	The ip block mask.
CY_U3P_DMA_SCK_MASK	The DMA socket mask.

CY_U3P_DMA_SCK_ID_MASK	The DMA socket id mask.
CyU3PDmaGetSckNum	Get the socket number from the socket ID.
CyU3PDmaGetIpNum	Get the ip number from the socket ID.
CyU3PDmaGetSckId	Get the socket ID from the socket number and the ip number.
CY_U3P_DMA_MAX_BUFFER_SIZE	Maximum allowed size for a single DMA buffer.
CY_U3P_DMA_BUFFER_AREA_BASE	Start address of the allowed buffer memory area for DMA operations.
CY_U3P_DMA_BUFFER_AREA_LIMIT	End address of the allowed buffer memory area for DMA operations.
CY_U3P_DMA_BUFFER_MARKER	The is a marker that is provided by s/w and can be observed by the IP. This bit is read only as far as the user application is concerned. It is used to indicate that a buffer has been marked for discarding. This bit has no effect on the hardware but is used by the DMA APIs.
CY_U3P_DMA_BUFFER_EOP	This is a marker indicating 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. The consuming IP observes this marker by inspecting its EOP return signal from its own DMA adapter.
CY_U3P_DMA_BUFFER_ERROR	Indicates the buffer data is valid (0) or in error (1).
CY_U3P_DMA_BUFFER_OCCUPIED	Indicates the buffer is in use (1) or empty (0).
CY_U3P_DMA_BUFFER_STATUS_MASK	The combined status field mask.
CY_U3P_DMA_BUFFER_STATUS_WRITE_MASK	The combined status field mask which can be modified by user application.
CY_U3P_DMA_MAX_AVAIL_COUNT	Maximum number of available buffers for a DMA channel. This is significant if prodAvailCount is non-zero.
CY_U3P_DMA_MAX_MULTI_SCK_COUNT	Macro defining the maximum allowed socket for multi socket DMA channels. Since the PIB has only four threads, the maximum allowed socket on one side is four.
CY_U3P_DMA_MIN_MULTI_SCK_COUNT	Macro defining the minimum required sockets for multi socket DMA channels. Since for single socket, single socket DMA channel can be used, the minimum number on multi side is 2.


Structures

Structure	Description
 CyU3PDmaChannel	DMA channel structure.
 CyU3PDmaChannelConfig_t	DMA channel parameter structure.
 CyU3PDmaMultiChannel	Multi socket DMA channel structure.
 CyU3PDmaMultiChannelConfig_t	Multi socket DMA channel parameter structure.
 CyU3PDmaBuffer_t	DMA channel buffer data structure.
 CyU3PDmaDescriptor_t	Descriptor data structure.
 CyU3PDmaSocketConfig_t	DMA socket configuration structure.
 CyU3PDmaSocket_t	DMA socket register structure.

Types

Type	Description
CyU3PDmaCallback_t	DMA channel callback function.
CyU3PDmaMultiCallback_t	Multi socket DMA channel callback function.
CyU3PDmaSocketCallback_t	DMA socket interrupt handler callback function.

Unions

Union	Description
 CyU3PDmaCBInput_t	DMA channel callback input.

5.5.1 CY_U3P_DMA_DSCR_COUNT

This is the number of descriptor in the pool. This value should not be modified.

C++

```
#define CY_U3P_DMA_DSCR_COUNT (512)
```

Group

[DMA Data Types](#)

File

cyu3descriptor.h

5.5.2 CY_U3P_DMA_DSCR_SIZE

This is the size of the descriptor in memory. This value should not be modified.

C++

```
#define CY_U3P_DMA_DSCR_SIZE (16)
```

Group

[DMA Data Types](#)

File

cyu3descriptor.h

5.5.3 CY_U3P_DMA_DSCR0_LOCATION

This is the location of the descriptor pool. These are special memory locations used for hardware configuration and this value must not be modified.

C++

```
#define CY_U3P_DMA_DSCR0_LOCATION (0x40000000)
```

Group

[DMA Data Types](#)

File

cyu3descriptor.h

5.5.4 CY_U3P_DMA_LPP_MIN_CONS_SCK

The min consumer socket number for Serial IOs.

C++

```
#define CY_U3P_DMA_LPP_MIN_CONS_SCK (0) /* The min consumer socket number for Serial  
IOs. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.5 CY_U3P_DMA_LPP_MAX_CONS_SCK

The max consumer socket number for Serial IOs.

C++

```
#define CY_U3P_DMA_LPP_MAX_CONS_SCK (4) /* The max consumer socket number for Serial  
IOs. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.6 CY_U3P_DMA_LPP_MIN_PROD_SCK

The min producer socket number for Serial IOs.

C++

```
#define CY_U3P_DMA_LPP_MIN_PROD_SCK (5) /* The min producer socket number for Serial  
IOs. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.7 CY_U3P_DMA_LPP_MAX_PROD_SCK

The max producer socket number for Serial IOs.

C++

```
#define CY_U3P_DMA_LPP_MAX_PROD_SCK (7) /* The max producer socket number for Serial  
IOs. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.8 CY_U3P_DMA_LPP_NUM_SCK

The number of sockets for Serial IOs.

C++

```
#define CY_U3P_DMA_LPP_NUM_SCK (8) /* The number of sockets for Serial IOs. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.9 CY_U3P_DMA_PIB_MIN_CONS_SCK

The min consumer socket number for GPIF-II.

C++

```
#define CY_U3P_DMA_PIB_MIN_CONS_SCK (0) /* The min consumer socket number for  
GPIF-II. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.10 CY_U3P_DMA_PIB_MAX_CONS_SCK

The max consumer socket number for GPIF-II.

C++

```
#define CY_U3P_DMA_PIB_MAX_CONS_SCK (15) /* The max consumer socket number for  
GPIF-II. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.11 CY_U3P_DMA_PIB_MIN_PROD_SCK

The min producer socket number for GPIF-II.

C++

```
#define CY_U3P_DMA_PIB_MIN_PROD_SCK (0) /* The min producer socket number for  
GPIF-II. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.12 CY_U3P_DMA_PIB_MAX_PROD_SCK

The max producer socket number for GPIF-II.

C++

```
#define CY_U3P_DMA_PIB_MAX_PROD_SCK (31) /* The max producer socket number for  
GPIF-II. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.13 CY_U3P_DMA_PIB_NUM_SCK

The number of sockets for GPIF-II.

C++

```
#define CY_U3P_DMA_PIB_NUM_SCK (32) /* The number of sockets for GPIF-II. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.14 CY_U3P_DMA_UIB_MIN_CONS_SCK

The min consumer socket number for USB egress EPs.

C++

```
#define CY_U3P_DMA_UIB_MIN_CONS_SCK (0) /* The min consumer socket number for USB  
egress EPs. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.15 CY_U3P_DMA_UIB_MAX_CONS_SCK

The max consumer socket number for USB egress EPs.

C++

```
#define CY_U3P_DMA_UIB_MAX_CONS_SCK (15) /* The max consumer socket number for USB egress EPs. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.16 CY_U3P_DMA_UIB_NUM_SCK

The number of sockets for USB egress EPs.

C++

```
#define CY_U3P_DMA_UIB_NUM_SCK (16) /* The number of sockets for USB egress EPs. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.17 CY_U3P_DMA_UIBIN_MIN_PROD_SCK

The min producer socket number for USB ingress EPs.

C++

```
#define CY_U3P_DMA_UIBIN_MIN_PROD_SCK (0) /* The min producer socket number for USB ingress EPs. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.18 CY_U3P_DMA_UIBIN_MAX_PROD_SCK

The max producer socket number for USB ingress EPs.

C++

```
#define CY_U3P_DMA_UIBIN_MAX_PROD_SCK (15) /* The max producer socket number for USB  
ingress EPs. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.19 CY_U3P_DMA_UIBIN_NUM_SCK

The number of sockets for USB ingress EPs.

C++

```
#define CY_U3P_DMA_UIBIN_NUM_SCK (16) /* The number of sockets for USB ingress EPs.  
*/
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.20 CY_U3P_DMA_CPU_NUM_SCK

The number of sockets for CPU.

C++

```
#define CY_U3P_DMA_CPU_NUM_SCK (2) /* The number of sockets for CPU. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.21 CY_U3P_IP_BLOCK_POS

The ip block position.

C++

```
#define CY_U3P_IP_BLOCK_POS (8) /* The ip block position. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.22 CY_U3P_IP_BLOCK_MASK

The ip block mask.

C++

```
#define CY_U3P_IP_BLOCK_MASK (0x3F) /* The ip block mask. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.23 CY_U3P_DMA_SCK_MASK

The DMA socket mask.

C++

```
#define CY_U3P_DMA_SCK_MASK (0xFF) /* The DMA socket mask. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.24 CY_U3P_DMA_SCK_ID_MASK

The DMA socket id mask.

C++

```
#define CY_U3P_DMA_SCK_ID_MASK (0x3FFF) /* The DMA socket id mask. */
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.25 CyU3PDmaGetSckNum

Get the socket number from the socket ID.

C++

```
#define CyU3PDmaGetSckNum(sckId) ((sckId) & CY\_U3P\_DMA\_SCK\_MASK)
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.26 CyU3PDmaGetIpNum

Get the ip number from the socket ID.

C++

```
#define CyU3PDmaGetIpNum(sckId) (((sckId) >> CY\_U3P\_IP\_BLOCK\_POS) &  
CY\_U3P\_IP\_BLOCK\_MASK)
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.27 CyU3PDmaGetSckId

Get the socket ID from the socket number and the ip number.

C++

```
#define CyU3PDmaGetSckId(ipNum, sckNum) (((sckNum) & CY_U3P_DMA_SCK_MASK) | (((ipNum) & CY_U3P_IP_BLOCK_MASK) << CY_U3P_IP_BLOCK_POS))
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.28 CY_U3P_DMA_MAX_BUFFER_SIZE

Maximum allowed size for a single DMA buffer.

C++

```
#define CY_U3P_DMA_MAX_BUFFER_SIZE (0xFFF0)
```

Group

[DMA Data Types](#)

File

cyu3dma.h

5.5.29 CY_U3P_DMA_BUFFER_AREA_BASE

Start address of the allowed buffer memory area for DMA operations.

C++

```
#define CY_U3P_DMA_BUFFER_AREA_BASE (uint8_t *) (0x40000000)
```

Group

[DMA Data Types](#)

File

cyu3dma.h

5.5.30 CY_U3P_DMA_BUFFER_AREA_LIMIT

End address of the allowed buffer memory area for DMA operations.

C++

```
#define CY_U3P_DMA_BUFFER_AREA_LIMIT (uint8_t *) (0x40080000)
```

Group

[DMA Data Types](#)

File

cyu3dma.h

5.5.31 CY_U3P_DMA_BUFFER_MARKER

There is a marker that is provided by s/w and can be observed by the IP. This bit is read only as far as the user application is concerned. It is used to indicate that a buffer has been marked for discarding. This bit has no effect on the hardware but is used by the DMA APIs.

C++

```
#define CY_U3P_DMA_BUFFER_MARKER (1u << 0)
```

Group

[DMA Data Types](#)

File

cyu3dma.h

5.5.32 CY_U3P_DMA_BUFFER_EOP

This is a marker indicating 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. The consuming IP observes this marker by inspecting its EOP return signal from its own DMA adapter.

C++

```
#define CY_U3P_DMA_BUFFER_EOP (1u << 1)
```

Group

[DMA Data Types](#)

File

cyu3dma.h

5.5.33 CY_U3P_DMA_BUFFER_ERROR

Indicates the buffer data is valid (0) or in error (1).

C++

```
#define CY_U3P_DMA_BUFFER_ERROR (1u << 2)
```

Group

[DMA Data Types](#)

File

cyu3dma.h

5.5.34 CY_U3P_DMA_BUFFER_OCCUPIED

Indicates the buffer is in use (1) or empty (0).

C++

```
#define CY_U3P_DMA_BUFFER_OCCUPIED (1u << 3)
```

Group

[DMA Data Types](#)

File

cyu3dma.h

5.5.35 CY_U3P_DMA_BUFFER_STATUS_MASK

The combined status field mask.

C++

```
#define CY_U3P_DMA_BUFFER_STATUS_MASK (0x000F)
```

Group

[DMA Data Types](#)

File

cyu3dma.h

5.5.36 CY_U3P_DMA_BUFFER_STATUS_WRITE_MASK

The combined status field mask which can be modified by user application.

C++

```
#define CY_U3P_DMA_BUFFER_STATUS_WRITE_MASK (0x000E)
```

Group

[DMA Data Types](#)

File

cyu3dma.h

5.5.37 CY_U3P_DMA_MAX_AVAIL_COUNT

Maximum number of available buffers for a DMA channel. This is significant if prodAvailCount is non-zero.

C++

```
#define CY_U3P_DMA_MAX_AVAIL_COUNT (31)
```

Group

[DMA Data Types](#)

File

cyu3dma.h

5.5.38 CY_U3P_DMA_MAX_MULTI_SCK_COUNT

Macro defining the maximum allowed socket for multi socket DMA channels. Since the PIB has only four threads, the maximum allowed socket on one side is four.

C++

```
#define CY_U3P_DMA_MAX_MULTI_SCK_COUNT (4)
```

Group

[DMA Data Types](#)

File

cyu3dma.h

5.5.39 CY U3P DMA MIN MULTI SCK COUNT

Macro defining the minimum required sockets for multi socket DMA channels. Since for single socket, single socket DMA channel can be used, the minimum number on multi side is 2.

C++

```
#define CY_U3P_DMA_MIN_MULTI_SCK_COUNT (2)
```

Group

DMA Data Types

File

cyu3dma.h

5.5.40 CyU3PDmaSocketId_t

DMA socket IDs for all sockets in the device.

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 USB egress is 3 and USB ingress is 4.

Each peripheral has a number of sockets. The LPP sockets are fixed and has 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 cannot be used for CPU-CPU transfers.

C++

```
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_UIB_SOCKET_CONS_0 = 0x0300,  
CY_U3P_UIB_SOCKET_CONS_1,  
CY_U3P_UIB_SOCKET_CONS_2,  
CY_U3P_UIB_SOCKET_CONS_3,  
CY_U3P_UIB_SOCKET_CONS_4,  
CY_U3P_UIB_SOCKET_CONS_5,  
CY_U3P_UIB_SOCKET_CONS_6,  
CY_U3P_UIB_SOCKET_CONS_7,  
CY_U3P_UIB_SOCKET_CONS_8,  
CY_U3P_UIB_SOCKET_CONS_9,  
CY_U3P_UIB_SOCKET_CONS_10,  
CY_U3P_UIB_SOCKET_CONS_11,  
CY_U3P_UIB_SOCKET_CONS_12,  
CY_U3P_UIB_SOCKET_CONS_13,  
CY_U3P_UIB_SOCKET_CONS_14,  
CY_U3P_UIB_SOCKET_CONS_15,  
CY_U3P_UIB_SOCKET_PROD_0 = 0x400,  
CY_U3P_UIB_SOCKET_PROD_1,  
CY_U3P_UIB_SOCKET_PROD_2,  
CY_U3P_UIB_SOCKET_PROD_3,  
CY_U3P_UIB_SOCKET_PROD_4,  
CY_U3P_UIB_SOCKET_PROD_5,  
CY_U3P_UIB_SOCKET_PROD_6,  
CY_U3P_UIB_SOCKET_PROD_7,  
CY_U3P_UIB_SOCKET_PROD_8,  
CY_U3P_UIB_SOCKET_PROD_9,  
CY_U3P_UIB_SOCKET_PROD_10,  
CY_U3P_UIB_SOCKET_PROD_11,  
CY_U3P_UIB_SOCKET_PROD_12,  
CY_U3P_UIB_SOCKET_PROD_13,  
CY_U3P_UIB_SOCKET_PROD_14,  
CY_U3P_UIB_SOCKET_PROD_15,  
CY_U3P_CPU_SOCKET_CONS = 0x3F00,  
CY_U3P_CPU_SOCKET_PROD  
};
```

Group

[DMA Data Types](#)

See Also

- [CyU3PDmaChannelConfig_t](#)
- [CyU3PDmaMultiChannelConfig_t](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaMultiChannelCreate](#)

Members

Members	Description
CY_U3P_LPP_SOCKET_I2S_LEFT = 0x0000	Left channel output to I2S port.
CY_U3P_LPP_SOCKET_I2S_RIGHT	Right channel output to I2S port.
CY_U3P_LPP_SOCKET_I2C_CONS	Outgoing data to I2C slave.
CY_U3P_LPP_SOCKET_UART_CONS	Outgoing data to UART peer.
CY_U3P_LPP_SOCKET_SPI_CONS	Outgoing data to SPI slave.
CY_U3P_LPP_SOCKET_I2C_PROD	Incoming data from I2C slave.
CY_U3P_LPP_SOCKET_UART_PROD	Incoming data from UART peer.
CY_U3P_LPP_SOCKET_SPI_PROD	Incoming data from SPI slave.
CY_U3P_PIB_SOCKET_0 = 0x0100	P-port socket number 0.
CY_U3P_PIB_SOCKET_1	P-port socket number 1.
CY_U3P_PIB_SOCKET_2	P-port socket number 2.
CY_U3P_PIB_SOCKET_3	P-port socket number 3.
CY_U3P_PIB_SOCKET_4	P-port socket number 4.
CY_U3P_PIB_SOCKET_5	P-port socket number 5.
CY_U3P_PIB_SOCKET_6	P-port socket number 6.
CY_U3P_PIB_SOCKET_7	P-port socket number 7.
CY_U3P_PIB_SOCKET_8	P-port socket number 8.
CY_U3P_PIB_SOCKET_9	P-port socket number 9.
CY_U3P_PIB_SOCKET_10	P-port socket number 10.
CY_U3P_PIB_SOCKET_11	P-port socket number 11.
CY_U3P_PIB_SOCKET_12	P-port socket number 12.
CY_U3P_PIB_SOCKET_13	P-port socket number 13.
CY_U3P_PIB_SOCKET_14	P-port socket number 14.
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_UIB_SOCKET_CONS_0 = 0x0300	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 = 0x400	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 = 0x3F00	Socket through which the FX3 CPU receives data.
CY_U3P_CPU_SOCKET_PROD	Socket through which the FX3 CPU produces data.

File

cyu3dma.h

5.5.41 CyU3PDmaType_t

Various DMA single channel types.

A DMA channel needs to be created before any DMA operation can be made. The following are the different types of DMA single (one to one) channels. The single channel types are the simple cases and all APIs for this has a [CyU3PDmaChannel](#) prefix.

C++

```
enum CyU3PDmaType_t {  
    CY_U3P_DMA_TYPE_AUTO = 0,  
    CY_U3P_DMA_TYPE_AUTO_SIGNAL,  
    CY_U3P_DMA_TYPE_MANUAL,  
    CY_U3P_DMA_TYPE_MANUAL_IN,  
    CY_U3P_DMA_TYPE_MANUAL_OUT,  
    CY_U3P_DMA_NUM_SINGLE_TYPES  
};
```

Group

[DMA Data Types](#)

See Also

- [CyU3PDmaChannelCreate](#)

Members

Members	Description
CY_U3P_DMA_TYPE_AUTO = 0	Auto mode DMA channel.
CY_U3P_DMA_TYPE_AUTO_SIGNAL	Auto mode with produce 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.

File

cyu3dma.h

5.5.42 CyU3PDmaMultiType_t

Various DMA multichannel types.

A DMA channel needs to be created before any DMA operation can be made. The following are the different types

of DMA multichannels. The multiple socket channel types are special cases and separate create calls must be made for this. All APIs for multi socket channels are different and has the [CyU3PDmaMultiChannel](#) prefix.

C++

```
enum CyU3PDmaMultiType_t {  
    CY_U3P_DMA_TYPE_AUTO_MANY_TO_ONE = (CY_U3P_DMA_NUM_SINGLE_TYPES),  
    CY_U3P_DMA_TYPE_AUTO_ONE_TO_MANY,  
    CY_U3P_DMA_TYPE_MANUAL_MANY_TO_ONE,  
    CY_U3P_DMA_TYPE_MANUAL_ONE_TO_MANY,  
    CY_U3P_DMA_TYPE_MULTICAST,  
    CY_U3P_DMA_NUM_TYPES  
};
```

Group

[DMA Data Types](#)

See Also

- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaEnableMulticast](#)

Members

Members	Description
CY_U3P_DMA_TYPE_AUTO_MANY_TO_ONE = (CY_U3P_DMA_NUM_SINGLE_TYPES)	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 function needs to be called before any multicast DMA channels are created.
CY_U3P_DMA_NUM_TYPES	Number of DMA channel types.

File

cyu3dma.h

5.5.43 CyU3PDmaState_t

Different DMA channel states.

The following are the different states that a DMA channel can be in. All states until CY_U3P_DMA_NUM_STATES are channel states whereas those after it are virtual state values returned on the GetStatus calls.

C++

```
enum CyU3PDmaState_t {  
    CY_U3P_DMA_NOT_CONFIGURED = 0,  
    CY_U3P_DMA_CONFIGURED,
```

```

CY_U3P_DMA_ACTIVE,
CY_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
};

```

Group

DMA Data Types

See Also

- [CyU3PDmaChannelGetStatus](#)

Members

Members	Description
CY_U3P_DMA_NOT_CONFIGURED = 0	DMA channel is unconfigured. This state occurs only when using stale 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: <ol style="list-style-type: none"> 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. This is just a place holder.
CY_U3P_DMA_XFER_COMPLETED	This is 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 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 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.

File

cyu3dma.h

5.5.44 CyU3PDmaMode_t

Different DMA transfer modes.

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.

C++

```
enum CyU3PDmaMode_t {  
    CY_U3P_DMA_MODE_BYTE = 0,  
    CY_U3P_DMA_MODE_BUFFER,  
    CY_U3P_DMA_NUM_MODES  
};
```

Group

[DMA Data Types](#)

See Also

- [CyU3PDmaChannelConfig_t](#)
- [CyU3PDmaMultiChannelConfig_t](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaMultiChannelUpdateMode](#)

Members

Members	Description
CY_U3P_DMA_MODE_BYTE = 0	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.

File

cyu3dma.h

5.5.45 CyU3PDmaCbType_t

DMA channel callback input values.

These are the parameters with which a DMA channel callback will be made. This is used to identify the DMA event. This is also used to generate the notification bit mask used for DMA channel configuration.

C++

```
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)  
};
```

Group

[DMA Data Types](#)

See Also

- [CyU3PDmaCallback_t](#)
- [CyU3PDmaChannelConfig_t](#)
- [CyU3PDmaMultiChannelConfig_t](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaMultiChannelCreate](#)

Members

Members	Description
CY_U3P_DMA_CB_XFER_CPLT = (1 << 0)	Transfer has been completed. This event is generated when a finite transfer queued with SetXfer is completed.
CY_U3P_DMA_CB_SEND_CPLT = (1 << 1)	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 = (1 << 2)	ReceiverBuffer call has been completed. This event is generated when data is received successfully on the producer socket.
CY_U3P_DMA_CB_PROD_EVENT = (1 << 3)	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 = (1 << 4)	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 = (1 << 5)	This event is generated when the Abort API is invoked.
CY_U3P_DMA_CB_ERROR = (1 << 6)	This event is generated when the hardware detects an error.
CY_U3P_DMA_CB_PROD_SUSP = (1 << 7)	This event is generated when the producer socket is suspended.
CY_U3P_DMA_CB_CONS_SUSP = (1 << 8)	This event is generated when the consumer socket is suspended.

File

cyu3dma.h

5.5.46 CyU3PDmaCBInput_t

DMA channel callback input.

The callback function registered for each DMA channel shall be used for various notifications and the input values for each notifications shall be passed to the callback via this union.

C++

```
union CyU3PDmaCBInput_t {  
    CyU3PDmaBuffer\_t buffer_p;  
};
```

Group

[DMA Data Types](#)

See Also

- [CyU3PDmaBuffer_t](#)
- [CyU3PDmaCallback_t](#)

Members

Members	Description
CyU3PDmaBuffer_t buffer_p;	Buffer details.

File

cyu3dma.h

5.5.47 CyU3PDmaSckSuspType_t

DMA socket suspend options.

The producer and consume sockets can be suspended on various options. Each of the suspend option behaves differently.

CY_U3P_DMA_SCK_SUSP_NONE No suspend option is applied. This is the default for all channels created.

CY_U3P_DMA_SCK_SUSP_EOP This option will suspend the socket after the current buffer with end of packet signalling. The buffer will not be stopped. This can be used if EOP is rare and has special value. This cannot be used for making any data modifications. This option is sticky and will not change until explicitly changed.

CY_U3P_DMA_SCK_SUSP_CUR_BUF This option is used to suspend the socket after the current buffer handling is completed. The buffer cannot be stopped or modified. 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 resume.

CY_U3P_DMA_SCK_SUSP_CONS_PARTIAL_BUF This option is valid only for consumer socket. The socket will be suspended before the data is transferred. This allows for data manipulation. This allows only very restricted access to buffer.

In case of multi channels, only the single socket side can be suspended. For a many to one channel, the producer sockets cannot be suspended and for one to many channel, the consumer sockets cannot be suspended.

C++

```
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  
};
```

Group

[DMA Data Types](#)

See Also

- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelSetSuspend](#)
- [CyU3PDmaChannelResume](#)
- [CyU3PDmaMultiChannelCreate](#)

Members

Members	Description
CY_U3P_DMA_SCK_SUSP_NONE = 0	Socket will not be suspended.
CY_U3P_DMA_SCK_SUSP_EOP	Socket will be suspended after handling buffer with EOP.
CY_U3P_DMA_SCK_SUSP_CUR_BUF	Socket will be suspended after the current buffer is completed.
CY_U3P_DMA_SCK_SUSP_CONS_PARTIAL_BUF	Consumer socket will be suspended before handling a partial buffer.

File

cyu3dma.h

5.5.48 CyU3PDmaCallback_t

DMA channel callback function.

The callback function that has to be registered with every DMA channel. If no callback function is registered or the corresponding notification events are not registered, the callback will not be made.

No blocking calls must be made from the callback functions. If data processing is required, it must be done outside of the callback function.

In case of produce events, 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 case of AUTO_SIGNAL channels, the input parameter points to the latest produced buffer. If the handling is delayed, the producer socket can potentially overwrite this buffer.

In case of MANUAL or MANUAL_IN channels, the input parameter points to 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. If data processing has to be done in MANUAL / MANUAL_IN channels, the [CyU3PDmaChannelGetBuffer](#) function must be used and the callback function must be just used as a notification.

C++

```
typedef void (* CyU3PDmaCallback_t)(CyU3PDmaChannel *handle, CyU3PDmaCbType\_t type, CyU3PDmaCBInput\_t *input);
```

Group

[DMA Data Types](#)

See Also

- [CyU3PDmaCBInput_t](#)
- [CyU3PDmaChannelConfig_t](#)
- [CyU3PDmaChannelCreate](#)

File

cyu3dma.h

5.5.49 CyU3PDmaMultiCallback_t

Multi socket DMA channel callback function.

The callback function that has to be registered with every DMA channel. If no callback function is registered or the corresponding notification events are not registered, the callback will not be made.

No blocking calls must be made from the callback functions. If data processing is required, it must be done outside of the callback function.

In case of produce events, the application is expected to be fast enough. The input parameter can potentially be stale, if the handling of the buffer is not done in a timely fashion.

In case of MANUAL channels, the input parameter points to 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. If data processing has to be done, the [CyU3PDmaMultiChannelGetBuffer](#) function must be used and the callback function must be just used as a notification.

C++

```
typedef void (* CyU3PDmaMultiCallback_t)(CyU3PDmaMultiChannel *handle,  
CyU3PDmaCbType\_t type, CyU3PDmaCBInput\_t *input);
```

Group

[DMA Data Types](#)

See Also

- [CyU3PDmaCBInput_t](#)
- [CyU3PDmaMultiChannelConfig_t](#)
- [CyU3PDmaMultiChannelCreate](#)

File

cyu3dma.h

5.5.50 CyU3PDmaSocketCallback_t

DMA socket interrupt handler callback function.

When a custom DMA interface needs to be built, the channel interface cannot be used and [CyU3PDmaSocketRegisterCallback](#) function needs to be invoked to set this. The function returns the interrupt status of the socket triggering the interrupt.

C++

```
typedef void (* CyU3PDmaSocketCallback_t)(uint16_t sckId, uint32_t status);
```

Group

[DMA Data Types](#)

File

cyu3socket.h

5.5.51 CyU3PBlockId_t

DMA IP block IDs.

Each IP block has an ID which is required in identifying its DMA sockets. The socket ID is made of the block ID and the socket number.

C++

```
enum CyU3PBlockId_t {  
    CY_U3P_LPP_IP_BLOCK_ID = 0,  
    CY_U3P_PIB_IP_BLOCK_ID = 1,  
    CY_U3P_RES_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  
};
```

Group

[DMA Data Types](#)

See Also

- [CyU3PDmaSocketId_t](#)

Members

Members	Description
CY_U3P_LPP_IP_BLOCK_ID = 0	LPP block contains UART, I2C, I2S and SPI.
CY_U3P_PIB_IP_BLOCK_ID = 1	P-port interface block.
CY_U3P_RES_IP_BLOCK_ID = 2	Reserved block ID.
CY_U3P_UIB_IP_BLOCK_ID = 3	USB interface block for egress sockets.
CY_U3P_UIBIN_IP_BLOCK_ID = 4	USB interface block for ingress sockets.
CY_U3P_NUM_IP_BLOCK_ID = 5	Sentinel value. Count of valid DMA IPs.
CY_U3P_CPU_IP_BLOCK_ID = 0x3F	Special value used to denote CPU as DMA endpoint.

File

cyu3socket.h

5.5.52 CyU3PDmaChannel

DMA channel structure.

Every DMA channel shall have a channel structure associated with it for storing info and status. This structure is updated and maintained by the library APIs and should not be modified by the application.

C++

```
struct CyU3PDmaChannel {  
    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 discardCount;
uint32_t notification;
uint32_t xferSize;
CyBool_t isDmaHandledCache;
CyU3PMutex lock;
CyU3PEvent flags;
CyU3PDmaCallback\_t cb;
};
```

Group

[DMA Data Types](#)

See Also

- [CyU3PDmaChannelConfig_t](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelDestroy](#)
- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelGetHandle](#)
- [CyU3PDmaChannelSetXfer](#)
- [CyU3PDmaChannelGetBuffer](#)
- [CyU3PDmaChannelCommitBuffer](#)
- [CyU3PDmaChannelDiscardBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)
- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelSetSuspend](#)
- [CyU3PDmaChannelResume](#)

- [CyU3PDmaChannelAbort](#)
- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

Members

Members	Description
uint32_t state;	The current state of the DMA channel
uint16_t type;	The type of the DMA channel
uint16_t size;	The buffer size associated with the channel
uint16_t count;	Number of buffers for the channel
uint16_t prodAvailCount;	Minimum available buffers before producer is active.
uint16_t firstProdIndex;	Head for the normal descriptor chain list
uint16_t firstConsIndex;	Head for the manual mode consumer descriptor chain list
uint16_t prodSckId;	The producer (ingress) socket ID
uint16_t consSckId;	The consumer (egress) socket ID
uint16_t overrideDscrIndex;	Internal data - Descriptor for override modes.
uint16_t currentProdIndex;	Internal data - Producer descriptor for the latest buffer produced.
uint16_t currentConsIndex;	Internal data - Consumer descriptor for the latest buffer produced.
uint16_t commitProdIndex;	Internal data - Producer descriptor for the buffer to be consumed.
uint16_t commitConsIndex;	Internal data - Consumer descriptor for the buffer to be consumed.
uint16_t activeProdIndex;	Internal data - Active producer descriptor.
uint16_t activeConsIndex;	Internal data - Active consumer descriptor.
uint16_t prodHeader;	The producer socket header offset
uint16_t prodFooter;	The producer socket footer offset
uint16_t consHeader;	The consumer socket header offset
uint16_t dmaMode;	Mode of DMA operation
uint16_t prodSusp;	Internal data - producer suspend option.
uint16_t consSusp;	Internal data - consumer suspend option.
uint16_t discardCount;	Internal data - number of pending discards.
uint32_t notification;	Registered notifications
uint32_t xferSize;	Internal data - current xfer size
CyBool_t isDmaHandleDCache;	Whether to do internal DMA cache handling.
CyU3PMutex lock;	Internal data - lock for the channel
CyU3PEvent flags;	Internal data - event flags for the channel
CyU3PDmaCallback_t cb;	Callback function which gets invoked on DMA events

File

cyu3dma.h

5.5.53 CyU3PDmaChannelConfig_t

DMA channel parameter structure.

This is the channel configuration structure which needs to be provided during channel initialization.

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

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.

If the data cache is enabled, the buffers should be 32 byte aligned and a multiple of 32 byte. This is because the cache line is 32 byte long. The buffers allocated by [CyU3PDmaBufferAlloc](#) function takes care of this. The size field need not be a multiple of 32 as the DMA API allocates channel memory from using the [CyU3PDmaBufferAlloc](#) call.

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 CommitBuffer 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 CommitBuffer 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.

More complex modes can be devised by enabling more than one offset.

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.

C++

```
struct CyU3PDmaChannelConfig_t {
    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;
};
```


Group[DMA Data Types](#)**See Also**

- [CyU3PDmaChannelCreate](#)

Members

Members	Description
uint16_t size;	The buffer size associated with the channel.
uint16_t count;	Number of buffers to be allocated for the channel. The count can be zero for MANUAL, MANUAL_OUT and MANUAL_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.
CyU3PDmaSocketId_t prodSckId;	The producer (ingress) socket ID.
CyU3PDmaSocketId_t consSckId;	The consumer (egress) socket ID.
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.
uint16_t prodHeader;	The producer socket header offset.
uint16_t prodFooter;	The producer socket footer offset.
uint16_t consHeader;	The consumer socket header offset.
CyU3PDmaMode_t dmaMode;	Mode of DMA operation.
uint32_t notification;	Registered notifications. This is a bit map based on CyU3PDmaCbType_t . This defines the events for which the callback is triggered.
CyU3PDmaCallback_t cb;	Callback function which gets invoked on DMA events.

File

cyu3dma.h

5.5.54 CyU3PDmaMultiChannel

Multi socket DMA channel structure.

Every DMA channel shall have a channel structure associated with it for storing info and status. This structure is updated and maintained by the library APIs and should not be modified by the application. This structure is for multi-socket operations.

C++

```
struct CyU3PDmaMultiChannel {  
    uint32_t state;  
    uint16_t type;
```

```

uint16_t size;
uint16_t count;
uint16_t validSckCount;
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 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;
};

```

Group

[DMA Data Types](#)

See Also

- [CyU3PDmaMultiChannelConfig_t](#)
- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelConfig_t](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelGetHandle](#)
- [CyU3PDmaMultiChannelUpdateMode](#)
- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelCommitBuffer](#)
- [CyU3PDmaMultiChannelDiscardBuffer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)
- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelSetWrapUp](#)

- [CyU3PDmaMultiChannelSetSuspend](#)
- [CyU3PDmaMultiChannelResume](#)
- [CyU3PDmaMultiChannelAbort](#)
- [CyU3PDmaMultiChannelReset](#)
- [CyU3PDmaMultiChannelGetStatus](#)
- [CyU3PDmaMultiChannelCacheControl](#)

Members

Members	Description
uint32_t state;	The current state of the DMA channel
uint16_t type;	The type of the DMA channel
uint16_t size;	The buffer size associated with the channel
uint16_t count;	Number of buffers for the channel
uint16_t validSckCount;	Number of sockets in the multi-socket operation.
uint16_t firstProdIndex[CY_U3P_DMA_MAX_MULTI_SCK_COUNT];	Head for the normal descriptor chain list
uint16_t firstConsIndex[CY_U3P_DMA_MAX_MULTI_SCK_COUNT];	Head for the manual mode consumer descriptor chain list
uint16_t prodSckId[CY_U3P_DMA_MAX_MULTI_SCK_COUNT];	The producer (ingress) socket ID
uint16_t consSckId[CY_U3P_DMA_MAX_MULTI_SCK_COUNT];	The consumer (egress) socket ID
uint16_t overrideDscrIndex;	Internal data - Descriptor for override modes.
uint16_t currentProdIndex;	Internal data - Producer descriptor for the latest buffer produced.
uint16_t currentConsIndex;	Internal data - Consumer descriptor for the latest buffer produced.
uint16_t commitProdIndex;	Internal data - Producer descriptor for the buffer to be consumed.
uint16_t commitConsIndex;	Internal data - Consumer descriptor for the buffer to be consumed.
uint16_t activeProdIndex[CY_U3P_DMA_MAX_MULTI_SCK_COUNT];	Internal data - Active producer descriptor.
uint16_t activeConsIndex[CY_U3P_DMA_MAX_MULTI_SCK_COUNT];	Internal data - Active consumer descriptor.
uint16_t prodHeader;	The producer socket header offset
uint16_t prodFooter;	The producer socket footer offset
uint16_t consHeader;	The consumer socket header offset
uint16_t prodAvailCount;	Minimum available buffers before producer is active.
uint16_t dmaMode;	Mode of DMA operation
uint16_t prodSusp;	Internal data - producer suspend option.
uint16_t consSusp;	Internal data - consumer suspend option.
uint16_t bufferCount[CY_U3P_DMA_MAX_MULTI_SCK_COUNT];	Internal data - number of active buffers available for consumer.
uint16_t discardCount[CY_U3P_DMA_MAX_MULTI_SCK_COUNT];	Internal data - number of buffers to be discarded.
uint32_t notification;	Registered notifications.

uint32_t xferSize;	Internal data - current xfer size
CyBool_t isDmaHandleDCache;	Whether to do internal DMA cache handling.
CyU3PMutex lock;	Internal data - lock for the channel
CyU3PEvent flags;	Internal data - event flags for the channel
CyU3PDmaMultiCallback_t cb;	Callback function which gets invoked on DMA events

File

cyu3dma.h

5.5.55 CyU3PDmaMultiChannelConfig_t

Multi socket DMA channel parameter structure.

This is the channel configuration structure which needs to be provided during multi socket channel initialization.

In case of many to one operations, 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] and 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 case of one to many operations, there shall be only one producer socket and 'validSckCount' number of consumer sockets. The consumer sockets needs to be updated in the required order of operation. The first buffer shall be send to the consSckId[0] and second from consSckId[1] and so on.

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

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.

If the data cache is enabled, the buffers should be 32 byte aligned and a multiple of 32 byte. This is because the cache line is 32 byte long. The buffers allocated by [CyU3PDmaBufferAlloc](#) function takes care of this. The size field need not be a multiple of 32 as the DMA API allocates channel memory from using the [CyU3PDmaBufferAlloc](#) call.

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 CommitBuffer 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 CommitBuffer 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.

More complex modes can be devised by enabling more than one offset.

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.

C++

```
struct CyU3PDmaMultiChannelConfig_t {  
    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;  
};
```

Group

DMA Data Types

See Also

- [CyU3PDmaMultiChannelCreate](#)

Members

Members	Description
uint16_t size;	The buffer size associated with the channel.
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.
uint16_t validSckCount;	Number of sockets in the multi-socket operation.
CyU3PDmaSocketId_t prodSckId[CY_U3P_DMA_MAX_MULTI_SCK_COUNT];	The producer (ingress) socket ID.
CyU3PDmaSocketId_t consSckId[CY_U3P_DMA_MAX_MULTI_SCK_COUNT];	The consumer (egress) socket ID.
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.
uint16_t prodHeader;	The producer socket header offset.
uint16_t prodFooter;	The producer socket footer offset.

uint16_t consHeader;	The consumer socket header offset.
CyU3PDmaMode_t dmaMode;	Mode of DMA operation
uint32_t notification;	Registered notifications. This is a bit map based on CyU3PDmaCbType_t . This defines the events for which the callback is triggered.
CyU3PDmaMultiCallback_t cb;	Callback function which gets invoked on multi socket DMA events.

File

cyu3dma.h

5.5.56 CyU3PDmaBuffer_t

DMA channel buffer data structure.

The data structure is used to describe the status of a buffer. It holds the address, size, valid count, and the status information. It is used in callbacks and DMA APIs to identify the data to be sent / received using DMA.

C++

```

struct CyU3PDmaBuffer_t {
    uint8_t * buffer;
    uint16_t count;
    uint16_t size;
    uint16_t status;
};

```

Group[DMA Data Types](#)**See Also**

- [CyU3PDmaCBInput_t](#)
- [CyU3PDmaChannelGetBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)
- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)

Members

Members	Description
uint8_t * buffer;	Pointer to the buffer
uint16_t count;	Byte count of valid data in buffer

uint16_t size;	Buffer size
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.

File

cyu3dma.h

5.5.57 CyU3PDmaDescriptor_t

Descriptor data structure.

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

The following members are composed of the corresponding fields. The fields are defined in sock_regs.h. Refer to sock_regs.h for more details about these fields.

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)

C++

```
struct CyU3PDmaDescriptor_t {
    uint8_t * buffer;
    uint32_t sync;
    uint32_t chain;
    uint32_t size;
};
```

Group

[DMA Data Types](#)

See Also

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

Members

Members	Description
uint8_t * buffer;	Pointer to buffer used.
uint32_t sync;	Consumer, Producer binding.

uint32_t chain;	Next descriptor links.
uint32_t size;	Current and maximum sizes of buffer.

File

cyu3descriptor.h

5.5.58 CyU3PDmaSocketConfig_t

DMA socket configuration structure.

DMA socket structure represents all DMA ingress and egress transactions. This structure should not be modified by outside code.

The following members are composed of the corresponding fields. The fields are defined in sock_regs.h. Refer to sock_regs.h for more details about these fields.

dscrChain: (CY_U3P_DSCR_LOW_MASK | CY_U3P_DSCR_COUNT_MASK | CY_U3P_DSCR_NUMBER_MASK)

xferSize: (CY_U3P_TRANS_SIZE_MASK)

xferCount: (CY_U3P_TRANS_COUNT_MASK)

status: (CY_U3P_GO_ENABLE | CY_U3P_GO_SUSPEND | CY_U3P_UNIT | CY_U3P_WRAPUP | CY_U3P_SUSP_EOP | CY_U3P_SUSP_TRANS | CY_U3P_SUSP_LAST | CY_U3P_SUSP_PARTIAL | CY_U3P_EN_CONS_EVENTS | CY_U3P_EN_PROD_EVENTS | CY_U3P_TRUNCATE | CY_U3P_ENABLED | CY_U3P_SUSPENDED | CY_U3P_ZLP_RCVD | CY_U3P_STATE_MASK | CY_U3P_AVL_ENABLE | CY_U3P_AVL_MIN_MASK | CY_U3P_AVL_COUNT_MASK)

intr: (CY_U3P_LAST_BUF | CY_U3P_PARTIAL_BUF | CY_U3P_TRANS_DONE | CY_U3P_ERROR | CY_U3P_SUSPEND | CY_U3P_STALL | CY_U3P_DSCR_NOT_AVL | CY_U3P_DSCR_IS_LOW | CY_U3P_CONSUME_EVENT | CY_U3P_PRODUCE_EVENT)

intrMask: (CY_U3P_LAST_BUF | CY_U3P_PARTIAL_BUF | CY_U3P_TRANS_DONE | CY_U3P_ERROR | CY_U3P_SUSPEND | CY_U3P_STALL | CY_U3P_DSCR_NOT_AVL | CY_U3P_DSCR_IS_LOW | CY_U3P_CONSUME_EVENT | CY_U3P_PRODUCE_EVENT)

C++

```
struct CyU3PDmaSocketConfig_t {
    uint32_t dscrChain;
    uint32_t xferSize;
    uint32_t xferCount;
    uint32_t status;
    uint32_t intr;
    uint32_t intrMask;
};
```

Group

[DMA Data Types](#)

See Also

- [CyU3PDmaSocket_t](#)

- [CyU3PDmaSocketSetConfig](#)
- [CyU3PDmaSocketGetConfig](#)

Members

Members	Description
uint32_t dscrChain;	The descriptor chain associateed with the socket
uint32_t xferSize;	Transfer size for the socket.
uint32_t xferCount;	Transfer status for the socket.
uint32_t status;	Socket status register.
uint32_t intr;	Interrupt status.
uint32_t intrMask;	Interrupt mask.

File

cyu3socket.h

5.5.59 CyU3PDmaSocket_t

DMA socket register structure.

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.

C++

```
struct CyU3PDmaSocket_t {
    uint32_t dscrChain;
    uint32_t xferSize;
    uint32_t xferCount;
    uint32_t status;
    uint32_t intr;
    uint32_t intrMask;
    uint32_t unused2[2];
    CyU3PDmaDescriptor\_t activeDscr;
    uint32_t unused19[19];
    uint32_t sckEvent;
};
```

Group

[DMA Data Types](#)

See Also

- [CyU3PDmaSocketConfig_t](#)

Members

Members	Description
uvint32_t dscrChain;	The descriptor chain associated with the socket
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 (bit no. 29) field in the status value.
uvint32_t xferCount;	The completed transfer count for this socket. The count is also specified in bytes or in terms of number of buffers depending on the UNIT field in the status value.
uvint32_t status;	Socket configuration and status register.
uvint32_t intr;	Interrupt status register.
uvint32_t intrMask;	Interrupt mask register.
uvint32_t unused2[2];	Reserved register space.
CyU3PDmaDescriptor_t activeDscr;	Active descriptor information. See cyu3descriptor.h for definition.
uvint32_t unused19[19];	Reserved register space.
uvint32_t sckEvent;	Generate event register.

File

cyu3socket.h

5.6 DMA Functions

This section documents the functions that are provided as part of the DMA manager module in the FX3 firmware library.

Group

[DMA Management](#)

Topics





Topic	Description
Buffer Functions	This section documents the functions that get or free buffers for DMA operations.
Descriptor Functions	<p>This section documents the utility functions that work with the DMA descriptors. These functions are supposed to be internal functions for the libraries' use and are not expected to be called directly by the user application.</p> <p>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 incorrect/invalid parameters can result in unpredictable behavior. In particular, the buffer address in the DMA descriptor needs to be in system memory area of 512KB. Any other buffer address can cause the entire DMA engine... more</p>

Socket Functions	<p>This section documents the functions that operate on the DMA sockets within the FX3 device. These functions are only for use of the DMA manager itself, and are not expected to be called directly by user applications.</p> <p>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 incorrect/invalid parameters can result in unpredictable behavior.</p> <p>In particular, the Socket ID passed to these functions has to be validated by either of CyU3PDmaSocketIsValid(), CyU3PDmaSocketIsValidProducer() OR CyU3PDmaSocketIsValidConsumer() functions.</p>
Channel Functions	This section documents the functions that operate on DMA channels.
Multi-Channel Functions	<p>Multi-channel are special versions of DMA channels that involve multiple producers or multiple consumers for a data flow. Since these channels will only be used rarely, the operations on such channel are kept separated in a different section which can be removed from the firmware binary when not used.</p> <p>This section documents the functions that operate on multi-channels. These operations are similar to those supported on regular DMA channels.</p>

5.6.1 Buffer Functions

This section documents the functions that get or free buffers for DMA operations.

Functions

Function	Description
 CyU3PDmaBufferInit	Initialize the buffer allocator.
 CyU3PDmaBufferDeInit	De-initialize the buffer allocator.
 CyU3PDmaBufferAlloc	Allocate the required amount of buffer memory.
 CyU3PDmaBufferFree	Free the previously allocated buffer area.

Group

[DMA Functions](#)

Legend

	Method
---	--------

5.6.1.1 CyU3PDmaBufferInit

Initialize the buffer allocator.

This function is the allocates memory required by the DMA engine. There are restrictions for the memory to be used with DMA. The buffer should be 32 byte aligned and should be a multiple of 32 bytes. This function needs to be implemented as part of the RTOS porting to the FX3 device. The function shall be invoked on DMA module initialization. This function must not be invoked by user application.

Returns

- None

C++

```
void CyU3PDmaBufferInit(  
    void  
);
```

Group

[Buffer Functions](#)

See Also

- [CyU3PDmaBufferDeInit](#)
- [CyU3PDmaBufferAlloc](#)
- [CyU3PDmaBufferFree](#)

File

cyu3os.h

5.6.1.2 CyU3PDmaBufferDeInit

De-initialize the buffer allocator.

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 invoked by user application.

Returns

- None

C++

```
void CyU3PDmaBufferDeInit(  
    void  
);
```

Group

[Buffer Functions](#)

See Also

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

File

cyu3os.h

5.6.1.3 CyU3PDmaBufferAlloc

Allocate the required amount of buffer memory.

This function is used to allocate memory required for DMA operations. It is called from the DMA module whenever a channel is created. It can also be used by user application for allocating memory which can be used for DMA operations. This function needs to be implemented as part of RTOS porting and is provided in source format. It can be modified, to suit the user application. The buffers allocated must be 32 byte aligned and should be a multiple of 32 bytes.

Parameters

Parameters	Description
uint16_t size	The size of the buffer required.

Returns

- Pointer to the allocated buffer. Return NULL in case of error.

C++

```
void * CyU3PDmaBufferAlloc(  
    uint16_t size  
);
```

Group

[Buffer Functions](#)

See Also

- [CyU3PDmaBufferInit](#)
- [CyU3PDmaBufferDeInit](#)
- [CyU3PDmaBufferFree](#)

File

cyu3os.h

5.6.1.4 CyU3PDmaBufferFree

Free the previously allocated buffer area.

This function frees the memory allocated. Care should be taken so that buffers allocated by the DMA module is not freed explicitly by the user application. This can lead corruption of DMA channels. User application can invoke this function for those buffers explicitly allocated. The function is also invoked when ever a DMA channel destroy call is made. The function is provided in source format and can be modified if required by the user application.

Parameters

Parameters	Description
<code>void * buffer</code>	Address of buffer to be freed.

Returns

- None

C++

```
void CyU3PDmaBufferFree(  
    void * buffer  
);
```

Group

[Buffer Functions](#)

See Also

- [CyU3PDmaBufferInit](#)
- [CyU3PDmaBufferDeInit](#)
- [CyU3PDmaBufferAlloc](#)

File










cyu3os.h

5.6.2 Descriptor Functions

This section documents the utility functions that work with the DMA descriptors. These functions are supposed to be internal functions for the libraries' 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 incorrect/invalid parameters can result in unpredictable behavior. In particular, the buffer address in the DMA descriptor needs to be in system memory area of 512KB. Any other buffer address can cause the entire DMA engine to freeze, requiring a reset.

Functions

Function	Description
 CyU3PDmaDscrGetFreeCount	Get the number of free descriptors available.
 CyU3PDmaDscrGet	Get a descriptor number from the free list.
 CyU3PDmaDscrPut	Add a descriptor number to the free list.
 CyU3PDmaDscrSetConfig	Set the descriptor configuration.
 CyU3PDmaDscrGetConfig	Get the descriptor configuration.
 CyU3PDmaDscrListCreate	Create (init) the free descriptor list.
 CyU3PDmaDscrListDestroy	Destroy (de-init) the free descriptor list.
 CyU3PDmaDscrChainCreate	Create a circular chain of descriptors.
 CyU3PDmaDscrChainDestroy	Frees the previously created chain of descriptors.

Group

[DMA Functions](#)

Legend

	Method
---	--------

5.6.2.1 CyU3PDmaDscrGetFreeCount

Get the number of free descriptors available.

This function returns the number of descriptors available for use. It is used by the DMA channel APIs. These can be used to customize descriptors and do advanced DMA operations.

Returns

- The number of free descriptors available.

C++

```
uint16_t CyU3PDmaDscrGetFreeCount(  
    void  
);
```

Group

[Descriptor Functions](#)

See Also

- [CyU3PDmaDscrGet](#)
- [CyU3PDmaDscrPut](#)

File

cyu3descriptor.h

5.6.2.2 CyU3PDmaDscrGet

Get a descriptor number from the free list.

This function searches the free list for the first available descriptor, and returns the index. This function is used by the DMA channel APIs. These can be used to customize descriptors and do advanced DMA operations.

Parameters

Parameters	Description
<code>uint16_t * index_p</code>	Output parameter which is filled with the free descriptor index.

Returns

- `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

C++

```
CyU3PReturnStatus\_t CyU3PDmaDscrGet(  
    uint16_t * index_p  
);
```

Group

[Descriptor Functions](#)

See Also

- [CyU3PDmaDscrPut](#)
- [CyU3PDmaDscrGetFreeCount](#)

File

`cyu3descriptor.h`

5.6.2.3 CyU3PDmaDscrPut

Add a descriptor number to the free list.

This function marks a descriptor as available in the free list. This function is used by the DMA channel APIs. These can be used to customize descriptors and do advanced DMA operations.

Parameters

Parameters	Description
uint16_t index	Descriptor index to be marked free.

Returns

- CY_U3P_SUCCESS - if the function call is successful
- CY_U3P_ERROR_BAD_ARGUMENT - if an invalid index is passed

C++

```
CyU3PReturnStatus\_t CyU3PDmaDscrPut(  
    uint16_t index  
);
```

Group

[Descriptor Functions](#)

See Also

- [CyU3PDmaDscrGet](#)
- [CyU3PDmaDscrGetFreeCount](#)

File

cyu3descriptor.h

5.6.2.4 CyU3PDmaDscrSetConfig

Set the descriptor configuration.

Since the descriptor cannot be set directly, load the required configuration into the parameter and configure the required descriptor. This function is used by the DMA channel APIs. These can be used to customize descriptors and do advanced DMA operations.

Parameters

Parameters	Description
uint16_t index	Index of the descriptor to be updated.
CyU3PDmaDescriptor_t * dscr_p	Pointer to descriptor structure containing the desired configuration.

Returns

- CY_U3P_SUCCESS - if the function call is successful
- CY_U3P_ERROR_BAD_ARGUMENT - if an invalid index or a NULL pointer is passed

C++

```
CyU3PReturnStatus_t CyU3PDmaDscrSetConfig(  
    uint16_t index,  
    CyU3PDmaDescriptor_t * dscr_p  
);
```

Group

[Descriptor Functions](#)

See Also

- [CyU3PDmaDscrGetConfig](#)

File

cyu3descriptor.h

5.6.2.5 CyU3PDmaDscrGetConfig

Get the descriptor configuration.

Since the descriptor cannot be read directly, read the configuration into a descriptor structure. This function is used by the DMA channel APIs. These can be used to customize descriptors and do advanced DMA operations.

Parameters

Parameters	Description
uint16_t index	Index of the descriptor to read.
CyU3PDmaDescriptor_t * dscr_p	Output parameter that will be filled with the descriptor values.

Returns

- CY_U3P_SUCCESS - if the function call is successful
- CY_U3P_ERROR_BAD_ARGUMENT - if an invalid index or a NULL pointer is passed

C++

```
CyU3PReturnStatus_t CyU3PDmaDscrGetConfig(  
    uint16_t index,  
    CyU3PDmaDescriptor_t * dscr_p  
);
```

Group

[Descriptor Functions](#)

See Also

- CyU3PDmaDscrGetConfig

File

cyu3descriptor.h

5.6.2.6 CyU3PDmaDscrListCreate

Create (init) the free descriptor list.

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.

Returns

- None

C++

```
void CyU3PDmaDscrListCreate(  
    void  
);
```

Group

[Descriptor Functions](#)

See Also

- [CyU3PDmaDscrListDestroy](#)

File

cyu3descriptor.h

5.6.2.7 CyU3PDmaDscrListDestroy

Destroy (de-init) the free descriptor list.

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 is not expected to be called explicitly.

Returns

- None

C++

```
void CyU3PDmaDscrListDestroy(  
    void  
);
```

Group

[Descriptor Functions](#)

See Also

- [CyU3PDmaDscrListCreate](#)

File

cyu3descriptor.h

5.6.2.8 CyU3PDmaDscrChainCreate

Create a circular chain of descriptors.

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. Buffer is allocated if the bufferSize variable is non zero. This function is used by the DMA channel APIs. These can be used to customize descriptors and do advanced DMA operations.

Parameters

Parameters	Description
uint16_t * dscrIndex_p	Output parameter that specifies the index of the head descriptor in the chain.
uint16_t count	Number of descriptors required in the chain.
uint16_t bufferSize	Size of the buffers to be associated with each descriptor. If set to zero, no buffers will be allocated.
uint32_t dscrSync	The sync field to be set in all descriptors. Specifies the producer and consumer socket information.

Returns

- CY_U3P_SUCCESS - if the function call is successful
- CY_U3P_ERROR_MEMORY_ERROR - if the number of descriptors required in this chain are not available OR the buffers could not be allocated
- CY_U3P_ERROR_BAD_ARGUMENT - if number of descriptors required is zero OR the index is NULL

C++

```
CyU3PReturnStatus\_t CyU3PDmaDscrChainCreate(  
    uint16_t * dscrIndex_p,  
    uint16_t count,  
    uint16_t bufferSize,  
    uint32_t dscrSync  
);
```

Group

[Descriptor Functions](#)

See Also

- [CyU3PDmaDscrChainDestroy](#)

File

cyu3descriptor.h

5.6.2.9 CyU3PDmaDscrChainDestroy

Frees the previously created chain of descriptors.

The function frees the chain of descriptors. This function must be invoked only after suspending or disabling the sockets. The buffers pointed to by the descriptors can be optionally freed. This function is used by the DMA channel APIs. These can be used to customize descriptors and do advanced DMA operations.

Parameters

Parameters	Description
uint16_t dscrIndex	Index of the head descriptor in the chain.
uint16_t count	Number of descriptors in the chain.
CyBool_t isProdChain	Specifies whether to traverse the producer chain or the consumer chain to get the next descriptor.
CyBool_t freeBuffer	Whether the DMA buffers associated with the descriptors should be freed.

Returns

- None

C++

```
void CyU3PDmaDscrChainDestroy(  
    uint16_t dscrIndex,  
    uint16_t count,  
    CyBool_t isProdChain,  
    CyBool_t freeBuffer  
);
```

Group

[Descriptor Functions](#)

See Also

- [CyU3PDmaDscrChainDestroy](#)

File

cyu3descriptor.h

5.6.3 Socket Functions













This section documents the functions that operate on the DMA sockets within the FX3 device. These functions are only for use of the DMA manager itself, and are not expected to be called directly by user applications.

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 incorrect/invalid parameters can result in unpredictable behavior.

In particular, the Socket ID passed to these functions has to be validated by either of [CyU3PDmaSocketIsValid\(\)](#),

[CyU3PDmaSocketIsValidProducer\(\)](#) OR [CyU3PDmaSocketIsValidConsumer\(\)](#) functions.

Functions

Function	Description
 CyU3PDmaSocketIsValid	Validates the socket ID.
 CyU3PDmaSocketIsValidProducer	Validates the socket ID as a producer.
 CyU3PDmaSocketIsValidConsumer	Validates the socket ID as a producer.
 CyU3PDmaSocketSetConfig	Sets the socket configuration.
 CyU3PDmaSocketGetConfig	Gets the socket configuration.
 CyU3PDmaSocketSendEvent	Send an event to the socket.
 CyU3PDmaSocketSetWrapUp	Sets the wrapup bit for the socket.
 CyU3PDmaUpdateSocketSuspendOption	Updates the options for the socket suspend.
 CyU3PDmaUpdateSocketResume	Resumes a socket from the suspended state.
 CyU3PDmaSocketEnable	Enables the selected socket.
 CyU3PDmaSocketDisable	Disables the selected socket.
 CyU3PDmaSocketRegisterCallback	Register a callback handler for custom socket interrupts.

Group

[DMA Functions](#)

Legend

	Method
---	--------

5.6.3.1 CyU3PDmaSocketIsValid

Validates the socket ID.

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 block itself is not started.

Parameters

Parameters	Description
uint16_t sckId	Socket id to be validated.

Returns

- CyTrue if the socket is valid, and CyFalse otherwise.

C++

```
CyBool_t CyU3PDmaSocketIsValid(  
    uint16_t sckId  
);
```

Group

[Socket Functions](#)

See Also

- [CyU3PDmaSocketIsValidProducer](#)
- [CyU3PDmaSocketIsValidConsumer](#)

File

cyu3socket.h

5.6.3.2 CyU3PDmaSocketIsValidProducer

Validates the socket ID as a producer.

The function validates if the given socket id exists as a producer on the device. CPU sockets are virtual sockets and so the function considers them as invalid. The function considers a socket invalid if the block itself is not started.

Parameters

Parameters	Description
uint16_t sckId	Socket id to be validated.

Returns

- CyTrue if the socket is a valid producer, and CyFalse otherwise.

C++

```
CyBool_t CyU3PDmaSocketIsValidProducer(  
    uint16_t sckId  
);
```

Group

[Socket Functions](#)

See Also

- [CyU3PDmaSocketIsValid](#)
- [CyU3PDmaSocketIsValidConsumer](#)

File

cyu3socket.h

5.6.3.3 CyU3PDmaSocketIsValidConsumer

Validates the socket ID as a producer.

The function validates if the given socket id exists as a consumer on the device. CPU sockets are virtual sockets and so the function considers them as invalid. The function considers a socket invalid if the block itself is not started.

Parameters

Parameters	Description
uint16_t sckId	Socket id to be validated.

Returns

- CyTrue if the socket is a valid consumer, and CyFalse otherwise.

C++

```
CyBool_t CyU3PDmaSocketIsValidConsumer(  
    uint16_t sckId  
);
```

Group

[Socket Functions](#)

See Also

- [CyU3PDmaSocketIsValid](#)
- [CyU3PDmaSocketIsValidProducer](#)

File

cyu3socket.h

5.6.3.4 CyU3PDmaSocketSetConfig

Sets the socket configuration.

Sets the socket configuration. Socket structures must not be set directly and must be done only via APIs.

Parameters

Parameters	Description
uint16_t sckId	Socket id whose configuration is to be updated.
CyU3PDmaSocketConfig_t * sck_p	Pointer to structure containing socket configuration.

Returns

- CY_U3P_SUCCESS - if the function call is successful
- CY_U3P_ERROR_BAD_ARGUMENT - if the sck_p pointer is NULL

C++

```
CyU3PReturnStatus\_t CyU3PDmaSocketSetConfig(  
    uint16_t sckId,  
    CyU3PDmaSocketConfig\_t * sck_p  
);
```

Group

[Socket Functions](#)

See Also

- [CyU3PDmaSocketGetConfig](#)

File

cyu3socket.h

5.6.3.5 CyU3PDmaSocketGetConfig

Gets the socket configuration.

Creates a copy of the socket structure and returns the current socket configuration.

Parameters

Parameters	Description
uint16_t sckId	Socket id whose configuration is to be retrieved.
CyU3PDmaSocketConfig_t * sck_p	Output parameter to be filled with the socket configuration.

Returns

- CY_U3P_SUCCESS - if the function call is successful
- CY_U3P_ERROR_BAD_ARGUMENT - if the sck_p pointer is NULL

C++

```
CyU3PReturnStatus\_t CyU3PDmaSocketGetConfig(  
    uint16_t sckId,  
    CyU3PDmaSocketConfig\_t * sck_p  
);
```

Group

[Socket Functions](#)

See Also

- [CyU3PDmaSocketSetConfig](#)

File

cyu3socket.h

5.6.3.6 CyU3PDmaSocketSendEvent

Send an event to the socket.

Send a produce / consume event to the selected socket.

Parameters

Parameters	Description
uint16_t sckId	Id of socket that should receive the event.
uint16_t dscrIndex	The descriptor index to associate with the event.
CyBool_t isOccupied	Status of the buffer associated with the event.

Returns

- None

C++

```
void CyU3PDmaSocketSendEvent(  
    uint16_t sckId,  
    uint16_t dscrIndex,  
    CyBool_t isOccupied  
);
```

Group

[Socket Functions](#)

See Also

- [CyU3PDmaSocketResume](#)

File

cyu3socket.h

5.6.3.7 CyU3PDmaSocketSetWrapUp

Sets the wrapup bit for the socket.

The function wraps up the active buffer of the socket. The wrapup is set only if the socket is active. The function does not wait for the socket to wrap up. Data may be lost in the process.

Parameters

Parameters	Description
uint16_t sckId	Id of socket to be wrapped up.

Returns

- None

C++

```
void CyU3PDmaSocketSetWrapUp(  
    uint16_t sckId  
);
```

Group

[Socket Functions](#)

See Also

- [CyU3PDmaSocketEnable](#)
- [CyU3PDmaSocketDisable](#)

File

cyu3socket.h

5.6.3.8 CyU3PDmaUpdateSocketSuspendOption

Updates the options for the socket suspend.

This API does not suspend the socket. It only updates the suspend conditions.

Parameters

Parameters	Description
uint16_t sckId	Id of socket to set the option.
uint16_t suspendOption	Suspend option.

Returns

- None

C++

```
void CyU3PDmaUpdateSocketSuspendOption(  
    uint16_t sckId,  
    uint16_t suspendOption  
);
```

Group

[Socket Functions](#)

See Also

- [CyU3PDmaUpdateSocketResume](#)

File

cyu3socket.h

5.6.3.9 CyU3PDmaUpdateSocketResume

Resumes a socket from the suspended state.

This function clears all suspend interrupts and resumes the sockets. It also sets the interrupt masks as per the suspend option specified.

Parameters

Parameters	Description
uint16_t sckId	Id of socket to set the option.
uint16_t suspendOption	Suspend option.

Returns

- None

C++

```
void CyU3PDmaUpdateSocketResume(  
    uint16_t sckId,  
    uint16_t suspendOption  
);
```

Group

[Socket Functions](#)

See Also

- [CyU3PDmaUpdateSocketSuspendOption](#)

File

cyu3socket.h

5.6.3.10 CyU3PDmaSocketEnable

Enables the selected socket.

The socket shall start functioning immediately and the descriptors are expected to be setup previously.

Parameters

Parameters	Description
uint16_t sckId	Id of socket to be enabled.

Returns

- None

C++

```
void CyU3PDmaSocketEnable(  
    uint16_t sckId  
);
```

Group

[Socket Functions](#)

See Also

- [CyU3PDmaSocketDisable](#)

File

cyu3socket.h

5.6.3.11 CyU3PDmaSocketDisable

Disables the selected socket.

The function disables the socket and returns only after the socket has been disabled. Data may be lost in the process.

Parameters

Parameters	Description
uint16_t sckId	Id of socket to be disabled.

Returns

- None

C++

```
void CyU3PDmaSocketDisable(  
    uint16_t sckId  
);
```

Group

[Socket Functions](#)

See Also

- [CyU3PDmaSocketEnable](#)

File

cyu3socket.h

5.6.3.12 CyU3PDmaSocketRegisterCallback

Register a callback handler for custom socket interrupts.

The handler needs to be registered only if there are custom DMA operations built on socket APIs.

Parameters

Parameters	Description
<code>CyU3PDmaSocketCallback_t cb</code>	Callback function pointer. Can be zero to unregister previous callback.

Returns

- None

C++

```
void CyU3PDmaSocketRegisterCallback(  
    CyU3PDmaSocketCallback\_t cb  
);
```

Group

[Socket Functions](#)

See Also

- [CyU3PDmaSocketCallback](#).

File













`cyu3socket.h`

5.6.4 Channel Functions


This section documents the functions that operate on DMA channels.

Functions

Function	Description
CyU3PDmaChannelCreate	Create a DMA channel.
CyU3PDmaChannelDestroy	Destroy a DMA channel.
CyU3PDmaChannelUpdateMode	Update the DMA mode for the channel.
CyU3PDmaChannelGetStatus	This functions returns the current channel status.
CyU3PDmaChannelGetHandle	Returns handle associated with the socket ID.
CyU3PDmaChannelCacheControl	This function enables / disables DMA API internal d-cache handling.
CyU3PDmaChannelSetXfer	Set a transfer on the DMA channel.

 CyU3PDmaChannelGetBuffer	Get the current buffer pointer.
 CyU3PDmaChannelCommitBuffer	Commit the buffer to be sent to the consumer.
 CyU3PDmaChannelDiscardBuffer	Discard buffer in the current DMA descriptor.
 CyU3PDmaChannelSetupSendBuffer	Send an external buffer to the consumer.
 CyU3PDmaChannelSetupRecvBuffer	Receive data to an external buffer.
 CyU3PDmaChannelWaitForRecvBuffer	Wait for the data to be filled.
 CyU3PDmaChannelWaitForCompletion	Wait for the current DMA transaction to complete.
 CyU3PDmaChannelSetWrapUp	Wraps up the current active buffer for the channel from the producer side.
 CyU3PDmaChannelSetSuspend	Set the suspend options for the sockets.
 CyU3PDmaChannelResume	Resume a suspended DMA channel.
 CyU3PDmaChannelReset	Aborts and resets a DMA channel.
 CyU3PDmaChannelAbort	Aborts a DMA channel.

Group[DMA Functions](#)**Legend**

	Method
---	--------

5.6.4.1 CyU3PDmaChannelCreate

Create a DMA channel.

A DMA channel requires a producer socket, a consumer socket, a set of buffers and a callback function. This function must not be called when in DMA callback.

Parameters

Parameters	Description
<code>CyU3PDmaChannel * handle</code>	Pointer to channel structure that should be initialized.
<code>CyU3PDmaType_t type</code>	Type of DMA channel desired.
<code>CyU3PDmaChannelConfig_t * config</code>	Channel configuration parameters.

Returns

- `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

C++

```
CyU3PReturnStatus_t CyU3PDmaChannelCreate(  
    CyU3PDmaChannel * handle,  
    CyU3PDmaType_t type,  
    CyU3PDmaChannelConfig_t * config  
);
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaType_t](#)
- [CyU3PDmaChannel](#)
- [CyU3PDmaChannelConfig_t](#)
- [CyU3PDmaChannelDestroy](#)
- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelGetHandle](#)
- [CyU3PDmaChannelSetXfer](#)
- [CyU3PDmaChannelGetBuffer](#)
- [CyU3PDmaChannelCommitBuffer](#)
- [CyU3PDmaChannelDiscardBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)

- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelSetSuspend](#)
- [CyU3PDmaChannelResume](#)
- [CyU3PDmaChannelAbort](#)
- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

File

cyu3dma.h

5.6.4.2 CyU3PDmaChannelDestroy

Destroy a DMA channel.

This should be called once the DMA channel is no longer required. This function must not be called when in DMA callback.

Parameters

Parameters	Description
<code>CyU3PDmaChannel * handle</code>	Pointer to DMA channel structure to be de-initialized.

Returns

- `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

C++

```
CyU3PReturnStatus\_t CyU3PDmaChannelDestroy(  
    CyU3PDmaChannel * handle  
);
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaChannel](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelGetHandle](#)
- [CyU3PDmaChannelSetXfer](#)
- [CyU3PDmaChannelGetBuffer](#)
- [CyU3PDmaChannelCommitBuffer](#)
- [CyU3PDmaChannelDiscardBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)
- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelSetSuspend](#)
- [CyU3PDmaChannelResume](#)

- [CyU3PDmaChannelAbort](#)
- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

File

cyu3dma.h

5.6.4.3 CyU3PDmaChannelUpdateMode

Update the DMA mode for the channel.

The DMA mode is specified during creation of the channel and if this needs to be changed, then the API needs to be invoked. This API call can be made only when the DMA channel is in configured state (when there are no transfers setup).

Parameters

Parameters	Description
<code>CyU3PDmaChannel * handle</code>	Handle to DMA channel to be modified.
<code>CyU3PDmaMode_t dmaMode</code>	Desired DMA operating mode. Can be byte mode or buffer mode.

Returns

- `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_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

C++

```
CyU3PReturnStatus\_t CyU3PDmaChannelUpdateMode(  
    CyU3PDmaChannel * handle,  
    CyU3PDmaMode\_t dmaMode  
);
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaMode_t](#)
- [CyU3PDmaChannel](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelDestroy](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelGetHandle](#)
- [CyU3PDmaChannelSetXfer](#)
- [CyU3PDmaChannelGetBuffer](#)
- [CyU3PDmaChannelCommitBuffer](#)
- [CyU3PDmaChannelDiscardBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)

- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelSetSuspend](#)
- [CyU3PDmaChannelResume](#)
- [CyU3PDmaChannelAbort](#)
- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

File

cyu3dma.h

5.6.4.4 CyU3PDmaChannelGetStatus

This functions returns the current channel status.

The function can be used to keep track of the data transfers and even for transfer progress. It also returns the current channel state. In case of override modes, this function always returns the byte count. The count field is updated only at buffer boundaries and so does not give real time information.

Parameters

Parameters	Description
CyU3PDmaChannel * handle	Handle to the DMA channel to query.
CyU3PDmaState_t * state	Output parameter that will be filled with state of the channel.
uint32_t * prodXferCount	Output parameter that will be filled with transfer count on the producer socket in DMA mode units.
uint32_t * consXferCount	Output parameter that will be filled with transfer count on the consumer socket in DMA mode units.

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PDmaChannelGetStatus(  
    CyU3PDmaChannel * handle,  
    CyU3PDmaState\_t * state,  
    uint32_t * prodXferCount,  
    uint32_t * consXferCount  
);
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaState_t](#)
- [CyU3PDmaChannel](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelDestroy](#)
- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetHandle](#)
- [CyU3PDmaChannelSetXfer](#)
- [CyU3PDmaChannelGetBuffer](#)
- [CyU3PDmaChannelCommitBuffer](#)
- [CyU3PDmaChannelDiscardBuffer](#)

- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)
- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelSetSuspend](#)
- [CyU3PDmaChannelResume](#)
- [CyU3PDmaChannelAbort](#)
- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

File

`cyu3dma.h`

5.6.4.5 CyU3PDmaChannelGetHandle

Returns handle associated with the socket ID.

This function is used to identify if there is any DMA channel associated with a socket. The function should be used only with single DMA channels.

Parameters

Parameters	Description
CyU3PDmaSocketId_t sckId	The socket ID for the socket whose channel handle is required.

Returns

- Handle of the channel associated with the specified socket. A NULL return indicates that an invalid socket ID was passed.

C++

```
CyU3PDmaChannel * CyU3PDmaChannelGetHandle(  
    CyU3PDmaSocketId\_t sckId  
);
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaChannel](#)
- [CyU3PDmaSocketId_t](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelDestroy](#)
- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelSetXfer](#)
- [CyU3PDmaChannelGetBuffer](#)
- [CyU3PDmaChannelCommitBuffer](#)
- [CyU3PDmaChannelDiscardBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)
- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelSetSuspend](#)
- [CyU3PDmaChannelResume](#)
- [CyU3PDmaChannelAbort](#)

- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

File

cyu3dma.h

5.6.4.6 CyU3PDmaChannelCacheControl

This function enables / disables DMA API internal d-cache handling.

This function provides control over DMA API internal d-cache handling feature per individual channel. The global behaviour is selected based on [CyU3PDeviceCacheControl](#) API. This API allows to override this option for a particular channel. It should be noted that the [CyU3PDeviceCacheControl](#) should be invoked before even initializing the RTOS whereas this API can be done after a channel is created. However the call should be made when the channel is idle (CONFIGURED state). This API makes sense only when the D-cache is enabled.

Parameters

Parameters	Description
CyU3PDmaChannel * handle	Handle to the DMA channel.
CyBool_t isDmaHandleDCache	Whether to enable handling or not.

Returns

- 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)

C++

```
CyU3PReturnStatus\_t CyU3PDmaChannelCacheControl(  
    CyU3PDmaChannel * handle,  
    CyBool\_t isDmaHandleDCache  
);
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaChannel](#)
- [CyU3PDeviceCacheControl](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelDestroy](#)

File

cyu3dma.h

5.6.4.7 CyU3PDmaChannelSetXfer

Set a transfer on the DMA channel.

The function starts a transaction the selected DMA channel. It should be invoked only when the channel is in CY_U3P_DMA_CONFIGURED state.

Parameters

Parameters	Description
CyU3PDmaChannel * handle	Handle to the channel to be modified.
uint32_t count	The desired transaction size in units corresponding to the selected DMA mode. Channel will revert to idle state when the specified size of data has been transferred. Can be set to zero to request an infinite data transfer.

Returns

- 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 the channel count is zero
- CY_U3P_ERROR_ALREADY_STARTED - if the channel is active
- CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired

C++

```
CyU3PReturnStatus_t CyU3PDmaChannelSetXfer(  
    CyU3PDmaChannel * handle,  
    uint32_t count  
);
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaChannel](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelDestroy](#)
- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelGetHandle](#)
- [CyU3PDmaChannelGetBuffer](#)
- [CyU3PDmaChannelCommitBuffer](#)
- [CyU3PDmaChannelDiscardBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)

- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelSetSuspend](#)
- [CyU3PDmaChannelResume](#)
- [CyU3PDmaChannelAbort](#)
- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

File

cyu3dma.h

5.6.4.8 CyU3PDmaChannelGetBuffer

Get the current buffer pointer.

This function acts differently for different types of DMA channels.

For CY_U3P_DMA_TYPE_MANUAL and CY_U3P_DMA_TYPE_MANUAL_IN channels, the function returns the data buffer received from the producer socket. The wait_option specifies the timeout value. In case of the CY_U3P_DMA_TYPE_MANUAL_OUT type of DMA channel, the function returns the pointer to the buffer for the CPU to fill the data.

The buffer pointer returned, points to the data produced by the producer socket. So the pointer and the count does not include the offsets assigned.

For example, if there is a producer header offset of 12 bytes, the valid data from the producer header can be reached at `buffer_p->buffer[-12]`. Care should be taken so that no more than allocated data is accessed.

For a MANUAL_OUT channel with a consumer header offset, the buffer pointer returned points to the beginning of the actual buffer allocated as the CPU is the producer and there are no producer offsets. The count value shall be zero if there is no consumer header offset and shall be equal to the consumer header offset in case of a non-zero consumer offset. This is for ease of use and the user can know the consumer offset.

For AUTO channels this is a special API to be called only when the consumer socket is in the suspended state. In all other cases this will return CY_U3P_ERROR_INVALID_SEQUENCE. This API is used to look at the current consumer buffer in case of a SUSP_CONS_PARTIAL_BUF situation.

This function must not be called when in DMA callback with a non-zero wait option.

Parameters

Parameters	Description
CyU3PDmaChannel * handle	Handle to the DMA channel on which to wait.
CyU3PDmaBuffer_t * buffer_p	Output parameter that will be filled with data about the buffer that was obtained.
uint32_t waitOption	Duration to wait before returning a timeout status.

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PDmaChannelGetBuffer(  
    CyU3PDmaChannel * handle,  
    CyU3PDmaBuffer\_t * buffer_p,  
    uint32_t waitOption  
);
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaBuffer_t](#)
- [CyU3PDmaChannel](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelDestroy](#)
- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelGetHandle](#)
- [CyU3PDmaChannelSetXfer](#)
- [CyU3PDmaChannelCommitBuffer](#)
- [CyU3PDmaChannelDiscardBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)
- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelSetSuspend](#)
- [CyU3PDmaChannelResume](#)
- [CyU3PDmaChannelAbort](#)
- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

File

cyu3dma.h

5.6.4.9 CyU3PDmaChannelCommitBuffer

Commit the buffer to be sent to the consumer.

The function is generally for the manual DMA channels. The function sends a buffer in the current DMA descriptor to the consumer socket. This is not valid for CY_U3P_DMA_TYPE_MANUAL_IN channels. The count of data provided is exact count that needs to be sent out of the consumer socket.

This API is used for AUTO channels for special case handling. It should be called only when the consumer socket is in the suspended state. In all other cases this will return CY_U3P_ERROR_INVALID_SEQUENCE. This API is used to commit the current consumer buffer in case of a SUSP_CONS_PARTIAL_BUF situation.

Parameters

Parameters	Description
CyU3PDmaChannel * handle	Handle to the DMA channel to be modified.
uint16_t count	Size of data in the buffer being committed. The buffer address is implicit and is fetched from the active descriptor for the channel.
uint16_t bufStatus	Current status (occupied and end of transfer bits) of the buffer being committed.

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PDmaChannelCommitBuffer(  
    CyU3PDmaChannel * handle,  
    uint16_t count,  
    uint16_t bufStatus  
);
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaChannel](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelDestroy](#)

- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelGetHandle](#)
- [CyU3PDmaChannelSetXfer](#)
- [CyU3PDmaChannelGetBuffer](#)
- [CyU3PDmaChannelDiscardBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)
- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelSetSuspend](#)
- [CyU3PDmaChannelResume](#)
- [CyU3PDmaChannelAbort](#)
- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

File

cyu3dma.h

5.6.4.10 CyU3PDmaChannelDiscardBuffer

Discard buffer in the current DMA descriptor.

The function is generally for the CY_U3P_DMA_TYPE_MANUAL and CY_U3P_DMA_TYPE_MANUAL_IN types of DMA channels to discard the current buffer.

AUTO Channel special case: This API is also used for AUTO channels for special case handling. For AUTO channels, this API is used to discard the current consumer buffer only in case of a SUSP_CONS_PARTIAL_BUF situation. The discard buffer will function only when the consumer socket is in the suspended state. If the socket is not suspended, this API will return CY_U3P_ERROR_INVALID_SEQUENCE.

Parameters

Parameters	Description
CyU3PDmaChannel * handle	Handle to the DMA channel to be modified.

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PDmaChannelDiscardBuffer(  
    CyU3PDmaChannel * handle  
) ;
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaChannel](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelDestroy](#)
- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelGetHandle](#)
- [CyU3PDmaChannelSetXfer](#)
- [CyU3PDmaChannelGetBuffer](#)

- [CyU3PDmaChannelCommitBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)
- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelSetSuspend](#)
- [CyU3PDmaChannelResume](#)
- [CyU3PDmaChannelAbort](#)
- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

File

cyu3dma.h

5.6.4.11 CyU3PDmaChannelSetupSendBuffer

Send an external buffer to the consumer.

This function is an override on the normal behavior of the channel and can send an external buffer to the consumer. This function can be called only from the CY_U3P_DMA_CONFIGURED state.

The buffers used for DMA operations are expected to be allocated using [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. This is to satisfy the 32 byte cache lines.

Parameters

Parameters	Description
CyU3PDmaChannel * handle	Handle to the DMA channel to be modified.
CyU3PDmaBuffer_t * buffer_p	Pointer to structure containing address, size and status of the DMA buffer to be sent out.

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PDmaChannelSetupSendBuffer(  
    CyU3PDmaChannel * handle,  
    CyU3PDmaBuffer_t * buffer_p  
) ;
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaBuffer_t](#)
- [CyU3PDmaChannel](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelDestroy](#)
- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelGetHandle](#)
- [CyU3PDmaChannelSetXfer](#)

- [CyU3PDmaChannelGetBuffer](#)
- [CyU3PDmaChannelCommitBuffer](#)
- [CyU3PDmaChannelDiscardBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)
- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelSetSuspend](#)
- [CyU3PDmaChannelResume](#)
- [CyU3PDmaChannelAbort](#)
- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

File

cyu3dma.h

5.6.4.12 CyU3PDmaChannelSetupRecvBuffer

Receive data to an external buffer.

This function is an override on the normal behavior of the channel and can receive data from producer to an external buffer. This function can be called only from the CY_U3P_DMA_CONFIGURED state.

The buffer that is passed as parameter for receiving the data has the following restrictions:

1. The buffer should have a size 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. All buffers used for DMA are expected to be allocated using the [CyU3PDmaBufferAlloc](#) call. This takes care of the 32 byte alignment and size restrictions. If for any reason, the buffer is not allocated using this API, then cache restrictions should be taken care of by the application.

Parameters

Parameters	Description
CyU3PDmaChannel * handle	Handle to the DMA channel to be modified.
CyU3PDmaBuffer_t * buffer_p	Pointer to structure containing the address and size of the buffer to be filled up with received data.

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PDmaChannelSetupRecvBuffer(  
    CyU3PDmaChannel * handle,  
    CyU3PDmaBuffer_t * buffer_p  
);
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaBuffer_t](#)
- [CyU3PDmaChannel](#)
- [CyU3PDmaChannelCreate](#)

- [CyU3PDmaChannelDestroy](#)
- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelGetHandle](#)
- [CyU3PDmaChannelSetXfer](#)
- [CyU3PDmaChannelGetBuffer](#)
- [CyU3PDmaChannelCommitBuffer](#)
- [CyU3PDmaChannelDiscardBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelSetSuspend](#)
- [CyU3PDmaChannelResume](#)
- [CyU3PDmaChannelAbort](#)
- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

File

cyu3dma.h

5.6.4.13 CyU3PDmaChannelWaitForRecvBuffer

Wait for the data to be filled.

This function requires that the [CyU3PDmaChannelSetupRecvBuffer](#) API to be first called. It waits for the transfer to complete and returns the buffer status. If the transfer is not completed within the specified timeout, then the function returns with a timeout error and can be called again. The [CyU3PDmaChannelWaitForCompletion](#) also waits for this but does not return the buffer status. This function must not be called when in DMA callback.

Parameters

Parameters	Description
CyU3PDmaChannel * handle	Handle to the DMA channel on which to wait.
CyU3PDmaBuffer_t * buffer_p	Output parameter which will be filled up with the address, count and status values of the DMA buffer into which data was received.
uint32_t waitOption	Duration to wait for the receive completion.

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PDmaChannelWaitForRecvBuffer(  
    CyU3PDmaChannel * handle,  
    CyU3PDmaBuffer\_t * buffer_p,  
    uint32_t waitOption  
);
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaBuffer_t](#)
- [CyU3PDmaChannel](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelDestroy](#)

- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelGetHandle](#)
- [CyU3PDmaChannelSetXfer](#)
- [CyU3PDmaChannelGetBuffer](#)
- [CyU3PDmaChannelCommitBuffer](#)
- [CyU3PDmaChannelDiscardBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)
- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelSetSuspend](#)
- [CyU3PDmaChannelResume](#)
- [CyU3PDmaChannelAbort](#)
- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

File

cyu3dma.h

5.6.4.14 CyU3PDmaChannelWaitForCompletion

Wait for the current DMA transaction to complete.

The function can be called from any thread and is invoked to wait until the current transfer is completed. This function is not supported for infinite transfers. This function waits for completion of the transfer and so is useful only for AUTO mode of operation and for external buffer operations. If DMA event notifications are enabled, then the notifications shall come before the function call returns. This function must not be called when in DMA callback.

Parameters

Parameters	Description
<code>CyU3PDmaChannel * handle</code>	Handle to the DMA channel to wait on.
<code>uint32_t waitOption</code>	Duration for which to wait.

Returns

- `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

C++

```
CyU3PReturnStatus\_t CyU3PDmaChannelWaitForCompletion(  
    CyU3PDmaChannel * handle,  
    uint32_t waitOption  
);
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaChannel](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelDestroy](#)
- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelGetHandle](#)
- [CyU3PDmaChannelSetXfer](#)
- [CyU3PDmaChannelGetBuffer](#)

- [CyU3PDmaChannelCommitBuffer](#)
- [CyU3PDmaChannelDiscardBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelSetSuspend](#)
- [CyU3PDmaChannelResume](#)
- [CyU3PDmaChannelAbort](#)
- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

File

cyu3dma.h

5.6.4.15 CyU3PDmaChannelSetWrapUp

Wraps up the current active buffer for the channel from the producer side.

The function affects only the producer socket and does not change the behavior of the consumer socket. The function will only wrap up the current buffer and the producer socket will move to the next available buffer and this call does not suspend or disable the producer socket. The call can result in data loss or unaligned packet boundaries.

The function is not intended for MANUAL_OUT channels. Also the function can be called only when the channel is in active mode or in consumer override mode.

Parameters

Parameters	Description
<code>CyU3PDmaChannel * handle</code>	Handle to the channel to be modified.

Returns

- `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

C++

```
CyU3PReturnStatus\_t CyU3PDmaChannelSetWrapUp(  
    CyU3PDmaChannel * handle  
);
```

Group

[Channel Functions](#)

Notes

This API will not work on p-port sockets if the data is less than 48 bytes. Use GPIF state-machine to trigger the buffer wrap-up in this case.

See Also

- [CyU3PDmaChannel](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelDestroy](#)
- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelGetHandle](#)
- [CyU3PDmaChannelSetXfer](#)

- [CyU3PDmaChannelGetBuffer](#)
- [CyU3PDmaChannelCommitBuffer](#)
- [CyU3PDmaChannelDiscardBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)
- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetSuspend](#)
- [CyU3PDmaChannelResume](#)
- [CyU3PDmaChannelAbort](#)
- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

File

cyu3dma.h

5.6.4.16 CyU3PDmaChannelSetSuspend

Set the suspend options for the sockets.

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 mode (SetXfer mode) and is a don't care override modes of operations.

For manual channels, this is largely not required as each buffer needs to be manually committed. But these options are still valid and can be used. For MANUAL_IN channel, only the producer socket option is considered and the consumer option is expected to be SUSP_NONE. Similarly for MANUAL_OUT channel, producer option is expected to be SUSP_NONE.

Parameters

Parameters	Description
CyU3PDmaChannel * handle	Handle to the channel to be modified.
CyU3PDmaSckSuspType_t prodSusp	Suspend option for the producer socket.
CyU3PDmaSckSuspType_t consSusp	Suspend option for the consumer socket.

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PDmaChannelSetSuspend(  
    CyU3PDmaChannel * handle,  
    CyU3PDmaSckSuspType_t prodSusp,  
    CyU3PDmaSckSuspType_t consSusp  
);
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaChannel](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelDestroy](#)
- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelGetHandle](#)

- [CyU3PDmaChannelSetXfer](#)
- [CyU3PDmaChannelGetBuffer](#)
- [CyU3PDmaChannelCommitBuffer](#)
- [CyU3PDmaChannelDiscardBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)
- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelResume](#)
- [CyU3PDmaChannelAbort](#)
- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

File

cyu3dma.h

5.6.4.17 CyU3PDmaChannelResume

Resume a suspended DMA channel.

The function can be called to resume a suspended channel. It can only be called when the channel is active after a SetXfer call. Producer and consumer suspends can be individually cleared.

For a MANUAL_IN channel, only producer socket can be cleared and for a MANUAL_OUT channel only a consumer socket can be cleared.

Parameters

Parameters	Description
CyU3PDmaChannel * handle	Handle to the channel to be resumed.
CyBool_t isProdResume	Whether to resume the producer socket.
CyBool_t isConsResume	Whether to resume the consumer socket.

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PDmaChannelResume(  
    CyU3PDmaChannel * handle,  
    CyBool_t isProdResume,  
    CyBool_t isConsResume  
);
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaChannel](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelDestroy](#)
- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelGetHandle](#)
- [CyU3PDmaChannelSetXfer](#)
- [CyU3PDmaChannelGetBuffer](#)
- [CyU3PDmaChannelCommitBuffer](#)

- [CyU3PDmaChannelDiscardBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)
- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelSetSuspend](#)
- [CyU3PDmaChannelAbort](#)
- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

File

cyu3dma.h

5.6.4.18 CyU3PDmaChannelReset

Aborts and resets a DMA channel.

The function shall abort both the producer and consumer. This function also resets the channel and flushes all buffers. The data in transition is lost and any active transaction cannot be resumed. But new transactions can be setup.

Parameters

Parameters	Description
<code>CyU3PDmaChannel * handle</code>	Handle to the channel to be reset.

Returns

- `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

C++

```
CyU3PReturnStatus\_t CyU3PDmaChannelReset(  
    CyU3PDmaChannel * handle  
);
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaChannel](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelDestroy](#)
- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelGetHandle](#)
- [CyU3PDmaChannelSetXfer](#)
- [CyU3PDmaChannelGetBuffer](#)
- [CyU3PDmaChannelCommitBuffer](#)
- [CyU3PDmaChannelDiscardBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)
- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelSetSuspend](#)

- [CyU3PDmaChannelResume](#)
- [CyU3PDmaChannelAbort](#)
- [CyU3PDmaChannelCacheControl](#)

File

cyu3dma.h

5.6.4.19 CyU3PDmaChannelAbort

Aborts a DMA channel.

The function shall abort both the producer and consumer. 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.

Parameters

Parameters	Description
<code>CyU3PDmaChannel * handle</code>	Handle to the channel to be aborted.

Returns

- `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

C++

```
CyU3PReturnStatus\_t CyU3PDmaChannelAbort(  
    CyU3PDmaChannel * handle  
);
```

Group

[Channel Functions](#)

See Also

- [CyU3PDmaChannel](#)
- [CyU3PDmaChannelCreate](#)
- [CyU3PDmaChannelDestroy](#)
- [CyU3PDmaChannelUpdateMode](#)
- [CyU3PDmaChannelGetStatus](#)
- [CyU3PDmaChannelGetHandle](#)
- [CyU3PDmaChannelSetXfer](#)
- [CyU3PDmaChannelGetBuffer](#)
- [CyU3PDmaChannelCommitBuffer](#)
- [CyU3PDmaChannelDiscardBuffer](#)
- [CyU3PDmaChannelSetupSendBuffer](#)
- [CyU3PDmaChannelSetupRecvBuffer](#)
- [CyU3PDmaChannelWaitForCompletion](#)
- [CyU3PDmaChannelWaitForRecvBuffer](#)
- [CyU3PDmaChannelSetWrapUp](#)
- [CyU3PDmaChannelSetSuspend](#)

- [CyU3PDmaChannelResume](#)
- [CyU3PDmaChannelReset](#)
- [CyU3PDmaChannelCacheControl](#)

File

cyu3dma.h





5.6.5 Multi-Channel Functions

Multi-channel are special versions of DMA channels that involve multiple producers or multiple consumers for a data flow. Since these channels will only be used rarely, the operations on such channel are kept separated in a different section which can be removed from the firmware binary when not used.

This section documents the functions that operate on multi-channels. These operations are similar to those supported on regular DMA channels.

Functions

Function	Description
CyU3PDmaEnableMulticast	Enable creation and management of DMA multicast channels.
CyU3PDmaMultiChannelCreate	Create a DMA channel. This function is to be used for only multi socket DMA channels.
CyU3PDmaMultiChannelDestroy	Destroy a DMA channel. This function is to be used for only multi socket DMA channels.
CyU3PDmaMultiChannelUpdateMode	Update the DMA mode for the multi channel.
CyU3PDmaMultiChannelGetHandle	Returns handle associated with the socket ID.
CyU3PDmaMultiChannelGetStatus	This functions returns the current multi channel status.
CyU3PDmaMultiChannelCacheControl	This function enables / disables DMA API internal d-cache handling for multi-channels.
CyU3PDmaMultiChannelSetXfer	Set a transfer on the DMA channel. This function should be used for only multi socket operations.
CyU3PDmaMultiChannelGetBuffer	Get the current buffer pointer. This function shall be used only with multi socket operations.
CyU3PDmaMultiChannelCommitBuffer	Commit the buffer to be sent to the consumer. This function shall be used only with multi socket operations.
CyU3PDmaMultiChannelDiscardBuffer	Discard buffer in the current DMA descriptor. This functions should be used only with multi socket DMA channels.
CyU3PDmaMultiChannelSetupSendBuffer	Send an external buffer to the consumer.
CyU3PDmaMultiChannelWaitForRecvBuffer	Wait for the data to be filled.
CyU3PDmaMultiChannelSetupRecvBuffer	Receive data to an external buffer.
CyU3PDmaMultiChannelWaitForCompletion	Wait for the current DMA transaction to complete. This function should be used only for the multi socket DMA channels.
CyU3PDmaMultiChannelSetWrapUp	Wraps up the current active buffer for the channel from the producer side.

 CyU3PDmaMultiChannelSetSuspend	Set the suspend options for the sockets.
 CyU3PDmaMultiChannelResume	Resume a suspended DMA channel.
 CyU3PDmaMultiChannelReset	Aborts and resets a DMA channel. This function shall be used only with multi socket operations.
 CyU3PDmaMultiChannelAbort	Aborts a DMA channel. This function shall be used only with multi socket operations.

Group

[DMA Functions](#)

Legend

	Method
---	--------

5.6.5.1 CyU3PDmaEnableMulticast

Enable creation and management of DMA multicast channels.

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.

Returns

- None

C++

```
void CyU3PDmaEnableMulticast(  
    void  
);
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaMultiChannelCreate](#)

File

cyu3dma.h

5.6.5.2 CyU3PDmaMultiChannelCreate

Create a DMA channel. This function is to be used for only multi socket DMA channels.

A DMA channel requires producer socket(s), consumer socket(s), a set of buffers and a callback function. This function must not be called when in DMA callback.

Parameters

Parameters	Description
CyU3PDmaMultiChannel * handle	Pointer to multi-channel structure that is to be initialized.
CyU3PDmaMultiType_t type	Type of DMA channel to be created.
CyU3PDmaMultiChannelConfig_t * config	Configuration information about the channel to be created.

Returns

- 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](#)

C++

```
CyU3PReturnStatus_t CyU3PDmaMultiChannelCreate(  
    CyU3PDmaMultiChannel * handle,  
    CyU3PDmaMultiType\_t type,  
    CyU3PDmaMultiChannelConfig\_t * config  
);
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaMultiType_t](#)
- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelConfig_t](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelGetHandle](#)
- [CyU3PDmaMultiChannelUpdateMode](#)
- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelCommitBuffer](#)
- [CyU3PDmaMultiChannelDiscardBuffer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)

- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelSetWrapUp](#)
- [CyU3PDmaMultiChannelSetSuspend](#)
- [CyU3PDmaMultiChannelResume](#)
- [CyU3PDmaMultiChannelAbort](#)
- [CyU3PDmaMultiChannelReset](#)
- [CyU3PDmaMultiChannelGetStatus](#)
- [CyU3PDmaMultiChannelCacheControl](#)

File

cyu3dma.h

5.6.5.3 CyU3PDmaMultiChannelDestroy

Destroy a DMA channel. This function is to be used for only multi socket DMA channels.

This should be called once the DMA channel is no longer required. This function must not be called when in DMA callback.

Parameters

Parameters	Description
CyU3PDmaMultiChannel * handle	Pointer to the multi-channel to be de-initialized.

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PDmaMultiChannelDestroy(  
    CyU3PDmaMultiChannel * handle  
);
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelGetHandle](#)
- [CyU3PDmaMultiChannelUpdateMode](#)
- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelCommitBuffer](#)
- [CyU3PDmaMultiChannelDiscardBuffer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)
- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelSetWrapUp](#)
- [CyU3PDmaMultiChannelSetSuspend](#)
- [CyU3PDmaMultiChannelResume](#)
- [CyU3PDmaMultiChannelAbort](#)

- [CyU3PDmaMultiChannelReset](#)
- [CyU3PDmaMultiChannelGetStatus](#)
- [CyU3PDmaMultiChannelCacheControl](#)

File

cyu3dma.h

5.6.5.4 CyU3PDmaMultiChannelUpdateMode

Update the DMA mode for the multi channel.

The DMA mode is specified during creation of the channel and if this needs to be changed, then the API needs to be invoked. This API call can be made only when the DMA channel is in configured state (when there are no transfers setup).

Parameters

Parameters	Description
CyU3PDmaMultiChannel * handle	Handle to the multi-channel to be modified.
CyU3PDmaMode_t dmaMode	Desired DMA mode.

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PDmaMultiChannelUpdateMode(  
    CyU3PDmaMultiChannel * handle,  
    CyU3PDmaMode\_t dmaMode  
);
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaMode_t](#)
- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelGetHandle](#)
- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelCommitBuffer](#)
- [CyU3PDmaMultiChannelDiscardBuffer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)
- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)

- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelSetWrapUp](#)
- [CyU3PDmaMultiChannelSetSuspend](#)
- [CyU3PDmaMultiChannelResume](#)
- [CyU3PDmaMultiChannelAbort](#)
- [CyU3PDmaMultiChannelReset](#)
- [CyU3PDmaMultiChannelGetStatus](#)

File

cyu3dma.h

5.6.5.5 CyU3PDmaMultiChannelGetHandle

Returns handle associated with the socket ID.

This function is used to identify if there is any DMA channel associated with a socket. This function should be used only with multi DMA channels.

Parameters

Parameters	Description
CyU3PDmaSocketId_t sckId	ID of the socket whose channel handle is required.

Returns

- Handle to the Multi-channel structure corresponding to the socket. A NULL return indicates
- that an invalid socket ID was passed.

C++

```
CyU3PDmaMultiChannel * CyU3PDmaMultiChannelGetHandle(  
    CyU3PDmaSocketId\_t sckId  
);
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaSocketId_t](#)
- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelGetStatus](#)

File

cyu3dma.h

5.6.5.6 CyU3PDmaMultiChannelGetStatus

This functions returns the current multi channel status.

The function can be used to keep track of the data transfers and even for transfer progress. It also returns the current channel state. In case of override modes, this function always returns the byte count. The count field is updated only at buffer boundaries and so does not give real time information. The function is for multi channels.

Parameters

Parameters	Description
CyU3PDmaMultiChannel * handle	Handle to the DMA channel to query.
CyU3PDmaState_t * state	Output parameter that will be filled with state of the channel.
uint32_t * prodXferCount	Output parameter that will be filled with transfer count on the producer socket in DMA mode units.
uint32_t * consXferCount	Output parameter that will be filled with transfer count on the consumer socket in DMA mode units.
uint8_t sckIndex	The socket index for retrieving information on the multi socket side.

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PDmaMultiChannelGetStatus(  
    CyU3PDmaMultiChannel * handle,  
    CyU3PDmaState_t * state,  
    uint32_t * prodXferCount,  
    uint32_t * consXferCount,  
    uint8_t sckIndex  
);
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaState_t](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)

- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelAbort](#)
- [CyU3PDmaMultiChannelReset](#)

File

cyu3dma.h

5.6.5.7 CyU3PDmaMultiChannelCacheControl

This function enables / disables DMA API internal d-cache handling for multi-channels.

This function provides control over DMA API internal d-cache handling feature per individual channel. The global behaviour is selected based on [CyU3PDeviceCacheControl](#) API. This API allows to override this option for a particular channel. It should be noted that the [CyU3PDeviceCacheControl](#) should be invoked before even initializing the RTOS whereas this API can be done after a channel is created. However the call should be made when the channel is idle (CONFIGURED state). This API makes sense only when the D-cache is enabled.

Parameters

Parameters	Description
CyU3PDmaMultiChannel * handle	Handle to the DMA channel.
CyBool_t isDmaHandledDCache	Whether to enable handling or not.

Returns

- [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)

C++

```
CyU3PReturnStatus\_t CyU3PDmaMultiChannelCacheControl(  
    CyU3PDmaMultiChannel * handle,  
    CyBool\_t isDmaHandledDCache  
);
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaMultiChannel](#)
- [CyU3PDeviceCacheControl](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)

File

cyu3dma.h

5.6.5.8 CyU3PDmaMultiChannelSetXfer

Set a transfer on the DMA channel. This function should be used for only multi socket operations.

The function starts a transaction the selected DMA channel. It should be invoked only when the channel is in CY_U3P_DMA_CONFIGURED state.

Parameters

Parameters	Description
CyU3PDmaMultiChannel * handle	Handle to the multi-channel to be modified.
uint32_t count	Size of the transfer to be started. Can be zero to denote infinite data transfer.
uint16_t multiSckOffset	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.

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PDmaMultiChannelSetXfer(  
    CyU3PDmaMultiChannel * handle,  
    uint32_t count,  
    uint16_t multiSckOffset  
);
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelGetHandle](#)
- [CyU3PDmaMultiChannelUpdateMode](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelCommitBuffer](#)
- [CyU3PDmaMultiChannelDiscardBuffer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)

- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelSetWrapUp](#)
- [CyU3PDmaMultiChannelSetSuspend](#)
- [CyU3PDmaMultiChannelResume](#)
- [CyU3PDmaMultiChannelAbort](#)
- [CyU3PDmaMultiChannelReset](#)
- [CyU3PDmaMultiChannelGetStatus](#)

File

cyu3dma.h

5.6.5.9 CyU3PDmaMultiChannelGetBuffer

Get the current buffer pointer. This function shall be used only with multi socket operations.

This function is generally for manual mode operations.

The function returns the data buffer received from the producer socket(s). In case of Many to one type of channels, this function receives buffers from each of the producer sockets in a round robin fashion. The starting point is decided by the multiSckOffset value specified during SetXfer phase. The wait_option specifies the timeout value.

The buffer pointer returned, points to the buffer actually populated by the producer socket and so does not include the producer offsets. The count returned also reports actual count of data received by the producer socket.

For example, if there is a producer header offset of 12 bytes, the valid data starts at buffer_p->buffer[0] and the producer header can be accessed at buffer_p->buffer[-12]. Care should be taken to only access memory locations actually allocated and not be access beyond producer header and footer region.

For MANY_TO_ONE AUTO channel this is a special API to be called only when the consumer socket is in the suspended state. In all other cases this will return CY_U3P_ERROR_INVALID_SEQUENCE. This API is used to look at the current consumer buffer in case of a SUSP_CONS_PARTIAL_BUF situation.

This function must not be called when in DMA callback with a non-zero wait option.

Parameters

Parameters	Description
CyU3PDmaMultiChannel * handle	Handle to the multi-channel to be modified.
CyU3PDmaBuffer_t * buffer_p	Output parameter that will be filled with address, size and status of the buffer with received data.
uint32_t waitOption	Duration for which to wait for data.

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PDmaMultiChannelGetBuffer(  
    CyU3PDmaMultiChannel * handle,  
    CyU3PDmaBuffer_t * buffer_p,  
    uint32_t waitOption  
);
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaBuffer_t](#)
- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelGetHandle](#)
- [CyU3PDmaMultiChannelUpdateMode](#)
- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelCommitBuffer](#)
- [CyU3PDmaMultiChannelDiscardBuffer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)
- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelSetWrapUp](#)
- [CyU3PDmaMultiChannelSetSuspend](#)
- [CyU3PDmaMultiChannelResume](#)
- [CyU3PDmaMultiChannelAbort](#)
- [CyU3PDmaMultiChannelReset](#)
- [CyU3PDmaMultiChannelGetStatus](#)

File

`cyu3dma.h`

5.6.5.10 CyU3PDmaMultiChannelCommitBuffer

Commit the buffer to be sent to the consumer. This function shall be used only with multi socket operations.

The function is generally for the manual multi socket DMA channels. The function sends a buffer in the current DMA descriptor to the consumer socket(s). The count of data provided must be the amount of data required to be sent out of the consumer. This does not include the consumer offset.

This API is used for MANY_TO_ONE AUTO channel for special case handling. It should be called only when the consumer socket is in the suspended state. In all other cases this will return CY_U3P_ERROR_INVALID_SEQUENCE. This API is used to commit the current consumer buffer in case of a SUSP_CONS_PARTIAL_BUF situation.

Parameters

Parameters	Description
CyU3PDmaMultiChannel * handle	Handle to the multi-channel to be modified.
uint16_t count	Size of the memory buffer being committed. The address of the buffer is implicit and is taken from the active descriptor.
uint16_t bufStatus	Status of the buffer being committed.

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PDmaMultiChannelCommitBuffer(  
    CyU3PDmaMultiChannel * handle,  
    uint16_t count,  
    uint16_t bufStatus  
);
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)

- [CyU3PDmaMultiChannelGetHandle](#)
- [CyU3PDmaMultiChannelUpdateMode](#)
- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelDiscardBuffer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)
- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelSetWrapUp](#)
- [CyU3PDmaMultiChannelSetSuspend](#)
- [CyU3PDmaMultiChannelResume](#)
- [CyU3PDmaMultiChannelAbort](#)
- [CyU3PDmaMultiChannelReset](#)
- [CyU3PDmaMultiChannelGetStatus](#)

File

cyu3dma.h

5.6.5.11 CyU3PDmaMultiChannelDiscardBuffer

Discard buffer in the current DMA descriptor. This functions should be used only with multi socket DMA channels.

The function is generally for the manual mode multi socket DMA channels to discard the current buffer.

AUTO Channel special case: This API is also used for AUTO channels for special case handling. For AUTO channels, this API is used to discard the current consumer buffer only in case of a SUSP_CONS_PARTIAL_BUF situation. The discard buffer will function only when the consumer socket is in the suspended state. If the socket is not suspended, this API will return CY_U3P_ERROR_INVALID_SEQUENCE.

Parameters

Parameters	Description
CyU3PDmaMultiChannel * handle	Handle to the multi-channel to be modified.

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PDmaMultiChannelDiscardBuffer(  
    CyU3PDmaMultiChannel * handle  
);
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelGetHandle](#)
- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelCommitBuffer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)

- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelSetWrapUp](#)
- [CyU3PDmaMultiChannelSetSuspend](#)
- [CyU3PDmaMultiChannelResume](#)
- [CyU3PDmaMultiChannelAbort](#)
- [CyU3PDmaMultiChannelReset](#)
- [CyU3PDmaMultiChannelGetStatus](#)

File

cyu3dma.h

5.6.5.12 CyU3PDmaMultiChannelSetupSendBuffer

Send an external buffer to the consumer.

This function is an override on the normal behavior of the channel and can send an external buffer to the consumer. This function can be called only from the CY_U3P_DMA_CONFIGURED state.

The buffers used for DMA operations are expected to be allocated using [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. This is to satisfy the 32 byte cache lines.

Parameters

Parameters	Description
CyU3PDmaMultiChannel * handle	Handle to the DMA multi channel to be modified.
CyU3PDmaBuffer_t * buffer_p	Pointer to structure containing address, size and status of the DMA buffer to be sent out.
uint16_t multiSckOffset	Consumer Socket id to send the data. For a many to one channel, this would be a zero; and for a one to many channel, this would be a consumer socket offset.

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PDmaMultiChannelSetupSendBuffer(  
    CyU3PDmaMultiChannel * handle,  
    CyU3PDmaBuffer_t * buffer_p,  
    uint16_t multiSckOffset  
);
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaBuffer_t](#)
- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelGetHandle](#)
- [CyU3PDmaMultiChannelUpdateMode](#)

- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelCommitBuffer](#)
- [CyU3PDmaMultiChannelDiscardBuffer](#)
- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelSetWrapUp](#)
- [CyU3PDmaMultiChannelSetSuspend](#)
- [CyU3PDmaMultiChannelResume](#)
- [CyU3PDmaMultiChannelAbort](#)
- [CyU3PDmaMultiChannelReset](#)
- [CyU3PDmaMultiChannelGetStatus](#)

File

cyu3dma.h

5.6.5.13 CyU3PDmaMultiChannelWaitForRecvBuffer

Wait for the data to be filled.

This function requires that the [CyU3PDmaMultiChannelSetupRecvBuffer](#) API to be first called. It waits for the transfer to complete and returns the buffer status. If the transfer is not completed within the specified timeout, then the function returns with a timeout error and can be called again. The [CyU3PDmaMultiChannelWaitForCompletion](#) also waits for this but does not return the buffer status. This function must not be called when in DMA callback.

Parameters

Parameters	Description
CyU3PDmaMultiChannel * handle	Handle to the DMA channel on which to wait.
CyU3PDmaBuffer_t * buffer_p	Output parameter which will be filled up with the address, count and status values of the DMA buffer into which data was received.
uint32_t waitOption	Duration to wait for the receive completion.

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PDmaMultiChannelWaitForRecvBuffer(  
    CyU3PDmaMultiChannel * handle,  
    CyU3PDmaBuffer_t * buffer_p,  
    uint32_t waitOption  
);
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaBuffer_t](#)
- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelGetHandle](#)

- [CyU3PDmaMultiChannelUpdateMode](#)
- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelCommitBuffer](#)
- [CyU3PDmaMultiChannelDiscardBuffer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)
- [CyU3PDmaMultiChannelSetWrapUp](#)
- [CyU3PDmaMultiChannelSetSuspend](#)
- [CyU3PDmaMultiChannelResume](#)
- [CyU3PDmaMultiChannelAbort](#)
- [CyU3PDmaMultiChannelReset](#)
- [CyU3PDmaMultiChannelGetStatus](#)

File

cyu3dma.h

5.6.5.14 CyU3PDmaMultiChannelSetupRecvBuffer

Receive data to an external buffer.

This function is an override on the normal behavior of the channel and can receive data from producer to an external buffer. This function can be called only from the CY_U3P_DMA_CONFIGURED state.

The buffer that is passed as parameter for receiving the data has the following restrictions:

1. The buffer should have a size 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. All buffers used for DMA are expected to be allocated using the [CyU3PDmaBufferAlloc](#) call. This takes care of the 32 byte alignment and size restrictions. If for any reason, the buffer is not allocated using this API, then cache restrictions should be taken care of by the application.

Parameters

Parameters	Description
CyU3PDmaMultiChannel * handle	Handle to the DMA multi channel to be modified.
CyU3PDmaBuffer_t * buffer_p	Pointer to structure containing the address and size of the buffer to be filled up with received data.
uint16_t multiSckOffset	Producer Socket id to receive the data. For a one to many channel, this would be a zero; and for a many to one channel, this would be a producer socket offset.

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PDmaMultiChannelSetupRecvBuffer(  
    CyU3PDmaMultiChannel * handle,  
    CyU3PDmaBuffer_t * buffer_p,  
    uint16_t multiSckOffset  
);
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaBuffer_t](#)
- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelGetHandle](#)
- [CyU3PDmaMultiChannelUpdateMode](#)
- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelCommitBuffer](#)
- [CyU3PDmaMultiChannelDiscardBuffer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelSetWrapUp](#)
- [CyU3PDmaMultiChannelSetSuspend](#)
- [CyU3PDmaMultiChannelResume](#)
- [CyU3PDmaMultiChannelAbort](#)
- [CyU3PDmaMultiChannelReset](#)
- [CyU3PDmaMultiChannelGetStatus](#)

File

cyu3dma.h

5.6.5.15 CyU3PDmaMultiChannelWaitForCompletion

Wait for the current DMA transaction to complete. This function should be used only for the multi socket DMA channels.

The function can be called from any thread and is invoked to wait until the current transfer is completed. This function is not supported for infinite transfers. This function waits for completion of the transfer and so is useful only for AUTO mode of operations. If DMA event notifications are enabled, then the notifications shall come before the function call returns. This function must not be called when in DMA callback.

Parameters

Parameters	Description
<code>CyU3PDmaMultiChannel * handle</code>	Handle to the multi-channel to wait on.
<code>uint32_t waitOption</code>	Duration for which to wait.

Returns

- `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

C++

```
CyU3PReturnStatus\_t CyU3PDmaMultiChannelWaitForCompletion(  
    CyU3PDmaMultiChannel * handle,  
    uint32_t waitOption  
);
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelCommitBuffer](#)
- [CyU3PDmaMultiChannelDiscardBuffer](#)

- [CyU3PDmaMultiChannelSetupSendBuffer](#)
- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelAbort](#)
- [CyU3PDmaMultiChannelReset](#)
- [CyU3PDmaMultiChannelGetStatus](#)

File

cyu3dma.h

5.6.5.16 CyU3PDmaMultiChannelSetWrapUp

Wraps up the current active buffer for the channel from the producer side.

The function affects only the producer socket and does not change the behavior of the consumer socket. The function will only wrap up the current buffer and the producer socket will move to the next available buffer and this call does not suspend or disable the producer socket. The call can result in data loss or unaligned packet boundaries.

The function can be called only when the channel is in active mode or in consumer override mode.

Parameters

Parameters	Description
<code>CyU3PDmaMultiChannel * handle</code>	Handle to the channel to be modified.
<code>uint16_t 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 be always zero.

Returns

- `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

C++

```
CyU3PReturnStatus_t CyU3PDmaMultiChannelSetWrapUp(  
    CyU3PDmaMultiChannel * handle,  
    uint16_t multiSckOffset  
);
```

Group

[Multi-Channel Functions](#)

Notes

This API will not work on p-port sockets if the data is less than 48 bytes. Use GPIF state-machine to trigger the buffer wrap-up in this case.

See Also

- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelGetHandle](#)
- [CyU3PDmaMultiChannelUpdateMode](#)

- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelCommitBuffer](#)
- [CyU3PDmaMultiChannelDiscardBuffer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)
- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelSetSuspend](#)
- [CyU3PDmaMultiChannelResume](#)
- [CyU3PDmaMultiChannelAbort](#)
- [CyU3PDmaMultiChannelReset](#)
- [CyU3PDmaMultiChannelGetStatus](#)

File

cyu3dma.h

5.6.5.17 CyU3PDmaMultiChannelSetSuspend

Set the suspend options for the sockets.

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 mode (SetXfer mode) and is a don't care override modes of operations.

For manual channels, this is largely not required as each buffer needs to be manually committed. But these options are still valid and can be used. Options can be set only for socket on the single side. For MANY_TO_ONE channel only consumer socket option can be used and for ONE_TO_MANY, only producer socket option can be used.

Parameters

Parameters	Description
CyU3PDmaMultiChannel * handle	Handle to the multi-channel to be suspended.
CyU3PDmaSckSuspType_t prodSusp	Suspend option for the producer socket.
CyU3PDmaSckSuspType_t consSusp	Suspend option for the consumer socket.

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PDmaMultiChannelSetSuspend(  
    CyU3PDmaMultiChannel * handle,  
    CyU3PDmaSckSuspType_t prodSusp,  
    CyU3PDmaSckSuspType_t consSusp  
);
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaSckSuspType_t](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelGetHandle](#)
- [CyU3PDmaMultiChannelUpdateMode](#)

- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelCommitBuffer](#)
- [CyU3PDmaMultiChannelDiscardBuffer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)
- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelSetWrapUp](#)
- [CyU3PDmaMultiChannelResume](#)
- [CyU3PDmaMultiChannelAbort](#)
- [CyU3PDmaMultiChannelReset](#)
- [CyU3PDmaMultiChannelGetStatus](#)

File

cyu3dma.h

5.6.5.18 CyU3PDmaMultiChannelResume

Resume a suspended DMA channel.

The function can be called to resume a suspended channel. It can only be called when the channel is active after a SetXfer call. Producer and consumer suspends can be individually cleared.

For MANY_TO_ONE channel, only consumer side can be resumed and for ONE_TO_MANY channel, only producer side can be resumed.

Parameters

Parameters	Description
CyU3PDmaMultiChannel * handle	Handle to the multi-channel to be resumed.
CyBool_t isProdResume	Whether to resume the producer socket.
CyBool_t isConsResume	Whether to resume the consumer socket.

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PDmaMultiChannelResume(  
    CyU3PDmaMultiChannel * handle,  
    CyBool_t isProdResume,  
    CyBool_t isConsResume  
);
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelGetHandle](#)
- [CyU3PDmaMultiChannelUpdateMode](#)
- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelCommitBuffer](#)
- [CyU3PDmaMultiChannelDiscardBuffer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)

- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelSetWrapUp](#)
- [CyU3PDmaMultiChannelSetSuspend](#)
- [CyU3PDmaMultiChannelAbort](#)
- [CyU3PDmaMultiChannelReset](#)
- [CyU3PDmaMultiChannelGetStatus](#)

File

cyu3dma.h

5.6.5.19 CyU3PDmaMultiChannelReset

Aborts and resets a DMA channel. This function shall be used only with multi socket operations.

The function shall abort both the producer and consumer. This function also resets the channel and flushes all buffers. The data in transition is lost and any active transaction cannot be resumed. But new transactions can be setup.

Parameters

Parameters	Description
<code>CyU3PDmaMultiChannel * handle</code>	Handle to the multi-channel to be reset.

Returns

- `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

C++

```
CyU3PReturnStatus_t CyU3PDmaMultiChannelReset(  
    CyU3PDmaMultiChannel * handle  
) ;
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelGetHandle](#)
- [CyU3PDmaMultiChannelUpdateMode](#)
- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelCommitBuffer](#)
- [CyU3PDmaMultiChannelDiscardBuffer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)
- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelSetWrapUp](#)
- [CyU3PDmaMultiChannelSetSuspend](#)

- [CyU3PDmaMultiChannelResume](#)
- [CyU3PDmaMultiChannelAbort](#)
- [CyU3PDmaMultiChannelGetStatus](#)

File

cyu3dma.h

5.6.5.20 CyU3PDmaMultiChannelAbort

Aborts a DMA channel. This function shall be used only with multi socket operations.

The function shall abort both the producer and consumer. 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.

Parameters

Parameters	Description
<code>CyU3PDmaMultiChannel * handle</code>	Handle to the multi-channel to be aborted.

Returns

- `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

C++

```
CyU3PReturnStatus_t CyU3PDmaMultiChannelAbort(  
    CyU3PDmaMultiChannel * handle  
) ;
```

Group

[Multi-Channel Functions](#)

See Also

- [CyU3PDmaMultiChannel](#)
- [CyU3PDmaMultiChannelCreate](#)
- [CyU3PDmaMultiChannelDestroy](#)
- [CyU3PDmaMultiChannelGetHandle](#)
- [CyU3PDmaMultiChannelUpdateMode](#)
- [CyU3PDmaMultiChannelSetXfer](#)
- [CyU3PDmaMultiChannelGetBuffer](#)
- [CyU3PDmaMultiChannelCommitBuffer](#)
- [CyU3PDmaMultiChannelDiscardBuffer](#)
- [CyU3PDmaMultiChannelSetupSendBuffer](#)
- [CyU3PDmaMultiChannelSetupRecvBuffer](#)
- [CyU3PDmaMultiChannelWaitForCompletion](#)
- [CyU3PDmaMultiChannelWaitForRecvBuffer](#)
- [CyU3PDmaMultiChannelSetWrapUp](#)
- [CyU3PDmaMultiChannelSetSuspend](#)

- [CyU3PDmaMultiChannelResume](#)
- [CyU3PDmaMultiChannelReset](#)
- [CyU3PDmaMultiChannelGetStatus](#)

File

cyu3dma.h

6 USB Management

The FX3 device has a single USB port which can function as a user programmable USB device port or single USB host port. The USB On-The-Go controller supports USB OTG 2.0 specification.

FX3 USB port in device mode of operation, it can function as per USB 3.0 specification and function at super speed, high speed and full speed depending upon the remote host attached. When configured a USB host, the port supports USB 2.0 operations at high speed, full speed and low speed according to the speed supported by the peripheral attached. 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 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.








Topics

Topic	Description
USB Constants	These are USB constants that are derived from the USB protocol specification.
USB OTG Management	The FX3 device supports programmable USB OTG 2.0 implementation which supports device mode operations at USB-SS, USB-HS and USB-FS speeds and host mode operations at USB-HS, USB-FS and USB-LS speeds. The single USB port can function as a device or as a host depending on the type of peripheral attached. It can also perform ACA or Motorola EMU charger detection based on the ID pin state.
USB Device Management	The FX3 device can function as a user programmable USB device (or peripheral) at Super Speed, High Speed and Full Speed. The USB device mode driver built into the firmware framework takes care of the connection setup and management at all connection speeds. A set of device mode APIs are provided so that the user application logic can talk to the USB driver to configure the behavior of the device.
USB Host Management	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.

6.1 USB Constants

These are USB constants that are derived from the USB protocol specification.

Enumerations


Enumeration	Description
 CyU3PUsbEpType_t	Enumeration of the endpoint types.
 CyU3PUsbSetupCmds	Standard device request codes.
 CyU3PUsbDescType	Enumeration of descriptor types.
 CyU3PUsbDevCapType	Device capability type codes.
 CyU3PUsb3PacketType	USB 3.0 packet type codes.
 CyU3PUsb3TpSubType	USB 3.0 transaction packet sub type codes.
 CyU3PUsbFeatureSelector	List of USB Feature selector codes.

 CyU3PUsbLinkState_t	Link state machine states.
---	----------------------------

Group

[USB Management](#)

Legend

	Enumeration
---	-------------

Macros

Macro	Description
CY_U3P_USB_STANDARD_RQT	The USB standard request
CY_U3P_USB_CLASS_RQT	The USB class request
CY_U3P_USB_VENDOR_RQT	The USB vendor request
CY_U3P_USB_GS_DEVICE	The standard device request, GET_STATUS device
CY_U3P_USB_GS_ENDPOINT	The standard device request, GET_STATUS endpoint
CY_U3P_USB_GS_INTERFACE	The standard device request, GET_STATUS interface
CY_U3P_USB_TARGET_DEVICE	The USB Target Device
CY_U3P_USB_TARGET_INTF	The USB Target Interface
CY_U3P_USB_TARGET_ENDPT	The USB Target Endpoint
CY_U3P_USB_TARGET_OTHER	The USB Target Other
CY_FX3_USB_MAX_STRING_DESC_INDEX	Number of string descriptors that are supported by the FX3 booter.
CY_U3P_USB_TARGET_MASK	The Target mask
CY_U3P_USB_TYPE_MASK	The request type mask
CY_U3P_USB3_TP_DEVADDR_POS	Position of device address field in USB 3.0 LMP and TP.
CY_U3P_USB3_TP_EPNUM_POS	Position of endpoint number field in USB 3.0 TP.
CY_U3P_USB_RESERVED_RQT	The USB reserved request
CY_U3P_USB_OTG_STATUS_SELECTOR	Index field of the setup request indicating OTG GET_STATUS request.

6.1.1 CY_U3P_USB_STANDARD_RQT

The USB standard request

C++

```
#define CY_U3P_USB_STANDARD_RQT (0x00) /* The USB standard request */
```

Group

[USB Constants](#)

File

cyu3usbconst.h

6.1.2 CY_U3P_USB_CLASS_RQT

The USB class request

C++

```
#define CY_U3P_USB_CLASS_RQT (0x20) /* The USB class request */
```

Group

[USB Constants](#)

File

cyu3usbconst.h

6.1.3 CY_U3P_USB_VENDOR_RQT

The USB vendor request

C++

```
#define CY_U3P_USB_VENDOR_RQT (0x40) /* The USB vendor request */
```

Group

[USB Constants](#)

File

cyu3usbconst.h

6.1.4 CY_U3P_USB_GS_DEVICE

The standard device request, GET_STATUS device

C++

```
#define CY_U3P_USB_GS_DEVICE (0x80) /* The standard device request, GET_STATUS device */
```

Group

[USB Constants](#)

File

cyu3usbconst.h

6.1.5 CY_U3P_USB_GS_ENDPOINT

The standard device request, GET_STATUS endpoint

C++

```
#define CY_U3P_USB_GS_ENDPOINT (0x82) /* The standard device request, GET_STATUS  
endpoint */
```

Group

[USB Constants](#)

File

cyu3usbconst.h

6.1.6 CY_U3P_USB_GS_INTERFACE

The standard device request, GET_STATUS interface

C++

```
#define CY_U3P_USB_GS_INTERFACE (0x81) /* The standard device request, GET_STATUS  
interface */
```

Group

[USB Constants](#)

File

cyu3usbconst.h

6.1.7 CY_U3P_USB_TARGET_DEVICE

The USB Target Device

C++

```
#define CY_U3P_USB_TARGET_DEVICE (0x00) /* The USB Target Device */
```

Group

[USB Constants](#)

File

cyu3usbconst.h

6.1.8 CY_U3P_USB_TARGET_INTF

The USB Target Interface

C++

```
#define CY_U3P_USB_TARGET_INTF (0x01) /* The USB Target Interface */
```

Group

[USB Constants](#)

File

cyu3usbconst.h

6.1.9 CY_U3P_USB_TARGET_ENDPT

The USB Target Endpoint

C++

```
#define CY_U3P_USB_TARGET_ENDPT (0x02) /* The USB Target Endpoint */
```

Group

[USB Constants](#)

File

cyu3usbconst.h

6.1.10 CY_U3P_USB_TARGET_OTHER

The USB Target Other

C++

```
#define CY_U3P_USB_TARGET_OTHER (0x03) /* The USB Target Other */
```

Group

[USB Constants](#)

File

cyu3usbconst.h

6.1.11 CY_FX3_USB_MAX_STRING_DESC_INDEX

Number of string descriptors that are supported by the FX3 booter.

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.

C++

```
#define CY_FX3_USB_MAX_STRING_DESC_INDEX (16)
```

Group

[USB Constants](#)

File

cyfx3usb.h

6.1.12 CyU3PUsbEpType_t

Enumeration of the endpoint types.

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 requirement.

C++

```
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  
};
```

Group

[USB Constants](#)

Members

Members	Description
CY_U3P_USB_EP_CONTROL = 0	Control Endpoint Type
CY_U3P_USB_EP_ISO = 1	Isochronous Endpoint Type
CY_U3P_USB_EP_BULK = 2	Bulk Endpoint Type
CY_U3P_USB_EP_INTR = 3	Interrupt Endpoint Type

File

cyu3usbconst.h

6.1.13 CyU3PUsbSetupCmds

Standard device request codes.

These are the various standard requests received from the USB host. The device is expected to respond to all these requests.

C++

```
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  
};
```

Group

[USB Constants](#)

Members

Members	Description
CY_U3P_USB_SC_GET_STATUS = 0x00	Get status request.
CY_U3P_USB_SC_CLEAR_FEATURE	Clear feature.
CY_U3P_USB_SC_RESERVED	Reserved command.
CY_U3P_USB_SC_SET_FEATURE	Set feature.
CY_U3P_USB_SC_SET_ADDRESS = 0x05	Set address.
CY_U3P_USB_SC_GET_DESCRIPTOR	Get descriptor.
CY_U3P_USB_SC_SET_DESCRIPTOR	Set descriptor.
CY_U3P_USB_SC_GET_CONFIGURATION	Get configuration.
CY_U3P_USB_SC_SET_CONFIGURATION	Set configuration.
CY_U3P_USB_SC_GET_INTERFACE	Get interface (alternate setting).
CY_U3P_USB_SC_SET_INTERFACE	Set interface (alternate setting).
CY_U3P_USB_SC_SYNC_FRAME	Synch frame.
CY_U3P_USB_SC_SET_SEL = 0x30	Set system exit latency.
CY_U3P_USB_SC_SET_ISOC_DELAY	Set isochronous delay.

File

cyu3usbconst.h

6.1.14 CyU3PUsbDescType

Enumeration of descriptor types.

A USB device is identified by the descriptors provided. The following are the standard defined descriptors.

C++

```
enum CyU3PUsbDescType {
    CY_U3P_USB_DEVICE_DESCR = 0x01,
    CY_U3P_USB_CONFIG_DESCR,
    CY_U3P_USB_STRING_DESCR,
    CY_U3P_USB_INTRFC_DESCR,
    CY_U3P_USB_ENDPNT_DESCR,
    CY_U3P_USB_DEVQUAL_DESCR,
    CY_U3P_USB_OTHERSPEED_DESCR,
    CY_U3P_USB_INTRFC_POWER_DESCR,
    CY_U3P_USB_OTG_DESCR,
    CY_U3P_BOS_DESCR = 0x0F,
    CY_U3P_DEVICE_CAPB_DESCR,
    CY_U3P_USB_HID_DESCR = 0x21,
    CY_U3P_USB_REPORT_DESCR,
    CY_U3P_SS_EP_COMPN_DESCR = 0x30
};
```

Group

[USB Constants](#)

Members

Members	Description
CY_U3P_USB_DEVICE_DESCR = 0x01	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_USB_OTG_DESCR	OTG descriptor
CY_U3P_BOS_DESCR = 0x0F	BOS descriptor
CY_U3P_DEVICE_CAPB_DESCR	Device Capability descriptor
CY_U3P_USB_HID_DESCR = 0x21	HID descriptor
CY_U3P_USB_REPORT_DESCR	Report descriptor

CY_U3P_SS_EP_COMPN_DESCR = 0x30	End Point companion descriptor
---------------------------------	--------------------------------

File

cyu3usbconst.h

6.1.15 CyU3PUsbDevCapType

Device capability type codes.

The following are the various extended capabilities of the USB device.

C++

```
enum CyU3PUsbDevCapType {  
    CY_U3P_WIRELESS_USB_CAPB_TYPE = 0x01,  
    CY_U3P_USB2_EXTN_CAPB_TYPE,  
    CY_U3P_SS_USB_CAPB_TYPE,  
    CY_U3P_CONTAINER_ID_CAPB_TYPE  
};
```

Group

[USB Constants](#)

Members

Members	Description
CY_U3P_WIRELESS_USB_CAPB_TYPE = 0x01	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	instance unique ID used to identify the instance across all operating modes.

File

cyu3usbconst.h

6.1.16 CyU3PUsb3PacketType

USB 3.0 packet type codes.

The following are the various USB 3.0 packets types.

C++

```
enum CyU3PUsb3PacketType {  
    CY_U3P_USB3_PACK_TYPE_LMP = 0x00,  
    CY_U3P_USB3_PACK_TYPE_TP = 0x04,  
    CY_U3P_USB3_PACK_TYPE_DPH = 0x08,  
};
```

```

    CY_U3P_USB3_PACK_TYPE_ITP = 0x0C
};

```

Group[USB Constants](#)**Members**

Members	Description
CY_U3P_USB3_PACK_TYPE_LMP = 0x00	Link Management Packet.
CY_U3P_USB3_PACK_TYPE_TP = 0x04	Transaction Packet.
CY_U3P_USB3_PACK_TYPE_DPH = 0x08	Data Packet Header.
CY_U3P_USB3_PACK_TYPE_ITP = 0x0C	Isochronous Timestamp Packet.

File

cyu3usbconst.h

6.1.17 CyU3PUsb3TpSubType

USB 3.0 transaction packet sub type codes.

The following are the various packet sub-types transmitted on super speed operation.

C++

```

enum CyU3PUsb3TpSubType {
    CY_U3P_USB3_TP_SUBTYPE_RES = 0,
    CY_U3P_USB3_TP_SUBTYPE_ACK,
    CY_U3P_USB3_TP_SUBTYPE_NRDY,
    CY_U3P_USB3_TP_SUBTYPE_ERDY,
    CY_U3P_USB3_TP_SUBTYPE_STATUS,
    CY_U3P_USB3_TP_SUBTYPE_STALL,
    CY_U3P_USB3_TP_SUBTYPE_NOTICE,
    CY_U3P_USB3_TP_SUBTYPE_PING,
    CY_U3P_USB3_TP_SUBTYPE_PINGRSP
};

```

Group[USB Constants](#)**Members**

Members	Description
CY_U3P_USB3_TP_SUBTYPE_RES = 0	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.

File

cyu3usbconst.h

6.1.18 CyU3PUsbFeatureSelector

List of USB Feature selector codes.

The following are the various features that can be selected using the SetFeature request or cleared using ClearFeature setup request. Refer to the USB specification for more information.

C++

```
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
};
```

Group[USB Constants](#)**Members**

Members	Description
CY_U3P_USBX_FS_EP_HALT = 0	USB Endpoint HALT feature. Sets or clears EP stall.
CY_U3P_USB2_FS_REMOTE_WAKE = 1	USB 2.0 Remote Wakeup.
CY_U3P_USB2_FS_TEST_MODE = 2	USB 2.0 Test mode.
CY_U3P_USB2_OTG_B_HNP_ENABLE = 3	USB 2.0 OTG HNP enable signal to B-device.
CY_U3P_USB2_OTG_A_HNP_SUPPORT = 4	USB 2.0 OTG HNP supported indication to B-device.
CY_U3P_USB3_FS_U1_ENABLE = 48	USB 3.0 U1 Enable.
CY_U3P_USB3_FS_U2_ENABLE = 49	USB 3.0 U2 Enable.
CY_U3P_USB3_FS_LTM_ENABLE = 50	USB 3.0 LTM Enable.

File

cyu3usbconst.h

6.1.19 CY_U3P_USB_TARGET_MASK

The Target mask

C++

```
#define CY_U3P_USB_TARGET_MASK (0x03) /* The Target mask */
```

Group

[USB Constants](#)

File

cyu3usbconst.h

6.1.20 CY_U3P_USB_TYPE_MASK

The request type mask

C++

```
#define CY_U3P_USB_TYPE_MASK (0x60) /* The request type mask */
```

Group

[USB Constants](#)

File

cyu3usbconst.h

6.1.21 CY_U3P_USB3_TP_DEVADDR_POS

Position of device address field in USB 3.0 LMP and TP.

C++

```
#define CY_U3P_USB3_TP_DEVADDR_POS (25)
```

Group

[USB Constants](#)

File

cyu3usbconst.h

6.1.22 CY_U3P_USB3_TP_EPNUM_POS

Position of endpoint number field in USB 3.0 TP.

C++

```
#define CY_U3P_USB3_TP_EPNUM_POS (8)
```

Group

[USB Constants](#)

File

cyu3usbconst.h

6.1.23 CY_U3P_USB_RESERVED_RQT

The USB reserved request

C++

```
#define CY_U3P_USB_RESERVED_RQT (0x60) /* The USB reserved request */
```

Group

[USB Constants](#)

File

cyu3usbconst.h

6.1.24 CY_U3P_USB_OTG_STATUS_SELECTOR

Index field of the setup request indicating OTG GET_STATUS request.

C++

```
#define CY_U3P_USB_OTG_STATUS_SELECTOR (0xF000)
```

Group

[USB Constants](#)

File

cyu3usbconst.h

6.1.25 CyU3PUsbLinkState_t

Link state machine states.

These are the following link states of interest to firmware.

C++

```
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_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
};
```

Group

[USB Constants](#)

Members

Members	Description
CY_U3P_UIB_LNK_STATE_SSDISABLED = 0x00	SS.Disabled
CY_U3P_UIB_LNK_STATE_RXDETECT_RES = 0x01	Rx.Detect.Reset
CY_U3P_UIB_LNK_STATE_RXDETECT_ACT = 0x02	Rx.Detect.Active
CY_U3P_UIB_LNK_STATE_POLLING_LFPS = 0x08	Polling.LFPS
CY_U3P_UIB_LNK_STATE_POLLING_RxEQ = 0x09	Polling.RxEq
CY_U3P_UIB_LNK_STATE_POLLING_ACT = 0x0A	Polling.Active
CY_U3P_UIB_LNK_STATE_POLLING_IDLE = 0x0C	Polling.Idle
CY_U3P_UIB_LNK_STATE_U0 = 0x10	U0 - Active state
CY_U3P_UIB_LNK_STATE_U1 = 0x11	U1

CY_U3P_UIB_LNK_STATE_U2 = 0x12	U2
CY_U3P_UIB_LNK_STATE_U3 = 0x13	U3 - Suspend state
CY_U3P_UIB_LNK_STATE_COMP = 0x17	Compliance
CY_U3P_UIB_LNK_STATE_RECOV_ACT = 0x18	Recovery.Active
CY_U3P_UIB_LNK_STATE_RECOV_CNFG = 0x19	Recovery.Configuration
CY_U3P_UIB_LNK_STATE_RECOV_IDLE = 0x1A	Recovery.Idle

File

cyu3usbconst.h

6.2 USB OTG Management

The FX3 device supports programmable USB OTG 2.0 implementation which supports device mode operations at USB-SS, USB-HS and USB-FS speeds and host mode operations at USB-HS, USB-FS and USB-LS speeds. The single USB port can function as a device or as a host depending on the type of peripheral attached. It can also perform ACA or Motorola EMU charger detection based on the ID pin state.

Group

[USB Management](#)





Topics

Topic	Description
USB OTG Data Types	This section documents the data types defined and used by the USB OTG mode APIs.
USB OTG Mode Functions	The USB OTG APIs are used to configure the single USB port functionality and peripheral detection.

6.2.1 USB OTG Data Types

This section documents the data types defined and used by the USB OTG mode APIs.



Enumerations

Enumeration	Description
 CyU3POtgMode_t	OTG modes of operation.
 CyU3POtgEvent_t	OTG events.
 CyU3POtgPeripheralType_t	OTG peripheral types.
 CyU3POtgChargerDetectMode_t	Various charger detection mechanism.

Group

[USB OTG Management](#)


Legend

	Enumeration
	Structure

Macros

Macro	Description
CY_U3P_OTG_SRP_MAX_REPEAT_INTERVAL	Maximum value in ms that can be used for the SRP repeat interval.

Structures

Structure	Description
 CyU3POtgConfig_t	OTG configuration information.

Types

Type	Description
CyU3POtgEventCallback_t	OTG event callback function.

6.2.1.1 CY_U3P_OTG_SRP_MAX_REPEAT_INTERVAL

Maximum value in ms that can be used for the SRP repeat interval.

C++

```
#define CY_U3P_OTG_SRP_MAX_REPEAT_INTERVAL (10000)
```

Group

[USB OTG Data Types](#)

File

cyu3usbotg.h

6.2.1.2 CyU3POtgMode_t

OTG modes of operation.

FX3 device has a single USB port which can function in multiple modes. It can act as a USB 2.0 host (no hub support) or as a USB 3.0 device. It can also route the lines to UART 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.

C++

```
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_PPORT,  
    CY_U3P_OTG_MODE_CARKIT_UART,  
    CY_U3P_OTG_NUM_MODES  
};
```

Group

[USB OTG Data Types](#)

See Also

[CyU3POtgConfig_t](#)

Members

Members	Description
CY_U3P_OTG_MODE_DEVICE_ONLY = 0	USB port acts in device only mode. The ID pin value is ignored. Charger detection is disabled.
CY_U3P_OTG_MODE_HOST_ONLY	USB port acts in host only mode. The ID pin value is ignored. Charger detection is disabled.
CY_U3P_OTG_MODE_OTG	USB port acts in OTG mode and identifies the mode of operation based on ID pin. This mode also does 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 / PIB_CTL12 pins. 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.

File

cyu3usbotg.h

6.2.1.3 CyU3POtgEvent_t

OTG events.

The enumeration lists the various OTG events. This is used to identify the type of callback invoked.

C++

```
enum CyU3POtgEvent_t {
    CY_U3P_OTG_PERIPHERAL_CHANGE = 0,
    CY_U3P_OTG_SRP_DETECT,
    CY_U3P_OTG_VBUS_VALID_CHANGE
};
```

Group

[USB OTG Data Types](#)

See Also

[CyU3POtgEventCallback_t](#)

Members

Members	Description
CY_U3P_OTG_PERIPHERAL_CHANGE = 0	The OTG peripheral attached to FX3 has changed / removed or a new peripheral got attached. The parameter to the event callback holds the type of peripheral attached.
CY_U3P_OTG_SRP_DETECT	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 valid and is CyFalse if VBUS is not valid.

File

cyu3usbotg.h

6.2.1.4 CyU3POtgPeripheralType_t

OTG peripheral types.

The enumeration lists describes the various types of OTG peripherals that can be detected using FX3.

C++

```
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
};
```

Group

[USB OTG Data Types](#)

See Also

[CyU3POtgEventCallback_t](#)

Members

Members	Description
CY_U3P_OTG_TYPE_DISABLED = 0	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. Since this is just detecting the state of the ID-pin, it cannot be determined whether remote device has been attached. The FX3 device can either enable the VBUS or wait for an SRP request depending upon the application use case.
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. Since this is just detecting the state of the ID-pin, it cannot be determined whether a remote host is actually connected. The FX3 device is expected to wait for the VBUS to be valid. If this does not happen, then CyU3POtgSrpStart API should 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.

File

cyu3usbotg.h

6.2.1.5 CyU3POtgChargerDetectMode_t

Various charger detection mechanism.

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. Charger detection is based on OTG ID pin and so is available only in CY_U3P_OTG_MODE_OTG mode of operation.

C++

```
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
};
```

Group

[USB OTG Data Types](#)

See Also

[CyU3POtgConfig_t](#)

Members

Members	Description
CY_U3P_OTG_CHARGER_DETECT_ACA_MODE = 0	Charger detection is based on standard ACA charger mode. This is the default mode even when charger detection is not required.
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.

File

cyu3usbotg.h

6.2.1.6 CyU3POtgEventCallback_t

OTG event callback function.

The OTG event callback function returns various OTG events. This includes the OTG mode identification events and charger detect events.

The input to the callback is dependant on the actual event. For CY_U3P_OTG_PERIPHERAL_CHANGE event, this returns the type of peripheral detected ([CyU3POtgPeripheralType_t](#)), for VBUS change interrupts it returns the status of VBUS and for all other events, the input is zero.

C++

```
typedef void (* CyU3POtgEventCallback_t)(CyU3POtgEvent\_t event, uint32_t input);
```

Group

[USB OTG Data Types](#)

See Also

[CyU3POtgEvent_t](#) [CyU3POtgConfig_t](#)

File

cyu3usbotg.h

6.2.1.7 CyU3POtgConfig_t

OTG configuration information.

The structure is given as parameter to the [CyU3POtgStart](#) function. The configuration of the USB port is done based on this.

C++

```
struct CyU3POtgConfig_t {
    CyU3POtgMode\_t otgMode;
    CyU3POtgChargerDetectMode\_t chargerMode;
    CyU3POtgEventCallback\_t cb;
};
```

Group

[USB OTG Data Types](#)

See Also

[CyU3POtgMode_t](#) [CyU3POtgEventCallback_t](#) [CyU3POtgStart](#)

Members

Members	Description
CyU3POtgMode_t otgMode;	USB port mode of operation.
CyU3POtgChargerDetectMode_t chargerMode;	Charger detect mode.
CyU3POtgEventCallback_t cb;	OTG event callback function.

File







cyu3usbotg.h

6.2.2 USB OTG Mode Functions

The USB OTG APIs are used to configure the single USB port functionality and peripheral detection.

Functions


Function	Description
CyU3POtgStart	This function initializes the OTG module.
CyU3POtgStop	This function disables the OTG module.
CyU3POtgGetMode	This function returns the current selected OTG mode.
CyU3POtgGetPeripheralType	This function returns the type of attached USB peripheral.
CyU3POtgIsStarted	This function returns whether the OTG module has been started.
CyU3POtgIsDeviceMode	This function returns whether the device mode of operation is allowed.
CyU3POtgIsHostMode	This function returns whether the host mode of operation is allowed.

 CyU3POtgIsVBusValid	The function checks if a valid VBUS is available.
 CyU3POtgSrpStart	Initiates an SRP request.
 CyU3POtgSrpAbort	Aborts SRP request.
 CyU3POtgHnpEnable	Initiates a role change.
 CyU3POtgIsHnpEnabled	The function checks if a role reversal mode is active or not.
 CyU3POtgRequestHnp	Requests remote host for the HNP process.

Group

[USB OTG Management](#)

Legend

	Method
---	--------

6.2.2.1 CyU3POtgStart

This function initializes the OTG module.

The function initializes the USB block to enable OTG detection. It does not actually start the module operation. Once the mode of operation is identified, 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 UART TX/RX IO lines. These IOs cannot be used for the normal function when the carKit mode is active. Also USB connection cannot be enabled when the carKit mode is active. For CY_U3P_OTG_MODE_CARKIT_PPORT to be used, the PIB_CTL11 / PIB_CTL12 pins should not be used for GPIF configuration, For CY_U3P_OTG_MODE_CARKIT_UART to be used, the LPP UART cannot be used and IO matrix should be configured with useUart as CyFalse. The carKit mode can be disabled by invoking the [CyU3POtgStop](#) API.

Parameters

Parameters	Description
CyU3POtgConfig_t * cfg	OTG configuration information.

Returns

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 any of 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 parameter(s) are invalid.

C++

```
CyU3PReturnStatus_t CyU3POtgStart(  
    CyU3POtgConfig_t * cfg  
) ;
```

Group

[USB OTG Mode Functions](#)

See Also

[CyU3POtgConfig_t](#) [CyU3PUsbStart](#) [CyU3PUsbHostStart](#) [CyU3POtgStop](#) [CyU3POtgIsStarted](#)

File

cyu3usbotg.h

6.2.2.2 CyU3POtgStop

This function disables the OTG module.

The function disables the USB block for OTG detection. It does not actually stop the module operation. The corresponding stop API has to be invoked before invoking this.

Returns

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.

C++

```
CyU3PReturnStatus\_t CyU3POtgStop(  
    void  
);
```

Group

[USB OTG Mode Functions](#)

See Also

[CyU3PUsbStop](#) [CyU3PUsbHostStop](#) [CyU3POtgStart](#) [CyU3POtgIsStarted](#)

File

cyu3usbotg.h

6.2.2.3 CyU3POtgGetMode

This function returns the current selected OTG mode.

This function returns the current configuration as done using the [CyU3POtgStart](#) call.

Returns

[CyU3POtgMode_t](#) - OTG mode selected.

C++

```
CyU3POtgMode\_t CyU3POtgGetMode(  
    void  
);
```

Group

[USB OTG Mode Functions](#)

See Also

[CyU3POtgMode_t](#) [CyU3POtgStart](#)

File

cyu3usbotg.h

6.2.2.4 CyU3POtgGetPeripheralType

This function returns the type of attached USB peripheral.

This function returns the type of USB peripheral attached. The call will return correct value only if there is a valid VBUS or VBATT.

Returns

[CyU3POtgPeripheralType_t](#) - Type of peripheral attached.

C++

```
CyU3POtgPeripheralType\_t CyU3POtgGetPeripheralType(  
    void  
);
```

Group

[USB OTG Mode Functions](#)

See Also

[CyU3POtgPeripheralType_t](#)

File

cyu3usbotg.h

6.2.2.5 CyU3POtgIsStarted

This function returns whether the OTG module has been started.

Since there can be various modes of USB operations this API returns whether [CyU3POtgStart](#) was invoked.

Returns

CyTrue - OTG module started
CyFalse - OTG module stopped or not started

C++

```
CyBool_t CyU3POtgIsStarted(  
    void  
);
```

Group

[USB OTG Mode Functions](#)

See Also

[CyU3POtgStart](#) [CyU3POtgStop](#)

File

cyu3usbotg.h

6.2.2.6 CyU3POtgIsDeviceMode

This function returns whether the device mode of operation is allowed.

The function determines the mode of operation and checks if the device mode of operation can be initiated.

Returns

CyTrue - Device mode of operation allowed. CyFalse - Device mode of operation not allowed.

C++

```
CyBool_t CyU3POtgIsDeviceMode(
    void
);
```

Group

[USB OTG Mode Functions](#)

See Also

[CyU3POtgStart](#) [CyU3PUsbStart](#)

File

cyu3usbotg.h

6.2.2.7 CyU3POtgIsHostMode

This function returns whether the host mode of operation is allowed.

The function determines the mode of operation and checks if the host mode of operation can be initiated.

Returns

CyTrue - Host mode of operation allowed. CyFalse - Host mode of operation not allowed.

C++

```
CyBool_t CyU3POtgIsHostMode(  
    void  
) ;
```

Group

[USB OTG Mode Functions](#)

See Also

[CyU3POtgStart](#) [CyU3PUsbHostStart](#)

File

cyu3usbotg.h

6.2.2.8 CyU3POtgIsVBusValid

The function checks if a valid VBUS is available.

The API can be used to determine the state of the VBUS. Since USB module can function properly only with the VBUS enabled, this can be used to determine when to start the device / host stacks. The VBUS state change can be received through the registered OTG event callback function ([CyU3POtgEventCallback_t](#)) as well.

Returns

CyTrue - VBUS is valid. CyFalse - A valid VBUS is not available.

C++

```
CyBool_t CyU3POtgIsVBusValid(
    void
);
```

Group

[USB OTG Mode Functions](#)

See Also

[CyU3POtgStart](#) [CyU3POtgEventCallback_t](#)

File

cyu3usbotg.h

6.2.2.9 CyU3POtgSrpStart

Initiates an SRP request.

This API is valid when FX3 is in device mode operation. This call will start the SRP request. If there is a remote host, it will respond by enabling the VBUS. When a valid VBUS is detected, the CY_U3P_OTG_VBUS_VALID_CHANGE event is generated. If there is no remote device, SRP will to be repeated periodically until a valid VBUS is detected or the peripheral type changes or CyU3POtgStpAbort API is called. The valid values of repeat period is in range of 500ms to 10s. SRP can be done only when the USB port is in device mode of operation and both device and host stack are disabled.

Parameters

Parameters	Description
uint32_t repeatInterval	Repeat interval in ms. The valid range is from 500ms to 10s.

Returns

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.

C++

```
CyU3PReturnStatus_t CyU3POtgSrpStart(  
    uint32_t repeatInterval  
);
```

Group

[USB OTG Mode Functions](#)

See Also

[CyU3POtgStart](#) [CyU3POtgSrpAbort](#)

File

cyu3usbotg.h

6.2.2.10 CyU3POtgSrpAbort

Aborts SRP request.

Since SRP need to be repeated until a valid VBUS is detected, this call will abort the periodic SRP requests. This might be because no host is detected in the given time period.

Returns

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.

C++

```
CyU3PReturnStatus\_t CyU3POtgSrpAbort(  
    void  
);
```

Group

[USB OTG Mode Functions](#)

See Also

[CyU3POtgStart](#) [CyU3POtgSrpStart](#)

File

cyu3usbotg.h

6.2.2.11 CyU3POtgHnpEnable

Initiates a role change.

This API must be called only when the USB bus is suspended. The input parameter determines whether to enable or disable HNP. This call should be called only when both host and device mode stacks are disabled.

isEnabled = CyTrue: 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.

isEnabled = CyFalse: This API allows the original role to be restored. The previous configuration has to be stopped before invoking this call.

The sequence to 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 requests 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. FX3 should then invoke [CyU3PUsbHostPortSuspend](#) () 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 specified 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 invoke [CyU3PUsbHostStop](#) () to stop the host mode of operation.
8. Enable role change by calling [CyU3POtgHnpEnable](#) (CyTrue).
9. Invoke [CyU3PUsbStart](#) () to start the device mode stack.

The sequence to 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 specified 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 invoking [CyU3PConnectState](#) (CyFalse, CyFalse).
5. Now FX3 should stop the device mode stack by cleaning up all DMA channels and then finally invoking [CyU3PUsbStop](#) ().
6. FX3 should then start the role change by invoking [CyU3POtgHnpEnable](#) (CyTrue).
7. FX3 should then start the host mode operation by invoking [CyU3PUsbHostStart](#) ().

It should be noted that HNP is role reversal and it has to be explicitly disabled. It will get disabled by only on [CyU3POtgStop](#) or [CyU3POtgHnpEnable](#) (CyFalse) calls or or if 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.

Returns

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.

C++

```
CyU3PReturnStatus\_t CyU3POtgHnpEnable(  
    CyBool\_t isEnabled  
);
```

Group

[USB OTG Mode Functions](#)

See Also

[CyU3POtgStart](#) [CyU3PUsbStart](#) [CyU3PUsbStop](#) [CyU3PUsbHostStart](#) [CyU3PUsbHostStop](#) [CyU3POtgRequestHnp](#)

File

[cyu3usbotg.h](#)

6.2.2.12 CyU3POtgIsHnpEnabled

The function checks if a role reversal mode is active or not.

The API just returns the current state of HNP. This call just retrieves the previously set state.

Returns

CyTrue - HNP role change is active. CyFalse - HNP role change is not active.

C++

```
CyBool_t CyU3POtgIsHnpEnabled(  
    void  
);
```

Group

[USB OTG Mode Functions](#)

See Also

[CyU3POtgStart](#) [CyU3POtgEventCallback_t](#)

File

cyu3usbotg.h

6.2.2.13 CyU3POtgRequestHnp

Requests remote host for the HNP process.

This API is valid only in device mode of operation. To initiate a HNP, the device need to set the session request bit in the GetStatus bit. When using normal enumeration model, this needs to be done by the user. Since this is a role change request, the flag is not cleared unless [CyU3PUsbStop](#) or [CyU3POtgRequestHnp](#) (CyFalse) is invoked or if 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.

Parameters

Parameters	Description
CyBool_t isEnabled	Whether to set or clear the host request flag.

Returns

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.

C++

```
CyU3PReturnStatus\_t CyU3POtgRequestHnp(  
    CyBool_t isEnabled  
);
```

Group

[USB OTG Mode Functions](#)

See Also

[CyU3POtgStart](#) [CyU3POtgHnpEnable](#)

File

cyu3usbotg.h

6.3 USB Device Management

The FX3 device can function as a user programmable USB device (or peripheral) at Super Speed, High Speed and Full Speed. The USB device mode driver built into the firmware framework takes care of the connection setup and management at all connection speeds. A set of device mode APIs are provided so that the user application logic can talk to the USB driver to configure the behavior of the device.

Group

[USB Management](#)

Topics

Topic	Description
Enumeration Modes	The 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.
USB Device Data Types	This section documents the data types defined and used by the USB device mode APIs.
USB Device Mode Functions	The USB device APIs are used to configure the USB device functionality and to perform USB data transfers.

6.3.1 Enumeration Modes

The 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.

Two different modes of enumeration control are supported by the USB driver on the FX3 device.

- 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 use 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.

- 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.

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.

Group







[USB Device Management](#)

6.3.2 USB Device Data Types

This section documents the data types defined and used by the USB device mode APIs.

Enumerations



Enumeration	Description
 CyU3PUSBSpeed_t	List of supported USB connection speeds.

 CyU3PUsbEventType_t	List of USB related events that are raised by the USB manager.
 CyU3PUSBSetDescType_t	Virtual descriptor types to be used to set descriptors.
 CyU3PUsbDevProperty	List of USB device properties that can be queried.
 CyU3PUsbLinkPowerMode	List of USB 3.0 link operating modes.
 CyU3PUsbEpEvtType	List of USB endpoint events.
 CyU3PUsbMgrStates_t	List of USB device manager states.

Group

[USB Device Management](#)

Legend


	Enumeration
	Structure

Macros

Macro	Description
CY_U3P_USB_INDEX_MASK	Setup packet has eight bytes and is provided as two 32 bit data in the <ul style="list-style-type: none"> • setup callback. Index is the fifth and sixth bytes of data and is • available in setupdat1 parameter. This definition is the bit mask for • accessing this field.
CY_U3P_USB_INDEX_POS	Setup packet has eight bytes and is provided as two 32 bit data in the <ul style="list-style-type: none"> • setup callback. Index is the fifth and sixth bytes of data and is • available in setupdat1 parameter. This definition is the bit position • for accessing this field.
CY_U3P_USB_LENGTH_MASK	Setup packet has eight bytes and is provided as two 32 bit data in the <ul style="list-style-type: none"> • setup callback. Length is the seventh and eighth bytes of data and is • available in setupdat1 parameter. This definition is the bit mask for • accessing this field.
CY_U3P_USB_LENGTH_POS	Setup packet has eight bytes and is provided as two 32 bit data in the <ul style="list-style-type: none"> • setup callback. Length is the seventh and eighth bytes of data and is • available in setupdat1 parameter. This definition is the bit position • for accessing this field.
CY_U3P_USB_REQUEST_MASK	Setup packet has eight bytes and is provided as two 32 bit data in the <ul style="list-style-type: none"> • setup callback. The Request is the second byte of data and is available • in setupdat0 parameter. This definition is the bit mask for accessing • this field.
CY_U3P_USB_REQUEST_POS	Setup packet has eight bytes and is provided as two 32 bit data in the <ul style="list-style-type: none"> • setup callback. Request is the second byte of data and is available • in setupdat0 parameter. This definition is the bit position for • accessing this field.

CY_U3P_USB_REQUEST_TYPE_MASK	Setup packet has eight bytes and is provided as two 32 bit data in the <ul style="list-style-type: none"> • setup callback. RequestType is the first byte of data and is • available in setupdat0 parameter. This definition is the bit mask for • accessing this field.
CY_U3P_USB_REQUEST_TYPE_POS	Setup packet has eight bytes and is provided as two 32 bit data in the <ul style="list-style-type: none"> • setup callback. RequestType is the first byte of data and is • available in setupdat0 parameter. This definition is the bit position • for accessing this field.
CY_U3P_USB_VALUE_MASK	Setup packet has eight bytes and is provided as two 32 bit data in the <ul style="list-style-type: none"> • setup callback. Value is the third and fourth bytes of data and is • available in setupdat0 parameter. This definition is the bit mask for • accessing this field.
CY_U3P_USB_VALUE_POS	Setup packet has eight bytes and is provided as two 32 bit data in the <ul style="list-style-type: none"> • setup callback. Value is the third and fourth bytes of data and is • available in setupdat0 parameter. This definition is the bit position • for accessing this field.
CY_U3P_MAX_STRING_DESC_INDEX	Number of string descriptors that are supported by the USB driver.

Structures

Structure	Description
 CyU3PEpConfig_t	Endpoint configuration information.

Types

Type	Description
CyU3PUSbEpEvtCb_t	Callback function type used for notification of USB endpoint events.
CyU3PUSbLPMReqCb_t	Event callback raised on host request to switch USB link to a low power state.
CyU3PUSBEvtCb_t	USB event callback type.
CyU3PUSBSetupCb_t	USB setup request handler type.

6.3.2.1 CyU3PUsbEpEvtCb_t

Callback function type used for notification of USB endpoint events.

This callback function type is used by the firmware application to receive notifications of USB endpoint events of interest.

C++

```
typedef void (* CyU3PUsbEpEvtCb_t)(CyU3PUsbEpEvtType evType, CyU3PUSBSpeed\_t
usbSpeed, uint8_t epNum);
```

Group

[USB Device Data Types](#)

See Also

- [CyU3PUsbEpEvtType](#)
- [CyU3PUsbRegisterEpEvtCallback](#)

File

cyu3usb.h

6.3.2.2 CyU3PUsbLPMReqCb_t

Event callback raised on host request to switch USB link to a low power state.

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.

C++

```
typedef CyBool_t (* CyU3PUsbLPMReqCb_t)(CyU3PUsbLinkPowerMode link_mode);
```

Group

[USB Device Data Types](#)

See Also

- [CyU3PUsbLinkPowerMode](#)
- [CyU3PUsbRegisterLPMRequestCallback](#)

File

cyu3usb.h

6.3.2.3 CyU3PUSBSpeed_t

List of supported USB connection speeds.

C++

```
enum CyU3PUSBSpeed_t {
    CY_U3P_NOT_CONNECTED = 0x00,
    CY_U3P_FULL_SPEED,
    CY_U3P_HIGH_SPEED,
    CY_U3P_SUPER_SPEED
};
```

Group

[USB Device Data Types](#)

Members

Members	Description
CY_U3P_NOT_CONNECTED = 0x00	USB device not connected.
CY_U3P_FULL_SPEED	USB full speed.
CY_U3P_HIGH_SPEED	High speed.
CY_U3P_SUPER_SPEED	Super speed.

File

cyu3usb.h

6.3.2.4 CyU3PEpConfig_t

Endpoint configuration information.

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.

C++

```
struct CyU3PEpConfig_t {
    CyBool_t enable;
    CyU3PUsbEpType\_t epType;
    uint16_t streams;
    uint16_t pktSize;
    uint8_t burstLen;
    uint8_t isoPkts;
};
```

Group

[USB Device Data Types](#)

See Also

- [CyU3PSetEpConfig](#)

Members

Members	Description
CyBool_t enable;	Endpoint status - enabled or not.
CyU3PUsbEpType_t epType;	The endpoint type - Isochronous = 1, Bulk = 2, Interrupt = 3. See USB specification for type values.
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.
uint16_t pktSize;	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.
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.
uint8_t isoPkts;	Number of packets per micro-frame for ISO endpoints.

File

cyu3usb.h

6.3.2.5 CyU3PUSBEventCb_t

USB event callback type.

Type of callback function that should be registered with the USB driver to get notifications of USB specific events.

evtype specifies the type of event and evdata is event specific data.

C++

```
typedef void (* CyU3PUSBEventCb_t)(CyU3PUsbEventType\_t evtype, uint16_t evdata);
```

Group

[USB Device Data Types](#)

See Also

- [CyU3PUsbEventType_t](#)
- [CyU3PUsbRegisterEventCallback](#)

File

cyu3usb.h

6.3.2.6 CyU3PUsbEventType_t

List of USB related events that are raised by the USB manager.

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.

C++

```
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_STAT_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
};
```

Group

[USB Device Data Types](#)

See Also

- [CyU3PUSBCb_t](#)
- [CyU3PUsbEnableITPEvent](#)

Members

Members	Description
CY_U3P_USB_EVENT_CONNECT = 0	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 Benicia's USB port.
CY_U3P_USB_EVENT_VBUS_REMOVED	Vbus power removed from Benicia'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_SRP	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.

File

cyu3usb.h

6.3.2.7 CyU3PUSBSetupCb_t

USB setup request handler type.

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.

C++

```
typedef CyBool_t (* CyU3PUSBSetupCb_t)(uint32_t setupdat0, uint32_t setupdat1);
```

Group

[USB Device Data Types](#)

Notes

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](#)

File

cyu3usb.h

6.3.2.8 CyU3PUSBSetDescType_t

Virtual descriptor types to be used to set descriptors.

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.

C++

```
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_OTG_DESCR
};
```

Group

[USB Device Data Types](#)

See Also

- [CyU3PUsbSetDesc](#)

Members

Members	Description
CY_U3P_USB_SET_SS_DEVICE_DESCR	Device descr for SS
CY_U3P_USB_SET_HS_DEVICE_DESCR	Device descr for HS/FS
CY_U3P_USB_SET_DEVQUAL_DESCR	Device Qualifier descriptor
CY_U3P_USB_SET_FS_CONFIG_DESCR	Configuration for FS/HS configuration
CY_U3P_USB_SET_HS_CONFIG_DESCR	Configuration for FS/HS configuration
CY_U3P_USB_SET_STRING_DESCR	String Descriptor
CY_U3P_USB_SET_SS_CONFIG_DESCR	Configuration descr for Super Speed configuration
CY_U3P_USB_SET_SS_BOS_DESCR	BOS descr for Super Speed configuration
CY_U3P_USB_SET_OTG_DESCR	OTG descr for OTG mode operation

File

cyu3usb.h

6.3.2.9 CyU3PUsbDevProperty

List of USB device properties that can be queried.

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.

C++

```
enum CyU3PUsbDevProperty {  
    CY_U3P_USB_PROP_DEVADDR = 0,  
    CY_U3P_USB_PROP_FRAMECNT,  
    CY_U3P_USB_PROP_LINKSTATE,  
    CY_U3P_USB_PROP_ITPINFO,  
    CY_U3P_USB_PROP_SYS_EXIT_LAT  
};
```

Group

[USB Device Data Types](#)

See Also

- [CyU3PUsbGetDevProperty](#)

Members

Members	Description
CY_U3P_USB_PROP_DEVADDR = 0	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.

File

cyu3usb.h

6.3.2.10 CyU3PUsbLinkPowerMode

List of USB 3.0 link operating modes.

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.

C++

```
enum CyU3PUsbLinkPowerMode {
    CyU3PUsbLPM_U0 = 0,
    CyU3PUsbLPM_U1,
    CyU3PUsbLPM_U2,
    CyU3PUsbLPM_U3,
    CyU3PUsbLPM_COMP,
    CyU3PUsbLPM_Unknown
};
```

Group

[USB Device Data Types](#)

See Also

- [CyU3PUsbLPMReqCb_t](#)

Members

Members	Description
CyU3PUsbLPM_U0 = 0	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.

File

cyu3usb.h

6.3.2.11 CY_U3P_USB_INDEX_MASK

Setup packet has eight bytes and is provided as two 32 bit data in the

- setup callback. Index is the fifth and sixth bytes of data and is
- available in setupdat1 parameter. This definition is the bit mask for
- accessing this field.

C++

```
#define CY_U3P_USB_INDEX_MASK (0x0000FFFF)
```

Group

[USB Device Data Types](#)

File

cyu3usb.h

6.3.2.12 CY_U3P_USB_INDEX_POS

Setup packet has eight bytes and is provided as two 32 bit data in the

- setup callback. Index is the fifth and sixth bytes of data and is
- available in setupdat1 parameter. This definition is the bit position
- for accessing this field.

C++

```
#define CY_U3P_USB_INDEX_POS (0)
```

Group

[USB Device Data Types](#)

File

cyu3usb.h

6.3.2.13 CY_U3P_USB_LENGTH_MASK

Setup packet has eight bytes and is provided as two 32 bit data in the

- setup callback. Length is the seventh and eighth bytes of data and is
- available in setupdat1 parameter. This definition is the bit mask for
- accessing this field.

C++

```
#define CY_U3P_USB_LENGTH_MASK (0xFFFF0000)
```

Group

[USB Device Data Types](#)

File

cyu3usb.h

6.3.2.14 CY_U3P_USB_LENGTH_POS

Setup packet has eight bytes and is provided as two 32 bit data in the

- setup callback. Length is the seventh and eighth bytes of data and is
- available in setupdat1 parameter. This definition is the bit position
- for accessing this field.

C++

```
#define CY_U3P_USB_LENGTH_POS (16)
```

Group

[USB Device Data Types](#)

File

cyu3usb.h

6.3.2.15 CY_U3P_USB_REQUEST_MASK

Setup packet has eight bytes and is provided as two 32 bit data in the

- setup callback. The Request is the second byte of data and is available
- in setupdat0 parameter. This definition is the bit mask for accessing
- this field.

C++

```
#define CY_U3P_USB_REQUEST_MASK (0x0000FF00)
```

Group

[USB Device Data Types](#)

File

cyu3usb.h

6.3.2.16 CY_U3P_USB_REQUEST_POS

Setup packet has eight bytes and is provided as two 32 bit data in the

- setup callback. Request is the second byte of data and is available
- in setupdat0 parameter. This definition is the bit position for
- accessing this field.

C++

```
#define CY_U3P_USB_REQUEST_POS (8)
```

Group

[USB Device Data Types](#)

File

cyu3usb.h

6.3.2.17 CY_U3P_USB_REQUEST_TYPE_MASK

Setup packet has eight bytes and is provided as two 32 bit data in the

- setup callback. RequestType is the first byte of data and is
- available in setupdat0 parameter. This definition is the bit mask for
- accessing this field.

C++

```
#define CY_U3P_USB_REQUEST_TYPE_MASK (0x000000FF)
```

Group

[USB Device Data Types](#)

File

cyu3usb.h

6.3.2.18 CY_U3P_USB_REQUEST_TYPE_POS

Setup packet has eight bytes and is provided as two 32 bit data in the

- setup callback. RequestType is the first byte of data and is
- available in setupdat0 parameter. This definition is the bit position
- for accessing this field.

C++

```
#define CY_U3P_USB_REQUEST_TYPE_POS (0)
```

Group

[USB Device Data Types](#)

File

cyu3usb.h

6.3.2.19 CY_U3P_USB_VALUE_MASK

Setup packet has eight bytes and is provided as two 32 bit data in the

- setup callback. Value is the third and fourth bytes of data and is
- available in setupdat0 parameter. This definition is the bit mask for
- accessing this field.

C++

```
#define CY_U3P_USB_VALUE_MASK (0xFFFF0000)
```

Group

[USB Device Data Types](#)

File

cyu3usb.h

6.3.2.20 CY_U3P_USB_VALUE_POS

Setup packet has eight bytes and is provided as two 32 bit data in the

- setup callback. Value is the third and fourth bytes of data and is
- available in setupdat0 parameter. This definition is the bit position
- for accessing this field.

C++

```
#define CY_U3P_USB_VALUE_POS (16)
```

Group

[USB Device Data Types](#)

File

cyu3usb.h

6.3.2.21 CY_U3P_MAX_STRING_DESC_INDEX

Number of string descriptors that are supported by the USB driver.

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.

C++

```
#define CY_U3P_MAX_STRING_DESC_INDEX (16)
```

Group

[USB Device Data Types](#)

See Also

- [CyU3PUsbSetDesc](#)

File

cyu3usb.h

6.3.2.22 CyU3PUsbEpEvtType

List of USB endpoint events.

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.

C++

```
enum CyU3PUsbEpEvtType {
    CYU3P_USBEP_NAK_EVT = 1,
    CYU3P_USBEP_ZLP_EVT = 2,
    CYU3P_USBEP_SLP_EVT = 4,
    CYU3P_USBEP_ISOERR_EVT = 8,
    CYU3P_USBEP_SS_RETRY_EVT = 16,
    CYU3P_USBEP_SS_SEQERR_EVT = 32,
    CYU3P_USBEP_SS_STREAMERR_EVT = 64
};
```

Group

[USB Device Data Types](#)

See Also

- [CyU3PUsbEpEvtCb_t](#)
- [CyU3PUsbRegisterEpEvtCallback](#)

Members

Members	Description
CYU3P_USBEP_NAK_EVT = 1	The endpoint sent NAK/NRDY in response to a host transfer request.
CYU3P_USBEP_ZLP_EVT = 2	A Zero Length Packet was sent/received on the endpoint.
CYU3P_USBEP_SLP_EVT = 4	A Short Length Packet was sent/received on the endpoint.
CYU3P_USBEP_ISOERR_EVT = 8	Out of sequence ISO PIDs or ZLP received on ISO endpoint.
CYU3P_USBEP_SS_RETRY_EVT = 16	An ACK TP with the RETRY bit was received on the IN endpoint.
CYU3P_USBEP_SS_SEQERR_EVT = 32	Sequence number error detected during endpoint data transfer.
CYU3P_USBEP_SS_STREAMERR_EVT = 64	A Bulk-Stream protocol error occurred on the endpoint.

File

cyu3usb.h

6.3.2.23 CyU3PUsbMgrStates_t

List of USB device manager states.

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.

C++

```
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  
};
```

Group

[USB Device Data Types](#)

Members

Members	Description
CY_U3P_USB_INACTIVE = 0	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.

File

cyu3usb.h

6.3.3 USB Device Mode Functions

The USB device APIs are used to configure the USB device functionality and to perform USB data transfers.












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.

This section documents the USB device mode functions that are provided to configure, control and use USB device mode connections through the device.

Functions

Function	Description
CyU3PUsbStart	Start the USB driver.
CyU3PUsbStop	Stop the USB driver.
CyU3PUsbIsStarted	This function returns whether the USB device module has been started.
CyU3PUsbRegisterEventCallback	Register a USB event callback function.
CyU3PUsbRegisterSetupCallback	Register a USB setup request handler.
CyU3PUsbRegisterEpEvtCallback	Register a callback function for notification of USB endpoint events.
CyU3PUsbRegisterLPMRequestCallback	Register a USB 3.0 LPM request handler callback.
CyU3PUsbSetDesc	Register a USB descriptor with the driver.
CyU3PSetEpConfig	Configure a USB endpoint's properties.
CyU3PSetEpPacketSize	Set the maximum packet size for a USB endpoint.
CyU3PUsbSetEpNak	Force or clear the NAK status on the specified endpoint.
CyU3PUsbResetEp	Resets the specified endpoint.
CyU3PConnectState	Enable / disable the USB connection.
CyU3PGetConnectState	Check whether the FX3 device is currently connected to a host.
CyU3PUsbGetSpeed	Get the connection speed at which USB is operating.
CyU3PUsbStall	Set or clear the stall status of an endpoint.
CyU3PUsbGetEpCfg	Retrieve the current state of the specified endpoint.
CyU3PUsbAckSetup	Complete the status handshake of a USB control request.
CyU3PUsbSendEP0Data	Send data to the USB host via EP0-IN.
CyU3PUsbGetEP0Data	Read data associated with a control-OUT transfer.
CyU3PUsbFlushEp	Clear (flush) all data buffers associated with a specified endpoint.
CyU3PUsbVBattEnable	Configure USB block on FX3 to work off Vbatt power instead of Vbus.
CyU3PUsbDoRemoteWakeup	Trigger USB 2.0 Remote Wakeup signalling to the host.
CyU3PUsbMapStream	Map a socket to stream of the specified endpoint.
CyU3PUsbChangeMapping	Map a socket to stream of the specified endpoint.
CyU3PUsbSendErDy	Function to send an ERDY TP to a USB 3.0 host.
CyU3PUsbSendNrdy	Function to send an NRDY TP to a USB 3.0 host.
CyU3PUsbGetDevProperty	Function that returns some properties of the USB device.
CyU3PUsbGetLinkPowerState	Function to get the current power state of the USB 3.0 link.
CyU3PUsbSetLinkPowerState	Function to request a device entry into one of the U0, U1 or U2 link power modes.
CyU3PUsbEnableITPEvent	Function to enable/disable notification of SOF/ITP events to the application.
CyU3PUsbSendDevNotification	Function to send a DEV_NOTIFICATION Transaction Packet to the host.
CyU3PUsbForceFullSpeed	Function to force the USB 2.0 device connection to full speed.

 CyU3PUsbLPMDisable	Function to prevent FX3 from entering U1/U2 states.
 CyU3PUsbLPMEnable	Function to re-enable automated handling of U1/U2 state entry.
 CyU3PUsbInitEventLog	Function to initiate logging of USB state changes into a circular buffer.
 CyU3PUsbGetEventLogIndex	Get the current event log buffer index.
 CyU3PUsbSetBooterSwitch	Function to enable/disable switching control back to the FX3 2-stage bootloader.
 CyU3PUsbJumpBackToBooter	Function to transfer control back to the FX3 2-stage bootloader.
 CyU3PUsbEpPrepare	Prepare all enabled USB device endpoints for data transfer.
 CyU3PUsbEnableEPPrefetch	Configure the USB DMA interface to continuously pre-fetch data.
 CyU3PUsbResetEndpointMemories	Reset and re-initialize the endpoint memory block on FX3.
 CyU3PUsbGetErrorCounts	Function to get the number of USB 3.0 PHY and LINK error counts detected by FX3.
 CyU3PUsbSetEpPktMode	Select whether a OUT endpoint will function in packet (one buffer per packet) mode or not.

Group

[USB Device Management](#)

Legend

	Method
---	--------

6.3.3.1 CyU3PUsbStart

Start the USB driver.

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.

Returns

- 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) fail
- CY_U3P_ERROR_NO_REENUM_REQUIRED - FX3 Booter supports the No ReEnumeration 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 No Re-Enumeration is successful and the sequence followed for the regular enumeration post CyU3PUsbStart () is to be skipped.

C++

```
CyU3PReturnStatus\_t CyU3PUsbStart(  
    void  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbStop](#)
- [CyU3PUsbIsStarted](#)
- [CyU3PUsbVBattEnable](#)
- [CyU3PUsbRegisterEventCallback](#)
- [CyU3PUsbRegisterSetupCallback](#)
- [CyU3PUsbRegisterEpEvtCallback](#)
- [CyU3PUsbRegisterLPMRequestCallback](#)
- [CyU3PUsbSetDesc](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)
- [CyU3PSetEpConfig](#)
- [CyU3PSetEpPacketSize](#)
- [CyU3PUsbGetEpCfg](#)
- [CyU3PUsbGetSpeed](#)
- [CyU3PUsbSendEP0Data](#)
- [CyU3PUsbGetEP0Data](#)
- [CyU3PUsbSetEpNak](#)
- [CyU3PUsbAckSetup](#)
- [CyU3PUsbFlushEp](#)

- [CyU3PUsbResetEp](#)
- [CyU3PUsbStall](#)
- [CyU3PUsbMapStream](#)
- [CyU3PUsbChangeMapping](#)
- [CyU3PUsbSendErDy](#)
- [CyU3PUsbSendNrDy](#)
- [CyU3PUsbGetLinkPowerState](#)
- [CyU3PUsbSetLinkPowerState](#)
- [CyU3PUsbLinkPowerMode](#)
- [CyU3PUsbGetDevProperty](#)
- [CyU3PUsbDevProperty](#)

File

cyu3usb.h

6.3.3.2 CyU3PUsbStop

Stop the USB driver.

This function stops the USB driver on the FX3 device. It also frees up the DMA channels created for the control endpoint.

Returns

- 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 EPO DMA channel destroy fails

C++

```
CyU3PReturnStatus\_t CyU3PUsbStop(  
    void  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbIsStarted](#)
- [CyU3PUsbVBattEnable](#)
- [CyU3PUsbRegisterEventCallback](#)
- [CyU3PUsbRegisterSetupCallback](#)
- [CyU3PUsbRegisterEpEvtCallback](#)
- [CyU3PUsbRegisterLPMRequestCallback](#)
- [CyU3PUsbSetDesc](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)
- [CyU3PSetEpConfig](#)
- [CyU3PSetEpPacketSize](#)
- [CyU3PUsbGetEpCfg](#)
- [CyU3PUsbGetSpeed](#)
- [CyU3PUsbSendEP0Data](#)
- [CyU3PUsbGetEP0Data](#)
- [CyU3PUsbSetEpNak](#)
- [CyU3PUsbAckSetup](#)
- [CyU3PUsbFlushEp](#)
- [CyU3PUsbResetEp](#)
- [CyU3PUsbStall](#)
- [CyU3PUsbMapStream](#)

- [CyU3PUsbChangeMapping](#)
- [CyU3PUsbSendErly](#)
- [CyU3PUsbSendNrly](#)
- [CyU3PUsbGetLinkPowerState](#)
- [CyU3PUsbSetLinkPowerState](#)
- [CyU3PUsbLinkPowerMode](#)
- [CyU3PUsbGetDevProperty](#)
- [CyU3PUsbDevProperty](#)

File

cyu3usb.h

6.3.3.3 CyU3PUsbIsStarted

This function returns whether the USB device module has been started.

Since there can be various modes of USB operations this API returns whether [CyU3PUsbStart](#) was invoked.

Returns

CyTrue - USB device module started
CyFalse - USB device module stopped or not started

C++

```
CyBool_t CyU3PUsbIsStarted(  
    void  
);
```

Group

[USB Device Mode Functions](#)

See Also

[CyU3PUsbStart](#) [CyU3PUsbStop](#)

File

cyu3usb.h

6.3.3.4 CyU3PUsbRegisterEventCallback

Register a USB event callback function.

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.

Parameters

Parameters	Description
CyU3PUSBEventCb_t callback	Event callback function pointer.

Returns

None

C++

```
void CyU3PUsbRegisterEventCallback(  
    CyU3PUSBEventCb\_t callback  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUSBEventCb_t](#)
- [CyU3PUsbEventType_t](#)
- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PUsbRegisterSetupCallback](#)
- [CyU3PUsbRegisterEpEvtCallback](#)
- [CyU3PUsbRegisterLPMRequestCallback](#)
- [CyU3PUsbSetDesc](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)

File

[cyu3usb.h](#)

6.3.3.5 CyU3PUsbRegisterSetupCallback

Register a USB setup request handler.

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.

Parameters

Parameters	Description
CyU3PUSBSetupCb_t callback	Setup request handler function.
CyBool_t fastEnum	Select fast enumeration mode by setting CyTrue.

Returns

None

C++

```
void CyU3PUsbRegisterSetupCallback(  
    CyU3PUSBSetupCb\_t callback,  
    CyBool_t fastEnum  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUSBSetupCb_t](#)
- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PUsbRegisterEventCallback](#)
- [CyU3PUsbRegisterEpEvtCallback](#)
- [CyU3PUsbRegisterLPMRequestCallback](#)
- [CyU3PUsbSetDesc](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)

File

cyu3usb.h

6.3.3.6 CyU3PUsbRegisterEpEvtCallback

Register a callback function for notification of USB endpoint events.

This function is used to register a callback function that will be invoked for notification of USB endpoint events during device operation.

Parameters

Parameters	Description
CyU3PUsbEpEvtCb_t cbFunc	Callback function pointer.
uint32_t eventMask	Bitmap variable representing the events that should be enabled.
uint16_t outEpMask	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.
uint16_t inEpMask	Bitmap variable representing the IN endpoints whose events are to be enabled.

Returns

None

C++

```
void CyU3PUsbRegisterEpEvtCallback(  
    CyU3PUsbEpEvtCb\_t cbFunc,  
    uint32_t eventMask,  
    uint16_t outEpMask,  
    uint16_t inEpMask  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbEpEvtCb_t](#)
- [CyU3PUsbEpEvtType](#)
- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PUsbRegisterEventCallback](#)
- [CyU3PUsbRegisterSetupCallback](#)
- [CyU3PUsbRegisterLPMRequestCallback](#)
- [CyU3PUsbSetDesc](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)
- [CyU3PSetEpConfig](#)
- [CyU3PUsbGetEpCfg](#)
- [CyU3PUsbSetEpNak](#)
- [CyU3PUsbAckSetup](#)

- [CyU3PUsbFlushEp](#)
- [CyU3PUsbResetEp](#)
- [CyU3PUsbStall](#)
- [CyU3PUsbMapStream](#)
- [CyU3PUsbChangeMapping](#)
- [CyU3PUsbSendErdy](#)
- [CyU3PUsbSendNrdy](#)

File

cyu3usb.h

6.3.3.7 CyU3PUsbRegisterLPMRequestCallback

Register a USB 3.0 LPM request handler callback.

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.

Parameters

Parameters	Description
CyU3PUsbLPMReqCb_t cb	Callback function pointer.

Returns

None

C++

```
void CyU3PUsbRegisterLPMRequestCallback(  
    CyU3PUsbLPMReqCb\_t cb  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbLPMReqCb_t](#)
- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PUsbRegisterEventCallback](#)
- [CyU3PUsbRegisterSetupCallback](#)
- [CyU3PUsbRegisterEpEvtCallback](#)
- [CyU3PUsbSetDesc](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)
- [CyU3PUsbGetLinkPowerState](#)
- [CyU3PUsbSetLinkPowerState](#)
- [CyU3PUsbLinkPowerMode](#)
- [CyU3PUsbGetDevProperty](#)
- [CyU3PUsbDevProperty](#)

File

cyu3usb.h

6.3.3.8 CyU3PUsbSetDesc

Register a USB descriptor with the driver.

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 upto 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.

Parameters

Parameters	Description
CyU3PUSBSetDescType_t desc_type	Type of descriptor to register.
uint8_t desc_index	Descriptor index: Used only for string descriptors.
uint8_t* desc	Pointer to buffer containing the descriptor.

Returns

- 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 from [CyU3PUSBSetDescType_t](#)
- CY_U3P_ERROR_BAD_INDEX - if the string descriptor index exceeds 16

C++

```
CyU3PReturnStatus_t CyU3PUsbSetDesc(
    CyU3PUSBSetDescType_t desc_type,
    uint8_t desc_index,
    uint8_t* desc
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUSBSetDescType_t](#)
- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PUsbRegisterEventCallback](#)
- [CyU3PUsbRegisterSetupCallback](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)

File

cyu3usb.h

6.3.3.9 CyU3PSetEpConfig

Configure a USB endpoint's properties.

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.

Parameters

Parameters	Description
uint8_t ep	Endpoint number to be configured.
CyU3PEpConfig_t * epinfo	Properties to associate with the endpoint.

Returns

- 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 the isoPkts(mult) field is specified as non-zero for endpoints other than 3 & 7.

C++

```
CyU3PReturnStatus_t CyU3PSetEpConfig(
    uint8_t ep,
    CyU3PEpConfig_t * epinfo
);
```

Group

[USB Device Mode Functions](#)

Notes

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.

See Also

- [CyU3PEpConfig_t](#)
- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)

- [CyU3PSetEpPacketSize](#)
- [CyU3PUsbGetEpCfg](#)
- [CyU3PUsbSendEP0Data](#)
- [CyU3PUsbGetEP0Data](#)
- [CyU3PUsbSetEpNak](#)
- [CyU3PUsbAckSetup](#)
- [CyU3PUsbFlushEp](#)
- [CyU3PUsbResetEp](#)
- [CyU3PUsbStall](#)
- [CyU3PUsbMapStream](#)
- [CyU3PUsbChangeMapping](#)
- [CyU3PUsbSendErDy](#)
- [CyU3PUsbSendNrDy](#)

File

cyu3usb.h

6.3.3.10 CyU3PSetEpPacketSize

Set the maximum packet size for a USB endpoint.

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.

Parameters

Parameters	Description
uint8_t ep	Endpoint number to be configured.
uint16_t maxPktSize	Maximum packet size for endpoint in bytes.

Returns

- CY_U3P_SUCCESS - when the call is successful
- CY_U3P_ERROR_BAD_ARGUMENT - if the endpoint or size specified is invalid.

C++

```
CyU3PReturnStatus_t CyU3PSetEpPacketSize(  
    uint8_t ep,  
    uint16_t maxPktSize  
);
```

Group

[USB Device Mode Functions](#)

Notes

Please note that the maximum packet size will be automatically adjusted to the connection speed when the USB 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.

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PSetEpConfig](#)
- [CyU3PUsbSetEpPktMode](#)
- [CyU3PUsbGetEpCfg](#)
- [CyU3PUsbGetSpeed](#)

- [CyU3PUsbSendEP0Data](#)
- [CyU3PUsbGetEP0Data](#)
- [CyU3PUsbSetEpNak](#)
- [CyU3PUsbAckSetup](#)
- [CyU3PUsbFlushEp](#)
- [CyU3PUsbResetEp](#)
- [CyU3PUsbStall](#)
- [CyU3PUsbMapStream](#)
- [CyU3PUsbChangeMapping](#)
- [CyU3PUsbSendErdy](#)
- [CyU3PUsbSendNrdy](#)

File

cyu3usb.h

6.3.3.11 CyU3PUsbSetEpNak

Force or clear the NAK status on the specified endpoint.

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.

Parameters

Parameters	Description
uint8_t ep	Endpoint to be updated.
CyBool_t nak	CyTrue to force NAK, CyFalse to clear NAK.

Returns

- 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.

C++

```
CyU3PReturnStatus\_t CyU3PUsbSetEpNak(
    uint8_t ep,
    CyBool_t nak
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PSetEpConfig](#)
- [CyU3PSetEpPacketSize](#)
- [CyU3PUsbGetEpCfg](#)
- [CyU3PUsbSendEP0Data](#)
- [CyU3PUsbGetEP0Data](#)
- [CyU3PUsbAckSetup](#)
- [CyU3PUsbFlushEp](#)
- [CyU3PUsbResetEp](#)
- [CyU3PUsbStall](#)
- [CyU3PUsbMapStream](#)
- [CyU3PUsbChangeMapping](#)
- [CyU3PUsbSendErdy](#)
- [CyU3PUsbSendNrdy](#)

File

cyu3usb.h

6.3.3.12 CyU3PUsbResetEp

Resets the specified endpoint.

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.

Parameters

Parameters	Description
uint8_t ep	Endpoint number to be cleared.

Returns

- CY_U3P_SUCCESS - when the call is successful
- CY_U3P_ERROR_BAD_ARGUMENT - when the endpoint number is invalid

C++

```
CyU3PReturnStatus\_t CyU3PUsbResetEp(
    uint8_t ep
);
```

Group

[USB Device Mode Functions](#)

Notes

This function does nothing if called at high/full speed.

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PSetEpConfig](#)
- [CyU3PSetEpPacketSize](#)
- [CyU3PUsbGetEpCfg](#)
- [CyU3PUsbGetSpeed](#)
- [CyU3PUsbSetEpNak](#)
- [CyU3PUsbFlushEp](#)
- [CyU3PUsbStall](#)
- [CyU3PUsbMapStream](#)
- [CyU3PUsbChangeMapping](#)
- [CyU3PUsbSendErdy](#)
- [CyU3PUsbSendNrdy](#)

File

cyu3usb.h

6.3.3.13 CyU3PConnectState

Enable / disable the USB connection.

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.

Parameters

Parameters	Description
CyBool_t connect	CyTrue to enable connection, CyFalse to disable connection.
CyBool_t ssEnable	Whether to enumerate as a SS device or FS/HS device

Returns

- 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.

C++

```
CyU3PReturnStatus_t CyU3PConnectState(
    CyBool_t connect,
    CyBool_t ssEnable
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PUsbSetDesc](#)
- [CyU3PGetConnectState](#)

File

cyu3usb.h

6.3.3.14 CyU3PGetConnectState

Check whether the FX3 device is currently connected to a host.

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.

Returns

CyTrue: VBUS is detected; CyFalse: No VBUS detected.

C++

```
CyBool_t CyU3PGetConnectState(  
    void  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PUsbRegisterEventCallback](#)
- [CyU3PUsbRegisterSetupCallback](#)
- [CyU3PUsbRegisterEpEvtCallback](#)
- [CyU3PUsbRegisterLPMRequestCallback](#)
- [CyU3PUsbSetDesc](#)
- [CyU3PConnectState](#)

File

cyu3usb.h

6.3.3.15 CyU3PUsbGetSpeed

Get the connection speed at which USB is operating.

This function is used to get the operating speed of the USB connection.

Returns

- [CyU3PUSBSpeed_t](#)

C++

```
CyU3PUSBSpeed\_t CyU3PUsbGetSpeed(  
    void  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUSBSpeed_t](#)
- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PUsbRegisterEventCallback](#)
- [CyU3PUsbRegisterSetupCallback](#)
- [CyU3PUsbRegisterEpEvtCallback](#)
- [CyU3PUsbRegisterLPMRequestCallback](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)

File

cyu3usb.h

6.3.3.16 CyU3PUsbStall

Set or clear the stall status of an endpoint.

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.

Parameters

Parameters	Description
uint8_t ep	Endpoint number to be modified.
CyBool_t stall	CyTrue: Set the stall condition, CyFalse: Clear the stall
CyBool_t toggle	CyTrue: Clear the data toggles in a Clear Stall call

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PUsbStall(  
    uint8_t ep,  
    CyBool_t stall,  
    CyBool_t toggle  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)
- [CyU3PSetEpConfig](#)
- [CyU3PSetEpPacketSize](#)
- [CyU3PUsbGetEpCfg](#)
- [CyU3PUsbSendEP0Data](#)
- [CyU3PUsbGetEP0Data](#)
- [CyU3PUsbSetEpNak](#)
- [CyU3PUsbAckSetup](#)
- [CyU3PUsbFlushEp](#)

- [CyU3PUsbResetEp](#)
- [CyU3PUsbMapStream](#)
- [CyU3PUsbChangeMapping](#)
- [CyU3PUsbSendErdy](#)
- [CyU3PUsbSendNrdy](#)

File

cyu3usb.h

6.3.3.17 CyU3PUsbGetEpCfg

Retrieve the current state of the specified endpoint.

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.

Parameters

Parameters	Description
uint8_t ep	Endpoint number to query.
CyBool_t * isNak	Return parameter which will be filled with the NAK status.
CyBool_t * isStall	Return parameter which will be filled with the STALL status.

Returns

- 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.

C++

```
CyU3PReturnStatus_t CyU3PUsbGetEpCfg(  
    uint8_t ep,  
    CyBool_t * isNak,  
    CyBool_t * isStall  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PUsbSetDesc](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)
- [CyU3PSetEpPacketSize](#)
- [CyU3PSetEpConfig](#)
- [CyU3PUsbSendEP0Data](#)
- [CyU3PUsbGetEP0Data](#)
- [CyU3PUsbSetEpNak](#)
- [CyU3PUsbFlushEp](#)
- [CyU3PUsbResetEp](#)
- [CyU3PUsbStall](#)
- [CyU3PUsbMapStream](#)

- [CyU3PUsbChangeMapping](#)
- [CyU3PUsbSendErdy](#)
- [CyU3PUsbSendNrdy](#)

File

cyu3usb.h

6.3.3.18 CyU3PUsbAckSetup

Complete the status handshake of a USB control request.

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.

Returns

None

C++

```
void CyU3PUsbAckSetup(  
    void  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PUsbRegisterEventCallback](#)
- [CyU3PUsbRegisterSetupCallback](#)
- [CyU3PUsbSetDesc](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)
- [CyU3PUsbSendEP0Data](#)
- [CyU3PUsbGetEP0Data](#)

File

cyu3usb.h

6.3.3.19 CyU3PUsbSendEP0Data

Send data to the USB host via EP0-IN.

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.

Parameters

Parameters	Description
uint16_t count	The amount of data in bytes.
uint8_t * buffer	Pointer to buffer containing the data.

Returns

- 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, CY_U3P_ERROR_ABORTED, CY_U3P_ERROR_DMA_FAILURE, CY_U3P_ERROR_DMA_FAILURE, CY_U3P_ERROR_ALREADY_STARTED, CY_U3P_ERROR_NOT_SUPPORTED - returned in case of DMA failure (EP0 DMA channel)

C++

```
CyU3PReturnStatus\_t CyU3PUsbSendEP0Data(
    uint16_t count,
    uint8_t * buffer
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PUsbRegisterEventCallback](#)
- [CyU3PUsbRegisterSetupCallback](#)
- [CyU3PUsbRegisterEpEvtCallback](#)
- [CyU3PUsbSetDesc](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)
- [CyU3PUsbGetEP0Data](#)

- [CyU3PUsbAckSetup](#)
- [CyU3PUsbFlushEp](#)
- [CyU3PUsbResetEp](#)
- [CyU3PUsbStall](#)

File

cyu3usb.h

6.3.3.20 CyU3PUsbGetEP0Data

Read data associated with a control-OUT transfer.

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.

Parameters

Parameters	Description
uint16_t count	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).
uint8_t * buffer	Pointer to buffer where the data should be placed.
uint16_t * readCount	Output parameter which will be filled with the actual size of data read.

Returns

- 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, size is not aligned or a DMA failure occurs
- CY_U3P_XFER_CANCELLED - is returned if the transaction has already been cancelled
- CY_U3P_ERROR_TIMEOUT, CY_U3P_ERROR_ABORTED, CY_U3P_ERROR_DMA_FAILURE, CY_U3P_ERROR_DMA_FAILURE, CY_U3P_ERROR_ALREADY_STARTED, CY_U3P_ERROR_NOT_SUPPORTED - returned in case of DMA failure (EP0 DMA channel)

C++

```
CyU3PReturnStatus\_t CyU3PUsbGetEP0Data(
    uint16_t count,
    uint8_t * buffer,
    uint16_t * readCount
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)

- [CyU3PUsbRegisterEventCallback](#)
- [CyU3PUsbRegisterSetupCallback](#)
- [CyU3PUsbRegisterEpEvtCallback](#)
- [CyU3PUsbSetDesc](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)
- [CyU3PUsbGetSpeed](#)
- [CyU3PUsbSendEP0Data](#)
- [CyU3PUsbAckSetup](#)
- [CyU3PUsbFlushEp](#)
- [CyU3PUsbResetEp](#)
- [CyU3PUsbStall](#)

File

cyu3usb.h

6.3.3.21 CyU3PUsbFlushEp

Clear (flush) all data buffers associated with a specified endpoint.

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.

Parameters

Parameters	Description
uint8_t ep	Endpoint number to be cleared.

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PUsbFlushEp(  
    uint8_t ep  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PSetEpConfig](#)
- [CyU3PSetEpPacketSize](#)
- [CyU3PUsbGetEpCfg](#)
- [CyU3PUsbGetSpeed](#)
- [CyU3PUsbSetEpNak](#)
- [CyU3PUsbAckSetup](#)
- [CyU3PUsbResetEp](#)
- [CyU3PUsbStall](#)
- [CyU3PUsbMapStream](#)
- [CyU3PUsbChangeMapping](#)
- [CyU3PUsbSendErdy](#)
- [CyU3PUsbSendNrdy](#)
- [CyU3PUsbGetLinkPowerState](#)
- [CyU3PUsbSetLinkPowerState](#)
- [CyU3PUsbLinkPowerMode](#)

- [CyU3PUsbGetDevProperty](#)
- [CyU3PUsbDevProperty](#)

File

cyu3usb.h

6.3.3.22 CyU3PUsbVBattEnable

Configure USB block on FX3 to work off Vbatt power instead of Vbus.

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.

Parameters

Parameters	Description
CyBool_t enable	CyTrue:Work off Vbatt, CyFalse: Do not work off VBatt

Returns

- 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.

C++

```
CyU3PReturnStatus\_t CyU3PUsbVBattEnable(  
    CyBool_t enable  
);
```

Group

[USB Device Mode Functions](#)

Notes

This API is expected to be called before the [CyU3PConnectState](#) API is called.

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PUsbRegisterEventCallback](#)
- [CyU3PUsbRegisterSetupCallback](#)
- [CyU3PUsbRegisterEpEvtCallback](#)
- [CyU3PUsbRegisterLPMRequestCallback](#)
- [CyU3PUsbSetDesc](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)
- [CyU3PUsbGetLinkPowerState](#)
- [CyU3PUsbSetLinkPowerState](#)
- [CyU3PUsbLinkPowerMode](#)
- [CyU3PUsbGetDevProperty](#)
- [CyU3PUsbDevProperty](#)

File

cyu3usb.h

6.3.3.23 CyU3PUsbDoRemoteWakeup

Trigger USB 2.0 Remote Wakeup signalling to the host.

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.

Returns

- 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.

C++

```
CyU3PReturnStatus\_t CyU3PUsbDoRemoteWakeup(  
    void  
);
```

Group

[USB Device Mode Functions](#)

File

cyu3usb.h

6.3.3.24 CyU3PUsbMapStream

Map a socket to stream of the specified endpoint.

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.

Parameters

Parameters	Description
uint8_t ep	Endpoint number for which the stream is to be mapped
uint8_t socketNum	Socket number for mapping the stream
uint16_t streamId	StreamId to be mapped

Returns

- 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

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)
- [CyU3PSetEpConfig](#)
- [CyU3PSetEpPacketSize](#)
- [CyU3PUsbGetEpCfg](#)
- [CyU3PUsbGetSpeed](#)
- [CyU3PUsbSetEpNak](#)
- [CyU3PUsbFlushEp](#)
- [CyU3PUsbResetEp](#)
- [CyU3PUsbStall](#)
- [CyU3PUsbChangeMapping](#)
- [CyU3PUsbSendErdy](#)
- [CyU3PUsbSendNrdy](#)

C++

```
CyU3PReturnStatus\_t CyU3PUsbMapStream(  
    uint8_t ep,  
    uint8_t socketNum,  
    uint16_t streamId  
);
```

Group

[USB Device Mode Functions](#)

File

cyu3usb.h

6.3.3.25 CyU3PUsbChangeMapping

Map a socket to stream of the specified endpoint.

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.

Parameters

Parameters	Description
uint8_t ep	Endpoint number to which the stream was mapped
uint8_t socketNum	Socket number to which the stream was mapped
CyBool_t remap	CyTrue: if remapping is to be done
uint16_t newstreamId	Stream id of the new stream to be mapped to the socket
uint8_t newep	Endpoint number of the new stream to be mapped to the socket

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PUsbChangeMapping(  
    uint8_t ep,  
    uint8_t socketNum,  
    CyBool_t remap,  
    uint16_t newstreamId,  
    uint8_t newep  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)
- [CyU3PSetEpConfig](#)
- [CyU3PSetEpPacketSize](#)
- [CyU3PUsbGetEpCfg](#)
- [CyU3PUsbGetSpeed](#)
- [CyU3PUsbSetEpNak](#)
- [CyU3PUsbFlushEp](#)
- [CyU3PUsbResetEp](#)

- [CyU3PUsbStall](#)
- [CyU3PUsbMapStream](#)
- [CyU3PUsbSendErDy](#)
- [CyU3PUsbSendNrdy](#)

File

cyu3usb.h

6.3.3.26 CyU3PUsbSendErdy

Function to send an ERDY TP to a USB 3.0 host.

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.

Parameters

Parameters	Description
uint8_t ep	Endpoint for which the ERDY TP is to be generated. Bit 7 of the endpoint indicates direction.
uint16_t bulkStream	Stream number for which the ERDY TP is to be generated. Is only valid for stream enabled bulk endpoints and is ignored otherwise.

Returns

- 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.

C++

```
CyU3PReturnStatus_t CyU3PUsbSendErdy(  
    uint8_t ep,  
    uint16_t bulkStream  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)
- [CyU3PSetEpConfig](#)
- [CyU3PUsbGetEpCfg](#)
- [CyU3PUsbGetSpeed](#)
- [CyU3PUsbSetEpNak](#)
- [CyU3PUsbAckSetup](#)
- [CyU3PUsbFlushEp](#)
- [CyU3PUsbResetEp](#)
- [CyU3PUsbStall](#)
- [CyU3PUsbMapStream](#)

- [CyU3PUsbChangeMapping](#)
- [CyU3PUsbSendNrdy](#)

File

cyu3usb.h

6.3.3.27 CyU3PUsbSendNrdy

Function to send an NRDY TP to a USB 3.0 host.

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.

Parameters

Parameters	Description
uint8_t ep	Endpoint for which the NRDY TP is to be generated. Bit 7 of the endpoint indicates direction.
uint16_t bulkStream	Stream number for which the NRDY TP is to be generated. Is only valid for stream enabled bulk endpoints and is ignored otherwise.

Returns

- 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.

C++

```
CyU3PReturnStatus_t CyU3PUsbSendNrdy(  
    uint8_t ep,  
    uint16_t bulkStream  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PUsbSetDesc](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)
- [CyU3PSetEpConfig](#)
- [CyU3PSetEpPacketSize](#)
- [CyU3PUsbGetEpCfg](#)
- [CyU3PUsbGetSpeed](#)
- [CyU3PUsbSetEpNak](#)
- [CyU3PUsbFlushEp](#)
- [CyU3PUsbResetEp](#)
- [CyU3PUsbStall](#)

- [CyU3PUsbMapStream](#)
- [CyU3PUsbChangeMapping](#)
- [CyU3PUsbSendErdy](#)

File

cyu3usb.h

6.3.3.28 CyU3PUsbGetDevProperty

Function that returns some properties of the USB device.

This function is used to retrieve properties of the USB device such as Device address, current ITP timestamp or frame number etc.

Parameters

Parameters	Description
<code>CyU3PUsbDevProperty</code> type	The type of property to be queried.
<code>uint32_t * buf</code>	Buffer into which the property value is copied. The format of the data in the buffer depends on the property being queried. See <code>CyU3PUsbDevProperty</code> for details on how the property data is returned.

Returns

- `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.

C++

```
CyU3PReturnStatus_t CyU3PUsbGetDevProperty(  
    CyU3PUsbDevProperty type,  
    uint32_t * buf  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbDevProperty](#)
- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PUsbVBattEnable](#)
- [CyU3PUsbRegisterEventCallback](#)
- [CyU3PUsbRegisterSetupCallback](#)
- [CyU3PUsbRegisterEpEvtCallback](#)
- [CyU3PUsbRegisterLPMRequestCallback](#)
- [CyU3PUsbSetDesc](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)
- [CyU3PUsbGetLinkPowerState](#)
- [CyU3PUsbSetLinkPowerState](#)

- [CyU3PUsbLinkPowerMode](#)
- [CyU3PUsbDevProperty](#)

File

cyu3usb.h

6.3.3.29 CyU3PUsbGetLinkPowerState

Function to get the current power state of the USB 3.0 link.

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.

Parameters

Parameters	Description
CyU3PUsbLinkPowerMode * mode_p	Return parameter that will be filled in with the current power state.

Returns

- 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.

C++

```
CyU3PReturnStatus_t CyU3PUsbGetLinkPowerState(  
    CyU3PUsbLinkPowerMode * mode_p  
) ;
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PUsbVBattEnable](#)
- [CyU3PUsbRegisterEventCallback](#)
- [CyU3PUsbRegisterSetupCallback](#)
- [CyU3PUsbRegisterEpEvtCallback](#)
- [CyU3PUsbRegisterLPMRequestCallback](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)
- [CyU3PUsbSetLinkPowerState](#)
- [CyU3PUsbLinkPowerMode](#)
- [CyU3PUsbGetDevProperty](#)
- [CyU3PUsbDevProperty](#)

File

cyu3usb.h

6.3.3.30 CyU3PUsbSetLinkPowerState

Function to request a device entry into one of the U0, U1 or U2 link power modes.

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.

Parameters

Parameters	Description
CyU3PUsbLinkPowerMode link_mode	Desired link power state.

Returns

- 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.

C++

```
CyU3PReturnStatus\_t CyU3PUsbSetLinkPowerState(  
    CyU3PUsbLinkPowerMode link_mode  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbStart](#)
- [CyU3PUsbStop](#)
- [CyU3PUsbVBattEnable](#)
- [CyU3PUsbRegisterEventCallback](#)
- [CyU3PUsbRegisterSetupCallback](#)
- [CyU3PUsbRegisterEpEvtCallback](#)
- [CyU3PUsbRegisterLPMRequestCallback](#)
- [CyU3PUsbSetDesc](#)
- [CyU3PConnectState](#)
- [CyU3PGetConnectState](#)
- [CyU3PUsbGetLinkPowerState](#)
- [CyU3PUsbLinkPowerMode](#)
- [CyU3PUsbGetDevProperty](#)

- [CyU3PUsbDevProperty](#)

File

cyu3usb.h

6.3.3.31 CyU3PUsbEnableITPEvent

Function to enable/disable notification of SOF/ITP events to the application.

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.

Parameters

Parameters	Description
CyBool_t enable	Whether SOF/ITP event notification should be enabled.

Returns

- CY_U3P_SUCCESS - if the event change was successful.
- CY_U3P_ERROR_NOT_STARTED - if the USB driver has not been started.

C++

```
CyU3PReturnStatus_t CyU3PUsbEnableITPEvent(  
    CyBool_t enable  
);
```

Group

[USB Device Mode Functions](#)

Notes

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.

See Also

- [CyU3PUsbEventType_t](#)
- [CyU3PUSBEventCb_t](#)

File

cyu3usb.h

6.3.3.32 CyU3PUsbSendDevNotification

Function to send a DEV_NOTIFICATION Transaction Packet to the host.

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.

Parameters

Parameters	Description
uint8_t notificationType	Notification type - No validation is performed by the API.
uint32_t param0	Notification Type Specific Data - DWORD1[31:8].
uint32_t param1	Notification Type Specific Data - DWORD2[31:0].

Returns

- 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.

C++

```
CyU3PReturnStatus\_t CyU3PUsbSendDevNotification(
    uint8_t notificationType,
    uint32_t param0,
    uint32_t param1
);
```

Group

[USB Device Mode Functions](#)

Notes

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.

File

cyu3usb.h

6.3.3.33 CyU3PUsbForceFullSpeed

Function to force the USB 2.0 device connection to full speed.

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.

Parameters

Parameters	Description
CyBool_t enable	Whether to enable/disable the forced full speed operation.

Returns

- 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.

C++

```
CyU3PReturnStatus\_t CyU3PUsbForceFullSpeed(  
    CyBool_t enable  
);
```

Group

[USB Device Mode Functions](#)

Notes

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.

File

cyu3usb.h

6.3.3.34 CyU3PUsbLPMDisable

Function to prevent FX3 from entering U1/U2 states.

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 Force_LINKPM_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.

Returns

- 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 is not in Super Speed mode.

C++

```
CyU3PReturnStatus\_t CyU3PUsbLPMDisable(  
    void  
);
```

Group

[USB Device Mode Functions](#)

Notes

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.

See Also

- [CyU3PUsbLPMEnable](#)

File

cyu3usb.h

6.3.3.35 CyU3PUsbLPMEnable

Function to re-enable automated handling of U1/U2 state entry.

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.

Returns

- 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 is not in Super Speed mode.

C++

```
CyU3PReturnStatus\_t CyU3PUsbLPMEnable(  
    void  
);
```

Group

[USB Device Mode Functions](#)

Notes

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.

See Also

- [CyU3PUsbLPMDisable](#)

File

cyu3usb.h

6.3.3.36 CyU3PUsbInitEventLog

Function to initiate logging of USB state changes into a circular buffer.

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.

Parameters

Parameters	Description
uint8_t * buffer	Pointer to memory buffer into which events are to be logged.
uint32_t bufSize	Size of memory buffer in bytes.

Returns

None

C++

```
void CyU3PUsbInitEventLog(
    uint8_t * buffer,
    uint32_t bufSize
);
```

Group

[USB Device Mode Functions](#)

File

cyu3usb.h

6.3.3.37 CyU3PUsbGetEventLogIndex

Get the current event log buffer index.

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.

Returns

Current buffer index

C++

```
uint16_t CyU3PUsbGetEventLogIndex(  
    void  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbInitEventLog](#)

File

cyu3usb.h

6.3.3.38 CyU3PUsbSetBooterSwitch

Function to enable/disable switching control back to the FX3 2-stage bootloader.

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\(\)](#).

Parameters

Parameters	Description
CyBool_t enable	CyTrue - Enable switching to booter CyFalse - Disable switching to booter

Returns

- None

C++

```
void CyU3PUsbSetBooterSwitch(
    CyBool_t enable
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbJumpBackToBooter](#)
- [CyU3PUsbStart](#)

File

cyu3usb.h

6.3.3.39 CyU3PUsbJumpBackToBooter

Function to transfer control back to the FX3 2-stage bootloader.

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.

Parameters

Parameters	Description
uint32_t address	Address of the FX3 2-stage bootloader's entry point.

Returns

- 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-returnable call.

C++

```
CyU3PReturnStatus\_t CyU3PUsbJumpBackToBooter(  
    uint32_t address  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PUsbSetBooterSwitch](#)

File

cyu3usb.h

6.3.3.40 CyU3PUsbEpPrepare

Prepare all enabled USB device endpoints for data transfer.

This function prepare 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.

Parameters

Parameters	Description
CyU3PUSBSpeed_t curSpeed	Current USB connection speed.

C++

```
void CyU3PUsbEpPrepare(
    CyU3PUSBSpeed\_t curSpeed
);
```

Group

[USB Device Mode Functions](#)

Notes

The caller of this function is expected to determine the current link operating speed, and then call this API with the appropriate parameter.

See Also

- [CyU3PUsbGetSpeed](#)
- [CyU3PSetEpPacketSize](#)
- [CyU3PUsbStall](#)

Return Values

- None

File

cyu3usb.h

6.3.3.41 CyU3PUsbEnableEPPrefetch

Configure the USB DMA interface to continuously pre-fetch data.

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.

C++

```
void CyU3PUsbEnableEPPrefetch(  
    void  
);
```

Group

[USB Device Mode Functions](#)

See Also

- CY_U3P_USB_EVENT_EP_UNDERRUN

Return Values

- None

File

cyu3usb.h

6.3.3.42 CyU3PUsbResetEndpointMemories

Reset and re-initialize the endpoint memory block on FX3.

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.

Return Values

- CY_U3P_SUCCESS in case of successful reset and re-initialization.
- CY_U3P_ERROR_NOT_STARTED if the USB block has not been started.

C++

```
CyU3PReturnStatus\_t CyU3PUsbResetEndpointMemories(  
    void  
);
```

Group

[USB Device Mode Functions](#)

File

cyu3usb.h

6.3.3.43 CyU3PUsbGetErrorCounts

Function to get the number of USB 3.0 PHY and LINK error counts detected by FX3.

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 Values

- 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

Parameters	Description
uint16_t * phy_err_cnt	Return parameter to be filled with phy error count.
uint16_t * lnk_err_cnt	Return parameter to be filled with link error count.

C++

```
CyU3PReturnStatus\_t CyU3PUsbGetErrorCounts(  
    uint16_t * phy_err_cnt,  
    uint16_t * lnk_err_cnt  
);
```

Group

[USB Device Mode Functions](#)

File

cyu3usb.h

6.3.3.44 CyU3PUsbSetEpPktMode

Select whether a OUT endpoint will function in packet (one buffer per packet) mode or not.

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.

Parameters

Parameters	Description
uint8_t ep	Endpoint number to be configured.
CyBool_t pktMode	Whether the endpoint will function in packet mode or not.

Returns

- 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.

C++

```
CyU3PReturnStatus_t CyU3PUsbSetEpPktMode(  
    uint8_t ep,  
    CyBool_t pktMode  
);
```

Group

[USB Device Mode Functions](#)

See Also

- [CyU3PSetEpPacketSize](#)

File

cyu3usb.h

6.4 USB Host Management

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.

Group

USB Management




Topics

Topic	Description
USB Host Data Types	This section documents the data types defined and used by the USB device mode APIs.
USB Host Mode Functions	The USB host APIs are used to configure the USB host functionality and to perform USB data transfers.

6.4.1 USB Host Data Types

This section documents the data types defined and used by the USB device mode APIs.



Enumerations

Enumeration	Description
 CyU3PUsbHostOpSpeed_t	Speed of operation for the usb host port.
 CyU3PUsbHostEventType_t	Different host mode events.
 CyU3PUsbHostEpXferType_t	Mode of transfer.

Group

USB Host Management

Legend



	Enumeration
	Structure

Macros

Macro	Description
CY_U3P_USB_HOST_EPS_EPNUM_POS	Position of EP number field.
CY_U3P_USB_HOST_EPS_EPNUM_MASK	Mask for the EP number field.
CY_U3P_USB_HOST_EPS_EPDIR	EP direction: 1 - OUT, 0 - IN.
CY_U3P_USB_HOST_EPS_ACTIVE	Whether the EP is still active.
CY_U3P_USB_HOST_EPS_HALT	Set if EP is halted (stalled).
CY_U3P_USB_HOST_EPS_OVER_UNDER_RUN	Set if EP buffer had an overrun or underrun happened.
CY_U3P_USB_HOST_EPS_BABBLE	Set if a babble was detected during transfer.
CY_U3P_USB_HOST_EPS_XACT_ERROR	Set if there was a transfer error of any kind.
CY_U3P_USB_HOST_EPS_PING	Set if there was a ping.
CY_U3P_USB_HOST_EPS_PHY_ERROR	Set if there was a PHY error during transfer.
CY_U3P_USB_HOST_EPS_PID_ERROR	Set if there was a PID error during transfer.

CY_U3P_USB_HOST_EPS_TIMEOUT_ERROR	Set if there was a timeout error during transfer.
CY_U3P_USB_HOST_EPS_ISO_ORUN_ERROR	Set if there was an ISO overrun error.
CY_U3P_USB_HOST_EPS_IOC_INT	Set if there was an IOC interrupt (high speed only).
CY_U3P_USB_HOST_EPS_BYTE_COUNT_POS	Position for remaining bytes to be transferred.
CY_U3P_USB_HOST_EPS_BYTE_COUNT_MASK	Mask for remaining bytes to be transferred.
CY_U3P_USB_HOST_PORT_STAT_CONNECTED	Set if a down-stream peripheral is connected.
CY_U3P_USB_HOST_PORT_STAT_ENABLED	Set if the port is enable using CyU3PUsbHostPortEnable API.
CY_U3P_USB_HOST_PORT_STAT_ACTIVE	Mask to identify if the port is active. The port is active if port is enabled and connected but not suspended.
CY_U3P_USB_HOST_PORT_STAT_SUSPENDED	Set if the port has been suspended.

Structures

Structure	Description
 CyU3PUsbHostConfig_t	Host mode configuration information.
 CyU3PUsbHostEpConfig_t	Host mode endpoint configuration structure.

Types

Type	Description
CyU3PUsbHostPortStatus_t	Host mode port status information.
CyU3PUsbHostEventCb_t	Host mode event callback function.
CyU3PUsbHostEpStatus_t	Host mode endpoint status.
CyU3PUsbHostXferCb_t	Host mode endpoint transfer complete callback function.

6.4.1.1 CY_U3P_USB_HOST_EPS_EPNUM_POS

Position of EP number field.

C++

```
#define CY_U3P_USB_HOST_EPS_EPNUM_POS (0) /* Position of EP number field. */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.2 CY_U3P_USB_HOST_EPS_EPNUM_MASK

Mask for the EP number field.

C++

```
#define CY_U3P_USB_HOST_EPS_EPNUM_MASK (0x0000000F) /* Mask for the EP number field.
*/
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.3 CY_U3P_USB_HOST_EPS_EPDIR

EP direction: 1 - OUT, 0 - IN.

C++

```
#define CY_U3P_USB_HOST_EPS_EPDIR (0x00000010) /* EP direction: 1 - OUT, 0 - IN. */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.4 CY_U3P_USB_HOST_EPS_ACTIVE

Whether the EP is still active.

C++

```
#define CY_U3P_USB_HOST_EPS_ACTIVE (0x00000020) /* Whether the EP is still active. */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.5 CY_U3P_USB_HOST_EPS_HALT

Set if EP is halted (stalled).

C++

```
#define CY_U3P_USB_HOST_EPS_HALT (0x00000040) /* Set if EP is halted (stalled). */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.6 CY_U3P_USB_HOST_EPS_OVER_UNDER_RUN

Set if EP buffer had an overrun or underrun happened.

C++

```
#define CY_U3P_USB_HOST_EPS_OVER_UNDER_RUN (0x00000080) /* Set if EP buffer had an  
overrun or underrun happened. */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.7 CY_U3P_USB_HOST_EPS_BABBLE

Set if a babble was detected during transfer.

C++

```
#define CY_U3P_USB_HOST_EPS_BABBLE (0x00000100) /* Set if a babble was detected  
during transfer. */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.8 CY_U3P_USB_HOST_EPS_XACT_ERROR

Set if there was a transfer error of any kind.

C++

```
#define CY_U3P_USB_HOST_EPS_XACT_ERROR (0x00000200) /* Set if there was a transfer  
error of any kind. */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.9 CY_U3P_USB_HOST_EPS_PING

Set if there was a ping.

C++

```
#define CY_U3P_USB_HOST_EPS_PING (0x00000400) /* Set if there was a ping. */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.10 CY_U3P_USB_HOST_EPS_PHY_ERROR

Set if there was a PHY error during transfer.

C++

```
#define CY_U3P_USB_HOST_EPS_PHY_ERROR (0x00000800) /* Set if there was a PHY error  
during transfer. */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.11 CY_U3P_USB_HOST_EPS_PID_ERROR

Set if there was a PID error during transfer.

C++

```
#define CY_U3P_USB_HOST_EPS_PID_ERROR (0x00001000) /* Set if there was a PID error  
during transfer. */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.12 CY_U3P_USB_HOST_EPS_TIMEOUT_ERROR

Set if there was a timeout error during transfer.

C++

```
#define CY_U3P_USB_HOST_EPS_TIMEOUT_ERROR (0x00002000) /* Set if there was a timeout  
error during transfer. */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.13 CY_U3P_USB_HOST_EPS_ISO_ORUN_ERROR

Set if there was an ISO overrun error.

C++

```
#define CY_U3P_USB_HOST_EPS_ISO_ORUN_ERROR (0x00004000) /* Set if there was an ISO
overrun error. */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.14 CY_U3P_USB_HOST_EPS_IOC_INT

Set if there was an IOC interrupt (high speed only).

C++

```
#define CY_U3P_USB_HOST_EPS_IOC_INT (0x00008000) /* Set if there was an IOC  
interrupt (high speed only. */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.15 CY_U3P_USB_HOST_EPS_BYTE_COUNT_POS

Position for remaining bytes to be transferred.

C++

```
#define CY_U3P_USB_HOST_EPS_BYTE_COUNT_POS (16) /* Position for remaining bytes to  
be transferred. */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.16 CY_U3P_USB_HOST_EPS_BYTE_COUNT_MASK

Mask for remaining bytes to be transferred.

C++

```
#define CY_U3P_USB_HOST_EPS_BYTE_COUNT_MASK (0xFFFF0000) /* Mask for remaining bytes  
to be transferred. */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.17 CY_U3P_USB_HOST_PORT_STAT_CONNECTED

Set if a down-stream peripheral is connected.

C++

```
#define CY_U3P_USB_HOST_PORT_STAT_CONNECTED (0x0001) /* Set if a down-stream  
peripheral is connected. */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.18 CY_U3P_USB_HOST_PORT_STAT_ENABLED

Set if the port is enable using [CyU3PUsbHostPortEnable](#) API.

C++

```
#define CY_U3P_USB_HOST_PORT_STAT_ENABLED (0x0002) /* Set if the port is enable  
using CyU3PUsbHostPortEnable API. */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.19 CY_U3P_USB_HOST_PORT_STAT_ACTIVE

Mask to identify if the port is active. The port is active if port is enabled and connected but not suspended.

C++

```
#define CY_U3P_USB_HOST_PORT_STAT_ACTIVE (0x0003) /* Mask to identify if the port is  
active. The port is active  
enabled and connected but not suspended. */                                if port is
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.20 CY_U3P_USB_HOST_PORT_STAT_SUSPENDED

Set if the port has been suspended.

C++

```
#define CY_U3P_USB_HOST_PORT_STAT_SUSPENDED (0x0004) /* Set if the port has been  
suspended. */
```

Group

[USB Host Data Types](#)

File

cyu3usbhost.h

6.4.1.21 CyU3PUsbHostOpSpeed_t

Speed of operation for the usb host port.

The usb host port can function in low, full and high speeds according to the capability of the device attached. This enumeration gives the various speed of operation.

C++

```
enum CyU3PUsbHostOpSpeed_t {
    CY_U3P_USB_HOST_LOW_SPEED = 0,
    CY_U3P_USB_HOST_FULL_SPEED,
    CY_U3P_USB_HOST_HIGH_SPEED
};
```

Group

[USB Host Data Types](#)

See Also

- [CyU3PUsbHostGetPortStatus](#)

Members

Members	Description
CY_U3P_USB_HOST_LOW_SPEED = 0	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.

File

cyu3usbhost.h

6.4.1.22 CyU3PUsbHostPortStatus_t

Host mode port status information.

The port status returned by the library is a 16 bit value with different fields. The fields are described in the macros listed.

C++

```
typedef uint16_t CyU3PUsbHostPortStatus_t;
```

Group

[USB Host Data Types](#)

See Also

[CyU3PUsbHostGetPortStatus](#)

File

cyu3usbhost.h

6.4.1.23 CyU3PUsbHostEventType_t

Different host mode events.

The different types of events reported to the application via the event callback function.

C++

```
enum CyU3PUsbHostEventType_t {
    CY_U3P_USB_HOST_EVENT_CONNECT = 0,
    CY_U3P_USB_HOST_EVENT_DISCONNECT
};
```

Group

[USB Host Data Types](#)

See Also

[CyU3PUsbHostEventCb_t](#)

Members

Members	Description
CY_U3P_USB_HOST_EVENT_CONNECT = 0	USB Connect event.
CY_U3P_USB_HOST_EVENT_DISCONNECT	USB Disconnect event.

File

cyu3usbhost.h

6.4.1.24 CyU3PUsbHostEventCb_t

Host mode event callback function.

The event callback function is registered during start of the host stack. The function is invoked by the library on getting events like peripheral connect / disconnect.

C++

```
typedef void (* CyU3PUsbHostEventCb_t)(CyU3PUsbHostEventType\_t evType, uint32_t  
evData);
```

Group

[USB Host Data Types](#)

See Also

[CyU3PUsbHostStart](#)

File

cyu3usbhost.h

6.4.1.25 CyU3PUsbHostEpXferType_t

Mode of transfer.

Depending upon the type of endpoint, the transfer can be normal or setup. A setup packet is send only on EP0. This is a parameter to the SetXfer call.

C++

```
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
};
```

Group

USB Host Data Types

See Also

- [CyU3PUsbHostEpSetXfer](#)

Members

Members	Description
CY_U3P_USB_HOST_EPXFER_NORMAL = 0	Normal transfer. All non-EP0 tranfers.
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.

File

cyu3usbhost.h

6.4.1.26 CyU3PUsbHostEpStatus_t

Host mode endpoint status.

The endpoint status returned by the scheduler is a 32 bit value with different fields listing the status of the transfer on the endpoint. The fields are described in the macros listed.

C++

```
typedef uint32_t CyU3PUsbHostEpStatus_t;
```

Group

[USB Host Data Types](#)

See Also

[CyU3PUsbHostXferCb_t](#)

File

cyu3usbhost.h

6.4.1.27 CyU3PUsbHostXferCb_t

Host mode endpoint transfer complete callback function.

The transfer callback function is registered during start of the host stack. The function is invoked by the library on completion of transfer.

C++

```
typedef void (* CyU3PUsbHostXferCb_t)(uint8_t ep, CyU3PUsbHostEpStatus\_t epStatus);
```

Group

[USB Host Data Types](#)

See Also

[CyU3PUsbHostStart](#) [CyU3PUsbHostEpSetXfer](#)

File

cyu3usbhost.h

6.4.1.28 CyU3PUsbHostConfig_t

Host mode configuration information.

The host mode stack can be started with the following configurations. These options cannot be changed dynamically. The stack has to be stopped and started for changing any parameters.

C++

```
struct CyU3PUsbHostConfig_t {  
    CyBool_t ep0LowLevelControl;  
    CyU3PUsbHostEventCb\_t eventCb;  
    CyU3PUsbHostXferCb\_t xferCb;  
};
```

Group

[USB Host Data Types](#)

See Also

[CyU3PUsbHostStart](#)

Members

Members	Description
CyBool_t ep0LowLevelControl;	Whether to enable EP0 low level control. If CyFalse, the EP0 DMA is handled by firmware. Only CyU3PUsbHostSendSetupRqt need to be called. If CyTrue, all DMA paths need to be handled by application. This allows fine control over setup, data and status phase. This should be used only if the EP0 data need to be routed to a different path. It is advised to leave this field as CyFalse. It should also be noted that the maxPktSize and fullPktSize should be the same if the flag is CyTrue.
CyU3PUsbHostEventCb_t eventCb;	Event callback function for USB host stack.
CyU3PUsbHostXferCb_t xferCb;	EP transfer completion callback for USB host stack.

File

cyu3usbhost.h

6.4.1.29 CyU3PUsbHostEpConfig_t

Host mode endpoint configuration structure.

The structure holds the information for configuring the endpoint.

C++

```
struct CyU3PUsbHostEpConfig_t {
    CyU3PUsbEpType\_t type;
    uint8_t mult;
    uint16_t maxPktSize;
    uint8_t pollingRate;
    uint16_t fullPktSize;
    CyBool_t isStreamMode;
};
```

Group

[USB Host Data Types](#)

See Also

[CyU3PUsbHostEpAdd](#)

Members

Members	Description
CyU3PUsbEpType_t type;	Type of endpoint to be created.
uint8_t mult;	Number of packets per micro-frame. This should be 1 for bulk and control and 1, 2, or 3 as ISO / interrupt requirements. The only valid values are 1, 2 and 3.
uint16_t maxPktSize;	The maximum packet size that can be received from the EP. Valid values are as per the USB 2.0 spec.
uint8_t pollingRate;	Rate at which the endpoint has to be polled in ms. A zero will indicate that polling will be done every micro-frame and any other value will poll at a rate specified. It should be noted that pollingRate is valid only when the request itself is larger than a single packet. For example if every packet to be received is queued as a separate request, then the polling rate has to be manually maintained by the application. This is valid only for the synchronous EP. For asynchronous EPs this value should be zero.
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 hardware when it sees a SLP or ZLP. So as long as data received is a multiple of fullPktSize, the buffer is not wrapped up. fullPktSize cannot be smaller than the maxPktSize.
CyBool_t isStreamMode;	Enable stream mode. This means that the EP is always active. This is valid only for an IN EP. It should be CyFalse for EP0 as well as for OUT EPs. When the flag is CyTrue, an IN EP will send IN token and collect data whenever there is free buffer.

















File

cyu3usbhost.h

6.4.2 USB Host Mode Functions

The USB host APIs are used to configure the USB host functionality and to perform USB data transfers.

Functions

Function	Description
 CyU3PUsbHostStart	This function initializes the USB 2.0 host stack.
 CyU3PUsbHostStop	This function de-inits the USB host stack.
 CyU3PUsbHostIsStarted	This function returns whether the USB host module has been started.
 CyU3PUsbHostPortEnable	This function enables the USB host port.
 CyU3PUsbHostPortDisable	Disable the USB host port.
 CyU3PUsbHostGetPortStatus	This function retrieves the current port status.
 CyU3PUsbHostPortReset	This function resets the USB host port.
 CyU3PUsbHostPortSuspend	This function suspends the USB host port.
 CyU3PUsbHostPortResume	This function resumes the previously suspended USB host port.
 CyU3PUsbHostSetDeviceAddress	This function sets the current downstream peripheral address.
 CyU3PUsbHostGetDeviceAddress	This function gets the current downstream peripheral address.
 CyU3PUsbHostGetFrameNumber	This function gets the current frame number to be used.
 CyU3PUsbHostSendSetupRqt	This function does EP0 setup transfer handling.
 CyU3PUsbHostEpAdd	This function adds an endpoint to the scheduler.
 CyU3PUsbHostEpRemove	This function removes an endpoint from the scheduler.
 CyU3PUsbHostEpReset	This function resets an endpoint state.
 CyU3PUsbHostEpSetXfer	This function starts a transfer for all non-EP0 endpoints and sets the transfer parameters for EP0. This function is relevant for EP0 only if the ep0LowLevelControl is set to CyTrue while starting the host stack. If the API is invoked for EP0 when the ep0LowLevelControl is CyFalse, then the behaviour is undefined as this API will be used by the library for setting up the EP0 transfers.
 CyU3PUsbHostEp0BeginXfer	This function enables and starts transfer on EP0. The function is relevant only if the ep0LowLevelControl is set to CyTrue while starting the host stack. If the API is invoked for EP0 when the ep0LowLevelControl is CyFalse, then the behaviour is undefined as this API will be used by the library for setting up the EP0 transfers.
 CyU3PUsbHostEpAbort	This function aborts pending transfers on selected endpoint.
 CyU3PUsbHostEpWaitForCompletion	This function waits for the current endpoint transfer to complete.

Group

[USB Host Management](#)

Legend

	Method
---	--------

6.4.2.1 CyU3PUsbHostStart

This function initializes the USB 2.0 host stack.

This function enables the USB block and configures it to function as USB host. The configuration parameters cannot be changed unless the stack is stopped and re-started.

Parameters

Parameters	Description
CyU3PUsbHostConfig_t * hostCfg	Pointer to the host configuration information.

Returns

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 wrong OTG mode or USB device stack is running.

C++

```
CyU3PReturnStatus_t CyU3PUsbHostStart(  
    CyU3PUsbHostConfig_t * hostCfg  
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostConfig_t](#) [CyU3PUsbHostStop](#) [CyU3PUsbHostIsStarted](#)

File

cyu3usbhost.h

6.4.2.2 CyU3PUsbHostStop

This function de-inits the USB host stack.

This function disables the USB host mode operation.

Returns

CY_U3P_SUCCESS - The call was successful. CY_U3P_ERROR_NOT_STARTED - The host mode stack is not running.

C++

```
CyU3PReturnStatus\_t CyU3PUsbHostStop(  
    void  
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostStart](#) [CyU3PUsbHostIsStarted](#)

File

cyu3usbhost.h

6.4.2.3 CyU3PUsbHostIsStarted

This function returns whether the USB host module has been started.

Since there can be various modes of USB operations this API returns whether [CyU3PUsbHostStart](#) was invoked.

Returns

CyTrue - USB host module started
CyFalse - USB host module stopped or not started

C++

```
CyBool_t CyU3PUsbHostIsStarted(  
    void  
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostStart](#) [CyU3PUsbHostStop](#)

File

cyu3usbhost.h

6.4.2.4 CyU3PUsbHostPortEnable

This function enables the USB host port.

When the host stack is started, the host mode port is not enabled completely. Only peripheral detection is turned on. USB 3.0 platform devices has only a single usb host port which can support only a single peripheral device. Hubs are not supported. This function is expected to be invoked by the application when a downstream peripheral is detected. This function enables the port and sets it to the correct speed of operation. When a down-stream peripheral gets disconnected, the port automatically gets disabled, and port needs to be explicitly enabled every time a connection is detected. The call can be made only when there is a downstream peripheral attached.

Returns

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.

C++

```
CyU3PReturnStatus\_t CyU3PUsbHostPortEnable(  
    void  
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostPortDisable](#) [CyU3PUsbHostGetPortStatus](#)

File

cyu3usbhost.h

6.4.2.5 CyU3PUsbHostPortDisable

Disable the USB host port.

The function will disable the host port activity. The port will no-longer be able to function or do data transfers. The port will still be able to do peripheral detection.

Returns

CY_U3P_SUCCESS - The call was successful. CY_U3P_ERROR_NOT_STARTED - The port is not enabled.

C++

```
CyU3PReturnStatus\_t CyU3PUsbHostPortDisable(  
    void  
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostPortEnable](#) [CyU3PUsbHostGetPortStatus](#)

File

cyu3usbhost.h

6.4.2.6 CyU3PUsbHostGetPortStatus

This function retrieves the current port status.

This function will just retrieve the current state of the port and the current speed of operation for the port.

Parameters

Parameters	Description
CyU3PUsbHostPortStatus_t * portStatus	Current port status.
CyU3PUsbHostOpSpeed_t * portSpeed	Current port speed.

Returns

CY_U3P_SUCCESS - The call was successful. CY_U3P_ERROR_NOT_STARTED - The host stack is not running.

C++

```
CyU3PReturnStatus_t CyU3PUsbHostGetPortStatus(  
    CyU3PUsbHostPortStatus_t * portStatus,  
    CyU3PUsbHostOpSpeed_t * portSpeed  
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostPortEnable](#) [CyU3PUsbHostPortDisable](#)

File

cyu3usbhost.h

6.4.2.7 CyU3PUsbHostPortReset

This function resets the USB host port.

A port reset is nothing but a [CyU3PUsbHostPortDisable](#) () call followed by a [CyU3PUsbHostPortEnable](#) () call. The port reset shall reset the host port's state machine and re-enumeration can take place.

Returns

CY_U3P_SUCCESS - The call was successful. CY_U3P_ERROR_NOT_STARTED - The host stack is not running.

C++

```
CyU3PReturnStatus\_t CyU3PUsbHostPortReset(  
    void  
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostPortEnable](#) [CyU3PUsbHostPortDisable](#) [CyU3PUsbHostGetPortStatus](#)

File

cyu3usbhost.h

6.4.2.8 CyU3PUsbHostPortSuspend

This function suspends the USB host port.

The port shall be suspended and will resume on subsequent resume call. It should be noted that remote wakeup is not supported and the host should wakeup the device whenever required.

Returns

CY_U3P_SUCCESS - The call was successful. CY_U3P_ERROR_NOT_STARTED - The port is not active.
CY_U3P_ERROR_INVALID_SEQUENCE - There is active on-going transfer..

C++

```
CyU3PReturnStatus\_t CyU3PUsbHostPortSuspend(  
    void  
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostPortResume](#) [CyU3PUsbHostGetPortStatus](#)

File

cyu3usbhost.h

6.4.2.9 CyU3PUsbHostPortResume

This function resumes the previously suspended USB host port.

The port resumes operation at this point. The function will succeed only if the bus is suspended.

Returns

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.
CY_U3P_ERROR_NOT_SUPPORTED - TODO Not implemented.

C++

```
CyU3PReturnStatus\_t CyU3PUsbHostPortResume(  
    void  
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostPortSuspend](#) [CyU3PUsbHostGetPortStatus](#)

File

cyu3usbhost.h

6.4.2.10 CyU3PUsbHostSetDeviceAddress

This function sets the current downstream peripheral address.

The function sets the device address to be used by the host controller to address the down-stream peripheral. On enabling the port, the device address set to zero by default. The address should be changed by the application accordingly after a successful SET_ADDRESS setup request to the device.

Parameters

Parameters	Description
uint8_t devAddr	Device address to be set.

Returns

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.

C++

```
CyU3PReturnStatus\_t CyU3PUsbHostSetDeviceAddress(  
    uint8_t devAddr  
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostGetDeviceAddress](#)

File

cyu3usbhost.h

6.4.2.11 CyU3PUsbHostGetDeviceAddress

This function gets the current downstream peripheral address.

The function returns the device address used by the host controller to address the down-stream peripheral. This call simply returns the address previously set. The address gets reset to zero if there is a [CyU3PUsbHostPortDisable](#) call or if there is a disconnect.

Parameters

Parameters	Description
uint8_t * devAddr	Pointer to load the current device address.

Returns

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.

C++

```
CyU3PReturnStatus\_t CyU3PUsbHostGetDeviceAddress(
    uint8_t * devAddr
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostSetDeviceAddress](#)

File

cyu3usbhost.h

6.4.2.12 CyU3PUsbHostGetFrameNumber

This function gets the current frame number to be used.

This function call is not synchronized with the scheduler and will return the frame number at the time of call. This can be used to time the synchronous transfers.

Parameters

Parameters	Description
<code>uint32_t * frameNumber</code>	Pointer to load the frame number.

Returns

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.

C++

```
CyU3PReturnStatus\_t CyU3PUsbHostGetFrameNumber(  
    uint32_t * frameNumber  
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostStart](#) [CyU3PUsbHostPortEnable](#)

File

`cyu3usbhost.h`

6.4.2.13 CyU3PUsbHostSendSetupRqt

This function does EP0 setup transfer handling.

This API is valid only if ep0LowLevelControl is CyFalse. The API initiates the transfer of setup packet, but does not wait. The setup, data and status phases of the transfer will be handled by the library and the transfer complete callback shall be invoked for EP0 in the end.

The data required for the transfer (incase of OUT data) and buffer required for the transfer (incase of IN data) should be available before starting the transfer.

Parameters

Parameters	Description
uint8_t * setupPkt	Pointer to 8 byte setup packet.
uint8_t * buf_p	Buffer for data phase. The buffer must be capable of doing DMA transfers and should be pre-allocated to size as per the setup packet.

Returns

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.

C++

```
CyU3PReturnStatus\_t CyU3PUsbHostSendSetupRqt (
    uint8_t * setupPkt,
    uint8_t * buf_p
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostStart](#) [CyU3PUsbHostEpAbort](#)

File

cyu3usbhost.h

6.4.2.14 CyU3PUsbHostEpAdd

This function adds an endpoint to the scheduler.

Depending upon the endpoints on the downstream peripheral, the corresponding endpoint must be added to the scheduler for enabling data transfer. The endpoint configuration parameter depends on the endpoint on the down-stream peripheral.

Parameters

Parameters	Description
uint8_t ep	Endpoint to be created.
CyU3PUsbHostEpConfig_t * cfg	Endpoint configuration information.

Returns

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 field is invalid.

C++

```
CyU3PReturnStatus_t CyU3PUsbHostEpAdd(  
    uint8_t ep,  
    CyU3PUsbHostEpConfig_t * cfg  
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostEpRemove](#)

File

cyu3usbhost.h

6.4.2.15 CyU3PUsbHostEpRemove

This function removes an endpoint from the scheduler.

This function must be called when the host application is done with using an endpoint or if the type of endpoint has to be changed.

Parameters

Parameters	Description
uint8_t ep	Endpoint to be removed from scheduler.

Returns

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.

C++

```
CyU3PReturnStatus\_t CyU3PUsbHostEpRemove(  
    uint8_t ep  
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostEpAdd](#)

File

cyu3usbhost.h

6.4.2.16 CyU3PUsbHostEpReset

This function resets an endpoint state.

The function flushes the internal buffers and resets the data toggle. It should be called only when there is no active transfer on the endpoint.

Parameters

Parameters	Description
uint8_t ep	Endpoint to be reset.

Returns

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. Abort before invoking this call.

C++

```
CyU3PReturnStatus\_t CyU3PUsbHostEpReset(  
    uint8_t ep  
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostEpAdd](#)

File

cyu3usbhost.h

6.4.2.17 CyU3PUsbHostEpSetXfer

This function starts a transfer for all non-EP0 endpoints and sets the transfer parameters for EP0. This function is relevant for EP0 only if the ep0LowLevelControl is set to CyTrue while starting the host stack. If the API is invoked for EP0 when the ep0LowLevelControl is CyFalse, then the behaviour is undefined as this API will be used by the library for setting up the EP0 transfers.

This function will setup the transfer and enable the endpoint for all EPs except for EP0. When EP0 low level control is enabled, the setup packet must be queued on the DMA pipe before it can be enabled. EP0 has two data pipes: one EP0_INGRESS and one EP0_EGRESS. The setup packet needs to be queued on the egress pipe. There is only one request required for both the ingress and egress pipes. The type of transfer is decided by the type parameter. Once the setup packet is queued on the DMA pipe EP0 needs to be explicitly turned on by invoking [CyU3PUsbHostEp0BeginXfer](#).

Parameters

Parameters	Description
uint8_t ep	Endpoint to configure.
CyU3PUsbHostEpXferType_t type	Type of transfer. This is meaningful only <ul style="list-style-type: none"> for EP0.
uint32_t count	Size of data to be transferred.

Returns

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.

C++

```
CyU3PReturnStatus\_t CyU3PUsbHostEpSetXfer(
    uint8_t ep,
    CyU3PUsbHostEpXferType\_t type,
    uint32_t count
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostEp0BeginXfer](#) [CyU3PUsbHostEpAbort](#)

File

cyu3usbhost.h

6.4.2.18 CyU3PUsbHostEp0BeginXfer

This function enables and starts transfer on EP0. The function is relevant only if the `ep0LowLevelControl` is set to `CyTrue` while starting the host stack. If the API is invoked for EP0 when the `ep0LowLevelControl` is `CyFalse`, then the behaviour is undefined as this API will be used by the library for setting up the EP0 transfers.

This function enables the endpoint and start the transfer for EP0. The setup packet must be queued on EP0 egress socket before invoking this function. Also [CyU3PUsbHostEpSetXfer](#) must be invoked prior to this to setup the transfer parameters.

Returns

`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.

C++

```
CyU3PReturnStatus\_t CyU3PUsbHostEp0BeginXfer(  
    void  
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostEpSetXfer](#) [CyU3PUsbHostEpAbort](#)

File

`cyu3usbhost.h`

6.4.2.19 CyU3PUsbHostEpAbort

This function aborts pending transfers on selected endpoint.

The function deactivates the endpoint for the ongoing data transfer. This does not abort the DMA channels and it has to be done explicitly.

Parameters

Parameters	Description
uint8_t ep	Endpoint to abort.

Returns

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.

C++

```
CyU3PReturnStatus\_t CyU3PUsbHostEpAbort(  
    uint8_t ep  
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostEpSetXfer](#) [CyU3PUsbHostEp0BeginXfer](#)

File

cyu3usbhost.h

6.4.2.20 CyU3PUsbHostEpWaitForCompletion

This function waits for the current endpoint transfer to complete.

The function shall wait for the transfer to complete or get aborted. If this does not happen within the timeout specified, it returns with a timeout error. This does not mean that the transfer failed. It just means that the transfer did not complete within the specified timeout. The function can be called again or the user can wait for the transfer complete callback.

Parameters

Parameters	Description
uint8_t ep	Endpoint to wait on.
CyU3PUsbHostEpStatus_t * epStatus	Endpoint status after transfer. If this is NULL, no status is returned.
uint32_t waitOption	Wait option for the function.

Returns

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 completed in the given timeout. CY_U3P_ERROR_STALLED - The transfer was stalled by the remote device.

C++

```
CyU3PReturnStatus\_t CyU3PUsbHostEpWaitForCompletion(  
    uint8_t ep,  
    CyU3PUsbHostEpStatus\_t * epStatus,  
    uint32_t waitOption  
);
```

Group

[USB Host Mode Functions](#)

See Also

[CyU3PUsbHostEpSetXfer](#) [CyU3PUsbHostEp0BeginXfer](#)

File

cyu3usbhost.h

7 P-port Management

The Processor or P-Port on the FX3 device can be used to connect to an external processor, ASIC, FPGA or peripheral device. The electrical interface between FX3 and the other device on the P-port can be configured to implement any protocol by using the flexible GPIF-II hardware.

The GPIF configuration and transfer APIs are described in the [GPIF-II Management](#) section.

This section describes the general P-port functionality that is independent of the electrical protocol.

Topics

Topic	Description
Mailbox Handler	The mailbox handler is responsible for sending/receiving short messages from an external processor through the mailbox registers.
PIB Interface Manager	The PIB interface manager module is responsible for handling the in-bound and out-bound transfer of data through the GPIF-II interface on the FX3 device.

7.1 Mailbox Handler

The mailbox handler is responsible for sending/receiving short messages from an external processor through the mailbox registers.

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.

Group

[P-port Management](#)

Topics

Topic	Description
Mailbox Data Types	This section documents the data types defined as part of the mailbox handler.
Mailbox Functions	This section documents the functions defined as part of the mailbox handler.


7.1.1 Mailbox Data Types

This section documents the data types defined as part of the mailbox handler.


Group

[Mailbox Handler](#)

Legend

	Structure
---	-----------

Structures

Structure	Description
 CyU3PMbox	Structure that holds a packet of mailbox data.

Types

Type	Description
CyU3PMboxCb_t	Type of function to be called to notify about a mailbox related interrupt.

7.1.1.1 CyU3PMboxCb_t

Type of function to be called to notify about a mailbox related interrupt.

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.

C++

```
typedef void (* CyU3PMboxCb_t)(CyBool_t mboxEvt);
```

Group

[Mailbox Data Types](#)

File

cyu3mbox.h

7.1.1.2 CyU3PMbox

Structure that holds a packet of mailbox data.

The FX3 device has 8 byte mailbox register 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.

C++

```
struct CyU3PMbox {  
    uint32_t w0;  
    uint32_t w1;  
};
```

Group

[Mailbox Data Types](#)

Members

Members	Description
uint32_t w0;	Contents of the lower mailbox register.
uint32_t w1;	Contents of the upper mailbox register.







File

cyu3mbox.h

7.1.2 Mailbox Functions

This section documents the functions defined as part of the mailbox handler.


Functions

Function	Description
 CyU3PMboxInit	Initialize the mailbox handler.
 CyU3PMboxDeInit	De-initialize the mailbox handler.
 CyU3PMboxRead	This function reads an incoming mailbox message.
 CyU3PMboxWrite	This function sends a mailbox message to the external processor.
 CyU3PMboxWait	Wait until the Mailbox register to send messages to P-port is free.
 CyU3PMboxReset	Reset the mailbox handler.

Group

[Mailbox Handler](#)

Legend

	Method
---	--------

7.1.2.1 CyU3PMboxInit

Initialize the mailbox handler.

Initiate the mailbox related structures and register a callback function that will be called on every mailbox related interrupt.

Parameters

Parameters	Description
CyU3PMboxCb_t callback	Callback function to be called on interrupt.

C++

```
void CyU3PMboxInit(  
    CyU3PMboxCb\_t callback  
);
```

Group

[Mailbox Functions](#)

See Also

- [CyU3PMboxDelInit](#)

File

cyu3mbox.h

7.1.2.2 CyU3PMboxDeInit

De-initialize the mailbox handler.

Destroys the mailbox related structures.

C++

```
void CyU3PMboxDeInit(  
    void  
);
```

Group

[Mailbox Functions](#)

See Also

- [CyU3PMboxInit](#)

File

cyu3mbox.h

7.1.2.3 CyU3PMboxRead

This function reads an incoming mailbox message.

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.

Parameters

Parameters	Description
CyU3PMbox * mbox	Pointer to buffer to hold the incoming message data.

C++

```
CyU3PReturnStatus_t CyU3PMboxRead(  
    CyU3PMbox * mbox  
) ;
```

Group

[Mailbox Functions](#)

See Also

- [CyU3PMboxWrite](#)

File

cyu3mbox.h

7.1.2.4 CyU3PMboxWrite

This function sends a mailbox message to the external processor.

This function writes 8 bytes of data to the outgoing mailbox registers after ensuring that the external processor has read out the previous message.

Parameters

Parameters	Description
CyU3PMbox * mbox	Pointer to mailbox message data.

C++

```
CyU3PReturnStatus_t CyU3PMboxWrite(  
    CyU3PMbox * mbox  
);
```

Group

[Mailbox Functions](#)

See Also

- [CyU3PMboxRead](#)

File

cyu3mbox.h

7.1.2.5 CyU3PMboxWait

Wait until the Mailbox register to send messages to P-port is free.

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.

Returns

- CY_U3P_SUCCESS - If the operation is successful
- CY_U3P_ERROR_TIMEOUT - If the wait times out.

C++

```
CyU3PReturnStatus\_t CyU3PMboxWait(  
    void  
);
```

Group

[Mailbox Functions](#)

File

cyu3mbox.h

7.1.2.6 CyU3PMboxReset

Reset the mailbox handler.

Clears the data structures and state related to mailbox handler. Can be called for error recovery.

C++

```
void CyU3PMboxReset(  
    void  
) ;
```

Group

[Mailbox Functions](#)

See Also

- [CyU3PMboxInit](#)

File

cyu3mbox.h

7.2 PIB Interface Manager

The PIB interface manager module is responsible for handling the in-bound and out-bound transfer of data through the GPIF-II interface on the FX3 device.

This section defines the data structures and the interfaces for PIB interface management.

Group

[P-port Management](#)




Topics

Topic	Description
P-port Data Types	This section documents the data types defined as part of the P-port interface APIs.
P-port Functions	This section documents the functions defined as part of the P-port interface APIs.

7.2.1 P-port Data Types

This section documents the data types defined as part of the P-port interface APIs.



Enumerations

Enumeration	Description
 CyU3PPibErrorType	Enumeration of P-port error types.
 CyU3PGpifErrorType	Enumeration of GPIF error types.
 CyU3PPibIntrType	Enumeration of P-port interrupt event types.

Group

[PIB Interface Manager](#)


Legend

	Enumeration
	Structure

Macros

Macro	Description
CYU3P_GET_GPIF_ERROR_TYPE	Get the GPIF error code from the CYU3P_PIB_INTR_ERROR callback argument.
CYU3P_GET_PIB_ERROR_TYPE	Get the PIB error code from the CYU3P_PIB_INTR_ERROR callback argument.

Structures

Structure	Description
 CyU3PPibClock_t	Clock configuration information for the PIB block.

Types

Type	Description
CyU3PPibIntrCb_t	Type of callback function that will be called on receiving a generic P-port interrupt.

7.2.1.1 CyU3PPibErrorType

Enumeration of P-port error types.

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.

C++

```
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_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_PIB_ERR_THR1_RD_BURST,
    CYU3P_PIB_ERR_THR2_SCK_INACTIVE = 0x22,
    CYU3P_PIB_ERR_THR2_ADAP_OVERRUN,
    CYU3P_PIB_ERR_THR2_ADAP_UNDERRUN,
    CYU3P_PIB_ERR_THR2_RD_FORCE_END,
    CYU3P_PIB_ERR_THR2_RD_BURST,
    CYU3P_PIB_ERR_THR3_SCK_INACTIVE = 0x2A,
    CYU3P_PIB_ERR_THR3_ADAP_OVERRUN,
    CYU3P_PIB_ERR_THR3_ADAP_UNDERRUN,
    CYU3P_PIB_ERR_THR3_RD_FORCE_END,
    CYU3P_PIB_ERR_THR3_RD_BURST
};
```

Group

[P-port Data Types](#)

See Also

- [CyU3PPibIntrType](#)
- [CyU3PPibRegisterCallback](#)

- [CyU3PGpifErrorType](#)
- [CYU3P_GET_PIB_ERROR_TYPE](#)

Members

Members	Description
CYU3P_PIB_ERR_NONE = 0	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	Bad 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	Bad 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 = 0x12	One of the Thread 0 sockets became inactive during data 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 = 0x1A	One of the Thread 1 sockets became inactive during data 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 = 0x22	One of the Thread 2 sockets became inactive during data 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 = 0x2A	One of the Thread 3 sockets became inactive during data 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.

File

cyu3pib.h

7.2.1.2 CyU3PGpifErrorType

Enumeration of GPIF error types.

In addition to the PIB level errors, there can be cases where GPIF state machine specific errors occur during GPIF 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.

C++

```
enum CyU3PGpifErrorType {  
    CYU3P_GPIF_ERR_NONE = 0,  
    CYU3P_GPIF_ERR_INADDR_OVERWRITE = 0x0400,  
    CYU3P_GPIF_ERR_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  
};
```

Group

[P-port Data Types](#)

See Also

- [CyU3PPibErrorType](#)
- [CYU3P_GET_GPIF_ERROR_TYPE](#)

Members

Members	Description
CYU3P_GPIF_ERR_NONE = 0	No GPIF state machine errors.
CYU3P_GPIF_ERR_INADDR_OVERWRITE = 0x0400	Content of INGRESS_ADDR register is overwritten before read.
CYU3P_GPIF_ERR_EGADDR_INVALID = 0x0800	Attempt to read from EGRESS_ADDR register before it is written to.
CYU3P_GPIF_ERR_DATA_READ_ERR = 0x0C00	Read from DMA data thread which is not ready.
CYU3P_GPIF_ERR_DATA_WRITE_ERR = 0x1000	Write to DMA data thread which is not ready.
CYU3P_GPIF_ERR_ADDR_READ_ERR = 0x1400	Read from DMA address thread which is not ready.
CYU3P_GPIF_ERR_ADDR_WRITE_ERR = 0x1800	Write to DMA address thread which is not ready.
CYU3P_GPIF_ERR_INVALID_STATE = 0x2000	GPIF state machine has reached an invalid state.

File

cyu3pib.h

7.2.1.3 CyU3PPibIntrType

Enumeration of P-port interrupt event types.

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.

C++

```
enum CyU3PPibIntrType {  
    CYU3P_PIB_INTR_DLL_UPDATE = 1,  
    CYU3P_PIB_INTR_PPCONFIG = 2,  
    CYU3P_PIB_INTR_ERROR = 4  
};
```

Group

[P-port Data Types](#)

See Also

- [CyU3PPibIntrCb_t](#)
- [CyU3PPibRegisterCallback](#)

Members

Members	Description
CYU3P_PIB_INTR_DLL_UPDATE = 1	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 = 2	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 = 4	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.

File

cyu3pib.h

7.2.1.4 CyU3PPibIntrCb_t

Type of callback function that will be called on receiving a generic P-port interrupt.

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.

C++

```
typedef void (* CyU3PPibIntrCb_t)(CyU3PPibIntrType cbType, uint16_t cbArg);
```

Group

[P-port Data Types](#)

See Also

- [CyU3PPibRegisterCallback](#)

File

cyu3pib.h

7.2.1.5 CyU3PPibClock_t

Clock configuration information for the PIB block.

The clock for the PIB block can be configured to required frequency The default values can be:

clkDiv = 2 isHalfDiv = CyFalse isDllEnable = CyFalse clkSrc = CY_U3P_SYS_CLK.

C++

```
struct CyU3PPibClock_t {
    uint16_t clkDiv;
    CyBool_t isHalfDiv;
    CyBool_t isDllEnable;
    CyU3PSysClockSrc\_t clkSrc;
};
```

Group

[P-port Data Types](#)

See Also

- [CyU3PSysClockSrc_t](#)
- [CyU3PPibInit](#)

Members

Members	Description
uint16_t clkDiv;	Divider value for the PIB clock. The min value is 2 and max value is 1024.
CyBool_t isHalfDiv;	This allows the clock to be divided by a non integral value of 0.5. This provides divider values from 2.5 to 256.5
CyBool_t isDllEnable;	This enables or disable the PIB DLL control
CyU3PSysClockSrc_t clkSrc;	The clock source to be used for this peripheral.

File

cyu3pib.h

7.2.1.6 CYU3P_GET_GPIF_ERROR_TYPE

Get the GPIF error code from the CYU3P_PIB_INTR_ERROR callback argument.

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.

C++

```
#define CYU3P_GET_GPIF_ERROR_TYPE(param) ((CyU3PGpifErrorType)((param) & 0x7C00))
```

Group

[P-port Data Types](#)

See Also

- [CyU3PGpifErrorType](#)

File

cyu3pib.h

7.2.1.7 CYU3P_GET_PIB_ERROR_TYPE

Get the PIB error code from the CYU3P_PIB_INTR_ERROR callback argument.

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.

C++

```
#define CYU3P_GET_PIB_ERROR_TYPE(param) ((CyU3PPibErrorType)((param) & 0x3F))
```

Group

[P-port Data Types](#)

See Also

- [CyU3PPibErrorType](#)






File

cyu3pib.h

7.2.2 P-port Functions

This section documents the functions defined as part of the P-port interface APIs.

Functions

Function	Description
 CyU3PPibInit	Initialize the P-port interface block.
 CyU3PPibDeInit	De-initialize the P-port interface block.
 CyU3PPibSelectIntSources	Select the sources that trigger an FX3 interrupt to external processor.
 CyU3PPibRegisterCallback	Register a callback function for notification of PIB interrupts.
 CyU3PSetPportDriveStrength	Set the IO drive strength for the Pport.

Group

[PIB Interface Manager](#)

Legend

	Method
---	--------

7.2.2.1 CyU3PPibInit

Initialize the P-port interface block.

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.

Parameters

Parameters	Description
CyBool_t doInit	Whether to initialize the PIB block. Should generally be set to CyTrue.
CyU3PPibClock_t * pibClock	PIB clock configuration

Returns

- CY_U3P_SUCCESS - If the operation is successful
- 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

C++

```
CyU3PReturnStatus_t CyU3PPibInit(  
    CyBool_t doInit,  
    CyU3PPibClock_t * pibClock  
);
```

Group

[P-port Functions](#)

File

cyu3pib.h

7.2.2.2 CyU3PPibDeInit

De-initialize the P-port interface block.

This function disables and powers off the P-port interface block.

C++

```
CyU3PReturnStatus\_t CyU3PPibDeInit(  
    void  
);
```

Group

[P-port Functions](#)

File

cyu3pib.h

7.2.2.3 CyU3PPibSelectIntSources

Select the sources that trigger an FX3 interrupt to external processor.

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.

Parameters

Parameters	Description
CyBool_t pibSockEn	Whether PIB socket DMA ready status should trigger an interrupt.
CyBool_t gpifIntEn	Whether GPIF state machine can trigger an interrupt.
CyBool_t pibErrEn	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.
CyBool_t mboxIntEn	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.
CyBool_t wakeupEn	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.

Returns

None

C++

```
void CyU3PPibSelectIntSources(  
    CyBool_t pibSockEn,  
    CyBool_t gpifIntEn,  
    CyBool_t pibErrEn,  
    CyBool_t mboxIntEn,  
    CyBool_t wakeupEn  
);
```

Group

[P-port Functions](#)

File

cyu3pib.h

7.2.2.4 CyU3PPibRegisterCallback

Register a callback function for notification of PIB interrupts.

This function registers a callback function that will be called for notification of PIB interrupts and also selects the PIB interrupt sources of interest.

Parameters

Parameters	Description
<code>CyU3PPibIntrCb_t cbFunc</code>	Callback function pointer being registered.
<code>uint32_t intMask</code>	Bitmask representing the various interrupts callbacks to be enabled. This value is derived by ORing the various callback types of interest from the <code>CyU3PPibIntrType</code> enumeration.

Returns

None

C++

```
void CyU3PPibRegisterCallback(  
    CyU3PPibIntrCb\_t cbFunc,  
    uint32_t intMask  
);
```

Group

[P-port Functions](#)

See Also

- [CyU3PPibIntrCb_t](#)
- [CyU3PPibIntrType](#)

File

`cyu3pib.h`

7.2.2.5 CyU3PSetPportDriveStrength

Set the IO drive strength for the Pport.

The function sets the IO Drive strength for the Pport. The default IO drive strength for the Pport is set to CY_U3P_DS_THREE_QUARTER_STRENGTH.

Parameters

Parameters	Description
CyU3PDriveStrengthState_t pportDriveStrength	Pport drive strength

Returns

- CY_U3P_SUCCESS - If the operation is successful
- CY_U3P_ERROR_BAD_ARGUMENT - If the Drive strength requested is invalid

C++

```
CyU3PReturnStatus_t CyU3PSetPportDriveStrength(  
    CyU3PDriveStrengthState_t pportDriveStrength  
);
```

Group

[P-port Functions](#)

See Also

- [CyU3PDriveStrengthState_t](#)

File

cyu3pib.h

8 GPIF-II Management

The GPIF-II APIs provide facilities to configure the GPIF behavior and to perform data transfers across the GPIF interface.

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.

Topics

Topic	Description
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.
GPIF Resources	The GPIF block implements multiple counter and comparator elements that can be initialized and controlled using the GPIF API.
GPIF State Machine Control	A set of GPIF APIs are provided to start and control the operation of the GPIF state machine.
GPIF Data Types	This section documents the data types defined and used by the GPIF APIs.
GPIF Functions	This section documents the GPIF specific functions that are part of the FX3 firmware.

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.

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.

The [CyU3PGpifRegisterConfig\(\)](#) function is another mechanism to load the complete GPIF configuration. This function is provided for debug/test functionality and is not intended for general use. The GPIF II designer tool does not support exporting the configuration data in a format suitable for use with this API.

Group

[GPIF-II Management](#)

8.2 GPIF Resources

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

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.

Group

[GPIF-II Management](#)

8.3 GPIF State Machine Control

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

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.




Group

[GPFI-II Management](#)

8.4 GPFI Data Types

This section documents the data types defined and used by the GPFI APIs.



Enumerations

Enumeration	Description
 CyU3PGpifComparatorType	List of supported comparators in the GPFI hardware.
 CyU3PGpifEventType	List of GPFI related events tracked by the driver.
 CyU3PGpifOutput_t	List of control outputs and flags from the GPFI hardware.

Group

[GPFI-II Management](#)



Legend

	Enumeration
	Structure

Macros

Macro	Description
CYU3P_GPFI_NUM_STATES	Number of states supported by the GPFI hardware.
CYU3P_GPFI_INVALID_STATE	Invalid state index for use in state machine control functions.
CYU3P_GPFI_NUM_TRANS_FNS	Number of distinct transfer functions supported by the GPFI hardware.
CYU3P_PIB_MAX_BURST_SETTING	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.
CYU3P_PIB_SOCKET_COUNT	Number of DMA sockets on the GPFI (P-port)
CYU3P_PIB_THREAD_COUNT	Number of DMA threads on the GPFI (P-Port)

Structures

Structure	Description
 CyU3PGpifConfig_t	Structure that holds all configuration inputs for the GPFI hardware.
 CyU3PGpifWaveData	Information on a single GPFI transition from one state to another.

Types

Type	Description
CyU3PGpifEventCb_t	Callback type that is invoked to inform the application about GPFI events.

8.4.1 CYU3P_GPIF_NUM_STATES

Number of states supported by the GPIF hardware.

C++

```
#define CYU3P_GPIF_NUM_STATES (256)
```

Group

[GPIF Data Types](#)

File

cyu3gpif.h

8.4.2 CYU3P_GPIF_INVALID_STATE

Invalid state index for use in state machine control functions.

C++

```
#define CYU3P_GPIF_INVALID_STATE (0xFFFF)
```

Group

[GPIF Data Types](#)

File

cyu3gpif.h

8.4.3 CYU3P_GPIF_NUM_TRANS_FNS

Number of distinct transfer functions supported by the GPIF hardware.

C++

```
#define CYU3P_GPIF_NUM_TRANS_FNS (32)
```

Group

[GPIF Data Types](#)

File

cyu3gpif.h

8.4.4 CYU3P_PIB_MAX_BURST_SETTING

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.

C++

```
#define CYU3P_PIB_MAX_BURST_SETTING (14)
```

Group

[GPIF Data Types](#)

File

cyu3gpif.h

8.4.5 CYU3P_PIB_SOCKET_COUNT

Number of DMA sockets on the GPIF (P-port)

C++

```
#define CYU3P_PIB_SOCKET_COUNT (32)
```

Group

[GPIF Data Types](#)

File

cyu3gpif.h

8.4.6 CYU3P_PIB_THREAD_COUNT

Number of DMA threads on the GPIF (P-Port)

C++

```
#define CYU3P_PIB_THREAD_COUNT (4)
```

Group

[GPIF Data Types](#)

File

cyu3gpif.h

8.4.7 CyU3PGpifEventCb_t

Callback type that is invoked to inform the application about GPIF events.

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.

C++

```
typedef void (* CyU3PGpifEventCb_t)(CyU3PGpifEventType event, uint8_t currentState);
```

Group

[GPIF Data Types](#)

See Also

- [CyU3PGpifRegisterCallback](#)

File

cyu3gpif.h

8.4.8 CyU3PGpifComparatorType

List of supported comparators in the GPIF hardware.

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.

C++

```
enum CyU3PGpifComparatorType {  
    CYU3P_GPIF_COMP_CTRL = 0,  
    CYU3P_GPIF_COMP_ADDR,  
    CYU3P_GPIF_COMP_DATA  
};
```

Group

[GPIF Data Types](#)

See Also

- [CyU3PGpifInitComparator](#)

Members

Members	Description
CYU3P_GPIF_COMP_CTRL = 0	Control signals comparator.
CYU3P_GPIF_COMP_ADDR	Address bus comparator.

CYU3P_GPIF_COMP_DATA	Data bus comparator.
----------------------	----------------------

File

cyu3gpif.h

8.4.9 CyU3PGpifEventType

List of GPIF related events tracked by the driver.

This type lists the various GPIF hardware and state machine related events that may be notified to the user application.

C++

```
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_ADDR_COMP,
    CYU3P_GPIF_EVT_DATA_COMP,
    CYU3P_GPIF_EVT_CTRL_COMP,
    CYU3P_GPIF_EVT_CRC_ERROR
};
```

Group

[GPIF Data Types](#)

Members

Members	Description
CYU3P_GPIF_EVT_END_STATE = 0	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.

File

cyu3gpif.h

8.4.10 CyU3PGpifOutput_t

List of control outputs and flags from the GPIF hardware.

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.

C++

```
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
};
```

Group

[GPIF Data Types](#)

See Also

- [CyU3PGpifOutputConfigure](#)

Members

Members	Description
CYU3P_GPIF_OP_ALPHA0 = 0	User defined Alpha output #0.
CYU3P_GPIF_OP_ALPHA1	User defined Alpha output #1.
CYU3P_GPIF_OP_ALPHA2	User defined Alpha output #2.
CYU3P_GPIF_OP_ALPHA3	User defined Alpha output #3.
CYU3P_GPIF_OP_BETA0 = 8	User defined Beta output #0.
CYU3P_GPIF_OP_BETA1	User defined Beta output #1.
CYU3P_GPIF_OP_BETA2	User defined Beta output #2.
CYU3P_GPIF_OP_BETA3	User defined Beta output #3.
CYU3P_GPIF_OP_THR0_READY = 16	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	Partial (user defined level) flag for Thread 0.
CYU3P_GPIF_OP_THR1_PART	Partial (user defined level) flag for Thread 1.
CYU3P_GPIF_OP_THR2_PART	Partial (user defined level) flag for Thread 2.
CYU3P_GPIF_OP_THR3_PART	Partial (user defined level) flag for Thread 3.
CYU3P_GPIF_OP_DMA_READY	DMA ready flag for the active DMA thread.
CYU3P_GPIF_OP_PARTIAL	Partial (user defined level) flag for the active DMA thread.
CYU3P_GPIF_OP_PPDRQ	PPDRQ interrupt status derived from the PP_DRQR5_MASK register.

File

cyu3gpif.h

8.4.11 CyU3PGpifConfig_t

Structure that holds all configuration inputs for the GPIF hardware.

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.

C++

```
struct CyU3PGpifConfig_t {
    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;
};
```

Group

[GPIF Data Types](#)

See Also

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

Members

Members	Description
const uint16_t stateCount;	Number of states to be initialized.
const CyU3PGpifWaveData * stateData;	Pointer to array containing state descriptors.

<code>const uint8_t * statePosition;</code>	Pointer to array index -> state number mapping.
<code>const uint16_t functionCount;</code>	Number of transition functions to be initialized.
<code>const uint16_t * functionData;</code>	Pointer to array containing transition function data.
<code>const uint16_t regCount;</code>	Number of GPIF config registers to be initialized.
<code>const uint32_t * regData;</code>	Pointer to array containing GPIF register values.

File

cyu3gpif.h

8.4.12 CyU3PGpifWaveData

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

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.

C++

```
struct CyU3PGpifWaveData {
    uint32_t leftData[3];
    uint32_t rightData[3];
};
```

Group[GPIF Data Types](#)**See Also**

- [CyU3PGpifWaveformLoad](#)

Members

Members	Description
<code>uint32_t leftData[3];</code>	12 byte left transition descriptor.
<code>uint32_t rightData[3];</code>	12 byte right transition descriptor.





















File

cyu3gpif.h

8.5 GPIF Functions

This section documents the GPIF specific functions that are part of the FX3 firmware.

Functions

Function	Description
 CyU3PGpifLoad	Function to program the user defined state machine into the GPIF registers.
 CyU3PGpifConfigure	Initialize the GPIF configuration registers.
 CyU3PGpifInitTransFunctions	Initialize the GPIF transition function registers.
 CyU3PGpifWaveformLoad	Initialize the GPIF state machine table.
 CyU3PGpifOutputConfigure	Configure the functionality of a GPIF output pin.
 CyU3PGpifRegisterConfig	Initialize the GPIF configuration registers.
 CyU3PGpifRegisterCallback	This function registers an event callback for the GPIF driver.
 CyU3PGpifSMStart	Start the waveform state machine from the specified state.
 CyU3PGpifGetSMState	Get the current state of the GPIF state machine.
 CyU3PGpifSMControl	Function to pause/resume the GPIF state machine.
 CyU3PGpifSMSwitch	This function is used to start GPIF state machine execution from a desired state.
 CyU3PGpifDisable	Disables the GPIF state machine and hardware.
 CyU3PGpifReadDataWords	Read a specified number of data words from the GPIF interface.
 CyU3PGpifWriteDataWords	Write a specified number of data words to the GPIF interface.
 CyU3PGpifControlSWInput	Function to set/clear the software controlled state machine input.
 CyU3PGpifInitAddrCounter	Function to configure the GPIF address counter.
 CyU3PGpifInitDataCounter	Function to configure the GPIF data counter.
 CyU3PGpifInitCtrlCounter	Function to configure the GPIF control counter.
 CyU3PGpifInitComparator	This function configures one of the comparators in the GPIF hardware.
 CyU3PGpifSocketConfigure	Function to select an active socket on the P-port and to configure it.

Group

[GPIF-II Management](#)

Legend

	Method
---	--------

8.5.1 CyU3PGpifLoad

Function to program the user defined state machine into the GPIF registers.

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.

Parameters

Parameters	Description
<code>const CyU3PGpifConfig_t * conf</code>	Pointer to GPIF configuration structure.

C++

```
CyU3PReturnStatus_t CyU3PGpifLoad(  
    const CyU3PGpifConfig_t * conf  
) ;
```

Group

[GPIF Functions](#)

See Also

- [CyU3PGpifConfig_t](#)

Return Values

- 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.

File

cyu3gpif.h

8.5.2 CyU3PGpifConfigure

Initialize the GPIF configuration registers.

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.

Parameters

Parameters	Description
uint8_t numRegs	Number of registers to configure (size of array).
const uint32_t * regData	Pointer to uint32_t[] array, where element [i] contains the data to be loaded into GPIF config register "i".

Returns

- 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.

C++

```
CyU3PReturnStatus\_t CyU3PGpifConfigure(
    uint8_t numRegs,
    const uint32_t * regData
);
```

Group

[GPIF Functions](#)

See Also

- [CyU3PGpifWaveformLoad](#)
- [CyU3PGpifInitTransFunctions](#)
- [CyU3PGpifRegisterConfig](#)

File

cyu3gpif.h

8.5.3 CyU3PGpifInitTransFunctions

Initialize the GPIF transition function registers.

This function initializes the GPIF transition function registers with data provided by the GPIF designer tool.

Parameters

Parameters	Description
uint16_t * fnTable	Pointer to the table (array) containing the values with which the registers should be initialized.

Returns

- 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.

C++

```
CyU3PReturnStatus\_t CyU3PGpifInitTransFunctions(
    uint16_t * fnTable
);
```

Group

[GPIF Functions](#)

See Also

- [CyU3PGpifConfigure](#)

File

cyu3gpif.h

8.5.4 CyU3PGpifWaveformLoad

Initialize the GPIF state machine table.

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 GPIF 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 multiple disjoint state machine configurations into distinct parts of the waveform memory by making multiple calls to this API. Each call needs to specify a distinct range of states to be initialized.

Parameters

Parameters	Description
uint8_t firstState	The first state to be initialized in this function call.
uint16_t stateCnt	Number of states to be initialized with data.
uint8_t * stateDataMap	Table that maps state indices to the descriptor table indices.
CyU3PGpifWaveData * transitionData	Table containing the transition information for various states.

Returns

- 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.

C++

```
CyU3PReturnStatus_t CyU3PGpifWaveformLoad(  
    uint8_t firstState,  
    uint16_t stateCnt,  
    uint8_t * stateDataMap,  
    CyU3PGpifWaveData * transitionData  
);
```

Group

[GPIF Functions](#)

See Also

- [CyU3PGpifWaveData](#)

File

cyu3gpif.h

8.5.5 CyU3PGpifOutputConfigure

Configure the functionality of a GPIF output pin.

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.

Parameters

Parameters	Description
uint8_t ctrlPin	Control pin number to be configured.
CyU3PGpifOutput_t opFlag	Selected output flag.
CyBool_t isActiveLow	Polarity: 0=Active high, 1=Active low.

C++

```
CyU3PReturnStatus\_t CyU3PGpifOutputConfigure(  
    uint8_t ctrlPin,  
    CyU3PGpifOutput\_t opFlag,  
    CyBool_t isActiveLow  
);
```

Group

[GPIF Functions](#)

Notes

This configuration is likely to be overridden by any GPIF configuration API calls such as [CyU3PGpifLoad](#) or [CyU3PGpifRegisterConfig](#).

Return Values

- 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](#)

File

cyu3gpif.h

8.5.6 CyU3PGpifRegisterConfig

Initialize the GPIF configuration registers.

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.

Parameters

Parameters	Description
uint16_t numRegs	Number of registers to configure (size of the matrix).
uint32_t regData[][2]	Reference to array of {address, data} tuples corresponding to GPIF registers to initialize.

Returns

- 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.

C++

```
CyU3PReturnStatus\_t CyU3PGpifRegisterConfig(  
    uint16_t numRegs,  
    uint32_t regData[][2]  
);
```

Group

[GPIF Functions](#)

See Also

- [CyU3PGpifConfigure](#)

File

cyu3gpif.h

8.5.7 CyU3PGpifRegisterCallback

This function registers an event callback for the GPIF driver.

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.

Parameters

Parameters	Description
CyU3PGpifEventCb_t cbFunc	Pointer to callback function.

C++

```
void CyU3PGpifRegisterCallback(
    CyU3PGpifEventCb\_t cbFunc
);
```

Group

[GPIF Functions](#)

See Also

- [CyU3PGpifEventCb_t](#)
- [CyU3PGpifEventType](#)

Return Values None

File

cyu3gpif.h

8.5.8 CyU3PGpifSMStart

Start the waveform state machine from the specified state.

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.

Parameters

Parameters	Description
uint8_t stateIndex	State from which to start execution. This should be 0 in most cases.
uint8_t initialAlpha	Initial alpha values to start GPIF state machine operation with.

Returns

- 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.

C++

```
CyU3PReturnStatus\_t CyU3PGpifSMStart(
    uint8_t stateIndex,
    uint8_t initialAlpha
```

```
);
```

Group

[GPIF Functions](#)

See Also

- [CyU3PGpifSMSwitch](#)

File

cyu3gpif.h

8.5.9 CyU3PGpifGetSMState

Get the current state of the GPIF state machine.

This function is used to fetch the current state of the GPIF state machine. This is primarily used to provide debug information.

Return Values

- 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.

Parameters

Parameters	Description
uint8_t * curState_p	Return parameter to be filled with current state information.

C++

```
CyU3PReturnStatus\_t CyU3PGpifGetSMState(  
    uint8_t * curState_p  
);
```

Group

[GPIF Functions](#)

See Also

- [CyU3PGpifSMStart](#)
- [CyU3PGpifSMSwitch](#)

File

cyu3gpif.h

8.5.10 CyU3PGpifSMControl

Function to pause/resume the GPIF state machine.

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 Values

- CY_U3P_SUCCESS if successful.
- CY_U3P_ERROR_NOT_CONFIGURED if the state machine has not been configured.

Parameters

Parameters	Description
CyBool_t pause	Whether to pause the state machine or resume it.

C++

```
CyU3PReturnStatus\_t CyU3PGpifSMControl(  
    CyBool_t pause  
);
```

Group

[GPIF Functions](#)

File

cyu3gpif.h

8.5.11 CyU3PGpifSMSwitch

This function is used to start GPIF state machine execution from a desired state.

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.

Parameters

Parameters	Description
uint16_t fromState	The state from which to do the switch to the desired state.
uint16_t toState	The state to which to transition from fromState.
uint16_t endState	The end state for this execution path.
uint8_t initialAlpha	Initial Alpha values to use when switching states.
uint32_t switchTimeout	Timeout setting for the switch operation in GPIF clock cycles.

C++

```
CyU3PReturnStatus\_t CyU3PGpifSMSwitch(  
    uint16_t fromState,  
    uint16_t toState,  
    uint16_t endState,  
    uint8_t initialAlpha,  
    uint32_t switchTimeout  
);
```

Group

[GPIF Functions](#)

See Also

- [CyU3PGpifSMStart](#)

Return Values

- CY_U3P_SUCCESS
- CY_U3P_ERROR_BAD_ARGUMENT

File

cyu3gpif.h

8.5.12 CyU3PGpifDisable

Disables the GPIF state machine and hardware.

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.

Parameters

Parameters	Description
CyBool_t forceReload	Whether a GPIF re-configuration is to be forced.

Returns

None

C++

```
void CyU3PGpifDisable(
    CyBool_t forceReload
);
```

Group

[GPIF Functions](#)

See Also

- [CyU3PGpifConfigure](#)

File

cyu3gpif.h

8.5.13 CyU3PGpifReadDataWords

Read a specified number of data words from the GPIF interface.

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 Values

- 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

Parameters	Description
uint32_t threadIndex	DMA thread index from which to read data.
CyBool_t selectThread	Whether the target thread should be enabled explicitly.
uint32_t numWords	Number of words of data to read.
uint32_t * buffer_p	Buffer pointer into which the data should be read.
uint32_t waitOption	Timeout duration to wait for data.

C++

```
CyU3PReturnStatus\_t CyU3PGpifReadDataWords(  
    uint32_t threadIndex,  
    CyBool_t selectThread,  
    uint32_t numWords,  
    uint32_t * buffer_p,  
    uint32_t waitOption  
);
```

Group

[GPIF Functions](#)

File

cyu3gpif.h

8.5.14 CyU3PGpifWriteDataWords

Write a specified number of data words to the GPIF interface.

The GPIF hardware supports a mode where data can be written to the P-port in terms of words of the specified interface width. The data can be written to any one of the four threads of data transfer across the P-port.

This function is used to write a specified number of data words from a buffer, one word at a time. Please note that each data word in the buffer is expected to be padded to 32 bits. A timeout period can be specified, and if any of the data words are not sent out within the specified period from the previous one, the operation will return with a timeout error.

Return Values

- CY_U3P_SUCCESS if successful.

Parameters

Parameters	Description
uint32_t threadIndex	DMA thread index through which to write data.
CyBool_t selectThread	Whether the target thread should be enabled explicitly.
uint32_t numWords	Number of words of data to write.
uint32_t * buffer_p	Pointer to buffer containing the data.
uint32_t waitOption	Timeout duration to wait for data sending.

C++

```
CyU3PReturnStatus\_t CyU3PGpifWriteDataWords(  
    uint32_t threadIndex,  
    CyBool_t selectThread,  
    uint32_t numWords,  
    uint32_t * buffer_p,  
    uint32_t waitOption  
);
```

Group

[GPIF Functions](#)

File

cyu3gpif.h

8.5.15 CyU3PGpifControlSWInput

Function to set/clear the software controlled state machine input.

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 Values None

Parameters

Parameters	Description
CyBool_t set	Whether to set (assert) the software input signal.

C++

```
void CyU3PGpifControlSWInput(
    CyBool_t set
);
```

Group

[GPIF Functions](#)

File

cyu3gpif.h

8.5.16 CyU3PGpifInitAddrCounter

Function to configure the GPIF address counter.

This function is used to configure the GPIF address counter with the desired initial value, limit and counting mode.

Return Values None

Parameters

Parameters	Description
uint32_t initValue	Initial value to start the counter from.
uint32_t limit	Counter limit at which the counter stops.

CyBool_t reload	Whether to reload the counter when the limit is reached.
CyBool_t upCount	Whether to count upwards from the initial value.
uint8_t increment	The value to be incremented/decremented from the counter at each step.

C++

```
void CyU3PGpifInitAddrCounter(
    uint32_t initValue,
    uint32_t limit,
    CyBool_t reload,
    CyBool_t upCount,
    uint8_t increment
);
```

Group

[GPIF Functions](#)

File

cyu3gpif.h

8.5.17 CyU3PGpifInitDataCounter

Function to configure the GPIF data counter.

This function is used to configure the GPIF data counter with the desired initial value, limit and counting mode.

Return Values None

Parameters

Parameters	Description
uint32_t initValue	Initial value to start the counter from.
uint32_t limit	Counter limit at which the counter stops.
CyBool_t reload	Whether to reload the counter when the limit is reached.
CyBool_t upCount	Whether to count upwards from the initial value.
uint8_t increment	The value to be incremented/decremented from the counter at each step.

C++

```
void CyU3PGpifInitDataCounter(
    uint32_t initValue,
    uint32_t limit,
    CyBool_t reload,
    CyBool_t upCount,
    uint8_t increment
);
```

Group

[GPIF Functions](#)

File

cyu3gpif.h

8.5.18 CyU3PGpifInitCtrlCounter

Function to configure the GPIF control counter.

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 Values None

Parameters

Parameters	Description
uint16_t initValue	Initial (reset) value for the counter.
uint16_t limit	Value at which to stop the counter and flag an event.
CyBool_t reload	Whether to reload the counter and continue after limit is hit.
CyBool_t upCount	Whether to count upwards from the initial value.
uint8_t outputBit	Selects counter bit to be connected to CTRL[9] output.

C++

```
void CyU3PGpifInitCtrlCounter(
    uint16_t initValue,
    uint16_t limit,
    CyBool_t reload,
    CyBool_t upCount,
    uint8_t outputBit
);
```

Group

[GPIF Functions](#)

File

cyu3gpif.h

8.5.19 CyU3PGpifInitComparator

This function configures one of the comparators in the GPIF hardware.

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.

Parameters

Parameters	Description
CyU3PGpifComparatorType type	The type of comparator to configure.
uint32_t value	The value to compare the signals against.
uint32_t mask	Mask that specifies which bits are to be used in the comparison.

C++

```
CyU3PReturnStatus_t CyU3PGpifInitComparator(  
    CyU3PGpifComparatorType type,  
    uint32_t value,  
    uint32_t mask  
);
```

Group

[GPIF Functions](#)

See Also

- [CyU3PGpifComparatorType](#)

Return Values

- CY_U3P_SUCCESS if successful.
- CY_U3P_ERROR_BAD_ARGUMENT if the type specified is invalid.

File

cyu3gpif.h

8.5.20 CyU3PGpifSocketConfigure

Function to select an active socket on the P-port and to configure it.

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 Values

- 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

Parameters	Description
uint8_t threadIndex	Thread index whose active socket is to be configured.
CyU3PDmaSocketId_t socketNum	The socket to be associated with this thread.
uint16_t watermark	Watermark level for this socket in number of 4-byte words.
CyBool_t flagOnData	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.
uint8_t burst	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.

C++

```
CyU3PReturnStatus_t CyU3PGpifSocketConfigure(
    uint8_t threadIndex,
    CyU3PDmaSocketId_t socketNum,
    uint16_t watermark,
    CyBool_t flagOnData,
    uint8_t burst
);
```

Group

[GPIF Functions](#)

File

cyu3gpif.h

9 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.

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. GPIOs
4. SPI
5. I2S

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.

The FX3 device supports configuration of several IO pins as general purpose IOs, which can be multiplexed to support other functions and interfaces. FX3 provides software controlled pull up or pull down resistors internally on all digital I/O pins. The pins can be pulled high through a resistor on all I/O pins or can be pulled low through a resistor on all I/O pins and can be used to prevent the pins from floating.

Topics

Topic	Description
UART Interface	The UART interface driver and APIs in the FX3 library provide a mechanism to configure the UART properties and to do data read/write transfers through the UART interface.
I2C Interface	The FX3 API library includes a I2C interface driver and provides a set of APIs that allow the configuration of the I2C interface properties and data exchange with one or more I2C slave devices.
SPI Interface	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.
I2S Interface	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 and send out audio data. The I2S driver module provides functions to configure the I2S interface and to send mono or stereo audio output on the I2S link.
GPIO Interface	<p>The GPIO interface manager module is responsible for handling general purpose IO pins. This section defines the data structures and software interfaces for GPIO interface management.</p> <p>GPIO(general purpose I/O) pins are a special (simple) case of low performance serial peripherals that do not need DMA capability. Two modes of GPIO pins are available with FX3 devices - Simple and Complex GPIOs.</p> <p>Simple GPIO provides software controlled and observable input and output capability only. Complex GPIO's contain a timer and support a variety of timed behaviors (pulsing, time measurements, one-shot etc.).</p> <p>The GPIOs can only be configured individually and multiple GPIOs... more</p>

9.1 UART Interface

The UART interface driver and APIs in the FX3 library provide a mechanism to configure the UART properties and to do data read/write transfers through the UART interface.

Group

[Serial Peripheral Interfaces](#)






Topics

Topic	Description
UART data types	This section documents the data types that are defined as part of the UART driver and API library.
UART Functions	This section documents the functions that are defined as part of the UART driver and API library.

9.1.1 UART data types

This section documents the data types that are defined as part of the UART driver and API library.



Enumerations

Enumeration	Description
 CyU3PUartBaudrate_t	List of baud rates supported by the UART.
 CyU3PUartParity_t	List of parity settings supported by the UART interface.
 CyU3PUartStopBit_t	List of number of stop bits to be used in UART communication.
 CyU3PUartError_t	List of UART specific error/status codes.
 CyU3PUartEvt_t	List of UART related event types.


Group

[UART Interface](#)

Legend

	Enumeration
	Structure

Structures

Structure	Description
 CyU3PUartConfig_t	Configuration parameters for the UART interface.

Types

Type	Description
CyU3PUartIntrCb_t	Prototype of UART event callback function.

9.1.1.1 CyU3PUartBaudrate_t

List of baud rates supported by the UART.

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 acheived for both 403.2MHz and 416MHz SYS_CLK is listed below.

C++

```
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
};
```

Group

[UART data types](#)

See Also

- [CyU3PUartConfig_t](#)

Members

Members	Description
CY_U3P_UART_BAUDRATE_100 = 100	Baud: 100, Actual @403MHz: 100.00, @416MHz: 100.00,

CY_U3P_UART_BAUDRATE_300 = 300	Baud: 300, Actual @403MHz: 300.00, @416MHz: 300.01,
CY_U3P_UART_BAUDRATE_600 = 600	Baud: 600, Actual @403MHz: 600.00, @416MHz: 599.99
CY_U3P_UART_BAUDRATE_1200 = 1200	Baud: 1200, Actual @403MHz: 1200.00, @416MHz: 1200.01
CY_U3P_UART_BAUDRATE_2400 = 2400	Baud: 2400, Actual @403MHz: 2400.00, @416MHz: 2399.96
CY_U3P_UART_BAUDRATE_4800 = 4800	Baud: 4800, Actual @403MHz: 4800.00, @416MHz: 4800.15
CY_U3P_UART_BAUDRATE_9600 = 9600	Baud: 9600, Actual @403MHz: 9600.00, @416MHz: 9599.41
CY_U3P_UART_BAUDRATE_10000 = 10000	Baud: 10000, Actual @403MHz: 10000.00, @416MHz: 10000.00
CY_U3P_UART_BAUDRATE_14400 = 14400	Baud: 14400, Actual @403MHz: 14400.00, @416MHz: 14400.44
CY_U3P_UART_BAUDRATE_19200 = 19200	Baud: 19200, Actual @403MHz: 19200.00, @416MHz: 19202.36
CY_U3P_UART_BAUDRATE_38400 = 38400	Baud: 38400, Actual @403MHz: 38385.38, @416MHz: 38404.73
CY_U3P_UART_BAUDRATE_50000 = 50000	Baud: 50000, Actual @403MHz: 50000.00, @416MHz: 50000.00
CY_U3P_UART_BAUDRATE_57600 = 57600	Baud: 57600, Actual @403MHz: 57600.00, @416MHz: 57585.83
CY_U3P_UART_BAUDRATE_75000 = 75000	Baud: 75000, Actual @403MHz: 75000.00, @416MHz: 75036.08
CY_U3P_UART_BAUDRATE_100000 = 100000	Baud: 100000, Actual @403MHz: 100000.00, @416MHz: 100000.00
CY_U3P_UART_BAUDRATE_115200 = 115200	Baud: 115200, Actual @403MHz: 115068.49, @416MHz: 115299.33
CY_U3P_UART_BAUDRATE_153600 = 153600	Baud: 153600, Actual @403MHz: 153658.54, @416MHz: 153392.33
CY_U3P_UART_BAUDRATE_200000 = 200000	Baud: 200000, Actual @403MHz: 200000.00, @416MHz: 200000.00
CY_U3P_UART_BAUDRATE_225000 = 225000	Baud: 225000, Actual @403MHz: 225000.00, @416MHz: 225108.23
CY_U3P_UART_BAUDRATE_230400 = 230400	Baud: 230400, Actual @403MHz: 230136.99, @416MHz: 230088.50
CY_U3P_UART_BAUDRATE_300000 = 300000	Baud: 300000, Actual @403MHz: 300000.00, @416MHz: 300578.03
CY_U3P_UART_BAUDRATE_400000 = 400000	Baud: 400000, Actual @403MHz: 400000.00, @416MHz: 400000.00
CY_U3P_UART_BAUDRATE_460800 = 460800	Baud: 460800, Actual @403MHz: 462385.32, @416MHz: 460176.99
CY_U3P_UART_BAUDRATE_500000 = 500000	Baud: 500000, Actual @403MHz: 499009.90, @416MHz: 500000.00
CY_U3P_UART_BAUDRATE_750000 = 750000	Baud: 750000, Actual @403MHz: 752238.81, @416MHz: 753623.19
CY_U3P_UART_BAUDRATE_921600 = 921600	Baud: 921600, Actual @403MHz: 916363.64, @416MHz: 928571.43
CY_U3P_UART_BAUDRATE_1M = 1000000	Baud: 1000000, Actual @403MHz: 1008000.00, @416MHz: 1000000.00
CY_U3P_UART_BAUDRATE_2M = 2000000	Baud: 2000000, Actual @403MHz: 2016000.00, @416MHz: 2000000.00

Serial Peripheral Interfaces

UART Interface

CY_U3P_UART_BAUDRATE_3M = 3000000	Baud: 3000000, Actual @403MHz: 2964705.88, @416MHz: 3058823.52
CY_U3P_UART_BAUDRATE_4M = 4000000	Baud: 4000000, Actual @403MHz: 3876923.08, @416MHz: 4000000.00
CY_U3P_UART_BAUDRATE_4M608K = 4608000	Baud: 4608000, Actual @403MHz: 4581818.18, @416MHz: 4727272.72

File

cyu3uart.h

9.1.1.2 CyU3PUartParity_t

List of parity settings supported by the UART interface.

This enumeration lists the various parity settings that the UART interface can be configured to support.

C++

```
enum CyU3PUartParity_t {
    CY_U3P_UART_NO_PARITY = 0,
    CY_U3P_UART_EVEN_PARITY,
    CY_U3P_UART_ODD_PARITY,
    CY_U3P_UART_NUM_PARITY
};
```

Group

[UART data types](#)

See Also

- [CyU3PUartConfig_t](#)

Members

Members	Description
CY_U3P_UART_NO_PARITY = 0	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.

File

cyu3uart.h

9.1.1.3 CyU3PUartStopBit_t

List of number of stop bits to be used in UART communication.

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.

C++

```
enum CyU3PUartStopBit_t {  
    CY_U3P_UART_ONE_STOP_BIT = 1,  
    CY_U3P_UART_TWO_STOP_BIT = 2  
};
```

Group

[UART data types](#)

See Also

- [CyU3PUartConfig_t](#)

Members

Members	Description
CY_U3P_UART_ONE_STOP_BIT = 1	1 stop bit
CY_U3P_UART_TWO_STOP_BIT = 2	2 stop bit

File

cyu3uart.h

9.1.1.4 CyU3PUartError_t

List of UART specific error/status codes.

This type lists the various UART specific error/status codes that are sent to the event callback as event data, when the event type is CY_U3P_UART_ERROR_EVT.

C++

```
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
};
```

Group

[UART data types](#)

See Also

- [CyU3PUartEvt_t](#)
- [CyU3PUartIntrCb_t](#)

Members

Members	Description
CY_U3P_UART_ERROR_NAK_BYTE_0 = 0	Missing stop bit.
CY_U3P_UART_ERROR_RX_PARITY_ERROR = 1	RX parity error.
CY_U3P_UART_ERROR_TX_OVERFLOW = 12	Overflow of FIFO during transmit operation.
CY_U3P_UART_ERROR_RX_UNDERFLOW = 13	Underflow in FIFO during receive/read operation.
CY_U3P_UART_ERROR_RX_OVERFLOW = 14	Overflow of FIFO during receive operation.

File

cyu3uart.h

9.1.1.5 CyU3PUartEvt_t

List of UART related event types.

This enumeration lists the various UART related event codes that are notified to the user application through an event callback.

C++

```
enum CyU3PUartEvt_t {  
    CY_U3P_UART_EVENT_RX_DONE = 0,  
    CY_U3P_UART_EVENT_TX_DONE,  
    CY_U3P_UART_EVENT_ERROR  
};
```

Group

[UART data types](#)

See Also

- [CyU3PUartError_t](#)
- [CyU3PUartIntrCb_t](#)

Members

Members	Description
CY_U3P_UART_EVENT_RX_DONE = 0	Reception is completed
CY_U3P_UART_EVENT_TX_DONE	Transmission is done
CY_U3P_UART_EVENT_ERROR	Error has happened

File

cyu3uart.h

9.1.1.6 CyU3PUartIntrCb_t

Prototype of UART event callback function.

This function type defines a callback to be called after UART interrupt has been 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.

C++

```
typedef void (* CyU3PUartIntrCb_t)(CyU3PUartEvt\_t evt, CyU3PUartError\_t error);
```

Group

[UART data types](#)

See Also

- [CyU3PUartEvt_t](#)
- [CyU3PUartError_t](#)
- [CyU3PRegisterUartCallBack](#)

File

cyu3uart.h

9.1.1.7 CyU3PUartConfig_t

Configuration parameters for the UART interface.

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 [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 large (user configurable size) blocks.

All of the parameters can be changed dynamically by calling the [CyU3PUartSetConfig](#) function repeatedly.

C++

```
struct CyU3PUartConfig_t {
    CyBool_t txEnable;
    CyBool_t rxEnable;
    CyBool_t flowCtrl;
    CyBool_t isDma;
    CyU3PUartBaudrate\_t baudRate;
    CyU3PUartStopBit\_t stopBit;
    CyU3PUartParity\_t parity;
};
```

Group

[UART data types](#)

See Also

- [CyU3PUartBaudrate_t](#)
- [CyU3PUartStopBit_t](#)
- [CyU3PUartParity_t](#)
- [CyU3PUartSetConfig](#)

Members

Members	Description
CyBool_t txEnable;	Enable the transmitter.
CyBool_t rxEnable;	Enable the receiver.
CyBool_t flowCtrl;	Enable hardware flow control for Both RX and TX.
CyBool_t isDma;	CyFalse: Byte by byte transfer; CyTrue: Block based transfer.
CyU3PUartBaudrate_t baudRate;	Baud rate for data transfer.
CyU3PUartStopBit_t stopBit;	The number of stop bits appended.
CyU3PUartParity_t parity;	Parity configuration











File

cyu3uart.h

9.1.2 UART Functions

This section documents the functions that are defined as part of the UART driver and API library.


Functions

Function	Description
 CyU3PUartSetClock	Function sets the required frequency for the UART block.
 CyU3PUartStopClock	This function disables the clock of the UART.
 CyU3PUartInit	Starts the UART hardware block on the device.
 CyU3PUartDeInit	Stops the UART hardware block.
 CyU3PUartSetConfig	Sets the UART interface parameters.
 CyU3PUartTransmitBytes	Transmits data through the UART interface on a byte by byte basis.
 CyU3PUartReceiveBytes	Receives data from the UART interface on a byte by byte basis.
 CyU3PUartTxSetBlockXfer	Sets the number of bytes to be transmitted by the UART.
 CyU3PUartRxSetBlockXfer	Sets the number of bytes to be received by the UART.
 CyU3PRegisterUartCallBack	This function register the call back function for notification of UART interrupt.

Group

[UART Interface](#)

Legend

	Method
---	--------

9.1.2.1 CyU3PUartSetClock

Function sets the required frequency for the UART block.

The clock for the UART block can be configured to required frequency.

Baudrate calculation: The maximum baud rate supported is 4MHz and the minimum is 100Hz. The UART block requires an internal clocking of 16X. It should be noted that even though the clock dividers and the API allows frequencies above and below the rated range, the device behaviour is not guaranteed. The actual clock rate is derived out of SYS_CLK for all baud rates ≥ 600 . For rates below this SYS_CLK / 16 is used. Since the divider needs to be integral or with half divider, the frequency approximation is done using the following algorithm:

If x is the actual divider and n is the required integral divider to be used, then following conditions are used to evaluate:

$x = (\text{source clock}) / (\text{baudrate} * 16)$; source clock = SYS_CLK for baud ≥ 600 and source clock = (SYS_CLK / 16) for baud below 600.

if $(x - \text{floor}(x)) < 0.25 \implies n = \text{floor}(x)$;

if $((x - \text{floor}(x)) \geq 0.25) \ \&\& \ (x - \text{floor}(x)) < 0.5 \implies n = \text{floor}(x) + \text{half divider}$;

if $(x - \text{floor}(x)) \geq 0.75 \implies n = \text{floor}(x) + 1$;

Parameters

Parameters	Description
uint32_t baudRate	Desired baud rate for the UART interface.

Returns

- 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
- CY_U3P_ERROR_NOT_SUPPORTED - If the FX3 part in use does not support the UART interface

C++

```
CyU3PReturnStatus\_t CyU3PUartSetClock(  
    uint32_t baudRate  
);
```

Group

[UART Functions](#)

See Also

- NONE

File

cyu3lpp.h

9.1.2.2 CyU3PUartStopClock

This function disables the clock of the UART.

Disable the clock to the UART block. This needs to be done after the UART driver deinit is complete.

Returns

- CY_U3P_SUCCESS - if the UART clock was successfully turned off.
- CY_U3P_ERROR_NOT_SUPPORTED - if the UART interface is not supported on the FX3 part.

C++

```
CyU3PReturnStatus\_t CyU3PUartStopClock(  
    void  
);
```

Group

[UART Functions](#)

See Also

- [CyU3PUartSetClock](#)

File

cyu3lpp.h

9.1.2.3 CyU3PUartInit

Starts the UART hardware block on the device.

This function powers up the UART hardware block on the device and should be the first UART related function called by the application.

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PUartInit(  
    void  
);
```

Group

[UART Functions](#)

See Also

- [CyU3PUartDeInit](#)
- [CyU3PUartSetConfig](#)
- [CyU3PUartTransmitBytes](#)
- [CyU3PUartReceiveBytes](#)
- [CyU3PUartTxSetBlockXfer](#)
- [CyU3PUartRxSetBlockXfer](#)

File

cyu3uart.h

9.1.2.4 CyU3PUartDeInit

Stops the UART hardware block.

This function disables and powers off the UART hardware block on the device.

Returns

- CY_U3P_SUCCESS - if the de-init was successful
- CY_U3P_ERROR_NOT_STARTED - if UART was not initialized

C++

```
CyU3PReturnStatus\_t CyU3PUartDeInit(  
    void  
);
```

Group

[UART Functions](#)

See Also

- [CyU3PUartInit](#)
- [CyU3PUartSetConfig](#)
- [CyU3PUartTransmitBytes](#)
- [CyU3PUartReceiveBytes](#)
- [CyU3PUartTxSetBlockXfer](#)
- [CyU3PUartRxSetBlockXfer](#)

File

cyu3uart.h

9.1.2.5 CyU3PUartSetConfig

Sets the UART interface parameters.

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 [CyU3PUartInit](#). But this will reset the FIFO and hence the data in pipe will be lost. If a DMA channel is present, a Reset has to be issued.

Baudrate calculation: The maximum baud rate supported is 4MHz and the minimum is 100Hz. The UART block requires an internal clocking of 16X. It should be noted that even though the clock dividers and the API allows frequencies above and below the rated range, the device behaviour is not guaranteed. The actual clock rate is derived out of SYS_CLK for all baud rates ≥ 600 . For rates below this $\text{SYS_CLK} / 16$ is used. Since the divider needs to be integral or with half divider, the frequency approximation is done using the following algorithm:

If x is the actual divider and n is the required integral divider to be used, then following conditions are used to evaluate:

$x = (\text{source clock}) / (\text{baudrate} * 16)$; // source clock = SYS_CLK for baud ≥ 600 and source clock = $(\text{SYS_CLK} / 16)$ for baud below 600.

if $(x - \text{floor}(x)) < 0.25 \implies n = \text{floor}(x)$;

if $((x - \text{floor}(x)) \geq 0.25) \ \&\& \ (x - \text{floor}(x)) < 0.5 \implies n = \text{floor}(x) + \text{half divider}$;

if $(x - \text{floor}(x)) \geq 0.75 \implies n = \text{floor}(x) + 1$;

Parameters

Parameters	Description
CyU3PUartConfig_t * config	Pointer to structure containing config information
CyU3PUartIntrCb_t cb	Callback for getting the events

Returns

- 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 - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus\_t CyU3PUartSetConfig(  
    CyU3PUartConfig\_t * config,  
    CyU3PUartIntrCb\_t cb  
);
```

Group

[UART Functions](#)

See Also

- [CyU3PUartEvt_t](#)
- [CyU3PUartError_t](#)
- [CyU3PUartParity_t](#)
- [CyU3PUartConfig_t](#)
- [CyU3PUartIntrCb_t](#)
- [CyU3PUartStopBit_t](#)
- [CyU3PUartBaudrate_t](#)
- [CyU3PUartInit](#)
- [CyU3PUartDeInit](#)
- [CyU3PUartTransmitBytes](#)
- [CyU3PUartReceiveBytes](#)
- [CyU3PUartTxSetBlockXfer](#)
- [CyU3PUartRxSetBlockXfer](#)

File

`cyu3uart.h`

9.1.2.6 CyU3PUartTransmitBytes

Transmits data through the UART interface on a byte by byte basis.

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.

Parameters

Parameters	Description
uint8_t * data_p	Pointer to the data to be transferred.
uint32_t count	Number of bytes to be transferred.
CyU3PReturnStatus_t * status	Status returned from the operation

Returns

Number of bytes that are successfully transferred.

status argument can take following values:

- CY_U3P_SUCCESS - if "count" bytes are transmitted successfully
- CY_U3P_ERROR_NULL_POINTER - if the data pointer is NULL
- CY_U3P_ERROR_BAD_ARGUMENT - if the count is zero
- CY_U3P_ERROR_TIMEOUT - if the data transfer times out
- CY_U3P_ERROR_NOT_CONFIGURED - if UART was configured for DMA mode of transfers OR if the UART was not configured or initialized
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
uint32_t CyU3PUartTransmitBytes(  
    uint8_t * data_p,  
    uint32_t count,  
    CyU3PReturnStatus\_t * status  
);
```

Group

[UART Functions](#)

See Also

- [CyU3PUartInit](#)
- [CyU3PUartDeInit](#)
- [CyU3PUartSetConfig](#)
- [CyU3PUartReceiveBytes](#)
- [CyU3PUartTxSetBlockXfer](#)
- [CyU3PUartRxSetBlockXfer](#)

File

cyu3uart.h

9.1.2.7 CyU3PUartReceiveBytes

Receives data from the UART interface on a byte by byte basis.

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.

Parameters

Parameters	Description
uint8_t * data_p	Pointer to location where the data read is to be placed.
uint32_t count	Number of bytes to be received.
CyU3PReturnStatus_t * status	Status returned from the operation

Returns

Number of bytes that are successfully received.

status argument can take following values:

- CY_U3P_SUCCESS - if "count" bytes are received successfully
- CY_U3P_ERROR_NULL_POINTER - if the data pointer is NULL
- CY_U3P_ERROR_BAD_ARGUMENT - if the count is zero
- CY_U3P_ERROR_TIMEOUT - if the data transfer times out
- CY_U3P_ERROR_NOT_CONFIGURED - if UART was configured for DMA mode of transfers OR if the UART was not configured or initialized
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
uint32_t CyU3PUartReceiveBytes(  
    uint8_t * data_p,  
    uint32_t count,  
    CyU3PReturnStatus\_t * status  
);
```

Group

[UART Functions](#)

See Also

- [CyU3PUartInit](#)
- [CyU3PUartDeInit](#)
- [CyU3PUartSetConfig](#)
- [CyU3PUartTransmitBytes](#)
- [CyU3PUartTxSetBlockXfer](#)
- [CyU3PUartRxSetBlockXfer](#)

File

cyu3uart.h

9.1.2.8 CyU3PUartTxSetBlockXfer

Sets the number of bytes to be transmitted by the UART.

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.

Parameters

Parameters	Description
uint32_t txSize	Desired transfer size.

Returns

- CY_U3P_SUCCESS - if the transfer size was set successfully
- CY_U3P_ERROR_NOT_CONFIGURED - if UART was not configured or initialized OR if the UART was configured for register mode
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus\_t CyU3PUartTxSetBlockXfer(  
    uint32_t txSize  
);
```

Group

[UART Functions](#)

See Also

- [CyU3PUartEvt_t](#)
- [CyU3PUartError_t](#)
- [CyU3PUartParity_t](#)
- [CyU3PUartConfig_t](#)
- [CyU3PUartIntrCb_t](#)
- [CyU3PUartStopBit_t](#)
- [CyU3PUartBaudrate_t](#)
- [CyU3PUartInit](#)
- [CyU3PUartDeInit](#)
- [CyU3PUartSetConfig](#)
- [CyU3PUartTransmitBytes](#)
- [CyU3PUartReceiveBytes](#)
- [CyU3PUartRxSetBlockXfer](#)

File

cyu3uart.h

9.1.2.9 CyU3PUartRxSetBlockXfer

Sets the number of bytes to be received by the UART.

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.

Parameters

Parameters	Description
uint32_t rxSize	Desired transfer size.

Returns

- CY_U3P_SUCCESS - if the transfer size was set successfully
- CY_U3P_ERROR_NOT_CONFIGURED - if UART was not configured or initialized OR if the UART was configured for register mode
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus\_t CyU3PUartRxSetBlockXfer(  
    uint32_t rxSize  
);
```

Group

[UART Functions](#)

See Also

- [CyU3PUartInit](#)
- [CyU3PUartDeInit](#)
- [CyU3PUartSetConfig](#)
- [CyU3PUartTransmitBytes](#)
- [CyU3PUartReceiveBytes](#)
- [CyU3PUartTxSetBlockXfer](#)

File

cyu3uart.h

9.1.2.10 CyU3PRegisterUartCallBack

This function register the call back function for notification of UART interrupt.

This function registers a callback function that will be called for notification of UART interrupts and also selects the UART interrupt sources of interest.

Returns

None

C++

```
void CyU3PRegisterUartCallBack(  
    CyU3PUartIntrCb\_t uartIntrCb  
) ;
```

Group

[UART Functions](#)

See Also

- [CyU3PUartEvt_t](#)
- [CyU3PUartError_t](#)

File

cyu3uart.h

9.2 I2C Interface

The FX3 API library includes a I2C interface driver and provides a set of APIs that allow the configuration of the I2C interface properties and data exchange with one or more I2C slave devices.

Group

[Serial Peripheral Interfaces](#)



Topics

Topic	Description
I2C Data Types	This section documents the data types that are defined as part of the I2C driver and API library.
I2C Functions	This section documents the functions defined as part of the I2C driver and API library.

9.2.1 I2C Data Types

This section documents the data types that are defined as part of the I2C driver and API library.



Enumerations

Enumeration	Description
 CyU3PI2cEvt_t	List of I2C related event types.
 CyU3PI2cError_t	List of I2C specific error/status codes.



Group

[I2C Interface](#)

Legend

	Enumeration
	Structure

Structures

Structure	Description
 CyU3PI2cPreamble_t	Structure defining the preamble to be sent on the I2C interface.
 CyU3PI2cConfig_t	Structure defining the configuration of the I2C interface.

Types

Type	Description
CyU3PI2cIntrCb_t	Prototype of I2C event callback function.

9.2.1.1 CyU3PI2cEvt_t

List of I2C related event types.

This enumeration lists the various I2C related event codes that are notified to the user application through an event callback.

C++

```
enum CyU3PI2cEvt_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
};
```

Group

[I2C Data Types](#)

Notes

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](#)
- [CyU3PI2cCb_t](#)

Members

Members	Description
CY_U3P_I2C_EVENT_RX_DONE = 0	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 arbitration
CY_U3P_I2C_EVENT_ERROR	Error has happened

File

cyu3i2c.h

9.2.1.2 CyU3PI2cError_t

List of I2C specific error/status codes.

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.

C++

```
enum CyU3PI2cError_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
};
```

Group

[I2C Data Types](#)

See Also

- [CyU3PI2cEvt_t](#)
- [CyU3PI2cCb_t](#)

Members

Members	Description
CY_U3P_I2C_ERROR_NAK_BYTE_0 = 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.

File

cyu3i2c.h

9.2.1.3 CyU3PI2cIntrCb_t

Prototype of I2C event callback function.

This function type defines a callback to be called after I2C interrupt has been received. A function of this type can be registered with the LPP driver as a callback function and will be called whenever an event of interest occurs.

The I2C has to be configured for DMA mode of transfer for callbacks to be registered.

C++

```
typedef void (* CyU3PI2cIntrCb_t)(CyU3PI2cEvt\_t evt, CyU3PI2cError\_t error);
```

Group

[I2C Data Types](#)

See Also

- [CyU3PI2cEvt_t](#)
- [CyU3PI2cError_t](#)
- [CyU3PRegisterI2cCallBack](#)

File

cyu3i2c.h

9.2.1.4 CyU3PI2cPreamble_t

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

All I2C data transfer requires a preamble, which contains the slave address and the direction of the transfer. Here we are extending this to include the command that will typically be sent to the I2C device which will trigger a data transfer.

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

For example if you look at an I2C EEPROM, it requires the address to read / write from. This is considered as the command and the data which is being read / written is considered as the actual data transfer. So the two I2C operations are combined into one I2C API call using the parameters of the structure.

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.

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 is 0x0004 as a start bit is required after the third byte (third bit is set).

C++

```
struct CyU3PI2cPreamble_t {
    uint8_t buffer[8];
    uint8_t length;
    uint16_t ctrlMask;
};
```

Group

[I2C Data Types](#)

See Also

- [CyU3PI2cSetConfig](#)

Members

Members	Description
uint8_t buffer[8];	The extended preamble information.
uint8_t length;	The length of the preamble to be sent. Should be between 1 and 8.

<code>uint16_t ctrlMask;</code>	This field controls the start stop condition after every byte of preamble data. Bits 0 - 7 is a bit mask for start condition and Bits 8 - 15 is a bit mask for stop condition. If both are set, then stop takes priority.
---------------------------------	---

File

`cyu3i2c.h`

9.2.1.5 CyU3PI2cConfig_t

Structure defining the configuration of the I2C interface.

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 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.

C++

```
struct CyU3PI2cConfig_t {  
    uint32_t bitRate;  
    CyBool_t isDma;  
    uint32_t busTimeout;  
    uint16_t dmaTimeout;  
};
```

Group

[I2C Data Types](#)

See Also

- [CyU3PI2cSetConfig](#)

Members

Members	Description
uint32_t bitRate;	Bit rate for the interface. (Eg: 100000 for 100KHz)
CyBool_t isDma;	CyFalse: Register transfer mode, CyTrue: DMA transfer mode
uint32_t busTimeout;	Number of core clocks SCK can be held low by the slave byte transmission before triggering a timeout error. 0xFFFFFFFFU means no timeout.
uint16_t dmaTimeout;	Number of core clocks DMA can remain not ready before flagging an error. 0xFFFF means no timeout.














File

cyu3i2c.h

9.2.2 I2C Functions

This section documents the functions defined as part of the I2C driver and API library.

Functions

Function	Description
 CyU3PI2cSetClock	Function sets the required frequency for the I2C block.
 CyU3PI2cStopClock	This function disables the clock of the I2C.
 CyU3PI2cInit	Starts the I2C interface block on the FX3.
 CyU3PI2cDeInit	Stops the I2C module.
 CyU3PI2cSetConfig	Sets the I2C interface parameters.
 CyU3PI2cSendCommand	Perform a read or write operation to the I2C slave.
 CyU3PI2cWaitForAck	Poll an I2C slave until all of the preamble is ACKed.
 CyU3PI2cReceiveBytes	Reads a small number of bytes one by one from an I2C slave.
 CyU3PI2cTransmitBytes	Writes a small number of bytes to an I2C slave.
 CyU3PI2cGetErrorCode	Retrieves the error code as defined by CyU3PI2cError_t .
 CyU3PI2cWaitForBlockXfer	Wait until the ongoing I2C data transfer is finished.
 CyU3PSetI2cDriveStrength	Set the IO drive strength for the I2C interface.
 CyU3PRegisterI2cCallBack	This function register the call back function for notification of I2C interrupt.

Group

[I2C Interface](#)

Legend

	Method
---	--------

9.2.2.1 CyU3PI2cSetClock

Function sets the required frequency for the I2C block.

The clock for the I2C block can be configured to required frequency.

Bitrate calculation: The maximum bitrate supported is 1MHz and minimum bitrate is 100KHz. 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 actual bit rate is derived out of the $\text{SYS_CLK} / 16$. The I2C block requires the clocking to be 10X the required bitrate. Since the divider needs to be integral or with half-divider, the frequency approximation is done with the following method:

If x is the actual divider and n is the required integral divider to be used, then following conditions are used to evaluate:

```
x = (SYS_CLK / 16) / (bitrate * 10);
```

```
if (x - floor(x)) < 0.25 ==> n = floor(x);
```

```
if (((x - floor(x)) >= 0.25) && (x - floor(x)) < 0.5) ==> n = floor(x) + half divider;
```

```
if (x - floor(x)) >= 0.75 ==> n = floor(x) + 1;
```

Parameters

Parameters	Description
uint32_t bitrate	Desired interface clock frequency.

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PI2cSetClock(  
    uint32_t bitrate  
);
```

Group

[I2C Functions](#)

See Also

- [CyU3PI2cStopClock](#)

File

cyu3lpp.h

9.2.2.2 CyU3PI2cStopClock

This function disables the clock of the I2C.

The clock for the I2C block is disabled. It is expected while deinitialization the I2C.

Returns

- CY_U3P_SUCCESS - If the I2C clock was successfully turned off.

C++

```
CyU3PReturnStatus\_t CyU3PI2cStopClock(  
    void  
);
```

Group

[I2C Functions](#)

See Also

- [CyU3PI2cSetClock](#)

File

cyu3lpp.h

9.2.2.3 CyU3PI2cInit

Starts the I2C interface block on the FX3.

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.

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PI2cInit(  
    void  
);
```

Group

[I2C Functions](#)

See Also

- [CyU3PI2cDeinit](#)
- [CyU3PI2cSetConfig](#)
- [CyU3PI2cSendCommand](#)
- [CyU3PI2cTransmitBytes](#)
- [CyU3PI2cReceiveBytes](#)
- [CyU3PI2cWaitForAck](#)
- [CyU3PI2cWaitForBlockXfer](#)
- [CyU3PSetI2cDriveStrength](#)

File

cyu3i2c.h

9.2.2.4 CyU3PI2cDeInit

Stops the I2C module.

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.

Returns

- CY_U3P_SUCCESS - When the DeInit is successful
- CY_U3P_ERROR_NOT_STARTED - When the I2C module has not been previously initialized

C++

```
CyU3PReturnStatus\_t CyU3PI2cDeInit(  
    void  
);
```

Group

[I2C Functions](#)

See Also

- [CyU3PI2cInit](#)
- [CyU3PI2cSetConfig](#)
- [CyU3PI2cSendCommand](#)
- [CyU3PI2cTransmitBytes](#)
- [CyU3PI2cReceiveBytes](#)
- [CyU3PI2cWaitForAck](#)
- [CyU3PI2cWaitForBlockXfer](#)
- [CyU3PSetI2cDriveStrength](#)

File

cyu3i2c.h

9.2.2.5 CyU3PI2cSetConfig

Sets the I2C interface parameters.

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](#). But this will reset the FIFO and hence the data in pipe will be lost. If a DMA channel is present, a Reset has to be issued.

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 required. User must pass NULL as cb when using the register mode.

Bitrate calculation: The maximum bitrate supported is 1MHz and minimum bitrate is 100KHz. 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 actual bit rate is derived out of the $SYS_CLK / 16$. The I2C block requires the clocking to be 10X the required bitrate. Since the divider needs to be integral or with half-divider, the frequency approximation is done with the following method:

If x is the actual divider and n is the required integral divider to be used, then following conditions are used to evaluate:

```
x = (SYS_CLK / 16) / (bitrate * 10);
```

```
if (x - floor(x)) < 0.25 ==> n = floor(x);
```

```
if (((x - floor(x)) >= 0.25) && (x - floor(x)) < 0.5) ==> n = floor(x) + half divider;
```

```
if (x - floor(x)) >= 0.75 ==> n = floor(x) + 1;
```

Parameters

Parameters	Description
CyU3PI2cConfig_t * config	I2C configuration settings
CyU3PI2cIntrCb_t cb	Callback for getting the events

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PI2cSetConfig(  
    CyU3PI2cConfig\_t * config,  
    CyU3PI2cIntrCb\_t cb  
);
```

Group

[I2C Functions](#)

See Also

- [CyU3PI2cCb_t](#)
- [CyU3PI2cEvt_t](#)
- [CyU3PI2cError_t](#)
- [CyU3PI2cIntrCb_t](#)
- [CyU3PI2cConfig_t](#)
- [CyU3PI2cIntr](#)
- [CyU3PI2cDeinit](#)
- [CyU3PI2cSendCommand](#)
- [CyU3PI2cTransmitBytes](#)
- [CyU3PI2cReceiveBytes](#)
- [CyU3PI2cWaitForAck](#)
- [CyU3PI2cWaitForBlockXfer](#)
- [CyU3PSetI2cDriveStrength](#)

File

[cyu3i2c.h](#)

9.2.2.6 CyU3PI2cSendCommand

Perform a read or write operation to the I2C slave.

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/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.

Parameters

Parameters	Description
CyU3PI2cPreamble_t * preamble	Preamble information to be sent out before the data transfer.
uint32_t byteCount	Size of the transfer in bytes.
CyBool_t isRead	Direction of transfer;
CyTrue	Read, CyFalse: Write.

Returns

- CY_U3P_SUCCESS - When the SendCommand is successful
- CY_U3P_ERROR_NULL_POINTER - When the preamble is NULL
- CY_U3P_ERROR_BAD_ARGUMENT - When the preamble length is 0 or greater than 8 Also when byteCount is 0xFFFFFFFF (only finite length transfers are allowed)
- 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

C++

```
CyU3PReturnStatus_t CyU3PI2cSendCommand(  
    CyU3PI2cPreamble\_t * preamble,  
    uint32_t byteCount,  
    CyBool_t isRead  
);
```

Group

[I2C Functions](#)

See Also

- [CyU3PI2cPreamble_t](#)
- [CyU3PI2cInit](#)
- [CyU3PI2cDeinit](#)
- [CyU3PI2cSetConfig](#)

- [CyU3PI2cTransmitBytes](#)
- [CyU3PI2cReceiveBytes](#)
- [CyU3PI2cWaitForAck](#)
- [CyU3PI2cWaitForBlockXfer](#)

File

cyu3i2c.h

9.2.2.7 CyU3PI2cWaitForAck

Poll an I2C slave until all of the preamble is ACKed.

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 retry is done continuously without any delay. If any delay is required, then it should be added in the application firmware. The function call can be repeated on ERROR_TIMEOUT without any error recovery.

The API plays the provided preamble bytes continuously until all bytes of the preamble are ACKed.

Parameters

Parameters	Description
CyU3PI2cPreamble_t * preamble	Preamble information to be sent out before the data transfer.
uint32_t retryCount	Number of times to retry request if preamble is NAKed or an error is encountered.

Returns

- CY_U3P_SUCCESS - When the device returns the desired handshake
- CY_U3P_ERROR_NULL_POINTER - When 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 reinit.
- CY_U3P_ERROR_TIMEOUT - I2C bus timeout occurred.
- CY_U3P_ERROR_LOST_ARBITRATION - Lost the bus arbitration or there was an invalid bus activity.

- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

- See Also

- [CyU3PI2cPreamble_t](#)
- [CyU3PI2cInit](#)
- [CyU3PI2cDeinit](#)
- [CyU3PI2cSetConfig](#)
- [CyU3PI2cSendCommand](#)
- [CyU3PI2cTransmitBytes](#)
- [CyU3PI2cReceiveBytes](#)
- [CyU3PI2cWaitForBlockXfer](#)

C++

```
CyU3PReturnStatus\_t CyU3PI2cWaitForAck(  
    CyU3PI2cPreamble\_t * preamble,  
    uint32_t retryCount
```

```
) ;
```

Group

[I2C Functions](#)

File

cyu3i2c.h

9.2.2.8 CyU3PI2cReceiveBytes

Reads a small number of bytes one by one from an I2C slave.

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. The function call can be repeated on ERROR_TIMEOUT without any error recovery. The retry is done continuously without any delay. If any delay is required, then it should be added in the application firmware.

The API will return when FX3 has received the data. If the slave device requires additional time before servicing the next request, then either sufficient delay must be provided or if the slave device supports WaitForAck functionality, then [CyU3PI2cWaitForAck](#) API can be used. Refer to the I2C slave datasheet for more details.

This function inherently calls [CyU3PI2cSendCommand](#). So user must not call [CyU3PI2cSendCommand](#) before calling this function.

Parameters

Parameters	Description
CyU3PI2cPreamble_t * preamble	Preamble information to be sent out before the data transfer.
uint8_t * data	Pointer to buffer where the data is to be placed.
uint32_t byteCount	Size of the transfer in bytes.
uint32_t retryCount	Number of times to retry request if preamble is NAKed or an error is encountered.

Returns

- CY_U3P_SUCCESS - When the ReceiveBytes is successful or byteCount is 0
- CY_U3P_ERROR_NULL_POINTER - When the preamble or data are NULL
- CY_U3P_ERROR_FAILURE - When a transfer fails with an error defined in [CyU3PI2cError_t](#).
- CY_U3P_ERROR_NOT_CONFIGURED - When the I2C is configured for DMA mode of operation OR I2C is not configured OR I2C is not initialized
- CY_U3P_ERROR_BLOCK_FAILURE - When the I2c block encounters a fatal error and requires reinit.
- CY_U3P_ERROR_TIMEOUT - I2C bus timeout occurred.
- CY_U3P_ERROR_LOST_ARBITRATION - Lost the bus arbitration or there was an invalid bus activity.
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus\_t CyU3PI2cReceiveBytes(  
    CyU3PI2cPreamble\_t * preamble,  
    uint8_t * data,  
    uint32_t byteCount,  
    uint32_t retryCount  
);
```

Group

[I2C Functions](#)

See Also

- [CyU3PI2cPreamble_t](#)
- [CyU3PI2cInit](#)
- [CyU3PI2cDeinit](#)
- [CyU3PI2cSetConfig](#)
- [CyU3PI2cSendCommand](#)
- [CyU3PI2cTransmitBytes](#)
- [CyU3PI2cWaitForAck](#)
- [CyU3PI2cWaitForBlockXfergg](#)

File

cyu3i2c.h

9.2.2.9 CyU3PI2cTransmitBytes

Writes a small number of bytes to an I2C slave.

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 on ERROR_TIMEOUT without any error recovery. The retry is done continuously without any delay. If any delay is required, then it should be added in the application firmware.

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 if the slave device supports WaitForAck functionality, then [CyU3PI2cWaitForAck](#) API can be used. Refer to the I2C slave datasheet for more details.

This function inherently calls [CyU3PI2cSendCommand](#). So user must not call [CyU3PI2cSendCommand](#) before calling this function.

Parameters

Parameters	Description
CyU3PI2cPreamble_t * preamble	Preamble information to be sent out before the data transfer.
uint8_t * data	Pointer to buffer containing data to be written.
uint32_t byteCount	Size of the transfer in bytes.
uint32_t retryCount	Number of times to retry request if a byte is NAKed by the slave.

Returns

- CY_U3P_SUCCESS - When the TransmitBytes is successful or byteCount is 0
- CY_U3P_ERROR_NULL_POINTER - When the preamble or data are 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 reinit.
- CY_U3P_ERROR_NOT_CONFIGURED - When the I2C is configured for DMA mode of operation OR I2C is not configured OR I2C is not initialized
- CY_U3P_ERROR_TIMEOUT - I2C bus timeout occurred.
- CY_U3P_ERROR_LOST_ARBITRATION - Lost the bus arbitration or there was an invalid bus activity.
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus\_t CyU3PI2cTransmitBytes(  
    CyU3PI2cPreamble\_t * preamble,  
    uint8_t * data,  
    uint32_t byteCount,  
    uint32_t retryCount  
);
```

Group

[I2C Functions](#)

See Also

- [CyU3PI2cPreamble_t](#)
- [CyU3PI2cInit](#)
- [CyU3PI2cDeinit](#)
- [CyU3PI2cSetConfig](#)
- [CyU3PI2cSendCommand](#)
- [CyU3PI2cReceiveBytes](#)
- [CyU3PI2cWaitForAck](#)
- [CyU3PI2cWaitForBlockXfer](#)

File

cyu3i2c.h

9.2.2.10 CyU3PI2cGetErrorCode

Retrieves the error code as defined by [CyU3PI2cError_t](#).

This function can be used to retrieve the error code when [CyU3PI2cTransmitBytes](#) / [CyU3PI2cReceiveBytes](#) / [CyU3PI2cWaitForAck](#) functions fail with error code CY_U3P_ERROR_FAILURE.

Parameters

Parameters	Description
CyU3PI2cError_t * error_p	Error code.

Returns

- 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_NULL_POINTER - When the pointer passed is NULL.
- CY_U3P_ERROR_NOT_STARTED - When there is no error flagged.
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus\_t CyU3PI2cGetErrorCode(  
    CyU3PI2cError\_t * error_p  
);
```

Group

[I2C Functions](#)

See Also

- [CyU3PI2cInit](#)
- [CyU3PI2cDeinit](#)
- [CyU3PI2cSetConfig](#)
- [CyU3PI2cSendCommand](#)
- [CyU3PI2cTransmitBytes](#)
- [CyU3PI2cReceiveBytes](#)
- [CyU3PI2cWaitForAck](#)

File

cyu3i2c.h

9.2.2.11 CyU3PI2cWaitForBlockXfer

Wait until the ongoing I2C data transfer is finished.

This function can be used to ensure that a previous I2C transaction has completed, in the case where the callback is not being used. This function is only meaningful in DMA mode of operation because the register based read/write APIs are blocking in nature.

The API will return when FX3 has received/transmitted the data. If the slave device requires additional time before servicing the next request, then sufficient delay must be provided. If a DMA operation fails, then the I2C block needs to be de-init'd and re-initialized before sending any further commands.

Parameters

Parameters	Description
CyBool_t isRead	Type of operation to wait on;
CyTrue	Read, CyFalse: Write

Returns

- 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 - When callbacks are enabled
- 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 reinit.
- CY_U3P_ERROR_TIMEOUT - I2C bus timeout occurred.
- CY_U3P_ERROR_LOST_ARBITRATION - Lost the bus arbitration or there was an invalid bus activity.
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus\_t CyU3PI2cWaitForBlockXfer(  
    CyBool_t isRead  
);
```

Group

[I2C Functions](#)

Notes

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.

See Also

- [CyU3PI2cInit](#)
- [CyU3PI2cDeinit](#)
- [CyU3PI2cSetConfig](#)

- [CyU3PI2cSendCommand](#)
- [CyU3PI2cTransmitBytes](#)
- [CyU3PI2cReceiveBytes](#)
- [CyU3PI2cWaitForAck](#)

File

cyu3i2c.h

9.2.2.12 CyU3PSetI2cDriveStrength

Set the IO drive strength for the I2C interface.

The function sets the IO Drive strength for the I2C interface. The default IO drive strength for I2C is set to CY_U3P_DS_THREE_QUARTER_STRENGTH.

Parameters

Parameters	Description
CyU3PDriveStrengthState_t i2cDriveStrength	Drive strength desired for I2C signals.

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PSetI2cDriveStrength(  
    CyU3PDriveStrengthState\_t i2cDriveStrength  
);
```

Group

[I2C Functions](#)

See Also

- [CyU3PDriveStrengthState_t](#)

File

cyu3lpp.h

9.2.2.13 CyU3PRegisterI2cCallback

This function register the call back function for notification of I2C interrupt.

This function registers a callback function that will be called for notification of I2C interrupts and also selects the I2C interrupt sources of interest.

Returns

None

C++

```
void CyU3PRegisterI2cCallback(  
    CyU3PI2cIntrCb\_t i2cIntrCb  
) ;
```

Group

[I2C Functions](#)

See Also

- [CyU3PI2cEvt_t](#)
- [CyU3PI2cError_t](#)

File

cyu3i2c.h

9.3 SPI Interface

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.

Group

[Serial Peripheral Interfaces](#)





Topics

Topic	Description
SPI data types	This section documents the enums and data structures that are defined as part of the SPI Interface Driver.
SPI Functions	This section documents the API functions that are defined as part of the SPI Interface Driver.

9.3.1 SPI data types

This section documents the enums and data structures that are defined as part of the SPI Interface Driver.



Enumerations

Enumeration	Description
 CyU3PSpiEvt_t	List of SPI related event types.
 CyU3PSpiError_t	List of SPI specific error / status codes.
 CyU3PSpiSsnCtrl_t	Enumeration defining the various ways in which the SSN for a SPI device can be controlled.
 CyU3PSpiSsnLagLead_t	Enumeration defining the lead and lag times of SSN with respect to SCK


Group

[SPI Interface](#)

Legend

	Enumeration
	Structure

Structures

Structure	Description
 CyU3PSpiConfig_t	Structure defining the configuration of SPI interface.

Types

Type	Description
CyU3PSpiIntrCb_t	Prototype of SPI event callback function.

9.3.1.1 CyU3PSpiEvt_t

List of SPI related event types.

This enumeration lists the various SPI related event codes that are notified to the user application through an event callback.

C++

```
enum CyU3PSpiEvt_t {
    CY_U3P_SPI_EVENT_RX_DONE = 0,
    CY_U3P_SPI_EVENT_TX_DONE,
    CY_U3P_SPI_EVENT_ERROR
};
```

Group

[SPI data types](#)

See Also

- [CyU3PSpiError_t](#)
- [CyU3PSpiCb_t](#)

Members

Members	Description
CY_U3P_SPI_EVENT_RX_DONE = 0	Reception is completed
CY_U3P_SPI_EVENT_TX_DONE	Transmission is done
CY_U3P_SPI_EVENT_ERROR	Error has occurred

File

cyu3spi.h

9.3.1.2 CyU3PSpiError_t

List of SPI specific error / status codes.

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.

C++

```
enum CyU3PSpiError_t {
    CY_U3P_SPI_ERROR_TX_OVERFLOW = 12,
    CY_U3P_SPI_ERROR_RX_UNDERFLOW = 13,
    CY_U3P_SPI_ERROR_NONE = 15
};
```

Group

[SPI data types](#)

See Also

- [CyU3PSpiEvt_t](#)

Members

Members	Description
CY_U3P_SPI_ERROR_TX_OVERFLOW = 12	Write to TX_FIFO when FIFO is full.
CY_U3P_SPI_ERROR_RX_UNDERFLOW = 13	Read from RX_FIFO when FIFO is empty.
CY_U3P_SPI_ERROR_NONE = 15	No error has happened.

File

cyu3spi.h

9.3.1.3 CyU3PSpiSsnCtrl_t

Enumeration defining the various ways in which the SSN for a SPI device can be controlled.

The SSN can be controlled by the firmware (API) which is not synchronized with SPI clock. It is asserted at the beginning of a transfer and is de-asserted at the end of the transfer.

The SSN can be controlled by the hardware. In this case the SSN assert and de-asserts are done in sync with SPI clock.

The SSN can be controlled by the application, external to the API. This can be done using GPIOs. In this case the hardware / API is not aware of the SSN.

The APIs allow only control of one SSN line. If there are more than one SPI slave device, then at-most one device can be controlled by hardware / API.

For example, if there are two devices, and SSN for one is controlled by hardware / API, SetConfig request needs to be called whenever the device to be addressed changes. On the other hand if the SSN for both devices are controlled externally (via GPIO), then SetConfig needs to be called only once, provided rest of the configuration stays the same.

C++

```
enum CyU3PSpiSsnCtrl_t {  
    CY_U3P_SPI_SSN_CTRL_FW = 0,  
    CY_U3P_SPI_SSN_CTRL_HW_END_OF_XFER,  
    CY_U3P_SPI_SSN_CTRL_HW_EACH_WORD,  
    CY_U3P_SPI_SSN_CTRL_HW_CPHA_BASED,  
    CY_U3P_SPI_SSN_CTRL_NONE,  
    CY_U3P_SPI_NUM_SSN_CTRL  
};
```

Group

[SPI data types](#)

See Also

- [CyU3PSpiConfig_t](#)
- [CyU3PSpiSetConfig](#)

Members

Members	Description
CY_U3P_SPI_SSN_CTRL_FW = 0	SSN is controlled by API and is not at clock boundaries. It is asserted at beginning of transfer is de-asserted at end of transfer.
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 is de-asserted at the end of a transfer or when no data is available to transmit (underrun).
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. So if a transfer consists of 64 word, the SSN is asserted and de-asserted 64 times.

Serial Peripheral Interfaces

SPI Interface

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. The SSN lines are selected by the application and is ignored by the hardware / API.
CY_U3P_SPI_NUM_SSN_CTRL	Number of enumerations.

File

cyu3spi.h

9.3.1.4 CyU3PSpiSsnLagLead_t

Enumeration defining the lead and lag times of SSN with respect to SCK

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.

C++

```
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
};
```

Group

[SPI data types](#)

See Also

- [CyU3PSpiConfig_t](#)
- [CyU3PSpiSetConfig](#)

Members

Members	Description
CY_U3P_SPI_SSN_LAG_LEAD_ZERO_CLK = 0	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 enumerations.

File

[cyu3spi.h](#)

9.3.1.5 CyU3PSpiIntrCb_t

Prototype of SPI event callback function.

This function type defines a callback to be called after SPI interrupt has been received. A function of this type can be registered with the LPP driver as a callback function and will be called whenever an event of interest occurs.

C++

```
typedef void (* CyU3PSpiIntrCb_t)(CyU3PSpiEvt\_t evt, CyU3PSpiError\_t error);
```

Group

[SPI data types](#)

See Also

- [CyU3PSpiEvt_t](#)
- [CyU3PSpiError_t](#)
- [CyU3PRegisterSpiCallBack](#)

File

cyu3spi.h

9.3.1.6 CyU3PSpiConfig_t

Structure defining the configuration of SPI interface.

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.

C++

```
struct CyU3PSpiConfig_t {  
    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;  
};
```

Group

[SPI data types](#)

See Also

- [CyU3PSpiSsnCtrl_t](#)
- [CyU3PSpiSclkParam_t](#)
- [CyU3PSpiSetConfig](#)

Members

Members	Description
CyBool_t isLsbFirst;	Data shift mode - CyFalse: MSB first, CyTrue: LSB first
CyBool_t cpol;	Clock polarity - CyFalse(0): SCK idles low, CyTrue(1): SCK idles high
CyBool_t cpha;	Clock phase - CyFalse(0): Slave samples at idle-active edge, CyTrue(1): Slave samples at active-idle edge
CyBool_t ssnPol;	Polarity of SSN line. CyFalse (0): SSN is active low, CyTrue (1): SSN is active high.
CyU3PSpiSsnCtrl_t ssnCtrl;	SSN control
CyU3PSpiSsnLagLead_t leadTime;	Time between SSN's assertion and first SCLK's edge. This is at the beginning of a transfer and is valid only for hardware controlled SSN. Zero lead time is not supported.
CyU3PSpiSsnLagLead_t lagTime;	Time between the last SCK edge to SSN's de-assertion. This is at the end of a transfer and is valid only for hardware controlled SSN. For CPHA = 1, lag time cannot be zero.
uint32_t clock;	SPI clock frequency in Hz.
uint8_t wordLen;	Word length in bits. Valid values are 4 - 32.













File

cyu3spi.h

9.3.2 SPI Functions

This section documents the API functions that are defined as part of the SPI Interface Driver.

Functions

Function	Description
 CyU3PSpiSetClock	This function set the required frequency the SPI block.
 CyU3PSpiStopClock	This function disables the clock of the SPI block.
 CyU3PSpiInit	Starts the SPI interface block on the device.
 CyU3PSpiDeInit	Stops the SPI module.
 CyU3PSpiSetConfig	Sets and configures SPI interface parameters.
 CyU3PSpiSetSsnLine	Assert / Deassert the SSN Line.
 CyU3PSpiTransmitWords	Transmits data word by word over the SPI interface.
 CyU3PSpiReceiveWords	Receives data word by word over the Spi interface.
 CyU3PSpiSetBlockXfer	This function enables the DMA transfer.
 CyU3PSpiWaitForBlockXfer	Wait until the ongoing SPI data transfer is finished.
 CyU3PSpiDisableBlockXfer	This function disabled the TX / RX DMA functionality.
 CyU3PRegisterSpiCallBack	This function register the call back function for notification of SPI interrupt.

Group

[SPI Interface](#)

Legend

	Method
---	--------

9.3.2.1 CyU3PSpiSetClock

This function set the required frequency the SPI block.

The clock for the SPI block can be configured to required frequency.

SPI-clock calculation: The maximum SPI clock supported is 33MHz and the minimum is 10KHz. The SPI block requires an internal clocking of 2X. It should be noted that even though the clock dividers and the API allows frequencies above and below the rated range, the device behaviour is not guaranteed. The actual clock rate is derived out of SYS_CLK. Since the divider needs to be integral or with half divider, the frequency approximation is done using the following algorithm:

If x is the actual divider and n is the required integral divider to be used, then following conditions are used to evaluate:

$x = (\text{SYS_CLK}) / (\text{SPI_clock} * 2);$

if $(x - \text{floor}(x)) < 0.25 \implies n = \text{floor}(x);$

if $((x - \text{floor}(x)) \geq 0.25 \ \&\& \ (x - \text{floor}(x)) < 0.5) \implies n = \text{floor}(x) + \text{half divider};$

if $(x - \text{floor}(x)) \geq 0.75 \implies n = \text{floor}(x) + 1;$

Parameters

Parameters	Description
uint32_t clock	Clock frequency desired.

Returns

- CY_U3P_SUCCESS - If the SPI clock has been successfully turned on.
- CY_U3P_ERROR_NOT_SUPPORTED - If the SPI interface is not supported by the current FX3 device.
- CY_U3P_ERROR_BAD_ARGUMENT - When invalid argument is passed to the function

C++

```
CyU3PReturnStatus\_t CyU3PSpiSetClock(  
    uint32_t clock  
);
```

Group

[SPI Functions](#)

See Also

- [CyU3PSpiStopClock](#)

File

cyu3lpp.h

9.3.2.2 CyU3PSpiStopClock

This function disables the clock of the SPI block.

The clock for the SPI block is disabled. This should be called after the corresponding driver has been deinitialized.

Returns

- CY_U3P_SUCCESS - if the SPI clock is turned off successfully.
- CY_U3P_ERROR_NOT_SUPPORTED - if the SPI interface is not supported by the current FX3 device.

C++

```
CyU3PReturnStatus\_t CyU3PSpiStopClock(  
    void  
);
```

Group

[SPI Functions](#)

See Also

- [CyU3PSpiSetClock](#)

File

cyu3lpp.h

9.3.2.3 CyU3PSpiInit

Starts the SPI interface block on the device.

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.

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PSpiInit(  
    void  
);
```

Group

[SPI Functions](#)

See Also

- [CyU3PSpiDelInit](#)
- [CyU3PSpiSetConfig](#)
- [CyU3PSpiSetSsnLine](#)
- [CyU3PSpiTransmitWords](#)
- [CyU3PSpiReceiveWords](#)
- [CyU3PSpiSetBlockXfer](#)
- [CyU3PSpiWaitForBlockXfer](#)
- [CyU3PSpiDisableBlockXfer](#)

File

cyu3spi.h

9.3.2.4 CyU3PSpiDeInit

Stops the SPI module.

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.

Returns

- CY_U3P_SUCCESS - When the de-init is successful
- CY_U3P_ERROR_NOT_STARTED - When the SPI block has not been initialized

C++

```
CyU3PReturnStatus\_t CyU3PSpiDeInit(  
    void  
);
```

Group

[SPI Functions](#)

See Also

- [CyU3PSpiInit](#)
- [CyU3PSpiSetConfig](#)
- [CyU3PSpiSetSsnLine](#)
- [CyU3PSpiTransmitWords](#)
- [CyU3PSpiReceiveWords](#)
- [CyU3PSpiSetBlockXfer](#)
- [CyU3PSpiWaitForBlockXfer](#)
- [CyU3PSpiDisableBlockXfer](#)

File

cyu3spi.h

9.3.2.5 CyU3PSpiSetConfig

Sets and configures SPI interface parameters.

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 will reset the FIFO and hence the data in pipe will be lost. If a DMA channel is present, a DMA reset has to be issued.

The callback parameter is used to specify an event callback function that will be called by the driver when an SPI interrupt occurs.

SPI-clock calculation: The maximum SPI clock supported is 33MHz and the minimum is 10KHz. The SPI block requires an internal clocking of 2X. It should be noted that even though the clock dividers and the API allows frequencies above and below the rated range, the device behaviour is not guaranteed. The actual clock rate is derived out of SYS_CLK. Since the divider needs to be integral or with half divider, the frequency approximation is done using the following algorithm:

If x is the actual divider and n is the required integral divider to be used, then following conditions are used to evaluate:

```
x = (SYS_CLK) / (SPI_clock * 2);
```

```
if (x - floor(x)) < 0.25 ==> n = floor(x);
```

```
if (((x - floor(x)) >= 0.25) && (x - floor(x)) < 0.5) ==> n = floor(x) + half divider;
```

```
if (x - floor(x)) >= 0.75 ==> n = floor(x) + 1;
```

Parameters

Parameters	Description
CyU3PSpiConfig_t * config	pointer to the SPI config structure
CyU3PSpiIntrCb_t cb	Callback for receiving SPI events

Returns

- CY_U3P_SUCCESS - When the SetConfig is successful
- CY_U3P_ERROR_NOT_STARTED - When the SPI block has not been initialized
- CY_U3P_ERROR_NULL_POINTER - When the config pointer is NULL
- CY_U3P_ERROR_BAD_ARGUMENT - When a configuration value is invalid
- CY_U3P_ERROR_INVALID_SEQUENCE - When API call sequence is invalid, eg: a previously set up DMA transfer has not been disabled
- CY_U3P_ERROR_TIMEOUT - When the operation times out
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus_t CyU3PSpiSetConfig(  
    CyU3PSpiConfig_t * config,  
    CyU3PSpiIntrCb_t cb
```

```
) ;
```

Group

[SPI Functions](#)

See Also

- [CyU3PSpiCb_t](#)
- [CyU3PSpiEvt_t](#)
- [CyU3PSpiError_t](#)
- [CyU3PSpiConfig_t](#)
- [CyU3PSpiIntrCb_t](#)
- [CyU3PSpiSsnCtrl_t](#)
- [CyU3PSpiSclkParam_t](#)
- [CyU3PSpiSsnLagLead_t](#)
- [CyU3PSpiInit](#)
- [CyU3PSpiDeInit](#)
- [CyU3PSpiSetSsnLine](#)
- [CyU3PSpiTransmitWords](#)
- [CyU3PSpiReceiveWords](#)
- [CyU3PSpiSetBlockXfer](#)
- [CyU3PSpiWaitForBlockXfer](#)
- [CyU3PSpiDisableBlockXfer](#)

File

[cyu3spi.h](#)

9.3.2.6 CyU3PSpiSetSsnLine

Assert / Deassert the SSN Line.

Asserts / deasserts SSN Line for the default slave device. This is possible only if the SSN line control is configured for FW controlled mode.

Parameters

Parameters	Description
CyBool_t isHigh	Cyfalse: Pull down the SSN line,
CyTrue	Pull up the SSN line

Returns

- CY_U3P_SUCCESS - When the call is successful
- CY_U3P_ERROR_NOT_CONFIGURED - When SPI has not been configured or initialized OR SSN is not firmware controlled.
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus\_t CyU3PSpiSetSsnLine(  
    CyBool_t isHigh  
);
```

Group

[SPI Functions](#)

See Also

- [CyU3PSpiInit](#)
- [CyU3PSpiDeInit](#)
- [CyU3PSpiSetConfig](#)
- [CyU3PSpiTransmitWords](#)
- [CyU3PSpiReceiveWords](#)
- [CyU3PSpiSetBlockXfer](#)
- [CyU3PSpiWaitForBlockXfer](#)
- [CyU3PSpiDisableBlockXfer](#)

File

cyu3spi.h

9.3.2.7 CyU3PSpiTransmitWords

Transmits data word by word over the SPI interface.

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 a SetBlockXfer was called, then a DisableBlockXfer needs to be called.

Parameters

Parameters	Description
uint8_t * data	Source data pointer. This needs to be padded to nearest byte if the word length is not byte aligned.
uint32_t byteCount	This needs to be a multiple of word length aligned where each word is aligned to a byte.

Returns

- CY_U3P_SUCCESS - When the TransmitWords is successful
- CY_U3P_ERROR_NOT_CONFIGURED - When SPI 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 - When the operation times out
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus_t CyU3PSpiTransmitWords(  
    uint8_t * data,  
    uint32_t byteCount  
);
```

Group

[SPI Functions](#)

See Also

- [CyU3PSpiInit](#)
- [CyU3PSpiDeInit](#)
- [CyU3PSpiSetConfig](#)
- [CyU3PSpiSetSsnLine](#)
- [CyU3PSpiReceiveWords](#)
- [CyU3PSpiSetBlockXfer](#)
- [CyU3PSpiWaitForBlockXfer](#)
- [CyU3PSpiDisableBlockXfer](#)

File

cyu3spi.h

9.3.2.8 CyU3PSpiReceiveWords

Receives data word by word over the Spi interface.

This function can be called only when there is no active DMA transfer. If there was any previous DMA operation with SPI, then a DisableBlockXfer needs to be called.

Parameters

Parameters	Description
uint8_t * data	Destination buffer pointer
uint32_t byteCount	The byte size of the buffer. This needs to be a multiple of word length aligned to nearest byte size. For example, if the word length is 18 bits then each word would be placed in 3 bytes (valid data on LSBs). If the byte count is not aligned, then the call returns with an error.

Returns

- CY_U3P_SUCCESS - When the ReceiveWords is successful
- CY_U3P_ERROR_NOT_CONFIGURED - When SPI 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 - When the operation times out
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus\_t CyU3PSpiReceiveWords(  
    uint8_t * data,  
    uint32_t byteCount  
);
```

Group

[SPI Functions](#)

See Also

- [CyU3PSpiInit](#)
- [CyU3PSpiDeInit](#)
- [CyU3PSpiSetConfig](#)
- [CyU3PSpiSetSsnLine](#)
- [CyU3PSpiTransmitWords](#)
- [CyU3PSpiSetBlockXfer](#)
- [CyU3PSpiWaitForBlockXfer](#)
- [CyU3PSpiDisableBlockXfer](#)

File

cyu3spi.h

9.3.2.9 CyU3PSpiSetBlockXfer

This function enables the DMA transfer.

This function switches SPI to DMA mode.

If the txSize parameter is non-zero then TX is enabled and if rxSize parameter is non-zero, then RX is enabled. If both are enabled, then SPI block will stall if any of the DMA pipe is stalled.

txSize == rxSize: In this case both TX and RX will function simultaneously. If the DMA pipe gets stalled due to buffer non-availability, both TX and RX will be stalled and will resume when both DMA pipes are active. So there is no loss of synchronization.

txSize > rxSize: In this case RX will receive only as many words as requested and will stop when the count reaches that point. TX will continue to receive data until its count runs down.

txSize < rxSize: In this case the TX and RX will function till all TX words are received and then will stall. The DisableBlockXfer API needs to be called explicitly for TX for the RX to resume.

A call to SetBlockXfer has to be followed by a call to DisableBlockXfer before the SPI can be used in Register mode.

Parameters

Parameters	Description
uint32_t txSize	Number of words to be transmitted (not bytes)
uint32_t rxSize	Number of words to be received (not bytes)

Returns

- CY_U3P_SUCCESS - When the SetBlockXfer is successful
- CY_U3P_ERROR_NOT_CONFIGURED - When SPI has not been configured or initialized
- CY_U3P_ERROR_ALREADY_STARTED - When a DMA transfer is active
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus\_t CyU3PSpiSetBlockXfer(  
    uint32_t txSize,  
    uint32_t rxSize  
);
```

Group

[SPI Functions](#)

See Also

- [CyU3PSpiInit](#)
- [CyU3PSpiDeInit](#)
- [CyU3PSpiSetConfig](#)
- [CyU3PSpiSetSsnLine](#)
- [CyU3PSpiTransmitWords](#)

- [CyU3PSpiReceiveWords](#)
- [CyU3PSpiWaitForBlockXfer](#)
- [CyU3PSpiDisableBlockXfer](#)

File

cyu3spi.h

9.3.2.10 CyU3PSpiWaitForBlockXfer

Wait until the ongoing SPI data transfer is finished.

This function can be used to ensure that a previous SPI transaction has completed, in the case where the callback is not being used.

Parameters

Parameters	Description
CyBool_t isRead	0: Write operation, 1: Read operation

Returns

- 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 - When the operation times out
- CY_U3P_ERROR_FAILURE - When the operation encounters an error
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus\_t CyU3PSpiWaitForBlockXfer(  
    CyBool_t isRead  
);
```

Group

[SPI Functions](#)

Notes

- 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.

- 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.

See Also

- [CyU3PSpiInit](#)
- [CyU3PSpiDeInit](#)
- [CyU3PSpiSetConfig](#)
- [CyU3PSpiSetSsnLine](#)
- [CyU3PSpiTransmitWords](#)
- [CyU3PSpiReceiveWords](#)

- [CyU3PSpiSetBlockXfer](#)
- [CyU3PSpiDisableBlockXfer](#)

File

cyu3spi.h

9.3.2.11 CyU3PSpiDisableBlockXfer

This function disabled the TX / RX DMA functionality.

The function can be called only when DMA has already been activated. The function will disable only the block requested for. If both TX and RX gets disabled, then it disables the DMA mode as well. This needs to be invoked to do register mode operations after a SetBlockXfer call.

Parameters

Parameters	Description
CyBool_t rxDisable	CyTrue: Disable TX block if active
CyBool_t txDisable	CyTrue: Disable RX block if active.
CyFalse	Ignore TX block. Ignore RX block.

Returns

- CY_U3P_SUCCESS - When the DisableBlockXfer is successful
- CY_U3P_ERROR_NOT_CONFIGURED - When SPI has not been configured or initialized no DMA transfer has been configured
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus\_t CyU3PSpiDisableBlockXfer(  
    CyBool_t rxDisable,  
    CyBool_t txDisable  
);
```

Group

[SPI Functions](#)

See Also

- [CyU3PSpiInit](#)
- [CyU3PSpiDeInit](#)
- [CyU3PSpiSetConfig](#)
- [CyU3PSpiSetSsnLine](#)
- [CyU3PSpiTransmitWords](#)
- [CyU3PSpiReceiveWords](#)
- [CyU3PSpiSetBlockXfer](#)
- [CyU3PSpiWaitForBlockXfer](#)

File

cyu3spi.h

9.3.2.12 CyU3PRegisterSpiCallBack

This function register the call back function for notification of SPI interrupt.

This function registers a callback function that will be called for notification of SPI interrupts and also selects the SPI interrupt sources of interest.

Returns

None

C++

```
void CyU3PRegisterSpiCallBack(  
    CyU3PSpiIntrCb\_t spiIntrCb  
);
```

Group

[SPI Functions](#)

See Also

- [CyU3PSpiEvt_t](#)
- [CyU3PSpiError_t](#)

File

cyu3spi.h

9.4 I2S Interface

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 and send out audio data. The I2S driver module provides functions to configure the I2S interface and to send mono or stereo audio output on the I2S link.

Group

[Serial Peripheral Interfaces](#)






Topics

Topic	Description
I2S data types	This section documents the data types that are defined as part of the I2S driver and API library.
I2S Functions	This section documents the functions defined as part of the I2S driver and API library.

9.4.1 I2S data types

This section documents the data types that are defined as part of the I2S driver and API library.



Enumerations

Enumeration	Description
 CyU3PI2sPadMode_t	List of the supported padding modes.
 CyU3PI2sSampleRate_t	List of supported sample rates.
 CyU3PI2sSampleWidth_t	List of supported bit widths.
 CyU3PI2sError_t	List of I2S specific error/status codes.
 CyU3PI2sEvt_t	List of I2S related event types.


Group

[I2S Interface](#)

Legend

	Enumeration
	Structure

Structures

Structure	Description
 CyU3PI2sConfig_t	Structure defining the configuration of the I2S interface.

Types

Type	Description
CyU3PI2sIntrCb_t	Prototype of I2S event callback function.

9.4.1.1 CyU3PI2sPadMode_t

List of the supported padding modes.

This types lists the padding modes supprted on the I2S interface.

C++

```
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
};
```

Group

[I2S data types](#)

See Also

- [CyU3PI2sConfig_t](#)

Members

Members	Description
CY_U3P_I2S_PAD_MODE_NORMAL = 0	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.

File

cyu3i2s.h

9.4.1.2 CyU3PI2sSampleRate_t

List of supported sample rates.

This type lists the supported sample rates for audio playback through the I2S interface.

C++

```
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
};
```

Group

[I2S data types](#)

See Also

- [CyU3PI2sConfig_t](#)

Members

Members	Description
CY_U3P_I2S_SAMPLE_RATE_8KHz = 8000	8 KHz
CY_U3P_I2S_SAMPLE_RATE_16KHz = 16000	16 KHz
CY_U3P_I2S_SAMPLE_RATE_32KHz = 32000	32 KHz
CY_U3P_I2S_SAMPLE_RATE_44_1KHz = 44100	44.1 KHz
CY_U3P_I2S_SAMPLE_RATE_48KHz = 48000	48 KHz
CY_U3P_I2S_SAMPLE_RATE_96KHz = 96000	96 KHz
CY_U3P_I2S_SAMPLE_RATE_192KHz = 192000	192 KHz

File

cyu3i2s.h

9.4.1.3 CyU3PI2sSampleWidth_t

List of supported bit widths.

This type lists the supported bitwidths on the I2S interface.

C++

```
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
};
```

Group

[I2S data types](#)

See Also

- [CyU3PI2sConfig_t](#)

Members

Members	Description
CY_U3P_I2S_WIDTH_8_BIT = 0	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.

File

cyu3i2s.h

9.4.1.4 CyU3PI2sError_t

List of I2S specific error/status codes.

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.

C++

```
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
};
```

Group

[I2S data types](#)

See Also

- [CyU3PI2sEvt_t](#)

Members

Members	Description
CY_U3P_I2S_ERROR_LTX_UNDERFLOW = 11	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.

File

cyu3i2s.h

9.4.1.5 CyU3PI2sEvt_t

List of I2S related event types.

This enumeration lists the various I2S related event codes that are notified to the user application through an event callback.

C++

```
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
};
```

Group

[I2S data types](#)

See Also

- [CyU3PI2sCb_t](#)
- [CyU3PI2sError_t](#)

Members

Members	Description
CY_U3P_I2S_EVENT_TXL_DONE = 0	Left Transmission is done
CY_U3P_I2S_EVENT_TXR_DONE	Right Transmission is done
CY_U3P_I2S_EVENT_PAUSED	Pause has taken effect
CY_U3P_I2S_EVENT_ERROR	Error has happened

File

cyu3i2s.h

9.4.1.6 CyU3PI2sConfig_t

Structure defining the configuration of the I2S interface.

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.

C++

```
struct CyU3PI2sConfig_t {
    CyBool_t isMono;
    CyBool_t isLsbFirst;
    CyBool_t isDma;
    CyU3PI2sPadMode\_t padMode;
    CyU3PI2sSampleRate\_t sampleRate;
    CyU3PI2sSampleWidth\_t sampleWidth;
};
```

Group

[I2S data types](#)

See Also

- [CyU3PI2sSetConfig](#)

Members

Members	Description
CyBool_t isMono;	CyTrue:Mono, CyFalse:Stereo
CyBool_t isLsbFirst;	CyTrue:LSB First, CyFalse:MSB First
CyBool_t isDma;	CyTrue:DMA mode, CyFalse:Register mode
CyU3PI2sPadMode_t padMode;	Type of padding to be used
CyU3PI2sSampleRate_t sampleRate;	Sample rate for this audio stream.
CyU3PI2sSampleWidth_t sampleWidth;	Bit width for samples in this audio stream.

File

cyu3i2s.h

9.4.1.7 CyU3PI2sIntrCb_t

Prototype of I2S event callback function.

This function type defines a callback to be called after I2S interrupt has been received. A function of this type can be registered with the LPP driver as a callback function and will be called whenever an event of interest occurs.

C++

```
typedef void (* CyU3PI2sIntrCb_t)(CyU3PI2sEvt\_t evt, CyU3PI2sError\_t error);
```

Group

[I2S data types](#)

See Also

- [CyU3PI2sEvt_t](#)
- [CyU3PI2sError_t](#)
- [CyU3PRegisterI2sCallBack](#)

File

cyu3i2s.h

9.4.2 I2S Functions

This section documents the functions defined as part of the I2S driver and API library.


Functions

Function	Description
CyU3PI2sSetClock	Function sets the required frequency for the I2S block.
CyU3PI2sStopClock	This function disables the clock of the I2S block.
CyU3PI2sInit	Starts the I2S interface block on the FX3.
CyU3PI2sDeInit	Stops the I2S interface block on the FX3.
CyU3PI2sSetConfig	Sets the I2S interface parameters.
CyU3PI2sSetMute	Mute / Un-mute I2S master.
CyU3PI2sSetPause	Pause / Resume I2S master.
CyU3PI2sTransmitBytes	Transmits data byte by byte over the I2S interface. Description. This function sends the data byte by byte over the I2S interface. This is allowed only when the I2S interface block is configured for register mode transfer. This transfer is always left justified.
CyU3PRegisterI2sCallBack	This function register the call back function for notification of I2S interrupt.

Group

[I2S Interface](#)

Legend

	Method
---	--------

9.4.2.1 CyU3PI2sSetClock

Function sets the required frequency for the I2S block.

The clock for the I2S block can be configured to required frequency.

Bit-clock calculation: The minimum sample rate supported is 8KHz and the maximum sample rate is 192KHz. The corresponding bit-clock frequency is calculated based on the sample length, number of samples and the padding mode used and is defined as per the I2S spec. The default calculation is bit clock = 64 * sample rate. It should be noted that even though the clock dividers and the API allows frequencies above and below the rated range, the device behaviour is not guaranteed. The actual sample rate is derived out of SYS_CLK. Since the divider needs to be integral or with half divider, the frequency approximation is done using the following algorithm:

If x is the actual divider and n is the required integral divider to be used, then following conditions are used to evaluate:

```
x = (SYS_CLK) / (bit clock);
```

```
if (x - floor(x)) < 0.25 ==> n = floor(x);
```

```
if (((x - floor(x)) >= 0.25) && (x - floor(x)) < 0.5) ==> n = floor(x) + half divider;
```

```
if (x - floor(x)) >= 0.75 ==> n = floor(x) + 1;
```

Parameters

Parameters	Description
uint32_t clkRate	Desired interface clock frequency.

Returns

- CY_U3P_SUCCESS - If the I2S interface clock was setup as required.
- CY_U3P_ERROR_NOT_SUPPORTED - If the FX3 device in use does not support the I2S interface.
- CY_U3P_ERROR_BAD_ARGUMENT - When invalid argument is passed to the function

C++

```
CyU3PReturnStatus\_t CyU3PI2sSetClock(  
    uint32_t clkRate  
);
```

Group

[I2S Functions](#)

See Also

- [CyU3PI2sStopClock](#)

File

cyu3lpp.h

9.4.2.2 CyU3PI2sStopClock

This function disables the clock of the I2S block.

The clock for the I2s block is disabled. This function should be called after the corresponding driver has been de-initialized.

Returns

- CY_U3P_SUCCESS - If the I2S clock is successfully turned off.
- CY_U3P_ERROR_NOT_SUPPORTED - If the I2S interface is not supported by the current FX3 device.

C++

```
CyU3PReturnStatus\_t CyU3PI2sStopClock(  
    void  
);
```

Group

[I2S Functions](#)

See Also

- [CyU3PI2sSetClock](#)

File

cyu3lpp.h

9.4.2.3 CyU3PI2sInit

Starts the I2S interface block on the FX3.

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 default value (CY_U3P_I2S_SAMPLE_RATE_8K).

Bit-clock calculation: The minimum sample rate supported is 8KHz and the maximum sample rate is 192KHz. The corresponding bit-clock frequency is calculated based on the sample length, number of samples and the padding mode used and is defined as per the I2S spec. The default calculation is bit clock = 64 * sample rate. It should be noted that even though the clock dividers and the API allows frequencies above and below the rated range, the device behaviour is not guaranteed. The actual sample rate is derived out of SYS_CLK. Since the divider needs to be integral or with half divider, the frequency approximation is done using the following algorithm:

If x is the actual divider and n is the required integral divider to be used, then following conditions are used to evaluate:

$x = (\text{SYS_CLK}) / (\text{bit clock});$

if $(x - \text{floor}(x)) < 0.25 \implies n = \text{floor}(x);$

if $((x - \text{floor}(x)) \geq 0.25) \ \&\& \ (x - \text{floor}(x)) < 0.5 \implies n = \text{floor}(x) + \text{half divider};$

if $(x - \text{floor}(x)) \geq 0.75 \implies n = \text{floor}(x) + 1;$

Returns

- 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 selected in the initial IO configuration

C++

```
CyU3PReturnStatus\_t CyU3PI2sInit(  
    void  
);
```

Group

[I2S Functions](#)

See Also

- CyU3PI2sDeinit
- [CyU3PI2sSetConfig](#)
- [CyU3PI2sTransmitBytes](#)
- [CyU3PI2sSetMute](#)
- [CyU3PI2sSetPause](#)

File

cyu3i2s.h

9.4.2.4 CyU3PI2sDeInit

Stops the I2S interface block on the FX3.

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.

Returns

- CY_U3P_SUCCESS - When the DeInit is successful
- CY_U3P_ERROR_NOT_STARTED - When the module has not been previously initialized

C++

```
CyU3PReturnStatus\_t CyU3PI2sDeInit(  
    void  
);
```

Group

[I2S Functions](#)

See Also

- [CyU3PI2sInit](#)
- [CyU3PI2sSetConfig](#)
- [CyU3PI2sTransmitBytes](#)
- [CyU3PI2sSetMute](#)
- [CyU3PI2sSetPause](#)

File

cyu3i2s.h

9.4.2.5 CyU3PI2sSetConfig

Sets the I2S interface parameters.

This function is used to configure the I2S master interface based on the desired settings. This function should be called repeatedly to change the settings if different settings are to be used.

This function can be called on the fly repetitively without calling [CyU3PI2sInit](#). But this will reset the FIFO and hence the data in pipe will be lost. If a DMA channel is present, a Reset has to be issued.

The callback parameter is used to specify an event callback function that will be called by the driver when an I2S interrupt occurs.

Parameters

Parameters	Description
CyU3PI2sConfig_t * config	I2S configuration settings
CyU3PI2sIntrCb_t cb	Callback for getting the events

Returns

- 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 - When a timeout occurs
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus\_t CyU3PI2sSetConfig(  
    CyU3PI2sConfig\_t * config,  
    CyU3PI2sIntrCb\_t cb  
);
```

Group

[I2S Functions](#)

See Also

- [CyU3PI2sCb_t](#)
- [CyU3PI2sEvt_t](#)
- [CyU3PI2sError_t](#)
- [CyU3PI2sIntrCb_t](#)
- [CyU3PI2sPadMode_t](#)
- [CyU3PI2sSampleRate_t](#)
- [CyU3PI2sSampleWidth_t](#)
- [CyU3PI2sConfig_t](#)
- [CyU3PI2sInit](#)

- `CyU3PI2sDeinit`
- [CyU3PI2sTransmitBytes](#)
- [CyU3PI2sSetMute](#)
- [CyU3PI2sSetPause](#)

File

`cyu3i2s.h`

9.4.2.6 CyU3PI2sSetMute

Mute / Un-mute I2S master.

This function will start discarding data and start driving zeros on the bus.

Parameters

Parameters	Description
CyBool_t isMute	CyTrue: Mute the I2S, CyFalse: Un-Mute the I2S

Returns

- CY_U3P_SUCCESS - When the Mute is successful
- CY_U3P_ERROR_NOT_CONFIGURED - When the I2S interface has not been configured or not been started
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus\_t CyU3PI2sSetMute(  
    CyBool_t isMute  
);
```

Group

[I2S Functions](#)

See Also

- [CyU3PI2sInit](#)
- [CyU3PI2sDeinit](#)
- [CyU3PI2sSetConfig](#)
- [CyU3PI2sTransmitBytes](#)
- [CyU3PI2sSetPause](#)

File

cyu3i2s.h

9.4.2.7 CyU3PI2sSetPause

Pause / Resume I2S master.

This function will stop sending data on the bus.

Parameters

Parameters	Description
CyBool_t isPause	CyTrue: Pause the I2S, CyFalse: Resume the I2S

Returns

- CY_U3P_SUCCESS - When the Pause is successful
- CY_U3P_ERROR_NOT_CONFIGURED - When the I2S interface has not been configured or not been started
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus\_t CyU3PI2sSetPause(  
    CyBool_t isPause  
);
```

Group

[I2S Functions](#)

See Also

- [CyU3PI2sInit](#)
- [CyU3PI2sDeinit](#)
- [CyU3PI2sSetConfig](#)
- [CyU3PI2sTransmitBytes](#)
- [CyU3PI2sSetMute](#)

File

cyu3i2s.h

9.4.2.8 CyU3PI2sTransmitBytes

Transmits data byte by byte over the I2S interface.

Description. This function sends the data byte by byte over the I2S interface. This is allowed only when the I2S interface block is configured for register mode transfer. This transfer is always left justified.

Returns

- CY_U3P_SUCCESS - When the TransmitByte 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 - When the I2S has not been configured for register mode OR the I2S interface has not been configured OR the I2S interface has not been started
- CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock

C++

```
CyU3PReturnStatus\_t CyU3PI2sTransmitBytes(  
    uint8_t * lData,  
    uint8_t * rData,  
    uint8_t lByteCount,  
    uint8_t rByteCount  
);
```

Group

[I2S Functions](#)

See Also

- [CyU3PI2sInit](#)
- [CyU3PI2sDeinit](#)
- [CyU3PI2sSetConfig](#)
- [CyU3PI2sSetMute](#)
- [CyU3PI2sSetPause](#)

File

cyu3i2s.h

9.4.2.9 CyU3PRegisterI2sCallBack

This function register the call back function for notification of I2S interrupt.

This function registers a callback function that will be called for notification of I2S interrupts and also selects the I2S interrupt sources of interest.

Returns

None

C++

```
void CyU3PRegisterI2sCallBack(  
    CyU3PI2sIntrCb\_t i2sIntrCb  
) ;
```

Group

[I2S Functions](#)

See Also

- [CyU3PI2sEvt_t](#)
- [CyU3PI2sError_t](#)

File

cyu3i2s.h

9.5 GPIO Interface

The GPIO interface manager module is responsible for handling general purpose IO pins. This section defines the data structures and software interfaces for GPIO interface management.

GPIO(general purpose I/O) pins are a special (simple) case of low performance serial peripherals that do not need DMA capability. Two modes of GPIO pins are available with FX3 devices - Simple and Complex GPIOs.

Simple GPIO provides software controlled and observable input and output capability only. Complex GPIO's contain a timer and support a variety of timed behaviors (pulsing, time measurements, one-shot etc.).

The GPIOs can only be configured individually and multiple GPIOs cannot be updated simultaneously.

Group

[Serial Peripheral Interfaces](#)






Topics

Topic	Description
GPIO data types	This section documents the enumerations and data types that are defined as part of the GPIO Interface Manager.
GPIO Functions	This section documents the API functions that are defined as part of the GPIO Interface Manager.

9.5.1 GPIO data types

This section documents the enumerations and data types that are defined as part of the GPIO Interface Manager.



Enumerations

Enumeration	Description
 CyU3PGpioIntrMode_t	Enumerated list for interrupt mode GPIOs.
 CyU3PGpioTimerMode_t	Enumerated list for timer mode GPIOs.
 CyU3PGpioComplexMode_t	Enumerated list of all GPIO complex modes.
 CyU3PGpioSimpleClkDiv_t	Clock divider values for sampling simple GPIOs.
 CyU3PGpioIoMode_t	Enumerated list for IO modes.




Group

[GPIO Interface](#)

Legend

	Enumeration
	Structure

Structures

Structure	Description
 CyU3PGpioClock_t	Clock configuration information for the GPIO block.
 CyU3PGpioSimpleConfig_t	Structure contains configuration information for simple GPIOs.
 CyU3PGpioComplexConfig_t	Structure contains configuration information for complex GPIO pins.

Types

Type	Description
CyU3PGpioIntrCb_t	Prototype of GPIO event callback function.

9.5.1.1 CyU3PGpioIntrCb_t

Prototype of GPIO event callback function.

This function type defines a callback to be called after GPIO interrupt has been received. A function of this type can be registered with the LPP driver as a callback function and will be called whenever an event of interest occurs.

C++

```
typedef void (* CyU3PGpioIntrCb_t)(uint8_t gpioId);
```

Group

[GPIO data types](#)

See Also

- [CyU3PRegisterGpioCallBack](#)

File

cyu3gpio.h

9.5.1.2 CyU3PGpioIntrMode_t

Enumerated list for interrupt mode GPIOs.

Enumerations to control the triggering of interrupts for GPIOs configured as interrupts.

C++

```
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_TIMER_THRES,  
    CY_U3P_GPIO_INTR_TIMER_ZERO  
};
```

Group

[GPIO data types](#)

See Also

- [CyU3PGpioComplexMode_t](#)
- [CyU3PGpioTimerMode_t](#)

Members

Members	Description
CY_U3P_GPIO_NO_INTR = 0	No Interrupt is triggered
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 for both edge of input.
CY_U3P_GPIO_INTR_LOW_LEVEL	Interrupt is triggered for Low level of input.
CY_U3P_GPIO_INTR_HIGH_LEVEL	Interrupt is triggered for High level of input.
CY_U3P_GPIO_INTR_TIMER_THRES	Interrupt is triggered when the timer crosses the threshold. Valid only for complex GPIO pin.
CY_U3P_GPIO_INTR_TIMER_ZERO	Interrupt is triggered when the timer reaches zero. Valid only for complex GPIO pin.

File

cyu3gpio.h

9.5.1.3 CyU3PGpioTimerMode_t

Enumerated list for timer mode GPIOs.

Enumerations to control the timer mode configuration of complex GPIO pins.

C++

```
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_ANY_EDGE,
    CY_U3P_GPIO_TIMER_RESERVED
};
```

Group

[GPIO data types](#)

See Also

- [CyU3PGpioComplexMode_t](#)
- [CyU3PGpioIntrMode_t](#)

Members

Members	Description
CY_U3P_GPIO_TIMER_SHUTDOWN = 0	Shut down timer
CY_U3P_GPIO_TIMER_HIGH_FREQ	Use High frequency (fast clock).
CY_U3P_GPIO_TIMER_LOW_FREQ	Use low frequency (slow clock).
CY_U3P_GPIO_TIMER_STANDBY_FREQ	Use standby frequency (32 KHz).
CY_U3P_GPIO_TIMER_POS_EDGE	Use positive edge of input.
CY_U3P_GPIO_TIMER_NEG_EDGE	Use negative edge of input.
CY_U3P_GPIO_TIMER_ANY_EDGE	Use dual edge of input.
CY_U3P_GPIO_TIMER_RESERVED	Reserved

File

cyu3gpio.h

9.5.1.4 CyU3PGpioComplexMode_t

Enumerated list of all GPIO complex modes.

This enumeration list complex modes supported. 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_HIGH_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 set by user. They need special APIs for operations. To use any of these modes, the complex GPIO must be configured with CY_U3P_GPIO_MODE_STATIC configuration.

C++

```
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  
};
```

Group

[GPIO data types](#)

See Also

- [CyU3PGpioTimerMode_t](#)
- [CyU3PGpioIntrMode_t](#)

Members

Members	Description
CY_U3P_GPIO_MODE_STATIC = 0	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	This mode is valid only for output configuration. Toggle value immediately, set timer = 0, then toggle output back when timer == threshold. Once this is done, set mode = static. This mode should not be set by the configure function. This will be done by the CyU3PGpioComplexPulseNow API.
CY_U3P_GPIO_MODE_PULSE	This mode is valid only for output configuration. Toggle value when timer = 0, then toggle output when timer = threshold. Once this is done it sets mode = static. This mode should not be set by the configure function. This will be done by the CyU3PGpioComplexPulse API.
CY_U3P_GPIO_MODE_PWM	This mode is valid only for output configurations. Toggle value when timer = 0, and toggle output when timer = threshold. This is a continuous operation.
CY_U3P_GPIO_MODE_MEASURE_LOW	This is valid only for input configuration. Measure the time the signal is low. It sets timer = 0 on negative edge of input, then loads threshold with timer value on positive edge of input. This is a continuous operation.
CY_U3P_GPIO_MODE_MEASURE_HIGH	This is valid only for input configuration. Measure the time the signal is high. It sets timer = 0 on positive edge of input, then loads threshold with timer value on negative edge of input. This is a continuous operation.
CY_U3P_GPIO_MODE_MEASURE_LOW_ONCE	This is valid only for input configuration. Measure the time the signal is low once. It sets timer = 0 on negative edge of input, then loads threshold with timer value on positive edge of input. This is a single operation and at the end of it, sets the mode to static. This mode should not be set by the configure function. This will be done by the CyU3PGpioComplexMeasureOnce API.
CY_U3P_GPIO_MODE_MEASURE_HIGH_ONCE	This is valid only for input configuration. Measure the time the signal is high once. It sets timer = 0 on positive edge of input, then loads threshold with timer value on negative edge of input. This is a single operation and at the end of it, sets the mode to static. This mode should not be set by the configure function. This will be done by the CyU3PGpioComplexMeasureOnce API.
CY_U3P_GPIO_MODE_MEASURE_NEG	This is valid only for input configuration. Measure when the signal goes low. It updates the threshold with value of timer on negative edge of input. This is a continuous operation.
CY_U3P_GPIO_MODE_MEASURE_POS	This is valid only for input configuration. Measure when the signal goes high. It updates the threshold with value of timer on positive edge of input. This is a continuous operation.
CY_U3P_GPIO_MODE_MEASURE_ANY	This is valid only for input configuration. Measure when the signal changes. It updates the threshold with value of timer on positive or negative edge of input. This is a continuous operation.
CY_U3P_GPIO_MODE_MEASURE_NEG_ONCE	This is valid only for input configuration. Measure when the signal goes low once. It updates the threshold with value of timer on negative edge of input. This is a single operation and at the end of it, sets the mode to static. This mode should not be set by the configure function. This will be done by the CyU3PGpioComplexMeasureOnce API.
CY_U3P_GPIO_MODE_MEASURE_POS_ONCE	This is valid only for input configuration. Measure when the signal goes high once. It updates the threshold with value of timer on positive edge of input. This is a single operation and at the end of it, sets the mode to static. This mode should not be set by the configure function. This will be done by the CyU3PGpioComplexMeasureOnce API.

CY_U3P_GPIO_MODE_MEASURE_ANY_ONCE

This is valid only for input configuration. Measure when the signal changes once. It updates the threshold with value of timer on positive or negative edge of input. This is a single operation and at the end of it, sets the mode to static. This mode should not be set by the configure function. This will be done by the [CyU3PGpioComplexMeasureOnce](#) API.

File

cyu3gpio.h

9.5.1.5 CyU3PGpioSimpleClkDiv_t

Clock divider values for sampling simple GPIOs.

Lpp modules

The simple GPIOs are always sampled at a frequency lower than the fast clock. This divider value can be selected for power optimizations.

C++

```
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
};
```

Group

[GPIO data types](#)

See Also

- [CyU3PGpioSimpleClkDiv_t](#)
- [CyU3PSysClkSrc_t](#)
- [CyU3PGpioInit](#)

Members

Members	Description
CY_U3P_GPIO_SIMPLE_DIV_BY_2 = 0	Fast clock by 2.
CY_U3P_GPIO_SIMPLE_DIV_BY_4	Fast clock by 4.
CY_U3P_GPIO_SIMPLE_DIV_BY_16	Fast clock by 16.
CY_U3P_GPIO_SIMPLE_DIV_BY_64	Fast clock by 64.
CY_U3P_GPIO_SIMPLE_NUM_DIV	Number of divider enumerations.

File

cyu3lpp.h

9.5.1.6 CyU3PGpioIoMode_t

Enumerated list for IO modes.

Enumerations to control the pull up or pull down feature on IOs. The pull-up or pull-down provided is very weak and it takes about 5us to be effective at the pads after configurations.

C++

```
enum CyU3PGpioIoMode_t {  
    CY_U3P_GPIO_IO_MODE_NONE = 0,  
    CY_U3P_GPIO_IO_MODE_WPU,  
    CY_U3P_GPIO_IO_MODE_WPD  
};
```

Group

[GPIO data types](#)

See Also

- [CyU3PGpioSetIoMode](#)

Members

Members	Description
CY_U3P_GPIO_IO_MODE_NONE = 0	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.

File

cyu3lpp.h

9.5.1.7 CyU3PGpioClock_t

Clock configuration information for the GPIO block.

The clock for the GPIO block can be configured to required frequency especially for the complex GPIO pins. The default values can be

fastClkDiv = 2 slowClkDiv = 0 simpleDiv = CY_U3P_GPIO_SIMPLE_DIV_BY_2 clkSrc = CY_U3P_SYS_CLK.

These default values must be used if only simple GPIO is required.

C++

```
struct CyU3PGpioClock_t {
    uint8_t fastClkDiv;
    uint8_t slowClkDiv;
    CyBool_t halfDiv;
    CyU3PGpioSimpleClkDiv\_t simpleDiv;
    CyU3PSysClockSrc\_t clkSrc;
};
```

Group

[GPIO data types](#)

See Also

- [CyU3PGpioSimpleClkDiv_t](#)
- [CyU3PSysClockSrc_t](#)
- [CyU3PGpioSetClock](#)

Members

Members	Description
uint8_t fastClkDiv;	Divider value for the GPIO fast clock. This is used for all complex GPIO sampling unless otherwise specified. The min value is 2 and max value is 16.
uint8_t slowClkDiv;	Divider value for the GPIO slow clock. The clock is based out of fast clock. The min value is 2 and max value is 64. If zero is used, then slow clock is not used and is disabled.
CyBool_t halfDiv;	This allows the fast clock to be divided by a non integral value of 0.5. This can be done only if slow clock is disabled.
CyU3PGpioSimpleClkDiv_t simpleDiv;	Divider value from fast clock for sampling simple GPIOs.
CyU3PSysClockSrc_t clkSrc;	The clock source to be used for this peripheral.

File

cyu3lpp.h

9.5.1.8 CyU3PGpioSimpleConfig_t

Structure contains configuration information for simple GPIOs.

The GPIO can be configured to behave in different ways.

The input and output stages can be configured separately. Since both the input and the output stages can be enabled simultaneously, care should be taken while configuration. For example, if an input pin has the output stage also enabled, the external device attached might be damaged.

For normal mode of output operation, when a GPIO needs to be configured as output, both driveLowEn and driveHighEn needs to be CyTrue and inputEn needs to be CyFalse. Similarly for normal input operation, 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.

C++

```
struct CyU3PGpioSimpleConfig_t {
    CyBool_t outValue;
    CyBool_t driveLowEn;
    CyBool_t driveHighEn;
    CyBool_t inputEn;
    CyU3PGpioIntrMode_t intrMode;
};
```

Group

[GPIO data types](#)

See Also

- [CyU3PGpioIntrMode_t](#)

Members

Members	Description
CyBool_t outValue;	Initial output on the GPIO if configured as
CyBool_t driveLowEn;	When set true, the output driver is enabled for outValue = CyFalse (0), otherwise tristated.
CyBool_t driveHighEn;	When set true, the output driver is enabled for outValue = CyTrue (1), otherwise tristated.
CyBool_t inputEn;	When set true, the input state is enabled.
CyU3PGpioIntrMode_t intrMode;	Interrupt mode for the GPIO.

File

cyu3gpio.h

9.5.1.9 CyU3PGpioComplexConfig_t

Structure contains configuration information for complex GPIO pins.

The complex GPIO can be configured to behave in different ways. The input and output stages can be configured separately. Since both the input and the output stages can be enabled simultaneously, care should be taken while configuration. For example, if an input pin has the output stage also enabled, the external device attached might be damaged.

For normal mode of output operation, when a GPIO needs to be configured as output, both driveLowEn and driveHighEn needs to be CyTrue and inputEn needs to be CyFalse. Similarly for normal input operation, 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.

The complex GPIO pins can be used to generate various signals like PWM. It can also be used to measure various input parameters like low time period. The behaviour of the pin is decided by the pinMode configuration. Refer to [CyU3PGpioComplexMode_t](#) for more details.

C++

```
struct CyU3PGpioComplexConfig_t {  
    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;  
};
```

Group

[GPIO data types](#)

See Also

- [CyU3PGpioComplexMode_t](#)
- [CyU3PGpioTimerMode_t](#)
- [CyU3PGpioIntrMode_t](#)

Members

Members	Description
CyBool_t outValue;	Initial output on the GPIO if configured as
CyBool_t driveLowEn;	When set true, the output driver is enabled for outValue = CyFalse(0), otherwise tristated.
CyBool_t driveHighEn;	When set true, the output driver is enabled for outValue = CyTrue(1), otherwise tristated.
CyBool_t inputEn;	When set true, the input state is enabled.

CyU3PGpioComplexMode_t pinMode;	The GPIO complex mode
CyU3PGpioIntrMode_t intrMode;	Interrupt mode for the GPIO.
CyU3PGpioTimerMode_t timerMode;	Timer mode.
uint32_t timer;	Timer initial value.
uint32_t period;	Timer period.
uint32_t threshold;	Timer threshold.

File






cyu3gpio.h

9.5.2 GPIO Functions

This section documents the API functions that are defined as part of the GPIO Interface Manager.

Functions


Function	Description
⇒ CyU3PGpioSetClock	This function configures and turns on the clocks used by the GPIO block on the FX3.
⇒ CyU3PGpioStopClock	This function disables the clock of the GPIO.
⇒ CyU3PGpioInit	Initializes the GPIO Interface Manager.
⇒ CyU3PGpioDeInit	De-initializes the GPIO Interface Manager.
⇒ CyU3PGpioDisable	Disables a GPIO pin.
⇒ CyU3PGpioSetSimpleConfig	Configures a simple GPIO.
⇒ CyU3PGpioSetComplexConfig	Configures a complex GPIO pin.
⇒ CyU3PGpioSetValue	Set the state of GPIO pin output.
⇒ CyU3PGpioGetValue	Query the state of GPIO input.
⇒ CyU3PDeviceGpioOverride	This function is used to override an IO as a GPIO.
⇒ CyU3PDeviceGpioRestore	This function is used to restore IO to normal mode.
⇒ CyU3PSetGpioDriveStrength	Set the IO drive strength for all the GPIOs.
⇒ CyU3PGpioSetIoMode	Set IO mode for the selected GPIO.
⇒ CyU3PGpioSimpleGetValue	Query the state of simple GPIO input.
⇒ CyU3PGpioSimpleSetValue	Set the state of simple GPIO pin output.
⇒ CyU3PGpioGetIOValues	States of all GPIOs in the system.
⇒ CyU3PGpioComplexGetThreshold	Read the latest value for complex GPIO threshold.
⇒ CyU3PGpioComplexMeasureOnce	Measures various values when the complex GPIO is configured. as input
⇒ CyU3PGpioComplexPulse	Applies a pulse on the corresponding GPIO.
⇒ CyU3PGpioComplexPulseNow	Applies a pulse on the corresponding GPIO immediately.
⇒ CyU3PGpioComplexSampleNow	Sample the GPIO timer value at an instant.
⇒ CyU3PGpioComplexUpdate	Update the complex GPIO threshold and period.

 CyU3PGpioComplexWaitForCompletion	Waits for completion of MeasureOnce, Pulse and PulseNow operations.
 CyU3PIsGpioSimpleIOConfigured	Check whether a specific pin has been selected as a simple GPIO.
 CyU3PIsGpioComplexIOConfigured	Check whether a specific pin has been selected as a complex GPIO.
 CyU3PIsGpioValid	Check whether a specified GPIO ID is valid on the current FX3 part.
 CyU3PRegisterGpioCallBack	This function register the call back function for notification of GPIO interrupt.

Group

[GPIO Interface](#)

Legend

	Method
---	--------

9.5.2.1 CyU3PGpioSetClock

This function configures and turns on the clocks used by the GPIO block on the FX3.

The GPIO block on the FX3 device makes use of two different clocks (fast clock and slow clock). This function is used to select the frequency for these clocks and to turn them on.

Parameters

Parameters	Description
CyU3PGpioClock_t * clk_p	The desired clock parameters.

Returns

- CY_U3P_SUCCESS - If the clocks were successfully configured and enabled.
- CY_U3P_ERROR_BAD_ARGUMENT - If the clock configuration specified is invalid.

C++

```
CyU3PReturnStatus\_t CyU3PGpioSetClock(  
    CyU3PGpioClock\_t * clk_p  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioClock_t](#)
- [CyU3PGpioStopClock](#)

File

cyu3lpp.h

9.5.2.2 CyU3PGpioStopClock

This function disables the clock of the GPIO.

The clock for the GPIO block is disabled. It is expected while deinitialization the GPIO.

Returns

- CY_U3P_SUCCESS - If the GPIO clocks were successfully turned off.

C++

```
CyU3PReturnStatus\_t CyU3PGpioStopClock(  
    void  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioSetClock](#)

File

cyu3lpp.h

9.5.2.3 CyU3PGpioInit

Initializes the GPIO Interface Manager.

This function is used to initialize the GPIO Interface Manager module. This function resets the GPIO hardware block and initializes all the registers. This function should be called only once during system initialization.

The application GPIO interrupt handler is registered using this function. The interrupt handler is invoked in the interrupt context based on the interrupt modes configured. Since the handler is invoked in interrupt context any blocking call should be avoided inside this handler. DMA APIs should not be invoked in this handler. Debug prints will not function in this handler. If any processing needs to be done in the handler then they will have to be handled in application thread using OS Events.

The fast clock can run at maximum frequency of $\text{SYS_CLK} / 2$ and a minimum of $\text{SYS_CLK} / 256$. The complex GPIOs can be clocked at these. The simple GPIOs will be clocked at a $\text{fast_clock} / \text{simpleDiv}$. The SYS_CLK can either be 403.2MHz or 416MHz depending on the input clock / crystal to the device. Refer to the datasheet for more information.

The slow clock, if enabled can run at a maximum frequency of $\text{fast_clock} / 2$ and a minimum of $\text{fast_clock} / 64.5$. The slow clock is only useful for setting a different clock for the timers.

There is only one instance of fast_clock and slow_clock and all the GPIOs are driven with this.

Returns

- `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

C++

```
CyU3PReturnStatus\_t CyU3PGpioInit(  
    CyU3PGpioClock\_t * clk_p,  
    CyU3PGpioIntrCb\_t irq  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioClock_t](#)
- [CyU3PGpioIntrCb_t](#)
- [CyU3PGpioDeInit](#)
- [CyU3PGpioIntrMode_t](#)
- [CyU3PGpioDisable](#)
- [CyU3PGpioDeInit](#)
- [CyU3PGpioSetSimpleConfig](#)
- [CyU3PGpioSetComplexConfig](#)
- [CyU3PGpioSetValue](#)

- [CyU3PGpioGetValue](#)
- [CyU3PGpioGetIOValues](#)
- [CyU3PGpioSimpleSetValue](#)
- [CyU3PGpioSimpleGetValue](#)
- [CyU3PGpioComplexUpdate](#)
- [CyU3PGpioComplexGetThreshold](#)
- [CyU3PGpioComplexWaitForCompletion](#)
- [CyU3PGpioComplexSampleNow](#)
- [CyU3PGpioComplexPulseNow](#)
- [CyU3PGpioComplexPulse](#)
- [CyU3PGpioComplexMeasureOnce](#)

File

`cyu3gpio.h`

9.5.2.4 CyU3PGpioDeInit

De-initializes the GPIO Interface Manager.

This function resets the GPIO hardware block and de-initializes it.

Returns

- CY_U3P_SUCCESS - If the GPIO de-init is successful
- CY_U3P_ERROR_NOT_STARTED - If the GPIO block was not previously initialized

C++

```
CyU3PReturnStatus\_t CyU3PGpioDeInit(  
    void  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioInit](#)
- [CyU3PGpioDisable](#)
- [CyU3PGpioSetSimpleConfig](#)
- [CyU3PGpioSetComplexConfig](#)
- [CyU3PGpioSetValue](#)
- [CyU3PGpioGetValue](#)
- [CyU3PGpioGetIOValues](#)
- [CyU3PGpioSimpleSetValue](#)
- [CyU3PGpioSimpleGetValue](#)
- [CyU3PGpioComplexUpdate](#)
- [CyU3PGpioComplexGetThreshold](#)
- [CyU3PGpioComplexWaitForCompletion](#)
- [CyU3PGpioComplexSampleNow](#)
- [CyU3PGpioComplexPulseNow](#)
- [CyU3PGpioComplexPulse](#)
- [CyU3PGpioComplexMeasureOnce](#)

File

cyu3gpio.h

9.5.2.5 CyU3PGpioDisable

Disables a GPIO pin.

This function is used to disable a gpio pin. A GPIO is enabled when a Config call is made.

Parameters

Parameters	Description
uint8_t gpioId	GPIO ID to be disabled

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PGpioDisable(  
    uint8_t gpioId  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioInit](#)
- [CyU3PGpioDeInit](#)
- [CyU3PGpioSetSimpleConfig](#)
- [CyU3PGpioSetComplexConfig](#)
- [CyU3PGpioSetValue](#)
- [CyU3PGpioGetValue](#)
- [CyU3PGpioGetIOValues](#)
- [CyU3PGpioSimpleSetValue](#)
- [CyU3PGpioSimpleGetValue](#)
- [CyU3PGpioComplexUpdate](#)
- [CyU3PGpioComplexGetThreshold](#)
- [CyU3PGpioComplexWaitForCompletion](#)
- [CyU3PGpioComplexSampleNow](#)
- [CyU3PGpioComplexPulseNow](#)
- [CyU3PGpioComplexPulse](#)
- [CyU3PGpioComplexMeasureOnce](#)

File

cyu3gpio.h

9.5.2.6 CyU3PGpioSetSimpleConfig

Configures a simple GPIO.

This function is used to configure and enable a simple GPIO pin. This function needs to be called before using the simple GPIO. Refer to [CyU3PGpioSimpleConfig_t](#) for more information on configuration parameters.

Parameters

Parameters	Description
uint8_t gpioId	GPIO id to be modified.
CyU3PGpioSimpleConfig_t * cfg_p	Pointer to the configure information.

Returns

- 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 GPIO pin has not been previously enabled in the IO matrix configuration

C++

```
CyU3PReturnStatus\_t CyU3PGpioSetSimpleConfig(  
    uint8_t gpioId,  
    CyU3PGpioSimpleConfig\_t * cfg_p  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioSimpleConfig_t](#)
- [CyU3PGpioInit](#)
- [CyU3PGpioDisable](#)
- [CyU3PGpioDeInit](#)
- [CyU3PGpioSetValue](#)
- [CyU3PGpioGetValue](#)
- [CyU3PGpioGetIOValues](#)
- [CyU3PGpioSimpleSetValue](#)
- [CyU3PGpioSimpleGetValue](#)

File

cyu3gpio.h

9.5.2.7 CyU3PGpioSetComplexConfig

Configures a complex GPIO pin.

This function is used to configure and enable a complex GPIO pin. This function needs to be called before using the complex GPIO pin.

There are only 8 complex GPIOs available. The 60 GPIO lines can be configured to modulo 8 complex pins. So one of GPIOs 0, 8, 16, ... can be configured to complex pin 0 and so on. It should be noted that only one of them can exist.

To use only the timer without any associated GPIO pin, the pin should be configured with `inputEn = CyFalse`, `driveLowEn = CyFalse`, `driveHighEn = CyFalse` and the timer number can be specified from 0 to 7. This override can be used only if none of the GPIOs that can be mapped to the complex pin are configured to be complex GPIO pin.

Refer to [CyU3PGpioComplexConfig_t](#) for more information on configuration parameters.

Parameters

Parameters	Description
<code>uint8_t gpioId</code>	GPIO id to be modified
<code>CyU3PGpioComplexConfig_t * cfg_p</code>	Config information.

Returns

- `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 GPIO pin has not been previously enabled in the IO matrix configuration

C++

```
CyU3PReturnStatus\_t CyU3PGpioSetComplexConfig(  
    uint8_t gpioId,  
    CyU3PGpioComplexConfig\_t * cfg_p  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioComplexMode_t](#)
- [CyU3PGpioComplexConfig_t](#)
- [CyU3PGpioInit](#)
- [CyU3PGpioDeInit](#)
- [CyU3PGpioDisable](#)

- [CyU3PGpioSetValue](#)
- [CyU3PGpioGetValue](#)
- [CyU3PGpioGetIOValues](#)
- [CyU3PGpioComplexUpdate](#)
- [CyU3PGpioComplexGetThreshold](#)
- [CyU3PGpioComplexWaitForCompletion](#)
- [CyU3PGpioComplexSampleNow](#)
- [CyU3PGpioComplexPulseNow](#)
- [CyU3PGpioComplexPulse](#)
- [CyU3PGpioComplexMeasureOnce](#)

File

cyu3gpio.h

9.5.2.8 CyU3PGpioSetValue

Set the state of GPIO pin output.

The function is valid only for output GPIOs. The function states what value needs to be pushed to the pin. A CyFalse means 0 and a CyTrue means 1. This is not failed for a pin configured as an input, but will not change anything on the hardware and the pin remains configured as input.

The function can only update the status of the selected GPIO. Output state of multiple GPIOs cannot be updated at the same time.

Parameters

Parameters	Description
uint8_t gpioId	GPIO id to be modified.
CyBool_t value	Value to set on the GPIO pin.

Returns

- 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 GPIO pin has not been previously enabled in the IO matrix configuration

C++

```
CyU3PReturnStatus_t CyU3PGpioSetValue(  
    uint8_t gpioId,  
    CyBool_t value  
) ;
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioInit](#)
- [CyU3PGpioDelInit](#)
- [CyU3PGpioDisable](#)
- [CyU3PGpioSetSimpleConfig](#)
- [CyU3PGpioSetComplexConfig](#)
- [CyU3PGpioGetValue](#)
- [CyU3PGpioGetIOValues](#)
- [CyU3PGpioSimpleSetValue](#)
- [CyU3PGpioSimpleGetValue](#)
- [CyU3PGpioComplexUpdate](#)
- [CyU3PGpioComplexGetThreshold](#)
- [CyU3PGpioComplexWaitForCompletion](#)

- [CyU3PGpioComplexSampleNow](#)
- [CyU3PGpioComplexPulseNow](#)
- [CyU3PGpioComplexPulse](#)
- [CyU3PGpioComplexMeasureOnce](#)

File

cyu3gpio.h

9.5.2.9 CyU3PGpioGetValue

Query the state of GPIO input.

For input pins this reads the latest status on the GPIO, while for output pins this returns the last updated outValue.

The function can only return the status of the selected GPIO. Input state of multiple GPIOs cannot be sampled at the same time.

Parameters

Parameters	Description
uint8_t gpioId	GPIO id to be queried.
CyBool_t * value_p	Output parameter that will be filled with the GPIO value.

Returns

- 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 GPIO pin has not been previously enabled in the IO matrix configuration

C++

```
CyU3PReturnStatus_t CyU3PGpioGetValue(  
    uint8_t gpioId,  
    CyBool_t * value_p  
) ;
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioInit](#)
- [CyU3PGpioDeInit](#)
- [CyU3PGpioDisable](#)
- [CyU3PGpioSetSimpleConfig](#)
- [CyU3PGpioSetComplexConfig](#)
- [CyU3PGpioSetValue](#)
- [CyU3PGpioGetIOValues](#)
- [CyU3PGpioSimpleSetValue](#)
- [CyU3PGpioSimpleGetValue](#)
- [CyU3PGpioComplexUpdate](#)
- [CyU3PGpioComplexGetThreshold](#)
- [CyU3PGpioComplexWaitForCompletion](#)

- [CyU3PGpioComplexSampleNow](#)
- [CyU3PGpioComplexPulseNow](#)
- [CyU3PGpioComplexPulse](#)
- [CyU3PGpioComplexMeasureOnce](#)

File

cyu3gpio.h

9.5.2.10 CyU3PDeviceGpioOverride

This function is used to override an IO as a GPIO.

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.

For example if SPI SSN is not used, then it can be configured as a GPIO and used even though SPI module is being used. [CyU3PDeviceConfigureIOMatrix](#) fails the configuration of this pin as GPIO but this function allows the configuration. The flip side is that there is no error check and if the configuration does not match the hardware configuration, it can result in unexpected behavior and hardware damage.

Parameters

Parameters	Description
uint8_t gpioId	The IO to be converted to GPIO.
CyBool_t isSimple	CyTrue: simple GPIO, CyFalse: complex GPIO

Returns

- CY_U3P_SUCCESS if the operation is successful
- CY_U3P_ERROR_BAD_ARGUMENT if the GPIO specified is invalid

C++

```
CyU3PReturnStatus\_t CyU3PDeviceGpioOverride(  
    uint8_t gpioId,  
    CyBool_t isSimple  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PDeviceGpioRestore](#)
- [CyU3PDeviceConfigureIOMatrix](#)

File

cyu3system.h

9.5.2.11 CyU3PDeviceGpioRestore

This function is used to restore IO to normal mode.

IOs that are overridden to be GPIO can be restored to normal mode of operation by invoking this API.

Parameters

Parameters	Description
uint8_t gpioId	The GPIO to be restored.

Returns

- CY_U3P_SUCCESS if the operation is successful
- CY_U3P_ERROR_BAD_ARGUMENT if the GPIO specified is invalid

C++

```
CyU3PReturnStatus\_t CyU3PDeviceGpioRestore(  
    uint8_t gpioId  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PDeviceGpioOverride](#)
- [CyU3PDeviceConfigureIOMatrix](#)

File

cyu3system.h

9.5.2.12 CyU3PSetGpioDriveStrength

Set the IO drive strength for all the GPIOs.

The function sets the IO Drive strength for all the GPIOs.

Parameters

Parameters	Description
CyU3PDriveStrengthState_t gpioDriveStrength	GPIO Drive strength

Returns

- 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

C++

```
CyU3PReturnStatus\_t CyU3PSetGpioDriveStrength(  
    CyU3PDriveStrengthState\_t gpioDriveStrength  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PDriveStrengthState_t](#)

File

cyu3lpp.h

9.5.2.13 CyU3PGpioSetIoMode

Set IO mode for the selected GPIO.

By default there is no pull-up or pull-down provided. The API can be used to provide a weak pull-up / pull-down on the IO. It takes about 5us to be active after configuration.

Parameters

Parameters	Description
uint8_t gpioId	GPIO Pin to be updated.
CyU3PGpioIoMode_t ioMode	Desired pull-up/pull-down mode.

Returns

- CY_U3P_SUCCESS - If the operation is successful
- CY_U3P_ERROR_BAD_ARGUMENT - If the gpioId or ioMode requested is invalid

C++

```
CyU3PReturnStatus_t CyU3PGpioSetIoMode(  
    uint8_t gpioId,  
    CyU3PGpioIoMode_t ioMode  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioIoMode_t](#)

File

cyu3lpp.h

9.5.2.14 CyU3PGpioSimpleGetValue

Query the state of simple GPIO input.

This is a lightweight API which does not check for errors. Use this only if a fast API access is required. For input pins this reads the latest status on the GPIO, while for output pins this returns the last updated outValue.

The function can only return the status of the selected GPIO. Input state of multiple GPIOs cannot be sampled at the same time.

Parameters

Parameters	Description
uint8_t gpioId	GPIO id to be queried.
CyBool_t * value_p	Output parameter that will be filled with the GPIO value.

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PGpioSimpleGetValue(  
    uint8_t gpioId,  
    CyBool_t * value_p  
) ;
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioInit](#)
- [CyU3PGpioDeInit](#)
- [CyU3PGpioDisable](#)
- [CyU3PGpioSetSimpleConfig](#)
- [CyU3PGpioSetValue](#)
- [CyU3PGpioGetValue](#)
- [CyU3PGpioGetIOValues](#)
- [CyU3PGpioSimpleSetValue](#)

File

cyu3gpio.h

9.5.2.15 CyU3PGpioSimpleSetValue

Set the state of simple GPIO pin output.

This is a lightweight API which does not check for errors. Use this only if a fast API access is required. The function is valid only for output GPIOs. The function states what value needs to be pushed to the pin. A CyFalse means 0 and a CyTrue means 1. This is not failed for a pin configured as an input, but will not change anything on the hardware and the pin remains configured as input.

The function can only update the status of the selected GPIO. Output state of multiple GPIOs cannot be updated at the same time.

Parameters

Parameters	Description
uint8_t gpioId	GPIO id to be modified.
CyBool_t value	Value to set on the GPIO pin.

Returns

- 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

C++

```
CyU3PReturnStatus_t CyU3PGpioSimpleSetValue(  
    uint8_t gpioId,  
    CyBool_t value  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioInit](#)
- [CyU3PGpioDeInit](#)
- [CyU3PGpioDisable](#)
- [CyU3PGpioSetSimpleConfig](#)
- [CyU3PGpioSetValue](#)
- [CyU3PGpioGetValue](#)
- [CyU3PGpioGetIOValues](#)
- [CyU3PGpioSimpleGetValue](#)

File

cyu3gpio.h

9.5.2.16 CyU3PGpioGetIOValues

States of all GPIOs in the system.

The API returns the status of every IO in the system. These values are prior to the input buffer and will return the IO states regardless of the configuration. Each bit corresponds to the IO for the system and the valid range is from 0-60. The upper 3 bits of gpioVal1 is invalid. The value is returned if the pointer passed to the function is non-zero.

Parameters

Parameters	Description
uint32_t * gpioVal0_p	32-bit bit mask for the GPIOs 0-31.
uint32_t * gpioVal1_p	32-bit bit mask for the GPIOs 32-60.

Returns

- CY_U3P_SUCCESS - If the operation is successful
- CY_U3P_ERROR_NOT_STARTED - If the GPIO block has not been initialized

C++

```
CyU3PReturnStatus\_t CyU3PGpioGetIOValues(  
    uint32_t * gpioVal0_p,  
    uint32_t * gpioVal1_p  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioInit](#)
- [CyU3PGpioDeInit](#)
- [CyU3PGpioDisable](#)
- [CyU3PGpioSetSimpleConfig](#)
- [CyU3PGpioSetComplexConfig](#)
- [CyU3PGpioSetValue](#)
- [CyU3PGpioGetValue](#)
- [CyU3PGpioSimpleSetValue](#)
- [CyU3PGpioSimpleGetValue](#)
- [CyU3PGpioComplexUpdate](#)
- [CyU3PGpioComplexGetThreshold](#)
- [CyU3PGpioComplexWaitForCompletion](#)
- [CyU3PGpioComplexSampleNow](#)
- [CyU3PGpioComplexPulseNow](#)
- [CyU3PGpioComplexPulse](#)
- [CyU3PGpioComplexMeasureOnce](#)

File

cyu3gpio.h

9.5.2.17 CyU3PGpioComplexGetThreshold

Read the latest value for complex GPIO threshold.

The function just reads the current threshold value. This is useful when the GPIO is used for measurement.

Parameters

Parameters	Description
uint8_t gpioId	The complex GPIO to sample.
uint32_t * threshold_p	Returns the current threshold value for the GPIO.

Returns

- 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.

C++

```
CyU3PReturnStatus\_t CyU3PGpioComplexGetThreshold(  
    uint8_t gpioId,  
    uint32_t * threshold_p  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioInit](#)
- [CyU3PGpioDelInit](#)
- [CyU3PGpioDisable](#)
- [CyU3PGpioSetComplexConfig](#)
- [CyU3PGpioSetValue](#)
- [CyU3PGpioGetValue](#)
- [CyU3PGpioGetIOValues](#)
- [CyU3PGpioComplexUpdate](#)
- [CyU3PGpioComplexWaitForCompletion](#)
- [CyU3PGpioComplexSampleNow](#)
- [CyU3PGpioComplexPulseNow](#)
- [CyU3PGpioComplexPulse](#)
- [CyU3PGpioComplexMeasureOnce](#)

File

cyu3gpio.h

9.5.2.18 CyU3PGpioComplexMeasureOnce

Measures various values when the complex GPIO is configured. as input

The function configures the complex GPIO to to measure the following features:

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 negative or positive edge (transition). To use this mode, the GPIO should be in input mode, and in CY_U3P_GPIO_MODE_STATIC and the timer should be operational.

This API does not wait for the measurement to complete. Invoke the [CyU3PGpioComplexWaitForCompletion](#) API to wait for completion or wait until the threshold field is non-zero. This is a busy loop until the operation is completed.

Parameters

Parameters	Description
uint8_t gpioId	The complex GPIO to measure.
CyU3PGpioComplexMode_t pinMode	Mode to be used for measurement.

Returns

- 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.

C++

```
CyU3PReturnStatus\_t CyU3PGpioComplexMeasureOnce(  
    uint8_t gpioId,  
    CyU3PGpioComplexMode\_t pinMode  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioComplexMode_t](#)
- [CyU3PGpioInit](#)
- [CyU3PGpioDelInit](#)
- [CyU3PGpioDisable](#)
- [CyU3PGpioSetComplexConfig](#)
- [CyU3PGpioSetValue](#)
- [CyU3PGpioGetValue](#)
- [CyU3PGpioGetIOValues](#)

- [CyU3PGpioComplexUpdate](#)
- [CyU3PGpioComplexGetThreshold](#)
- [CyU3PGpioComplexWaitForCompletion](#)
- [CyU3PGpioComplexSampleNow](#)
- [CyU3PGpioComplexPulseNow](#)
- [CyU3PGpioComplexPulse](#)

File

cyu3gpio.h

9.5.2.19 CyU3PGpioComplexPulse

Applies a pulse on the corresponding GPIO.

The function configures the complex GPIO to to apply a pulse on the associated GPIO. The behaviour is as follows: Toggles the GPIO value when the timer value becomes 0, then toggles output back when timer == threshold. Once this is done, the GPIO mode is reverted back to static. This API does not wait until the pulse operation is completed, but just initiates the operation. To use this mode, the GPIO should be in output mode, and in CY_U3P_GPIO_MODE_STATIC and the timer should be operational.

Parameters

Parameters	Description
uint8_t gpioId	The complex GPIO to pulse now.
uint32_t threshold	Updates the threshold value for the GPIO.

Returns

- 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.

C++

```
CyU3PReturnStatus_t CyU3PGpioComplexPulse(  
    uint8_t gpioId,  
    uint32_t threshold  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioInit](#)
- [CyU3PGpioDelInit](#)
- [CyU3PGpioDisable](#)
- [CyU3PGpioSetComplexConfig](#)
- [CyU3PGpioSetValue](#)
- [CyU3PGpioGetValue](#)
- [CyU3PGpioGetIOValues](#)
- [CyU3PGpioComplexUpdate](#)
- [CyU3PGpioComplexGetThreshold](#)
- [CyU3PGpioComplexWaitForCompletion](#)
- [CyU3PGpioComplexSampleNow](#)
- [CyU3PGpioComplexPulseNow](#)

- [CyU3PGpioComplexMeasureOnce](#)

File

cyu3gpio.h

9.5.2.20 CyU3PGpioComplexPulseNow

Applies a pulse on the corresponding GPIO immediately.

The function configures the complex GPIO to to apply a pulse on the associated GPIO. The behaviour is as follows: Toggles the GPIO value immediately, sets timer = 0, then toggles output back when timer == threshold. Once this is done, the GPIO mode is reverted back to static. This API does not wait until the pulse operation is completed, but just initiates the operation. To use this mode, the GPIO should be in output mode, and in CY_U3P_GPIO_MODE_STATIC and the timer should be operational.

Parameters

Parameters	Description
uint8_t gpioId	The complex GPIO to pulse now.
uint32_t threshold	Updates the threshold value for the GPIO.

Returns

- 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.

C++

```
CyU3PReturnStatus_t CyU3PGpioComplexPulseNow(  
    uint8_t gpioId,  
    uint32_t threshold  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioInit](#)
- [CyU3PGpioDelInit](#)
- [CyU3PGpioDisable](#)
- [CyU3PGpioSetComplexConfig](#)
- [CyU3PGpioSetValue](#)
- [CyU3PGpioGetValue](#)
- [CyU3PGpioGetIOValues](#)
- [CyU3PGpioComplexUpdate](#)
- [CyU3PGpioComplexGetThreshold](#)
- [CyU3PGpioComplexWaitForCompletion](#)
- [CyU3PGpioComplexSampleNow](#)
- [CyU3PGpioComplexPulse](#)

- [CyU3PGpioComplexMeasureOnce](#)

File

cyu3gpio.h

9.5.2.21 CyU3PGpioComplexSampleNow

Sample the GPIO timer value at an instant.

The function configures the complex GPIO to sample the current value of the timer. To use this feature, the GPIO should be configured as CY_U3P_GPIO_MODE_STATIC and the timer should be operational.

Parameters

Parameters	Description
uint8_t gpioId	The complex GPIO to sample.
uint32_t * value_p	Returns the current value of the timer when the API was invoked.

Returns

- 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.

C++

```
CyU3PReturnStatus\_t CyU3PGpioComplexSampleNow(  
    uint8_t gpioId,  
    uint32_t * value_p  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioInit](#)
- [CyU3PGpioDeInit](#)
- [CyU3PGpioDisable](#)
- [CyU3PGpioSetComplexConfig](#)
- [CyU3PGpioSetValue](#)
- [CyU3PGpioGetValue](#)
- [CyU3PGpioGetIOValues](#)
- [CyU3PGpioComplexUpdate](#)
- [CyU3PGpioComplexGetThreshold](#)
- [CyU3PGpioComplexWaitForCompletion](#)
- [CyU3PGpioComplexPulseNow](#)
- [CyU3PGpioComplexPulse](#)
- [CyU3PGpioComplexMeasureOnce](#)

File

cyu3gpio.h

9.5.2.22 CyU3PGpioComplexUpdate

Update the complex GPIO threshold and period.

The function configures the complex GPIO to update the threshold and period for the specified GPIO. This should be modified only when the GPIO mode is static or PWM.

Parameters

Parameters	Description
uint8_t gpioId	The complex GPIO to sample.
uint32_t threshold	New timer threshold value to be updated.
uint32_t period	New timer period value to be updated.

Returns

- 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.

C++

```
CyU3PReturnStatus_t CyU3PGpioComplexUpdate(  
    uint8_t gpioId,  
    uint32_t threshold,  
    uint32_t period  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioInit](#)
- [CyU3PGpioDeInit](#)
- [CyU3PGpioDisable](#)
- [CyU3PGpioSetComplexConfig](#)
- [CyU3PGpioSetValue](#)
- [CyU3PGpioGetValue](#)
- [CyU3PGpioGetIOValues](#)
- [CyU3PGpioComplexGetThreshold](#)
- [CyU3PGpioComplexWaitForCompletion](#)
- [CyU3PGpioComplexSampleNow](#)
- [CyU3PGpioComplexPulseNow](#)
- [CyU3PGpioComplexPulse](#)
- [CyU3PGpioComplexMeasureOnce](#)

File

cyu3gpio.h

9.5.2.23 CyU3PGpioComplexWaitForCompletion

Waits for completion of MeasureOnce, Pulse and PulseNow operations.

The function waits for the following modes to complete: CY_U3P_GPIO_MODE_PULSE_NOW, CY_U3P_GPIO_MODE_PULSE, CY_U3P_GPIO_MODE_MEASURE_LOW_ONCE, CY_U3P_GPIO_MODE_MEASURE_HIGH_ONCE, CY_U3P_GPIO_MODE_MEASURE_NEG_ONCE, CY_U3P_GPIO_MODE_MEASURE_POS_ONCE and CY_U3P_GPIO_MODE_MEASURE_ANY_ONCE. If the isWait option is CyTrue, this is a busy loop until the operation is completed. If the isWait option is CyFalse, it just check if the operation completed. If the operation is completed, the API returns CY_U3P_SUCCESS and if the operation is still on-going, then returns CY_U3P_ERROR_TIMEOUT. A timeout error is not an abort. It is just an indication that the operation is still on-going. The API can be called again to check if the operation is completed.

Parameters

Parameters	Description
uint8_t gpioId	The complex GPIO to wait for completion.
uint32_t * threshold_p	The current threshold value after completion.
CyBool_t isWait	Whether to wait for completion or just to check if the operation is completed.

Returns

- 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_NOT_SUPPORTED - If the GPIO is not configured correctly for this operation.

C++

```
CyU3PReturnStatus\_t CyU3PGpioComplexWaitForCompletion(  
    uint8_t gpioId,  
    uint32_t * threshold_p,  
    CyBool_t isWait  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PGpioInit](#)
- [CyU3PGpioDeInit](#)
- [CyU3PGpioDisable](#)
- [CyU3PGpioSetComplexConfig](#)
- [CyU3PGpioSetValue](#)
- [CyU3PGpioGetValue](#)
- [CyU3PGpioGetIOValues](#)
- [CyU3PGpioComplexUpdate](#)

- [CyU3PGpioComplexGetThreshold](#)
- [CyU3PGpioComplexSampleNow](#)
- [CyU3PGpioComplexPulseNow](#)
- [CyU3PGpioComplexPulse](#)
- [CyU3PGpioComplexMeasureOnce](#)

File

cyu3gpio.h

9.5.2.24 CyU3PIsGpioSimpleIOConfigured

Check whether a specific pin has been selected as a simple GPIO.

This function checks whether a specific FX3 pin has been selected to function as a simple GPIO. The IO Matrix configuration is queried for this information.

Parameters

Parameters	Description
uint32_t gpioId	Pin number to be queried.

Returns

- CyTrue if the pin is selected as a simple GPIO, CyFalse otherwise.

C++

```
CyBool_t CyU3PIsGpioSimpleIOConfigured(  
    uint32_t gpioId  
);
```

Group

[GPIO Functions](#)

See Also

- [CyU3PDeviceConfigureIOMatrix](#)
- [CyU3PIsGpioComplexIOConfigured](#)

File

cyu3system.h

9.5.2.25 CyU3PIsGpioComplexIOConfigured

Check whether a specific pin has been selected as a complex GPIO.

This function is used to check whether a specific FX3 pin has been selected to function as a complex GPIO, while setting up the IO matrix.

Returns

- CyTrue if the pin has been selected as a complex GPIO, CyFalse otherwise.

C++

```
CyBool_t CyU3PIsGpioComplexIOConfigured(  
    uint32_t gpioId  
) ;
```

Group

[GPIO Functions](#)

See Also

- [CyU3PIsGpioSimpleIOConfigured](#)
- [CyU3PDeviceConfigureIOMatrix](#)

File

cyu3system.h

9.5.2.26 CyU3PIsGpioValid

Check whether a specified GPIO ID is valid on the current FX3 part.

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.

Parameters

Parameters	Description
uint8_t gpioId	GPIO number to be checked for validity.

Returns

- CyTrue if the GPIO ID is valid, CyFalse otherwise.

C++

```
CyBool_t CyU3PIsGpioValid(  
    uint8_t gpioId  
);
```

Group

[GPIO Functions](#)

File

cyu3system.h

9.5.2.27 CyU3PRegisterGpioCallBack

This function register the call back function for notification of GPIO interrupt.

This function registers a callback function that will be called for notification of GPIO interrupts and also selects the GPIO interrupt sources of interest.

Returns

None

C++

```
void CyU3PRegisterGpioCallBack(  
    CyU3PGpioIntrCb\_t gpioIntrCb  
) ;
```

Group

[GPIO Functions](#)

See Also

- None

File

cyu3gpio.h

10 Logging Support

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

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. All logs generated by the FX3 firmware framework and library will be of this kind.
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 and 9 is the lowest. The logger implementation allows the user to set a priority threshold value at runtime, and only messages with a higher priority will be logged.

All log messages will include a source ID which identifies the thread that generated the message, the priority assigned to the message and the message ID. In case the message is a codified one, the message ID can range from 0x0000 to 0xFFFFE; and the message will also contain a 4 byte parameter field.

If the message is a verbose one, the message ID will be set to 0xFFFF and the parameter will indicate the length of the message string. The next length bytes will form the actual text of the string.


Topics

Topic	Description
Logging Data Types	This section documents the data types used for data logging.
Logging Functions	This section documents the functions defined as part of the firmware logger function.

10.1 Logging Data Types

This section documents the data types used for data logging.



Enumerations

Enumeration	Description
 CyU3PSysThreadId_t	FX3 API library thread information.


Group

[Logging Support](#)

Legend

	Enumerati on
	Structure

Structures

Structure	Description
 CyU3PDebugLog_t	FX3 debug logger data type.

10.1.1 CyU3PSysThreadId_t

FX3 API library thread information.

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.

C++

```
enum CyU3PSysThreadId_t {  
    CY_U3P_THREAD_ID_INT = 0,  
    CY_U3P_THREAD_ID_DMA = 1,  
    CY_U3P_THREAD_ID_SYSTEM = 2,  
    CY_U3P_THREAD_ID_PIB = 3,  
    CY_U3P_THREAD_ID_UIB = 4,  
    CY_U3P_THREAD_ID_LPP = 5,  
    CY_U3P_THREAD_ID_DEBUG = 8,  
    CY_U3P_THREAD_ID_LIB_MAX = 15  
};
```

Group

[Logging Data Types](#)

See Also

- [CyU3PDebugEnable](#)
- [CyU3PDebugDisable](#)

Members

Members	Description
CY_U3P_THREAD_ID_INT = 0	Interrupt context.
CY_U3P_THREAD_ID_DMA = 1	Library DMA module thread.
CY_U3P_THREAD_ID_SYSTEM = 2	Library system module thread.
CY_U3P_THREAD_ID_PIB = 3	Library p-port module thread.
CY_U3P_THREAD_ID_UIB = 4	Library USB module thread.
CY_U3P_THREAD_ID_LPP = 5	Library low performance peripheral module thread.
CY_U3P_THREAD_ID_DEBUG = 8	Library debug module thread.
CY_U3P_THREAD_ID_LIB_MAX = 15	Max library reserved thread ID.

File

cyu3system.h

10.1.2 CyU3PDebugLog_t

FX3 debug logger data type.

The structure defines the data type sent out by the logger function. For [CyU3PDebugPrint](#) function, the preamble will be the same with msg = 0xFFFF and params giving the size of the message in bytes.

C++

```
struct CyU3PDebugLog_t {
    uint8_t priority;
    uint8_t threadId;
    uint16_t msg;
    uint32_t param;
};
```

Group

[Logging Data Types](#)

See Also

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

Members

Members	Description
uint8_t priority;	Priority of the message
uint8_t threadId;	Thread Id
uint16_t msg;	Message Index
uint32_t param;	32 bit Parameter


File












cyu3system.h

10.2 Logging Functions

This section documents the functions defined as part of the firmware logging function.

Functions

Function	Description
 CyU3PDebugInit	Initialize the firmware logging functionality.

 CyU3PDebugDelInit	De-initialize the logging function.
 CyU3PDebugSysMemInit	Initializes the SYS_MEM based logger facility.
 CyU3PDebugSysMemDelInit	Disables the SYS_MEM logger.
 CyU3PDebugPrint	Free form message logging function.
 CyU3PDebugPreamble	Inhibits the preamble bytes preceding the print message.
 CyU3PDebugSetTraceLevel	This function sets the priority threshold above which debug traces will be logged.
 CyU3PDebugLog	Log a codified message.
 CyU3PDebugLogFlush	Flush any pending logs out of the internal buffers.
 CyU3PDebugLogClear	Drop any pending logs in the internal buffers.
 CyU3PDebugEnable	Enable log messages from specified threads.
 CyU3PDebugDisable	Disable log messages from all library threads.

Group

[Logging Support](#)

Legend

	Method
---	--------

10.2.1 CyU3PDebugInit

Initialize the firmware logging functionality.

This function initializes the firmware logging functionality and creates a DMA channel to send the log traces out through the selected DMA socket.

Return Values

- 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.

Parameters

Parameters	Description
CyU3PDmaSocketId_t destSckId	Socket through which the logs are to be output.
uint8_t traceLevel	Priority level beyond which logs should be output.

C++

```
CyU3PReturnStatus\_t CyU3PDebugInit(  
    CyU3PDmaSocketId\_t destSckId,  
    uint8_t traceLevel  
);
```

Group

[Logging Functions](#)

See Also

- [CyU3PDebugDeInit](#)

File

cyu3system.h

10.2.2 CyU3PDebugDeInit

De-initialize the logging function.

This function de-initializes the firmware logging function and destroys the DMA channel created to output the logs.

Returns

- CY_U3P_SUCCESS - When successful.
- CY_U3P_ERROR_NOT_STARTED - When the logger is not started.

C++

```
CyU3PReturnStatus\_t CyU3PDebugDeInit(  
    void  
);
```

Group

[Logging Functions](#)

See Also

- [CyU3PDebugInit](#)

File

cyu3system.h

10.2.3 CyU3PDebugSysMemInit

Initializes the SYS_MEM based logger facility.

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 logger can be cleared by invoking [CyU3PDebugLogClear](#).

Return Values

- 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.

Parameters

Parameters	Description
uint8_t * buffer	The buffer memory to be used for writing the log.
uint16_t size	The size of memory available for logging.
CyBool_t isWrapAround	CyTrue - wrap up and start logging from beginning CyFalse - Stop logging on reaching the last memory address.
uint8_t traceLevel	Priority level beyond which logs should be output.

C++

```
CyU3PReturnStatus\_t CyU3PDebugSysMemInit(  
    uint8_t * buffer,  
    uint16_t size,  
    CyBool_t isWrapAround,  
    uint8_t traceLevel  
);
```

Group

[Logging Functions](#)

See Also

- [CyU3PDebugSysMemDeInit](#)
- [CyU3PDebugLog](#)
- [CyU3PDebugLogClear](#)

File

cyu3system.h

10.2.4 CyU3PDebugSysMemDeInit

Disables the SYS_MEM logger.

The function de-initializes the SYS_MEM logger. The normal logger can now be initialized using [CyU3PDebugInit](#) call.

Return Values

- CY_U3P_SUCCESS - when successfully initialized.
- CY_U3P_ERROR_NOT_STARTED - when the logger is not yet started.

C++

```
CyU3PReturnStatus\_t CyU3PDebugSysMemDeInit(  
    void
```

```
);
```

Group

[Logging Functions](#)

See Also

- [CyU3PDebugSysMemInit](#)
- [CyU3PDebugLog](#)
- [CyU3PDebugLogClear](#)

File

cyu3system.h

10.2.5 CyU3PDebugPrint

Free form message logging function.

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.

Parameters

Parameters	Description
uint8_t priority	Priority level for this message.
char * message	Format specifier string. : Variable argument list.

C++

```
CyU3PReturnStatus\_t CyU3PDebugPrint(
    uint8_t priority,
    char * message,
    ...
);
```

Group

[Logging Functions](#)

Notes

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 Values

- 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](#)
- [CyU3PDebugLog](#)
- [CyU3PDebugSetTraceLevel](#)
- [CyU3PDebugPreamble](#)

File

cyu3system.h

10.2.6 CyU3PDebugPreamble

Inhibits the preamble bytes preceding the print message.

The [CyU3PDebugPrint](#) function internally sends 8 byte preamble data preceding the actual message. This preamble data is used for tool to decode the message in appropriate manner. It can be called to enable and disable sending of the preamble data. This is useful in the case where the debug data is visually interpreted (and not decoded by a tool).

Parameters

Parameters	Description
CyBool_t sendPreamble	CyFalse : disable preamble data before actual message,
CyTrue	Send preamble data before actual message

Returns

- None

C++

```
void CyU3PDebugPreamble(  
    CyBool_t sendPreamble  
);
```

Group

[Logging Functions](#)

See Also

- [CyU3PDebugPrint](#)

File

cyu3system.h

10.2.7 CyU3PDebugSetTraceLevel

This function sets the priority threshold above which debug traces will be logged.

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 Values

- None

Parameters

Parameters	Description
uint8_t level	Lowest debug trace priority level that is enabled.

C++

```
void CyU3PDebugSetTraceLevel(
    uint8_t level
);
```

Group

[Logging Functions](#)

See Also

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

File

cyu3system.h

10.2.8 CyU3PDebugLog

Log a codified message.

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 0xFFFE.

The log messages are written to a log buffer. This log buffer will be written out to the UART when it is full.

The [CyU3PDebugLogFlush](#) can be used to flush the contents of the log buffer to the UART.

The [CyU3PDebugLogClear](#) can be used to clear the contents of the log buffer.

Return Values

- 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

Parameters

Parameters	Description
uint8_t priority	Priority level for the message.
uint16_t message	Message ID for the message.
uint32_t parameter	Parameter associated with the message.

C++

```
CyU3PReturnStatus_t CyU3PDebugLog(  
    uint8_t priority,  
    uint16_t message,  
    uint32_t parameter  
);
```

Group

[Logging Functions](#)

See Also

- [CyU3PDebugPrint](#)
- [CyU3PDebugLogFlush](#)
- [CyU3PDebugLogClear](#)
- [CyU3PDebugSetTraceLevel](#)

File

cyu3system.h

10.2.9 CyU3PDebugLogFlush

Flush any pending logs out of the internal buffers.

All log messages are collected into internal memory buffers on the FX3 device and sent out to the target when the buffer is full or when a free form message is committed. This function is used to force the logger to send out any pending logs messages to the target and flush its internal buffers.

Return Values

- 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

C++

```
CyU3PReturnStatus\_t CyU3PDebugLogFlush(  
    void  
);
```

Group

[Logging Functions](#)

See Also

- [CyU3PDebugLogClear](#)

File

cyu3system.h

10.2.10 CyU3PDebugLogClear

Drop any pending logs in the internal buffers.

This function is used to ask the logger to drop any pending logs in its internal buffers.

Return Values

- 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

C++

```
CyU3PReturnStatus\_t CyU3PDebugLogClear(  
    void  
);
```

Group

[Logging Functions](#)

See Also

- [CyU3PDebugLogFlush](#)

File

cyu3system.h

10.2.11 CyU3PDebugEnable

Enable log messages from specified threads.

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 Values

- None

Parameters

Parameters	Description
uint16_t threadMask	ThreadID mask whose logs are to be enabled. 1 means enabled.

C++

```
void CyU3PDebugEnable(  
    uint16_t threadMask  
);
```

Group

[Logging Functions](#)

See Also

- [CyU3PDebugDisable](#)

File

cyu3system.h

10.2.12 CyU3PDebugDisable

Disable log messages from all library threads.

The API returns the previous threadMask set, so that it can be used for re-enabling the logs.

Return Values The previous threadMask

C++

```
uint16_t CyU3PDebugDisable(  
    void  
);
```

Group

[Logging Functions](#)

See Also

- [CyU3PDebugEnable](#)

File


cyu3system.h

11 Error Codes


Error codes returned by FX3 APIs.

The FX3 API error codes can be used to identify the cause of failure. Each API has documentation about its return values.

Enumerations

Enumeration	Description
 CyU3PErrorCode_t	List of error codes defined and returned by the FX3 firmware library.

Legend

	Enumeration
---	-------------

Types

Type	Description
CyU3PReturnStatus_t	Data type for return codes. 0 indicates success. The specific error codes <ul style="list-style-type: none">supported are defined separately.

11.1 CyU3PReturnStatus_t

Data type for return codes. 0 indicates success. The specific error codes

- supported are defined separately.

C++

```
typedef unsigned int CyU3PReturnStatus_t;
```

Group

[Error Codes](#)

File

cyu3types.h

11.2 CyU3PErrorCode_t

List of error codes defined and returned by the FX3 firmware library.

The Error codes help identify the cause of the failure. These error can be RTOS returned error codes or API returned codes. The API error codes start from 0x40.

C++

```

enum CyU3PErrorCode_t {
    CY_U3P_SUCCESS = 0,
    CY_U3P_ERROR_DELETED,
    CY_U3P_ERROR_BAD_POOL,
    CY_U3P_ERROR_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_RESUME_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_WAIT_ABORTED,
    CY_U3P_ERROR_WAIT_ABORT_FAILED,
    CY_U3P_ERROR_BAD_MUTEX,
    CY_U3P_ERROR_MUTEX_FAILURE,
    CY_U3P_ERROR_MUTEX_PUT_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_TIMEOUT,
    CY_U3P_ERROR_NOT_SUPPORTED,
    CY_U3P_ERROR_INVALID_SEQUENCE,
    CY_U3P_ERROR_ABORTED,
    CY_U3P_ERROR_DMA_FAILURE,
    CY_U3P_ERROR_FAILURE,
    CY_U3P_ERROR_BAD_INDEX,
    CY_U3P_ERROR_BAD_ENUM_METHOD,
    CY_U3P_ERROR_INVALID_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_ARBITRATION,
    CY_U3P_ERROR_NO_REENUM_REQUIRED = 0xFE,
    CY_U3P_ERROR_OPERN_DISABLED = 0xFF
};

```

Group

Error Codes

Members

Members	Description
CY_U3P_SUCCESS = 0	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 = 0x40	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.

Error Codes

CyU3PErrorCode_t

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	When the peripheral block 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_NO_REENUM_REQUIRED = 0xFE	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 = 0xFF	The requested feature / operation is not enabled in current configuration.

File






cyu3error.h

12 Utility Functions

Utility functions provided for ease of use.

The utility APIs functions are generic functions that provides easy usage function. They are also used by the FX3 library for its internal use.

Functions

Function	Description
 CyU3PBusyWait	Delay function based on busy spinning in a loop.
 CyU3PMemCopy32	Copy data word-wise from one memory location to another.
 CyU3PComputeChecksum	Compute a checksum over a user specified data buffer.
 CyU3PReadDeviceRegisters	Function to read one or more FX3 device registers.
 CyU3PWriteDeviceRegisters	Function to write one or more FX3 device registers.

Legend

	Method
---	--------

Macros

Macro	Description
CY_U3P_MAX	Maximum of two numbers.
CY_U3P_MIN	Minimum of two numbers.
CY_U3P_GET_LSB	Retrieves the LSB from a 16 bit number.
CY_U3P_GET_MSB	Retrieves the MSB from a 16 bit number.
CY_U3P_MAKEWORD	Creates a word from two eight bit numbers.
CY_U3P_DWORD_GET_BYTE0	Retrieves byte 0 from a 32 bit number.
CY_U3P_DWORD_GET_BYTE1	Retrieves byte 1 from a 32 bit number.
CY_U3P_DWORD_GET_BYTE2	Retrieves byte 2 from a 32 bit number.
CY_U3P_DWORD_GET_BYTE3	Retrieves byte 3 from a 32 bit number.
CY_U3P_MAKEDWORD	Creates a word from four eight bit numbers.
CyU3PAssert	Assert function.

12.1 CY_U3P_MAX

Maximum of two numbers.

The macro determines the larger of the two numbers passed to the function.

C++

```
#define CY_U3P_MAX(a, b) (((a) > (b)) ? (a) : (b))
```

Group

[Utility Functions](#)

File

cyu3utils.h

12.2 CY_U3P_MIN

Minimum of two numbers.

The macro determines the smaller of the two numbers passed to the function.

C++

```
#define CY_U3P_MIN(a, b) (((a) > (b)) ? (b) : (a))
```

Group

[Utility Functions](#)

File

cyu3utils.h

12.3 CY_U3P_GET_LSB

Retrieves the LSB from a 16 bit number.

The macro retrieves the least significant byte from a 16 bit unsigned number. The numbers are expected to be stored in little endian format.

C++

```
#define CY_U3P_GET_LSB(w) ((uint8_t)((w) & UINT8_MAX))
```

Group

[Utility Functions](#)

File

cyu3utils.h

12.4 CY_U3P_GET_MSB

Retrieves the MSB from a 16 bit number.

The macro retrieves the most significant byte from a 16 bit unsigned number. The numbers are expected to be stored in little endian format.

C++

```
#define CY_U3P_GET_MSB(w) ((uint8_t)((w) >> 8))
```

Group

[Utility Functions](#)

File

cyu3utils.h

12.5 CY_U3P_MAKEWORD

Creates a word from two eight bit numbers.

The macro combines two eight bit unsigned numbers to form a single 16 bit unsigned number.

C++

```
#define CY_U3P_MAKEWORD(u, l) ((uint16_t)(((u) << 8) | (l)))
```

Group

[Utility Functions](#)

File

cyu3utils.h

12.6 CY_U3P_DWORD_GET_BYTE0

Retrieves byte 0 from a 32 bit number.

The macro retrieves the first byte (LSB) from a 32 bit number. The numbers are expected to be stored in little endian format.

C++

```
#define CY_U3P_DWORD_GET_BYTE0(d) ((uint8_t)((d) & 0xFF))
```

Group

[Utility Functions](#)

File

cyu3utils.h

12.7 CY_U3P_DWORD_GET_BYTE1

Retrieves byte 1 from a 32 bit number.

The macro retrieves the second byte from a 32 bit number. The numbers are expected to be stored in little endian format.

C++

```
#define CY_U3P_DWORD_GET_BYTE1(d) ((uint8_t)(((d) >> 8) & 0xFF))
```

Group

[Utility Functions](#)

File

cyu3utils.h

12.8 CY_U3P_DWORD_GET_BYTE2

Retrieves byte 2 from a 32 bit number.

The macro retrieves the third byte from a 32 bit number. The numbers are expected to be stored in little endian format.

C++

```
#define CY_U3P_DWORD_GET_BYTE2(d) ((uint8_t)(((d) >> 16) & 0xFF))
```

Group

[Utility Functions](#)

File

cyu3utils.h

12.9 CY_U3P_DWORD_GET_BYTE3

Retrieves byte 3 from a 32 bit number.

The macro retrieves the fourth byte from a 32 bit number. The numbers are expected to be stored in little endian format.

C++

```
#define CY_U3P_DWORD_GET_BYTE3(d) ((uint8_t)(((d) >> 24) & 0xFF))
```

Group[Utility Functions](#)**File**

cyu3utils.h

12.10 CY_U3P_MAKEDWORD

Creates a word from four eight bit numbers.

The macro combines four eight bit unsigned numbers to form a single 32 bit unsigned number.

C++

```
#define CY_U3P_MAKEDWORD(b3, b2, b1, b0) ((uint32_t)((((uint32_t)(b3)) << 24) |  
(((uint32_t)(b2)) << 16) | (((uint32_t)(b1)) << 8) | ((uint32_t)(b0))))
```

Group[Utility Functions](#)**File**

cyu3utils.h

12.11 CyU3PAssert

Assert function.

If the assert expression evaluates to FALSE, the assert macro calls the `__aeabi_assert()` function.

C++

```
#define CyU3PAssert(cond) assert(cond)
```

Group[Utility Functions](#)**File**

cyu3utils.h

12.12 CyU3PBusyWait

Delay function based on busy spinning in a loop.

This function is used to insert small delays (of the order of micro-seconds) into the firmware application. The delay is implemented as a busy spin loop and can be used anywhere. The delay loop when operating at 208MHz provides about 1us. This API should not be used for large delays as other lower or same priority threads will not be able to run during this.

Parameters

Parameters	Description
uint16_t usWait	Delay duration in micro-seconds.

Returns

None

C++

```
void CyU3PBusyWait(  
    uint16_t usWait  
);
```

Group

[Utility Functions](#)

See Also

[CyU3PThreadSleep](#)

File

cyu3utils.h

12.13 CyU3PMemCopy32

Copy data word-wise from one memory location to another.

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. The implementation does not handle the case of overlapping buffers.

Parameters

Parameters	Description
uint32_t * dest	Pointer to destination buffer.

uint32_t * src	Pointer to source buffer.
uint32_t count	Size of the buffer in words to be copied.

Returns

None

C++

```
void CyU3PMemCopy32(  
    uint32_t * dest,  
    uint32_t * src,  
    uint32_t count  
);
```

Group

[Utility Functions](#)

See Also

[CyU3PMemCopy](#)

File

cyu3utils.h

12.14 CyU3PComputeChecksum

Compute a checksum over a user specified data buffer.

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.

Parameters

Parameters	Description
uint32_t * buffer	Pointer to data buffer on which to calculate the checksum.
uint8_t length	Length of data buffer on which to calculate the checksum.
uint32_t * chkSum	Pointer to buffer that will be filled with the checksum.

Returns

- CY_U3P_SUCCESS if the checksum is successfully computed.
- CY_U3P_ERROR_BAD_ARGUMENT if the parameters provided are erroneous.

C++

```
CyU3PReturnStatus\_t CyU3PComputeChecksum(  

```

```
uint32_t * buffer,  
uint8_t length,  
uint32_t * chkSum  
);
```

Group

[Utility Functions](#)

File

cyu3utils.h

12.15 CyU3PReadDeviceRegisters

Function to read one or more FX3 device registers.

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.

Parameters

Parameters	Description
uint32_t * regAddr	Address of first register to be read.
uint8_t numRefs	Number of registers to be read.
uint32_t * dataBuf	Pointer to data buffer into which the registers are to be read.

Returns

- CY_U3P_SUCCESS if the register read(s) is/are successful.
- CY_U3P_ERROR_BAD_ARGUMENT if the arguments passed in are erroneous.

C++

```
CyU3PReturnStatus\_t CyU3PReadDeviceRegisters(  
    uint32_t * regAddr,  
    uint8_t numRefs,  
    uint32_t * dataBuf  
);
```

Group

[Utility Functions](#)

See Also

- [CyU3PWriteDeviceRegisters](#)

File

cyu3utils.h

12.16 CyU3PWriteDeviceRegisters

Function to write one or more FX3 device registers.

This function is used to write one or more contiguous registers from the register space of the FX3 device.

Parameters

Parameters	Description
uvint32_t * regAddr	Address of first register to be written.
uint8_t numRefs	Number of registers to be written.
uint32_t * dataBuf	Pointer to data buffer containing data to be written to the registers.

Returns

- CY_U3P_SUCCESS if the register write(s) is/are successful.
- CY_U3P_ERROR_BAD_ARGUMENT if the arguments passed in are erroneous.

C++

```
CyU3PReturnStatus_t CyU3PWriteDeviceRegisters(  
    uvint32_t * regAddr,  
    uint8_t numRefs,  
    uint32_t * dataBuf  
);
```

Group[Utility Functions](#)**Notes**

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.

See Also

- [CyU3PReadDeviceRegisters](#)

File

cyu3utils.h

13 FX3 Boot APIs

The FX3 software boot configures minimal functionality of the FX3 device for a secondary image. This boot code does not use an RTOS and the image footprint is significantly smaller as compared to the final application image.

The software boot supports the following interfaces on the FX3 device

1. USB
2. SPI
3. I2C
4. UART
5. GPIO

A set of APIs are provided to configure these interfaces. The final image can be downloaded from the USB, SPI or I2C.

The boot library (cyfx3_boot.a) is present at \$FX3_INSTALL_PATH\firmware\boot_fw\lib

Topics

Topic	Description
FX3 Boot Device Interface	The boot device interface initializes the FX3 device and configures the system clock.
FX3 Boot USB Interface	The USB interface driver and APIs configure the USB block of the FX3 for USB2.0 or USB3.0 operation.
FX3 Boot UART Interface	The UART interface driver and APIs in the FX3 booter provide a mechanism to configure the UART properties and to do data read/write transfers through the UART interface.
FX3 Boot SPI Interface	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.
FX3 Boot I2C Interface	The FX3 booter includes a I2C interface driver and provides a set of APIs that allow the configuration of the I2C interface properties and data exchange with one or more I2C slave devices.
FX3 Boot GPIO Interface	The GPIO interface module is responsible for handling general purpose IO pins. This section defines the data structures and software interfaces for GPIO interface management. GPIO(general purpose I/O) pins are a special (simple) case of low performance serial peripherals that do not need DMA capability. Simple GPIO provides software controlled and observable input and output capability only.

13.1 FX3 Boot Device Interface

The boot device interface initializes the FX3 device and configures the system clock.

Group

[FX3 Boot APIs](#)




Topics

Topic	Description
FX3 Boot Device Data Types	This section documents the data types which are part of the FX3 Boot Device APIs.
FX3 Boot Device Functions	These section documents the functions that are part of the FX3 Boot Device APIs.

13.1.1 FX3 Boot Device Data Types

This section documents the data types which are part of the FX3 Boot Device APIs.



Enumerations

Enumeration	Description
 CyFx3BootSysClockSrc_t	Clock source for a peripheral block.
 CyFx3PartNumber_t	Enumeration of EZ-USB FX3 part numbers.
 CyFx3BootErrorCode_t	Enumeration of error codes returned by the Bootloader


Group

[FX3 Boot Device Interface](#)

Legend

	Enumerati on
	Structure

Structures

Structure	Description
 CyFx3BootIoMatrixConfig_t	Defines the IO matrix configuration parameters

13.1.1.1 CyFx3BootSysClockSrc_t

Clock source for a peripheral block.

The peripheral blocks can take various clock values based on the system clock supplied. This is all derived from the SYS_CLK_PLL supplied to the device.

C++

```
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
};
```

Group

[FX3 Boot Device Data Types](#)

Members

Members	Description
CY_FX3_BOOT_SYS_CLK_BY_16 = 0	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.
CY_FX3_BOOT_NUM_CLK_SRC	Number of clock source enumerations.

File

cyfx3device.h

13.1.1.2 CyFx3PartNumber_t

Enumeration of EZ-USB FX3 part numbers.

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.

C++

```
enum CyFx3PartNumber_t {  
    CYPART_USB3014 = 0,  
    CYPART_USB3012,  
    CYPART_USB3013,  
    CYPART_USB3011  
};
```

Group

[FX3 Boot Device Data Types](#)

See Also

- [CyFx3BootGetPartNumber](#)

Members

Members	Description
CYPART_USB3014 = 0	CYUSB3014: 512 KB RAM; GPIF can be 32 bit; UART, SPI and I2S supported.
CYPART_USB3012	CYUSB3012: 256 KB RAM; GPIF can be 32 bit; UART, SPI and I2S supported.
CYPART_USB3013	CYUSB3013: 512 KB RAM; GPIF supports 16 bit; no UART, SPI or I2S.
CYPART_USB3011	CYUSB3011: 256 KB RAM; GPIF supports 16 bit; no UART, SPI or I2S.

File

cyfx3device.h

13.1.1.3 CyFx3BootErrorCode_t

Enumeration of error codes returned by the Bootloader

C++

```
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_FAILURE
};
```

Group

[FX3 Boot Device Data Types](#)

Members

Members	Description
CY_FX3_BOOT_SUCCESS = 0x0	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	Peripheral block is not 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 application. The user need not go through the cycle of setting the descriptors and issuing connect call if this error code is returned from the CyFx3BootUsbStart() .
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_FAILURE	Generic error code.

File

cyfx3error.h

13.1.1.4 CyFx3BootIoMatrixConfig_t

Defines the IO matrix configuration parameters

This structure defines all the IO configuration parameters that are required to be set at startup.

C++

```
struct CyFx3BootIoMatrixConfig_t {
    CyBool_t isDQ32Bit;
    CyBool_t useUart;
    CyBool_t useI2C;
    CyBool_t useI2S;
    CyBool_t useSpi;
    uint32_t gpioSimpleEn[2];
};
```

Group

[FX3 Boot Device Data Types](#)

See Also

- [CyFx3BootDeviceConfigureIOMatrix](#)

Members

Members	Description
CyBool_t isDQ32Bit;	CyTrue: The GPIF bus width is 32 bit CyFalse: The GPIF bus width is 16 bit
CyBool_t useUart;	CyTrue: The UART interface is to be used CyFalse: The UART interface is not to be used
CyBool_t useI2C;	CyTrue: The I2C interface is to be used CyFalse: The I2C interface is not to be used
CyBool_t useI2S;	CyTrue: The I2S interface is to be used CyFalse: The I2S interface is not to be used Booter doesn't support I2S feature as of now.
CyBool_t useSpi;	CyTrue: The SPI interface is to be used CyFalse: The SPI interface is not to be used
uint32_t gpioSimpleEn[2];	Bitmap variable that identifies pins that should be configured as simple GPIOs.


File







cyfx3device.h

13.1.2 FX3 Boot Device Functions

These section documents the functions that are part of the FX3 Boot Device APIs.

Functions


Function	Description
 CyFx3BootDeviceInit	This function initializes the device.

 CyFx3BootDeviceConfigureIOMatrix	Configures the IO matrix for the device
 CyFx3BootWatchdogConfigure	Enable/disable the watchdog timer.
 CyFx3BootWatchdogClear	Clear the watchdog timer to prevent device reset.
 CyFx3BootJumpToProgramEntry	Function to transfer the control to the specified address.
 CyFx3BootGetPartNumber	This function is used to get the part number of the FX3 part in use.
 CyFx3BootRetainGpioState	Request to keep the GPIO block powered ON across control transfer to the full firmware.

Group

[FX3 Boot Device Interface](#)

Legend

	Method
---	--------

13.1.2.1 CyFx3BootDeviceInit

This function initializes the device.

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.

Parameters

Parameters	Description
CyBool_t setFastSysClk	Indicates whether the FX3 system clock should be set to faster than 400 MHz or not. Should be set to CyTrue if the GPIF will be used in Synchronous 32-bit mode at 100 MHz.

C++

```
void CyFx3BootDeviceInit(  
    CyBool_t setFastSysClk  
);
```

Group

[FX3 Boot Device Functions](#)

File

cyfx3device.h

13.1.2.2 CyFx3BootDeviceConfigureIOMatrix

Configures the IO matrix for the device

The function configures the GPIOs to do specialized functions. Since the pins are multiplexed, the IO matrix should be configured before using any of the ports. This function must be called after the [CyFx3BootDeviceInit](#) call from the main() function. IO matrix cannot be dynamically changed and needs to be invoked during the device initialization.

Parameters

Parameters	Description
CyFx3BootIoMatrixConfig_t * cfg_p	Pointer to Configuration parameters.

Returns

- 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

C++

```
CyFx3BootErrorCode\_t CyFx3BootDeviceConfigureIOMatrix(  
    CyFx3BootIoMatrixConfig\_t * cfg_p  
);
```

Group

[FX3 Boot Device Functions](#)

Notes

If the GPIF is 32bit, then SPI module cannot be used.

See Also

- [CyFx3BootIoMatrixConfig_t](#)

File

cyfx3device.h

13.1.2.3 CyFx3BootWatchdogConfigure

Enable/disable the watchdog timer.

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.

Parameters

Parameters	Description
CyBool_t enable	Whether the watchdog timer is to be enabled or disabled.
uint32_t period	Period for the timer in milliseconds. Used only for enable calls.

Returns

None

C++

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

Group

[FX3 Boot Device Functions](#)

See Also

- [CyFx3BootWatchdogClear](#)

File

cyfx3device.h

13.1.2.4 CyFx3BootWatchdogClear

Clear the watchdog timer to prevent device reset.

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.

Returns

None

C++

```
void CyFx3BootWatchdogClear(
    void
);
```

Group

[FX3 Boot Device Functions](#)

See Also

- [CyFx3BootWatchdogConfigure](#)

File

cyfx3device.h

13.1.2.5 CyFx3BootJumpToProgramEntry

Function to transfer the control to the specified address.

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

Parameters

Parameters	Description
uint32_t address	The program entry address

C++

```
void CyFx3BootJumpToProgramEntry(  
    uint32_t address  
);
```

Group

[FX3 Boot Device Functions](#)

File

cyfx3device.h

13.1.2.6 CyFx3BootGetPartNumber

This function is used to get the part number of the FX3 part in use.

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.

Returns

Part number of the FX3 device in use.

C++

```
CyFx3PartNumber\_t CyFx3BootGetPartNumber(  
    void  
);
```

Group

[FX3 Boot Device Functions](#)

See Also

- [CyFx3PartNumber_t](#)

File

cyfx3device.h

13.1.2.7 CyFx3BootRetainGpioState

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

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.

Returns

None

C++

```
void CyFx3BootRetainGpioState(  
    void  
);
```

Group

[FX3 Boot Device Functions](#)

File

cyfx3device.h

13.2 FX3 Boot USB Interface

The USB interface driver and APIs configure the USB block of the FX3 for USB2.0 or USB3.0 operation.

Group

[FX3 Boot APIs](#)


Topics

Topic	Description
FX3 Boot USB Data Types	This section documents the data types taht are part of the FX3 Boot USB APIs.
FX3 Boot USB Functions	These section documents the functions that are part of the FX3 Boot USB APIs.

13.2.1 FX3 Boot USB Data Types

This section documents the data types taht are part of the FX3 Boot USB APIs.

Enumerations



Enumeration	Description
 CyFx3BootUsbSpeed_t	List of supported USB connection speeds.
 CyFx3BootUsbEventType_t	Enumeration of USB event types.

 CyFx3BootUsbEpType_t	Enumeration of the endpoint types.
--	------------------------------------




Group

[FX3 Boot USB Interface](#)

Legend

	Enumeration
	Structure

Structures

Structure	Description
 CyU3PUsbDescPtrs	Pointer to the various descriptors.
 CyFx3BootUsbEp0Pkt_t	USB Default Control Pipe Setup Packet data structure.
 CyFx3BootUsbEpConfig_t	Endpoint configuration information.

Types

Type	Description
CyFx3BootUSBEventCb_t	USB Events notification callback
CyFx3BootUSBSetupCb_t	USB setup request handler type.

13.2.1.1 CyFx3BootUsbSpeed_t

List of supported USB connection speeds.

C++

```
enum CyFx3BootUsbSpeed_t {
    CY_FX3_BOOT_NOT_CONNECTED = 0x00,
    CY_FX3_BOOT_FULL_SPEED,
    CY_FX3_BOOT_HIGH_SPEED,
    CY_FX3_BOOT_SUPER_SPEED
};
```

Group

[FX3 Boot USB Data Types](#)

Members

Members	Description
CY_FX3_BOOT_NOT_CONNECTED = 0x00	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.

File

cyfx3usb.h

13.2.1.2 CyFx3BootUsbEventType_t

Enumeration of USB event types.

C++

```
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
};
```

Group

[FX3 Boot USB Data Types](#)

Members

Members	Description
CY_FX3_BOOT_USB_CONNECT = 0x00	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

File

cyfx3usb.h

13.2.1.3 CyFx3BootUsbEpType_t

Enumeration of the endpoint types.

C++

```
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  
};
```

Group

[FX3 Boot USB Data Types](#)

Members

Members	Description
CY_FX3_BOOT_USB_EP_CONTROL = 0	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

File

cyfx3usb.h

13.2.1.4 CyFx3BootUSBEventCb_t

USB Events notification callback

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 this function. This is purely an event notification function.

C++

```
typedef void (* CyFx3BootUSBEventCb_t)(CyFx3BootUsbEventType\_t event);
```

Group

[FX3 Boot USB Data Types](#)

File

cyfx3usb.h

13.2.1.5 CyFx3BootUSBSetupCb_t

USB setup request handler type.

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.

C++

```
typedef void (* CyFx3BootUSBSetupCb_t)(uint32_t setupDat0, uint32_t setupDat1);
```

Group

[FX3 Boot USB Data Types](#)

File

cyfx3usb.h

13.2.1.6 CyU3PUsbDescrPtrs

Pointer to the various descriptors.

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

C++

```
struct CyU3PUsbDescrPtrs {
    uint8_t * usbDevDesc_p;
    uint8_t * usbSSDevDesc_p;
    uint8_t * usbDevQualDesc_p;
    uint8_t * usbConfigDesc_p;
    uint8_t * usbOtherSpeedConfigDesc_p;
    uint8_t * usbHSCfgDesc_p;
    uint8_t * usbFSCfgDesc_p;
    uint8_t * usbSSCfgDesc_p;
    uint8_t * usbStringDesc_p[CY_FX3_USB_MAX_STRING_DESC_INDEX];
    uint8_t * usbIntfDescHS_p;
    uint8_t * usbIntfDescFS_p;
    uint8_t * usbIntfDescSS_p;
    uint8_t * usbSSBOSDesc_p;
};
```

Group

[FX3 Boot USB Data Types](#)

Members

Members	Description
uint8_t * usbDevDesc_p;	Pointer to device desc of device
uint8_t * usbSSDevDesc_p;	Pointer to SS device desc of device
uint8_t * usbDevQualDesc_p;	Pointer to device qualifier desc of device
uint8_t * usbConfigDesc_p;	Pointer to config desc of device
uint8_t * usbOtherSpeedConfigDesc_p;	Pointer to other speed configuration desc of device
uint8_t * usbHSCfgDesc_p;	Pointer to HIGH SPEED speed configuration desc of device
uint8_t * usbFSCfgDesc_p;	Pointer to FULL SPEED speed configuration desc of device
uint8_t * usbSSCfgDesc_p;	Pointer to SUPER SPEED speed configuration desc of device
uint8_t * usbStringDesc_p[CY_FX3_USB_MAX_STRING_DESC_INDEX];	Array of pointers to string descriptors.
uint8_t * usbIntfDescHS_p;	Pointer to interface descriptor
uint8_t * usbIntfDescFS_p;	Pointer to interface descriptor
uint8_t * usbIntfDescSS_p;	Pointer to the SS Interface Descriptor
uint8_t * usbSSBOSDesc_p;	Pointer to Super speed BOS descriptor

File

cyfx3usb.h

13.2.1.7 CyFx3BootUsbEp0Pkt_t

USB Default Control Pipe Setup Packet data structure.

Control Endpoint setup packet data strucutre.

C++

```
struct CyFx3BootUsbEp0Pkt_t {
    uint8_t bmReqType;
    uint8_t bReq;
    uint8_t bVal0;
    uint8_t bVal1;
    uint8_t bIdx0;
    uint8_t bIdx1;
    uint16_t wLen;
    uint8_t * pData;
};
```

Group

FX3 Boot USB Data Types

Members

Members	Description
uint8_t bmReqType;	Direction, type of request and intended recipient
uint8_t bReq;	Request being made
uint8_t bVal0;	Field to pass parameter with the request
uint8_t bVal1;	Field to pass parameter with the request
uint8_t bIdx0;	Field to pass parameter with the request
uint8_t bIdx1;	Field to pass parameter with the request
uint16_t wLen;	Number of bytes to be transferred as part of the data phase if any.
uint8_t * pData;	Pointer to the data

File

cyfx3usb.h

13.2.1.8 CyFx3BootUsbEpConfig_t

Endpoint configuration information.

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.

C++

```
struct CyFx3BootUsbEpConfig_t {  
    CyBool_t enable;  
    CyFx3BootUsbEpType\_t epType;  
    uint16_t streams;  
    uint16_t pktSize;  
    uint8_t burstLen;  
    uint8_t isoPkts;  
};
```

Group

[FX3 Boot USB Data Types](#)

Members

Members	Description
CyBool_t enable;	Endpoint status - enabled or not.
CyFx3BootUsbEpType_t epType;	The endpoint type
uint16_t streams;	Number of bulk streams used by the endpoint.
uint16_t pktSize;	Maximum packet size for the endpoint. Valid range <1 - 1024>
uint8_t burstLen;	Maximum burst length in packets.
uint8_t isoPkts;	Number of packets per micro-frame for ISO endpoints.

File














cyfx3usb.h

13.2.2 FX3 Boot USB Functions

These section documents the functions that are part of the FX3 Boot USB APIs.

Functions

Function	Description
CyFx3BootUsbStart	Start the USB driver of the booter.
CyFx3BootUsbSetDesc	Register a USB descriptor with the booter.
CyFx3BootUsbGetDesc	Retrieves the pointer to the USB descriptors.
CyFx3BootRegisterSetupCallback	Function to register a USB setup request handler.
CyFx3BootUsbSetEpConfig	Configure a USB endpoint's properties.

 CyFx3BootUsbGetEpCfg	Retrieve the current state of the specified endpoint.
 CyFx3BootUsbVBattEnable	Configure USB block on FX3 to work off Vbatt power instead of Vbus.
 CyFx3BootUsbConnect	Enable/Disable the USB connection.
 CyFx3BootUsbDmaXferData	Function to transfer data using the DMA
 CyFx3BootUsbGetSpeed	Get the connection speed at which USB is operating.
 CyFx3BootUsbSetClrFeature	Set/Clear Feature USB Request
 CyFx3BootUsbStall	Set or clear the stall status of an endpoint.
 CyFx3BootUsbAckSetup	Complete the status handshake of a USB control request.
 CyFx3BootUsbCheckUsb3Disconnect	This function checks if the device state is in USB 3.0 disconnect state.
 CyFx3BootUsbEp0StatusCheck	Function to check if the status stage of a EP0 transfer is pending.
 CyFx3BootUsbSendCompliancePatterns	This function is used to send the compliance patterns.
 CyFx3BootUsbLPMDisable	Disable acceptance of U1/U2 entry requests from USB 3.0 host.
 CyFx3BootUsbLPMEEnable	Re-enable acceptance of U1/U2 entry requests from USB 3.0 host.

Group

[FX3 Boot USB Interface](#)

Legend

	Method
---	--------

13.2.2.1 CyFx3BootUsbStart

Start the USB driver of the booter.

This function is used to start the USB driver of the booter. The application can register for the USB events of type [CyFx3BootUSBEventCb_t](#).

Parameters

Parameters	Description
CyBool_t noReEnum	CyTrue: Re-Enumeration will not be done.
CyFx3BootUSBEventCb_t cb	USB Event handler
CyFalse	Re-Enumeration will be done.

Returns

- CY_FX3_BOOT_ERROR_NO_REENUM_REQUIRED - FX3 Booter supports the No ReEnumeration feature that enables to have
- a single USB enumeration across the FX3 booter and the final application.
- The booter also supports the final application transferring the control
- back to the booter while keeping the USB connection active. This error
- code is returned by CyFx3BootUsbStart () to indicate that the transfer
- of control from the final application to the booter has been done successfully.
- The error code indicates that the USB connection is active and the sequence
- followed for the regular enumeration post CyFx3BootUsbStart () is to be skipped.
- CY_FX3_BOOT_SUCCESS - On Success
- None

C++

```
CyFx3BootErrorCode\_t CyFx3BootUsbStart(  
    CyBool_t noReEnum,  
    CyFx3BootUSBEventCb\_t cb  
);
```

Group

[FX3 Boot USB Functions](#)

File

cyfx3usb.h

13.2.2.2 CyFx3BootUsbSetDesc

Register a USB descriptor with the booter.

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.

Parameters

Parameters	Description
CyU3PUSBSetDescType_t descType	Type of the descriptor
uint8_t desc_index	Descriptor index for string descriptors Not applicable for other descriptors.
uint8_t * desc	Descriptor to be set

Returns

- 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.

C++

```
CyFx3BootErrorCode_t CyFx3BootUsbSetDesc(
    CyU3PUSBSetDescType_t descType,
    uint8_t desc_index,
    uint8_t * desc
);
```

Group

[FX3 Boot USB Functions](#)

Notes

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.

File

cyfx3usb.h

13.2.2.3 CyFx3BootUsbGetDesc

Retrieves the pointer to the USB descriptors.

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 [CyU3PUsbDescPtrs](#). This pointer can be used to retrieve the relevant data while handling the standard usb requests. This call is expected to be made before the [CyFx3BootUsbConnect\(\)](#) call.

Returns

Pointer of type [CyU3PUsbDescPtrs](#) on Success NULL on Failure

C++

```
CyU3PUsbDescPtrs * CyFx3BootUsbGetDesc(  
    void  
);
```

Group

[FX3 Boot USB Functions](#)

File

cyfx3usb.h

13.2.2.4 CyFx3BootRegisterSetupCallback

Function to register a USB setup request handler.

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.

Parameters

Parameters	Description
CyFx3BootUSBSetupCb_t callback	Setup request handler

Returns

- None

C++

```
void CyFx3BootRegisterSetupCallback(  
    CyFx3BootUSBSetupCb\_t callback  
);
```

Group

[FX3 Boot USB Functions](#)

File

cyfx3usb.h

13.2.2.5 CyFx3BootUsbSetEpConfig

Configure a USB endpoint's properties.

There are 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.

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.

Parameters

Parameters	Description
uint8_t ep	Ep Number to configured.
CyFx3BootUsbEpConfig_t * epinfo	EP configuration information

Returns

- 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.

C++

```
CyFx3BootErrorCode_t CyFx3BootUsbSetEpConfig(  
    uint8_t ep,  
    CyFx3BootUsbEpConfig_t * epinfo  
);
```

Group

[FX3 Boot USB Functions](#)

File

cyfx3usb.h

13.2.2.6 CyFx3BootUsbGetEpCfg

Retrieve the current state of the specified endpoint.

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.

Parameters

Parameters	Description
uint8_t ep	Endpoint number to query.
CyBool_t * isNak	Return parameter which will be filled with the NAK status.
CyBool_t * isStall	Return parameter which will be filled with the STALL status.

Returns

- CY_FX3_BOOT_SUCCESS - On Success.
- CY_FX3_BOOT_ERROR_NULL_POINTER - If isNak or the isStall parameter is NULL.

C++

```
CyFx3BootErrorCode_t CyFx3BootUsbGetEpCfg(
    uint8_t ep,
    CyBool_t * isNak,
    CyBool_t * isStall
);
```

Group

[FX3 Boot USB Functions](#)

File

cyfx3usb.h

13.2.2.7 CyFx3BootUsbVBattEnable

Configure USB block on FX3 to work off Vbatt power instead of Vbus.

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.

Parameters

Parameters	Description
CyBool_t enable	CyTrue:Work off Vbatt, CyFalse: Do not work off VBatt

Returns

- None

C++

```
void CyFx3BootUsbVBattEnable(  
    CyBool_t enable  
);
```

Group

[FX3 Boot USB Functions](#)

Notes

This call is expected to be done prior to the [CyFx3BootUsbConnect](#).

File

cyfx3usb.h

13.2.2.8 CyFx3BootUsbConnect

Enable/Disable the USB connection.

This function is used to enable the USB PHYs and to control the connection to the USB host in that manner.

ssEnable parameter controls the SS or HS/FS enumeration. Even if SS enumeration fails there is fallback to HS/FS enumeration.

connect parameter enables/disables the USB connection. On connect the appropriate USB PHY is enabled and on disconnect the USB PHY is disabled.

Parameters

Parameters	Description
CyBool_t connect	0 - Disable USB Connection 1 - Enable USB Connection
CyBool_t ssEnable	0 - HS/FS Enumeration 1 - SS Enumeration

Returns

- None

C++

```
void CyFx3BootUsbConnect(
    CyBool_t connect,
    CyBool_t ssEnable
);
```

Group

[FX3 Boot USB Functions](#)

File

cyfx3usb.h

13.2.2.9 CyFx3BootUsbDmaXferData

Function to transfer data using the DMA

This function is used to create a one shot DMA channel to transfer to the data. This function doesn't verify the validity of the parameters, the application is expected to pass the correct parameters. Incorrect parameters might result in erroneous behaviour.

Parameters

Parameters	Description
uint8_t epNum	Endpoint number to/from which to transfer data.
uint32_t address	SYSMEM address from/to which data is to be transferred.
uint32_t length	Length in bytes
uint32_t timeout	Timeout in 100s of us. Also refer the macros CY_FX3_BOOT_NO_WAIT and CY_FX3_BOOT_WAIT_FOREVER

Returns

- CY_FX3_BOOT_SUCCESS - In case of succesful DMA transfer.
- CY_FX3_BOOT_ERROR_XFER_FAILURE - In case of DMA transfer failure.
- CY_FX3_BOOT_ERROR_TIMEOUT - In case of DMA transfer timesout.

C++

```
CyFx3BootErrorCode_t CyFx3BootUsbDmaXferData(  
    uint8_t epNum,  
    uint32_t address,  
    uint32_t length,  
    uint32_t timeout  
);
```

Group

[FX3 Boot USB Functions](#)

File

cyfx3usb.h

13.2.2.10 CyFx3BootUsbGetSpeed

Get the connection speed at which USB is operating.

This function is used to get the operating speed of the USB connection.

Returns

One of the value of the enum [CyFx3BootUsbSpeed_t](#)

C++

```
CyFx3BootUsbSpeed\_t CyFx3BootUsbGetSpeed(  
    void  
);
```

Group

[FX3 Boot USB Functions](#)

File

cyfx3usb.h

13.2.2.11 CyFx3BootUsbSetClrFeature

Set/Clear Feature USB Request

This function supports endpoint set/clear feature.

Parameters

Parameters	Description
uint32_t sc	1 - Set feature 0 - Clear feature
CyFx3BootUsbEp0Pkt_t * pEp0	Pointer to the EP0 packet

Returns

- CY_FX3_BOOT_SUCCESS - On Success.
- CY_FX3_BOOT_ERROR_FAILURE - On Failure.

C++

```
CyFx3BootErrorCode_t CyFx3BootUsbSetClrFeature(  
    uint32_t sc,  
    CyFx3BootUsbEp0Pkt_t * pEp0  
);
```

Group

[FX3 Boot USB Functions](#)

File

cyfx3usb.h

13.2.2.12 CyFx3BootUsbStall

Set or clear the stall status of an endpoint.

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.

Parameters

Parameters	Description
uint8_t ep	Endpoint number to be modified.
CyBool_t stall	CyTrue: Set the stall condition, CyFalse: Clear the stall
CyBool_t toggle	CyTrue: Clear the data toggles in a Clear Stall call

Returns

- None

C++

```
void CyFx3BootUsbStall(
    uint8_t ep,
    CyBool_t stall,
    CyBool_t toggle
);
```

Group

[FX3 Boot USB Functions](#)

File

cyfx3usb.h

13.2.2.13 CyFx3BootUsbAckSetup

Complete the status handshake of a USB control request.

This function is used to complete the status handshake of a USB control request that does not involve any data transfer.

C++

```
void CyFx3BootUsbAckSetup(  
    void  
);
```

Group

[FX3 Boot USB Functions](#)

File

cyfx3usb.h

13.2.2.14 CyFx3BootUsbCheckUsb3Disconnect

This function checks if the device state is in USB 3.0 disconnect state.

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.

Returns

- None

C++

```
void CyFx3BootUsbCheckUsb3Disconnect(
    void
);
```

Group

[FX3 Boot USB Functions](#)

File

cyfx3usb.h

13.2.2.15 CyFx3BootUsbEp0StatusCheck

Function to check if the status stage of a EP0 transfer is pending.

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.

Returns

- None

C++

```
void CyFx3BootUsbEp0StatusCheck(  
    void  
);
```

Group

[FX3 Boot USB Functions](#)

File

cyfx3usb.h

13.2.2.16 CyFx3BootUsbSendCompliancePatterns

This function is used to send the compliance patterns.

This function is used to send the 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.

Returns

- None

C++

```
void CyFx3BootUsbSendCompliancePatterns(
    void
);
```

Group

[FX3 Boot USB Functions](#)

File

cyfx3usb.h

13.2.2.17 CyFx3BootUsbLPMDisable

Disable acceptance of U1/U2 entry requests from USB 3.0 host.

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.

Returns

- None

C++

```
void CyFx3BootUsbLPMDisable(  
    void  
);
```

Group

[FX3 Boot USB Functions](#)

See Also

- [CyFx3BootUsbLPMEable](#)

File

cyfx3usb.h

13.2.2.18 CyFx3BootUsbLPMEnable

Re-enable acceptance of U1/U2 entry requests from USB 3.0 host.

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.

Returns

- None

C++

```
void CyFx3BootUsbLPMEnable(
    void
);
```

Group

[FX3 Boot USB Functions](#)

See Also

- [CyFx3BootUsbLPMDisable](#)

File

cyfx3usb.h

13.3 FX3 Boot UART Interface

The UART interface driver and APIs in the FX3 booter provide a mechanism to configure the UART properties and to do data read/write transfers through the UART interface.

Group

[FX3 Boot APIs](#)




Topics

Topic	Description
FX3 Boot UART Data Types	This section documents the data types that are defined as part of the Boot UART APIs.
FX3 Boot UART Functions	This section documents the functions that are defined as part of the Boot UART APIs.

13.3.1 FX3 Boot UART Data Types

This section documents the data types that are defined as part of the Boot UART APIs.



Enumerations

Enumeration	Description
 CyFx3BootUartBaudrate_t	List of baud rates supported by the UART.
 CyFx3BootUartParity_t	List of parity settings supported by the UART interface.
 CyFx3BootUartStopBit_t	List of number of stop bits to be used in UART communication.


Group

[FX3 Boot UART Interface](#)

Legend

	Enumeration
	Structure

Structures

Structure	Description
 CyFx3BootUartConfig_t	Configuration parameters for the UART interface.

13.3.1.1 CyFx3BootUartBaudrate_t

List of baud rates supported by the UART.

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.

C++

```
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
};
```

Group

[FX3 Boot UART Data Types](#)

See Also

- [CyFx3BootUartConfig_t](#)

Members

Members	Description
CY_FX3_BOOT_UART_BAUDRATE_4800 = 4800	4800 baud.
CY_FX3_BOOT_UART_BAUDRATE_9600 = 9600	9600 baud.
CY_FX3_BOOT_UART_BAUDRATE_19200 = 19200	19200 baud.
CY_FX3_BOOT_UART_BAUDRATE_38400 = 38400	38400 baud.
CY_FX3_BOOT_UART_BAUDRATE_57600 = 57600	57600 baud.
CY_FX3_BOOT_UART_BAUDRATE_115200 = 115200	115200 baud.

File

cyfx3uart.h

13.3.1.2 CyFx3BootUartParity_t

List of parity settings supported by the UART interface.

This enumeration lists the various parity settings that the UART interface can be configured to support.

C++

```
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
};
```

Group

[FX3 Boot UART Data Types](#)

See Also

- [CyFx3BootUartConfig_t](#)

Members

Members	Description
CY_FX3_BOOT_UART_NO_PARITY = 0	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.

File

cyfx3uart.h

13.3.1.3 CyFx3BootUartStopBit_t

List of number of stop bits to be used in UART communication.

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.

C++

```
enum CyFx3BootUartStopBit_t {
    CY_FX3_BOOT_UART_ONE_STOP_BIT = 1,
    CY_FX3_BOOT_UART_TWO_STOP_BIT = 2
};
```

Group

[FX3 Boot UART Data Types](#)

See Also

- [CyFx3BootUartConfig_t](#)

Members

Members	Description
CY_FX3_BOOT_UART_ONE_STOP_BIT = 1	1 stop bit
CY_FX3_BOOT_UART_TWO_STOP_BIT = 2	2 stop bit

File

cyfx3uart.h

13.3.1.4 CyFx3BootUartConfig_t

Configuration parameters for the UART interface.

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.

The isDma member specifies whether the UART should be configured to transfer data one byte at a time, or in terms of large (user configurable size) blocks.

All of the parameters can be changed dynamically by calling the [CyFx3BootUartSetConfig](#) function repeatedly.

C++

```
struct CyFx3BootUartConfig_t {  
    CyBool_t txEnable;  
    CyBool_t rxEnable;  
    CyBool_t flowCtrl;  
    CyBool_t isDma;  
    CyFx3BootUartBaudrate\_t baudRate;  
    CyFx3BootUartStopBit\_t stopBit;  
    CyFx3BootUartParity\_t parity;  
};
```

Group

[FX3 Boot UART Data Types](#)

See Also

- [CyFx3BootUartBaudrate_t](#)
- [CyFx3BootUartStopBit_t](#)
- [CyFx3BootUartParity_t](#)
- [CyFx3BootUartSetConfig](#)

Members

Members	Description
CyBool_t txEnable;	Enable the transmitter.
CyBool_t rxEnable;	Enable the receiver.
CyBool_t flowCtrl;	Enable Flow control for Both RX and TX.
CyBool_t isDma;	CyFalse: Byte by byte transfer; CyTrue: Block based transfer.
CyFx3BootUartBaudrate_t baudRate;	Baud rate for data transfer.
CyFx3BootUartStopBit_t stopBit;	The number of stop bits appended.
CyFx3BootUartParity_t parity;	Parity configuration









File

cyfx3uart.h

13.3.2 FX3 Boot UART Functions

This section documents the functions that are defined as part of the Boot UART APIs.

Functions

Function	Description
 CyFx3BootUartInit	Starts the UART hardware block on the device.
 CyFx3BootUartDeInit	Stops the UART hardware block.
 CyFx3BootUartSetConfig	Sets the UART interface parameters.
 CyFx3BootUartTxSetBlockXfer	Sets the number of bytes to be transmitted by the UART.
 CyFx3BootUartRxSetBlockXfer	Sets the number of bytes to be received by the UART.
 CyFx3BootUartTransmitBytes	Transmits data through the UART interface on a byte by byte basis.
 CyFx3BootUartReceiveBytes	Receives data from the UART interface on a byte by byte basis.
 CyFx3BootUartDmaXferData	This function is used to setup a dma from CPU to UART or vice versa.

Group

[FX3 Boot UART Interface](#)

Legend

	Method
---	--------

13.3.2.1 CyFx3BootUartInit

Starts the UART hardware block on the device.

This function powers up the UART hardware block on the device and should be the first UART related function called by the application.

Returns

- CY_FX3_BOOT_SUCCESS - When the UART block is successfully initialized
- CY_FX3_BOOT_ERROR_NOT_SUPPORTED - If the FX3 part in use does not support the UART interface

C++

```
CyFx3BootErrorCode\_t CyFx3BootUartInit(  
    void  
);
```

Group

[FX3 Boot UART Functions](#)

See Also

- [CyFx3BootUartDeInit](#)

File

cyfx3uart.h

13.3.2.2 CyFx3BootUartDeInit

Stops the UART hardware block.

This function disables and powers off the UART hardware block on the device.

Returns

- None

C++

```
void CyFx3BootUartDeInit(
    void
);
```

Group

[FX3 Boot UART Functions](#)

See Also

- [CyFx3BootUartInit](#)

File

cyfx3uart.h

13.3.2.3 CyFx3BootUartSetConfig

Sets the UART interface parameters.

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.

Parameters

Parameters	Description
CyFx3BootUartConfig_t * config	Pointer to structure containing config information

Returns

- 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

C++

```
CyFx3BootErrorCode\_t CyFx3BootUartSetConfig(  
    CyFx3BootUartConfig\_t * config  
);
```

Group

[FX3 Boot UART Functions](#)

See Also

- [CyFx3BootUartConfig_t](#)

File

cyfx3uart.h

13.3.2.4 CyFx3BootUartTxSetBlockXfer

Sets the number of bytes to be transmitted by the UART.

This function sets the size of the desired data transmission through the UART. Infinite transfers are not supported.

This function is to be used when the UART is configured for DMA mode of transfer.

Parameters

Parameters	Description
uint32_t txSize	Desired transfer size.

Returns

- None

C++

```
void CyFx3BootUartTxSetBlockXfer(
    uint32_t txSize
);
```

Group

[FX3 Boot UART Functions](#)

See Also

- [CyFx3BootUartRxSetBlockXfer](#)

File

cyfx3uart.h

13.3.2.5 CyFx3BootUartRxSetBlockXfer

Sets the number of bytes to be received by the UART.

This function sets the size of the desired data reception through the UART. Infinite transfers are not supported.

This function is to be used when the UART is configured for DMA mode of transfer.

Parameters

Parameters	Description
uint32_t rxSize	Desired transfer size.

Returns

- None

C++

```
void CyFx3BootUartRxSetBlockXfer(  
    uint32_t rxSize  
);
```

Group

[FX3 Boot UART Functions](#)

See Also

- [CyFx3BootUartTxSetBlockXfer](#)

File

cyfx3uart.h

13.3.2.6 CyFx3BootUartTransmitBytes

Transmits data through the UART interface on a byte by byte basis.

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.

Parameters

Parameters	Description
uint8_t * data_p	Pointer to the data to be transferred.
uint32_t count	Number of bytes to be transferred.

Returns

0 - if the transmit bytes function failed. The failure could be due to the invalid parameters having been passed to the function or the block not being setup correctly. Non-zero value - Number of bytes that were successfully transmitted. This may not be the same as that of the expected bytes to be transmitted. The caller is expected to handle these checks.

C++

```
uint32_t CyFx3BootUartTransmitBytes(
    uint8_t * data_p,
    uint32_t count
);
```

Group

[FX3 Boot UART Functions](#)

See Also

- [CyFx3BootUartReceiveBytes](#)

File

cyfx3uart.h

13.3.2.7 CyFx3BootUartReceiveBytes

Receives data from the UART interface on a byte by byte basis.

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.

Parameters

Parameters	Description
uint8_t * data_p	Pointer to location where the data read is to be placed.
uint32_t count	Number of bytes to be received.

Returns

0 - if the receive bytes function failed. The failure could be due to the invalid parameters having been passed to the function or the block not being setup correctly. Non-zero value - Number of bytes that were successfully received. This may not be the same as that of the expected bytes to be received. The caller is expected to handle these checks. Number of bytes that are successfully received.

C++

```
uint32_t CyFx3BootUartReceiveBytes(  
    uint8_t * data_p,  
    uint32_t count  
);
```

Group

[FX3 Boot UART Functions](#)

See Also

- [CyFx3BootUartTransmitBytes](#)

File

cyfx3uart.h

13.3.2.8 CyFx3BootUartDmaXferData

This function is used to setup a dma from CPU to UART or vice versa.

This function is a blocking call. This function is used to read/write length number of bytes from/to UART. This function can be used only if the UART has been configured for DMA transfer mode. Infinite transfers are not supported.

Parameters

Parameters	Description
CyBool_t isRead	isRead = CyTrue for read operations isRead = CyFalse for write operations
uint32_t address	address of the buffer from/to which data is to be transferred
uint32_t length	length of the data to be transferred
uint32_t timeout	Timeout in 100s of us. Also refer the macros CY_FX3_BOOT_NO_WAIT and CY_FX3_BOOT_WAIT_FOREVER

Returns

- CY_FX3_BOOT_SUCCESS - if data transfer is successful
- CY_FX3_BOOT_ERROR_XFER_FAILURE - if the data transfer encountered any error
- CY_FX3_BOOT_ERROR_TIMEOUT - if the data transfer times out

C++

```
CyFx3BootErrorCode_t CyFx3BootUartDmaXferData(
    CyBool_t isRead,
    uint32_t address,
    uint32_t length,
    uint32_t timeout
);
```

Group

[FX3 Boot UART Functions](#)

See Also

- [CyFx3BootUartTransmitBytes](#)
- [CyFx3BootUartReceiveBytes](#)
- [CyFx3BootUartTxSetBlockXfer](#)
- [CyFx3BootUartRxSetBlockXfer](#)

File

cyfx3uart.h

13.4 FX3 Boot SPI Interface

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.

Group

[FX3 Boot APIs](#)



Topics

Topic	Description
FX3 Boot SPI Data Types	This section documents the enums and data structures that are defined as part of the SPI Interface Driver.
FX3 Boot SPI Functions	This section documents the API functions that are defined as part of the SPI Interface Driver.

13.4.1 FX3 Boot SPI Data Types

This section documents the enums and data structures that are defined as part of the SPI Interface Driver.



Enumerations

Enumeration	Description
 CyFx3BootSpiSsnLagLead_t	Enumeration defining the lead and lag times of SSN with respect to SCK.
 CyFx3BootSpiSsnCtrl_t	Enumeration defining the various ways in which the SSN for a SPI device can be controlled.


Group

[FX3 Boot SPI Interface](#)

Legend

	Enumeration
	Structure

Structures

Structure	Description
 CyFx3BootSpiConfig_t	Structure defining the configuration of SPI interface.

13.4.1.1 CyFx3BootSpiSsnLagLead_t

Enumeration defining the lead and lag times of SSN with respect to SCK.

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.

C++

```
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
};
```

Group

[FX3 Boot SPI Data Types](#)

See Also

- [CyFx3BootSpiConfig_t](#)
- [CyFx3BootSpiSetConfig](#)

Members

Members	Description
CY_FX3_BOOT_SPI_SSN_LAG_LEAD_ZERO_CLK = 0	SSN is 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.

File

cyfx3spi.h

13.4.1.2 CyFx3BootSpiSsnCtrl_t

Enumeration defining the various ways in which the SSN for a SPI device can be controlled.

The SSN can be controlled by the firmware (API) which is not synchronized with SPI clock. It is asserted at the beginning of a transfer and is de-asserted at the end of the transfer.

The SSN can be controlled by the hardware. In this case the SSN assert and de-asserts are done in sync with SPI clock.

The SSN can be controlled by the application, external to the API. This can be done using GPIOs. In this case the hardware / API is not aware of the SSN.

The APIs allow only control of one SSN line. If there are more than one SPI slave device, then at-most one device can be controlled by hardware / API.

For example, if there are two devices, and SSN for one is controlled by hardware / API, SetConfig request needs to be called whenever the device to be addressed changes. On the other hand if the SSN for both devices are controlled externally (via GPIO), then SetConfig needs to be called only once, provided rest of the configuration stays the same.

C++

```
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_BASED,
    CY_FX3_BOOT_SPI_SSN_CTRL_NONE,
    CY_FX3_BOOT_SPI_NUM_SSN_CTRL
};
```

Group

FX3 Boot SPI Data Types

See Also

- [CyFx3BootSpiConfig_t](#)
- [CyFx3BootSpiSetConfig](#)

Members

Members	Description
CY_FX3_BOOT_SPI_SSN_CTRL_FW = 0	SSN is controlled by API and is not at clock boundaries. It is asserted at beginning of transfer is de-asserted at end of transfer.
CY_FX3_BOOT_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 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 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. So if a transfer consists of 64 word, the SSN is asserted and de-asserted 64 times.
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 is ignored by the hardware / API.
CY_FX3_BOOT_SPI_NUM_SSN_CTRL	Number of enumerations.

File

cyfx3spi.h

13.4.1.3 CyFx3BootSpiConfig_t

Structure defining the configuration of SPI interface.

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.

C++

```
struct CyFx3BootSpiConfig_t {
    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;
};
```

Group

[FX3 Boot SPI Data Types](#)

See Also

- [CyFx3BootSpiSsnCtrl_t](#)
- [CyFx3BootSpiSclkParam_t](#)
- [CyFx3BootSpiSetConfig](#)

Members

Members	Description
CyBool_t isLsbFirst;	Data shift mode - CyFalse: MSB first, CyTrue: LSB first
CyBool_t cpol;	Clock polarity - CyFalse(0): SCK idles low, CyTrue(1): SCK idles high
CyBool_t cpha;	Clock phase - CyFalse(0): Slave samples at idle-active edge, CyTrue(1): Slave samples at active-idle edge
CyBool_t ssnPol;	Polarity of SSN line. CyFalse (0): SSN is active low, CyTrue (1): SSN is active high.
CyFx3BootSpiSsnCtrl_t ssnCtrl;	SSN control
CyFx3BootSpiSsnLagLead_t leadTime;	Time between SSN's assertion and first SCLK's edge. This is at the beginning of a transfer and is valid only for hardware controlled SSN. Zero lead time is not supported.
CyFx3BootSpiSsnLagLead_t lagTime;	Time between the last SCK edge to SSN's de-assertion. This is at the end of a transfer and is valid only for hardware controlled SSN. For CPHA = 1, lag time cannot be zero.
uint32_t clock;	SPI clock frequency in Hz.
uint8_t wordLen;	Word length in bits. Valid values are 4 - 32.










File

cyfx3spi.h

13.4.2 FX3 Boot SPI Functions

This section documents the API functions that are defined as part of the SPI Interface Driver.


Functions

Function	Description
 CyFx3BootSpiInit	Starts the SPI interface block on the device.
 CyFx3BootSpiDeInit	Stops the SPI module.
 CyFx3BootSpiSetSsnLine	Assert / Deassert the SSN Line.
 CyFx3BootSpiSetConfig	Sets and configures Spi interface parameters.
 CyFx3BootSpiSetBlockXfer	This function enables the DMA transfer.
 CyFx3BootSpiDisableBlockXfer	This function disabled the TX / RX DMA functionality.
 CyFx3BootSpiTransmitWords	Transmits data word by word over the SPI interface
 CyFx3BootSpiReceiveWords	Receives data word by word over the SPI interface.
 CyFx3BootSpiDmaXferData	This function is used to setup a dma from CPU to SPI or vice versa.

Group

[FX3 Boot SPI Interface](#)

Legend

	Method
---	--------

13.4.2.1 CyFx3BootSpiInit

Starts the SPI interface block on the device.

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.

Returns

- CY_FX3_BOOT_SUCCESS - if the SPI block was successfully initialized.
- CY_FX3_BOOT_ERROR_NOT_SUPPORTED - if the FX3 part in use does not support the SPI interface.

C++

```
CyFx3BootErrorCode\_t CyFx3BootSpiInit(  
    void  
);
```

Group

[FX3 Boot SPI Functions](#)

See Also

- [CyFx3BootSpiDeinit](#)

File

[cyfx3spi.h](#)

13.4.2.2 CyFx3BootSpiDeInit

Stops the SPI module.

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.

Returns

- None

C++

```
void CyFx3BootSpiDeInit(
    void
);
```

Group

[FX3 Boot SPI Functions](#)

See Also

- [CyFx3BootSpiInit](#)

File

cyfx3spi.h

13.4.2.3 CyFx3BootSpiSetSsnLine

Assert / Deassert the SSN Line.

Asserts / deasserts SSN Line for the default slave device. This is possible only if the SSN line control is configured for FW controlled mode.

Parameters

Parameters	Description
CyBool_t isHigh	Cyfalse: Pull down the SSN line,
CyTrue	Pull up the SSN line

Returns

- None

C++

```
void CyFx3BootSpiSetSsnLine(  
    CyBool_t isHigh  
);
```

Group

[FX3 Boot SPI Functions](#)

See Also

None

File

cyfx3spi.h

13.4.2.4 CyFx3BootSpiSetConfig

Sets and configures Spi interface parameters.

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 will reset the FIFO and hence the data in pipe will be lost.

The callback parameter is used to specify an event callback function that will be called by the driver when an SPI interrupt occurs.

Parameters

Parameters	Description
CyFx3BootSpiConfig_t * config	pointer to the SPI config structure

Returns

- 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

C++

```
CyFx3BootErrorCode_t CyFx3BootSpiSetConfig(
    CyFx3BootSpiConfig_t * config
);
```

Group

[FX3 Boot SPI Functions](#)

See Also

- [CyFx3BootSpiConfig_t](#)
- [CyFx3BootSpiCb_t](#)

File

cyfx3spi.h

13.4.2.5 CyFx3BootSpiSetBlockXfer

This function enables the DMA transfer.

This function switches SPI to DMA mode.

If the txSize parameter is non-zero then TX is enabled and if rxSize parameter is non-zero, then RX is enabled. If both are enabled, then SPI block will stall if any of the DMA pipe is stalled.

txSize == rxSize: In this case both TX and RX will function simultaneously. If the DMA pipe gets stalled due to buffer non-availability, both TX and RX will be stalled and will resume when both DMA pipes are active. So there is no loss of synchronization.

txSize > rxSize: In this case RX will receive only as many words as requested and will stop when the count reaches that point. TX will continue to receive data until its count runs down.

txSize < rxSize: In this case the TX and RX will function till all TX words are received and then will stall. The DisableBlockXfer API needs to be called explicitly for TX for the RX to resume.

A call to SetBlockXfer has to be followed by a call to DisableBlockXfer before the SPI can be used in Register mode.

Parameters

Parameters	Description
uint32_t txSize	Number of words to be transmitted (not bytes)
uint32_t rxSize	Number of words to be received (not bytes)

Returns

- None

C++

```
void CyFx3BootSpiSetBlockXfer(  
    uint32_t txSize,  
    uint32_t rxSize  
);
```

Group

[FX3 Boot SPI Functions](#)

See Also

- [CyFx3BootSpiDisableBlockXfer](#)

File

cyfx3spi.h

13.4.2.6 CyFx3BootSpiDisableBlockXfer

This function disabled the TX / RX DMA functionality.

The function can be called only when DMA has already been activated. The function will disable only the block requested for. If both TX and RX gets disabled, then it disables the DMA mode as well. This needs to be invoked to do register mode operations after a SetBlockXfer call.

Returns

- None

C++

```
void CyFx3BootSpiDisableBlockXfer(
    void
);
```

Group

[FX3 Boot SPI Functions](#)

See Also

- [CyFx3BootSpiSetBlockXfer](#)

File

cyfx3spi.h

13.4.2.7 CyFx3BootSpiTransmitWords

Transmits data word by word over the SPI interface

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 a SetBlockXfer was called, then a DisableBlockXfer needs to be called.

Parameters

Parameters	Description
uint8_t * data	Source data pointer. This needs to be padded to nearest byte if the word length is not byte aligned.
uint32_t byteCount	This needs to be a multiple of word length aligned where each word is aligned to a byte.

Returns

- 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 TransmitWords fails to transfer the data.
- CY_FX3_BOOT_ERROR_TIMEOUT - When the TransmitWords timesout.

C++

```
CyFx3BootErrorCode\_t CyFx3BootSpiTransmitWords(  
    uint8_t * data,  
    uint32_t byteCount  
);
```

Group

[FX3 Boot SPI Functions](#)

See Also

- [CyFx3BootSpiReceiveWords](#)

File

cyfx3spi.h

13.4.2.8 CyFx3BootSpiReceiveWords

Receives data word by word over the SPI interface.

This function can be called only when there is no active DMA transfer. If there was any previous DMA operation with SPI, then a DisableBlockXfer needs to be called.

Parameters

Parameters	Description
uint8_t * data	Destination buffer pointer
uint32_t byteCount	The byte size of the buffer. This needs to be a multiple of word length aligned to nearest byte size. For example, if the word length is 18 bits then each word would be placed in 3 bytes (valid data on LSBs). If the byte count is not aligned, then the call returns with an error.

Returns

- 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 ReceiveWords fails.
- CY_FX3_BOOT_ERROR_TIMEOUT - When the ReceiveWords timesout.

C++

```
CyFx3BootErrorCode_t CyFx3BootSpiReceiveWords(
    uint8_t * data,
    uint32_t byteCount
);
```

Group

[FX3 Boot SPI Functions](#)

See Also

- [CyFx3BootSpiTransmitWords](#)

File

cyfx3spi.h

13.4.2.9 CyFx3BootSpiDmaXferData

This function is used to setup a dma from CPU to SPI or vice versa.

This function is a blocking call. This function is used to read/write length number of bytes from/to SPI. This function can be used only if the SPI has been configured for DMA transfer mode. This function is used to create a one shot DMA channel to transfer to the data. Infinite transfers are not supported.

Parameters

Parameters	Description
CyBool_t isRead	isRead = CyTrue for read operations isRead = CyFalse for write operations
uint32_t address	address of the buffer from/to which data is to be transferred
uint32_t length	length of the data to be transferred. Maximum length of the data that can be transferred is as defined by the Page size of the SPI device.
uint32_t timeout	Timeout in 10s of us. Also refer the macros CY_FX3_BOOT_NO_WAIT and CY_FX3_BOOT_WAIT_FOREVER

Returns

- CY_FX3_BOOT_SUCCESS - if the data transfer is successful
- CY_FX3_BOOT_ERROR_XFER_FAILURE - if the data transfer encountered any error
- CY_FX3_BOOT_ERROR_TIMEOUT - if the data transfer times out

C++

```
CyFx3BootErrorCode\_t CyFx3BootSpiDmaXferData(  
    CyBool_t isRead,  
    uint32_t address,  
    uint32_t length,  
    uint32_t timeout  
);
```

Group

[FX3 Boot SPI Functions](#)

See Also

- [CyFx3BootSpiDisableBlockXfer](#)
- [CyFx3BootSpiSetBlockXfer](#)

File

cyfx3spi.h

13.5 FX3 Boot I2C Interface

The FX3 booter includes a I2C interface driver and provides a set of APIs that allow the configuration of the I2C interface properties and data exchange with one or more I2C slave devices.

Group

[FX3 Boot APIs](#)


Topics

Topic	Description
FX3 Boot I2C Data Types	This section documents the data types that are defined as part of the I2C driver.
FX3 Boot I2C Functions	This section documents the functions defined as part of the I2C driver.

13.5.1 FX3 Boot I2C Data Types

This section documents the data types that are defined as part of the I2C driver.



Enumerations

Enumeration	Description
 CyFx3BootI2cError_t	List of I2C specific error/status codes.



Group

[FX3 Boot I2C Interface](#)

Legend

	Enumeration
	Structure

Structures

Structure	Description
 CyFx3BootI2cConfig_t	Structure defining the configuration of the I2C interface.
 CyFx3BootI2cPreamble_t	Structure defining the preamble to be sent on the I2C interface.

13.5.1.1 CyFx3BootI2cError_t

List of I2C specific error/status codes.

This type lists the various I2C specific error/status codes.

C++

```
enum CyFx3BootI2cError_t {
    CY_FX3_BOOT_I2C_ERROR_NAK_BYTE_0 = 0,
    CY_FX3_BOOT_I2C_ERROR_NAK_BYTE_1,
    CY_FX3_BOOT_I2C_ERROR_NAK_BYTE_2,
    CY_FX3_BOOT_I2C_ERROR_NAK_BYTE_3,
    CY_FX3_BOOT_I2C_ERROR_NAK_BYTE_4,
    CY_FX3_BOOT_I2C_ERROR_NAK_BYTE_5,
    CY_FX3_BOOT_I2C_ERROR_NAK_BYTE_6,
    CY_FX3_BOOT_I2C_ERROR_NAK_BYTE_7,
    CY_FX3_BOOT_I2C_ERROR_NAK_DATA,
    CY_FX3_BOOT_I2C_ERROR_PREAMBLE_EXIT_NACK_ACK,
    CY_FX3_BOOT_I2C_ERROR_PREAMBLE_EXIT,
    CY_FX3_BOOT_I2C_ERROR_NAK_TX_UNDERFLOW,
    CY_FX3_BOOT_I2C_ERROR_NAK_TX_OVERFLOW,
    CY_FX3_BOOT_I2C_ERROR_NAK_RX_UNDERFLOW,
    CY_FX3_BOOT_I2C_ERROR_NAK_RX_OVERFLOW
};
```

Group

[FX3 Boot I2C Data Types](#)

Members

Members	Description
CY_FX3_BOOT_I2C_ERROR_NAK_BYTE_0 = 0	Slave NACK-ed the zeroth byte of the preamble.
CY_FX3_BOOT_I2C_ERROR_NAK_BYTE_1	Slave NACK-ed the first byte of the preamble.
CY_FX3_BOOT_I2C_ERROR_NAK_BYTE_2	Slave NACK-ed the second byte of the preamble.
CY_FX3_BOOT_I2C_ERROR_NAK_BYTE_3	Slave NACK-ed the third byte of the preamble.
CY_FX3_BOOT_I2C_ERROR_NAK_BYTE_4	Slave NACK-ed the fourth byte of the preamble.
CY_FX3_BOOT_I2C_ERROR_NAK_BYTE_5	Slave NACK-ed the fifth byte of the preamble.
CY_FX3_BOOT_I2C_ERROR_NAK_BYTE_6	Slave NACK-ed the sixth byte of the preamble.
CY_FX3_BOOT_I2C_ERROR_NAK_BYTE_7	Slave NACK-ed the seventh byte of the preamble.
CY_FX3_BOOT_I2C_ERROR_NAK_DATA	Slave sent a NACK during the data phase of a transfer.
CY_FX3_BOOT_I2C_ERROR_PREAMBLE_EXIT_NACK_ACK	Poll operation has exited due to the slave returning an ACK or a NACK handshake.
CY_FX3_BOOT_I2C_ERROR_PREAMBLE_EXIT	Poll operation with address repetition timed out.
CY_FX3_BOOT_I2C_ERROR_NAK_TX_UNDERFLOW	Underflow in buffer during transmit/write operation.
CY_FX3_BOOT_I2C_ERROR_NAK_TX_OVERFLOW	Overflow of buffer during transmit operation.
CY_FX3_BOOT_I2C_ERROR_NAK_RX_UNDERFLOW	Underflow in buffer during receive/read operation.
CY_FX3_BOOT_I2C_ERROR_NAK_RX_OVERFLOW	Overflow of buffer during receive operation.

File

cyfx3i2c.h

13.5.1.2 CyFx3BootI2cConfig_t

Structure defining the configuration of the I2C interface.

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.

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 [CyFx3BootI2cSetConfig](#) API expects that no callback is specified when the register mode is selected.

C++

```
struct CyFx3BootI2cConfig_t {  
    uint32_t bitRate;  
    CyBool_t isDma;  
    uint32_t busTimeout;  
    uint16_t dmaTimeout;  
};
```

Group

[FX3 Boot I2C Data Types](#)

See Also

- [CyFx3BootI2cSetConfig](#)

Members

Members	Description
uint32_t bitRate;	Bit rate for the interface. (Eg: 100000 for 100KHz
CyBool_t isDma;	CyFalse: Register transfer mode, CyTrue: DMA transfer mode
uint32_t busTimeout;	Number of core clocks SCK can be held low by the slave byte transmission before triggering a timeout error. 0xFFFFFFFFU means no timeout.
uint16_t dmaTimeout;	Number of core clocks DMA can remain not ready before flagging an error. 0xFFFF means no timeout.

File

cyfx3i2c.h

13.5.1.3 CyFx3BootI2cPreamble_t

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

All I2C data transfer requires a preamble, which contains the slave address and the direction of the transfer. Here we are extending this to include the command that will typically be sent to the I2C device which will trigger a data transfer.

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

For example if you look at an I2C EEPROM, it requires the address to read / write from. This is considered as the command and the data which is being read / written is considered as the actual data transfer. So the two I2C operations are combined into one I2C API call using the parameters of the structure.

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.

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 is 0x0004 as a start bit is required after the third byte (third bit is set).

C++

```
struct CyFx3BootI2cPreamble_t {
    uint8_t buffer[8];
    uint8_t length;
    uint16_t ctrlMask;
};
```

Group

[FX3 Boot I2C Data Types](#)

See Also

- [CyFx3BootI2cSetConfig](#)

Members

Members	Description
uint8_t buffer[8];	The extended preamble information.
uint8_t length;	The length of the preamble to be sent. Should be between 1 and 8.

uint16_t ctrlMask;	This field controls the start stop condition after every byte of preamble data. Bits 0 - 7 is a bit mask for start condition and Bits 8 - 15 is a bit mask for stop condition. If both are set, then stop takes priority.
--------------------	---









File

cyfx3i2c.h

13.5.2 FX3 Boot I2C Functions

This section documents the functions defined as part of the I2C driver.

Functions

Function	Description
 CyFx3BootI2cInit	Starts the I2C interface block.
 CyFx3BootI2cDeInit	Stops the I2C module.
 CyFx3BootI2cSetConfig	Sets the I2C interface parameters.
 CyFx3BootI2cSendCommand	Perform a read or write operation to the I2C slave.
 CyFx3BootI2cTransmitBytes	Writes a small number of bytes to an I2C slave.
 CyFx3BootI2cReceiveBytes	Reads a small number of bytes one by one from an I2C slave.
 CyFx3BootI2cWaitForAck	Poll an I2C slave until all of the preamble is ACKed.
 CyFx3BootI2cDmaXferData	This function is used to setup a dma from CPU to I2C or vice versa.

Group

[FX3 Boot I2C Interface](#)

Legend

	Method
---	--------

13.5.2.1 CyFx3BootI2cInit

Starts the I2C interface block.

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 rate of 100KHz.

Returns

- CY_FX3_BOOT_SUCCESS - When the SetConfig is successful
- CY_FX3_BOOT_ERROR_NOT_SUPPORTED - If the FX3 part in use does not support the I2C feature.

C++

```
CyFx3BootErrorCode_t CyFx3BootI2cInit(
    void
);
```

Group

[FX3 Boot I2C Functions](#)

See Also

- CyFx3BootI2cDeinit

File

cyfx3i2c.h

13.5.2.2 CyFx3BootI2cDeInit

Stops the I2C module.

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.

Returns

- None

C++

```
void CyFx3BootI2cDeInit(  
    void  
);
```

Group

[FX3 Boot I2C Functions](#)

See Also

- [CyFx3BootI2cInit](#)

File

cyfx3i2c.h

13.5.2.3 CyFx3BootI2cSetConfig

Sets the I2C interface parameters.

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 [CyFx3BootI2cInit](#). But this will reset the FIFO and hence the data in pipe will be lost. In register mode, all APIs are blocking in nature.

Parameters

Parameters	Description
CyFx3BootI2cConfig_t * config	I2C configuration settings

Returns

- CY_FX3_BOOT_SUCCESS - When the SetConfig is successful
- CY_FX3_BOOT_ERROR_NOT_STARTED - If the I2C block has not been initialized
- CY_FX3_BOOT_ERROR_NULL_POINTER - When the config parameter is NULL
- CY_FX3_BOOT_ERROR_TIMEOUT - When there is timeout happening during configuration

C++

```
CyFx3BootErrorCode\_t CyFx3BootI2cSetConfig(  
    CyFx3BootI2cConfig\_t * config  
);
```

Group

[FX3 Boot I2C Functions](#)

See Also

- [CyFx3BootI2cConfig_t](#)

File

cyfx3i2c.h

13.5.2.4 CyFx3BootI2cSendCommand

Perform a read or write operation to the I2C slave.

This function is used to send the extended preamble over the I2C bus. This is used in conjunction with data transfer phase in DMA/Register mode.

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. Data transfer can be done either in the DMA mode or the register mode.

The CyFx3BootI2cWaitForCompletion() can be used to detect the end of a DMA data transfer that is requested through this function.

Parameters

Parameters	Description
CyFx3BootI2cPreamble_t * preamble	Preamble information to be sent out before the data transfer.
uint32_t byteCount	Size of the transfer in bytes.
CyBool_t isRead	Direction of transfer;
CyTrue	Read, CyFalse: Write.

Returns

- 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

C++

```
CyFx3BootErrorCode_t CyFx3BootI2cSendCommand(  
    CyFx3BootI2cPreamble_t * preamble,  
    uint32_t byteCount,  
    CyBool_t isRead  
);
```

Group

[FX3 Boot I2C Functions](#)

See Also

- [CyFx3BootI2cReceiveBytes](#)
- [CyFx3BootI2cTransmitBytes](#)

File

cyfx3i2c.h

13.5.2.5 CyFx3BootI2cTransmitBytes

Writes a small number of bytes to an I2C slave.

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 on ERROR_TIMEOUT without any error recovery. The retry is done continuously without any delay. If any delay is required, then it should be added in the application firmware.

This function internally calls [CyFx3BootI2cSendCommand](#). So user must not call [CyFx3BootI2cSendCommand](#) before calling this function.

Parameters

Parameters	Description
CyFx3BootI2cPreamble_t * preamble	Preamble information to be sent out before the data transfer.
uint8_t * data	Pointer to buffer containing data to be written.
uint32_t byteCount	Size of the transfer in bytes.
uint32_t retryCount	Number of times to retry request if a byte is NAKed by the slave.

Returns

- CY_FX3_BOOT_SUCCESS - When the TransmitBytes is successful or byteCount is 0
- CY_FX3_BOOT_ERROR_NULL_POINTER - When the preamble or data are NULL
- CY_FX3_BOOT_ERROR_XFER_FAILURE - When a transfer fails
- CY_FX3_BOOT_ERROR_TIMEOUT - When the transfer times out

C++

```
CyFx3BootErrorCode_t CyFx3BootI2cTransmitBytes(
    CyFx3BootI2cPreamble_t * preamble,
    uint8_t * data,
    uint32_t byteCount,
    uint32_t retryCount
);
```

Group

[FX3 Boot I2C Functions](#)

See Also

- [CyFx3BootI2cSetConfig](#)
- [CyFx3BootI2cReceiveBytes](#)

File

cyfx3i2c.h

13.5.2.6 CyFx3BootI2cReceiveBytes

Reads a small number of bytes one by one from an I2C slave.

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. The function call can be repeated on ERROR_TIMEOUT without any error recovery. The retry is done continuously without any delay. If any delay is required, then it should be added in the application firmware.

This function internally calls [CyFx3BootI2cSendCommand](#). So user must not call [CyFx3BootI2cSendCommand](#) before calling this function.

Parameters

Parameters	Description
CyFx3BootI2cPreamble_t * preamble	Preamble information to be sent out before the data transfer.
uint8_t * data	Pointer to buffer where the data is to be placed.
uint32_t byteCount	Size of the transfer in bytes.
uint32_t retryCount	Number of times to retry request if preamble is NAKed or an error is encountered.

Returns

- CY_FX3_BOOT_SUCCESS - When the ReceiveBytes is successful or byteCount is 0
- CY_FX3_BOOT_ERROR_NULL_POINTER - When the preamble or data are NULL
- CY_FX3_BOOT_ERROR_XFER_FAILURE - When a transfer fails
- CY_FX3_BOOT_ERROR_TIMEOUT - When the transfer times out

C++

```
CyFx3BootErrorCode_t CyFx3BootI2cReceiveBytes(  
    CyFx3BootI2cPreamble_t * preamble,  
    uint8_t * data,  
    uint32_t byteCount,  
    uint32_t retryCount  
);
```

Group

[FX3 Boot I2C Functions](#)

See Also

- [CyFx3BootI2cSetConfig](#)
- [CyFx3BootI2cTransmitBytes](#)

File

cyfx3i2c.h

13.5.2.7 CyFx3BootI2cWaitForAck

Poll an I2C slave until all of the preamble is ACKed.

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 retry is done continuously without any delay. If any delay is required, then it should be added in the application firmware. The function call can be repeated on ERROR_TIMEOUT without any error recovery.

Parameters

Parameters	Description
CyFx3BootI2cPreamble_t * preamble	Preamble information to be sent out before the data transfer.
uint32_t retryCount	Number of times to retry request if preamble is NAKed or an error is encountered.

Returns

- CY_FX3_BOOT_SUCCESS - When the device returns the desired handshake
- CY_FX3_BOOT_ERROR_NULL_POINTER - When the preamble is NULL.
- CY_FX3_BOOT_ERROR_TIMEOUT - When a timeout occurs.

C++

```
CyFx3BootErrorCode_t CyFx3BootI2cWaitForAck(
    CyFx3BootI2cPreamble_t * preamble,
    uint32_t retryCount
);
```

Group

[FX3 Boot I2C Functions](#)

File

cyfx3i2c.h

13.5.2.8 CyFx3BootI2cDmaXferData

This function is used to setup a dma from CPU to I2C or vice versa.

This function is a blocking call. This function is used to read/write length number of bytes from/to I2C. This function can be used only if the I2C has been configured for DMA transfer mode. This function is used to create a one shot DMA channel to transfer to the data. Infinite transfers are not supported.

Parameters

Parameters	Description
CyBool_t isRead	isRead = CyTrue for read operations isRead = CyFalse for write operations
uint32_t address	address of the buffer from/to which data is to be transferred
uint32_t length	length of the data to be transferred Maximum length of the data that can be transferred is as defined by the Page size of the I2C device.
uint32_t timeout	Timeout in 10s of us. Also refer the macros CY_FX3_BOOT_NO_WAIT and CY_FX3_BOOT_WAIT_FOREVER

Returns

- CY_FX3_BOOT_SUCCESS - if the data transfer is successful
- CY_FX3_BOOT_ERROR_XFER_FAILURE - if the data transfer encountered any error
- CY_FX3_BOOT_ERROR_TIMEOUT - if the data transfer times out

C++

```
CyFx3BootErrorCode_t CyFx3BootI2cDmaXferData(  
    CyBool_t isRead,  
    uint32_t address,  
    uint32_t length,  
    uint32_t timeout  
);
```

Group

[FX3 Boot I2C Functions](#)

See Also

- [CyFx3BootSpiDisableBlockXfer](#)
- [CyFx3BootSpiSetBlockXfer](#)

File

cyfx3i2c.h

13.6 FX3 Boot GPIO Interface

The GPIO interface module is responsible for handling general purpose IO pins. This section defines the data structures and software interfaces for GPIO interface management.

GPIO(general purpose I/O) pins are a special (simple) case of low performance serial peripherals that do not need DMA capability.

Simple GPIO provides software controlled and observable input and output capability only.

Group

[FX3 Boot APIs](#)


Topics

Topic	Description
FX3 Boot GPIO Data Types	This section documents the enumerations and data types that are defined as part of the GPIO Interface Manager.
FX3 Boot GPIO Functions	This section documents the API functions that are defined as part of the GPIO Interface Manager.

13.6.1 FX3 Boot GPIO Data Types

This section documents the enumerations and data types that are defined as part of the GPIO Interface Manager.



Enumerations

Enumeration	Description
 CyFx3BootGpioIntrMode_t	Enumerated list for interrupt mode GPIOs.


Group

[FX3 Boot GPIO Interface](#)

Legend

	Enumeration
	Structure

Structures

Structure	Description
 CyFx3BootGpioSimpleConfig_t	Structure contains configuration information for simple GPIOs.

13.6.1.1 CyFx3BootGpioIntrMode_t

Enumerated list for interrupt mode GPIOs.

Enumerations to control the triggering of interrupts for GPIOs configured as interrupts.

C++

```
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
};
```

Group

[FX3 Boot GPIO Data Types](#)

Notes

Interrupts are not supported in this release of booter.

Members

Members	Description
CY_FX3_BOOT_GPIO_NO_INTR = 0	No Interrupt is triggered
CY_FX3_BOOT_GPIO_INTR_POS_EDGE	Interrupt is triggered for positive edge of input.
CY_FX3_BOOT_GPIO_INTR_NEG_EDGE	Interrupt is triggered for negative edge of input.
CY_FX3_BOOT_GPIO_INTR_BOTH_EDGE	Interrupt is triggered for both edge of input.
CY_FX3_BOOT_GPIO_INTR_LOW_LEVEL	Interrupt is triggered for Low level of input.
CY_FX3_BOOT_GPIO_INTR_HIGH_LEVEL	Interrupt is triggered for High level of input.

File

cyfx3gpio.h

13.6.1.2 CyFx3BootGpioSimpleConfig_t

Structure contains configuration information for simple GPIOs.

The GPIO can be configured to behave in different ways. This means that not all configurations are active all the time.

If the pin is configured as input, then the fields driveLowEn and driveHighEn should be CyFalse. Also the field outValue is not considered.

For normal mode of output operation, both driveLowEn and driveHighEn needs to be CyTrue. Also the inputEn should be CyFalse.

C++

```
struct CyFx3BootGpioSimpleConfig_t {
    CyBool_t outValue;
    CyBool_t driveLowEn;
    CyBool_t driveHighEn;
    CyBool_t inputEn;
    CyFx3BootGpioIntrMode\_t intrMode;
};
```

Group

[FX3 Boot GPIO Data Types](#)

See Also

- [CyFx3BootGpioIntrMode_t](#)

Members

Members	Description
CyBool_t outValue;	Initial output on the GPIO if configured as
CyBool_t driveLowEn;	When set true, the output driver is enabled for outValue = CyFalse (0), otherwise tristated.
CyBool_t driveHighEn;	When set true, the output driver is enabled for outValue = CyTrue (1), otherwise tristated.
CyBool_t inputEn;	When set true, the input state is enabled.
CyFx3BootGpioIntrMode_t intrMode;	Interrupt mode for the GPIO.







File

cyfx3gpio.h

13.6.2 FX3 Boot GPIO Functions

This section documents the API functions that are defined as part of the GPIO Interface Manager.

Functions

Function	Description
 CyFx3BootGpioInit	Initializes the GPIO Interface Manager.
 CyFx3BootGpioDeInit	De-initializes the GPIO Interface Manager.
 CyFx3BootGpioSetSimpleConfig	Configures a simple GPIO.
 CyFx3BootGpioDisable	Disables a GPIO pin.
 CyFx3BootGpioGetValue	Query the state of GPIO input.
 CyFx3BootGpioSetValue	Set the state of GPIO pin output.

Group

[FX3 Boot GPIO Interface](#)

Legend

	Method
---	--------

13.6.2.1 CyFx3BootGpioInit

Initializes the GPIO Interface Manager.

This function is used to initialize the GPIO Interface Manager module. This function resets the GPIO hardware block and initializes all the registers.

Returns

- None

C++

```
void CyFx3BootGpioInit(  
    void  
);
```

Group

[FX3 Boot GPIO Functions](#)

See Also

- [CyFx3BootGpioIntrCb_t](#)
- [CyFx3BootGpioDeInit](#)

File

cyfx3gpio.h

13.6.2.2 CyFx3BootGpioDeInit

De-initializes the GPIO Interface Manager.

This function resets the GPIO hardware block and de-initializes it.

Returns

- None

C++

```
void CyFx3BootGpioDeInit(  
    void  
);
```

Group

[FX3 Boot GPIO Functions](#)

See Also

- [CyFx3BootGpioInit](#)

File

cyfx3gpio.h

13.6.2.3 CyFx3BootGpioSetSimpleConfig

Configures a simple GPIO.

This function is used to configure and enable a simple GPIO pin. This function needs to be called before using the simple GPIO.

Parameters

Parameters	Description
uint8_t gpioId	GPIO id to be modified.
CyFx3BootGpioSimpleConfig_t * cfg_p	Pointer to the configure information.

Returns

- 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 GPIO pin has not been previously enabled in the IO matrix configuration

C++

```
CyFx3BootErrorCode_t CyFx3BootGpioSetSimpleConfig(
    uint8_t gpioId,
    CyFx3BootGpioSimpleConfig_t * cfg_p
);
```

Group

[FX3 Boot GPIO Functions](#)

See Also

- [CyFx3BootGpioSimpleConfig_t](#)

File

cyfx3gpio.h

13.6.2.4 CyFx3BootGpioDisable

Disables a GPIO pin.

This function is used to disable a gpio pin. A GPIO is enabled when a Config call is made.

Parameters

Parameters	Description
uint8_t gpioId	GPIO ID to be disabled

Returns

- None

C++

```
void CyFx3BootGpioDisable(  
    uint8_t gpioId  
);
```

Group

[FX3 Boot GPIO Functions](#)

See Also

- [CyFx3BootGpioGetValue](#)
- [CyFx3BootGpioSetValue](#)

File

cyfx3gpio.h

13.6.2.5 CyFx3BootGpioGetValue

Query the state of GPIO input.

For input pins this reads the latest status on the GPIO, while for output pins this returns the last updated outValue.

Parameters

Parameters	Description
uint8_t gpioId	GPIO id to be queried.
CyBool_t * value_p	Output parameter that will be filled with the GPIO value.

Returns

- 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 GPIO pin has not been previously enabled in the IO matrix configuration

C++

```
CyFx3BootErrorCode_t CyFx3BootGpioGetValue(
    uint8_t gpioId,
    CyBool_t * value_p
);
```

Group

[FX3 Boot GPIO Functions](#)

See Also

- [CyFx3BootGpioSetValue](#)

File

cyfx3gpio.h

13.6.2.6 CyFx3BootGpioSetValue

Set the state of GPIO pin output.

The function is valid only for output GPIOs. The function states what value needs to be pushed to the pin. A CyFalse means 0 and a CyTrue means 1. This is not failed for a pin configured as an input, but will not change anything on the hardware and the pin remains configured as input.

Parameters

Parameters	Description
uint8_t gpioId	GPIO id to be modified.
CyBool_t value	Value to set on the GPIO pin.

Returns

- CY_FX3_BOOT_SUCCESS - If the operation is successful
- CY_FX3_BOOT_ERROR_NOT_CONFIGURED - If the GPIO pin has not been previously enabled in the IO matrix configuration

C++

```
CyFx3BootErrorCode\_t CyFx3BootGpioSetValue(  
    uint8_t gpioId,  
    CyBool_t value  
);
```

Group

[FX3 Boot GPIO Functions](#)

See Also

- [CyFx3BootGpioGetValue](#)

File

cyfx3gpio.h

14 Symbol Reference

14.1 Functions

The following table lists functions in this documentation.

14.2 Macros

The following table lists macros in this documentation.

14.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

14.4 Types

The following table lists types in this documentation.

Index

B

Buffer Functions 175

C

Cache and Memory Management 31

Cache Manager Functions 31

Channel Functions 202

CY_FX3_USB_MAX_STRING_DESC_INDEX 284

CY_U3P_DMA_BUFFER_AREA_BASE 144

CY_U3P_DMA_BUFFER_AREA_LIMIT 145

CY_U3P_DMA_BUFFER_EOP 145

CY_U3P_DMA_BUFFER_ERROR 146

CY_U3P_DMA_BUFFER_MARKER 145

CY_U3P_DMA_BUFFER_OCCUPIED 146

CY_U3P_DMA_BUFFER_STATUS_MASK 146

CY_U3P_DMA_BUFFER_STATUS_WRITE_MASK 147

CY_U3P_DMA_CPU_NUM_SCK 141

CY_U3P_DMA_DSCR_COUNT 135

CY_U3P_DMA_DSCR_SIZE 135

CY_U3P_DMA_DSCR0_LOCATION 135

CY_U3P_DMA_LPP_MAX_CONS_SCK 136

CY_U3P_DMA_LPP_MAX_PROD_SCK 137

CY_U3P_DMA_LPP_MIN_CONS_SCK 136

CY_U3P_DMA_LPP_MIN_PROD_SCK 136

CY_U3P_DMA_LPP_NUM_SCK 137

CY_U3P_DMA_MAX_AVAIL_COUNT 147

CY_U3P_DMA_MAX_BUFFER_SIZE 144

CY_U3P_DMA_MAX_MULTI_SCK_COUNT 147

CY_U3P_DMA_MIN_MULTI_SCK_COUNT 148

CY_U3P_DMA_PIB_MAX_CONS_SCK 138

CY_U3P_DMA_PIB_MAX_PROD_SCK 139

CY_U3P_DMA_PIB_MIN_CONS_SCK 137

CY_U3P_DMA_PIB_MIN_PROD_SCK 138

CY_U3P_DMA_PIB_NUM_SCK 139

CY_U3P_DMA_SCK_ID_MASK 143

CY_U3P_DMA_SCK_MASK 142

CY_U3P_DMA_UIB_MAX_CONS_SCK 140

CY_U3P_DMA_UIB_MIN_CONS_SCK 139

CY_U3P_DMA_UIB_NUM_SCK 140

CY_U3P_DMA_UIBIN_MAX_PROD_SCK 141

CY_U3P_DMA_UIBIN_MIN_PROD_SCK 140

CY_U3P_DMA_UIBIN_NUM_SCK 141

CY_U3P_DWORD_GET_BYTE0 695

CY_U3P_DWORD_GET_BYTE1 696

CY_U3P_DWORD_GET_BYTE2 696

CY_U3P_DWORD_GET_BYTE3 696

CY_U3P_GET_LSB 694

CY_U3P_GET_MSB 694

CY_U3P_IP_BLOCK_MASK 142

CY_U3P_IP_BLOCK_POS 142

CY_U3P_MAKEDWORD 697

CY_U3P_MAKEWORD 695

CY_U3P_MAX 693

CY_U3P_MAX_STRING_DESC_INDEX 342

CY_U3P_MIN 694

CY_U3P_OTG_SRP_MAX_REPEAT_INTERVAL 295

CY_U3P_USB_CLASS_RQT 281

CY_U3P_USB_GS_DEVICE 281

CY_U3P_USB_GS_ENDPOINT 282

CY_U3P_USB_GS_INTERFACE 282

CY_U3P_USB_HOST_EPS_ACTIVE 418

CY_U3P_USB_HOST_EPS_BABBLE 421

CY_U3P_USB_HOST_EPS_BYTE_COUNT_MASK 430

CY_U3P_USB_HOST_EPS_BYTE_COUNT_POS 429

CY_U3P_USB_HOST_EPS_EPDIR 417

CY_U3P_USB_HOST_EPS_EPNUM_MASK 416

CY_U3P_USB_HOST_EPS_EPNUM_POS 415

CY_U3P_USB_HOST_EPS_HALT 419

CY_U3P_USB_HOST_EPS_IOC_INT 428

CY_U3P_USB_HOST_EPS_ISO_ORUN_ERROR 427

CY_U3P_USB_HOST_EPS_OVER_UNDER_RUN 420

CY_U3P_USB_HOST_EPS_PHY_ERROR 424

CY_U3P_USB_HOST_EPS_PID_ERROR 425

CY_U3P_USB_HOST_EPS_PING 423

CY_U3P_USB_HOST_EPS_TIMEOUT_ERROR 426

CY_U3P_USB_HOST_EPS_XACT_ERROR 422

CY_U3P_USB_HOST_PORT_STAT_ACTIVE 433

CY_U3P_USB_HOST_PORT_STAT_CONNECTED 431

CY_U3P_USB_HOST_PORT_STAT_ENABLED 432

CY_U3P_USB_HOST_PORT_STAT_SUSPENDED 434

CY_U3P_USB_INDEX_MASK 332

CY_U3P_USB_INDEX_POS 333

CY_U3P_USB_LENGTH_MASK 334

CY_U3P_USB_LENGTH_POS 335

CY_U3P_USB_OTG_STATUS_SELECTOR 291

CY_U3P_USB_REQUEST_MASK 336	CyFx3BootSpiDmaXferData 774
CY_U3P_USB_REQUEST_POS 337	CyFx3BootSpiInit 766
CY_U3P_USB_REQUEST_TYPE_MASK 338	CyFx3BootSpiReceiveWords 773
CY_U3P_USB_REQUEST_TYPE_POS 339	CyFx3BootSpiSetBlockXfer 770
CY_U3P_USB_RESERVED_RQT 291	CyFx3BootSpiSetConfig 769
CY_U3P_USB_STANDARD_RQT 280	CyFx3BootSpiSetSsnLine 768
CY_U3P_USB_TARGET_DEVICE 282	CyFx3BootSpiSsnCtrl_t 762
CY_U3P_USB_TARGET_ENDPT 283	CyFx3BootSpiSsnLagLead_t 761
CY_U3P_USB_TARGET_INTF 283	CyFx3BootSpiTransmitWords 772
CY_U3P_USB_TARGET_MASK 290	CyFx3BootSysClockSrc_t 705
CY_U3P_USB_TARGET_OTHER 283	CyFx3BootUartBaudrate_t 747
CY_U3P_USB_TYPE_MASK 290	CyFx3BootUartConfig_t 750
CY_U3P_USB_VALUE_MASK 340	CyFx3BootUartDelInit 753
CY_U3P_USB_VALUE_POS 341	CyFx3BootUartDmaXferData 759
CY_U3P_USB_VENDOR_RQT 281	CyFx3BootUartInit 752
CY_U3P_USB3_TP_DEVADDR_POS 290	CyFx3BootUartParity_t 748
CY_U3P_USB3_TP_EPNUM_POS 291	CyFx3BootUartReceiveBytes 758
CyFx3BootDeviceConfigureIOMatrix 711	CyFx3BootUartRxSetBlockXfer 756
CyFx3BootDeviceInit 710	CyFx3BootUartSetConfig 754
CyFx3BootErrorCode_t 707	CyFx3BootUartStopBit_t 749
CyFx3BootGetPartNumber 715	CyFx3BootUartTransmitBytes 757
CyFx3BootGpioDelInit 794	CyFx3BootUartTxSetBlockXfer 755
CyFx3BootGpioDisable 796	CyFx3BootUsbAckSetup 740
CyFx3BootGpioGetValue 797	CyFx3BootUsbCheckUsb3Disconnect 741
CyFx3BootGpioInit 793	CyFx3BootUsbConnect 735
CyFx3BootGpioIntrMode_t 790	CyFx3BootUsbDmaXferData 736
CyFx3BootGpioSetSimpleConfig 795	CyFx3BootUsbEp0Pkt_t 725
CyFx3BootGpioSetValue 798	CyFx3BootUsbEp0StatusCheck 742
CyFx3BootGpioSimpleConfig_t 791	CyFx3BootUsbEpConfig_t 726
CyFx3BootI2cConfig_t 778	CyFx3BootUsbEpType_t 720
CyFx3BootI2cDelInit 782	CyFx3BootUSBEventCb_t 721
CyFx3BootI2cDmaXferData 788	CyFx3BootUsbEventType_t 719
CyFx3BootI2cError_t 776	CyFx3BootUsbGetDesc 730
CyFx3BootI2cInit 781	CyFx3BootUsbGetEpCfg 733
CyFx3BootI2cPreamble_t 779	CyFx3BootUsbGetSpeed 737
CyFx3BootI2cReceiveBytes 786	CyFx3BootUsbLPMDisable 744
CyFx3BootI2cSendCommand 784	CyFx3BootUsbLPMEnable 745
CyFx3BootI2cSetConfig 783	CyFx3BootUsbSendCompliancePatterns 743
CyFx3BootI2cTransmitBytes 785	CyFx3BootUsbSetClrFeature 738
CyFx3BootI2cWaitForAck 787	CyFx3BootUsbSetDesc 729
CyFx3BootIoMatrixConfig_t 708	CyFx3BootUsbSetEpConfig 732
CyFx3BootJumpToProgramEntry 714	CyFx3BootUSBSetupCb_t 722
CyFx3BootRegisterSetupCallback 731	CyFx3BootUsbSpeed_t 718
CyFx3BootRetainGpioState 716	CyFx3BootUsbStall 739
CyFx3BootSpiConfig_t 764	CyFx3BootUsbStart 728
CyFx3BootSpiDelInit 767	CyFx3BootUsbVBattEnable 734
CyFx3BootSpiDisableBlockXfer 771	CyFx3BootWatchdogClear 713

CyFx3BootWatchdogConfigure 712	CyU3PDeviceInit 10
CyFx3PartNumber_t 706	CyU3PDeviceReset 15
CyFxApplicationDefine 15	CyU3PDmaBuffer_t 170
CYU3P_GET_GPIF_ERROR_TYPE 484	CyU3PDmaBufferAlloc 178
CYU3P_GET_PIB_ERROR_TYPE 485	CyU3PDmaBufferDeInit 177
CYU3P_GPIF_INVALID_STATE 494	CyU3PDmaBufferFree 179
CYU3P_GPIF_NUM_STATES 494	CyU3PDmaBufferInit 176
CYU3P_GPIF_NUM_TRANS_FNS 494	CyU3PDmaCallback_t 159
CYU3P_PIB_MAX_BURST_SETTING 495	CyU3PDmaCbInput_t 157
CYU3P_PIB_SOCKET_COUNT 495	CyU3PDmaCbType_t 156
CYU3P_PIB_THREAD_COUNT 495	CyU3PDmaChannel 161
CyU3PAbsortHandler 25	CyU3PDmaChannelAbort 239
CyU3PApplicationDefine 65	CyU3PDmaChannelCacheControl 214
CyU3PAssert 697	CyU3PDmaChannelCommitBuffer 219
CyU3PBlockAlloc 75	CyU3PDmaChannelConfig_t 164
CyU3PBlockFree 76	CyU3PDmaChannelCreate 204
CyU3PBlockId_t 160	CyU3PDmaChannelDestroy 206
CyU3PBlockPool 51	CyU3PDmaChannelDiscardBuffer 221
CyU3PBlockPoolCreate 73	CyU3PDmaChannelGetBuffer 217
CyU3PBlockPoolDestroy 74	CyU3PDmaChannelGetHandle 212
CyU3PBusyWait 698	CyU3PDmaChannelGetStatus 210
CyU3PByteAlloc 71	CyU3PDmaChannelReset 237
CyU3PByteFree 72	CyU3PDmaChannelResume 235
CyU3PBytePool 51	CyU3PDmaChannelSetSuspend 233
CyU3PBytePoolCreate 69	CyU3PDmaChannelSetupRecvBuffer 225
CyU3PBytePoolDestroy 70	CyU3PDmaChannelSetupSendBuffer 223
CyU3PComputeChecksum 699	CyU3PDmaChannelSetWrapUp 231
CyU3PConnectState 367	CyU3PDmaChannelSetXfer 215
CyU3PDebugDeInit 679	CyU3PDmaChannelUpdateMode 208
CyU3PDebugDisable 686	CyU3PDmaChannelWaitForCompletion 229
CyU3PDebugEnable 685	CyU3PDmaChannelWaitForRecvBuffer 227
CyU3PDebugInit 678	CyU3PDmaDescriptor_t 171
CyU3PDebugLog 683	CyU3PDmaDscrChainCreate 188
CyU3PDebugLog_t 677	CyU3PDmaDscrChainDestroy 189
CyU3PDebugLogClear 685	CyU3PDmaDscrGet 182
CyU3PDebugLogFlush 684	CyU3PDmaDscrGetConfig 185
CyU3PDebugPreamble 682	CyU3PDmaDscrGetFreeCount 181
CyU3PDebugPrint 681	CyU3PDmaDscrListCreate 186
CyU3PDebugSetTraceLevel 683	CyU3PDmaDscrListDestroy 187
CyU3PDebugSysMemDeInit 680	CyU3PDmaDscrPut 183
CyU3PDebugSysMemInit 679	CyU3PDmaDscrSetConfig 184
CyU3PDeviceCacheControl 11	CyU3PDmaEnableMulticast 242
CyU3PDeviceConfigureIOMatrix 12	CyU3PDmaGetIpNum 143
CyU3PDeviceGetPartNumber 22	CyU3PDmaGetSckId 144
CyU3PDeviceGetSysClkFreq 14	CyU3PDmaGetSckNum 143
CyU3PDeviceGpioOverride 648	CyU3PDmaMode_t 155
CyU3PDeviceGpioRestore 649	CyU3PDmaMultiCallback_t 159

CyU3PDmaMultiChannel	165
CyU3PDmaMultiChannelAbort	277
CyU3PDmaMultiChannelCacheControl	252
CyU3PDmaMultiChannelCommitBuffer	257
CyU3PDmaMultiChannelConfig_t	168
CyU3PDmaMultiChannelCreate	243
CyU3PDmaMultiChannelDestroy	245
CyU3PDmaMultiChannelDiscardBuffer	259
CyU3PDmaMultiChannelGetBuffer	255
CyU3PDmaMultiChannelGetHandle	249
CyU3PDmaMultiChannelGetStatus	250
CyU3PDmaMultiChannelReset	275
CyU3PDmaMultiChannelResume	273
CyU3PDmaMultiChannelSetSuspend	271
CyU3PDmaMultiChannelSetupRecvBuffer	265
CyU3PDmaMultiChannelSetupSendBuffer	261
CyU3PDmaMultiChannelSetWrapUp	269
CyU3PDmaMultiChannelSetXfer	253
CyU3PDmaMultiChannelUpdateMode	247
CyU3PDmaMultiChannelWaitForCompletion	267
CyU3PDmaMultiChannelWaitForRecvBuffer	263
CyU3PDmaMultiType_t	152
CyU3PDmaSckSusptType_t	157
CyU3PDmaSocket_t	173
CyU3PDmaSocketCallback_t	160
CyU3PDmaSocketConfig_t	172
CyU3PDmaSocketDisable	201
CyU3PDmaSocketEnable	200
CyU3PDmaSocketGetConfig	195
CyU3PDmaSocketId_t	148
CyU3PDmaSocketIsValid	191
CyU3PDmaSocketIsValidConsumer	193
CyU3PDmaSocketIsValidProducer	192
CyU3PDmaSocketRegisterCallback	202
CyU3PDmaSocketSendEvent	196
CyU3PDmaSocketSetConfig	194
CyU3PDmaSocketSetWrapUp	197
CyU3PDmaState_t	153
CyU3PDmaType_t	152
CyU3PDmaUpdateSocketResume	199
CyU3PDmaUpdateSocketSuspendOption	198
CyU3PDriveStrengthState_t	4
CyU3PEpConfig_t	324
CyU3PEErrorCode_t	689
CyU3PEvent	52
CyU3PEventCreate	116
CyU3PEventDestroy	117
CyU3PEventGet	118
CyU3PEventSet	119
CyU3PFiqContextRestore	61
CyU3PFiqContextSave	60
CyU3PFirmwareEntry	9
CyU3PFreeHeaps	78
CyU3PGetConnectState	368
CyU3PGetTime	127
CyU3PGpifComparatorType	496
CyU3PGpifConfig_t	499
CyU3PGpifConfigure	502
CyU3PGpifControlSWInput	513
CyU3PGpifDisable	510
CyU3PGpifErrorType	479
CyU3PGpifEventCb_t	496
CyU3PGpifEventType	497
CyU3PGpifGetSMState	508
CyU3PGpifInitAddrCounter	513
CyU3PGpifInitComparator	515
CyU3PGpifInitCtrlCounter	515
CyU3PGpifInitDataCounter	514
CyU3PGpifInitTransFunctions	503
CyU3PGpifLoad	501
CyU3PGpifOutput_t	498
CyU3PGpifOutputConfigure	505
CyU3PGpifReadDataWords	511
CyU3PGpifRegisterCallback	506
CyU3PGpifRegisterConfig	506
CyU3PGpifSMControl	509
CyU3PGpifSMStart	507
CyU3PGpifSMSwitch	509
CyU3PGpifSocketConfigure	516
CyU3PGpifWaveData	500
CyU3PGpifWaveformLoad	504
CyU3PGpifWriteDataWords	512
CyU3PGpioClock_t	629
CyU3PGpioComplexConfig_t	631
CyU3PGpioComplexGetThreshold	656
CyU3PGpioComplexMeasureOnce	658
CyU3PGpioComplexMode_t	624
CyU3PGpioComplexPulse	660
CyU3PGpioComplexPulseNow	662
CyU3PGpioComplexSampleNow	664
CyU3PGpioComplexUpdate	666
CyU3PGpioComplexWaitForCompletion	668
CyU3PGpioDelInit	638
CyU3PGpioDisable	639

CyU3PGpioGetIOValues 654	CyU3PI2sTransmitBytes 618
CyU3PGpioGetValue 646	CyU3PInitPageTable 45
CyU3PGpioInit 636	CyU3PloMatrixConfig_t 5
CyU3PGpioIntrCb_t 621	CyU3PloMatrixLppMode_t 4
CyU3PGpioIntrMode_t 622	CyU3PIrqContextRestore 59
CyU3PGpioIoMode_t 628	CyU3PIrqContextSave 57
CyU3PGpioSetClock 634	CyU3PIrqNestingStart 62
CyU3PGpioSetComplexConfig 642	CyU3PIrqNestingStop 63
CyU3PGpioSetIoMode 651	CyU3PIrqVectoredContextSave 58
CyU3PGpioSetSimpleConfig 641	CyU3PIsGpioComplexIOConfigured 671
CyU3PGpioSetValue 644	CyU3PIsGpioSimpleIOConfigured 670
CyU3PGpioSimpleClkDiv_t 627	CyU3PIsGpioValid 672
CyU3PGpioSimpleConfig_t 630	CyU3PIsLppIOConfigured 19
CyU3PGpioSimpleGetValue 652	CyU3PKernelEntry 64
CyU3PGpioSimpleSetValue 653	CyU3PLppDeInit 21
CyU3PGpioStopClock 635	CyU3PLppGpioBlockIsOn 20
CyU3PGpioTimerMode_t 623	CyU3PLppInit 20
CyU3PI2cConfig_t 553	CyU3PLppInterruptHandler 7
CyU3PI2cDeInit 558	CyU3PMbox 468
CyU3PI2cError_t 548	CyU3PMboxCb_t 467
CyU3PI2cEvt_t 547	CyU3PMboxDeInit 470
CyU3PI2cGetErrorCode 569	CyU3PMboxInit 469
CyU3PI2cInit 557	CyU3PMboxRead 471
CyU3PI2cIntrCb_t 550	CyU3PMboxReset 474
CyU3PI2cPreamble_t 551	CyU3PMboxWait 473
CyU3PI2cReceiveBytes 565	CyU3PMboxWrite 472
CyU3PI2cSendCommand 561	CyU3PMemAlloc 79
CyU3PI2cSetClock 555	CyU3PMemCmp 81
CyU3PI2cSetConfig 559	CyU3PMemCopy 83
CyU3PI2cStopClock 556	CyU3PMemCopy32 698
CyU3PI2cTransmitBytes 567	CyU3PMemFree 80
CyU3PI2cWaitForAck 563	CyU3PMemInit 77
CyU3PI2cWaitForBlockXfer 570	CyU3PMemSet 82
CyU3PI2sConfig_t 606	CyU3PMutex 52
CyU3PI2sDeInit 613	CyU3PMutexCreate 111
CyU3PI2sError_t 604	CyU3PMutexDestroy 112
CyU3PI2sEvt_t 605	CyU3PMutexGet 113
CyU3PI2sInit 611	CyU3PMutexPut 114
CyU3PI2sIntrCb_t 607	CyU3POsTimerHandler 66
CyU3PI2sPadMode_t 601	CyU3POsTimerInit 67
CyU3PI2sSampleRate_t 602	CyU3POtgChargerDetectMode_t 300
CyU3PI2sSampleWidth_t 603	CyU3POtgConfig_t 302
CyU3PI2sSetClock 609	CyU3POtgEvent_t 297
CyU3PI2sSetConfig 614	CyU3POtgEventCallback_t 301
CyU3PI2sSetMute 616	CyU3POtgGetMode 306
CyU3PI2sSetPause 617	CyU3POtgGetPeripheralType 307
CyU3PI2sStopClock 610	CyU3POtgHnpEnable 314

CyU3POtgIsDeviceMode	309
CyU3POtgIsHnpEnabled	316
CyU3POtgIsHostMode	310
CyU3POtgIsStarted	308
CyU3POtgIsVBusValid	311
CyU3POtgMode_t	296
CyU3POtgPeripheralType_t	298
CyU3POtgRequestHnp	317
CyU3POtgSrpAbort	313
CyU3POtgSrpStart	312
CyU3POtgStart	304
CyU3POtgStop	305
CyU3PPartNumber_t	3
CyU3PPibClock_t	483
CyU3PPibDelInit	487
CyU3PPibErrorType	476
CyU3PPibInit	486
CyU3PPibIntrCb_t	482
CyU3PPibIntrType	481
CyU3PPibRegisterCallback	489
CyU3PPibSelectIntSources	488
CyU3PPrefetchHandler	26
CyU3PQueue	53
CyU3PQueueCreate	99
CyU3PQueueDestroy	100
CyU3PQueueFlush	104
CyU3PQueuePrioritySend	102
CyU3PQueueReceive	103
CyU3PQueueSend	101
CyU3PReadDeviceRegisters	700
CyU3PRegisterGpioCallback	673
CyU3PRegisterI2cCallback	573
CyU3PRegisterI2sCallback	619
CyU3PRegisterSpiCallback	599
CyU3PRegisterUartCallback	545
CyU3PReturnStatus_t	689
CyU3PSemaphore	53
CyU3PSemaphoreCreate	106
CyU3PSemaphoreDestroy	107
CyU3PSemaphoreGet	108
CyU3PSemaphorePut	109
CyU3PSetEpConfig	359
CyU3PSetEpPacketSize	361
CyU3PSetGpioDriveStrength	650
CyU3PSetI2cDriveStrength	572
CyU3PSetPportDriveStrength	490
CyU3PSetSerialIoDriveStrength	13
CyU3PSetTime	128
CyU3PSpiConfig_t	581
CyU3PSpiDelInit	586
CyU3PSpiDisableBlockXfer	598
CyU3PSpiError_t	576
CyU3PSpiEvt_t	575
CyU3PSpiInit	585
CyU3PSpiIntrCb_t	580
CyU3PSpiReceiveWords	592
CyU3PSpiSetBlockXfer	594
CyU3PSpiSetClock	583
CyU3PSpiSetConfig	587
CyU3PSpiSetSsnLine	589
CyU3PSpiSsnCtrl_t	577
CyU3PSpiSsnLagLead_t	579
CyU3PSpiStopClock	584
CyU3PSpiTransmitWords	590
CyU3PSpiWaitForBlockXfer	596
CyU3PSysBarrierSync	32
CyU3PSysCacheDRegion	41
CyU3PSysCacheIRegion	40
CyU3PSysCleanDCache	38
CyU3PSysCleanDRegion	39
CyU3PSysClearDCache	39
CyU3PSysClearDRegion	40
CyU3PSysClockConfig_t	6
CyU3PSysClockSrc_t	2
CyU3PSysDisableCacheMMU	35
CyU3PSysDisableDCache	34
CyU3PSysDisableICache	33
CyU3PSysDisableMMU	43
CyU3PSysEnableCacheMMU	34
CyU3PSysEnableDCache	33
CyU3PSysEnableICache	32
CyU3PSysEnableMMU	42
CyU3PSysEnterSuspendMode	18
CyU3PSysFlushCaches	36
CyU3PSysFlushDCache	36
CyU3PSysFlushDRegion	37
CyU3PSysFlushICache	35
CyU3PSysFlushIRegion	37
CyU3PSysFlushTLBEntry	45
CyU3PSysGetApiVersion	16
CyU3PSysInitTCMs	43
CyU3PSysLoadTLB	44
CyU3PSysLockTLBEntry	44
CyU3PSysThreadId_t	676

CyU3PSysWatchDogClear 18	CyU3PUsbDevCapType 287
CyU3PSysWatchDogConfigure 17	CyU3PUsbDevProperty 330
CyU3PThread 49	CyU3PUsbDoRemoteWakeup 383
CyU3PThreadCreate 85	CyU3PUsbEnableEPPrefetch 408
CyU3PThreadDestroy 87	CyU3PUsbEnableIITPEvent 398
CyU3PThreadEntry_t 50	CyU3PUsbEpEvtCb_t 321
CyU3PThreadIdentify 88	CyU3PUsbEpEvtType 343
CyU3PThreadInfoGet 89	CyU3PUsbEpPrepare 407
CyU3PThreadPriorityChange 91	CyU3PUsbEpType_t 284
CyU3PThreadRelinquish 95	CyU3PUSBEventCb_t 325
CyU3PThreadResume 93	CyU3PUsbEventType_t 326
CyU3PThreadSleep 94	CyU3PUsbFeatureSelector 289
CyU3PThreadStackErrorHandler_t 50	CyU3PUsbFlushEp 379
CyU3PThreadStackErrorNotify 97	CyU3PUsbForceFullSpeed 400
CyU3PThreadSuspend 92	CyU3PUsbGetBooterVersion 23
CyU3PThreadWaitAbort 96	CyU3PUsbGetDevProperty 392
CyU3PTimer 54	CyU3PUsbGetEP0Data 377
CyU3PTimerCb_t 54	CyU3PUsbGetEpCfg 372
CyU3PTimerCreate 121	CyU3PUsbGetErrorCounts 410
CyU3PTimerDestroy 123	CyU3PUsbGetEventLogIndex 404
CyU3PTimerModify 126	CyU3PUsbGetLinkPowerState 394
CyU3PTimerStart 124	CyU3PUsbGetSpeed 369
CyU3PTimerStop 125	CyU3PUsbHostConfig_t 442
CyU3PToolChainInit 9	CyU3PUsbHostEp0BeginXfer 462
CyU3PUartBaudrate_t 522	CyU3PUsbHostEpAbort 463
CyU3PUartConfig_t 530	CyU3PUsbHostEpAdd 458
CyU3PUartDeInit 535	CyU3PUsbHostEpConfig_t 443
CyU3PUartError_t 527	CyU3PUsbHostEpRemove 459
CyU3PUartEvt_t 528	CyU3PUsbHostEpReset 460
CyU3PUartInit 534	CyU3PUsbHostEpSetXfer 461
CyU3PUartIntrCb_t 529	CyU3PUsbHostEpStatus_t 440
CyU3PUartParity_t 525	CyU3PUsbHostEpWaitForCompletion 464
CyU3PUartReceiveBytes 540	CyU3PUsbHostEpXferType_t 439
CyU3PUartRxSetBlockXfer 544	CyU3PUsbHostEventCb_t 438
CyU3PUartSetClock 532	CyU3PUsbHostEventType_t 437
CyU3PUartSetConfig 536	CyU3PUsbHostGetDeviceAddress 455
CyU3PUartStopBit_t 526	CyU3PUsbHostGetFrameNumber 456
CyU3PUartStopClock 533	CyU3PUsbHostGetPortStatus 450
CyU3PUartTransmitBytes 538	CyU3PUsbHostIsStarted 447
CyU3PUartTxSetBlockXfer 542	CyU3PUsbHostOpSpeed_t 435
CyU3PUndefinedHandler 27	CyU3PUsbHostPortDisable 449
CyU3PUsb3PacketType 287	CyU3PUsbHostPortEnable 448
CyU3PUsb3TpSubType 288	CyU3PUsbHostPortReset 451
CyU3PUsbAckSetup 374	CyU3PUsbHostPortResume 453
CyU3PUsbChangeMapping 386	CyU3PUsbHostPortStatus_t 436
CyU3PUsbDescPtrs 723	CyU3PUsbHostPortSuspend 452
CyU3PUsbDescType 286	CyU3PUsbHostSendSetupRqt 457

CyU3PUsbHostSetDeviceAddress 454
CyU3PUsbHostStart 445
CyU3PUsbHostStop 446
CyU3PUsbHostXferCb_t 441
CyU3PUsbInitEventLog 403
CyU3PUsbIsStarted 351
CyU3PUsbJumpBackToBooter 406
CyU3PUsbLinkPowerMode 331
CyU3PUsbLinkState_t 292
CyU3PUsbLPMDisable 401
CyU3PUsbLPMEnable 402
CyU3PUsbLPMReqCb_t 322
CyU3PUsbMapStream 384
CyU3PUsbMgrStates_t 344
CyU3PUsbRegisterEpEvtCallback 354
CyU3PUsbRegisterEventCallback 352
CyU3PUsbRegisterLPMRequestCallback 356
CyU3PUsbRegisterSetupCallback 353
CyU3PUsbResetEndpointMemories 409
CyU3PUsbResetEp 365
CyU3PUsbSendDevNotification 399
CyU3PUsbSendEP0Data 375
CyU3PUsbSendErdy 388
CyU3PUsbSendNrdy 390
CyU3PUsbSetBooterSwitch 405
CyU3PUsbSetDesc 357
CyU3PUSBSetDescType_t 329
CyU3PUsbSetEpNak 363
CyU3PUsbSetEpPktMode 412
CyU3PUsbSetLinkPowerState 396
CyU3PUSBSetupCb_t 328
CyU3PUsbSetupCmds 285
CyU3PUSBSpeed_t 323
CyU3PUsbStall 370
CyU3PUsbStart 347
CyU3PUsbStop 349
CyU3PUsbVBattEnable 381
CyU3PVicDisableAllInterrupts 29
CyU3PVicEnableInterrupts 28
CyU3PWriteDeviceRegisters 701

D

Descriptor Functions 179
Device Configuration Data Types 1
Device Configuration Functions 8
DMA Buffer 130

DMA Channel 130
DMA Data Types 132
DMA Descriptor 130
DMA Functions 174
DMA Management 129
DMA Socket 129

E

Embedded Real Time OS 47
Enumeration Modes 318
Error Codes 689
Event Flag Functions 114
Exception Handler Functions 25
Exceptions and Interrupts 25

F

FX3 Boot APIs 703
FX3 Boot Device Data Types 704
FX3 Boot Device Functions 708
FX3 Boot Device Interface 703
FX3 Boot GPIO Data Types 789
FX3 Boot GPIO Functions 791
FX3 Boot GPIO Interface 788
FX3 Boot I2C Data Types 775
FX3 Boot I2C Functions 780
FX3 Boot I2C Interface 774
FX3 Boot SPI Data Types 760
FX3 Boot SPI Functions 765
FX3 Boot SPI Interface 759
FX3 Boot UART Data Types 745
FX3 Boot UART Functions 751
FX3 Boot UART Interface 745
FX3 Boot USB Data Types 716
FX3 Boot USB Functions 726
FX3 Boot USB Interface 716
FX3 Device Configuration 1

G

GPIF Configuration 491
GPIF Data Types 493
GPIF Functions 500
GPIF Resources 492
GPIF State Machine Control 492
GPIF-II Management 491
GPIO data types 620
GPIO Functions 632

GPIO Interface 619

I

I2C Data Types 546
I2C Functions 553
I2C Interface 545
I2S data types 600
I2S Functions 607
I2S Interface 599
Interrupt Handling 27

K

Kernel Functions 55

L

Logging Data Types 675
Logging Functions 677
Logging Support 675

M

Mailbox Data Types 465
Mailbox Functions 468
Mailbox Handler 465
Memory Functions 67
Message Queue Functions 98
MMU Control Functions 42
Multi-Channel Functions 240
Mutex functions 109

P

PIB Interface Manager 474
P-port Data Types 474
P-port Functions 485
P-port Management 465

R

RTOS Constants 47
RTOS Data Types 48
RTOS Functions 55

S

Semaphore Functions 104
Serial Peripheral Interfaces 519
Socket Functions 189
SPI data types 574

SPI Functions 582

SPI Interface 573

T

Thread Functions 83
Timer Functions 119

U

UART data types 520
UART Functions 531
UART Interface 520
USB Constants 279
USB Device Data Types 318
USB Device Management 317
USB Device Mode Functions 344
USB Host Data Types 413
USB Host Management 412
USB Host Mode Functions 444
USB Management 279
USB OTG Data Types 293
USB OTG Management 293
USB OTG Mode Functions 302
Utility Functions 693